Modelling the Living Place Project using Algebraic Higher Order Nets

Susann Gottmann
and
Nico Nachtigall

Diploma Thesis

Supervisor:
Prof. Dr. Hartmut Ehrig
Dr.-Ing. Kathrin Hoffmann

Institut für Softwaretechnik und Theoretische Informatik
Technische Universität Berlin

June 3, 2011

Eidesstattliche Versicherung

Die selbstständige und eigenhändige Anfertigung versichert an Eides statt:

Berlin, den 30. Mai 2011

Susann Gottmann

Nico Nachtigall

Acknowledgements

We would like to thank all people helping and supporting us in this work: First of all we would like to thank our supervisor Dr. Kathrin Hoffmann, for her patience and support. Also thanks to Karsten Gabriel and Tony Modica for their support regarding the formal techniques and to Prof. Dr. Hartmut Ehrig and the whole TFS team for their constructive criticism.
Furthermore we would like to thank the team of the Living Place Project on the HAW Hamburg, in particular Jens Ellenberg and Sören Voskuhl for their support, patience and long descriptions of the Living Place Project. Also thanks to Prof. Dr. Kai von Luck and Prof. Dr. Julia Padberg.
Last but not least, thanks to our families.

Zusammenfassung

Innerhalb dieser Arbeit wird das System des *Living Place Hamburg* als ein System betrachtet, dem die Konzepte *Ubiquitouts Computing* sowie *Ambient Intelligence* zugrunde liegen, indem ein formales Modell des Systemverhaltens erarbeitet wird, so dass dieses Modell uns helfen kann, unser Verständnis über Ubiquitous Computing Systeme im Allgemeinen und unser Verständnis über das modellierte System im Speziellen zu erweitern.

Die hierbei verwendeten formalen Modellierungstechniken umfassen ein neues Konzept markierter Petri Netze, den sogenannten Algebraischen High-Level Netzen mit individuellen Token, kurz *AHLI* Netze, sowie das Konzept der regelbasierten Transformation solcher Netze basierend auf dem Double-Pushout Ansatz.

Dieser neue Ansatz von individuellen Token ermöglicht es, im Gegensatz zu den zuvor betrachteten Ansatz von kollektiven Token, markierungsverändernde Regeln zu formulieren, wobei die Verwendung derartiger Regeln innerhalb der Modellierung zu einem eleganten Modell des Living Place Systems, dem Hauptergebnis dieser Arbeit, führt.

Anhand des vorgestellten Modells wird ein beispielhaftes Szenario, in einer Form wie es durch das Living Place System verarbeitet werden kann, simuliert. Dabei werden die wesentlichen Bestandteile des Modells berücksichtigt.

Außerdem werden mögliche formale Analysetechniken eingeführt, die auf dem Modell angewendet werden können, um so bestimmte Eigenschaften des internen Systemverhaltens des Living Places Systems zu erhalten bzw. nachzuweisen und es werden einige Ergebnisse betrachtet, die aus solch einer Anwendung resultieren.

Abstract

Within this work the system of the *Living Place Hamburg* is considered as a system of *ubiquitous computing* and *ambient intelligence* by providing a formal model of the internal system behaviour of this system, so that this model helps us to improve our understanding of ubiquitous computing systems in general and in particular helps us to improve our understanding of the modelled system itself.

The thereby used formal modelling techniques include a new variant of Petri nets with markings called Algebraic High-Level Nets with Individual Tokens, short *AHLI* nets, as well as rule-based transformation of such nets following the double pushout approach.

This new approach of individual tokens in contrast to the former approach of collective tokens enables the formulation of marking-changing rules, where the usage of such rules within the modelling leads into an elegant model of the Living Place system as the main result of this work.

On the basis of the presented model an examplary scenario, as it can be processed by the Living Place system, will be simulated. Thereby, the essential components of the model will be taken into account.

Furthermore, possible formal analysis techniques are introduced that can be applied upon that model to obtain resp. verify certain properties concerning the internal system behaviour of the Living Place system as well as some results resulting from such an application are considered.

Contents

Chapter 1
Introduction

As a result of the rapidly increasing development of technology a new vision pioneered by Mark Weiser [Wei91] arose, called ubiquitous computing, which includes that systems of computing technology becomes part of our environment participating in our everyday life, so that the presence of computation will be ubiquitous whereas the used systems of computing technology should be invisible to humans not interrupting but autonomously supporting our actions so that the presence of computational technology seems to disappear.

Such ubiquitous computing, ambient intelligent systems are subject of current research and based on [Wei91], [CK04], [HK04] and [INK06] such systems can be considered in the context of this work as systems consisting out of multiple computing devices communicating with each other to operate in such a manner, so that they as an amalgamation are able to intelligently support the users of such a system in their everyday life by fulfilling certain everyday tasks, whereas the system itself mostly seems to be invisible to common awareness, so that people simply use them unconsciously.

Robin Milner [Mil06] stated, that although it seems that

> *[...] no one can cope with the complexity of such distributed systems, neither mathematically nor legally [...]* [Hil05, p. 17]

there must exist an understanding of such ubiquitous computing systems

> *[...] not only for scientists, but also to a certain degree for the general public. It may be argued that a user, being largely unaware of the performance of a ubiquitous computing system, has no need to understand it [...] but [...] as we aim to create new systems that we do not yet fully understand, we shall need to create models—and modelling tools—that help us to improve our understanding, both as scientists and as users.* [Mil06, p. 1]

Within this work the system of the *Living Place Hamburg* is considered as an example for such a system of *ubiquitous computing* and *ambient intelligence* by providing a formal model of the internal system behaviour of this system as the main result of this work.

So this work aims to provide a model of the internal system behaviour of a specific ubiquitous computing system that helps to improve our understanding of ubiquitous computing systems in general and that in particular helps to improve our understanding of this Living Place system itself.

Chapter 2 introduces the *Living Place Hamburg* as a smart home and show flat and as part of the Living Place Project, a project that includes the construction of this flat as a completed home containing different test set-ups, so that certain experimental usability studies in real-life situations of a single-person household taking place there, can be performed within this project. Furthermore, the system of this smart home, the so called system of the Living Place, is considered, whereas it is clarified, that the system consists of several computing components that as an amalgamation are able to support the residents of the Living Place in their everyday procedures taking place there by fulfiling certain tasks of their everyday life like reminding them of certain appointments that must be met.

Section 2.1 introduces the terms ubiquitous computing as well as ambient intelligence and illustrates that these two concepts are basic underlying concepts of the Living Place system, so that this system can be considered as a system of ubiquitous computing and ambient intelligence.

Considering the modelling of the internal system behaviour of this system, the architecture of the Living Place system is described within section 2.2, so that the so delivered detailed description of the system can serve as a basis for all further considerations of the following chapters concerning the modelling of that system.

Henceforth, the term *model of the Living Place system* or simply *model*, *model of the system* or *overall model of the system* often is used as a shorthand notation for *model of the internal system behaviour of the system of the Living Place.*

Chapter 3 is the first chapter that deals exclusively with the modelling of the Living Place system. Therein an overview of those submodels is given, that need to be elaborated to obtain an overall model of this system. Furthermore, this chapter presents a classification of these submodels into four levels. This classification provides a well-ordered overall overview of the submodels and their important relationships to each other. The presented classification will be maintained in all further considerations relating to the modelling of the Living Place system of the following chapters.

In chaper 4 requirements are defined, that need to be met by the resulting model and therefore that need to be met by the resulting submodels constituting this overall model of the system. Therefore, if the resulting model fulfils the requirements defined in chapter 4, it can be regarded as an adequate model of the Living Place system.

Chapter 5 is the first chapter of this work, that deals with the modelling techniques that are used within the elaboration of an overall model of the Living Place system by defining requirements towards these modelling techniques.

In chapter 6 these modelling techniques are finally introduced as formal constructs and it is checked whether the so defined techniques meet the requirements of the previous chapter 5. Therefore, an answer to the following question is given: "Are the used formal modelling techniques expressive enough and therefore suitable for modelling the Living Place system?". So, by using formal definitions as techniques for modelling the system of the Living Place, the resulting model is a formal well-defined construct. This is one of the main concerns of this work to obtain a formal model of the Living Place system compared to the informal model that is given by the system descriptions of section 2.2. The defined formal modelling techniques involve a new variant of Petri nets with markings called Algebraic High-Level Nets with Individual Tokens, short $AHLI$ nets, as presented in [MGE$^+$10], Algebraic Higher-Order Nets with Individual Tokens, short $AHOI$ nets, as a special type of $AHLI$ nets in comparison to [HM03] and [HEM05], i.e. nets that features nets and rules as tokens. Furthermore, the formal modelling techniques involve the transformation of such nets via transformation rules following the double pushout approach as presented in [EEPT06] and [MGE$^+$10] as well as amalgamated transformations as considered in [GBEE10] and [Gol10] and the theory of reconfigurable $AHLI$ systems as introduced in [EBE$^+$09]. By using $AHLI$ net transformations as a modelling technique, i.e. transformations with "individual" tokens as markings, instead of approaches with "collective" markings, the marking of a net with individual tokens can be manipulated with rules, i.e. marking-changing rules can be formulated, which is not possible with the collective token approach. This possibility to manipulate the marking of a net with rules is a feature of the used formal modelling techniques that significantly contributes to the expressiveness of these techniques and that is heavily used within the modelling of the Living Place system leading into an elegant model.

In chapter 7 the overall formal model of the Living Place system is elaborated by using the formal modelling techniques of chapter 6 and it is checked whether the so defined model meet the requirements of chapter 4.

In chapter 8 a possible scenario that can occur in the Living Place and therefore can be processed by the Living Place system is considered and the model of chapter 7 is used by simulating the considered scenario based on the model. Therefore, since the model is a model of the internal system behaviour of the Living Place system, on the basis of this model it can be reconstructed, how the system behaves within the presented scenario. So, this chapter shows up one possibility of how the obtained formal model of the Living Place system can be used to help us to improve our understanding of this system.

Chapter 9 finally considers some formal analysis techniques that can be applied upon the formal model of chapter 7. Therefor, these analysis techniques are described including the concept of parallel independence as defined in [EEPT06]. This concept allows to analyse the model in a way, so that it is clarified which parts of the modell and therefore which system behaviours of the Living Place system can proceed in parallel. By formally analysing this model it is shown, that some requirement towards the model of chapter 4 are fulfilled by the model. So, by analysing the formal model of the Living Place system as it is elaborated in chapter 7 using formal analysis techniques, results can be obtained that helps us to improve our understanding of this system.

So, the aim of this work is to elaborate a formal model of the Living Place system as a system of ubiquitous computing and ambient intelligence to obtain a reflection of the internal system behaviour of this system in this way, so that it is possible to simulate processes of the Living Place system by using this formal model. Furthermore, it is desirable to be able to perform formal analysis upon that model to analyse specific properties of the system behaviour of the Living Place system.

Additionally, it should be checked within this work, to what extent the used formal modelling techniques are expressive enough and therefore to what extent they are suitable for modelling the Living Place system as well as systems of such high complexity in an elegant manner.

The next chapter 2 starts with the introduction of the Living Place, the Living Place Project and the system of the Living Place.

1.1 Assignments of the Chapters

Chapters elaborated by Susann Gottmann:

- Chapter 2,
- Chapter 3,
- Chapter 4,
- Chapter 5,
- Chapter 8 and
- Chapter 9.

Chapters elaborated by Nico Nachtigall:

- Chapter 6 and
- Chapter 7.

Chapter 2

Living Place Project, Living Place and the System of the Living Place

As a result of the rapidly increasing development of technology, especially in the computer industry, a new vision of *Ubiquitous Computing* arose in the early 1990's. Mark Weiser as a pioneer of that vision described *Ubiquitous Computing* as computing technology becoming part of our environment participating in our everyday life (see [Wei91]). The consequence of that vision will be that the presence of computation will be ubiquitous. But it should be invisible to humans not interrupting but autonomously supporting the human actions so that the ubiquity of computational technology seems to disappear. In reality such a system needs many components which need to communicate and coorperate with each other leading to an ad-hoc network which alters permanently. Devices will connect or disconnect constantly and unpredictably.
The concept of Ubiquitous Computing and as a consequence of Ambient Intelligence will be introduced in more detail in section 2.1.

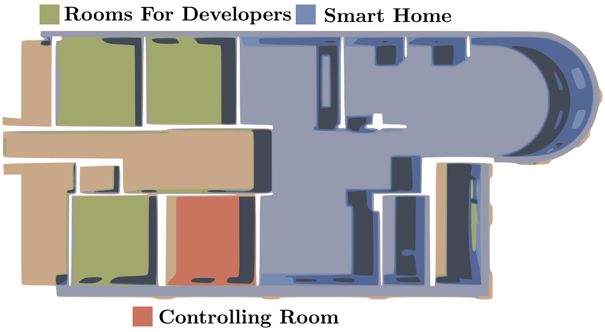


Figure 2.1: Ground plan of the Living Place ([OV10, p. 4])

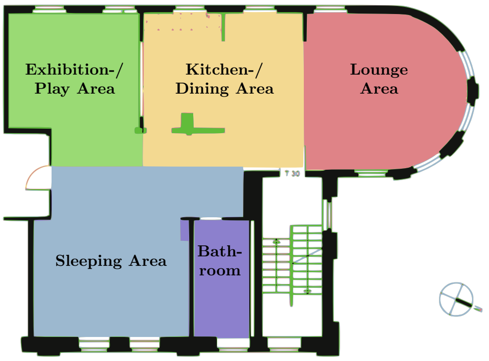


Figure 2.2: Allocation of the Living Place ([RV09, p. 7])

Since January 2009 a “home for the future” - a smart home - is developed at the University of Applied Sciences Hamburg (HAW): the *Living Place Hamburg.* It is a show flat and a laboratory for applied science and is based on the vision of Ubiquitous Computing investigating different areas of Ambient Intelligence. (See [Liv10])
The Living Place will be build as a completed home containing different test setups to perform experimental usability studies in real-life situations of a single-person household taking place there. These situations are caused by acts of the resident of the Living Place, that he performs there in the form of actions. The smart home should adapt dynamically and appropriate to different such situations, i.e. the smart home should immediately react to the residents actions, which caused these situations. It should enable the constantly altering and replacement of different components.

The rooms for the Living Place Project are separated in three parts as shown in the ground plan in figure 2.1: the smart home (blue) which contains a studio apartment for real-live experiments. Furthermore the Living Place consists of rooms for developers (green) and the controlling room (red) containing the system for usability studies,

servers etc.. The smart home and the other parts are strictly separated so that the person testing the ambient intelligent home will be relocated into real-life situations.
The *Living Place Project*, which includes the construction of the Living Place as well as the performance of the mentioned experimental usibility studies, will be realised by different areas of applied science, like architecture, light design, interactive design or computer science.

Figure 2.2 shows the allocation of the different areas within the smart home. The apartment contains a fully functional bathroom (violet area). The second room of the Living Place, the studio apartment contains a sleeping area (blue), a lounge area (red), a kitchen with a dining area (yellow) and an exhibition/play area (green). Each area is equipped with specific components like an Intelligent Bed in the sleeping area which is able to recognize the sleep phase of the person lying in bed. In the kitchen area will be a kitchen counter which is capable of multitouch events and will provide a daily planner which possesses all relevant dates and appointments of the inhabitant. In coorperation with the kitchen counter and the Intelligent Bed and other devices the whole flat will provide a new Alarm Clock 2.0 which automatically adjusts itself and will be able to alert the inhabitant properly (e.g. in turning on the radio or light, etc.). The whole studio apartment will contain an ambient light installation which automatically switchs on in order to alert the inhabitant. These computing components form the *Living Place system* as a system of Ubiquitous Computing and Ambient Intelligence.

In the following sections an introduction in the concepts of *Ambient Intelligence* and *Ubiquitous Computing* will be given. Afterwards the architecture of the Living Place system will be presented by characterizing the components and their correlation to each other.

2.1 The System of the Living Place as a System of Ubiquitous Computing and Ambient Intelligence

The system of the Living Place is a set of computing components, that are located in the Living Place and that are related to each other by communicating to each other in such a way so that they as an amalgamation are able to operate in order to pursue their common aim to support the resident of the Living Place in his everyday procedures taking place there by fulfilling certain tasks of his everyday life.

In order that these tasks can be fulfilled adequately by the system of the Living Place, this system must often include context information in its operations and decisions and the resident of the Living Place must interact with this system, i.e. interactions between the resident of the Living Place and the system of the Living Place must take place.

The system of the Living Place and therefore its computing components and its operations are almost completely invisible to common awareness. There are just some parts of this system, that enable the conscious interaction between a resident of the Living Place and the system, that are visible to common awareness.

Furthermore, each computing component of the system is exchangeable, therefore computing components might come and go, i.e. might turn on resp. go online or might turn off resp. go offline, leading to a highly dynamic system.

As an example, a cutout of the system is considered in the following: The system calculates when the resident of the Living Place got to go to get to work on time including traffic and weather information into this calculation. In order that the system can perform this calculation, the resident must enter his working hours as an appointment into the system, i.e. he must interact with the system. If the calculated timepoint is reached, the system reminds the resident of his appointment by giving a signal. This signal should start some time before he should leave the apartment in order to include some preparation time and the signal is not given, if the resident is in deep sleep. Therefore, the system must include context information in its decision to give a signal. By giving the signal, the system decides that the resident has to get up to get to work on time. Thus, the system fulfils the everyday task of having to check and having to decide when the resident needs to get up to get to work on time. This example illustrates, that the system consists of several computing components that communicate to each other to accomplish this task. There is a computing component that receives weather and traffic information. Another component calculates the timepoint on which the system gives the signal. There exists a component that allows the resident to enter the start of his working hours. Then there is a component that gives the signal. Additionally, there is a component that collects context information, i.e. details about the current phase of sleep of the resident. Furthermore, each computing component of the system that is included in this example is exchangeable. The most computing components of the system and its operations that are included in this example, are completely invisible to common awareness. Just those computing components of this example, that enable the conscious interaction between the resident of the Living Place and the system, are visible to common awareness, that includes the component that allows the resident to enter the start of his working hours and the component that gives the signal.

In subsection 2.1.1 the terms Ubiquitous Computing and Ubiquitous Computing system are introduced. The system of the Living Place, as it is described briefly in this section, includes the characteristics of such a Ubiquitous Computing System. Therefore, this system can be considered as a system of Ubiquitous Computing.

So, Ubiquitous Computing is an underlying concept of the Living Place system. Subsection 2.1.2 introduces the term Ambient Intelligence, short AmI, and shows, that as a consequence Ambient Intelligence is also such an underlying concept of the Living Place system. Thus, this system also can be considered as a system of Ambient Intelligence.

2.1.1 Ubiquitous Computing

In 1988 Mark Weiser introduced the term *Ubiquitous Computing*. His vision was about technology resp. computers residing in the human world and participating into everyday life, in order to enhance this world and support persons in their everyday life without being noticed to the human being. The user will be habituated to the omnipresence of technology so that it seems to disappear and getting practical invisble.
Weiser compares the Ubiquitous Computing with the electricity people are used to:

> *"Hundreds of computers in a room could seem intimidationg at first, just as hundereds of volts coursing through wires in the wall did at one time. But like the wires in the walls, these hundreds of computers will come to be invisible to common awareness. People will simply use them unconsciously to accomplish everyday tasks."* [Wei91]

This vision leads to a usage of systems consisting out of multiple computing devices communicating with each other to fulfil a certain task or multiple tasks of the everyday life. In order that these tasks can be fulfilled, this usage usually involves human-system interactions. Each component of such a Ubiquitous Computing system is exchangeable, therefore devices might come and go leading to a highly dynamic system. Due to the fact that the smart home should support the resident's actions hiddenly, data about the resident and his environment will be collected hiddenly in order to create an accurate context influencing the behaviour of the whole Living Place system. The pieces of information used to create a overall context will be collected from (a large amount) of different sensor devices which will be fused by different processes (see [Vos10]). Then, the most suitable actions will be infered without expecting an explicit input by the resident.
The usage of the prevailing context to compute an appropriate behaviour of the system is called *Context Aware Computing*. Thus the usability and efficiency will increase and the resident will not be interrupted by the smart home in his everyday life. The whole Ubiquitous Computing environment resp. system is in its functioning dependent on the location where it is situated and the person who is in reach of the system.

Ubiquitous Computing applications, short UbiComp applications, resp. systems are described as follows in [CK04]:

> *"UbiComp applications are dynamic, distinguishable, functional configurations of associated artifacts, which communicate and/or collaborate in order to realize a collective behavior."*

2.1.1.1 Artifact

The term *Artifact* is a common term in the field of Ubiquitous Computing. The term *Artifact* is equivalent to each of the aforementioned terms *Computing Component* and *Computing Device* as they are used in this work. In the following considerations of this work, the term *Computing Component* or *Computing Device* is used instead of the term *Artifact*.

Artifacts are everyday objects which are extended by new functions as described in the following quote:

> *"One proposed way to realize the AmI vision is to turn everyday objects into artifacts (by adding sensing, computation and communication abilities) and then use them as components of UbiComp applications within AmI environments."* [CK04, p. 13]

Artifacts are associated with each other forming a collaborate environment with collective behaviour as result. But they are working independently, because each artifact forms an organizational unit, which will be used by multiple participants. A reconfiguration of the connections between different artifacts is possible. Consider, that each computing component of the Living Place system as a system of Ubiquitous Computing is exchangable, therefore computing components might come and go leading to a reconfiguration of the connections between computing components. To realize communication between different artifacts, *"[e]ach artifact makes visible its properties, capabilities and services through specific interfaces [...]"* [CK04, p. 14].

2.1.2 Ambient Intelligence

The term *Ambient Intelligence* is based on the concept of Ubiquitous Computing. Both terms are directly connected with each other, because Ambient Intelligence can not be fully described without using the substance Ubiquitous Computing.

A description of Ambient Intelligence is given by Hellenschmidt and Kirste in [HK04]:

> *"The vision of Ambient Intelligence [...] is based on the ubiquity of information technology, the presence of computation, communication, and sensorial capabilities in an unlimited abundance of everyday appliances and environments."*

The origin of the Ambient Intelligence lies in the artifacts, the connection between them and their resulting interaction. For a further description of the term artifact, see 2.1.1.1. In forming relations between several artifacts to a network makes it possible to create an intelligent-looking environment, which enhances our world. For an example, an alarm clock alone adds nothing to an intelligent acting environment, but in creating an artifact out of the alarm clock and combine it with an extended bed, which then is also an artifact, intelligent-looking conclusions can be drawn out of these components, e.g. depending on the phase of sleep an intelligent alarm can be generated by the Alarm Clock, since the rising of an alarm during a light sleep phase instead of a deep sleep phase might be more pleasant to the user.

Hence, the communication between multiple artifacts is a basis for the environment appearing intelligent. The combination of artifacts and, consequently, their ability to compositionality is an essential prerequisite to form smart environments resp. to put the vision of Ambient Intelligence into reality.

Even if the ambient intelligent environment seem to act autonomically, the user will always have the control over the whole system as it is stated in [INK06]:

> *"The scenarios still emphasize the user control and the user's feeling of being in control of Ambient Intelligence. In most cases the user actuates the interaction with the Ambient Intelligence. If the environment is the actuator, rather than acting automatically, the Ambient Intelligence applications provide the user with justified suggestions than the user can approve or refuse. Automatic actions are based on profiles and personalisation and the user has the control to easily activate and deactivate them."*

2.2 Architecture of the System of the Living Place

As mentioned in section 2.1, the system of the Living Place is a set of computing components, that are located in the Living Place and that are related to each other by communicating to each other in such a way so that they as an amalgamation are able to operate in order to pursue their common aim to support the resident of the Living Place in his everyday procedures taking place there by fulfilling certain tasks of his everyday life. Therefore, these computing components are considered as computing subsystems of the Living Place system.

As mentioned in the introduction of chapter 1, this work aims to provide a model of the internal system behaviour of the Living Place system. Therefore, in the following, in this chapter the architecture of the system of the Living Place in terms of the system behaviour is described as it is partly presented in [OV11b], i.e. the computing **(I.) components of the system**, their system-related **(II.) possible relationships** to each other as well as their system-related **(III.) most important possible direct relationships to entities outside of the system** are described.

The system-related relationships of the computing system components are relationships, that are purely based on interactions between these computing components resp. that are purely based on user operations upon these computing components whereupon the system behaves in an appropriate manner, so that the computing system components as an amalgamation are able to persue their common aim to support the resident of the Living Place.

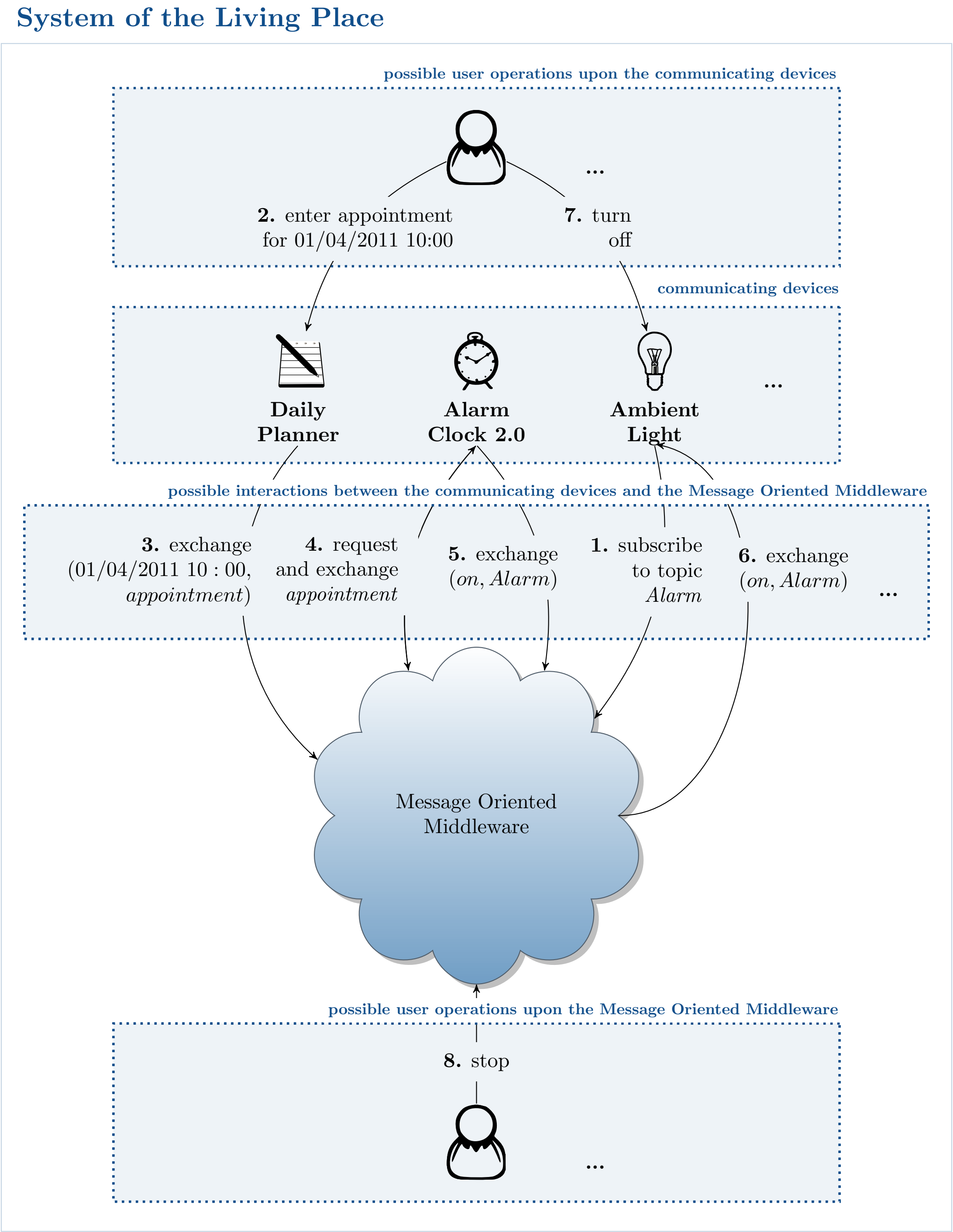

System of the Living Place
possible user operations upon the communicating devices
...
2. enter appointment for 01/04/2011 10:00
7. turn off
communicating devices
Daily Planner
Alarm Clock 2.0
Ambient Light
...
possible interactions between the communicating devices and the Message Oriented Middleware
3. exchange (01/04/2011 10 : 00, appointment)
4. request and exchange appointment
5. exchange (on, Alarm)
1. subscribe to topic Alarm
6. exchange (on, Alarm)
...
Message Oriented Middleware
possible user operations upon the Message Oriented Middleware
8. stop
...

Figure 2.3: Architecture of the Living Place system - a First Overview

The user operations upon the computing system components represent the in section 2.1 mentioned interactions between the resident of the Living Place and the system of the Living Place, that must take place, in order that the system can fulfil adequately certain tasks of his everyday life.

In all of the following considerations within this work, the words “computing” and “system-related” are omitted when mentioning the computing components of the system resp. their system-related relationships.

By describing the architecture of the system of the Living Place, the components of the system, their possible relationships to each other as well as their most important possible direct relationships to entities outside of the system are described. Therefore, the description of the architecture of the system in this chapter provides a detailed description of the system.

Apart from [OV11b], this chapter tries to provide a more fundamental overview of the architecture of the Living Place system, so that this overview can provide a basis for a basic approach to an understanding of the model of this system, as it is presented in chapter 7, and also can serve as a basis for the modelling of this system. So this chapter is intended to give an introduction to the modelling of the Living Place system by providing a detailed description of this system and therefore by describing all parts of the system that must be considered while modelling.

Since the system of the Living Place is intended to help people by supporting the resident of the Living Place in his everyday procedures taking place there, a natural acting, intuitive human-system interaction is necessary, where in particular the system of the Living Place must interact in a manner which is expected by the resident.[1]

To attain these aims, the system contains a composite of a variety of **communicating devices**, that are able to interact with the resident and that are able to perform certain the resident supporting tasks. By directly relating the term *communicating* to these devices it should be emphasized, that it is essential for the operation of the devices that the devices communicate with each other. To what extent communication is important for the operation of the devices, is explained in sections 2.2.2.1 to 2.2.2.8, where the devices and their respective operations are discussed in detail. Figure 2.3 gives three examples of such communicating devices - the Daily Planner, the Alarm Clock 2.0 and the Ambient Light. As an example for the possibility of the devices to interact with the resident resp. the possibility of the devices to perform certain the resident supporting tasks, action **2.** in figure 2.3 illustrates, that the user / resident is able to interact with the Daily Planner by entering an appointment resp. action **5.** in figure 2.3 shows, that the Alarm Clock 2.0 is able to trigger an alarm as soon as the alarm time towards the entered date is reached.

Furthermore figure 2.3 illustrates an additional component of the system, that is called **Message Oriented Middleware**, which among other things enables the communication between the communicating devices. Note that although it is essential for the devices to communicate with each other, the devices of the system are only able to indirectly communicate to each other by interacting with the Message Oriented Middleware. In figure 2.3 an example for this kind of indirect communication is given, where the Alarm Clock 2.0 indirectly communicates with the Ambient Light by exchanging the information *(on, Alarm)* via interactions with the Message Oriented Middleware - i.e. the Alarm Clock 2.0 exchanges the information *(on, Alarm)* with the Message Oriented Middleware in action **5.** by sending this information to the Middleware whereupon the Middleware receives that information and thereupon directly exchanges this information with the Ambient Light analogously in the reverse manner in action **6.**, i.e. the Middleware directly forwards resp. sends the information to the Ambient Light whereupon the Ambient Light finally receives the information.

The Message Oriented Middleware together with the set of communicating devices form the **(I.) components of the system**.

The set of **(II.) possible relationships** between these components is given by the different **possible interactions between the communicating devices and the Message Oriented Middleware**, since all possible relationships between components of the system are purely based upon interactions between these components, i.e. interactions between one or several communicating devices and the Message Oriented Middleware, since these are the only possible interactions between components of the system. This also illustrates the fact, that in the whole system, no direct interactions between different communicating devices exist. Therefore it is sufficient to describe the possible interactions between the devices and the Message Oriented Middleware when trying to grasp the set of possible relationships between the components of the system. Figure 2.3 illustrates some examples of such possible interactions, such as the topic-based exchange of information as it is exemplified by the exchange of *(on,Alarm)* from the Alarm Clock 2.0 to the Message Oriented Middleware in action **5.**. The topic-based exchange of information means, that the so sent resp. received information are typed over a topic, i.e. for example in *(on, Alarm)* the information *on* is typed over the topic *Alarm*. Thus, *on* in conjunction with *Alarm* can be interpreted as a call to raise the alarm in an appropriate way.

The set of the **(III.) most important possible direct relationships from components of the system to entitites outside of the system** contains the relationships, which trigger a change of one or more internal states of the components as a result. Therefore this set of relationships is purely based

[1] Compare to [OV11b].

on and thus is given by the set of **possible user operations upon the communicating devices** together with the set of **possible user operations upon the Message Oriented Middleware**, since within the system, as it is presented in this work, these operations are the only possibilities to enter into direct relationship with the components of the system from the outside of the system, so that a change of one or more internal states of the system is initiated. So a user resp. a resident of the Living Place is the only entity from the outside of the system, which can enter into direct relationship with the components of the system, so that a change of internal states of these components is triggered from the outside as a result. Figure 2.3 gives some examples of such operations, such as entering an appointment into the daily planner in action **2.** or stopping the Message Oriented Middleware in action **8.**. [2]

The **(I.) communicating devices** together with the **(I.) Message Oriented Middleware**, the **(II.) possible interactions between the communicating devices and the Message Oriented Middleware** and the set of **(III.) possible user operations upon the communicating devices** resp. **(III.) upon the Message Oriented Middleware** are the five elements that constitute the architecture of the system of the Living Place. By describing these five elements, we describe the architecture of the system of the Living Place.

Figure 2.3 represents a scenario, that can be processed by the system of the Living Place and which combines these five the architecture-forming elements, that must be described. Therefore, this scenario contributes to a better understanding of the architecture of the system. Also the scenario illustrates an everyday procedure of the residents at the Living Place and how the system can achieve his aim to support the residents of the Living Place in this everyday procedure. In this first simple scenario a user resp. a resident of the Living Place is able to enter an everyday appointment into the system of the Living Place, that he must keep, e.g. the beginning of his working hours. Thereupon the system dynamically generates an alarm as soon as the time is reached where the resident must be reminded of his everyday appointment, so he is able to keep this appointment. Afterwards the resident has the ability to turn off the alarm as well as any other device in the Living Place before leaving the Living Place. So the system of the Living Place supports the resident in his everyday procedure of having to decide when he got to go so he is able to keep his everyday appointment by punctual reminding the resident of one of his everyday appointments, so that he no longer must make this everyday decision. The system seems to act like a regular alarm clock, however, this scenario is intended to be as simple as possible. The detailed descriptions of the components of the system and their functionalities in sections 2.2.1 to 2.2.2.8 reveals a much more complex behaviour of the system of the Living Place, which allows the system to support the resident in even complex everyday procedures at the Living Place. In particular a much more complex scenario, that is given in chapter 8, deals with a much more complex behaviour of the system and illustrates a complex everyday procedure at the Living Place and how the system can achieve its aim to support the resident in this complex everyday procedure.

As shown in figure 2.3, the scenario is separated into eight different actions, that are numbered from 1 to 8. In **action 1** the Ambient Light subscribes to the topic *Alarm* at the Message Oriented Middleware, so every time when the Message Oriented Middleware receives information from a device, that are typed over the topic *Alarm*, the Message Oriented Middleware forwards this information to the Ambient Light. In **action 2** the user of the system resp. the resident of the Living Place enters an appointment for the first of April 2011 10 o'clock into the Daily Planner. Afterwards in **action 3** the Daily Planner exchanges this information with the Message Oriented Middleware by forwarding this appointment to the Middleware. **action 4** is represented by a request by the Alarm Clock 2.0 device to the Message Oriented Middleware, to obtain all appointments, that were sent to the Message Oriented Middleware in the past. As a response to that request the Message Oriented Middleware exhanges that information with the Alarm Clock 2.0 and so the Alarm Clock 2.0 finally receives the appointment, that was previously sent by the Daily Planner in action 3. In **action 5** the Alarm Clock 2.0 exchanges and therefore sends punctual a message $(on, Alarm)$, i.e. an information *on* which is typed over the topic *Alarm*, with resp. to the middleware to indicate, that now an alarm should be triggered in an appropriate way, so that the resident of the Living Place is able to keep his appointment. Since in action 1 the Ambient Light has expressed its interest in receiving such information of topic *Alarm*, the Message Oriented Middleware exchanges resp. sends the message $(on, Alarm)$ with resp. to the Ambient Light in **action 6**, whereupon the Ambient Light consequently receives this message. As a result the Ambient Light turns on and thus triggers the alarm. In a next step in **action 7** the user resp. resident of the Living Place turns off the

[2] In section 2.2.2.6 the Weather- and Traffic Information Service device is introduced, which retrieves its information from an entity outside of the system. This additional direct relationship between a component of the system and an entity outside of the system, which leads into a change of the internal state of the component, is described in section 2.2.2.6, however is omitted in this section to allow a better understanding of the architecture of the overall system.

alarm by turning off the Ambient Light. Finally in **action 8** he stops the Message Oriented Middleware and perhaps also some other components of the system and leaves the Living Place so he is able to appear for his appointment in time.

Consider, that the communicating devices, the possible interactions between these devices and the Message Oriented Middleware as well as the possible user operations upon these devices resp. upon the Message Oriented Middleware, that were used in the previous scenario, are just some examples of their respective categories. This first scenario is just an introduction to a better understanding of the architecture of the system. In considering the architecture of the Living Place system within this work and in particalur within this chapter, several more communicating devices, possible interactions as well as possible user operations are considered. In the following sections of this chapter 2 all five architecture-forming elements of the system of the Living Place are described in detail.

Section 2.2.1 deals with the **(I.) Message Oriented Middleware** as the central component of the overall system as well as with the **(III.) possible user operations** upon this component. Additionally an overview of the **(II.) possible interactions between the communicating devices and the Message Oriented Middleware** is given. The **(I.) communicating devices** are introduced in sections 2.2.2.1 to 2.2.2.8, where each device together with the **(III.) possible user operations** upon this device and the **(II.) possible interactions between this device and the Message Oriented Middleware** is treated within a separate section. Section 2.2.2.1 deals with the Alarm Clock 2.0 as a communicating device, section 2.2.2.2 introduces the Display and 2.2.2.3 the Daily Planner, in section 2.2.2.4 the Intelligent Bed is described, section 2.2.2.5 treats the Indoor-Positioning-System, section 2.2.2.6 deals with the Weather- and Traffic Information Service, in section 2.2.2.7 the Location-Based Screen is shown and finally in section 2.2.2.8, the last considered device, the Ambient Light is described.

In comparison to figure 2.3 figure 2.4 of the next section 2.2.1 illustrates a detailed overview of the architecture of the Living Place system containing all communicating devices as well as all possible interactions between the communicating devices and the Message Oriented Middleware.

Consider, that within this chapter the architecture of the Living Place system in terms of the system behaviour is considered. System behaviour in general involves all possible state changes of the system and therefore all possible states of the system as well as all the actions that causes these state changes as soon as they are applied. Therefore, in the following each section that deals with a component of the system contains a subsection that describes the internal system behaviour of that component, a subsection that describes the possible user operations upon that component, since every user operation causes a corresponding behaviour of the system component by causing a change of the internal state of that component, as well as a subsection that describes the possible interactions between the corresponding communicating device and the Message Oriented Middleware, since such an interaction is considered as system behaviour. System behaviour that is caused by a user operation is so called **extrinsic behaviour**, since it occurs only by direct influence from the outside of the system - all other system behaviours that are considered within this work, like the internal system behaviour of the components as well as the considered interactions between these components, are so called **intrinsic behaviours**, since this behaviour is performed by the system itself without direct influence from the outside of the system.

2.2.1 Message Oriented Middleware

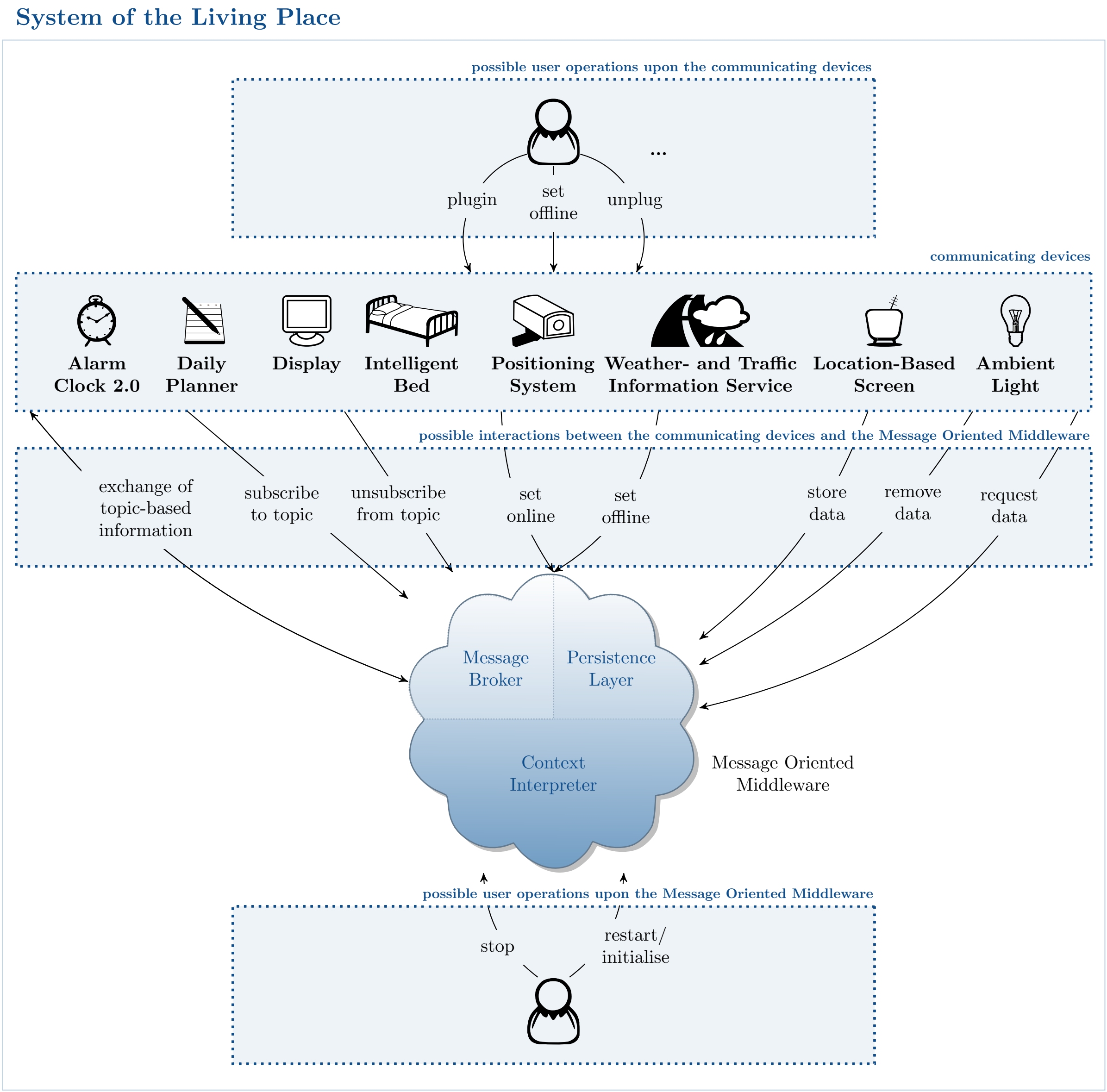

System of the Living Place
possible user operations upon the communicating devices
...
plugin
set offline
unplug
communicating devices
Alarm Clock 2.0
Daily Planner
Display
Intelligent Bed
Positioning System
Weather- and Traffic Information Service
Location-Based Screen
Ambient Light
possible interactions between the communicating devices and the Message Oriented Middleware
exchange of topic-based information
subscribe to topic
unsubscribe from topic
set online
set offline
store data
remove data
request data
Message Broker
Persistence Layer
Context Interpreter
Message Oriented Middleware
possible user operations upon the Message Oriented Middleware
stop
restart/ initialise

Figure 2.4: Architecture of the Living Place system - a Detailed Overview

The whole communication within the Living Place system will be managed by the *Message Oriented Middleware* which takes over the pivotal role. It is determined as a management system for physical devices organizing and distributing huge amount of data between large amounts of devices.[3] As pictured in figure 2.4, the Message Oriented Middleware is the central component of the whole Living Place system, where eight different communicating devices can interact with the Middleware in different ways. The Message Oriented Middleware consists of the parts: *Message Broker*, *Context Interpreter* and the *Persistence Layer*.

All available devices within the Living Place system are connected indirectly with each other over the Message Oriented Middleware. Incoming messages will be obtained by the Message Oriented Middleware and will be stored internally and passed to all devices which are interested in this kind of message. Devices express their interest in different kind of messages in subscribing to a specific topic. Therefore, each

[3] An introduction to the Message Oriented Middleware infrastructure is given in [YK10]. Within the Living Place Project group, the term "sensor cloud" is also used which encompasses the Message Oriented Middleware and the devices connected to it.

message can be passed to none, one or more receiving devices, depending on the available subscriptions to the specific topic of a message. Within the Living Place system, each device only knows itself but no other component. As a consequence, the communication between devices will be managed by the Message Oriented Middleware, so no messages will be sent directly from one device to another.

Each device is loosely linked to the whole system hence it can be removed from or connected to the system at any time without influencing any other device. Furthermore, it is possible to connect and disconnect devices dynamically to the whole system, without any special adaption, since the Message Oriented Middleware has the ability to abstract from the implementation details and technical data of each device. Constant system changes will be flexibly handled by the Message Oriented Middleware, therefore each device needs to register to the Middleware in order to connect to the whole system. A logout of devices or a missing accessibility of devices has to be processed by the Middleware.

The *Message Broker* as part of the *Message Oriented Middleware*, serves as a mediator of the communication and the mutual pooling of information between different devices of a system. In particular the devices are indirectly related to each other concerning their mutual exchange of information and hence are loosely coupled with each other regarding their communication.

As illustrated in figure 2.4 the Message Oriented Middleware establishes an indirect connection between devices and therefore contributes to each device communicative abilities. In such a messaging system, which mainly consists of a Message Oriented Middleware and communicating devices connected to it, the devices communicate over the Message Oriented Middleware so that it is able to fulfil the required assumption, which is the enabling of communication skills for each device.[4] But it is not possible to map a direct communication between different communicating devices, they always have to exchange information over the Message Oriented Middleware. The message broker will be introduced in sction 2.2.1.3.

Furthermode, a part of the Message Oriented Middleware is the *Context Interpreter*, as introduced in section 2.2.1.5, which contains inference rules for generating new context information out of single data or a combination of data in order to create a more precise context for generating correct or at least more accurate decisions within the Living Place system. Furthermore, the Message Oriented Middleware includes a persistence layer for storing all messages. The persistence layer will be presented in detail in section 2.2.1.4.

As indicated in figure 2.4, the user is able to apply user operations to the Message Oriented Middleware causing a change of its behaviour, e.g. stopping the Middleware. These user operations are described in next section 2.2.1.1. Furthermore, possible interactions between the Message Oriented Middleware and the communicating devices will be presented in section 2.2.1.2.

2.2.1.1 Possible User Operations

In section 2.2, a short overview of the architecture of the Living Place system is given. It is stated, that the inhabitant of the Living Place apartment is able to interact directly with the Message Oriented Middleware. In the following, the possible user operations with the Message Oriented Middleware will be presented.

As illustrated in figure 2.4, the user is able to stop and restart resp. initialise the whole Message Oriented Middleware, which consists of the meassage broker, the persistence layer and the Context Interpreter and to which devices can be connected.

For stopping the Message Oriented Middleware, all devices need to be removed from the system. Then no communication is able to take place, therefore the system is stopped.

When the user restarts resp. initialises the Message Oriented Middleware, all currently processed messages within the message broker and all data within the persistence layer will be deleted. Because of the deletion of all data, an initialisation of the Message Oriented Middleware should be done as seldom as possible.

2.2.1.2 Possible Interactions between the Communicating Devices and the Message Oriented Middleware

Interactions are mutual operations between two components. The interactions considered within the modelling of the Living Place system contain the following aspects:

[4] See [Mah04, p. 5]

1. A communicating device sends information to the Message Oriented Middleware and therefore operates upon this Middleware by writing that information into a buffer which can be recognized by the Middleware.

2. Thereupon, the Middleware receives that information and therefore operates upon the communicating device in an analogous manner by taking out that information from the buffer. Thereby, the Middleware changes the internal state of the communicating device, since the buffer and its content are parts of that device.

All interactions between communicating devices and the Message Oriented Middleware are based on these aspects. The possible interactions illustrated in figure 2.4 are presented in the following.

The Message Oriented Middelware manages the connections of the devices, therefore, a device, which is added to the system, is able to send a request to the Middelware, asking to get switched into online mode by the Middleware. The Middleware collects this message and reacts in an appropriate manner in setting the device into online mode, i.e. the internal state of the device will be changed by the Middleware. The reverse, the disconnecting of a device is processed in a similar manner: the device announces, that it wants to get disconnected resp. go offline in sending this request to the Middleware. The Middleware collects the request and processes it in changing the state of the device into offline mode.

The exchange of information between communicating devices and the Message Oriented Middleware is topic-based. In order to receive topic-based information, which will be distributed by the Middleware, a device gets the possibilty to subscribe to one or more specific topics. In this process, the devices sends an announcement to the Middleware, which receives and processes this announcement in subscribing the device to this topic.
The opposite interaction is also possible: the device gets the opportunity to unsubscribe from topics in order to stop the receipt of information regarding this topic. The process for unsubscribing is analogous to the process of subscribing topics.

Each communicating device which is able to send messages to the Middleware[5] needs to equip each message with a topic. After receiving the topic-based message, the Middleware distributes this information only to connected devices which are interested to the corresponding topic.

These steps are provided by the message broker which is part of the Message Oriented Middleware. The following interactions are provided between communicating devices and the persistence layer of the Message Oriented Middleware.

As soon as a device sends a topic-based information to the Middleware, this device asks the persistence layer of the Middleware to store this data in parallel. In some cases, the opposite is also possible, that data that are already stored within the persistence layer should be removed. For that the corresponding device sends a request to the persistence layer for removing information of a certain topic.

Furthermore, each device has the possibility to send a topic-based request to the persistence layer of the Middleware in order to receive all stored data according to this topic. For that, the device creates a request, which includes the relevant topic. Thereupon, the persistence layer collects this request and processes it in collecting all data that are stored within the persistence layer and which contain the topic of the request. Then the collected data records will be sent back to the requesting device.

Asynchronous Communication

All these mentioned possible interactions between the communicating devices and the Message Oriented Middleware are based on asynchronous communication as described in the following.

The communicating devices send resp. receive information from the Message Oriented Middleware without blocking. In an analogous manner, the Message Oriented Middleware sends resp. receives information from the communicating devices without blocking, i.e. the devices and the Middleware can interact with each other, so that other operations within these devices resp. the Message Oriented Middleware can proceed in parallel.

[5] Compare with 2.2.2.

2.2.1.3 Message Broker - Internal System Behaviour

The *Message Broker* as part of the Message Oriented Middleware is used as a link in communication between different components of the Living Place system. The message broker manages the distribution of messages between communication partners like connected devices or the Context Interpreter (see 2.2.1.5), as a consequence no such component is directly linked with an other component, i.e. they are completely decoupled from each other. The message broker serves, organises and establishes an indirect connection between different devices. Each component has exactly one connection to the message broker, which manages the connection resp. the identification of each device with the help of connection IDs, and therefore the message broker is the only component of the system which is able to distinguish each device regarding its designation. As a result, only the message broker is able to assign data from and to specific components.
Because no direct connection between devices can be established, the message broker needs to mediate information between devices. Therefore, the message broker is completly neutral to each component, because it abstracts from the characteristics and complexity of each component. The indirect connection serves for data exchange between devices, so only the message broker knows the interfaces to each device, which should be standardized for each such device to prevent incorrect interpretation of data.

The basis of each communicating device is a topic-based interface over which each device is connected with the message broker and over which the communication takes place and is regulated. A topic designates a category of information, e.g. "Weather" or "Alarm", etc.
Consider, that as already mentioned the whole communication within the Living Place system is asynchronous.

The communication between the devices via the message broker as the central mediator follow the Publish/Subscribe model which will be introduced in the following.

Publish/Subscribe Model - The Suitable Messaging Model of the Living Place system

A communicating device that exlusively sends information to the message broker is a so called Sender, a communicating device that exclusively receives information from the message broker is a so called receiver and a communicating device that sends and receives information to resp. from the message broker is a so called transceiver.

Since multiple communicating devices, which are connected to the message broker, depend on receiving and processing the same message that was sent by another device, the publish/subscribe model is used within the Living Place system. Figure 2.5 illustrates the structure of the publish/subscribe model. Each device which is interested to a certain type of message subscribes to its specified topic. A topic can be described as a "*mini message broker that gathers and distributes messages addressed to it.*" [HBS+02, p. 79]. It consists of an identity which encapsulates the specific name of the topic. In this model each receiver actively subscribes to one or more specific topics in order to receive all incoming messages belonging to these specified topics without the need to request for them, whereas each sender sends information of specific topics. The message broker (see 2.2.1.3) organises the distribution of all incoming topic-based information according to all available devices that are subscribed to the corresponding topic. Each device is able to register with as many topics as needed, i.e. an exclusive sender has no need to subscribe to a specific topic, whereas a receiver resp. transceiver is able to subscribe to as many topics of information as it is interested in.

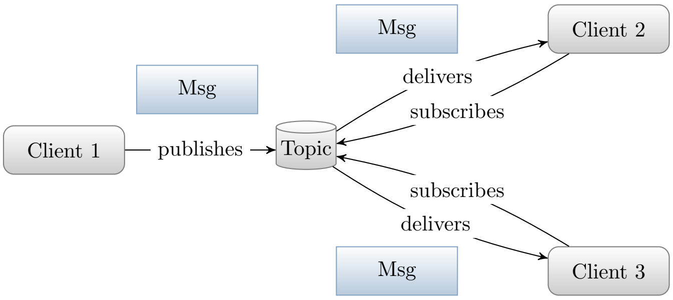


Figure 2.5: Communication in publish/subscribe model (acc. to [Haa02])

Communicating devices that send messages of a certain topic are called "Topic Publisher". If such a device decides to send messages of a specified topic it sends this message without consideration if an according subscriber exists. Communicating devices that are subscribed to specific topics to receive messages of these topics are called "Topic Subscriber". As can be seen in figure 2.5, for each topic several publishers and subscribers may exist, so each message can be assigned to several receivers.

Publishers and subscribers are loosely connected and therefore are anonymous with each other, because only the message broker is able to identify each device concerning their own identifier as described in section 2.2.1.3. Consequently the message broker provides the distribution of messages from the various publishers of a topic to all subscribers. Each incoming message will be copied by the message broker regarding the number of subscribers and sends each copy to one subscriber. Thereby each topic only contains the message as long as it takes to deliver all messages to the current subscribers. No message will be delivered twice and additionally each message will only be sent once. If a new device subscribes to a certain topic it will only receive all messages which arrive from the moment of the subscription on. The receiver should be active to receive the messages properly. Hence a timing dependency between sender and receiver exist.

In the publish/subscribe model the default subscription of each receiver to a topic lasts from the time of subscription to the time of cancellation of the device.

2.2.1.4 Persistence Layer - Internal System Behaviour

Apart from the Message Broker, the *Persistence Layer* is another part of the Message Oriented Middleware, which stores messages sent by communicating devices into a database as an external data storage. Therefore, old messages can be requested by the communicating devices.[6]

To request past information from the persistence layer, the interested device requests data belonging to a specific topic in sending an inquiry containing the specified topic and the identification of the requesting device to the persistence layer of the Message Oriented Middleware. The answer is generated by the server in collecting all past informations belonging to the given topic. The requesting device has to be subscribed to the respective topic in order to fetch the answer generated within the persistence layer. The communication regarding making a request is bidirectional, i.e. it takes place between the device posing the request, and the persistence layer.

2.2.1.5 Context Interpreter - Internal System Behaviour

The *Context Interpreter* is the third main part of the Message Oriented Middleware and is described in the following.

Context in the Living Place

This section deals with the term "context" regarding the articles [DAS01], [Gre01]. For the usage of context within smart environment Jang and Woo introduced 5W1H, a context representation which is used within the Living Place system ([JW05]).

Within the Message Oriented Middleware in the Living Place all incoming messages will be collected by the Context Interpreter and put into a general context which can be distributed to interested receivers to conclude new correct actions resp. data.

In [DAS01, p. 11] Day, Abword and Sabler define the term "context" as follows:

> *"**Context**: any information that can be used to characterize the situation of entities (i.e. whether a person, place or object) that are considered relevant to the interaction between a user and an application, including the user and the application themselves. Context is typically the location, identity and state of people, groups and computational and physical objects."*

Additionally in their previous work [SDA99, p.1] they state:

> *"Of major interest are context-aware applications, which sense context information and modify their behavior accordingly without explicit user intervention."*

In [DAS01, p. 11] they support the opinion that context is the "constantly changing execution environment". They separate the environment in three parts: the computing environment (components of the smart home, connections, network, etc.), the user environment (location, people, social situation) and physical environment (lighting and noise level). All three parts of the environments are represented

[6] Further details are described in [OV11a] and [OV10].

within the Living Place and all are exposed to permanent changes. Hence the current context needs to be adapted dynamically regarding new information which will be measured through sensor devices or other new input data like information from internet services (weather forecast, or traffic information) or direct user input (e.g. new dates entered in daily planner).

Furthermore, Day, Abword and Sabler distinguish between explicit input, like direct input by the user, and implicit input meaning information aquired by input devices, like sensors. However, they prefer a generalized model for context-aware applications, which will be applied to the Living Place. There, the context will be infered from all kinds of information, regardless if they are implicit or explicit.

In [Gre01, p. 259] Greenberg refers to three different theories of context, whereas the "Activity Theory" will be taken as basis in the Living Place (see [Ell10b, p. 3]). This theory sets activities of the user into the center for defining a context. It is defined as:

> *"Activity theory (Nardi, 1997a) claims that activity defines context, where an activity comprises a subject (the person or group doing the activity), an object (the need or desire that motivates the activity), and operations (the way an activity is carried out). Artifacts and environment are seen as entities that mediate activity. Nardi (1997b) then argued that activity-as-context includes not only external resources (people, artifacts, settings) but internal processes (objects and goals) as well."* [Gre01, p. 260]

The activity theory considers the dynamic changes of the environment and as a consequence the constant adaption of the context. The course of the activity might change in the process of customizing the context. For detecting a suitable context, the internal state of the user should be taken into consideration, which can only be estimated. As described in [Ell10b, p. 3], this factor will be optional in the first steps. The combination of all extern collected information, and even a hint to the internal state of the user which will be get by questioning as an example, will never form a perfectly suitable context, therefore the infered context will always be an approximation resp. a guess of the real context at this moment.[7]

Another question deals with the representation of context. In [JW05] the 5W1H model is introduced which is used in the Living Place Project (see [Ell10b, p. 3] and [Ell11b]).

Who A specified user, which will be described through a name, ID and a profile.

Where The certain location of the user, which will be done with x- and y-coordinates or through the abstract description of the place, like "kitchen" or "at the multitouch kitchen counter".

When This attribute describes the moment or the time interval of the validity of the context.

What The object the user is paying attention to or the object the user needs for his action.

How Characterizes the action the user is performing or how the person uses an object.

Why The intention or emotion which is the reason for performing a certain action. This attribute describes the internal state of the resident, which will will not be regarded as explained before.

The 5W1H model puts the user into its center, so that the infered context will be user-centric but general enough to be independent from other devices or applications. Each receiving component of this certain context needs to interpret it to its own necessity. The model provides enough information for each component to initiate appropriate actions.

Context Interpreter

As mentioned, the system of the Living Place often include context information of the Living Place in its operations. Therefore, an important task within a smart environment is to infer the right context. The *Context Interpreter* as part of the Message Oriented Middleware, fulfiling this task, is based on the ideas presented in [DAS99] and [SDA99]. Dey, Abword and Sabler describe a system where different components are indirectly connected with each other over one or more servers. To guarantee an unproblematic communication betweeen all components the server needs to receive raw data from sending devices which will be decoded in device-specific format. Therefore the server needs to convert the data to a general

[7] Cp. [Gre01, p. 262]

format and aggregate different data to a more abstract information. So, to obtain the prevailing context, the aggregation and interpretation of low-level data is necessary because the whole system within a smart home is highly variable, e.g. different sensors can measure the same matter but express this matter by different values depending on their implementation or the aggregation of low-level data of several sensors is required to obtain the prevailing context - e.g. to discover whether the resident of the Living Place dresses himself at the moment as the prevailing context the aggregation of low-level data of two different sensors is required, on the one hand the data of the sensor that detects the location of the resident are required and on the other hand the data of the sensor that detects whether the resident is in bed are required, since based on the assumption that the resident is in the beedroom but actually is not in the bed it can be inferred that the resident dresses himself at the moment.

In the previous section 2.2.1.5 the term "context" is defined and the use of this term will be explained in relation to the Living Place Project.

In the smart environment a context describes an aggregated or non-aggregated information about the environment, which will mainly be collected by sensor devices. From different contexts new information can be aggregated, where the component that performs this aggregation and that provides access to context information is called "context widget" by Dey, Abword and Sabler. Within this work a "Context widget" is the equivalent to the context interpreter as part of the Message Oriented Middleware. They define the context widget as follows:

> *"A context widget is a software component that provides applications with access to context information from their operating environment."* [SDA99, p.2]

The Context Interpreter presented in [DAS99] helps in interpreting the context.

In the Living Place Project the context interperer is based on the work by Dey, Abword and Sabler however, with slight differences.[8] First, each device will convert its device specific information into a global data format, therefore the converting of data is shifted into the devices.

Each incoming message will be evaluated regarding its topic, interpreted and distributed by the message broker. The message will be passed to the Context Interpreter which analyzes the information and tries to aggregate new information out of the information or tries to combine the information with an other information. A new context is able to be inferred within the Context Interpreter according to certain inference rules. The Context Interpreter gets all incoming data and infers a new 5W1H context ot of them as this form of context is described in the section above. The information will be sent back to the message broker, which distributes it to the devices. The resulting information depends on the rules available within the Context Interpreter .

The Context Interpreter is a special part of the Message Oriented Middleware which receives all incoming messages in order to derive new 5W1H contexts regarding a given set of inference rules.

2.2.2 Communicating Devices

In smart environments, like the Living Place system, a lot of different devices need to collaborate, interact and communicate with each other. In figure 2.6 a possible configuration of a Message Oriented Middleware is shown, where a device communicating over a Middleware can be a sender, receiver or transceiver. In figure 2.6, device A and D are transceivers, B is a receiver and C a sender.

As pictured in 2.6 the Living Place system contains the following different kinds of devices:

- Sender (Producer): Devices that send data to the Message Oriented Middleware. Consider, that all sensor devices within the Living Place are devices of the kind sender, e.g. pressure sensors in the Intelligent Bed, etc.
- Receiver (Consumer): Devices which receive data from the Message Oriented Middleware whereupon they perform specific actions, e.g. Ambient Light, etc.
- Transceiver (Producer+Consumer): Devices which send and receive data to resp. from the Message Oriented Middleware. This kind of device is a combination of sender and receiver. It waits for information and provides data for other devices, e.g. Alarm Clock 2.0.

[8] For a description of the approach in the Living Place Project see [Vos09] and [OV10, p. 18].

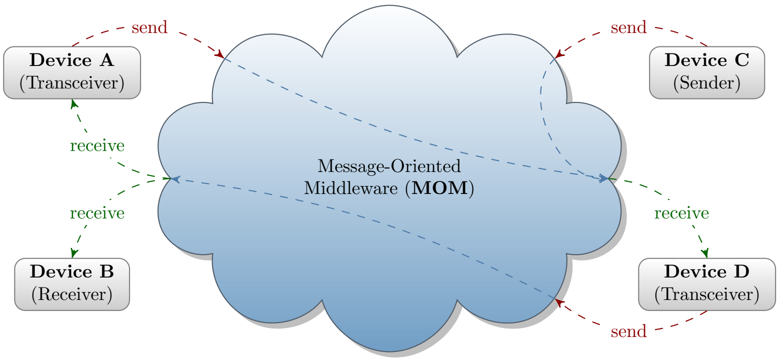


Figure 2.6: Possible configuration of a system based on Message Oriented Middlware - components and their connections

The devices presented in the next sections can be categorized to the different device types, as follows:

- Alarm Clock 2.0 → Transceiver
- Display → Receiver
- Daily Planner → Sender
- Intelligent Bed → Sender
- Indoor-Positioning-System → Sender
- Weather Information System → Sender
- Traffic Information System → Sender
- Location-Based Screen → Receiver
- Ambient Light → Receiver

2.2.2.1 Alarm Clock 2.0

This section deals with the description of the *Alarm Clock 2.0* device. It is divided into the parts: internal system behaviour of the device and the provided possible user operations on the Alarm Clock 2.0. The following figure 2.7 gives an overview of the Alarm Clock 2.0.

The possible interactions between the Alarm Clock 2.0 and the Message Oriented Middleware are presented in figure 2.7, where the detailed descriptions of these possible interactions are given in section 2.2.1.2.

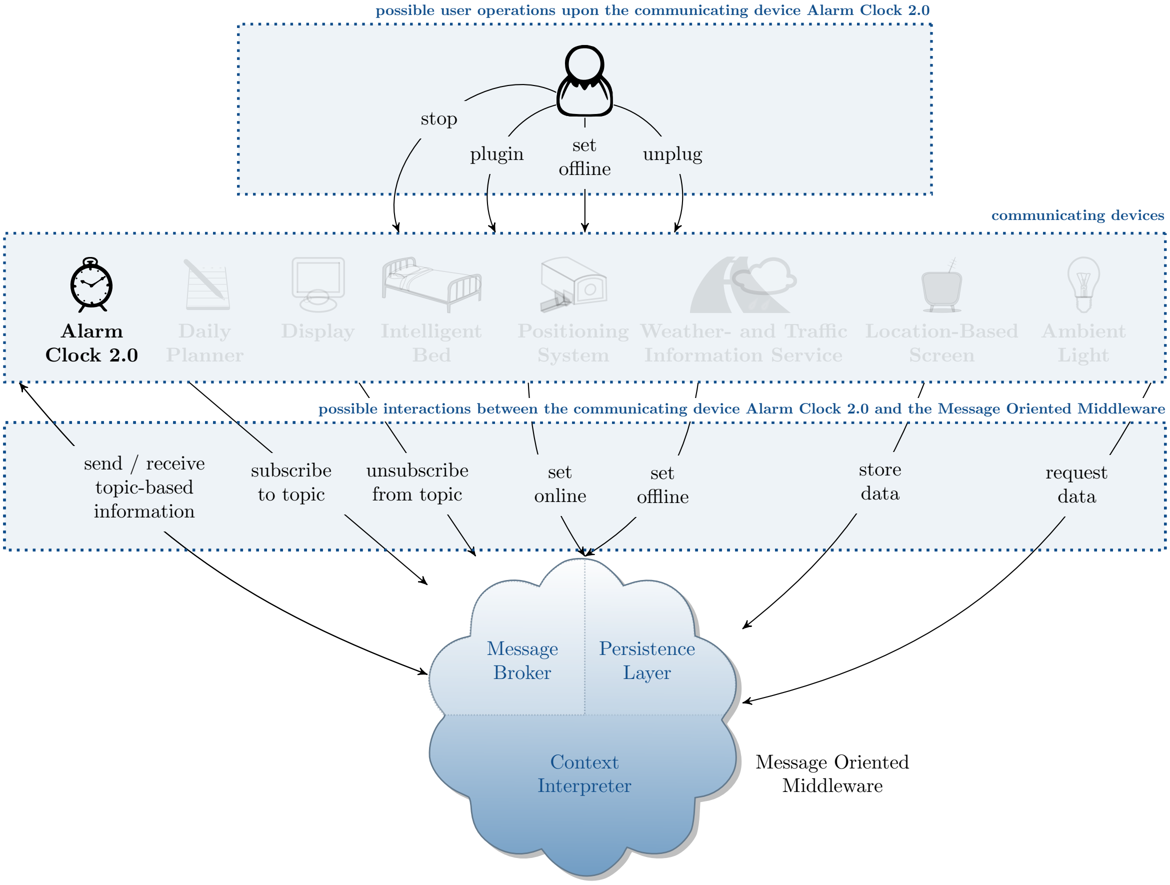


Figure 2.7: Alarm Clock 2.0 - an Overview

2.2.2.1.1 Internal System Behaviour

The *Alarm Clock 2.0*[9] denotes a new concept for an alarm clock, which shares the task of alarming the resident with the original form. Apart from that it differs completely from the common idea. The Alarm Clock 2.0 reminds the person of forthcomming events and additionally calculating an appropriate time for reminding taking a preparation time and the overall situation outside of the Living Place apartment into account.

The Alarm Clock 2.0 will be invisible for the user, still it will collect context data, i.e. general data directly provided by other components or all 5W1H contexts infered by the Context Interpreter (see 2.2.1.5), about the inhabitant and the overall situation of the inside and outside of the Living Place apartment from other devices connected to the Living Place system, thus it is able to adapt itself flexibly to its environment. Informations regarding the outside of the apartment will be collected over the internet, i.e. data concerning weather forcasts or traffic condions, both components will be presented in section 2.2.2.6.

The Alarm Clock 2.0 is subscribed to specific topics in order to receive information from other components connected to the Message Oriented Middleware of the Living Place system. The Alarm Clock 2.0 dynamically calculates an optimal moment to initiate an alarm in order to remind the person about an occuring event out of all available data regarding appointments, supplied by the Daily Planner, which is introduced in section 2.2.2.3. For infering an appropriate point of time for waking-up resp. reminding the user, all available informations will be taken into account, e.g. if a traffic jam was announced on the way to work, which entails additional time, an earlier timepoint for reminding will be generated. If bad weather or traffic conditions occur and are sent to the Alarm Clock 2.0, additionally time will be calculated which will be regarded for an approptiate wake up time resp. moment for reminding the user.

The alarm should occur at the right location, which means at the place where the person sojourns.

[9] See [Ell09], [Ell10a], [Ell10b] and [Ell11a].

For that location information will be required. The Alarm Clock 2.0 only creates an information, that an alarm should be provoked and sends it to the Message Oriented Middleware. There, one or more proper devices will react on this message and execute the alarm (e.g. light will be switched on and the TV set in the field of view of the user switches on).

The Alarm Clock 2.0 alerts the user of a begin of a period of time, e.g. beginning of work, taking a make-ready time into account. This preparation time includes the time for getting-up, the morning toilet, eating, dressing and to meet an appointment, as described in [Ell11a]. The Alarm Clock 2.0 needs to flexibly remind the inhabitant of fulfiling all these steps in the progress of make-ready. The internal states of the inhabitant and its tasks, which will be administered by the Alarm Clock 2.0, are presented in the following table 2.1: In order to get ready for an appointment, the user should fulfil these tasks, whose progress will be controlled by the Alarm Clock 2.0. The device regularily reminds the user of tasks, as long as at least one of them is still open. The fulfilment of these tasks is not dependent on a given order.

State	**What to do?**
asleep	The person is still asleep and needs to get woken up by the Living Place system.
hungry	The person is hungry and needs to go into the kitchen area in order to eat something.
unwashed	The person needs to wash himself in the bathroom.
undressed	The person is undressed or wrongly dressed and needs to find a dress in the wardrobe which will be situated in the sleeping area and put it on.

Table 2.1: Tasks to Fulfil by the User During the Preparation Time in the Alarm Clock 2.0

Additional to an appropriate start time, the Alarm Clock 2.0 also infers an appropriate moment to stop the alarm. If either all of these tasks are fulfilled, or the moment for stopping the alarm is reached, the Alarm Clock 2.0 automatically resets itself in order to get ready for the next forthcoming appointment.

In [Ell11a] the following components of the Alarm Clock 2.0 are introduced:

- TimeModule : The TimeModule is responsible for supervising the time and send every minute the current time to the Alarm Clock 2.0.
- ContextFassade : The ContextFassade is one part of the communication unit with the Message Oriented Middleware. It gets the global context and converts it into specific information for the Alarm Clock 2.0.
- AlarmMessageAdapter : It is a part for providing the communication with the Message Oriented Middleware. It gets the internal alarm message of the Alarm Clock 2.0 and transforms it into the global format of the whole system.
- ConfigFassade : With the help of the ContextFassade, configurations of the Alarm Clock 2.0 can be done by the user, i.e. switching the Alarm Clock 2.0 off.

2.2.2.1.2 Possible User Operations

Like all other devices, the Alarm Clock 2.0 is connected to the Message Oriented Middleware of the Living Place system. It is able to be set into offline mode, get connected resp. disconnected dynamically without interrupting the Message Oriented Middleware or other devices in their operations, i.e. all operations of the Message Oriented Middleware resp. of the devices not referring to the Alarm Clock 2.0 will be executed unmodified. The operations *plugin*, *unplug* and $set\ offline$ are provided as user operations and can be executed by the resident in order to add, remove or set the Alarm Clock 2.0 into offline mode.

The Alarm Clock 2.0 reminds the inhabitant regularily of the fulfilment of the tasks mentioned in table 2.1. The user is able to stop the Alarm Clock 2.0 from constantly sending an alarm according to the current appointment and consequently is able to initiate a reset of the device.

2.2.2.2 Display in the Multitouch Kitchen Counter

This section gives an introduction to the *Display* being part of the Multitouch Kitchen Counter. The following figure 2.8 shows an overview of the Display device, illustrating the individual provided user operations and the possible interactions between the Display connected to the Living Place system and the Message Oriented Middleware.

The possible interactions between the Display and the Message Oriented Middleware are presented in figure 2.8, where the detailed descriptions of these possible interactions are given in section 2.2.1.2.

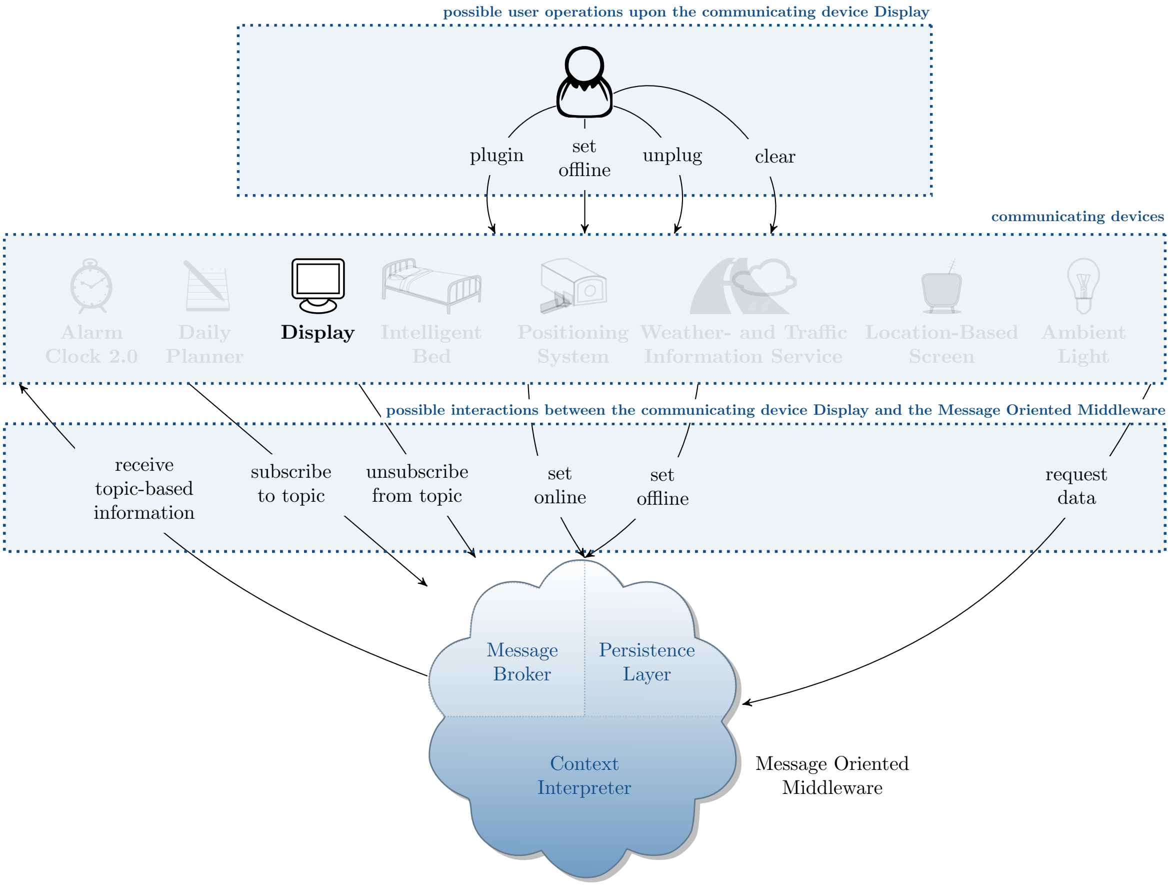


Figure 2.8: Display - an Overview

2.2.2.2.1 Internal System Behaviour

The *Multitouch Kitchen Counter* will be realized as solid multitouch tabletop. It contains a Display, which is able to show different kinds of information, i.e. information regarding appointments previously entered into the Daily Planner, and data sent by the Weather and Traffic Information System presented in section 2.2.2.6. It is a receiver device, which is subscribed to the topics representing calendar, weather and traffic data.

2.2.2.2.2 Possible User Operations

The user is able to perform the default operations, like plug in, unplug the Display device or set the Display device into offline mode. Additionally the user is able to remove selected data records from the screen of the Display device by performing the user operation *clear*.

2.2.2.3 Daily Planner

The *Daily Planner* and its operation which are shown in the following figure 2.9 are presented in this section.

The possible interactions between the Daily Planner and the Message Oriented Middleware are presented in figure 2.9, where the detailed descriptions of these possible interactions are given in section 2.2.1.2.

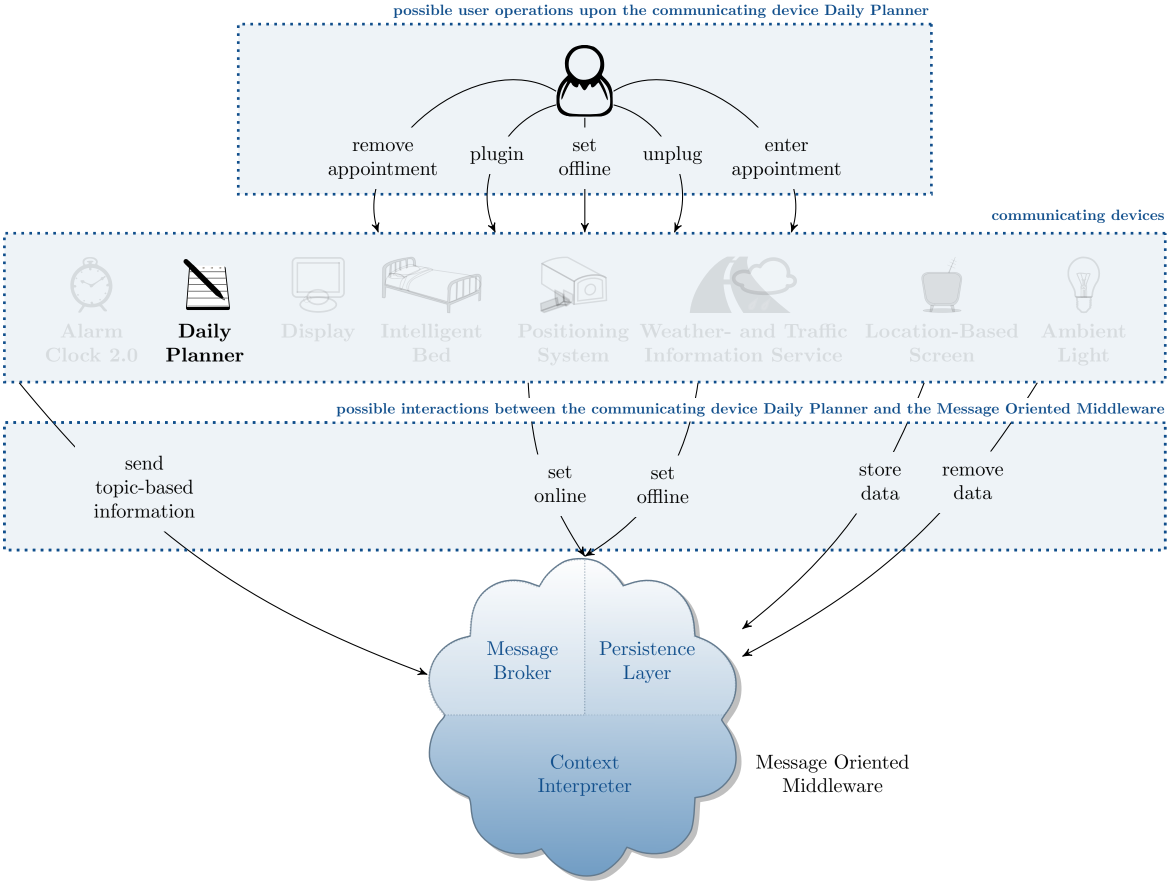


Figure 2.9: Daily Planner - an Overview

2.2.2.3.1 Internal System Behaviour

The Multitouch Kitchen Counter also provides an interface for the resident to enter data for the Living Place system. It contains a Daily Planner[10] to which the resident can enter his appointments in providing the timestamp and the description of the corresponding appointment. E.g. the resident enters the start of his working hours into the Daily Planner. The Daily Planner forwards the entered data to the Message Oriented Middleware.

In addition, the user is able to remove existing calendar data.

2.2.2.3.2 Possible User Operations

The device of the Daily Planner can be plugged in, set into offline mode and plugged out by the user. Additionally the user is able to enter new calendar data into the Daily Planner (*enter appointment*) or delete calendar data from the Living Place system, which were previously entered into the Daily Planner (*remove appointment*).

2.2.2.4 Intelligent Bed

This section deals with the *Intelligent Bed*: It presents the Intelligent Bed itself, which is able to recognize different sleep stages and provides its result to the Message Oriented Middleware. For that, a short introduction to the human sleep stages is given in addition.
The following illustration 2.10 gives an overview of the Intelligent Bed and its interactions with the Message Oriented Middleware as well as the possible user operations upon that device.

The possible interactions between the Intelligent Bed and the Message Oriented Middleware are presented in figure 2.10, where the detailed descriptions of these possible interactions are given in section 2.2.1.2.

[10] See [Bar09] and [Bar10].

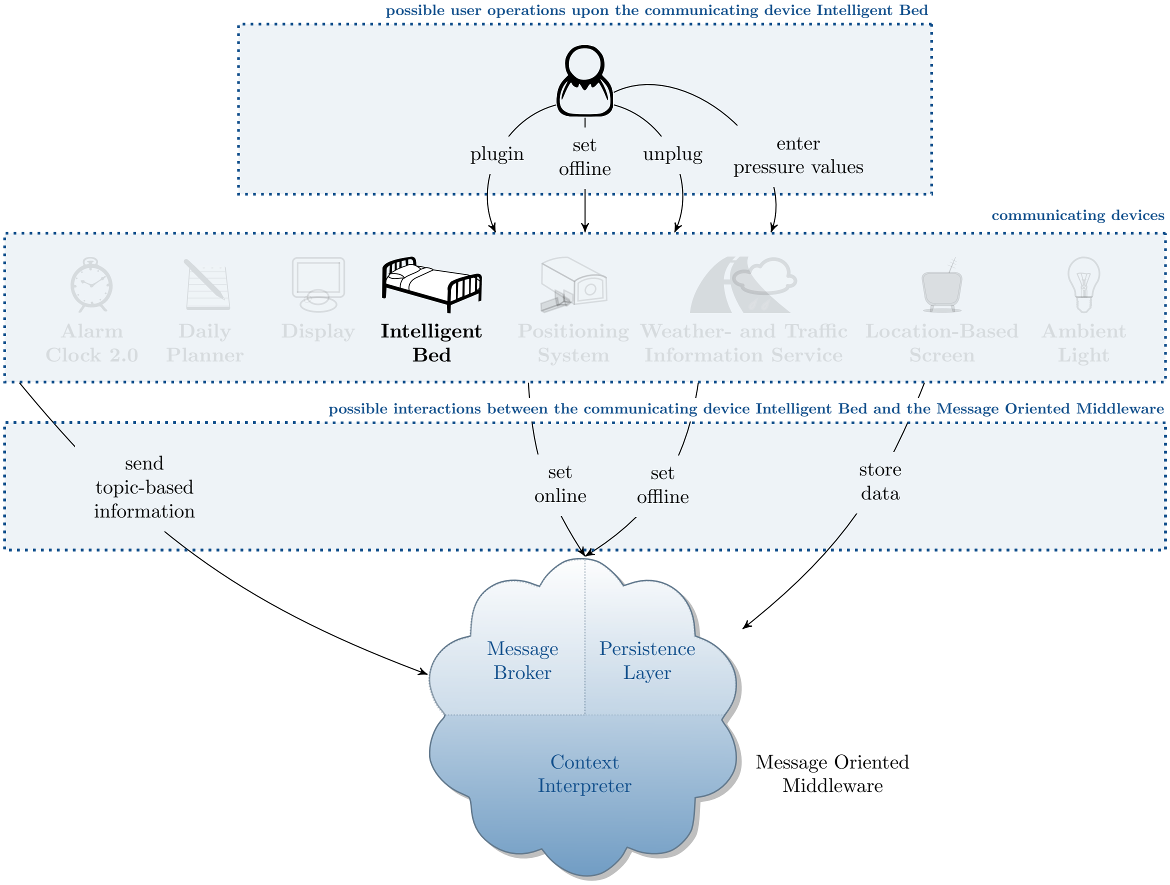

possible user operations upon the communicating device Intelligent Bed
plugin
set offline
unplug
enter pressure values
communicating devices
Alarm Clock 2.0
Daily Planner
Display
Intelligent Bed
Positioning System
Weather- and Traffic Information Service
Location-Based Screen
Ambient Light
possible interactions between the communicating device Intelligent Bed and the Message Oriented Middleware
send topic-based information
set online
set offline
store data
Message Broker
Persistence Layer
Context Interpreter
Message Oriented Middleware

Figure 2.10: Intelligent Bed - an Overview

2.2.2.4.1 Internal System Behaviour

The main aim of the Living Place system is to make the resident's daily routines easier. The process of getting up in the morning will be enhanced by the *Intelligent Bed*. The Intelligent Bed gets information about the inhabitant's sleep stages, interpretes them and sends the resulting information to the Message Oriented Middleware. Retrieving the data will be hidden from the residents sight and without any body contact to the person. Therefore collecting the data will be done with the help of pressure sensors, where each sensor collects bodymonitoring data, i.e. pressure as result from the movement of the user in bed. The Intelligent Bed itself should not alter or intervene into the persons daily routines regarding bedtime.[11]

Human sleep stages

While sleeping each human runs through the following sleep stages. The first stage is the phase of *falling asleep* whereas the muscle tone is middle to high. The muscle tone is defined as the unconsciously constant contraction of the muscles helping the body to keep the posture. In the phase of falling asleep the the person gets hypnic jerks and the awake consciousness fades. Afterwards the sleeping person runs through the second stage, the phase of *light sleep*, where the muscle tone is middle to high but no awake consciousness is available anymore. The next stage ist the *middle sleep* phase. The muscle tone gets low and the ability of being woken up by something or someone is significantly reduced. The fourth stage is the *deep sleep* which is only a very small part of the whole night sleep. During this phase the muscle tone reduces and the ability of getting woken up is minimal. The fifth stage is called *REM sleep* the phase where the person dreams. During this phase the muscle tone is nearly suspended to prevent the person from injurys.
All sleeping phases form a cycle of 80-110 minutes. While sleeping each human runs through four to five complete cycles, where the depth of the sleep reduces from cycle to cycle and the duration of the fourth

[11] For further details about the Intelligent Bed, especially the technical implementation, see [Har09].

stage decreases while the duration of the fifth stage increases. In stage two and nearly awake phases the muscle tone is increased and hypnic jerks can occur.

Implementation details of the Intelligent Bed

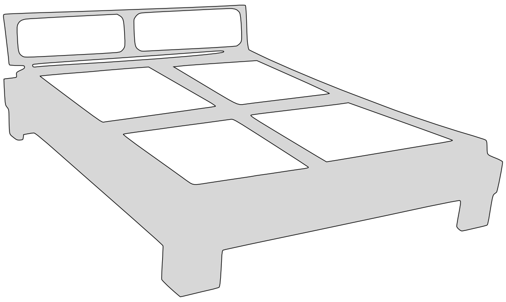


Figure 2.11: Implementation of sensors in bed, from [Har10a, p. 1]

The main functionality of the Intelligent Bed is the recognition of the light and deep sleep stages. Additionally, it recognizes, whether a person is sitting in the bed or no one is in the bed by evaluating the sensor values, so that no pressure exerted to any of the sensors means that no one is in the bed whereas the exertion of pressure to the sensors means that someone is in the bed. For the detection of the sleeping stages, six sensors will be used whereas the sensors will be situated as illustrated in figure 2.11: Two sensors will be situated at the head of the bed to identify a person sitting and leaning on the bed (e.g. while reading), the other four sensors will be mounted below of the matress to recognize the residents sleep stages.[12] The data will be collected as raw data and filtered to prevent inaccurate data because of measurment errors or strong deviations. Afterwards they will be classified and a semantical interpretation will be generated, e.g. “The user in a light sleep.”. As a result the Intelligent Bed sends an information to the Message Oriented Middelware about the sleep stage the person is in.

2.2.2.4.2 Possible User Operations

Like all other devices, the Intelligent Bed is able to be plugged in, unplugged and set into offline mode by the user. Furthermore, the inhabitant is able to indirectly enter the pressure values by sitting on, lying or moving in the Intelligent Bed. To supply pressure values for all six sensors, the user operation “enter pressure values” is provided.

2.2.2.5 Indoor-Positioning-System

In the present section, the *Indoor-Positioning-System* is presented according to figure 2.12 as shown in the following. First, the internal system behaviour is described, whereas the second part enumerates the possible user operations upon the Indoor-Positioning-System.

The possible interactions between the Indoor-Positioning-System and the Message Oriented Middleware are presented in figure 2.12, where the detailed descriptions of these possible interactions is given in section 2.2.1.2.

[12] See [Har10b], [Har10a] and [Har11].

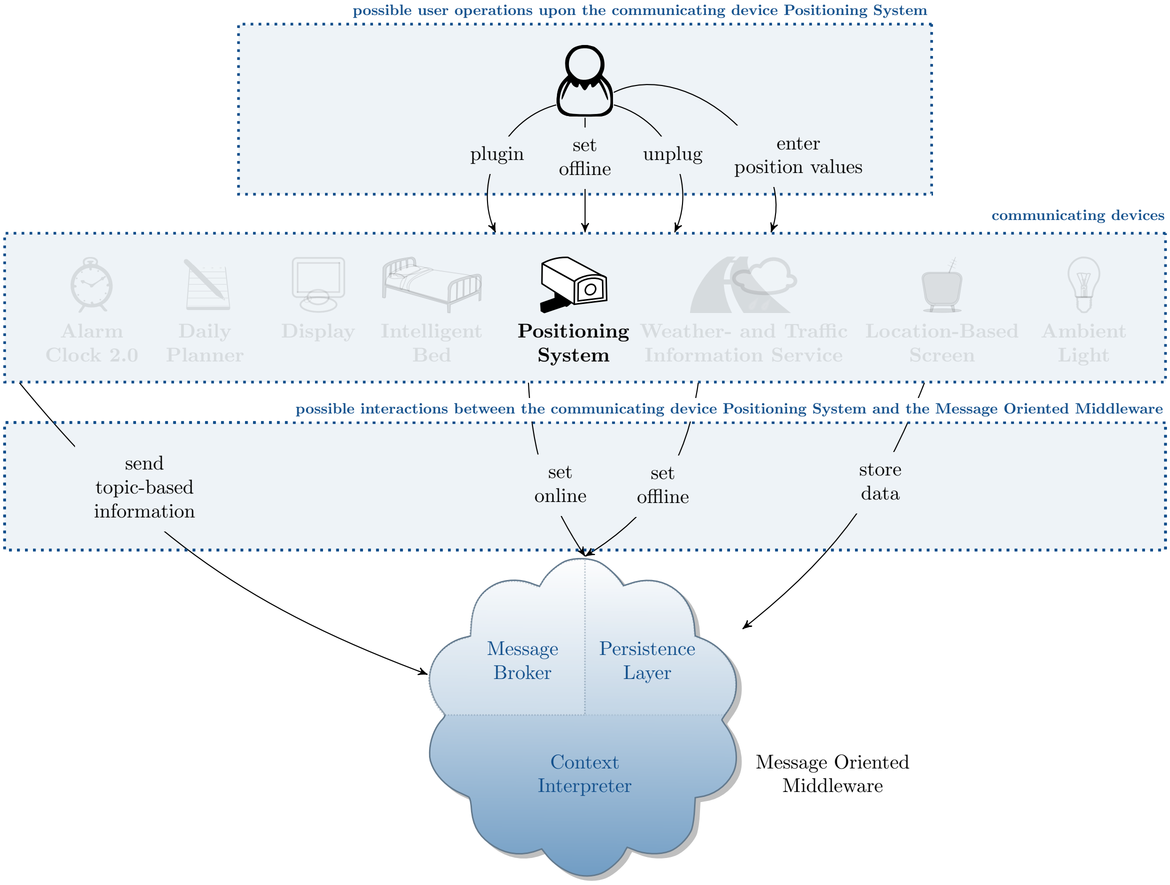

possible user operations upon the communicating device Positioning System
plugin
set offline
unplug
enter position values
communicating devices
Alarm Clock 2.0
Daily Planner
Display
Intelligent Bed
Positioning System
Weather- and Traffic Information Service
Location-Based Screen
Ambient Light
possible interactions between the communicating device Positioning System and the Message Oriented Middleware
send topic-based information
set online
set offline
store data
Message Broker
Persistence Layer
Context Interpreter
Message Oriented Middleware

Figure 2.12: Indoor-Positioning-System - an Overview

2.2.2.5.1 Internal System Behaviour

The Living Place will be equipped with an *Indoor-Positioning-System* which will be used by the Location-Based Screen. The Indoor-Positioning-System will be presented in the following and the Location-Based Screen in section 2.2.2.7.

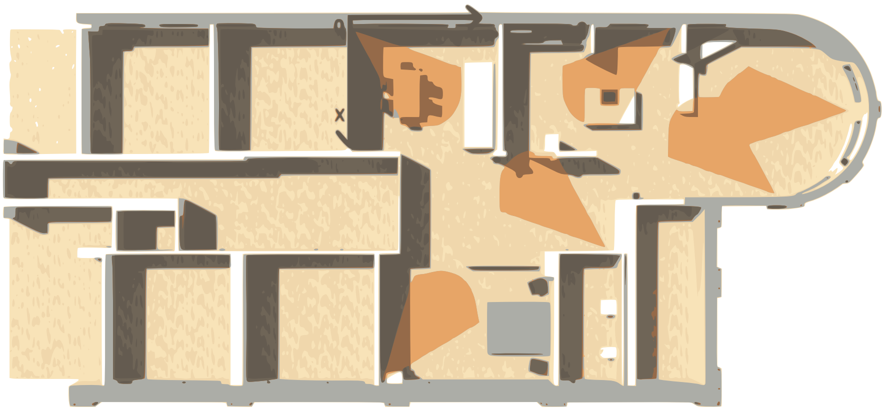


Figure 2.13: Position of indoor positioning system sensors in the Living Place flat, from [OV11a, p. 16]

The Indoor-Positioning-System consists of six electromagnetic sensors arranged as illustrated in figure 2.13. They will measure the current position of the inhabitant. Each sensor covers a specific range in the Living Place apartment, so that the combination off all sensors encirle the whole apartment. A sensor knows its own position and angle in the apartment (x,y,z-position, yaw, pitch and roll). After measuring a position of the user, provided as 3D-coordinate, each sensor sends a message containing its own identification and the measurement data (see [OV11a, p. 17]) to an application which interpretes

that data and sends its conclusion to the message broker.
Consequently, the Indoor-Positioning-System consists of six sensors and an application collecting and interpreting the data.

The apartment is splitted in four parts as described in table 2.2 and 2.3.

Range	y-coordinate
North	$10.0 \leq y \leq 15.8$
Middle	$5.5 \leq y < 10.0$
South	$0.0 \leq y < 5.5$

Table 2.2: Partition of the Living Place flat according to the y-coordinate of a location-based sensor

Range	x-coordinate
Middle	$0.0 \leq x \leq 8.0$
Bathroom	$8.0 < x$

Table 2.3: Partition of the Living Place flat according to the x-coordinate of a location-based sensor

The application interpretes the y-position of the inhabitant and compares the value of the y-position with 2.2. In the south, the sleeping area is situated, whereas in the lounge area lies in the north of the Living Place apartment (compare figure 2.13). Regarding the y-coordinate, the middle contains the kitchen area and the bathroom. To distinguish both areas, the x-coordinate will be interpreted according to the table 2.3. According to the results measured with the Indoor-Positioning-System, the application sends the information in which area the person stays to the message broker in using the topic “UbiTV”.

2.2.2.5.2 Possible User Operations

The Indoor-Positioning-System provides the user operations, plug in, unplug and set offline the device. Additionally it provides a user operation for entering the position values of the user. The sensors recognize the current position of the user and measure his position data. Therefore, the input of position data is indirect.

2.2.2.6 Weather- and Traffic Information Service

The *Weather and Traffic Information Service* are two different devices, but they operate in the same manner, so that they are presented together in this section. The following figure 2.14 illustrates the possible user operations provided for the user and the possible interactions between the communicating device and the Message Oriented Middleware. Each of both devices provide the same operations.

The possible interactions between the Weather- and Traffic Information Service and the Message Oriented Middleware are presented in figure 2.14, where the detailed descriptions of these possible interactions is given in section 2.2.1.2.

2.2.2.6.1 Internal System Behaviour

The Living Place has an internet connection from which it requests information from the outside of the Living Place apartment, in particular regarding weather reports or traffic news. The weather information system will provide information regarding the weather conditions in the direct surroundings of the apartment which might be received by other devices connected to the Message Oriented Middleware, so that their inference process can be influenced, e.g. the Alarm Clock 2.0 (see section 2.2.2.1): If the weather information system sends that currently there are bad weather conditions, like rain, storm or snow, and the inhabitant should be at work on time, the way to work might take longer because of the bad weather conditions, so the user should be waken-up some time earlier than normal.
Analog, the traffic service will influence the Alarm Clock 2.0, too, so that the user might be remembered to a certain appointment earlier than normally. E.g. bad weather and traffic conditions, like snow and traffic jams, might influence the Alarm Clock 2.0 in inferring that the alarm time will be modified, so that it lies a long time before the normal time.

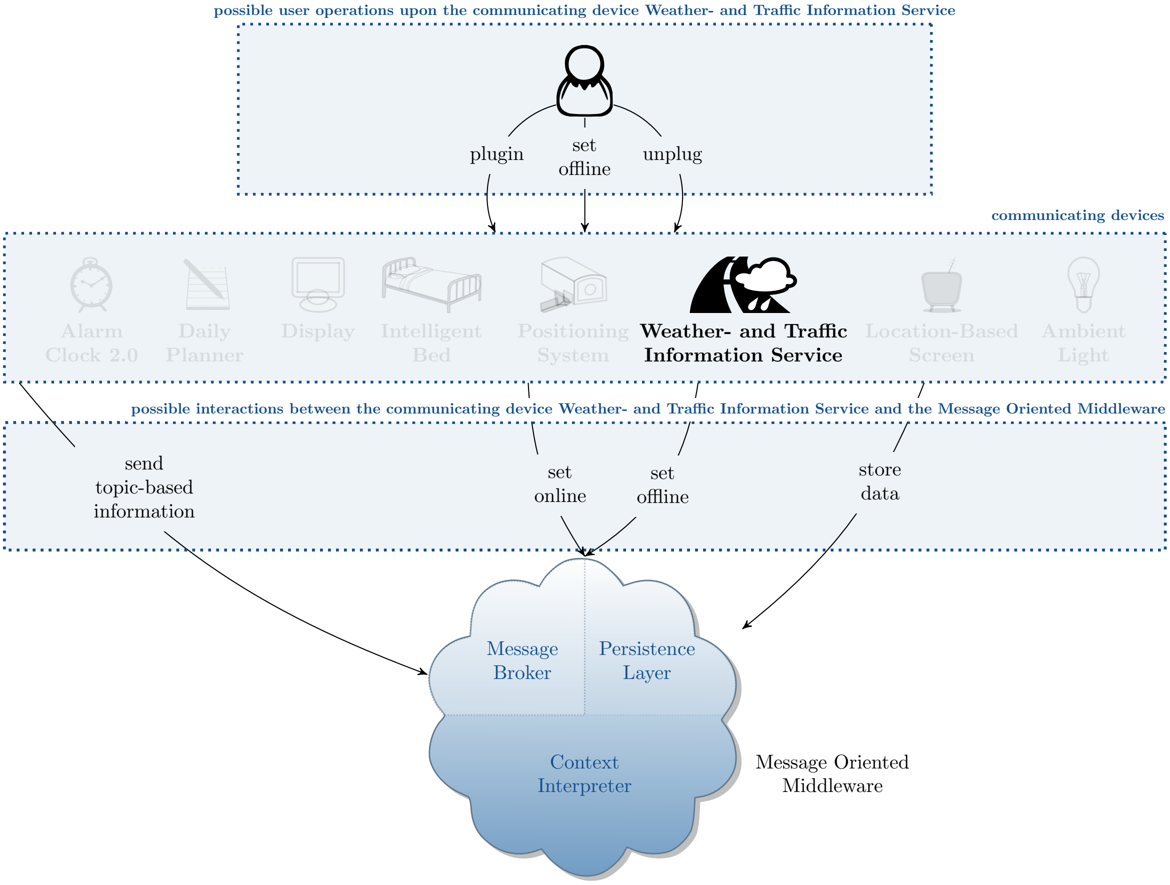

possible user operations upon the communicating device Weather- and Traffic Information Service
plugin
set offline
unplug
communicating devices
Alarm Clock 2.0
Daily Planner
Display
Intelligent Bed
Positioning System
Weather- and Traffic Information Service
Location-Based Screen
Ambient Light
possible interactions between the communicating device Weather- and Traffic Information Service and the Message Oriented Middleware
send topic-based information
set online
set offline
store data
Message Broker
Persistence Layer
Context Interpreter
Message Oriented Middleware

Figure 2.14: Weather- and Traffic Information Service - an Overview

2.2.2.6.2 Possible User Operations

The Weather Information Service resp. the Traffic Information Service only provide the standard user operations, plug in, plug off and set the device offline, which are available for each device.

2.2.2.7 Location-Based Screen

The *Location-Based Screen* will be presented in the following. The figure 2.15 gives a short overview of the possible user operations resp. possible user interactions between the Location-Based Screen connected to the Living Place system and the Message Oriented Middelware.

The possible interactions between the Location-Based Screen and the Message Oriented Middleware are presented in figure 2.15, where the detailed descriptions of these possible interactions is given in section 2.2.1.2.

2.2.2.7.1 Internal System Behaviour

The Living Place apartment is equipped with two television sets, one in the lounge area (north), the other one in the sleeping area (south). They are connected to an application which receives data belonging to the topic “UbiTV”, to which the device is subscribed, and evaluates it. The combination of all those parts mentioned, form the *Location-Based Screen.*
The device receives all messages sent by the Indoor-Positioning-System which is described in section 2.2.2.5. The message broker distributes all messages, so the application controlling the television set is able to fetch all data belonging to the topic “UbiTV”. It will interprete this data according to the following rules and activates or deactivates the television sets:

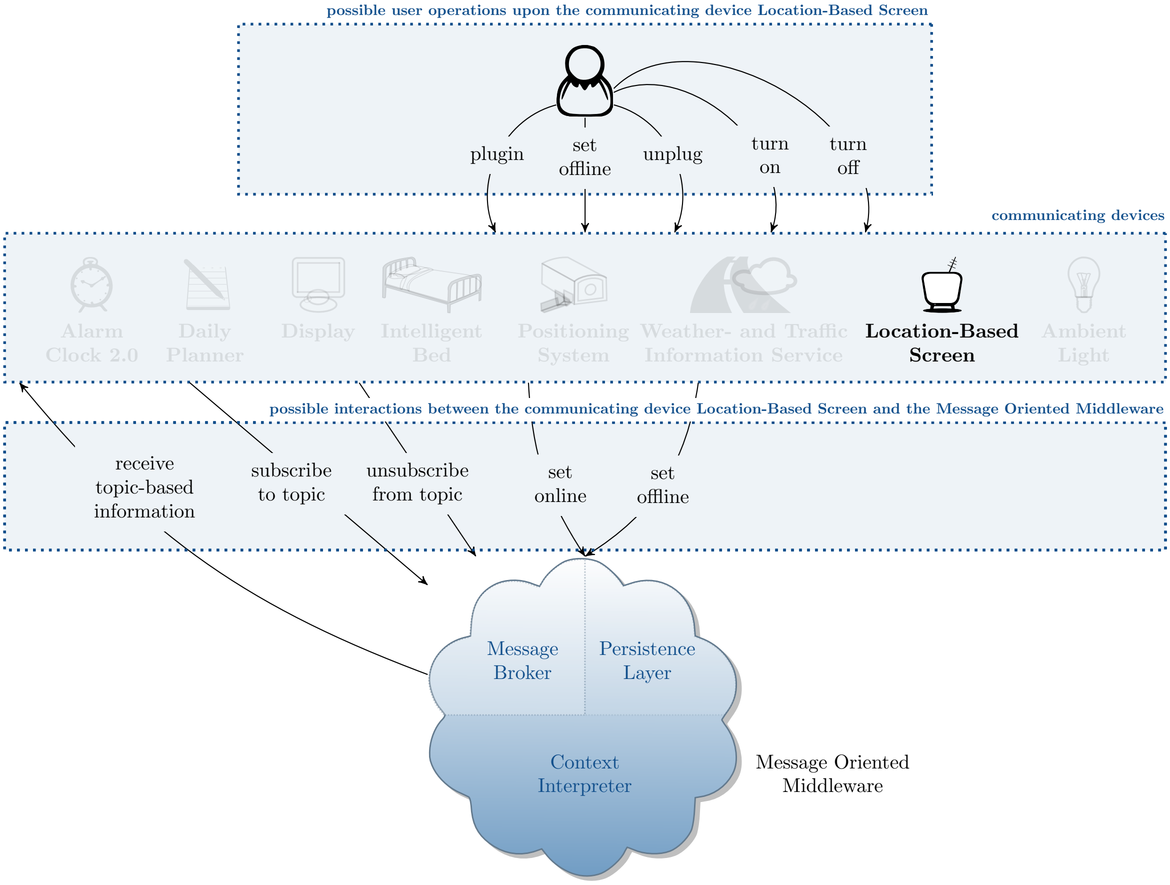

possible user operations upon the communicating device Location-Based Screen
plugin
set offline
unplug
turn on
turn off
communicating devices
Alarm Clock 2.0
Daily Planner
Display
Intelligent Bed
Positioning System
Weather- and Traffic Information Service
Location-Based Screen
Ambient Light
possible interactions between the communicating device Location-Based Screen and the Message Oriented Middleware
receive topic-based information
subscribe to topic
unsubscribe from topic
set online
set offline
Message Broker
Persistence Layer
Context Interpreter
Message Oriented Middleware

Figure 2.15: Location-Based Screen - an Overview

- If the resident is in the north, the TV set in the lounge area is activated, the other TV set will be deactivated.
- If the user resides in the south, the TV set in the sleeping area is activated, the other TV set is inactive.
- If neither the resident is in the north nor in the south, both TV sets should be switched off.

Both TV sets are only able to show the program alternatively. If one TV set is active, the other is inactive. But both TV sets have the same on/off state in order to switch the program from the screen of one TV to the screen of the other TV immediately if the user moves from one area containing a TV to the other one. The TV set needs position data of the inhabitant in order to work correctly. Currently, the position data of the user are measured by the Indoor-Positioning-System. The relationship between the Indoor-Positioning-System and the Location-Based Screen is shown in the following figure 2.16.

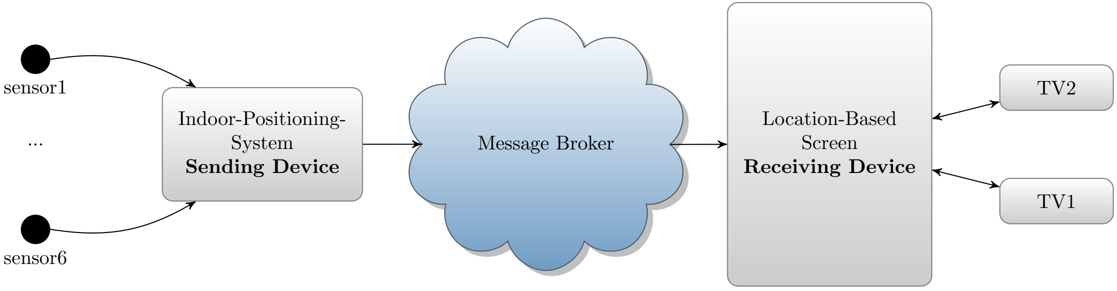


Figure 2.16: Relationship Between Indoor-Positioning-System and Location-Based Screen

2.2.2.7.2 Possible User Operations

The Location-Based Screen provides the user operations, which are available for all devices, i.e. *plugin*, *set offline* and *unplug*. Besides, it provides user operations for changing the on/off state of both TV sets simultaneously. One operation initiates the switching on of both TV sets (*turn on*), the other is the inverse and initiates the switching off of both TV sets (*turn off*).

2.2.2.8 Ambient Light

The *Ambient Light* will be presented in the following. The figure 2.17 gives a short overview of the possible user operations resp. possible user interactions between the Ambient Light connected to the Living Place system and the Message Oriented Middelware.

The possible interactions between the Ambient Light and the Message Oriented Middleware are presented in figure 2.17, where the detailed descriptions of these possible interactions is given in section 2.2.1.2.

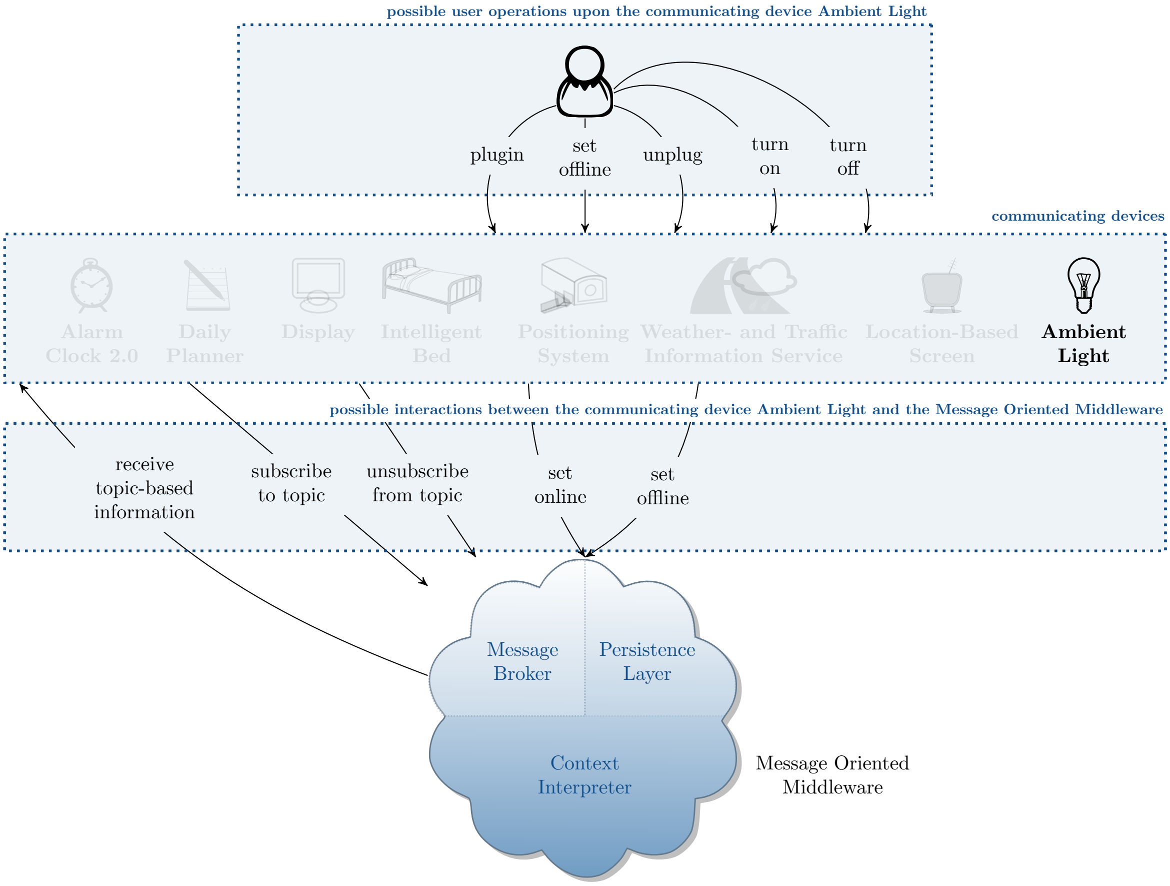


Figure 2.17: Ambient Light - an Overview

2.2.2.8.1 Internal System Behaviour

The Living Place will be equipped with an *Ambient Light* installation. According to the ubiquitousness of the Living Place the light in the apartment automatically switches on when an appropriate signal was sent over the Message Oriented Middleware. Currently the Ambient Light reacts on the alarm signals of the Alarm Clock 2.0 device (see 2.2.2.1) in switching the light on into dimmed mode.

2.2.2.8.2 Possible User Operations

Additionally to the automatic powering on of the Ambient Light in dimmed mode, the user is able to apply user operations for switching the light on (*turn on*) resp. off (*turn off*) in the whole Living Place apartment. Furthermore the user operations *plugin*, *unplug* and *setdeviceoffline* are provided.

2.3 Summary

This chapter introduces the Living Place Project, the Living Place and the System of the Living Place as a system of *Ubiquitous Computing* and *Ambient Intelligence*. For that, the basic characteristics of this system as well as the terms Ubiquitous Computing and Ambient Intelligence are described in section 2.1.

Furthermore, it is clarified that this work deals with the modelling of the internal system behaviour of the Living Place system.

In section 2.2 of this chapter the architecture of the system of the Living Place in terms of the system behaviour is described in detail, i.e. the components of the system, their possible relationships to each other as well as their most important possible direct relationships to entitites outside of the system are described.

Therefore, this description of the architecture of the system provides a detailed description of the system. So, the descriptions of sections 2.1 and 2.2 can serve as a basis for the modelling of the Living Place system.

The next chapter 3 is the first chapter that deals exclusively with the modelling of the Living Place system.

Chapter 3

Modelling the System of the Living Place: Levels of Modelling - Data Level, Object Level, System Level, User Level

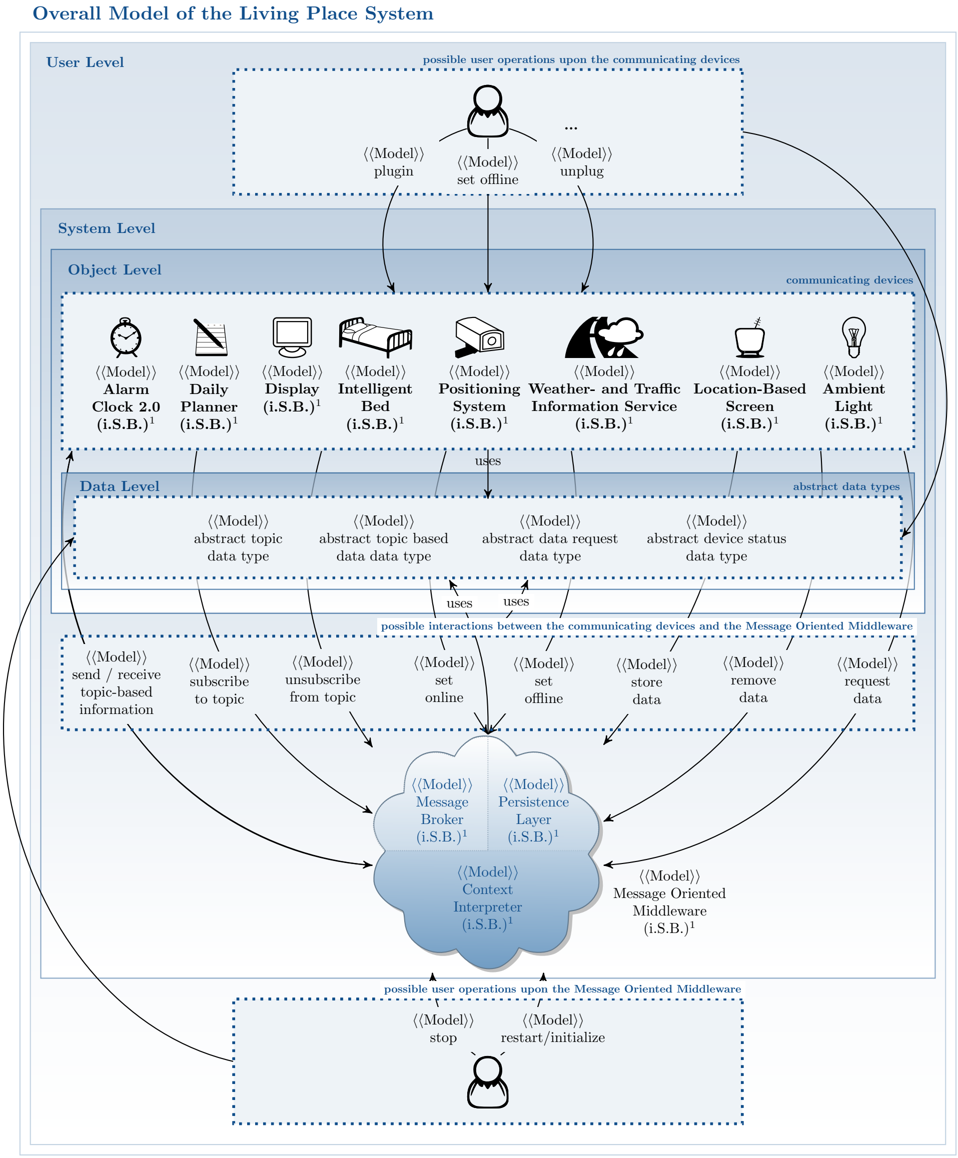

Overall Model of the Living Place System
User Level
possible user operations upon the communicating devices
...
⟨⟨Model⟩⟩ plugin
⟨⟨Model⟩⟩ set offline
⟨⟨Model⟩⟩ unplug
System Level
Object Level
communicating devices
⟨⟨Model⟩⟩ Alarm Clock 2.0 (i.S.B.)1
⟨⟨Model⟩⟩ Daily Planner (i.S.B.)1
⟨⟨Model⟩⟩ Display (i.S.B.)1
⟨⟨Model⟩⟩ Intelligent Bed (i.S.B.)1
⟨⟨Model⟩⟩ Positioning System (i.S.B.)1
⟨⟨Model⟩⟩ Weather- and Traffic Information Service (i.S.B.)1
⟨⟨Model⟩⟩ Location-Based Screen (i.S.B.)1
⟨⟨Model⟩⟩ Ambient Light (i.S.B.)1
uses
uses
Data Level
abstract data types
⟨⟨Model⟩⟩ abstract topic data type
⟨⟨Model⟩⟩ abstract topic based data data type
⟨⟨Model⟩⟩ abstract data request data type
⟨⟨Model⟩⟩ abstract device status data type
uses
uses
possible interactions between the communicating devices and the Message Oriented Middleware
⟨⟨Model⟩⟩ send / receive topic-based information
⟨⟨Model⟩⟩ subscribe to topic
⟨⟨Model⟩⟩ unsubscribe from topic
⟨⟨Model⟩⟩ set online
⟨⟨Model⟩⟩ set offline
⟨⟨Model⟩⟩ store data
⟨⟨Model⟩⟩ remove data
⟨⟨Model⟩⟩ request data
uses
⟨⟨Model⟩⟩ Message Broker (i.S.B.)1
⟨⟨Model⟩⟩ Persistence Layer (i.S.B.)1
⟨⟨Model⟩⟩ Context Interpreter (i.S.B.)1
⟨⟨Model⟩⟩ Message Oriented Middleware (i.S.B.)1
possible user operations upon the Message Oriented Middleware
⟨⟨Model⟩⟩ stop
⟨⟨Model⟩⟩ restart/initialize

Figure 3.1: Levels of Modelling - an Overview

[1] i.S.B. = internal System Behaviour

The description of the architecture of the Living Place system, given in chapter 2.2, provides a detailed description of the Living Place system by describing the components of this system, their possible relationships to each other as well as their most important possible direct relationships to entities outside of the system.

Therefore, this description of the system architecture serves as a basis for the modelling of the Living Place system, i.e. to obtain an overall model of the system, the elements of the system architecture resp. the system components, the possible relationships between them as well as their most important possible direct relationships to entities outside of the system have to be modelled, where the quantitative union of these so obtained models forms the overall model of the system. For that a description of these elements is required, which is given by the description of the system architecture in section 2.2. To illustrate the relationship between the description of the system architecture of section 2.2 and the modelling of the system, according to figure 2.4 of section 2.2, in figure 3.1 an overview of those system components and those relationships is given, that need to be modelled to obtain an overall model of the Living Place system.

Therefore, based on the statements of section 2.2, this chapter is intended to give an extended introduction to the modelling of the system of the Living Place by providing a cohesive overview of those system parts, that need to be modelled to obtain an overall model of the Living Place system as the main result of the modelling and therefore as a main result of this work.

First of all, the system components need to be modelled, i.e. since these components are considered as computing components, by modelling the system components the modelling of the internal system behaviours of these components is meant. Since, as shown in section 2.2, the system components are given by the Message Oriented Middleware together with the set of communicating devices, a **model of the internal system behaviour of the Message Oriented Middleware** together with models of the internal system behaviours of the communicating devices need to be elaborated, i.e. a **model of the internal system behaviour of the Alarm Clock 2.0**, a **model of the internal system behaviour of the Daily Planner**, a **model of the internal system behaviour of the Display**, a **model of the internal system behaviour of the Intelligent Bed**, a **model of the internal system behaviour of the Positioning System**, a **model of the internal system behaviour of the Weather- and Traffic Information Service**, a **model of the internal system behaviour of the Location-Based Screen** and a **model of the internal system behaviour of the Ambient Light** need to be elaborated. As the Message Oriented Middleware consists of the following three parts: **Message Broker**, **Persistence Layer** and **Context Interpreter**, a model of the internal system behaviour of each of these three parts need to be elaborated to obtain a **model of the internal system behaviour of the Message Oriented Middleware**.

Furthermore, the mentioned relationhips have to be modelled. It is stated in section 2.2 that the possible relationships between the system components resp. the most important possible direct relationships from the system components to entities outside of the system are given by the different possible interactions between the communicating devices and the Message Oriented Middleware resp. the possible user operations upon the communicating devices together with the possible user operations upon the Message Oriented Middleware. Therefore, a model of each possible interaction as well as a model of each possible user operation upon the communicating devices resp. upon the Message Oriented Middleware need to be elaborated, that includes a **model of send/receive topic-based information**, a **model of subscribe to topic**, a **model of unsubscribe from topic**, a **model of set online**, a **model of set offline**, a **model of store data**, a **model of remove data** and a **model of request data** in terms of the possible interactions and a **model of stop**, a **model of restart/initialize** resp. a **model of plugin**, a **model of set offline** and a **model of unplug** in terms of the possible user operations. Consider, that several more possible user operations upon the communicating devices are defined in sections 2.2.2.1 to 2.2.2.8, which however are omitted in this section, to be able to give a clearer overview. Therefore, additionally for every such possible user operation a model need to be elaborated. Also consider, that the models of the possible user operations upon the system of the Living Place only consider those parts of these user operations, that proceed inside of the system of the Living Place, since this work deals with the modelling of the internal system behaviour of this system, i.e. since every such user operation causes immediately a certain change of the internal state of this system, these models depict these changes of the internal state of this system which are caused by these user operations.

Finally, since all the mentioned models involve data and operations that are defined on these data, the abstract data types, that includes these data and operations also need to be modelled so that the mentioned models can use these so obtained models of the abstract data types, i.e. in particular a **model of the abstract topic data type**, a **model of the abstract topic based data data type**, a **model**

of the abstract data request data type and a **model of the abstract device status data type** need to be elaborated. Which model involves which data and operations and therefore uses which models of the abstract data types is clarified in chapter 7 by defining these models in detail therein. However, an example is given here in advance: Every model of the internal system behaviour of a communicating device uses the model of the abstract device status data type and especially the therin defined device status data, which indicate whether the respective device is in state online, offline or any other state.

As illustrated in figure 3.1, the quantitative union of all the so far mentioned models forms the **overall model of the Living Place system**. Thus, they can be considered as submodels of the overall system model. So the elaboration of these submodels, as it is especially carried out in chapter 7 by specifically defining these submodels, delivers the overall model of the system as a main result of this work.

Figure 3.1 shows a classification of the submodels into the following four levels: **Data Level, Object Level, System Level** and **User Level**, where these levels are organized hierarchically to each other by the following hierarchical relation: the Data Level, as the lowest level, is contained in the Object Level, the Object Level is contained in the System Level and the System Level is contained in the highest level resp. the User Level, so that the models classified in lower levels are used by models classified in the corresponding directly higher level in a manner as it is described in sections 3.1, 3.2, 3.3 and 3.4.

So, this classification of the models provides an organization of the models into appropriate categories, where these categories are organized hierarchically to each other, so that the models of lower categories are used by models of the corresponding directly higher category.

Thus, this classification of the models provides the answer of the question "Which model is used by which model?" and therefore also provides a well-ordered overall overview of these models and their important relationships to each other.

The classification is described in detail in the following sections 3.1 to 3.4.

3.1 Data Level

The lowest level - the **Data Level** - contains the models of the abstract data types, which define the data and functions on these data, that are used by all the models of higher levels such as the models of the internal system behaviours of the communicating devices of the Object Level, the models of the possible interactions and the internal system behaviour of the Message Oriented Middleware of the System Level as well as the models of the possible user operations of the User Level.

Since the models of the abstract data types are used by all the models of higher level, the Data Level is contained in the Object Level, since the Object Level is contained in the System Level, the Data Level is contained in the System Level and analogously the Data Level is also contained in the User Level.

3.2 Object Level

The **Object Level** includes the models of the internal system behaviours of the communicating devices, which are considered as the objects of the Living Place system, and as mentioned before the Object Level also includes the Data Level and therefore the models of the abstract data types contained in the Data Level, since the models of the internal system behaviours of the communicating devices uses these models of the abstract data types by operating on the data and functions that are defined by these models.

Since the models of the internal system behaviours of the communicating devices are considered, in this work the communicating devices are considered as computing systems. The internal system behaviour of such a device includes all possible transformations of the possible internal states of that device into new possible internal states by applying actions. Therefore, a model of the internal system behaviour of such a device must describe all possible internal states of that device as well as the possible transformations of these states by describing all possible actions that can be applied as well as which internal states must prevail, so that such an action can be applied and which internal state results from the application of such an action.

3.3 System Level

The **System Level** includes the model of the internal system behaviour of the Message Oriented Middleware and therefore includes also the following models, that constitute this model: the model of the internal system behaviour of the Message Broker, the model of the internal system behaviour of the Context Interpreter as well as the model of the internal system behaviour of the Persistence Layer.

Since the model of the internal system behaviour of the Message Oriented Middleware is considered, in this work the Message Oriented Middleware is considered as a computing system in a manner analogous to the communicating devices that are depicted by the models of the Object Level in section 3.2,

What such an internal system behaviour and therefore its model must include, is described in the previous section 3.2.

Furthermore this level contains the models of the possible interactions between the communicating devices and the Message Oriented Middleware as well as the Object Level and therefore the models of these levels, since the models of the possible interactions relate the model of the internal system behaviour of the Middleware to the models of the internal system behaviours of the communicating devices that are defined in the Object Level because these models of the internal system behaviours include descriptions of those actions that are involved in the possible interactions. So, the one part of the model of such a possible interaction is included in the model of the internal system behaviour of the Middleware and the other part is included in the model of the internal system behaviour of the corresponding communicating device. However, the models of the possible interactions are contained in this level, because these models use the models of the internal system behaviours of the communicating devices by using the therein described possible internal states of these devices to indicate the result of the corresponding interaction.

Moreover, since the System Level contains the Object Level, the System Level also contains the Data Level and its models, since the models of the System Level use the models of the abstract data types contained in the Data Level by operating on the data and functions that are defined by these models.

Consider, that the model of the internal system behaviour of the Message Oriented Middleware and the models of the possible interactions, that are contained in this level, are considered as the central parts of the Living Place system.

3.4 User Level

The highest level - the **User Level** - includes the models, that represent the considered possible operations a human user can perform on the system, i.e. the models of the possible user operations upon the communicating devices resp. upon the Message Oriented Middleware.

Since, as depicted in section 2.2, these user operations initiate a change of one or more internal states of the system and therefore the models of these user operations operate on the models of the internal system behaviours of the communicating devices resp. the Message Oriented Middleware. The User Level contains the System Level as well as the Object Level and therefore also the therein contained models of the internal system behaviours.

Since, the User Level contains the System- and Object Level, the User Level also contains the Data Level and its models, since the models of the possible user operations of the User Level use the models of the abstract data types contained in the Data Level by operating on the data and functions that are defined by these models.

3.5 Summary

This chapter presents an overview of those models, that need to be elaborated to obtain an overall model of the Living Place system as a main result of this work. Since every model has a reference to that part of the system which is considered by the model, this overview is also an overview of those system parts, that need to be modelled to obtain an overall model of the Living Place system.

Furthermore, in this chapter, a classification of these models into the following four levels is given: Data Level, Object Level, System Level and User Level. Therefore, these levels are considered as the levels of modelling. This classification provides a well-ordered overall overview of the models and their important relationships to each other by organizing these models into appropriate categories and illustrating which model is used by which model.

Figure 3.1 illustrates this classification and therefore gives a pictorial overview of those models, that constitute the overall model of the Living Place system and thus can be considered as submodels of the overall model.

The classification of these models into the considered four levels, will be maintained in all further considerations relating to the modelling of the Living Place system.

The models of the Data Level are formulated in sections 7.1.1 and 7.2.1. In sections 7.1.2 and 7.2.2 the models of the Object Level are elaborated. In sections 7.1.3 and 7.2.3 the models of the System Level are defined and in sections 7.1.4 and 7.2.4 the models of the User Level are given.

Chapter 4
Modelling the System of the Living Place: Requirements Towards the Model

As illustrated in section 3, the overall model of the Living Place system is constituted of several submodels which represent different parts of the system, where the elaboration of these submodels delivers the overall model of the system as a main result of this work.

With regard to the elaboration of these submodels in chapter 7, this chapter describes requirements, that must be fulfilled by the submodels of the overall system model. Thus, the descriptions in this chapter relate directly to the definitions of these submodels, as they are given in chapter 7.

Since the quantitative union of the submodels forms the overall modell of the system, the set of the requirements towards these submodels represents the set of requirements towards the overall model of the Living Place system - therefore, requirements towards these submodels are considered as requirements towards the overall model of the system.

The requirements towards the submodels and therefore towards the overall model of the Living Place system result on the one hand from the description of the Living Place system resp. from the descriptions of those parts of the system from section 2.2, that are represented by these submodels and on the other hand, since the system of the Living Place is a system of ubiquitous computing and ambient intelligence, from the characteristics of such ubiquitous computing resp. ambient intelligence systems, that are given in section 2.1.

Therefore, in section 4.2 the requirements that result from the description of the Living Place system are formulated and section 4.1 describes the requirements that result from the characteristics of ubiquitous computing resp. ambient intelligence systems.

In sections 4.2 and 4.1, the requirements towards the submodels are described in seperated subsections by maintaining the classification of the submodels into the four levels Data Level, Object Level, System Level and User Level, as presented in chapter 3. The requirements towards the models of the Data Level, Object Level, System Level resp. User Level that result from the description of the Living Place system are described in sections 4.2.1, 4.2.2, 4.2.3 resp. 4.2.4. The requirements towards the models of the Data Level, Object Level, System Level resp. User Level that result from the characteristics of ubiquitous computing resp. ambient intelligence systems are described in sections 4.1.1, 4.1.2, 4.1.3 resp. 4.1.4.

4.1 Requirements Resulting From the Characteristics of Ubiquitous Computing resp. Ambient Intelligence Systems

As described in section 2.1, the system of the Living Place includes the characteristics of a ubiquitous computing and ambient intelligence system. Therefore, this system can be considered as a system of Ubiquitous Computing and Ambient Intelligence. Thus, a model of the system of the Living Place must reflect these characteristics. For that, such a model must represent these characteristics, i.e. the representations of these characteristics can be considered as requirements towards the model of the Living Place system.

4.1.1 Data Level

$\mathbf{r_{ubid_1}}$: Since the communicating devices are able to communicate and each communication process involves data, the data of these communication processes must be represented individually for each communicating device.

4.1.2 Object Level

$\mathbf{r_{ubio_1}}$: All devices connected to the system are exchangeable without interrupting or influencing the processing of any other devices.

$\mathbf{r_{ubio_2}}$: All devices are able to communicate with each other.

$\mathbf{r_{ubio_3}}$: The prevailing context of the Living Place will be taken into account by the communicating devices within their processing steps.

4.1.3 System Level

$\mathbf{r_{ubis_1}}$: All devices connected to the system are exchangeable without interrupting the data processing within the Message Oriented Middleware.

$\mathbf{r_{ubis_2}}$: The communication of all devices will be guaranteed by the Message Oriented Middleware.

$\mathbf{r_{ubis_3}}$: In general, the Message Oriented Middleware has to react in an appropriate manner to user operations.

4.1.4 User Level

$\mathbf{r_{ubiu_1}}$: The user is able to exchange devices and therefore the Message Oriented Middleware has to react in an appropriate manner.

$\mathbf{r_{ubiu_2}}$: In general, users of the system have the possibility to provoke a reaction of the system in applying user operations.

$\mathbf{r_{ubiu_3}}$: The user can always take control over the system by shutting it down or restarting the whole system.

4.2 Requirements Resulting From the Description of the System

A model of the internal system behaviour of the Living Place system must reflect all important aspects of that behaviour. Therefore, the descriptions of the system architecture in terms of the internal system behaviour of section 2.2 serves as a basis to formulate requirements towards such a model.

In the following discussion of this chapter, such requirements towards the model resulting from the description of the system of section 2.2 are given.

As the model consists of several sub-models, for each sub-model the requirements are seperately presented. An overview of all sub-models is given in figure 3.1 of the previous section 3.

The sum of requirements of these sub-models form the set of requirements for the overall model of the Living Place project.

The following chapter is organized as follows: Section 4.2.2 describes the requirements towards the models of the Object Level, i.e. the general requirements regarding the device types: receiver, sender and transceiver. Afterwards the requirements regarding each concrete model will be worked out, i.e. of the Alarm Clock 2.0 in section 4.2.2.4, the Display in 4.2.2.5, the Daily Planner in 4.2.2.6, the Intelligent Bed in 4.2.2.7, the Indoor-Positioning-System in section 4.2.2.8, the Weather Information System in 4.2.2.9, the Traffic Information System in 4.2.2.10, the Location-Based Screen in 4.2.2.11 and the Ambient Light in 4.2.2.12.

Section 4.2.3 describes the requirements towards the models of the System Level, i.e. the requirements regarding to the model of the message broker in section 4.2.3.1, the requirements regarding to the model of the persistence layer in section 4.2.3.2 and the requirements towards the model of the context interpreter in section 4.2.3.3.

$\mathbf{r_{gen}}$: First of all, independently of all other requirements, it would be preferable, that the levels of modelling, as presented in section 3, are reflected by the overall model of the Living Place system itself in an intuitive way.

4.2.1 Data Level

$\mathbf{r_{data_1}}$: The communication in the Living Place system between all connected devices takes place with the help of topics (see 2.2.1.3), hence, the model on the data level has to provide a data type for topic with a corresponding carrier set enabling identifiers for all relevant topics.

4.2.2 Object Level

In chapter 2, the Living Place system and its components are introduced, whereas the following three device types are introduced:

- Transceiver,
- Receiver,
- Sender.

Each device belongs to one of the specific types. Therefore, general requirements regarding these three device types, i.e. transceiver, receiver and sender, will be worked out in this section. The reqirements regarding the corresponding type have to be fulfilled by the concrete device, which is either a transceiver, a receiver or a sender.
Additionally, each device has to fulfil specific requirements, which will be presented afterwards for each device depending on the description of the specific device presented in chapter 2. The devices which will be analysed are:

- the Alarm Clock 2.0, which is introduced in 2.2.2.1,
- the Display in the Multitouch Kitchen Counter, which is presented in 2.2.2.2,
- the Daily Planner, which is illustrated in 2.2.2.3,
- the Intelligent Bed, see section 2.2.2.4,
- the Indoor-Positioning-System, whose description is in 2.2.2.5,
- the Weather- and the Traffic Information Service, which are presented in section 2.2.2.6,
- the Location-Based Screen, introduced in section 2.2.2.7 and
- the Ambient Light, presented in section 2.2.2.8.

E.g. the Alarm Clock 2.0 is a transceiver, therefore it should fulfil the general requirements of a transceiver presented in section 4.2.2.3, furthermore is should fulfil its specific requirements established for the Alarm Clock 2.0, which are presented in section 4.2.2.4.

4.2.2.1 Receiver

$\mathbf{r_{receiver_1}}$: Each device should be able to initiate a change from offline into online mode and reverse. The online resp. offline mode should be stored internally by the device.

$\mathbf{r_{receiver_2}}$: Each device contains a specific connection identification which will be set by the Message Oriented Middleware when the device is connected to the Living Place system and switches into online mode.

$\mathbf{r_{receiver_3}}$: In online mode, the receiver should be able to subscribe itself to a specific topic. Additionally, the reverse, i.e. the device should be able to unsubscribe from already subscribed topics, should be possible.

$\mathbf{r_{receiver_4}}$: The device should be able to receive topic-based data distributed by the Message Oriented Middleware.

$\mathbf{r_{receiver_5}}$: Each device, which is able to receive data, should be able to request data regarding one or more topics, to which the device is subscribed, from the persistence layer. Therefore, a receiver should get the possibility to request stored data from the persistence layer of the Message Oriented Middleware.

$\mathbf{r_{receiver_6}}$: Receiving data, requesting data and subscribing topics resp. announcing a switch into offline mode can proceed in parallel. As long as the device is not switched into offline mode, because the Message Oriented Middleware did not process this announcement yet, the device is able to process messages, requests to the persistence layer resp. subscribe topics.

4.2.2.2 Sender

$\mathbf{r_{sender_1}}$: Each device should be able to initiate a change from offline into online mode and reverse. The online resp. offline mode should be stored internally by the device.

$\mathbf{r_{sender_2}}$: Each device contains a specific connection identification which will be set by the Message Oriented Middleware when the device is connected to the Living Place system and switches into online mode.

$\mathbf{r_{sender_3}}$: A sender is able to send data to the Message Oriented Middleware for processing.

$\mathbf{r_{sender_4}}$: Additionally, the data which will be sent to the Message Oriented Middleware should be stored to the persistence layer.

4.2.2.3 Transceiver

A transceiver is a combination of a receiver and a sender. Consequently, the requirements posed for all receivers and for all senders should be fulfilled by all transceivers, but because of the transceiver device type being a combination of both device types, i.e. receiver and sender, the requirements fulfilled by the receiver device type resp. sender device type, will be automatically met by all transceiving devices.
In addition, the transceiver should fulfil the following requirement resulting out of the combination of the other two device types:

$\mathbf{r_{transceiver_1}}$: The transceiver is able to send, receive and request data, subscribe to topics, and announcing that the device wants to go into offline mode in parallel. Similar to the receiver, it is possible for a transceiver to process data, i.e. sending, receiving and requesting information, and to subscribe topics, even if a switch into offline mode is announced, as long as the Message Oriented Middleware has not processed this announcement.

4.2.2.4 Alarm Clock 2.0

$\mathbf{r_{AlarmClock2.0_1}}$: Since the Alarm Clock 2.0 is able to receive topic-based information, it is able to subscribe to and unsubscribe from the following topics: “Weather”, “Traffic”, “Calendar”, “Bed” and “Context”.

$\mathbf{r_{AlarmClock2.0_2}}$: The Alarm Clock 2.0 device does not alert by itself, instead it sends a signal, that an alarm $(on, Alarm)$ should be provoked, to the Message Oriented Middleware, belonging to the topic “Alarm”. Before sending, it converts the signal into a global format.

$\mathbf{r_{AlarmClock2.0_3}}$: It is able to request past data regarding the topic “Calendar” in order to get all appointments entered to the Living Place system. This kind of data forms the basis for inferring an appropriate moment for alarming.

$\mathbf{r_{AlarmClock2.0_4}}$: The Alarm Clock 2.0 starts to check for the next appointment 240 minutes before the appointment starts. If the timestamp given in the calendar data is less or equal than 240 minutes, take this data.

$\mathbf{r}_{\mathbf{AlarmClock2.0_5}}$: The Alarm Clock 2.0 automatically infers a wake up time resp. a moment for reminding the user initially according to calendar data belonging to the topic "Calendar" and regarding traffic resp. weather conditions occuring during the time of getting to the appointment. According to the defined additional time for the weather resp. the traffic conditions, mentioned in the following tables 4.1 resp. 4.2, which occurs during the time of getting to the appointment, additional time will be taken into account. For example, if the weather conditions are bad because it is snowing and besides a traffic jam occurs, an extra preparation time of 60 minutes must be taken into account. In contrast, good weather and traffic conditions do not need any further time for preparation.

Weather Condition	**Additional Time (in Minutes)**
sun	0
wind	0
rain	10
storm	15
hail	15
snow	30

Table 4.1: Additional Times for Different Weather Conditions

Traffic News	**Additional Time (in Minutes)**
free	0
roadwork	15
jam	30
accident	30

Table 4.2: Additional Times for Different Traffic Conditions

$\mathbf{r}_{\mathbf{AlarmClock2.0_6}}$: In addition, the Alarm Clock 2.0 should evaluate data according the current sleeping phase the user is in, in order to generate an appropriate wake up time. If the inhabitant is in depth sleep phase, the Alarm Clock 2.0 should wait and check regularily until this sleeping phase is finished. If the user is in any other sleeping phase, the Alarm Clock 2.0 does not need to wait and is able send an alarm signal immediately.

$\mathbf{r}_{\mathbf{AlarmClock2.0_7}}$: Because of the possible case, that the person is currently in the depth sleep phase and the Alarm Clock 2.0 needs to wait until this phase is finished, an additionaly time buffer of 30 minutes is assumed for infering a wake up time.

$\mathbf{r}_{\mathbf{AlarmClock2.0_8}}$: The device contains an internal storage for administering the four tasks the user should fulfil during the preparation time: getting up, washing, dressing and eating. A further description of all states will be given in table 2.1. An average duration of 15 minutes is assumed for each task, so, every 15 minutes, the Alarm Clock 2.0 reminds the user of fulfiling the actions, as long as there are still any open.
Because of the device managing four tasks with a duration of 15 minutes for each, a overall duration of the preparation time of 60 minutes is assumed.

$\mathbf{r}_{\mathbf{AlarmClock2.0_9}}$: The Alarm Clock 2.0 constantly checks for 5W1H context data. If a wake up time is inferred, the Alarm Clock 2.0 evaluates all incoming context data. If a context data describes a task, which is currently performed by the person, this task will be removed from the internal storage of tasks.

$\mathbf{r}_{\mathbf{AlarmClock2.0_{10}}}$: The Alarm Clock 2.0 constantly checks for 5W1H context data. If the Alarm Clock 2.0 is in idle mode, that means, no valid appointment is selected, all incoming context data need to be removed.

$\mathbf{r}_{\mathbf{AlarmClock2.0_{11}}}$: It is assumed, that the way to get to the appointment takes 30 minutes. Data according to weather and traffic conditions only need to be checked for the period of time between 30 minutes before the appointment plus the already inferred additional time and the appointment.

$\mathbf{r}_{\mathbf{AlarmClock2.0_{12}}}$: The Alarm Clock 2.0 contains an internal timer, representing the TimeModule, which provides an actual timestamp. This timestamp can be compared to timestamps within the information the Alarm Clock 2.0 receives from the Message Oriented Middleware, like information regarding the topics "Weather", "Traffic" and "Calendar".

$\mathbf{r_{AlarmClock2.0_{13}}}$: The device is able to end the constant reminding, if no open tasks are available, which need to be fulfilled. The reminding takes place in generating a new (on,Alarm) signal and sending it to the Message Oriented Middleware.

$\mathbf{r_{AlarmClock2.0_{14}}}$: The device is able to end the constant reminding, if the time for leaving the Living Place apartment is reached. This moment is calculated out of: start of the appointment - (additional time because of weather and traffic conditions + 30 minutes for getting to the appointment).

$\mathbf{r_{AlarmClock2.0_{15}}}$: If the time of leaving the apartment is reached, the Alarm Clock 2.0 resets itself automatically. A reset of the Alarm Clock 2.0 contains:

- Deleting the currently processed appointment.
- Removing the current timestamp for reminding.
- Resetting the internally stored tasks.
- Switching into idle mode in order to delete all incoming 5W1H context data.
- Reset waiting time because of bad weather or bad traffic conditions.

$\mathbf{r_{AlarmClock2.0_{16}}}$: Additionally, the user is able to stop and therefore reset the Alarm Clock 2.0. For further details see 4.2.4.2.1.

$\mathbf{r_{AlarmClock2.0_{17}}}$: The Alarm Clock 2.0 is a transceiver.

4.2.2.5 Display

$\mathbf{r_{display_1}}$: Since the Display device is able to receive topic-based information and should visualize calendar, weather and traffic data, it is able to subscribe to and unsubscribe from the following topics: “Weather”, “Traffic” and “Calendar”.

$\mathbf{r_{display_2}}$: It needs to convert received data, belonging to the subscribed topics, into a specific format for visualizing them.

$\mathbf{r_{display_3}}$: The user is able to remove single data records from the display of the device. Also see 4.2.4.2.2.

$\mathbf{r_{display_4}}$: The Display is a receiver.

4.2.2.6 Daily Planner

$\mathbf{r_{planner_1}}$: The user is able to enter new appointments to the Daily Planner. Each information contains a timestamp quoting the begin of the event and a description of the event. The user enters the data by an user operation, see 4.2.4.2.3.

$\mathbf{r_{planner_2}}$: The user is able to delete single data which was previously entered into the Daily Planner. For furthre details see 4.2.4.2.3.

$\mathbf{r_{planner_3}}$: The Daily Planner converts the specific data into a global format.

$\mathbf{r_{planner_4}}$: The Daily Planner is a sender.

4.2.2.7 Intelligent Bed

$\mathbf{r_{bed_1}}$: The Intelligent Bed receives pressure data, which were enterd by a user operation into the device, representing the measurement of pressure data. For further details see 4.2.4.2.4.

$\mathbf{r_{bed_2}}$: The Intelligent Bed processes six different values, each representing one of the six pressure sensors.

$\mathbf{r_{bed_3}}$: To detect a movement, the Intelligent Bed always needs to store the last measurement data, made of a set of six values for each sensor, in order to compare the new data with the old one.

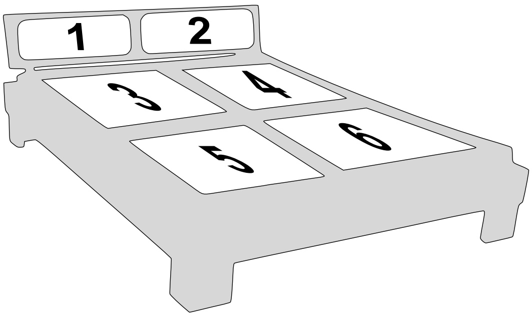


Figure 4.1: Numbering of the sensors in the Intelligent Bed (original image from [Har10a, p. 1])

$\mathbf{r_{bed_4}}$: The pressure of each sensor is measured in a range of $[0, ..., 10]$.

$\mathbf{r_{bed_5}}$: To detect human pressure, a threshold of 6 is set which needs to be exceeded by each sensor value in order to be a valid pressure data. If the value is below of this threshold, it will be treated as zero. E.g. slight pressure might be supplied by the pillow or bedcover, etc.

$\mathbf{r_{bed_6}}$: The pressure data will be interpreted if one of the following rules is fulfilled:

- Sensor 1 exceeds the threshold ⇒ The person is awake.
- Sensor 2 exceeds the threshold ⇒ The person is awake.
- Sensor 3 and Sensor 5 both exceed the threshold and in comparison with the last values, both sensor values did not change ⇒ The person is in depth sleep phase.
- Sensor 4 and Sensor 6 both exceed the threshold and in comparison with the last values, both sensor values did not change ⇒ The person is in depth sleep phase.
- Sensor 3 and Sensor 5 both exceed the threshold and in comparison with the last values, both sensor values changed ⇒ The person is in light sleep phase.
- Sensor 4 and Sensor 6 both exceed the threshold and in comparison with the last values, both sensor values changed ⇒ The person is in light sleep phase.
- If none of all sensors exceeded the threshold ⇒ The person is not in bed.

For the location of the pressure sensors, see image 4.2.2.7.

$\mathbf{r_{bed_7}}$: The resulting interpretation will be transformed from the device specific data into a global format.

$\mathbf{r_{bed_8}}$: The Intelligent Bed is a sender.

4.2.2.8 Indoor-Positioning-System

$\mathbf{r_{indoor_1}}$: The Indoor-Positioning-System receives its measurement data by applying a user operation, see 4.2.4.2.5.

$\mathbf{r_{indoor_2}}$: The data will be sent as 3D-position data for each sensor.

$\mathbf{r_{indoor_3}}$: It analyses the data according to the following rules:

- If the y-value of sensor 1 y_1 is in $0.0 \leq y_1 < 5.5 \Rightarrow$ The user is in the south of the apartment.
- If the y-value of sensor 2 y_2 is in $0.0 \leq y_2 < 5.5 \Rightarrow$ The user is in the south of the apartment.
- If the y-value of sensor 5 y_5 is in $10.0 \leq y_5 \leq 15.8 \Rightarrow$ The user is in the north of the apartment.
- If the y-value of sensor 6 y_6 is in $10.0 \leq y_6 \leq 15.8 \Rightarrow$ The user is in the north of the apartment.
- If the y-value of sensor 3 y_3 is in $5.5 \leq y_3 < 10.0$ and the x-value of this triple $x_3 \leq 8.0 \Rightarrow$ The user is in the middle of the apartment.
- If the y-value of sensor 4 y_3 is in $5.5 \leq y_4 < 10.0$ and the x-value of this triple $x_4 \leq 8.0 \Rightarrow$ The user is in the middle of the apartment.
- If the y-value of sensor 3 y_3 is in $5.5 \leq y_3 < 10.0$ and the x-value of this triple $x_3 > 8.0 \Rightarrow$ The user is in the bathroom.
- If the y-value of sensor 4 y_4 is in $5.5 \leq y_4 < 10.0$ and the x-value of this triple $x_4 > 8.0 \Rightarrow$ The user is in the bathroom.
- If none of the previous conditions hold $\Rightarrow$ No information.

south represents the sleeping area, *north* denotes the lounge area, *middle* stands for the kitchen area and *bathroom* for the bathroom. If no information could be collected, the information *nothing* is inferred.

$\mathbf{r_{indoor_4}}$: The resulting information will be converted from the device specific format into the global format.

$\mathbf{r_{indoor_5}}$: The Indoor-Positioning-System is a sender.

4.2.2.9 Weather Information System

$\mathbf{r_{weather_1}}$: The device creates weather data internally in order to emulate the receiving of data from the outside - by the internet.

$\mathbf{r_{weather_2}}$: The Weather Information System generates a tuple containing a timestamp mentioning the start time of this weather forecast and one of the following information: sun, rain, storm, snow, wind or hail.

$\mathbf{r_{weather_3}}$: The component converts the internal data into a global format.

$\mathbf{r_{weather_4}}$: The device is a sender.

4.2.2.10 Traffic Information System

$\mathbf{r_{traffic_1}}$: The device creates traffic data internally in order to emulate the receiving of data from the outside - by the internet.

$\mathbf{r_{traffic_2}}$: The Traffic Information System generates a tuple containing one of the following data and a timestamp describing the start time of this traffic condition: jam, accident, free or roadwork.

$\mathbf{r_{traffic_3}}$: The component converts the internal data into a global format.

$\mathbf{r_{traffic_4}}$: The device is a sender.

4.2.2.11 Location-Based Screen

$\mathbf{r_{screen_1}}$: The model of the Location-Based Screen should represent each of the two TV sets which are located in the sleeping area resp. the lounge area of the Living Place apartment.

$\mathbf{r_{screen_2}}$: The Location-Based Screen is able to receive topic-based information, so it is able to subscribe resp. unsubscribe to the topics “UbiTV” and “Alarm”.

$\mathbf{r_{screen_3}}$: It receives both kinds of data, i.e. the device receives data regarding both topics, and splits them according to the specified type. Both types of data are processed differently.

$\mathbf{r_{screen_4}}$: Data belonging to the topic “Alarm” will be evaluated in order to switch both TV sets on at the same time.

$\mathbf{r_{screen_5}}$: Data belonging to the topic “UbiTV” will be evaluated regarding its specified position:

- If the message contains *south*, the TV set in the sleeping area should be activated, whereas the other one should be set to inactive.
- If the message contains *north*, the TV set in the lounge area should be activated, whereas the second one should be set to inactive.
- In all other cases, both TV set should be deactivated.

$\mathbf{r_{screen_6}}$: Each TV set has two states: One describing the global on/off state of both TV sets. They are always identical for both TV sets. The other state stands for the activation state of both TV sets, which is either opposite within both TV sets, i.e. if the TV set in the sleeping area is activated, then the TV set in the lounge area is inactive, and the other way round. An other possibility is, that both TV sets are inactive at the same time.

$\mathbf{r_{screen_7}}$: The Location-Based Screen is a receiver.

$\mathbf{r_{screen_8}}$: The user is able to switch both TV sets of the Location-Based Screen on resp. off. For further details see 4.2.4.2.6.

4.2.2.12 Ambient Light

$\mathbf{r_{light_1}}$: The model of the Ambient Light contains a representation of the light status within the Living Place apartment.

$\mathbf{r_{light_2}}$: It is able to receive data belonging to a certain topic, so the device is able to subscribe resp. unsubscribe to the topics “Light” and “Alarm”.

$\mathbf{r_{light_3}}$: The device processes the received data according to one of the following rules:

- If topic is “Alarm” then switch the light on in dimmed mode.
- If the topic is “Light”, then change the state of the light, as given in the message. These available states for the light are: *bright*, *dimmed*, off.

$\mathbf{r_{light_4}}$: The Ambient Light is a receiver.

$\mathbf{r_{light_5}}$: The user is able to switch the light on resp. off. For further details, see 4.2.4.2.7.

4.2.3 System Level

The sending of information to the Message Oriented Middleware consists of the asynchronous dispatch of that information to the message broker and the reservation of that information for a future storage in the persistent layer of the Message Oriented Middleware. Therefore, for every artifact there exists a place, where the asynchronously dispatched information are cached and additionally a place in every artifact, where reservations of information are stored. Furthermore, every artifact has exactly one connection to the message broker and additionally exactly one connection to the persistence layer. Once an artifact has sent information to the Message Oriented Middleware, this information are stored in the persistence layer of the Message Oriented Middleware and the message broker processes the information for further distribution in parallel.[1] Neither the order, in which the information are stored in the persistence layer, is deterministic, nor the order in which the message broker processes the information.

4.2.3.1 Message Broker

The message broker, as presented in section 2.2.1.3, constitutes the central element of the Living Place system. It controls the whole communication between all components. Each device is connected to the message broker for sending its information to the message broker. In addition, the persistence layer and the Context Interpreter are directly connected to the message broker, which processes each message and distributes it to the specific components, i.e. the devices or to the Context Interpreter. Information collected by the persistence layer or by the Context Interpreter will be forwarded to the message broker for an appropriate distribution.

$\mathbf{r_{mom_{broker_1}}}$: In order to get connected to the Living Place system, each device announces this request. The message broker checks for this request and sets the device into online mode. In this process, it assigns an idividual connection id to the device.

$\mathbf{r_{mom_{broker_2}}}$: If a device requests to get disconnected, the message broker clears the input data buffer of the device.

$\mathbf{r_{mom_{broker_3}}}$: If a device requests to get disconnected, the message broker detects that request and sets the device into offline mode. In doing so, it removes the connection id from the device and from its internal storage. But, a device is able to be set into offline mode, if no topics are subscribed to the message broker resp. no topic is announced to get subscribed. Furthermore, no messages should be within the data input buffer of the device, nor within the output buffer for the message broker or within the output buffer for the persistence layer. Additionally, no request to the persistence layer should be announced within the device.

$\mathbf{r_{mom_{broker_4}}}$: The device announces, that a topic should be subscribed, which will be checked by the message broker. The message broker subscribes the topic in creating an association between the specific connection identification of the device and the given topic. The message broker notfies the device of the successful subscription.

$\mathbf{r_{mom_{broker_5}}}$: The device announces, that a topic should get unsubscribed. This will be checked by the message broker, which removes the subscription in deleting the association between the specific connection identification of the device and the given topic. The message broker notfies the device of the successful unsubscription.

$\mathbf{r_{mom_{broker_5}}}$: If a device crashs, the message broker handles it in removing the device from the Living Place system and the internally stored associations between the specific connection identification with all topics to which the device was subscribed. As a result, no messages will be distributed to this device anymore.

$\mathbf{r_{mom_{broker_7}}}$: The whole communication within the Living Place system takes place over the message broker. As a result, each message produced by a device should be received by the message broker.

$\mathbf{r_{mom_{broker_{8(a)}}}}$: The message broker can receive information from different devices in parallel.

$\mathbf{r_{mom_{broker_{8(b)}}}}$: The message broker can **not** receive information from the same device in parallel.

$\mathbf{r_{mom_{broker_9}}}$: The message broker receives finished data from the persistence layer and from the Context Interpreter in order to distribute them to the devices.

[1] compare section 2.2 in [OV11b]

$\mathbf{r_{mom_{broker_{10}}}}$: The distribution of messages is based on the topic interface. Therefore, the meassage broker needs to store all connected device identifications internally in order to assign messages to devices correctly. Furthermode, the association between each connection identfication and between each topic, to which the specific device is subscribed needs to be memorised by the message broker. If a device is subscribed to more than one topic, then, more than one association have to be stored for this connection identification.

$\mathbf{r_{mom_{broker_{11}}}}$: Messages according to a specific topic will be distributed by the message broker to all devices which are subscribed to this topic. All other devices should not get this message.

$\mathbf{r_{mom_{broker_{12}}}}$: All messages received by the message broker should be forwared to the Context Interpreter.

$\mathbf{r_{mom_{broker_{13}}}}$: Only the Message Oriented Middleware knows the interface of each device. It abstracts from the implementation details of each communicating device.

4.2.3.2 Persistence Layer

The Persistence Layer was introduced in section 2.2.1.4. Its model should fulfil the requirements presented in the following:

$\mathbf{r_{mom_{persistence_{1}}}}$: The persistence layer receives all data sent by the devices.

$\mathbf{r_{mom_{persistence_{2}}}}$: The persistence layer contains an internal data storage where data can be stored.

$\mathbf{r_{mom_{persistence_{3}}}}$: Single data records can be deleted.

$\mathbf{r_{mom_{persistence_{4}}}}$: Devices are able to send requests for past data to the persistence layer. It receives these requests from the devices.

$\mathbf{r_{mom_{persistence_{5}}}}$: A request will be processed in collecting all data records regarding the provided topic.

$\mathbf{r_{mom_{persistence_{6}}}}$: The resulting information containing all past data belonging to the given topic should be distributed to the requesting device with the help of the message broker.

4.2.3.3 Context Interpreter

The Context Interpreter is an additional part of the Message Oriented Middleware which infers new context information out of data sent by the devices. This section enumerates the requirements, which follow out of the description of the Context Interpreter in section 2.2.1.5.

$\mathbf{r_{mom_{ci_{1}}}}$: The Context Interpreter gets all data received and processed by the message broker. Therefore, all data should be given to the component of the Context Interpreter.

$\mathbf{r_{mom_{ci_{2}}}}$: Only actual data should be processed by the Context Interpreter.

$\mathbf{r_{mom_{ci_{3}}}}$: The Context Interpreter contains different rules for inferring new data. The resulting information should describe a 5W1H context, as described in section 2.2.1.5.

$\mathbf{r_{mom_{ci_{4}}}}$: The Context Interpreter infers new information out of single data records, but also out of a combination of different data records.

$\mathbf{r_{mom_{ci_{5}}}}$: The resulting information, describing a new 5W1H context should be sent to the Message Oriented Middleware in order to be provided for all devices subscribed to the corresponding topic "Context'.

4.2.4 User Level

For every possible user operation upon the system of the Living Place there is a model defined in chapter 7. Consider, that these models of the possible user operations upon the system of the Living Place only consider those parts of these user operations, that proceed inside of the system of the Living Place, since this work deals with the modelling of this system, i.e. since every such user operation causes immediately a certain change of the internal state of this system, these models depict these changes of the internal state of this system which are caused by these user operations.

4.2.4.1 User Operations Causing a Change of the Internal State of the Message Oriented Middleware

$\mathbf{r_{mom_{user_1}}}$: The user is able to set a device into offline mode.

$\mathbf{r_{mom_{user_2}}}$: The user is able to add a device which is in offline mode to the Living Place system.

$\mathbf{r_{mom_{user_3}}}$: A device in offline mode can be removed from the Living Place system by the user.

$\mathbf{r_{mom_{user_4}}}$: The user is able to switch the whole Living Place system off, e.g. in case of an emergency.

$\mathbf{r_{mom_{user_5}}}$: The user is able to initialise the whole Living Place system. For that, the following steps need to be executed:

- Delete all queues.
- Delete all currently available requests.
- Remove stored persistent data.

$\mathbf{r_{mom_{user_6}}}$: Those changes of the internal state of the system of the Living Place, which are caused by the execution of the user operations *plugin* device and *unplug* device regarding two different devices, can proceed in parallel.

4.2.4.2 User Operations Causing a Change of the Internal State of a Device

$\mathbf{r_{user}}$: Those changes of internal states of the Living Place System which are caused by the execution of different user operations upon communicating devices should generally proceed in parallel.

4.2.4.2.1 Alarm Clock 2.0

$\mathbf{r_{alarm_{user_1}}}$: The user is able to switch off the Alarm Clock 2.0. The following steps need to be executed when stopping the Alarm Clock 2.0:

- Remove the currently selected appointments within the Alarm Clock 2.0.
- Reset the sleeping phase of the user.
- Reset the current context.
- Delete the current wake up time.
- Reset the internal tasks list.
- Reset the additional waiting time to 0 minutes, which was calculated because of bad weather resp. traffic conditions.
- Reset the internal mode to idle mode.

4.2.4.2.2 Display

$\mathbf{r_{display_{user_1}}}$: The user is able to remove single data from the screen of the Display device.

4.2.4.2.3 Daily Planner

$\mathbf{r_{planner_{user_1}}}$: The user is able to enter new appointments to the Daily Planner.

$\mathbf{r_{planner_{user_2}}}$: The user is able to remove previously entered appointments from the Living Place system.

4.2.4.2.4 Intelligent Bed

$\mathbf{r_{bed_{user_1}}}$: A user operation is provided for entering pressure measurement data.

4.2.4.2.5 Indoor-Positioning-System

$\mathbf{r_{indoor_{user_1}}}$: A user operation is provided for entering position measurement data.

4.2.4.2.6 Location-Based Screen

$\mathbf{r_{screen_{user_1}}}$: The user is able to switch on both TV sets of the Location-Based Screen simultaneously.

$\mathbf{r_{screen_{user_2}}}$: The user is able to switch off both TV sets of the Location-Based Screen simultaneously.

4.2.4.2.7 Ambient Light

$\mathbf{r_{light_{user_1}}}$: The user is able to switch on the light.

$\mathbf{r_{light_{user_2}}}$: The user is able to switch off the light.

4.3 Summary

This chapter presents the requirements towards the model of the Living Place system which therefore are requirements towards the submodels constituting this overall model of the system. So, if such a model fulfils these requirements, it can be regarded as an adequate model of the Living Place system.

The next chapter 5 deals with the requirements towards the used modelling techniques.

Chapter 5

Modelling the System of the Living Place: Requirements Towards the Modelling Techniques

This chapter describes requirements that need to be met by the modelling techniques that are used within the modelling of the Living Place system, so that these techniques are suitable for the modelling of this system. These requirements follow directly from the requirements towards the model of the previous chapter 4, since an adequate model of the Living Place system must meet the therein defined requirements and therefore the used modelling techniques need to be expressive enough to be able to express the therein defined demanded things.

Therefore, the considerations within this chapter give primarily checkpoints concerning to the used modelling techniques that need to be validated, so that an answer to the following question can be given: "Are the used formal modelling techniques expressive enough and therefore suitable for modelling the Living Place system?"

At first three general requirements are defined.

$\mathbf{r_{tech_{gen1}}}$: Since this works aims to provide a model concerning the internal system behaviour of the Living Place system, the possible internal system states need to be represented.

$\mathbf{r_{tech_{gen2}}}$: Since this works aims to provide a model concerning the internal system behaviour of the Living Place system, the actions that cause a change of the internal system states need to be represented.

$\mathbf{r_{tech_{gen3}}}$: According to requirement $\mathbf{r_{gen}}$ of chapter 3, the modelling technique should provide a possibility, to model the whole system on distinguishable levels. The levels presented in this chapter are: data level, object level, system level and user level.

5.1 Data Level

$\mathbf{r_{tech_1}}$: The data and operations that are involved in the processing of the system, especially concerning the communicating processes within the system, need to be represented.

5.2 Object Level

$\mathbf{r_{tech_2}}$: Each device is an own distinguishable component, seperated from all other components, which should be supported by the modelling technique.

$\mathbf{r_{tech_3}}$: The communication within the Living Place system is asynchronous, which will be portrayed by the modelling technique.

5.3 System Level

$\mathbf{r_{tech_4}}$: According to the concept of a Message Oriented Middleware (compare section 2.2.1), the communication between the models of all devices situated on object level should be guaranteed by the Message Oriented Middleware on system level. Following, messages assigned within the models on object level should be passed by the model of the Message Oriented Middleware which is on the next higher level.

$\mathbf{r_{tech_5}}$: The Message Broker should be able to distribute information that it receives from one device to several other devices.

5.4 User Level

$\mathbf{r_{tech_6}}$: The user is able to apply user operations to the Message Oriented Middleware, therefore, the modelling technique should enable to express the change of the model on system level after an instigation on the user level.

$\mathbf{r_{tech_7}}$: The user is able to apply user operations to the models of the devices on object level. Consequently, the modelling technique should provide the possibility to change the behaviour of a model on object level after an initiation on user level.

5.5 Summary

This chapter describes requirements towards the modelling techniques that are used within the modelling of the Living Place system.

In the next chapter 6 these modelling techniques are defined and it is clarified, whether the techniques fulfil the herein defined requirements leading to an answer to the question: “Are the used formal modelling techniques expressive enough and therefore suitable for modelling the Living Place system?”.

Chapter 6

Modelling the System of the Living Place: Modelling Techniques

This chapter presents the formal modelling techniques that are used when modelling the system of the Living Place in chapter 7. Furthermore, it is checked whether the so defined techniques meet the requirements of chapter 5. For a detailed overview of the here presented formalisms see [MGE+10] and [EEPT06].

6.1 Algebraic High-Level Nets with Individual Tokens (AHLI Nets)

6.1.1 Definition

A marked algebraic high-level net with individual tokens, short AHLI net, is defined as

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

, where

- $AN = (\Sigma, P, T, pre, post, cond, type, A)$ is a classical AHL net with
  - signature $\Sigma = (S, OP, X)$ of sorts S, operation symbols OP and variables $X = (X_s)_{s \in S}$,
  - sets of places P and transitions T,
  - $pre, post : T \to (T_{OP}(X) \otimes P)^{\oplus 1}$, defining the transitions' pre- and postdomains,
  - $cond : T \to P_{fin}(Eqns(S, OP, X))$, assigning a finite set of Σ-equations (L, R, X) as firing conditions to each transition,
  - $type : P \to S$ typing the places of the signature's sorts,
  - a Σ-Algebra A[2],
- I is the (possibly infinite) set of individual tokens of ANI, and
- $m : I \to A \otimes P$ is the marking function, assigning the individual tokens to the data elements on the places. $m(I)$ defines the actual set of data elements on the places of ANI. m does not have to be injective.

[MGE+10, Definition 3.1]

6.1.2 Firing Behaviour

6.1.2.1 Consistent Transition Assignments

[1] $T_{OP}(X) \otimes P = \{(t, p) \in T_{OP}(X) \times P | t \in T_{OP,type(p)}(X)\}$

[2] For an introduction to signatures & algebras see [EEPT06] or [EMC+01].

[3] $\overline{asg} : T_{OP}(X) \to A$ is the evaluation of Σ-terms over variables in X to values in A

The set of consistent transition assignments is defined as

$$CT = \{(t, asg) \in T \times (Var(t) \to A) | \forall (L, R, X) \in cond(t) : \overline{asg}^3(L) = \overline{asg}^3(R)\}$$

, where $Var(t) \subseteq X$ is the set of variables occuring in equations and on the environment arcs of t, i.e. all firing conditions of t are valid when evaluated with the variable assignment asg. [MGE$^+$10, Definition 3.1]

6.1.2.2 Token Selection

A token selection regarding to an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

is defined as (M, m, N, n), where

- $M \subseteq I$,
- m is the token mapping of ANI,
- N is a set with $(I \setminus M) \cap N = \emptyset$,
- $n : N \to A \otimes P$

M selects the individual tokens to be consumed and N contains the set of individual tokens to be produced in a firing step as it is described in section 6.1.2.3. [MGE$^+$10, Definition 3.2]

6.1.2.3 Firing of AHLI Nets

A consistent transition asssignment $(t, asg) \in CT$ for an AHLI net

$$ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$$

is *enabled* under a token selection (M, m, N, n), if it meets the following *token selection condition*:

$$\sum_{i \in M} m(i) = pre_A(t, asg) \wedge \sum_{i \in N} n(i) = post_A(t, asg)$$

, where $pre_A, post_A : CT \to CP^{\oplus}$ with $CP = (A \otimes P) = \{(a, p) \in A \times P | a \in A_{type(p)}\}$ is defined as

$$pre_A(t, asg) = (\overline{asg}^3 \otimes id_P)^{\oplus}(pre(t))$$

$$post_A(t, asg) = (\overline{asg}^3 \otimes id_P)^{\oplus}(post(t))$$

.

If such an asg-enabled t fires, the follower marking (I', m') is given by

$$I' = (I \setminus M) \cup N, m' : I' \to A \otimes P \ with \ m'(x) = \begin{cases} m(x), & x \in I \setminus M \\ n(x), & x \in N \end{cases}$$

,

leading to $ANI' = (AN, I', m')$ as the new AHLI net in the *firing step* $ANI \overset{t,asg}{\rightarrowtail} ANI'$ via token selection (M, m, N, n).[MGE$^+$10, Definition 3.2]

6.1.3 AHLI Net Morphisms

Given two AHLI nets $ANI_i = (\Sigma_i, P_i, T_i, pre_i, post_i, cond_i, type_i, A_i, I_i, m_i), i \in \{1, 2\}$, an AHLI net morphism $f : ANI_1 \to ANI_2$ is a pentuple

$$f = (f_\Sigma : \Sigma_1 \to \Sigma_2, f_P : P_1 \to P_2, f_T : T_1 \to T_2, f_A : A_1 \to V_{f_\Sigma}(A_2), f_I : I_1 \to I_2)$$

such that the following diagrams commute (componentwise for pre_i and $post_i$)[4]:

[4] V_{f_Σ} is the forgetful functor induced by signature homomorphism f_Σ, such that $f_A : A_1 \to V_{f_\Sigma}(A_2)$ is a generalized Σ_1-homomorphism. $f_\Sigma^\#$ is the extension of f_Σ to terms and equations.

$$\begin{tikzcd}
\mathcal{P}_{fin}(Eqns(\Sigma_1)) \arrow[d, "\mathcal{P}_{fin}(f_\Sigma^\#)"] \arrow[dr, phantom, "="] & T_1 \arrow[l, "cond_1"'] \arrow[d, "f_T"] \arrow[r, shift left, "pre_1"] \arrow[r, shift right, "post_1"'] \arrow[dr, phantom, "="] & (T_{OP_1}(X_1) \otimes P_1)^\oplus \arrow[d, "(f_\Sigma^\# \otimes f_P)^\oplus"] \\
\mathcal{P}_{fin}(Eqns(\Sigma_2)) & T_2 \arrow[l, "cond_2"'] \arrow[r, shift left, "pre_2"] \arrow[r, shift right, "post_2"'] & (T_{OP_2}(X_2) \otimes P_2)^\oplus
\end{tikzcd}$$

$$\begin{tikzcd}
P_1 \arrow[r, "type_1"] \arrow[d, "f_P"] \arrow[dr, phantom, "="] & S_1 \arrow[d, "f_\Sigma"] & I_1 \arrow[r, "m_1"] \arrow[d, "f_I"] \arrow[dr, phantom, "="] & A_1 \otimes P_1 \arrow[d, "f_A \otimes f_P"] \\
P_2 \arrow[r, "type_2"] & S_2 & I_2 \arrow[r, "m_2"] & A_2 \otimes P_2
\end{tikzcd}$$

Figure 6.1: AHLI Net Morphism

[MGE$^+$10, Definition 3.3]

Consider that all AHLI net morphisms considered within the modelling of this work are morphisms between AHLI nets that share the same signature Σ. Therefore, for each of these morphisms it is defined, that $f_\Sigma = id_\Sigma$.

6.1.4 Category AHLINets and AHLINets(Σ)

The category **AHLINets** consists of all AHLI nets as objects and all AHLI net morphisms.[5] [MGE$^+$10, Definition 3.3]

AHLINets(Σ) is the full subcategory of **AHLINets** containing all AHLI nets with signature Σ. [MGE$^+$10, Compare to Theorem 5.3]

6.2 Transformation of AHLI Nets

6.2.1 Construction of Pushouts in AHLINets

For an introduction to pushouts refer to [EEPT06, Definition 2.16].

Pushouts in **AHLINets** are constructed componentwise in **AHLNets** and **Sets**. So, (1) is a PO in **AHLINets** iff (2) is a PO in **AHLNets** and (3) is a PO in **Sets** with the components of the **AHLINets** morphisms, where $m_3 : I_3 \to A_3 \otimes P_3$ is induced by PO object I_3 in the commutating cube below (whose front's place components let the front commutate because of the PO in the net structure).

$$\begin{tikzcd}
ANI_0 \arrow[r, "f_1"] \arrow[d, "f_2"] \arrow[dr, phantom, "(1)"] & ANI_1 \arrow[d, "g_1"] \\
ANI_2 \arrow[r, "g_2"] & ANI_3
\end{tikzcd}$$

$$\begin{tikzcd}
AN_0 \arrow[r, "f_1'"] \arrow[d, "f_2'"] \arrow[dr, phantom, "(2)"] & AN_1 \arrow[d, "g_1'"] \\
AN_2 \arrow[r, "g_2'"] & AN_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[r, "f_{1I}"] \arrow[d, "f_{2I}"] \arrow[dr, phantom, "(3)"] & I_1 \arrow[d, "g_{1I}"] \\
I_2 \arrow[r, "g_{2I}"] & I_3
\end{tikzcd}$$

$$\begin{tikzcd}
I_0 \arrow[rr, "f_{1I}"] \arrow[dr, "m_0"] \arrow[dd, "f_{2I}"'] & & I_1 \arrow[dr, "m_1"] \arrow[dd, "g_{1I}"] & \\
& A_0 \otimes P_0 \arrow[rr, "f_{1A} \otimes f_{1P}"'] \arrow[dd, "f_{2A} \otimes f_{2P}"] & & A_1 \otimes P_1 \arrow[dd, "g_{1A} \otimes g_{1P}"] \\
I_2 \arrow[rr, "g_{2I}"'] \arrow[dr, "m_2"'] & & I_3 \arrow[dr, dashed, "m_3"'] & \\
& A_2 \otimes P_2 \arrow[rr, "g_{2A} \otimes g_{2P}"'] & & A_3 \otimes P_3
\end{tikzcd}$$

Figure 6.2: Construction of Pushouts in **AHLINets**

[5] For an introduction to category theory see [EEPT06] or [EMC$^+$01]

If f_{1X}, for $X \in \{P, T, I\}$, is injective then g_{2X} is as well and similar for components of f_2 and g_1 and the other diagrams. [MGE+10, Fact 3.4]

6.2.2 AHLI Transformation Rules

An AHLI transformation rule is a span of injective **AHLINets** morphisms

$$\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$$

. [MGE+10, Definition 3.6]

Consider, that the morphsisms l, r of the AHLI transformation rules used within this work are morphisms of the class $\mathcal{M}_I$ with $\mathcal{M}_I = \{f \in Mor_{\mathbf{AHLINets(\Sigma)}} | f_\Sigma = id_\Sigma, f_A\ isomorphic, and\ f_P, f_T, f_I\ injective\}$, so that the nets L, K, R share the same signature Σ and also the same Σ-Algebra A as well as $l_\Sigma = r_\Sigma = id_\Sigma$ and $l_A = r_A = id_A$.

Also consider, that because of the individual token approach of **AHLINets**, the morphisms l, r not need to be strict on the tokens. Taking into account the applicability of such an *AHLI* transformation rule it is clarified, that the definition enables the formulation of marking-changing rules, so that the application of such an *AHLI* transformation rule can change the marking of an *AHLI* net. This features is used heavily within the modelling of chapter 7 leading into an elegant model, whereas this feature is not possible within the former "collective" token approach.

6.2.2.1 Negative Application Conditions

A simple negative application condition is of the form $NAC(x)$, where $x : L \to X$ is a morphism. [EEPT06, Definition 7.8]

So, an AHLI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ with a negative application condition $NAC(x)$ is of the form $X \xleftarrow{x} L \xleftarrow{l} K \xrightarrow{r} R$, where x is an **AHLINets** morphism into AHLI net X.

6.2.3 AHLI Transformation

Given an AHLI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and an AHLI net ANI_1 with an AHLI net morphism $m : L \to ANI_1$, called the match, a direct AHLI net transformation $ANI_1 \overset{\varrho,m}{\Longrightarrow} ANI_2$ from ANI_1 to the AHLI net ANI_2 is given by the following double-pushout diagram (DPO diagram) in the category **AHLINets**:

$$\begin{tikzcd}
\mathrm{L} \arrow[d, "m"'] \arrow[dr, phantom, "(PO_1)"] & \mathrm{K} \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dr, phantom, "(PO_2)"] & \mathrm{R} \arrow[d, "m^*"] \\
ANI_1 & ANI_0 \arrow[l] \arrow[r] & ANI_2
\end{tikzcd}$$

Figure 6.3: Direct AHLI Net Transformation

[MGE+10, Definition 3.7]

6.2.4 Gluing Condition in AHLINets

To be able to decide whether an AHLI transformation rule is applicable at a certain match, a gluing condition for AHLI nets is formulated in the following.

Given AHLI nets K, L, and ANI and AHLI morphisms $l : K \to L$ und $f : L \to ANI$.

- The set of identification points is defined as $IP = IP_P \cup IP_T \cup IP_I$ with
  - $IP_P = \{x \in P_L | \exists x' \neq x : f_P(x) = f_P(x')\}$
  - $IP_T = \{x \in T_L | \exists x' \neq x : f_T(x) = f_T(x')\}$
  - $IP_I = \{x \in I_L | \exists x' \neq x : f_I(x) = f_I(x')\}$
- The set of dangling points is defined as $DP = DP_T \cup DP_I$ with
  - $DP_T = \{p \in P_L | \exists t \neq T_{ANI} \setminus f_T(T_L) : f_P(p) \in ENV_P(t)\}$
  - $DP_I = \{p \in P_L | \exists i \in I_{ANI} \setminus f_I(I_L) : f_P(p) = \pi_P(m_{ANI}(i))\}$

- The set if gluing points is defined as $GP = l_P(P_K) \cup l_T(T_K) \cup l_I(I_K)$

l and f satisfy the gluing condition if $IP \cup DP \subseteq GP$. Given an AHLI rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$, ϱ and f satisfy the gluing condition iff l and f satisfy the gluing condition.

$$\begin{tikzcd}
L \arrow[d, "f"] & K \arrow[l, "l"'] \arrow[r, "r"] & R \\
ANI & &
\end{tikzcd}$$

Figure 6.4: Gluing Condition in **AHLINets**

[MGE$^+$10, Definition 3.8]

6.2.5 Applicability of AHLI Transformation Rules

Given an AHLI rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ and a match $m : L \to ANI_1$ into an AHLI net $ANI_1 = (AN, I, m : I \to P_{AN})$. The rule ϱ is applicable on match m, i.e. there exists a (unique) pushout complement ANI_0 in the diagram below, iff ϱ and m satisfy the gluing condition in **AHLINets**. [MGE$^+$10, Fact 3.12]

$$\begin{tikzcd}
L \arrow[d, "m"] \arrow[dr, phantom, "(PO)"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, dashed, "m'"] & R \\
ANI_1 & ANI_0 \arrow[l, dashed] &
\end{tikzcd}$$

Figure 6.5: Gluing Condition for AHLI transformation

The application of such an applicable AHLI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ to an AHLI net ANI_1 via a match $m : L \to ANI_1$ is constructed as two pushouts (PO_1) and (PO_2) leading to a direct AHLI net transformation $ANI_1 \overset{\varrho,m}{\Longrightarrow} ANI_2$ as shown in 6.2.3.

Note that the injective **AHLINets** morphisms l, r not have to be strict on the tokens. Therefore, by using the herein defined "individual" token approach of **AHLINets** nets instead of the "collective" token approach, transformation rules can be defined, which manipulate the marking of nets as soon as they are applied.

6.2.6 Applicability of AHLI Transformation Rules with Negative Application Conditions

Given an AHLI rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$, a match $f : L \to ANI$ into an AHLI net $ANI = (AN, I, m : I \to P_{AN})$ and a negative application condition $NAC(x)$ with morphism $x : L \to X$ into AHLI net X. The rule ϱ with negative application condition $NAC(x)$ is applicable on match f, if ϱ and f satisfy the gluing condition in **AHLINets** and $f : L \to ANI$ satisfies $NAC(x)$.

A morphism $f : L \to ANI$ satisfies $NAC(x)$ with $x : L \to X$ if there does not exist a morphism $p : X \to ANI$ in $\mathcal{M}'$ with $p \circ x = f$:

$$\begin{tikzcd}
X \arrow[dr, "p"', "/" marking] & & L \arrow[ll, "x"'] \arrow[dl, "f"] \\
& ANI &
\end{tikzcd}$$

$=$

Figure 6.6: Negative Application Condition

[MGE$^+$10, Fact 3.12] and [EEPT06, Definition 7.8]

The application of such an applicable AHLI transformation rule $\varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ with negative application condition $NAC(x)$ to an AHLI net ANI via a match $f : L \to ANI$ is constructed as two pushouts (1) and (2) leading to a direct AHLI net transformation $ANI \overset{\varrho,f}{\Longrightarrow} ANI'$ as shown in 6.2.3.

6.2.6.1 Problem Concerning the Modelling Within this Work

The model of the Living Place system as presented in chapter 7 contains some AHLI transformation rules $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ with negative application conditions of the form $NAC(x)$, where $x : L \to X$ is an **AHLINets** morphism into AHLI net X and AHLI nets L, K, R and X share the same signature $\Sigma = (S, OP, X)$ as well as the same Σ-Algebra $T_\Sigma(X)$[6], i.e. the term algebra with variables X of signature Σ.

These rules are applied to AHLI nets ANI that also include signature Σ however a concrete Σ-Algebra A instead of the term algebra $T_\Sigma(X)$ of AHLI nets L, K, R and X.

Concerning the applicability of such an AHLI transformation rule with a negative application condition on a match $f : L \to ANI$, it must be checked whether an **AHLINets($\boldsymbol{\Sigma}$)** morphism $p : X \to ANI$ in $\mathcal{M}'$ with $p \circ x = f$ exists.

In [MGE$^+$10, Theorem 5.3] **AHLINets($\boldsymbol{\Sigma}$)** $\mathcal{M}'$ morphisms are defined as **AHLINets($\boldsymbol{\Sigma}$)** morphisms where inter alia f_A is isomorphic. There does not exist any isomorphism between the used Σ-Algebras $T_\Sigma(X)$ of net X and A of net ANI, since not every element of the carrier sets of algebra A has an associated term as an element of the carrier sets of algebra $T_\Sigma(X)$. Therefore, there never exists such an **AHLINets($\boldsymbol{\Sigma}$)** $\mathcal{M}'$ morphism $p : X \to ANI$, so that the match morphism $f : L \to ANI$ always satisfies $NAC(x)$ with $x : L \to X$. Thus, the definition of the negative application condition seems useless without any effect to the applicability of the rule.

Therefor, a new concept of satisfiability regarding to a negative application condition need to be developed. However, in the following considerations within this work negative application conditions are used unchanged in combination with AHLI transformation rules in a manner as it is defined within this chapter.

6.3 Algebraic Higher-Order Nets with Individual Tokens (AHOI Nets) as a Special Type of AHLI Nets

An algebraic higher-order net with individual tokens, short AHOI net, is an AHLI net that features nets and transformation rules as tokens.[7] Within this work the term $AHOI_{AHLI}$ net is used for an AHLI net that features AHLI nets, AHLI transformation rules and AHLI interaction schemata as tokens.

6.4 Transformation of AHOI Nets

The transformation of AHOI nets is analogously defined to the transformation of AHLI nets presented in section 6.2 where AHOI nets are used instead of AHLI nets, since an AHOI net is a special type of an AHLI net as definied in the previous section 6.3.

6.5 Reconfigurable AHLI/AHOI Systems

A reconfigurable AHLI system is a tuple $(\mathcal{N}, \mathcal{R})$, where an AHLI net $\mathcal{N}$ is the start configuration of the system and $\mathcal{R}$ is the set of AHLI transformation rules.

Since an AHOI net is a special type of an AHLI net, a reconfigurable AHOI system is defined analogously as a tuple $(\mathcal{N}, \mathcal{R})$, where an AHOI net $\mathcal{N}$ is the start configuration of the system and $\mathcal{R}$ is the set of AHOI transformation rules. Also see [EBE$^+$09] for more details on the definition of reconfigurable AHLI systems.

Additionally, $(\mathcal{N}, \mathcal{R}, \mathcal{M})$ defines a reconfigurable AHLI resp. AHOI system with a set of AHLI net morphisms $\mathcal{M}$, so that the transformation rules of $\mathcal{R}$ can only be applied by using match morphisms containd in $\mathcal{M}$.

[6] For the definition of the term algebra $T_\Sigma(X)$ with variables X of a given signature $\Sigma = (S, OP, X)$ see [EMC$^+$01]

[7] Compare to [HM03] and [HEM05].

6.6 Amalgamated Rules, Transformations & Interaction Schemes

Referring to [GBEE10] and [Gol10] the concept of amalgamated rules & transformations is introduced within this section.

6.6.1 Kernel Morphism

Given rules $p_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ for $i = 0, ..., n$, a kernel morphism $s_i : p_0 \to p_i$ consists of morphisms $s_{i,L} : L_0 \to L_i, s_{i,K} : K_0 \to K_i, s_{i,R} : R_0 \to R_i$ such that in the diagram (1_i) and (2_i) are pullbacks[8] and (1_i) has a pushout complement for $s_{i,L} \circ l_0$, i.e. $s_{i,L}$ satisfies the gluing condition w.r.t. l_0. The pullbacks (1_i) and (2_i) mean that K_0 is the intersection of K_i with L_0 and also of K_i with R_0.

$$\begin{tikzcd}
L_0 \arrow[d, "s_{i,L}"] & K_0 \arrow[l, "l_0"'] \arrow[r, "r_0"] \arrow[d, "s_{i,K}"] \arrow[dl, phantom, "(1_i)"] \arrow[dr, phantom, "(2_i)"] & R_0 \arrow[d, "s_{i,R}"] \\
L_i & K_i \arrow[l] \arrow[r] & R_i
\end{tikzcd}$$

Figure 6.7: Kernel Morphism

[GBEE10, Definition 1]

6.6.2 Amalgamated Rule

Given rules $p_i = (L_i \xleftarrow{l_i} K_i \xrightarrow{r_i} R_i)$ for $i = 0, ..., n$ with kernel morphisms $s_i : p_0 \to p_i (i = 1, ..., n)$, then the amalgamated rule $\tilde{p} = (\tilde{L} \longleftarrow \tilde{K} \longrightarrow \tilde{R})$ of $p_1, ..., p_n$ via p_0 is constructed as the componentwise gluing of $p_1, ..., p_n$ along p_0. $\tilde{L}$ is the gluing of $L_1, ..., L_n$ with shared L_0 leading to $t_{i,L} : L_i \to \tilde{L}$. Similar gluing constructions lead to $\tilde{K}$ and $\tilde{R}$ and we obtain kernel morphisms $t_i : p_i \to \tilde{p}$ and $t_i \circ s_i = t_0$ for $i = 1, ..., n$. We call p_0 kernel rule, and $p_1, ..., p_n$ multi rules. [GBEE10, Definition 2]

6.6.3 Amalgamated Transformation

An amalgamated transformation $ANI_1 \overset{\tilde{p}}{\Longrightarrow} ANI_2$ is a transformation via the amalgamated rule $\tilde{p}$. [GBEE10, Definition 2]

6.6.4 Interaction Scheme

An interaction scheme $is = \{s_1, ..., s_k\}$ is a set of kernel morphisms $s_i : p_0 \to p_j$ with $j = (1, ..., k)$, where p_0 is the so called kernel rule and $p_1, ..., p_k$ are the multi rules. (Compare to [GBEE10])

Consider that the model as it is presented within this work only includes interaction schemes of the form $is = \{s_1\}$, i.e. interaction schemes that contain exactly one kernel rule as well as one multi rule.

6.6.5 Amalgamated Rules Over Maximal Weakly Disjoint Matchings

Given an interaction scheme, it is desirable to apply as many multirules p_j as often as possible over a certain match of the kernel rule p_0. Therefor, the concept of maximal weakly disjoint matchings is introduced, where multi-amalgamable matchings of the multi rules are required, that are disjoint up to the match of the kernel rule as well as maximal in the sense that no more valid matches for any multi rule in the interaction scheme can be found. This leads to a bundle of kernel morphisms $s'_i : p_0 \to p'_i$ $(i = 1, ..., n)$ with $\tilde{p}'$ being the amalgamated rule of $p'_1, ..., p'_n$ via p_0 over maximal weakly disjoint matchings. (Compare to [GBEE10])

Consider that this construction already delivers an amalgamated match $\tilde{m}$ on which $\tilde{p}'$ is applicable as well as the amalgamated transformation $ANI_1 \overset{\tilde{p}', \tilde{m}}{\Longrightarrow} ANI_2$. However, the model of chapter 7 uses the here presented concept by first generating such an amalgamated rule over maximal weakly disjoint matchings by an operation that is defined within a transition conditions of a certain transition, then finding a suitable match for this amalgamated rule, testing whether the rule is applicable on this match and finally applying that amalgamated rule via the found match.

[8] For an introduction to pullbacks refer to [EEPT06].

6.7 The Functor $\mathcal{V}_{PTNet}$: AHLINets → PTNet

First in section 6.7.1 the category **PTNet** is introduced whereupon in section 6.7.2 the functor $\mathcal{V}_{PTNet}$: **AHLINets** → **PTNet** is defined.

6.7.1 Category PTNet

The category **PTNet** consists of all P/T nets as objects and all P/T net morphisms. [PPE+05, Lemma 8.14]

6.7.1.1 P/T nets

A P/T net is defined as $N = (P, T, pre, post)$ where

- P is the set of places,
- T is the set if transitions,
- $pre, post : T \to P^{\oplus}$, defining the transitions' pre- and postdomains, where $P^{\oplus}$ is the free commutative monoid over P.

[PPE+05, Definition 8.11]

6.7.1.2 P/T net morphisms

Given two P/T nets $N_i = (P_i, T_i, pre_i, post_i)$, $i \in \{1, 2\}$, a P/T net morphism $f : N_1 \to N_2$ is a tuple

$$f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$$

such that the following diagram commute componentwise:

$$\begin{tikzcd}
T_1 \arrow[rr, bend left, "pre_1"] \arrow[rr, bend right, "post_1"'] \arrow[d, "f_T"] & & P_1^{\oplus} \arrow[d, "f_P^{\oplus}"] \\
T_2 \arrow[rr, bend left, "pre_2"] \arrow[rr, bend right, "post_2"'] & & P_2^{\oplus}
\end{tikzcd}$$

Figure 6.8: P/T Net Morphism

[PPE+05, Definition 8.12]

6.7.1.2.1 P/T net isomorphism A P/T net morphism $f : N_1 \to N_2$ with $f = (f_P : P_1 \to P_2, f_T : T_1 \to T_2)$ is a P/T net isomorphism, iff f_P, f_T are bijective, i.e. f is componentwise bijective. [PPE+05, Lemma 8.17]

Given two P/T nets $N_i = (P_i, T_i, pre_i, post_i)$, $i \in \{1, 2\}$, N_1 and N_2 are isomorphic, written $N_1 \cong N_2$, if there exists a P/T net isomorphism $i : N_1 \to N_2$.

6.7.2 $\mathcal{V}_{PTNet}$: AHLINets → PTNet

The following construction converts an AHLI net $ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$ into a P/T net $N = (P_N, T_N, pre_N, post_N)$ by forgetting all algebra specific notations and individual tokens whereas the net structure is preserved:

$\mathcal{V}_{PTNet}((\Sigma, P, T, pre, post, cond, type, A, I, m)) := (P, T, pre_N, post_N : T \to P^{\oplus})$ where

$$pre_N(t \in T) = \sum_{p \in PRE_P(t)} p$$

and

$$post_N(t \in T) = \sum_{p \in POST_P(t)} p$$

and $\mathcal{V}_{PTNet}((f_\Sigma, f_P, f_T, f_A, f_I)) := (f_P, f_T)$ with

[9] Compare to [MGE+10, Definition 3.1]

- $PRE(t) = \{(term, p) \in (T_{OP}(X) \otimes P) | pre(t)(term, p) \neq 0\}^9$,
- $POST(t) = \{(term, p) \in (T_{OP}(X) \otimes P) | post(t)(term, p) \neq 0\}^9$,
- $PRE_P(t) = \pi_P(PRE(t)) \subseteq P$ the place predomain of t^9,
- $POST_P(t) = \pi_P(POST(t)) \subseteq P$ the place postdomain of t^9,

It can be easily seen that functor $\mathcal{V}_{PTNet}$ preserves isomorphisms, i.e. if $f \in Mor_{\mathbf{AHLINets}}$ is an AHLI net isomorphism $\mathcal{V}_{PTNet}(f) \in Mor_{\mathbf{PTNet}}$ is also a P/T net isomorphism.[10]

6.8 Revisiting the Requirements Towards the Modelling Techniques

This section revisits the requirements of chapter 5 by checking whether the formal modelling techniques as defined above fulfill these requirements.

$\mathbf{r_{tech_{gen1}}}$: An internal system state of the Living Place system can be represented by the current structure and marking of an $AHOI_{AHLI}$ net. This includes also the current structure and marking of the $AHLI$ nets that are contained as tokens within this $AHOI_{AHLI}$ net. By changing the structure or marking, another internal state of the system is obtained. So, requirement $r_{tech_{gen1}}$ is fulfilled.

$\mathbf{r_{tech_{gen2}}}$: The actions that cause a change of the internal system state can be represented by transitions of corresponding nets or transformation rules. The firing of a transition or the application of a transformation rule represents the performance of such an action. Consider, that by defining a transition or a transformation rule as a representation for an action of the system behaviour, the pre- and postconditions of this action also are formulated. The pre- and post domain of the transition define these conditions and the left and right hand side of each transformation rule define these conditions in an analogous manner. The application of a transformation rule with individual tokens to a net with individual tokens can change the marking and the structure of this net, therefore leading into a new internal system state. Therefore, by using such transformation rules and nets as tokens within a superordinated net, the firing of a corresponding transition of this superordinated net, which applies such a rule of the transition's predomain to such a net of the transition's predomain, can change the marking and structure of that net token. Consider, that changing the marking of a net by using transformation rules is only possible by using the new approach of individual tokens. The usage of marking-changing rules is heavily used within the model of the Living Place system of chapter 7 leading into an elegant model. Therefore, requirement $r_{tech_{gen2}}$ is fulfilled by the modelling technique.

$\mathbf{r_{tech_{gen3}}}$: Modelling the whole system within distinguishable levels is enabled by using the defined modelling techniques. The Data Level is represented by a signature & algebra contained in $AHLI$ nets, the Object Level is represented by these $AHLI$ nets that are contained within an $AHOI_{AHLI}$ net of the System Level as tokens. Furthermore, $AHOI$ transformation rules upon this $AHOI_{AHLI}$ net as part of a reconfigurable $AHOI$ system constitute the User Level.

6.8.1 Data Level

$\mathbf{r_{tech_1}}$: The data and operations that are involved in the processing of the system, especially concerning the communicating processes within the system, are represented by corresponding signatures & algebras for the $AHOI_{AHLI}$ nets and rules and for the $AHIO$ nets and rules.

6.8.2 Object Level

$\mathbf{r_{tech_2}}$: The requirement, that each device is an own distinguishable component, seperated from all other components, can be fulfilled by modelling each device as a seperate $AHLI$ net which is assigned as an individual token to places within an $AHOI_{AHLI}$ net. Each individual token is distinguishable from all other individual tokens and does not change the structure of all other nets assigned to the same place within the same $AHOI_{AHLI}$ net.

[10] Compare to [EMC$^+$01, Definition 26.2.1].

$\mathbf{r_{tech_3}}$: The communication within the Living Place system can be modelled by firing corresponding transitions. Since, all parallel enabled transitions can fire in parallel or in any abitrary order the possibility to model asynchronous behaviour is given by the nature of the firing behaviour of the used $AHLI$ nets.

6.8.3 System Level

$\mathbf{r_{tech_4}}$: By defining $AHLI$ transformation rules as tokens assigned to places of an superordinated $AHOI_{AHLI}$ net and communicating devices as $AHLI$ net tokens assigned to places of the same superordinated net in a way, so that the $AHLI$ nets can be transformed by applying token rules in a firing step of the $AHOI_{AHLI}$ net, the communication between several devices can be expressed. Therefor, the used concept of individual token is suitable, since the communication can be modelled by marking-changing transformation rules as tokens, that will be applied to $AHLI$ net tokens within a firing step.

$\mathbf{r_{tech_5}}$: The simultaneous distribution of information to several devices can be expressed by amalgamated rules.

6.8.4 User Level

$\mathbf{r_{tech_6}}$: The change of the model on system level after the user applies a user operation upon the system can be expressed by $AHOI$ transformation rules as part of a reconfigurable $AHOI$ system.

$\mathbf{r_{tech_7}}$: Analogously to requirement $\mathbf{r_{tech_6}}$ this can be expressed by $AHOI$ transformation rules as part of a reconfigurable $AHOI$ system. Furthermore, $AHLI$ transformation rules are available as tokens for the transformation of $AHLI$ object nets within a firing step of a superordinated $AHOI_{AHLI}$ net.

6.9 Summary

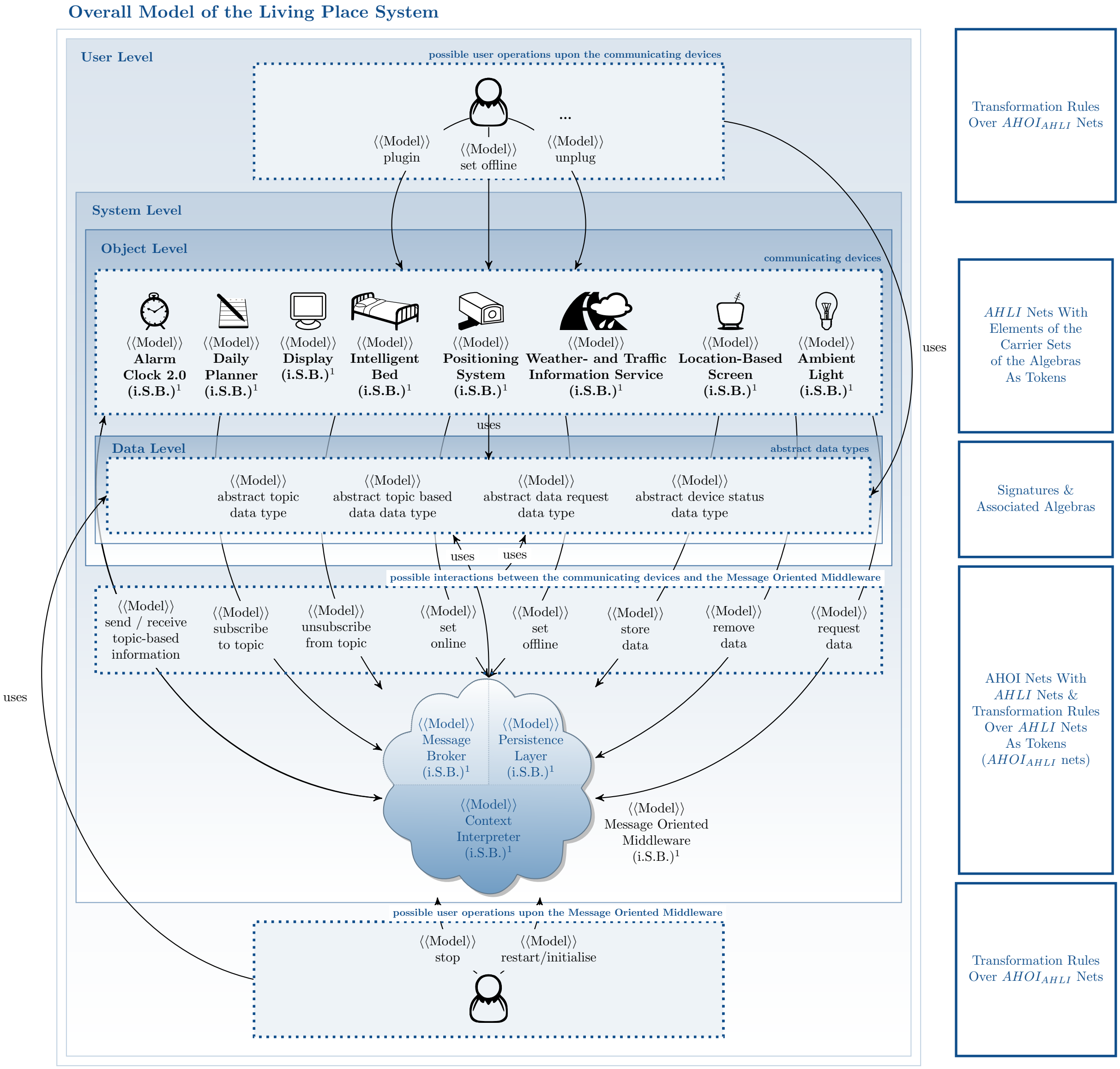
The formal modelling techniques regarding to the modelling of the Living Place system of chapter 7 are defined and it is checked whether the so defined techniques meet the requirements of chapter 5 leading into the result that the techniques fulfill the requirements leading to the answer *Yes* concerning the question: “Are the used formal modelling techniques expressive enough and therefore suitable for modelling the Living Place system?”.

Futhermore it is clarified, that the theory of negative application conditions in combination with AHLI transformation rules as it is used within the modelling of this work need to be reviewed in future work, i.e. a new concept of satisfiability regarding to a negative application condition need to be developed.

The next chapter 7 presents the overall model of the internal system behaviour of the Living Place system.

Chapter 7

Modelling the System of the Living Place: Model of the System of the Living Place



Overall Model of the Living Place System
User Level
possible user operations upon the communicating devices
⟨⟨Model⟩⟩ plugin
⟨⟨Model⟩⟩ set offline
⟨⟨Model⟩⟩ unplug
...
System Level
Object Level
communicating devices
⟨⟨Model⟩⟩ Alarm Clock 2.0 (i.S.B.)1
⟨⟨Model⟩⟩ Daily Planner (i.S.B.)1
⟨⟨Model⟩⟩ Display (i.S.B.)1
⟨⟨Model⟩⟩ Intelligent Bed (i.S.B.)1
⟨⟨Model⟩⟩ Positioning System (i.S.B.)1
⟨⟨Model⟩⟩ Weather- and Traffic Information Service (i.S.B.)1
⟨⟨Model⟩⟩ Location-Based Screen (i.S.B.)1
⟨⟨Model⟩⟩ Ambient Light (i.S.B.)1
uses
Data Level
abstract data types
⟨⟨Model⟩⟩ abstract topic data type
⟨⟨Model⟩⟩ abstract topic based data data type
⟨⟨Model⟩⟩ abstract data request data type
⟨⟨Model⟩⟩ abstract device status data type
uses
uses
possible interactions between the communicating devices and the Message Oriented Middleware
⟨⟨Model⟩⟩ send / receive topic-based information
⟨⟨Model⟩⟩ subscribe to topic
⟨⟨Model⟩⟩ unsubscribe from topic
⟨⟨Model⟩⟩ set online
⟨⟨Model⟩⟩ set offline
⟨⟨Model⟩⟩ store data
⟨⟨Model⟩⟩ remove data
⟨⟨Model⟩⟩ request data
⟨⟨Model⟩⟩ Message Broker (i.S.B.)1
⟨⟨Model⟩⟩ Persistence Layer (i.S.B.)1
⟨⟨Model⟩⟩ Context Interpreter (i.S.B.)1
⟨⟨Model⟩⟩ Message Oriented Middleware (i.S.B.)1
uses
uses
possible user operations upon the Message Oriented Middleware
⟨⟨Model⟩⟩ stop
⟨⟨Model⟩⟩ restart/initialise
Transformation Rules Over AHOI_AHLI Nets
AHLI Nets With Elements of the Carrier Sets of the Algebras As Tokens
Signatures & Associated Algebras
AHOI Nets With AHLI Nets & Transformation Rules Over AHLI Nets As Tokens (AHOI_AHLI nets)
Transformation Rules Over AHOI_AHLI Nets

Figure 7.1: Submodels of the Overall Model of the Living Place System - an Overview

[1] i.S.B. = internal System Behaviour

In this chapter the modelling of the Living Place system will be presented. Because of the usage of the modelling techniques as described in 6, in comparison to figure 7.1 the modelling is able to take place in the following levels, where these levels represent the levels of modelling of chapter 3. Therefore, the resulting model of the Living Place system reflects these levels of modelling in an intuitive way so that requirement $\mathbf{r_{gen}}$ of chapter 4 is fulfilled by the model.

1. **User Level**: The User Level is the topmost level. It provides $AHOI_{AHLI}$ transformation rules as part of an reconfigurable $AHOI$ system which allow different opportunities for the user to interact with the system. In section 7.1.4 rules which can be applied by the user to interfere in the system will be given. According to the concept of ubiquitous computing and ambient intelligence (see 2.1), the whole Living Place system should act independently and automatically, but the user will always keep the control over the whole system and should be able to intervene at any time.

2. **System Level**: On this level the Message Oriented Middleware will be modeled with the help of $AHOI_{AHLI}$ nets. The AHOI nets contain AHLI-nets, representing the next level - the Object Level, and AHLI transformation rules, which will influence the Object Level, as tokens. The Message Oriented Middleware controls the behaviour of the whole system and regulates the communications of all connected components (for further details see section 2.2.1). The Message Oriented Middleware consists of a Message Broker, a Context Interpreter and a Persistence Layer. Additionally the System Level contains a User Operation Processing Unit.

3. **Object Level**: The Object Level consists of AHLI nets with elements of the carrier sets of a given algebra as tokens. Within this level all connected devices and their specific functionality will be modelled. Each component in the Object Level is able to change pieces of information from the Data Level.
   In the modelling which is presented in this chapter, the Object Level and System Level are able to exchange data with the help of graph transformation rules, because the devices within the Living Place system exchange information indirectly over the Message Oriented Middleware. This component needs to be modelled on a higher level than the devices, because it should be able to control the communication and the connection of all devices.[2]

4. **Data Level**: The lowest level is the Data Level which contains the given signature Σ_{OL} and the Σ_{OL}-algebra A_{OL}. The algebra and its associated signature will be described in section 11.1 resp. 11.2 in the appendix. The Data Level contains primitive data of the devices, like pieces of information which will be exchanged between different devices or internal states.

The initial system $AHOI_{AHLI}$ net of figure 7.26 together with the $AHOI$ transformation rules as described in the following of this chapter form the topmost construct, the reconfigurable $AHOI$ system. Consider, that this work aims to provide a model of the internal system behaviour of the Living Place system. So, this initial net represents the initial state of the system. Every application of an $AHOI$ transformation rule leads into another possible state of the system. In analogous manner, every firing step within the topmost $AHOI_{AHLI}$ net leads into another state of the system and therefore every firing step within the object nets of the $AHOI_{AHLI}$ net also leads into another state of the system. So the following presented model illustrates all possible system states of the Living Place system as well as all possible actions that can lead into new states, where actions are defined as transformation rules and transitions. In this way, by defining transition conditions and choosing the right net structure for every such action it is specified which pre- und post-conditions concerning the application of such an action exist, i.e. for every such action it is defined, which internal system state must prevail, so that the action can be applied, and which internal system state results from the application of the corresponding action.

In the following the visual modelling of the Data Level, Object Level, System Level and User Level will be introduced.

[2] See section 2.2.1.

7.1 Model in a Visual Description

7.1.1 Data Level

The data within the Data Level consists of primitive data elements and operations which are visualized by tokens assigned to the places within the object nets. The formal description of the Data Level is depicted by carrier sets within the algebra of the Object Level in section 11.1.

7.1.2 Object Level

This section presents templates of each kind of device in offline resp. online mode which will be connected to the Message Oriented Middleware. These templates will be used by all devices which will be modelled for the Living Place project. The model of the Living Place system distinguishes between the following three device types:

- Transceiver
- Receiver
- Sender

A more detailed introduction to those device types is given in 2.2.2. In the following a template for each kind of device will be presented.

Most of the devices provide the possibility of direct user access, like the Alarm Clock 2.0 (see 7.1.2.7), the Display and Daily Planner in the Multitouch Kitchen Counter (see 7.1.2.8 and 7.1.2.9), the Ambient Light (see 7.1.2.13), the Location-Based Screen (see 7.1.2.12) or the Intelligent Bed and the Indoor-Positioning-System (see 7.1.2.10 and 7.1.2.11). The modelling of those devices and their interfaces is done on all three levels:

- **Object Level**: Like each other device, the device and its functionality is modelled as an object net.
- **User Level**: If the user should be able to directly interact with this device, e.g. by inputing his appointments or points of time to remind into the Daily Planner device, the device provide interfaces to the outside, to enable direct user input. These interfaces will be modelled as an AHOI rules.
- **System Level**: The Object Level and User Level are not able to exchange information directly with each other, e.g. the modelling of the Daily Planner device, which is modelled as net on the Object Level, is not able to directly receive new dates which will be provided with the help of AHOI rules on the User Level. Instead, the System Level will mediate between them in applying a corresponding rule which puts the relevant information directly to a place within the object net.

In the modelling of the devices which provide an interface for direct interaction with the user, it will be mentioned explicitly that they supply additional AHOI and AHLI rules. The AHOI rules will be presented in section 7.1.4 and their corresponding AHLI rules in section 7.1.3.

7.1.2.1 Template of a Receiver in Offline Mode

A receiving device is not able to send any pieces of information to the message broker, but it is able to send requests to the persistence layer in the Message Oriented Middleware to order old data.

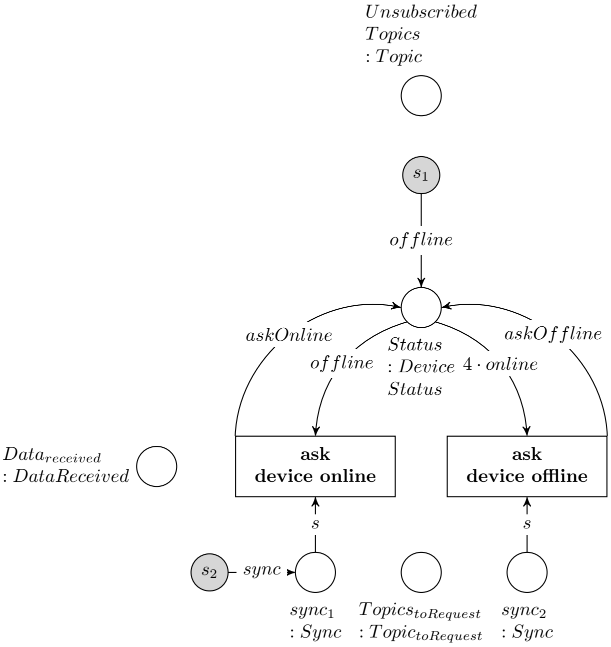


Figure 7.2: Net $device^{offline}_{receiver}$

7.1.2.2 Template of a Sender in Offline Mode

A sender is a device, e.g. a pressure sensor in the Intelligent Bed (see section 2.2.2.4), which only sends data and holds no possibility of receiving any data. Even requesting past data over the persistence layer will not be feasible for sensor devices.

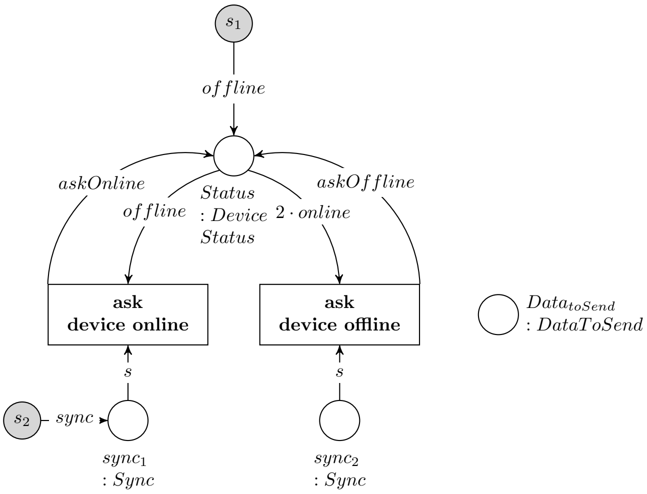


Figure 7.3: Net $device^{offline}_{sender}$

7.1.2.3 Template of a Transceiver in Offline Mode

A transceiver is able to receive and send messages. In contrast to a receiver which is only able to receive data, e.g. devices executing commands, like switch on the light, and a sender which only sends data, e.g. a sensor device. The difference between these three kinds of components lies in the presence of the places $Data_{received}$: *DataReceived* or $Data_{toSend}$: *DataToSend* and *UnsubscribedTopics* : *Topic*.
In offline mode, a device is not able to send or receive any messages and therefore is not able to process

any task, even if the device is present in the Living Place system.
The marking of the template net is: One token *offline* is assigned to the place *Status* : *DeviceStatus* representing the status of the device which is in offline mode. Another token is assigned to the place : *Sync* which is connected to the transition *ask device online*. In using this token, the status of the device can be set to *askOnline* which signals to the message broker to set this device to online mode.
The transition *ask device offline* is the inverse to the transition *ask device online*, which will, if activated, set a token *askOffline* to the place *Status* : *DeviceStatus*.
If an individual device, which uses this template net, is interested in receiving any topics, tokens representing those topics will be assigned to the place *UnsubscribedTopics* : *Topic*. The places '$Data_{received}$: *DataReceived* and $Data_{toSend}$: *DataToSend* represent the data input and data output buffer of the device.
The whole Living Place system holds the opportunity of requesting old data over a persistence layer. Past data are saved into a database which can be queried by connected devices. The persistence layer in the Message Oriented Middleware answers these requests in sending old data belonging to the queried topic back to the device. The requesting devices should be subscribed to this topic to be able to fetch these pieces of information. The place $Topics_{toRequest}$: $Topic_{toRequest}$ contains tokens representing topics which will be necessary for receiving the requested old data.

The initial marking of a device in offline mode - the marking is identical for all devices in offline mode, no matter if it is a transceiver, receiver or sender - is illustrated in the images 7.4, 7.2 and 7.3. It has a token $offline$ assigned to the place *Status* and a second token *sync* assigned to the place *Sync* which is connected to the transition *ask device online* enabling the device to initiate the switch into online mode.

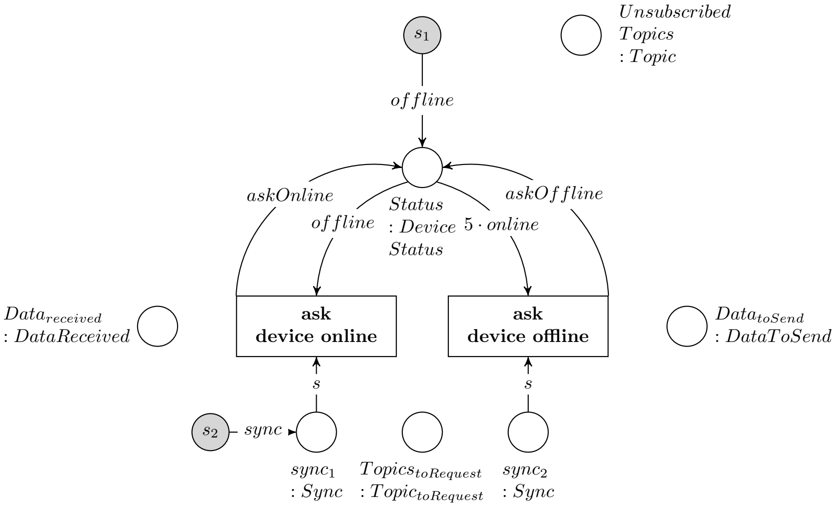


Figure 7.4: Net $device^{offline}_{transceiver}$

7.1.2.4 Template of a Receiver in Online Mode

Even in online mode, the template of a receiver does not contain any possibility to send data to the Message Oriented Middleware, except from sending requests to the persistence layer of the Message Oriented Middleware. Apart from that the template of the receiver in online mode is analog to the template of the transceiver in online mode.
Three tokens *online* are assigned to the place *Status* : *DeviceStatus* because of the transitions *receive data*, *request data* and *ask device online* resp. *ask device offline* should be able to fire parallel. The tokens *sync* on the places : *Sync* are responsible for synchronizing the firing behaviour of each transition *receive data*, *request data* and *subscribe topic*.

7.1.2.4.1 Revisiting the Requirements Towards the Model of the Template of a Receiver

$\mathbf{r_{receiver_1}}$: The object net $device^{offline}_{receiver}$ resp. $device^{online}_{receiver}$ contains the transition *ask device online* which is initially able to fire and replaces the token $offline$ assigned to *Status* : *DeviceStatus* by *askOnline*. That means, the device internally announces, that it wants to be set into online mode. This token will be checked by the message broker which transforms the object net from $device^{offline}_{receiver}$, i.e. the object net of a receiver, into the net $device^{online}_{receiver}$, i.e. the object net in online mode and at the same time provides an individual connection identification to this object net. After changing into online mode, the three tokens *online* will be assigned to $Status$, representing the internal mode of the device: online. The place $Status$ always represents the internal status of the device:

- $offline$ means, the device is in offline mode.
- *askOnline* means, the device is in offline mode, but announces, that it wants to switch into online mode.
- *online* means, the device is in online mode and able to process information.
- $askOffline$ means, the device is in online mode, but announces, that it wants to switch into offline mode.

The device is able to initiate the switch into offline mode in firing the transition *ask device offline* which is the reverse to *ask device online.* Consequently the requirement $r_{receiver_1}$ is fulfilled.

$\mathbf{r_{receiver_2}}$: The individual connection identification will be set by the message broker, when the device is set into online mode. The internal connection id is assigned as token to the place $connection_{id}$: $ConnectionID$. (For the process of setting a device into online mode, see 7.29.) Therefore, the requirement $r_{receiver_2}$ is fulfilled.

$\mathbf{r_{receiver_3}}$: In online mode, the receiver contains the transition *subscribe topic* (resp. *unsubscribe topic*) which will be fired for announcing that a topic should be subscribed (resp. unsubscribed). As a result, the requirement $r_{receiver_3}$ is fulfilled.

$\mathbf{r_{receiver_4}}$: The message broker put new data for the device to the place $Data_I$: *InputData*. The device collects this data in firing the transition *receive data.* Therefore, the reqirement $r_{receiver_4}$ is met.

$\mathbf{r_{receiver_5}}$: The receiver contains the place $Topics_{toRequest}$: $Topic_{toRequest}$ to which tokens can be assigned which represent topics to which requests for past data to the persistence layer should be queried. For requesting stored data from the persistence layer, the transition *request data* will fire, but only if the device is already subscribed to the corresponding topic of the request to the Living Place system. Then, the request will be put to the place $Data_R$: *RequestData* which will be collected by the Message Oriented Middleware and particularly processed by the persistence layer. Following, the requirement $r_{receiver_5}$ is fulfilled.

$\mathbf{r_{receiver_6}}$: In online mode, four tokens *online* are assigned to the place *Status* leading to the possibility of firing the transitions *receive data*, *request data*, *subscribe topic* resp. *ask device online* in parallel.

To summarise, all requirements demanded to the object net of a receiver are fulfilled.

7.1.2.4.2 Visual Model of the Template of a Receiver

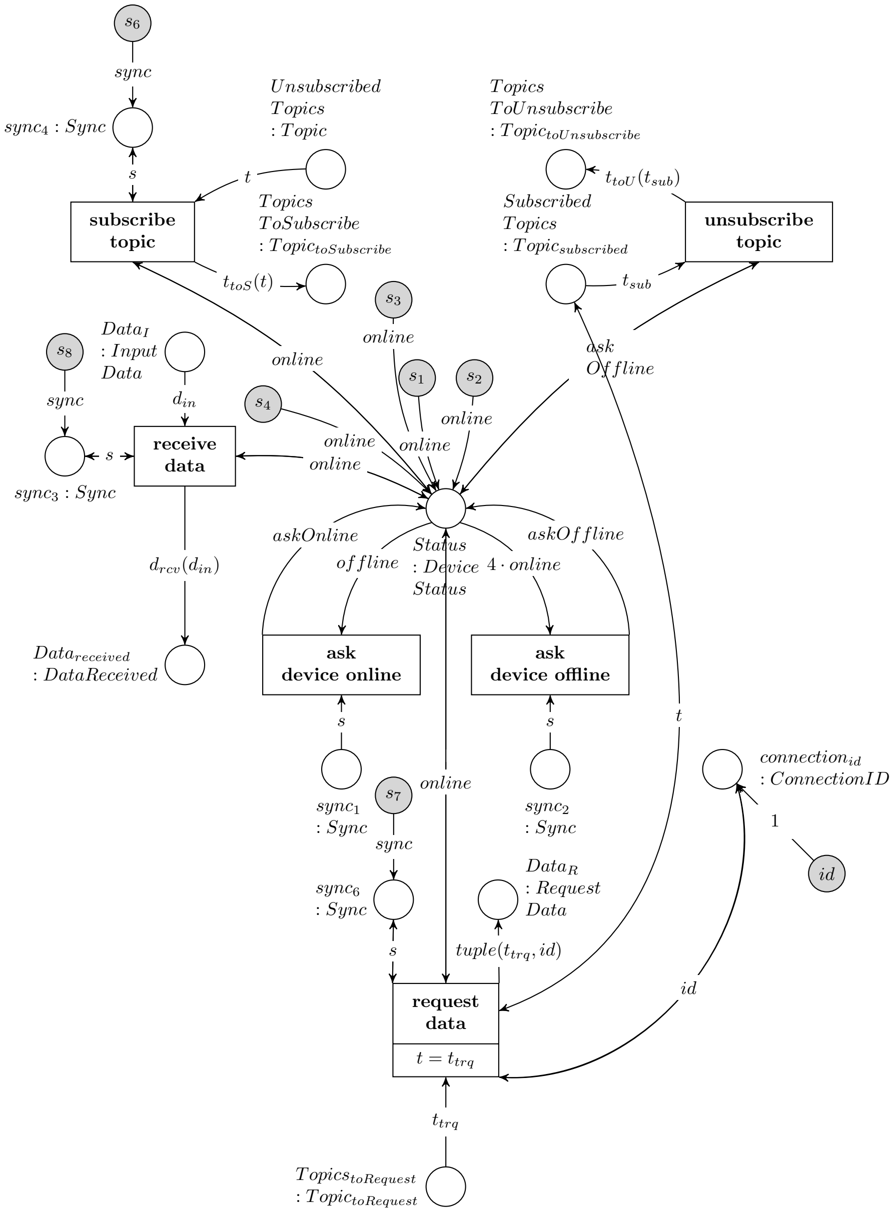


Figure 7.5: Net $device^{online}_{receiver}$

7.1.2.5 Template of a Sender in Online Mode

In contrast to a receiver, the sender is not able to receive any data and additionally is not able to pose any requests to the persistence layer of the Message Oriented Middleware. Similar to the receiver, the template of the sender in online mode is analog to the template of the transceiver in online mode.
Two tokens *online* are assigned to the place *Status* : *deviceStatus* because of the transitions *send data* and *ask device online* resp. *ask device offline* should be able to fire parallel. Furthermore two tokens *sync* are each assigned to a place : *Sync* for synchronizing the firing of *send data*.

7.1.2.5.1 Revisiting the Requirements Towards the Model of the Template of a Sender

$\mathbf{r_{sender_1}}$: Analog to a receiver, the object net $device^{offline}_{sender}$ resp. $device^{online}_{sender}$ contains the transitions *ask device online* resp. *ask device offline* in order to announce, that the device wants to switch into online resp. offline mode. Because of that, this requirement r_{sender_1} is fulfilled. For further details see requirement $r_{receiver_1}$ in 7.1.2.4.1.

$\mathbf{r_{sender_2}}$: Analog to the receiver, each sender contains the place *connection*$_{id}$: $ConnectionID$, to which the individual connection identification will be assigned as token provided by the Message Oriented Middleware after transforming the net from $device^{offline}_{sender}$ into $device^{online}_{sender}$ when switching into online mode. Consequently, the requirement r_{sender_2} is fulfilled.

$\mathbf{r_{sender_3}}$: To send data to the Message Oriented Middleware, a sender holds the transition *send data* and its connected places. After firing, the prepared data assigned to $Data_{toSend}$: $DataToSend$ will be removed, extended by the specific connection identification in order to convert the information into a global data format. The resulting message which is ready for getting processed by the message broker will be assigned to $Data_O$: *OutputData*. The requirement r_{sender_3} is fulfilled.

$\mathbf{r_{sender_4}}$: In addition to assigning data as tokens to $Data_O$, a new token will be assigned to $Data_P$: $DataForPersistenceLayer$ after firing *send data*, so that the persistence layer within the Message Oriented Middleware is able to fetch this token and store it. Therefore, this requirement r_{sender_4} is met.

To summarise, all requirements demanded are fulfilled by the model of a template object net of a sender.

7.1.2.5.2 Visual Model of the Template of a Sender

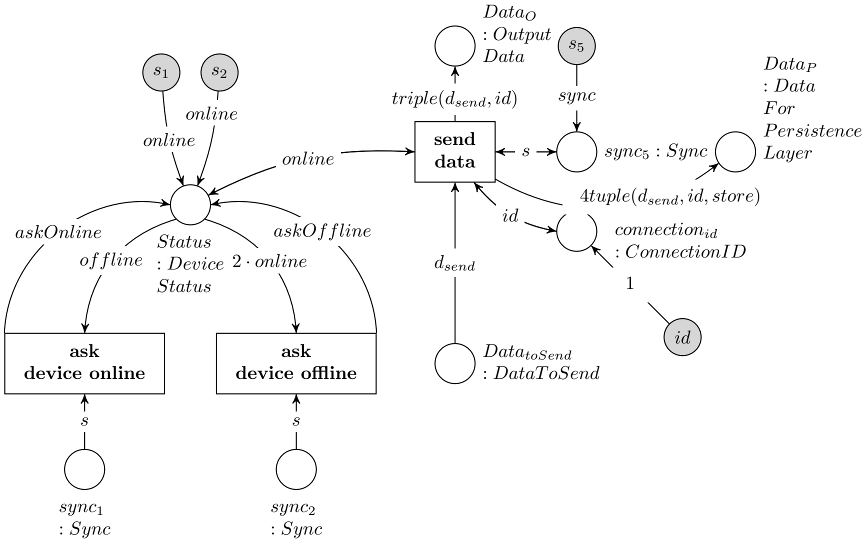


Figure 7.6: Net $device^{online}_{sender}$

7.1.2.6 Template of a Transceiver in Online Mode

After switching to online mode, essential parts are added to the device net of a transceiver, in order to go into online mode and subscribe topics. After that the device is able to process data from and to the Message Oriented Middleware.
After setting a device online, the place : $ConnectionID$ is added to the device net to create an unique identity of the connection between device and message broker. To subscribe topics for receiving data from the Message Oriented Middleware, the transition *subscribe topic* can be fired and a token from $UnsubscribedTopics$: $Topic$ will be put to $TopicsToSubscribe$: $Topic_{toSubscribe}$. The Message Oriented Middleware will check this place and executes the final step for subscribing a topic (see 7.1.3.1.7). Reverse to subscribing a topic, the unsubscribing of a topic will be performed analogously (see 7.41).

The token *askoffline* which was assigned to the place $Status$: $DeviceStatus$ is replaced by a number of tokens *online*. The transceiver has five tokens assigned to the place $Status$ because of the transitions *receive data*, *send data* and *request data* should be able to fire simultaneously. Each of the last mentioned three transitions are connected to a place : $Sync$ which synchronizes the sending, requesting and receiving of data, the subscribing of topics and the firing of *ask device offline*. All these transitions are able to fire in parallel but each transition can only process one token at any time.
To receive data, the Message Oriented Middleware puts data for the specific device to the place $Data_I$: $InputData$ which represents a queue belonging to a certain device which holds all the data which can be collected by the device. If the device wants to receive data it should fire the transition *receive data* to collect all pieces of information from $Data_I$ and puts them to the place $Data_{received}$: $DataReceived$. Therefore $Data_{received}$ represents the internal input data buffer of the device. In an individual transceiver device which will contain this template in online mode, the part performing the tasks will be directly connected to the places $Data_{received}$ and $Data_{toSend}$: $DataToSend$.
In contrast to receiving data, pieces of information which should be sent to the Message Oriented Middleware will be assigned as tokens to the place $Data_{toSend}$ depicting the internal output buffer of the device. To send pieces of information to the Message Oriented Middleware the transition *send data* fires to put the data to the place $Data_O$: $OutputData$, which corresponds to the external output queue of the device whose tokens will be fetched by the Message Oriented Middleware. Additionally, the data to send will be expanded by the individual connection identification of the device in order to publish the individual sender of the information to the message broker, e.g. because to write the information to the persistence layer, a connection identification is required.
Similar to sending data to the Message Oriented Middleware, requests to the persistence layer of the Message Oriented Middleware can be sent in using the transition *request data* and the places $Data_R$: $RequestData$, : $Sync$ and $Data_R$: $RequestData$. Besides, in order to request past pieces of information from the persistence layer, the appropriate topic have to be subscribed to the Message Oriented Middleware. Requests will be sent to Message Oriented Middleware in order to collect past data from the time before the device was connected, which might be relevant for the device, e.g. the Alarm Clock 2.0 will need all dates from the Daily Planner which were registered in the past.
For requesting data from the persistence layer from the Message Oriented Middleware a token will be set at startup to $Topics_{toRequest}$: $Topic_{toRequest}$. This token will be consumed in performing the request. During processing, no additional request is needed, because the device will receive every new message belonging to this topic.

7.1.2.6.1 Revisiting the Requirements Towards the Model of the Template of a Transceiver

In section 4.2.2.3 it is already mentioned, that the transceiver is a combination of a sender and of a receiver, so it should fulfil all requirements demanded for a receiver (see 4.2.2.1) and for a sender (see 4.2.2.2). Because of the transceiver being a combination of both device types, the fulfilment of all these requirements is obvious. In addition, the transceiver has to meet the requirement $r_{transceiver_1}$.

$\mathbf{r_{transceiver_1}}$: In online mode, five tokens are assigned to the place $Status$: $DeviceStatus$ in order to enable the transitions *send data*, *receive data*, *request data*, *subscribe topic* and *ask device offline* to fire in parallel. As a result, the requirement $r_{transceiver_1}$ is met.

7.1.2.6.2 Visual Model of the Template of a Transceiver

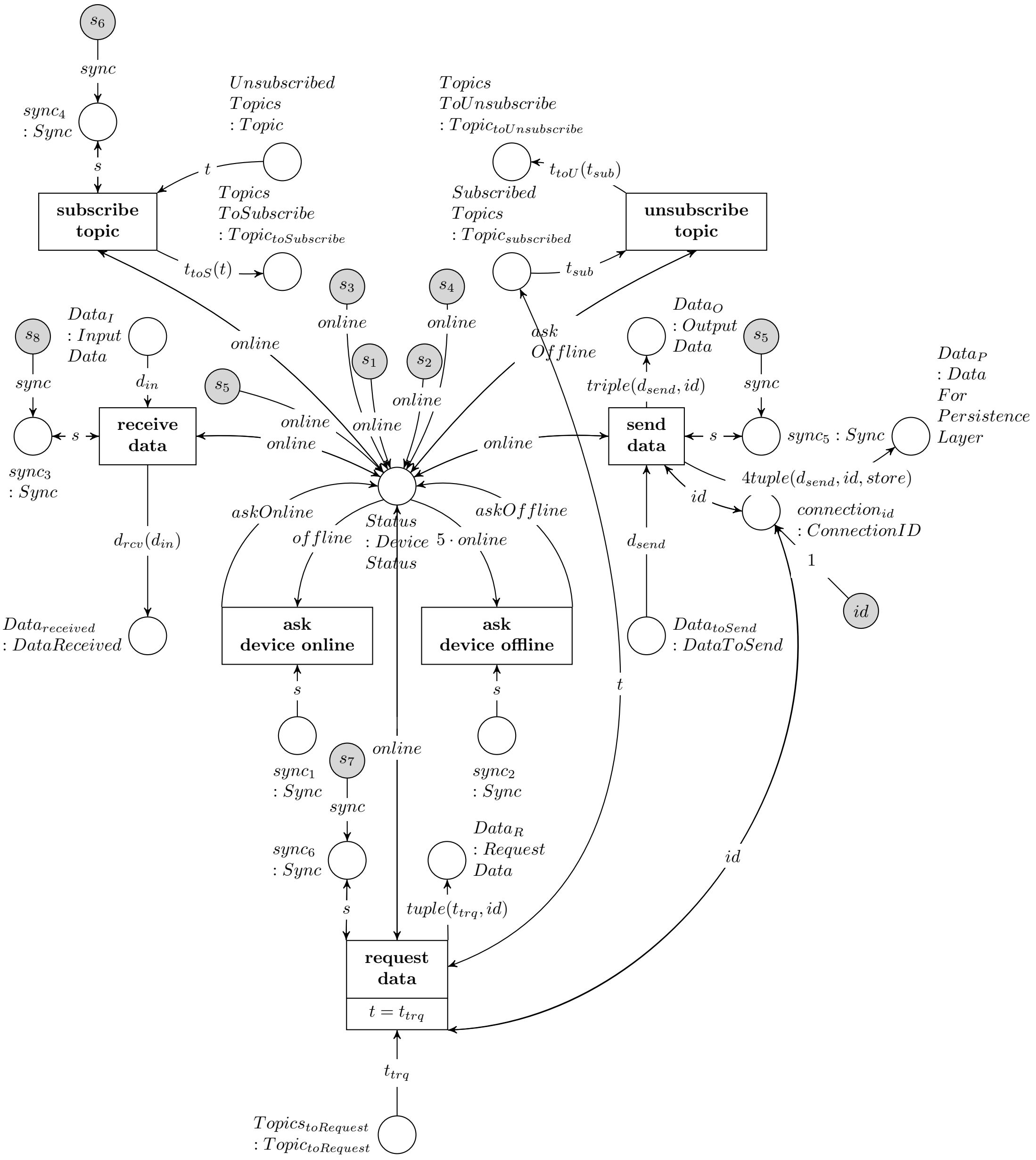


Figure 7.7: Net $device^{online}_{transceiver}$

7.1.2.7 Alarm Clock 2.0

In the following the visual model of the Alarm Clock 2.0 in offline (see figure 7.8) resp. in online mode (see figure 7.9) will be presented. It is a transceiving device which is subscribed to the topics *Calendar*, *Weather*, *Traffic*, *Context* and *Bed*, and after processing, it sends a new message belonging to the topic *Alarm*.

The main task of the Alarm Clock 2.0 will be the process of waking up and reminding the inhabitant of the different steps of preparation, in order to attend an appointment in time. The preparation and wake up time consists of the following steps:

- End sleeping,
- dressing,

- washing and
- eating.

In concerning a preparation time, the Alarm Clock 2.0 does not alert the resident at a certain point of time, instead it calculates preparation time to fulfil fixed steps before the appointment. For each step, a duration of 15 minutes is assumed. To sum up, the pure preparation time for the person takes 60 minutes. Additionally, the device takes different external information, like weather and traffic service, or internal information, like the current individual sleeping stage, into account to calculate the appropriate wake up time for the user. The task of the Alarm Clock 2.0 as presented in [Ell11a] will be the waking up of a sleeping user to a suitable moment.

The states *asleep*, *hungry*, *unwashed* and *undressed*, which are enumerated in table 2.1, are represented by the place $Person : PersonState$ to which the 4-tuple $(asleep, hungry, unwashed, undressed)$ will be assigned as token, when a new wake up time was able to be calculated in firing the transition *calculate wake up time*. Initially a token *(undefined,undefined,undefined,undefined)* is assigned to this place, which means that the user has no tasks to fulfil if no valid or forthcoming appointment is available.

In order to operate, the Alarm Clock 2.0 requires appointment data which will be entered by the resident in the Daily Planner whose model is presented in section 7.1.2.9. All information introduced within the Daily Planner will be received by the Alarm Clock 2.0 because it is subscribed to the topic *Calendar*. Additionally, the Alarm Clock 2.0 contains an internal timer which generates the current time. As described in 2.2.2.1 resp. [Ell11a], the Alarm Clock 2.0 contains a *TimeModule* sending the current time every minute. In the model, the timer is represented by the transition *set time* and the place $Timer : Timestamp$. Because of the formal modelling techniques, as presented in chapter 6, the timer can not be completely corresponding to a real internal clock. A further discussion of the problem regarding time in AHOI systems will be presented in chapter 10.

In receiving new calendar data, the Alarm Clock 2.0 checks all available dates and compares them with the current time generated by the timer. If the given timestamp of a calendar data token assigned to the place $Data_{received} : DataReceived$, is in the near future, which means within the next 240 minutes, then this token will be transferred to the place $NextAppointment : CalendarData$ and will be ready for inferring the appropriate wake up time. Initially an invalid calendar data token $(0, unknown)$ is assigned to the place $NextAppointment$. In the model only one token is able to be set to that place at one time because the Alarm Clock 2.0 should only be able to process one appointment at the same time. After the appointment is passed, the token assigned to this place is initialised so that a new data record from the topic *Calendar* is able to be checked.

Additionally, the Alarm Clock 2.0 is subscribed to the topics *Weather* and *Traffic* to receive the latest weather and traffic information. After receiving, both kinds of data will be compared and according to the rules in tables 4.1 resp. 4.2 additional time for preparation will be gathered. Both, weather and traffic data will be collected analogously in checking all data on $Data_{received}$ for weather resp. traffic information in firing the transition $analyse\ traffic\ data$ resp. *analyse weather data*. Thereby, the extra time because of possibly bad weather or traffic conditions will be calculated with the help of the operations $calcAddTimeWeather$ resp. $calcAddTimeTraffic$. All operations used within the transitions will be characterized in detail within the Σ_{OL}-algebra A_{OL} of the Object Level in 11.1. As a result the extra time in minutes will be put to the place $AddTime : Timestamp$. Initially a token with the value of 0 minutes is assigned to the place meaning that good weather and good traffic conditions are assumed.
As next step, the wake up time will be calculated in firing the transition *calculate wake up time*, in order to generate a token representing the next appointment which will be assigned to the place $NextAppointment$. For that, the timestamp from $AddTime$ will be required. In doing so the operation $calcWakeUpTime$ calculates the optimal wake up time of the inhabitant. The operation is described in 11.1. In short it calculates:

$$time = x - (time_{trafficAndWeather} + 120), \text{ where}$$

x is the timestamp from the next appointment,

$time_{trafficAndWeather}$ contains extra time in minutes because of the traffic and weather conditions,

120 because of:
60 is the assumed preparation time in minutes,
30 is the assumed time the user will need to get to the appointment in minutes and
30 is a time buffer in minutes because the Alarm Clock 2.0 possibly needs to wait
for the light sleeping phase to wake up the inhabitant.

The resulting wake up time will be placed to $WakeUpTime$: *Timestamp*. At the same time all tokens representing the resident tasks for the preparation time will be placed to $Person$: $PersonState$. As next step, the wake up time needs to be compared to the current timestamp generated by the internal timer whose result is assigned as token to the place $Timer$. For that, the transition *wakeup* needs to be fired. Additionally, if both timestamps are identical, the actual sleeping phase of the user needs to be checked in analysing the token assigned to *Bed* : *BedData*. If the user is in deep sleep phase, the Alarm Clock 2.0 should wait until the resident reaches the next phase to make the phase of getting woken up more comfortable to the resident.
For getting the current sleeping phase, the transition *analyse bed data* collects a token belonging to the topic *Bed* from $Data_{received}$ and puts it to the place *Bed*. If the user is in the deep sleep phase, the wake up time will be increased by one minute and put back to the place $WakeUpTime$ in order enable the check if the person has changed his sleeping stage every minute and as long as necessary. In contrast, if the user is not in a deep sleep phase, the Alarm Clock 2.0 initiates an alarm signal in the Living Place system in putting $(on, Alarm)$ to the place $Data_{toSend}$. This means, that the data *on* belonging to the topic *Alarm* will be sent from the Alarm Clock 2.0 device to the Message Oriented Middleware. Furthermode, the token containing the wake up time will be increased by 15 minutes and transferred to the place $WakeUpTime'$: *Timestamp* and the state *asleep* will be removed from the place *Person* representing that the user has been woken up and is not asleep anymore.

As shown before, the resident has four states which need to be fulfilled in order to stop the Alarm Clock 2.0 from reminding the user of his tasks, which are assigned as 4-tuple to the place *Person*. After firing the transition *wake up*, the state *asleep* will be removed in replacing the first position of the tuple containing the state *asleep* by $undefined$. The remaining tasks *hungry*, *unwashed* and *undressed* will be removed according to the received context data. For that, the transition *analyse context data* fires and collects a token belonging to the topic *Context* (describing a 5W1H context) and transfers it to the place *Context* : *ContextData*. Afterwards, the tansitions *washing in bathroom*, *eating in kitchen* and *dressing in sleeping area* are able to check, if one of the states, included in the 4-tuple assigned to *Person*, can be removed. To remove a task, the defined position of the task is replaced by $undefined$. The static positions are: *asleep, unwashed, undressed, hungry.*
E.g. if the person is awake, has nothing else fulfilled until now and is currently eating, the 4-tuple *(undefined, unwashed, undressed, hungry)* will be replaced by *(undefined, unwashed, undressed, undefined).*

During the preparation time, that means, if the moment of the appointment is still in the future, the Alarm Clock 2.0 device needs to remember the user of his tasks as long as they are not fulfilled. That means, as long as there is a token assigned to *Person*, which is not equal to *(undefined,undefined,undefined,undefined)*, the transition *alarm again* is able to fire. Then, this transition checks, if 15 minutes are gone in comparing the wake up time with the current timestamp. If 15 minutes are gone and at lesat one task is still open, a new alarm $(on, Alarm)$ will be generated and sent to $Data_{toSend}$. As long as the internal timestamp of the Alarm Clock 2.0 is not equal to the timestamp of the token assigned to *NextAppointment*, menaing that the appointment is not reached until now, this procedure will be repeated until the user finished all of his tasks or the appointment has passed.

The transition *is appointment over* checks, if the current appointment has already passed an initiates a reset of the relevant places: *NextAppointment* : *CalendarData* in changing the token assigned to *NextAppointment* into the invalid calendar data token $(0, unkwnon)$ and removing a synchronize token from the place $WakeUpMode$: *Sync* while a new synchronize token is set to *RemoveContext* : *Sync*. Both synchronize places regulate the deletion or processing of context data. If a token is assigned to *RemoveContext*, the transition *remove context* is active and removes all context data from the received data. This transition is only active, when no wake up time is available and the Alarm Clock 2.0 device is in idle mode. Then, all incoming context data will be removed in order to prevent a wrong inference process within the Alarm Clock 2.0 because of old and expired data. If the Alarm Clock 2.0 calculated a new wake up time, it removes the synchronize token from *RemoveContext* and puts a new token to

$WakeUpMode$ in order to deactivate the transition *remove context* and to activate the transition *analyse context data* which processes all incoming context data, in transforming it to the format of *ContextData* and provides it to the transitions *washing in bathroom*, *eating in kitchen* and *dressing in sleeping area* in assigning the data to the place *Context* : *ContextData*. The token assigned to *Context* consists of a 3-tuple containing information regarding the tasks the inhabitant should perform. Initially it is set to *(unknown, unknown, unknown)*, because no information will be available, and in further processing steps, it is able to be replaced by relevant context information. Every position represents a defined kind of information regarding the tasks (see $Person$: *PersonState*), whereas the positions have the meaning: *unwashed, hungry, undressed.* They will be replaced by the corresponding information, if an incoming context belongs to one of the three relevant information.

The Alarm Clock 2.0 device is modelled on the Object Level. But in addition the inhabitant should be able to interact with the Alarm Clock 2.0 in order to stop the device. For that, the AHOI rule $StopAlarmClock_{user}$ will be provided (see 7.1.4.5). It belongs to the User Level and resets the Alarm Clock 2.0 in using the following transformation rules from System Level (see 7.1.3.5.1):

1. $ruleDeleteNextAppointments$, see 7.63,
2. $ruleDeleteBedData$, see 7.64,
3. $ruleDeleteContextData$, see 7.65,
4. $ruleDeleteWakeUpTime_1$, see 7.66,
5. $ruleDeleteWakeUpTime_2$, see 7.67,
6. $ruleDeletePersonState$, see 7.68,
7. $ruleResetTimestamp$, see 7.69, and
8. $ruleResetSync$, see 7.70.

7.1.2.7.1 Revisiting the Requirements Towards the Model of the Alarm Clock 2.0

In 4.2.2.4 the requirements regarding the description of the Alarm Clock 2.0 device were introduced. In the following part, each specific requirement of the Alarm Clock 2.0 device will be revised for its fulfilment by its modelling. Additionally, the Alarm Clock 2.0 is a transceiver, whose general requirements are presented in 4.2.2.3 and the fulfilment of all general requirements for a transceiver device are checked in 7.1.2.6.
The model of the Alarm Clock 2.0 device in offline resp. online mode is visualized in 7.8 resp. 7.9.

$\mathbf{r_{AlarmClock2.0_1}}$: The first requirement demands, that the Alarm Clock 2.0 device should be able to subscribe to resp. unsubscribe from the topics *Weather*, *Traffic*, *Calendar*, *Bed* and *Context* in order to receive topic-based information. Because of the object net of the Alarm Clock 2.0 is a transceiver, it is able to receive topic-based information (see 7.1.2.6). Additionally, the tokens $Calendar, Weather, Traffic, Context$ and *Bed* are initially assigned to the place *UnsubscribedTopics* : *Topic*, all of them represent one of the five topics to which the object net $alarmClock2.0_{online}$ will be able to subscribe, after it is set into online mode.
Consequently, the requirement $r_{AlarmClock2.0_1}$ is fulfilled.

$\mathbf{r_{AlarmClock2.0_2}}$: The model of the Alarm Clock 2.0 device is able to provoke a signal in firing one of the transitions *wake up*, *alarm again* or *is appointment over.* Then a new token $(on, Alarm)$ will be generated and assigned to the place $Data_{toSend}$: *DataToSend*. This tuple points to the data *on* belonging to the topic *Alarm.* Afterwards this data will be extended to a triple and assigned to the place $Data_O$: *OutputData.* This triple corresponds to a general data format which is readable for all other components of the Living Place system, i.e. all other connected devices on Object Level and the Message Oriented Middleware on System Level. The message broker checks this place and distributes this information to all devices subscribed to the specific topic *Alarm.* These devices are able to perform any action in order to alarm the inhabitant of the Living Place apartment. Therefore the action of alarming will not be executed by the Alarm Clock 2.0 device, instead it sends a message containing the signal for

other devices, that an alert should be provoked.
As a result, the requirement $r_{AlarmClock2.0_2}$ is met by the model of the Alarm Clock 2.0 device.

$\mathbf{r_{AlarmClock2.0_3}}$: The description of the Alarm Clock 2.0 requires, that the device should be able to request past data according to the topic *Calendar* from the persistence layer of the Message Oriented Middleware of the Living Place system. For that, a token *Calendar* is initially assigned to the place $Topics_{toRequest} : Topic_{toRequest}$. This token is able to be processed by the device net $alarmClock2.0_{online}$. After this token is processed by the persistence layer and the message broker, all past information regarding the calendar data will be provided in assigning each data record as token to the place $Data_I$: *InputData*.
Therefore, this requirement $r_{AlarmClock2.0_3}$ is fulfilled by the model.

$\mathbf{r_{AlarmClock2.0_4}}$: The transition *analyse calendar data* checks all available calendar with the following operations

- $isCalendarData(second_{dataReceived}(d_r)) = tt$
  This condition checks, if the token assigned to $Data_{received}$: *DataReceived* belongs to the topic *Alarm*.
- $isSoon(first_{dataReceived}(d_r), ts_2) = tt$
  This operation checks, if the timestamp of the selected token describes the time within the next 240 minutes.
- $first_{dataReceived}(d_r) = cal$
  The timestamp of the selected calendar data will be extracted from the triple.
- $getInfo(cal') = unknown$
  The token assigned to *NextAppointment* : *CalendarData* will only be replaced, if the processing of the last appointment is finished. This will be indicated, when the second part of the tuple is *unknown*.

Because of the usage of the operation *isSoon*, the requirement $r_{AlarmClock2.0_4}$ is met.

$\mathbf{r_{AlarmClock2.0_5}}$ **and** $\mathbf{r_{AlarmClock2.0_{11}}}$: To infer a new wake up time, the model of the Alarm Clock 2.0 fires the transitions *analyse calendar data* (see last requirement $r_{AlarmClock2.0_4}$) and *analyse traffic data* resp. *analyse weather data*. Each of them selects weather resp. traffic information taking place in the period of time of the way to the appointment. To get to the appointment, a duration of 30 minutes is assumed.
If bad conditions are registered, additional time will be calculated and assigned as token to the place *AddTime* : $Timestamp$. The additional time because of bad weather resp. traffic conditions are based on the tables 4.1 and 4.2. The corresponding operations *calcAddTimeWeather* and $calcAddTimeTraffic$ are mentioned in detail within the Σ_{OL}-algebra A_{OL} of the Object Level in 11.1. For the computation of an appropriate wake up time, the transition *calculate wake up time* fires and, in the process, takes the selected appointment assigned as token to *NextAppointment* and the additional time assigned to *AddTime* for its calculations. The wake up time will be assigned to $WakeUpTime : Timestamp$.
The requirements $r_{AlarmClock2.0_5}$ and $r_{AlarmClock2.0_{11}}$ are fulfilled.

$\mathbf{r_{AlarmClock2.0_6}}$ **and** $\mathbf{r_{AlarmClock2.0_7}}$: After infering an appropriate wake up time, the device net $alarmClock2.0_{online}$ checks the current sleeping phase the user is in assigned to the place *Bed* : *BedData* after firing *analyse bed data*. If the user is in deep sleep phase, the transition *change wake up time* will fire and in doing so, it increases the wake up time by one minute and replaces the previous wake up time assigned to $WakeUpTime : Timestamp$ by the new one. Because of a possible waiting time, an additional buffer of 30 minutes is taken into account when infering the wake up time in firing the transition *calculate wake up time* and especially using the operation $calcWakeUpTime(cal, Time_1) = time$. The operation is presented in explicit in 11.1.
If the user is not in a deep sleep phase, the object net of the Alarm Clock 2.0 initiates an alarm in firing the transition *wake up* which generates the signal $(on, Alarm)$.
The model of the Alarm Clock 2.0 device fulfils the requirements $r_{AlarmClock2.0_6}$ and $r_{AlarmClock2.0_7}$.

$\mathbf{r_{AlarmClock2.0_8}}$: The requirement $r_{AlarmClock2.0_8}$ demands an internal storage for administering the four tasks the user should fulfil during the preparation time: getting up, washing, dressing and eating. This storage is represented by the place $Person : PersonState$ and the token assigned to this place. Additionally, $r_{AlarmClock2.0_8}$ demands a duration of 15 minutes for each task. The Alarm Clock 2.0 reminds the user of the fulfilment of each task after 15 minutes, if at least one task is still open. This will be done by the transition *alarm again* which contains the following conditions:

- $is15MinutesLater(ts_1, ts_2) = tt$
  This condition checks, if 15 minutes have passed since the last alarm.

- $isBeforeNextAppointment(cal, ts_1, time) = tt$
  This operation checks, if the time for leaving the apartment in order to get to the appointment has not passed.

- $+_{time}(ts_1, 15) = ts_1'$
  The next alarm should possibly take place in 15 minutes, therefore the timestamp for the next alarm will be increased by 15 minutes.

- $isReadyStates(ps) = ff$
  An alarm will be provoked as long, as at least one task is not fulfilled.

When infering the wake up time in firing *calculate wake up time*, 60 minutes will be taken into account for the preparation phase.
To summarise, the requirement $r_{AlarmClock2.0_8}$ is fulfilled by the model of the Alarm Clock 2.0.

$\mathbf{r_{AlarmClock2.0_9}}$: To check context data constantly, the transition *analyse context data* collects data records belonging to the topic *Context* and classifies it. Afterwards one of the transitions *washing in bathroom*, *eating in kitchen* and *dressing in sleeping area* evaluates the collected data and simultaneously, if a new information describes a task, it will be removed from the task list assigned to $Person : PersonState$. Consequently, the requirement $r_{AlarmClock2.0_9}$ is met.

$\mathbf{r_{AlarmClock2.0_{10}}}$: The device net $alarmClock2.0_{online}$ has two modes: If an appointment is currently available and an alert is processed, context data will be collected and evaluated in order recognize the processing of the tasks for the preparation time by the inhabitant of the Living Place apartment. For that, a synchronize token will be assigned to the place $WakeUpMode : Sync$, whereas no token is set to $RemoveContext : Sync$.
If no appointment is currently processed, the Alarm Clock 2.0 is in idle mode which is represented by a token assigned to $RemoveContext$ and no token assigned to $WakeUpMode$. Then, all incoming context data will be removed with the help of the transition *remove context*.
As a result, the requirement $r_{AlarmClock2.0_{10}}$ is fulfilled.

$\mathbf{r_{AlarmClock2.0_{12}}}$: The requirement $r_{AlarmClock2.0_{12}}$ demands an internal timer representing the TimeModule. This time is represented by the place $Timer : Timestamp$, the token assigned to this place and the transition *set time*. The token assigned to $Timer$ contains the current time which will be used by the transition *analyse calendar data*, which collects a new appointment, the transition *wake up* which initiates a new message $(on, Alarm)$, the transition *alarm again*, which generates the message $(on, Alarm)$, again and increases the wake up time by 15 minutes, and *is appointment over* which checks, if the point of time for leaving the Living Place apartment is reached, which initiates the final $(on, Alarm)$ signal and resets the Alarm Clock 2.0 (see requirement $r_{AlarmClock2.0_{14}}$).
The requirement $r_{AlarmClock2.0_{12}}$, which demands an internal timer, is met by the model of the Alarm Clock 2.0 device.

$\mathbf{r_{AlarmClock2.0_{13}}}$: To remind the user of the fulfilment of his tasks during the preparation time, a message $(on, Alarm)$ will be created after firing the transition *alarm again*. The transition fires every 15 minutes, but as long, as there are still some tasks open.
Therefore, this requirement $r_{AlarmClock2.0_{13}}$ is met by the model.

$\mathbf{r_{AlarmClock2.0_{14}}}$: The transition *is appointment over* checks, if the time for leaving the Living Place apartment in order to get to the appointment in time, is reached. The following conditions are taken into account:

- $-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
  This operation calculates the time for leaving the apartment, whereas
  - $getTimestamp_{Calendar}(cal')$ extracts the timestamp of the currently processed appointment assigned as token to $NextAppointment : CalendarData$.
  - $time$ denotes the additional time assigned to $AddTime : Timestamp$, resulting out of bad weather and traffic conditions for the time for getting to the appointment, i.e. the time from leaving the apartment and the exact timestamp of the appointment.
  - 30 represents the assumed duration for the way to get to the appointment.
  - ts_2 is the current timestamp.
- $cal = (0, unknown)$
  After firing the transition *is appointment over*, the appointment data assigned to $NextAppointment : CalendarData$ needs to be reset in replacing the current token by the tuple $(0, unknown)$.

The transition *is appointment over* and *alarm again* are defined containing inverse transition conditions to each other, so that they are in conflict, i.e. either *alarm again* or *is appointment over* is able to fire for the same token set to the place $WakeUpTime' : Timestamp$.
Consequently, the requirement $r_{AlarmClock2.0_{14}}$ is fulfilled.

$\mathbf{r_{AlarmClock2.0_{15}}}$: After firing the transition *is appointment over*, the token assigned to $AddTime : Timestamp$ will be replaced by 0_t, which is described within the Σ_{OL}-algebra A_{OL} of the object net in 11.1. Additionally, the token assigned to $NextAppointment : CalendarData$ is replaced by $(0, unknown)$. The syncronize token assigned to $WakeUpMode : Sync$ will be removed, instead a token will be assigned to $RemoveContext : Sync$ in order to switch from processing into idle mode and to be able to remove all incoming context data without evaluating them. In firing this transition, the timestamp for the next reminding assigned as token to $WakeUpTime' : Timestamp$ will be removed.
In order to reset the whole Alarm Clock 2.0, the internally stored tasks need to be reset. The reset of the place $Person : PersonState$ will be done, when a new wake up time will be calculated in firing *calculate wake up time*.
Because of the removing of the current wake up time, the reset of the appointment, the additional time and the internal mode of the Alarm Clock 2.0 (i.e. processing mode resp. idle mode), after firing the transition *is appointment over*, and the reset of the internal task list after firing the transition *calculate wake up time*, the requirement $r_{AlarmClock2.0_{15}}$ is fulfilled.

$\mathbf{r_{AlarmClock2.0_{16}}}$: The AHOI rule $StopAlarmClock_{user}$ is provided, as described in section 7.1.4.5 enabling the inhabitant to stop and reset the device net $alarmClock2.0_{online}$ in using a set of AHLI rules presented in 7.1.3.5.1.
Therefore, the requirement $r_{AlarmClock2.0_{16}}$ is met.

$\mathbf{r_{AlarmClock2.0_{17}}}$: The model of the Alarm Clock 2.0 is a transceiver, thus, the requirement $r_{AlarmClock2.0_{17}}$ is fulfilled.

The model of the Alarm Clock 2.0 device in offline resp. online mode fulfils all requirements which are presented in section 4.2.2.4.

7.1.2.7.2 Alarm Clock 2.0 in Offline Mode

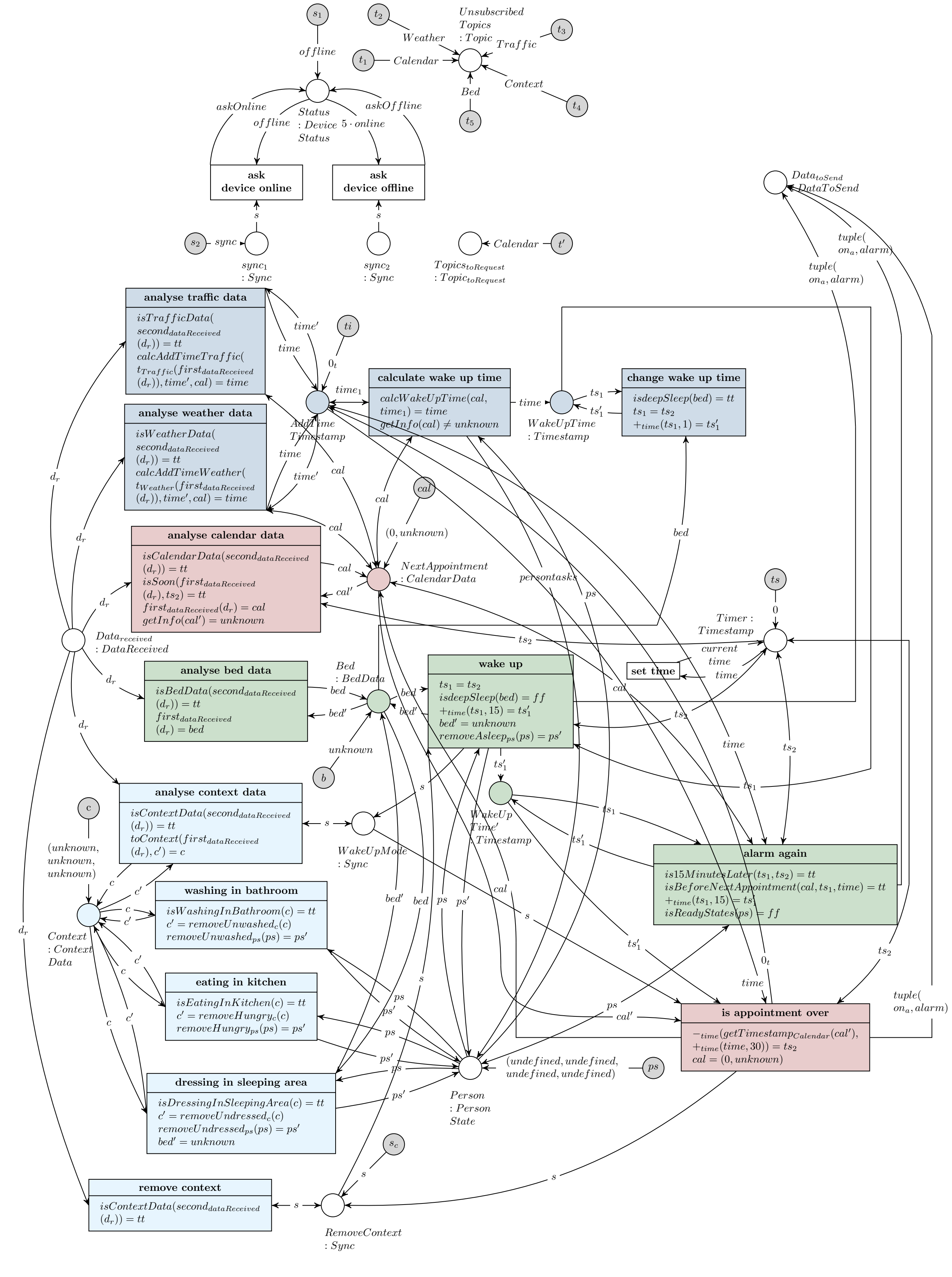


Figure 7.8: Net of the Alarm Clock 2.0 $alarmClock2.0_{offline}$

7.1.2.7.3 Alarm Clock 2.0 in Online Mode

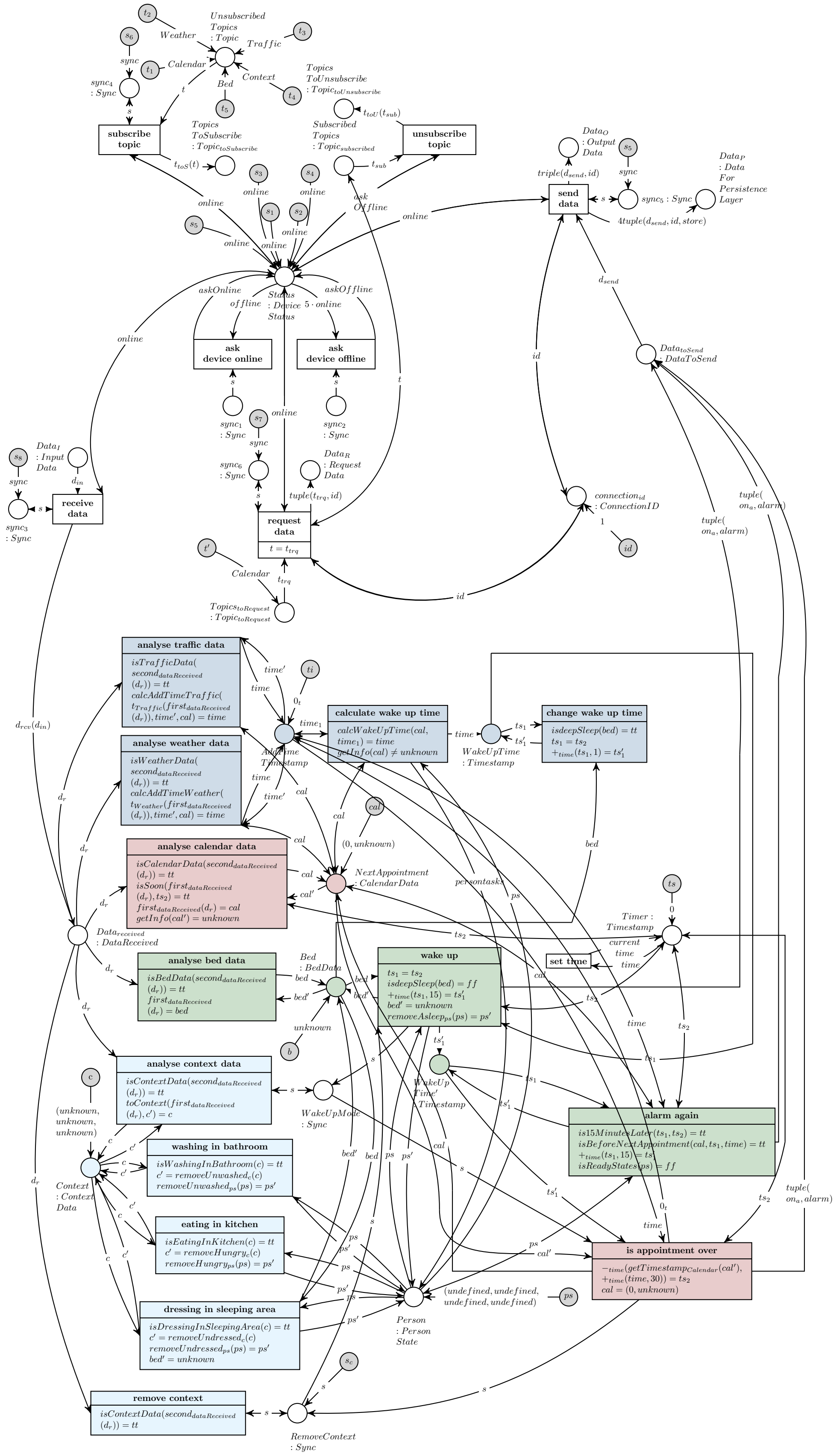

Unsubscribed Topics : Topic
Weather
Traffic
Calendar
Bed
Context
sync
sync4 : Sync
subscribe topic
Topics ToSubscribe : Topic_toSubscribe
$t_{toS}(t)$
Topics ToUnsubscribe : Topic_toUnsubscribe
$t_{toU}(t_{sub})$
Subscribed Topics : Topic_subscribed
t_{sub}
unsubscribe topic
Data_O : Output Data
triple(d_send, id)
send data
sync5 : Sync
Data_P : Data For Persistence Layer
4tuple(d_send, id, store)
online
askOffline
Status : Device Status
askOnline
offline
5 · online
ask device online
ask device offline
sync1 : Sync
sync2 : Sync
d_send
Data_toSend DataToSend
id
t
Data_I : Input Data
d_in
receive data
sync3 : Sync
sync6 : Sync
Data_R : Request Data
tuple(t_trq, id)
request data
t = t_trq
connection_id : ConnectionID
tuple(on_a, alarm)
Calendar
t_trq
Topics_toRequest : Topic_toRequest
d_rcv(d_in)
analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time', cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time', cal) = time
time'
time
0_t
time_1
AddTime Timestamp
calculate wake up time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
WakeUpTime : Timestamp
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts_1'
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal') = unknown
(0, unknown)
NextAppointment : CalendarData
persontasks
Timer : Timestamp
current time
set time
Data_received : DataReceived
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
Bed : BedData
unknown
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts_1'
bed' = unknown
removeAsleep_ps(ps) = ps'
WakeUp Time' : Timestamp
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts_1'
isReadyStates(ps) = ff
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c') = c
WakeUpMode : Sync
(unknown, unknown, unknown)
Context : Context Data
washing in bathroom
isWashingInBathroom(c) = tt
c' = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps'
eating in kitchen
isEatingInKitchen(c) = tt
c' = removeHungry_c(c)
removeHungry_ps(ps) = ps'
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c' = removeUndressed_c(c)
removeUndressed_ps(ps) = ps'
bed' = unknown
is appointment over
−_time(getTimestamp_Calendar(cal'), +_time(time, 30)) = ts_2
cal = (0, unknown)
(undefined, undefined, undefined, undefined)
Person : Person State
remove context
isContextData(second_dataReceived(d_r)) = tt
RemoveContext : Sync

Figure 7.9: Net of the Alarm Clock 2.0 $alarmClock2.0_{online}$

7.1.2.8 Display in Multitouch Kitchen Counter

The Display of the Multitouch Kitchen Counter will be modelled as a receiving device. As described in section 2.2.2.2, it is subscribed to information about weather, traffic and calendar, which will be collected and shown on the screen of Multitouch Kitchen Counter.

In the model of the Living Place system, the functionality of the Multitouch Kitchen Counter is splitted in two different devices: The Display described in this section and the Daily Planner explained in section 7.1.2.9. The Display device represents the screen of the Multitouch Kitchen Counter showing different kinds of information to the inhabitant. The Daily Planner is an application, collecting the residents appointments and moments to be remembered, which uses the Multitouch Kitchen Counter as an input device.
Because of the possibility of direct user access, the modelling of the Display device and its interface providing direct access is done on all three levels (see 7.1.2).

The topics *Weather*, *Traffic* and *Calendar*, to which the Display device is subscribed are initially assigned as tokens to the place $UnsubscribedTopics : Topic$. When receiving this kind of data, the necessary information will be filtered in executing the operation $first_{dataReceived}(d_r) = d$ (and $t_{display}(d) = d_{display}$ for casting) in firing the transition *receive and output data*. The extracted information will be transfered as token to the place *display* : *DisplayData*, which represents the output on the screen of the Multitouch Kitchen Counter. Data presented on the screen can be removed by the user in applying the AHOI rule $ClearDisplay_{user}$ which is described in section 7.1.4.6. Then, the corresponding AHLI rule $ruleClearDisplay_{display}$ can be applied to remove a token from the place *Display*.

The Display device only requests data with respect to the topic *Calendar* from the persistence layer. Old weather and traffic information are not relevant anymore, because it just Displays new incoming information regarding weather and traffic on the screen, but in contrast it shows all available calendar data.

7.1.2.8.1 Display in Offline Mode

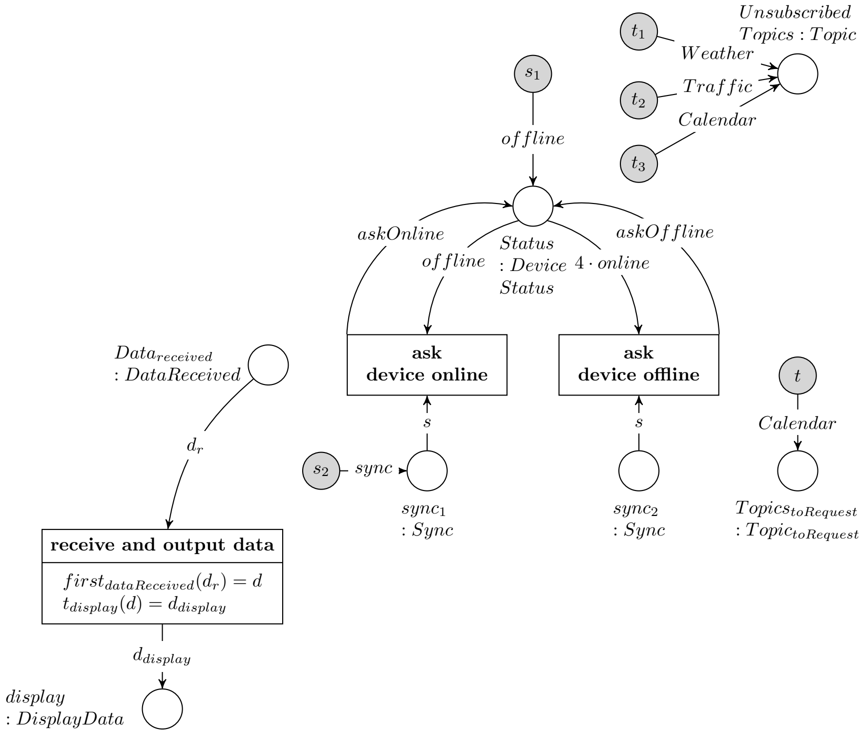


Figure 7.10: Net $display_{offline}$

7.1.2.8.2 Display in Online Mode

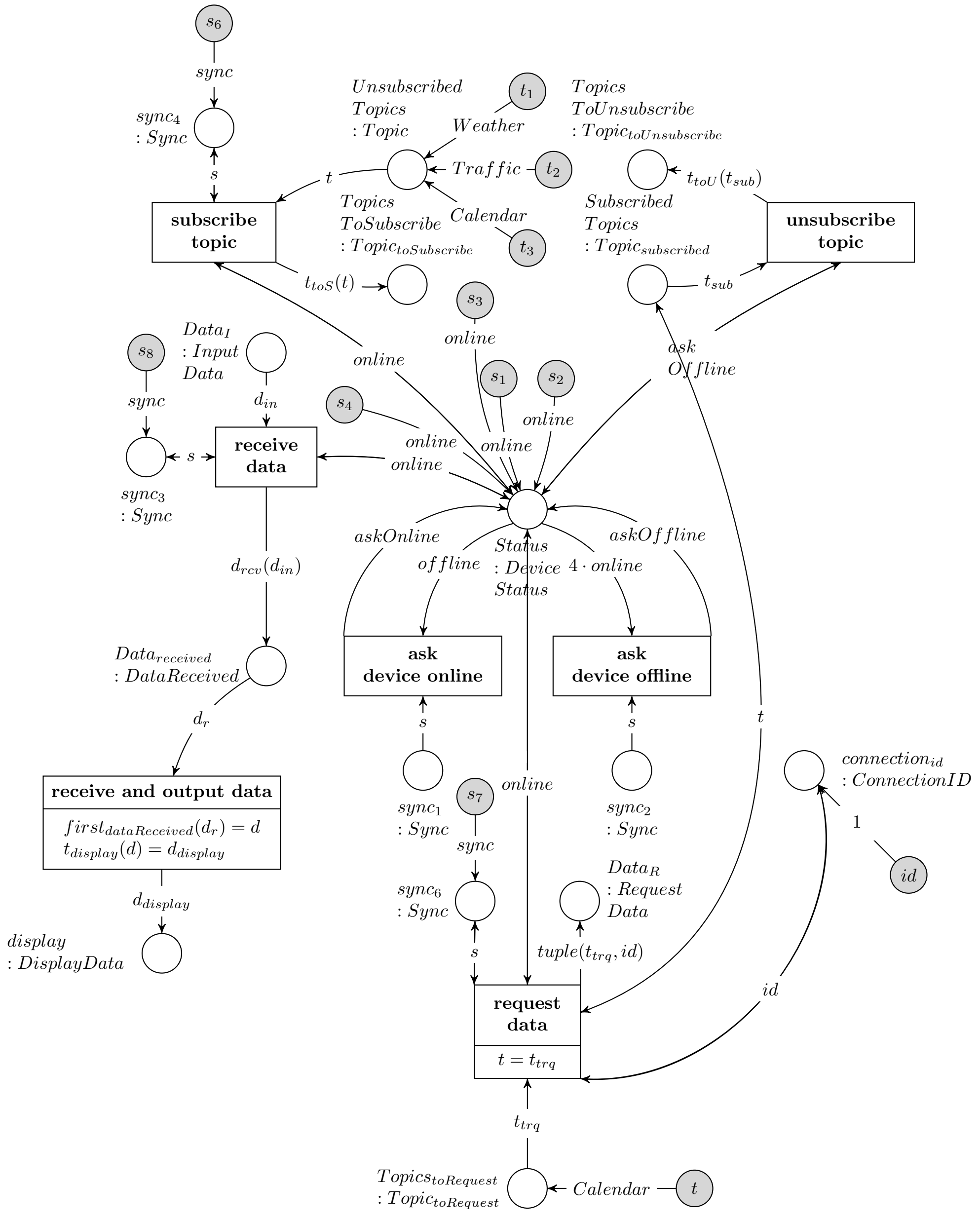


Figure 7.11: Net $display_{online}$

7.1.2.8.3 Revisiting the Requirements Towards the Model of the Display in the Multitouch Kitchen Counter

The requirements that are imposed to the model of the Display device, will be revised in this section. The requirements for this device net are introduced in section 4.2.2.5.

$\mathbf{r_{Display_1}}$: The Display device is able to subscribe resp. unsubscribe the topics *Weather*, *Traffic* and *Calendar*. The topics are initially assigned as tokens to the place *UnsubscribedTopics* : *Topic*. Therefore, the requirement $r_{Display_1}$ is fulfilled.

$\mathbf{r_{Display_2}}$: The transition *receive and output data* gets information belonging to one of the subscribed topics. The transition extracts the first component out of the information and transfers it to the place

$display : DisplayData$ representing the screen within the Multitouch Kitchen Counter.
As a result, the requirement $r_{Display_2}$ is met.

$\mathbf{r_{Display_3}}$: The AHOI rule $ClearDisplay_{user}$ will be provided to remove single data records from the Display device. The AHOI rule is presented in 7.1.4.6.1.
Because of the availability of the AHOI rule $ClearDisplay_{user}$, the requirement $r_{Display_3}$ is fulfilled.

$\mathbf{r_{Display_4}}$: The object net of the Display device is modeled as a receiver, therefore the requirement $r_{Display_4}$ is met.

7.1.2.9 Daily Planner

The Daily Planner, as described in section 2.2.2.3, is able to manage dates which will be input manually by the inhabitant. The Daily Planner provides an interface for direct user access and therefore is modelled on all three levels (see 7.1.2).

The Daily Planner is modelled as a sender, which is shown below in offline (7.1.2.9.2) and online mode (7.1.2.9.3). The main task of the Daily Planner is to collect calendar data, the user wants to be reminded of. For that, the resident enters his new calendar information directly into the system in using the Multitouch Kitchen Counter. The direct input of new data by user the will be represented by the AHOI rule $EnterDataIntoDailyPlanner_{user}$ which is presented in section 7.1.4.7.
Afterwards, the corresponding AHLI rule $ruleAddData_{planner}$ which is presented in section 7.1.3.5.3 will be applied which transforms the Daily Planner device net on Object Level in putting the new data as token to the place $newCalendarData : CalendarData$. The new information will be converted and sent to the message broker in firing the transition *analyse data*. The input data is typed as $CalendarData$ and will be transformed to the type $DataToSend$ (as described in detail as operation of $\Sigma_{OL} - algebraA_{OL}$ of the Object Level in section 11.1).
All calendar information entered by the inhabitant will be stored within the repository in the persistence layer. To delete existing calendar data from repository in the persistence layer, which was previously entered into the Daily Planner, the an AHOI rule out of the infinite set of AHOI rules $DeleteDataFromDaily- Planner_{user,t,d}$ can be used. It needs a corresponding AHLI rule out of the infinite set of AHLI rule $ruleRemoveData_{planner,t,d}$, which assigns a 4-tuple as token to the place $Data_P$ containing *remove* as forth component instead of *store*, which is the default value. The term *remove* signals to the persistence layer of the Message Oriented Middleware, that a corresponding data record should be removed. The AHOI rule is presented in section 7.1.4.7 and the AHLI rule is introduced in 7.1.3.5.4.

7.1.2.9.1 Revisiting the Requirements Towards the Model of the Daily Planner

The requirements for the Daily Planner are presented in section 4.2.2.6, which will be revised in the following section.

$\mathbf{r_{planner_1}}$: The AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$, presented in explicit in 7.1.4.7, is provided for entering new calendar data to the Daily Planner. It uses a corresponding AHLI rule $rule- AddData_{t,d}$ introduced in 7.1.3.5.3.
Therefore, the requirement $r_{planner_1}$ is met.

$\mathbf{r_{planner_2}}$: A second AHOI rule $RemoveDataFromDailyPlanner_{user,t,d}$, introduced in explicit in 7.99, is available for removing calendar data which was previously entered to the Daily Planner. Consequently, the requirement $r_{planner_2}$ is fulfilled.

$\mathbf{r_{planner_3}}$: The transition *analyse data* transforms the new calendar data entered with the help of the AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$ and assigns the data to the place $Data_{toSend} : DataToSend$. After firing the transition *send data*, the token assigned to $Data_{toSend}$ will be extended and assigned to $Data_O$. Then the Message Oriented Middleware is able to collect and process the data. Therefore, the requirement $r_{planner_3}$ is met.

$\mathbf{r_{planner_4}}$: The object net of the Daily Planner is modeled as a sender, thus the requirement $r_{planner_4}$ is fulfilled.

7.1.2.9.2 Daily Planner in Offline Mode

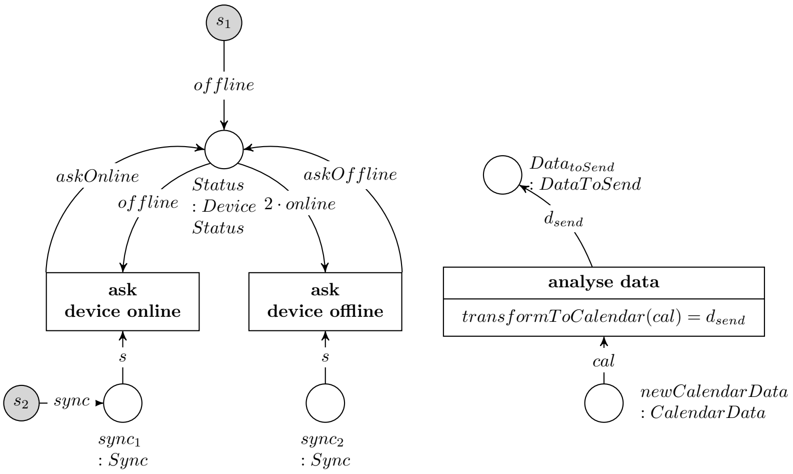


Figure 7.12: Net $dailyPlanner_{offline}$

7.1.2.9.3 Daily Planner in Online Mode

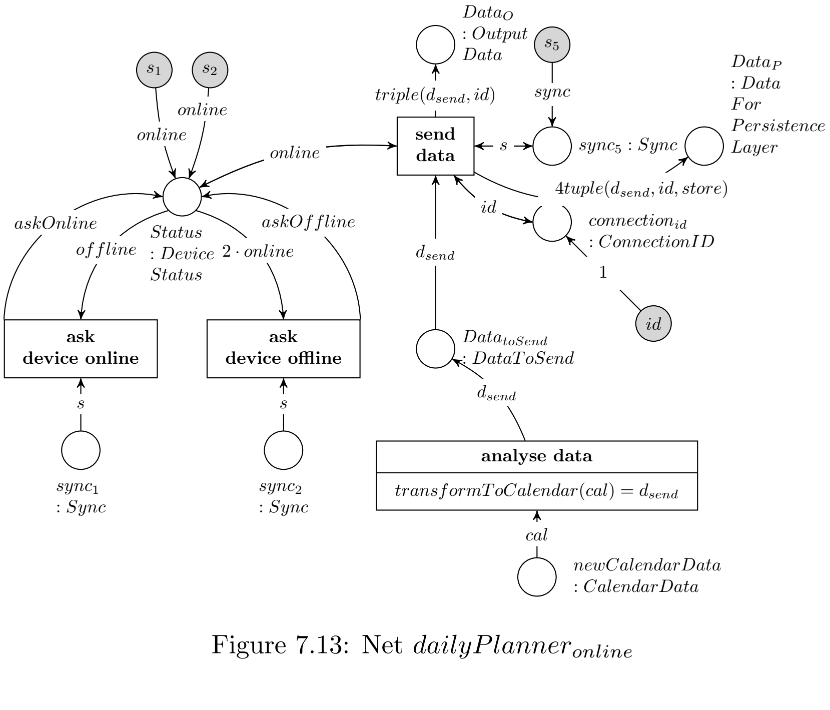


Figure 7.13: Net $dailyPlanner_{online}$

7.1.2.10 Intelligent Bed

Similar to the description in section 2.2.2.4, the model of the Intelligent Bed contains parts representing the six pressure sensors. The collection of all six sensors are represented by the place $Sensor$: $PressureData$. Pressure values resulting from measurement will be set as token to the corresponding place $Sensor$. The value consists of a 6-tuple, where each position in this tuple represents a special sensor according to the numbering in image 4.2.2.7. Initially a 6-tuple $(0, 0, 0, 0, 0, 0)$ is assigned as token to this place, meaning, that there is no pressure on the intelligent bed.
Because of the sensors should measure the pressure that is exerted by the weight of the human body which

sits or lays on the bed, the model of the intelligent bed is done on all three levels: The pressure produced by the weight of the body is regarded as an user input and therefore an AHOI rule exists for entering the pressure data for all six sensors. The AHOI rule $MeasurePressureInBed_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ will be presented in section 7.1.4.8 and the corresponding AHLI rule $ruleMeasurePressure_{bed,v_1,v_2,v_3,}$ $_{v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, for assigning the given 6-tuple with new measurement values as token to the place $Sensor$ within the device of the Intelligent Bed, is shown in section 7.1.3.5.5.

During processing, the transition *receive data* checks, if the token assigned to the place $Sensor$ exceeds a certain threshold [3]. Additionally, it checks if the value has changed, which will be compared with a buffer containing one data token, which consists of a 6-tuple. The last value will be stored in changing the token assigned to $LastValueSensor : PressureData_{OLD}$. Initially the values $(0, 0, 0, 0, 0, 0)$ are set to the place $LastValueSensor$.
To recognize a sleeping phase for waking up, the comparison with the old and the new value needs to be done, because a person in deep sleep phase, in which the inhabitant should not be woken up, has a minimal muscle tone, so he does not move[4], as a result the pressure on all sensors will be constant. A change of pressure means a movement. Besides, the location of the pressure will be important: E.g. if a sensor on the head of the bed (sensor 1 or 2) will exceed the determinated threshold, the resident will be sitting in bed.

All the data from each sensor will be processed together and at the end, in firing the transition *evaluate data*, compared with each other to infer, if the person is in deep sleeping phase, an other sleeping phase, where waking-up is possible, or if he is awake.

The whole visual model of the Intelligent Bed in offline mode is presented in 7.14 and the model in online mode is in 7.15.

7.1.2.10.1 Revisiting the Requirements Towards the Model of the Intelligent Bed

The requirements for the model of the Intelligent Bed will be revised in the following section. The requirements are introduced in section 4.2.2.7.

$\mathbf{r_{bed_1}}$: The pressure data will be entered by the inhabitant of the Living Place apartment with the AHOI rule $MeasurePressureInBed_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ and its corresponding AHLI rule $ruleMeasurePressure_{bed,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ presented in detail in image 7.101 and 7.74. Therefore, the requirement r_{bed_1} is met.

$\mathbf{r_{bed_2}}$: The data entered consists of a 6-tuple, where each component represents the pressure values for each sensor. In image 4.2.2.7, the position of each sensor is illustrated. The operations in the transition conditions of *receive data* and *evaluate data* take over the scheme specified in that image when they are evaluated.
The requirement r_{bed_2} is fulfilled.

$\mathbf{r_{bed_3}}$: The last measurement values are stored within the token assigned to $LastValueSensor$: $PressureData_{OLD}$, which will be used by the transition *receive data* for comparing the old with the new pressure in order to detect a change of the values, i.e. in order to detect a movement. Initially the token assigned to $LastValueSensor$ contains the 6-tuple $(0, 0, 0, 0, 0, 0)$, which means that all sensors measure no pressure.
Because of the availability of the place $LastValueSensor$ and the token assigned to it containing the last values measured, the requirement r_{bed_3} is met.

$\mathbf{r_{bed_4}}$: Each x_i within the 6-tuple takes a value of $x_i \in [0, ..., 10], i \in \{1, 2, 3, 4, 5, 6\}$, which will be set by the AHOI rule and its corresponding AHLI rule (see 7.1.4.8 and 7.1.3.5.5).
Therefore this requirement r_{bed_4} is fulfilled.

[3] For further information see [Har11].

[4] See section 2.2.2.4.

$\mathbf{r_{bed_5}}$: The transition *receive data* contains the following conditions:

- $thresholdExceeded(p) = p''$
  This operation checks, if a defined threshold of 6 is exceeded by each value. If a value is below of this threshold, it means that the pressure is too low to be excerted by the inhabitant, this value will be replaced by 0. If the value is above of or equal to 6, it stays unchanged. The operation is defined within the algebra of the Object Level in 11.1.
  Because of the availability of this condition, this requirement r_{bed_5} is met.

- $hasChanged(f_P(p'), p'') = p_1$
  The second operation compares if the 6-tuple containing the last pressure values differ from the new ones. It returns a tuple containing tt, if they differ, else ff, and containing the previously modified 6-tuple of pressure measurement values. This operation is part of the fulfilment condition for requirement r_{bed_3}, as already mentioned.

$\mathbf{r_{bed_6}}$ **and** $\mathbf{r_{bed_7}}$:

- $checkPressureChange(p_1) = res$
  The operation *checkPressureChange* checks the previously evaluated values done by the transition conditions in *receive data* and infers the correct sleeping phase. It gets a tuple containing the boolean value $b = tt$ or $b = ff$ and the 6-tuple $(x_1, x_2, x_3, x_4, x_5, x_6)$ containing the modified pressure measurement values. This operation performs the following conditions and returns the corresponding value (as defined in 11.1):

  - *awake*
    if $x_1 \neq 0 \vee x_2 \neq 0$
  - *deepSleep*
    if $(x_3 \neq 0 \wedge b = false) \vee (x_5 \neq 0 \wedge b = false)$
  - *deepSleep*
    if $(x_4 \neq 0 \wedge b = false) \vee (x_6 \neq 0 \wedge b = false)$
  - *lightSleep*
    if $(x_3 \neq 0 \wedge b = true) \vee (x_5 \neq 0 \wedge b = true)$
  - *lightSleep*
    if $(x_4 \neq 0 \wedge b = true) \vee (x_6 \neq 0 \wedge b = true)$
  - *notInBed*
    if $x_1 = 0 \wedge x_2 = 0 \wedge x_3 = 0 \wedge x_4 = 0 \wedge x_5 = 0 \wedge x_6 = 0$

- $transformToBed(res) = d_{send}$
  The value $res \in \{awake, deepSleep, lightSleep, notInBed\}$ resulting out of *checkPressureChange* will be taken and a tuple containing this value and the corresponding topic *Bed* will be created.

After firing the transition *evaluate data*, the resulting tuple created by *transformToBed* will be put to the place $Data_{toSend}$: *DataToSend*. Afterwards it will be taken and expanded to the global readable triple, which will be assigned to $Data_O$: *OutputData*, so that the Message Oriented Middleware is able to process this data.
Consequently the requirements r_{bed_6} and r_{bed_7} are fulfilled.

$\mathbf{r_{bed_8}}$: The device net of the Intelligent Bed is modeled as sensor, consequently the requirement r_{bed_7} is met.

As a result of this revision, all requirements posed in section 4.2.2.7 are fulfilled by the model of the Intelligent Bed.

7.1.2.10.2 Intelligent Bed in Offline Mode

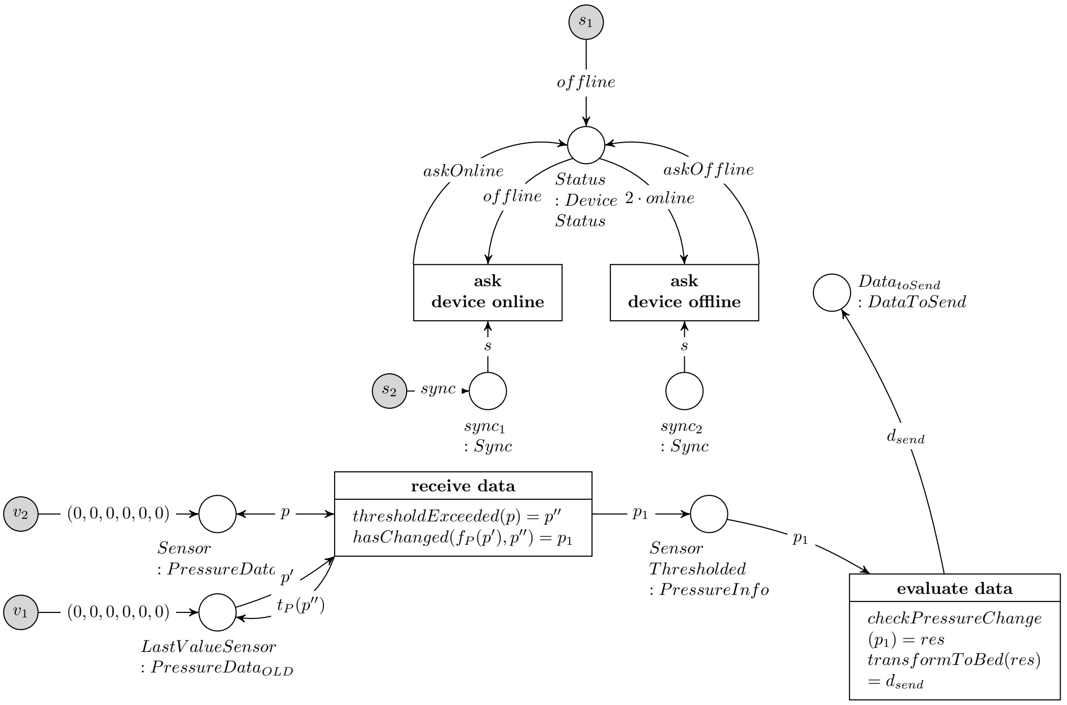

s_1
$offline$
$askOnline$
$askOffline$
$offline$
$Status$ $: Device$ $Status$
$2 \cdot online$
ask device online
ask device offline
$Data_{toSend}$ $: DataToSend$
s
s
s_2
$sync$
$sync_1$ $: Sync$
$sync_2$ $: Sync$
d_{send}
receive data
$thresholdExceeded(p) = p''$
$hasChanged(f_P(p'), p'') = p_1$
v_2
$(0,0,0,0,0,0)$
p
p_1
$Sensor$ $: PressureData$
p'
$t_P(p'')$
v_1
$(0,0,0,0,0,0)$
$LastValueSensor$ $: PressureData_{OLD}$
$Sensor$ $Thresholded$ $: PressureInfo$
p_1
evaluate data
$checkPressureChange$ $(p_1) = res$ $transformToBed(res)$ $= d_{send}$

Figure 7.14: Net $intelligentBed_{offline}$

7.1.2.10.3 Intelligent Bed in Online Mode

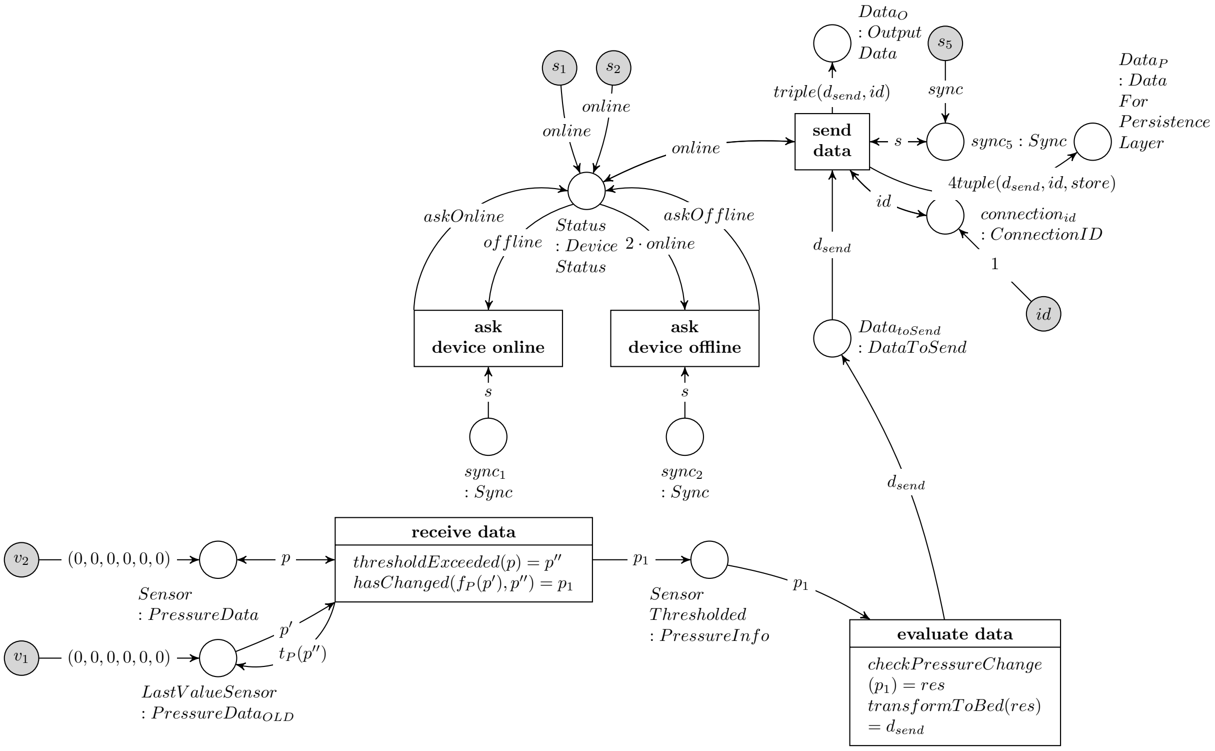

$Data_O$ $: Output$ $Data$
s_5
$Data_P$ $: Data$ For $Persistence$ $Layer$
s_1
s_2
$online$
$online$
$triple(d_{send}, id)$
$sync$
send data
$online$
s
$sync_5 : Sync$
$4tuple(d_{send}, id, store)$
$askOnline$
$askOffline$
id
$connection_{id}$ $: ConnectionID$
$offline$
$Status$ $: Device$ $Status$
$2 \cdot online$
d_{send}
1
id
ask device online
ask device offline
$Data_{toSend}$ $: DataToSend$
s
s
$sync_1$ $: Sync$
$sync_2$ $: Sync$
d_{send}
receive data
$thresholdExceeded(p) = p''$
$hasChanged(f_P(p'), p'') = p_1$
v_2
$(0,0,0,0,0,0)$
p
p_1
$Sensor$ $: PressureData$
p'
$t_P(p'')$
v_1
$(0,0,0,0,0,0)$
$LastValueSensor$ $: PressureData_{OLD}$
$Sensor$ $Thresholded$ $: PressureInfo$
p_1
evaluate data
$checkPressureChange$ $(p_1) = res$ $transformToBed(res)$ $= d_{send}$

Figure 7.15: Net $intelligentBed_{online}$

7.1.2.11 Indoor-Positioning-System

The Indoor Positioning System is described in 2.2.2.5. It measures the position of the inhabitant of the Living Place with the help of six sensors and sends an interpretation of the sensor data to the message broker in stating, if the user currently stays in the north, in the south, in the middle, in the bathroom or outside of the apartment.
All six sensors and the application receiving and interpreting the data will be modelled as one sending component. In the following two parts, the visual modelling of the sending device in offline resp. online mode will be presented. The formal model of the used signature and algebra is presented in 11.1.

The sensors of the Indoor-Positioning-System measure the current position of the resident, hence, the measured values will be entered with the help of an AHOI rule. Therefore the Indoor-Positioning-System will be modelled as object net, but additionally the sensor values will be entered with the help of an AHOI rule and the System Level provides rules for exchanging the data from user to Object Level. The sensor values consist of a 6-tuple containing triples, where each triple represents the measured values of one sensor in 3D-coordinates: (X,Y,Z). So, the 6-tuple contains all data of all sensors: $(s_1, s_2, s_3, s_4, s_5, s_6)$ where $s_i = (X_i, Y_i, Z_i)$ where $X, Y, Z \in [0.0, ..., 15.8]$. Initially it is set to $((0,0,0),(0,0,0),(0,0,0),(0,0,0),(0,0,0),(0,0,0))$.
The AHOI rule for entering data $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ is presented in section 7.1.4.9. The corresponding AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ is shown in section 7.1.3.5.6.
With the help of the AHOI rule, the 6-tuple containing measurement values of each sensor will be entered. Afterwards the AHLI rule transfers the data to the place $SensorData : SensorDataTuple$ in the object net of the Indoor-Positioning-System $indoorPositioningSystem_{online}$. Assigning new data to the net is only possible in online mode. As the next step, the sensor specific data will be interpreted as shown in 2.2.2.5 and also in the formal description of the operation $transformToUbiTV(sensordata)$ within the Σ_{OL}-algebra A_{OL} of the Object Level as listed in 11.1. As a result, a tuple containing the interpreted value *north*, *south*, *bathroom*, *middle* or *nothing* and the name of the topic *UbiTV* will be send to the place $Data_{toSend} : DataToSend$. From that place the information token is available for the next step, where the token will be expanded by the connection identification of this sepcific device and afterwards be sent to the message broker for further processing.

7.1.2.11.1 Revisiting the Requirements Towards the Model of the Indoor-Positioning-System

The Indoor-Positioning-System in online mode should fulfil the requirements which were set up in section 4.2.2.8. The following section revises those requirements in regard to their fulfilment.

$\mathbf{r_{indoor_1}}$: The Indoor-Positioning-System is composed of six sensors. The measurement data will be entered by the AHOI rule $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ which uses the AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$. The AHOI rule is described in explicit in 7.1.4.9, the AHLI rule in 7.1.3.5.6.
The requirement r_{indoor_1} is fulfilled.

$\mathbf{r_{indoor_2}}$: The token assigned to the place $SensorData : SensorDataTuple$ represents the measurement data of all six sensors. Each component of this 6-tuple contains a triple which represents the 3D position values $(x_i, y_i, z_i), (i \in \{1,2,3,4,5,6\})$ for each sensor. If the token will be replaced in applying an AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, a token of the same sturcture has to be assigned to this place.
Consequently, the requirement r_{indoor_2} is fulfilled.

$\mathbf{r_{indoor_3}}$: The transition *analyse data* executes the transition condition *transformToUbiTV(sensordata)* $= d_{send}$ according to the following conditions (see definition of this operation in 11.1):

- (south, UbiTV)
  if $(0.0 \leq second_{real_A}(x_1) < 5.5) \vee (0.0 \leq second_{real_A}(x_2) < 5.5)$
- (north, UbiTV)
  if $(10.0 \leq second_{real_A}(x_5) \leq 15.8) \vee (10.0 \leq second_{real_A}(x_6) \leq 15.8)$

- (middle, UbiTV)
  if $((5.5 \leq second_{real_A}(x_3) < 10.0) \wedge (first_{real_A}(x_3) \leq 8.0))$
  $\vee((5.5 \leq second_{real_A}(x_4) < 10.0) \wedge (first_{real_A}(x_4) \leq 8.0))$
- (bathroom, UbiTV)
  if $((5.5 \leq second_{real_A}(x_3) < 10.0) \wedge (first_{real_A}(x_3) > 8.0))$
  $\vee((5.5 \leq second_{real_A}(x_4) < 10.0) \wedge (first_{real_A}(x_4) > 8.0))$
- (nothing, UbiTV)
  else

The operation returns a tuple containing the resulting position data $\in \{north, south, middle, bathroom, nothing\}$ as first component and the topic *UbiTV* as second component. This tuple will be assigned to the place $Data_{toSend}$: *DataToSend*.
Hence, the requirement r_{indoor_3} is met.

$\mathbf{r_{indoor_4}}$: The tuple assigned to $Data_{toSend}$ will be taken and extended to a global format in firing the transition *send data* and set to the place $Data_O$: *OutputData*. This data is able to be processed by the message broker on System Level.
Consequently, the requirement r_{indoor_4} is fulfilled.

$\mathbf{r_{indoor_5}}$: The Indoor-Positioning-System is modeled as sender, thus the requirement r_{indoor_5} is met.

To summarise, all requirements set up in section 4.2.2.8 are fulfilled by the model of the Indoor-Positioning-System.

7.1.2.11.2 Indoor-Positioning-System in Offline Mode

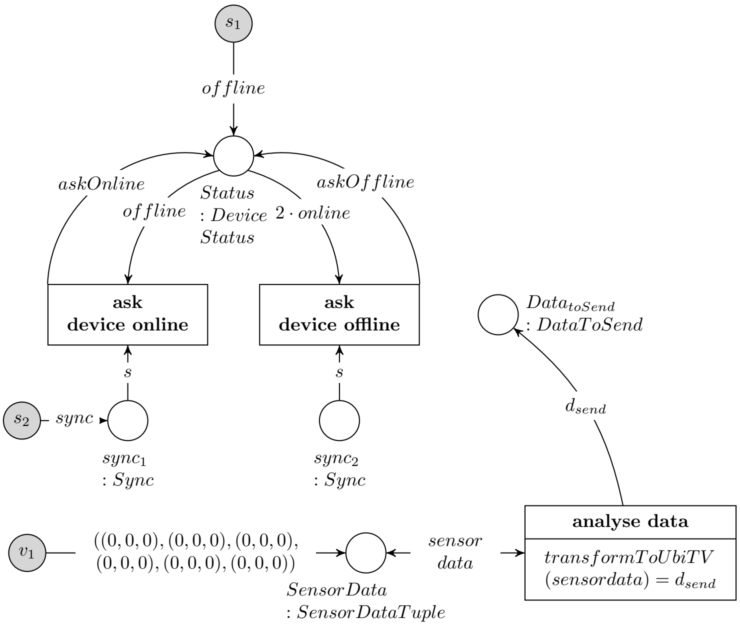


Figure 7.16: Net $indoorPositioningSystem_{offline}$

7.1.2.11.3 Indoor-Positioning-System in Online Mode

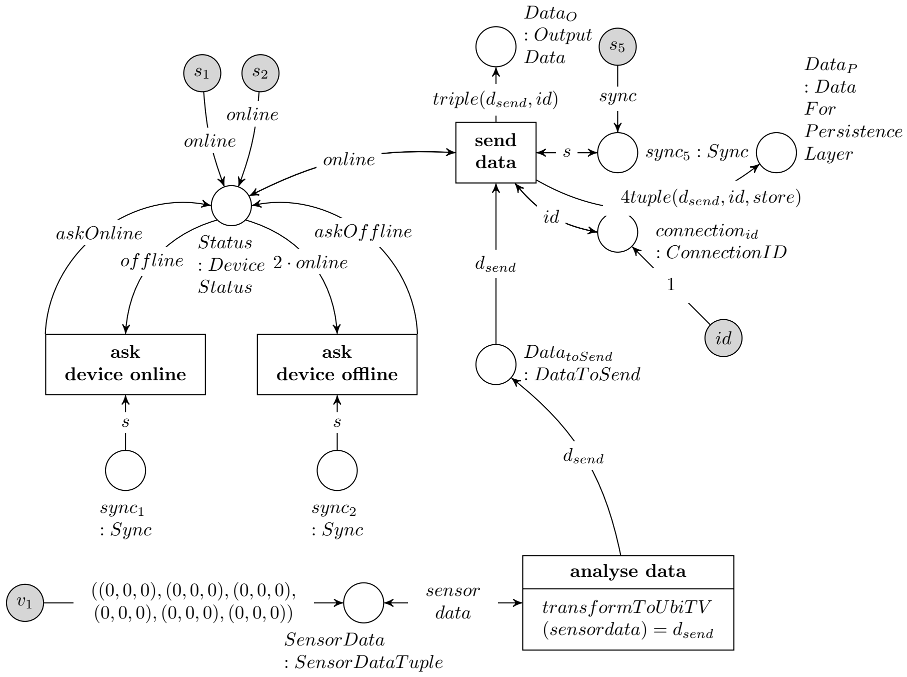


Figure 7.17: Net $indoorPositioningSystem_{online}$

7.1.2.12 Location-Based Screen

The Location-Based Screen is introduced in section 2.2.2.7. In the following the visual modelling of the Location-Based Screen will be presented.

The component representing the Location-Based Screen is modeled as receiving device, therefore the template for a receiving device is expanded by a part which is responsible for processing the received data belonging to the topic $UbiTV$ and sending a corresponding activation signal to both television sets being a part of this device. Each television set is represented by a place $TV_1 : TVState$ resp. $TV_2 : TVState$. TV_1 represents the television set which is located in the lounge area and TV_2 the TV set in the sleeping area. The tokens assigned to these places show the current state of the corresponding television set. The device states can be composed as shown in the table 7.1. Consequently two dimensions describing the device state are available: on/off is a state which will be set by the user in applying an AHOI transformation rule (see 7.1.4.6.1), and the activation state which will be set by this Location-Based Screen device. The on/off state shows if all television sets are switched on or off by the user, this state has the priority. The activation state describes which TV set in the Living Place should show the current tv program, if the user has switched the TV sets on. Consequently, only one TV set is activated, while all TV sets always have the same on/off state and the user is only able to switch all TV sets on or off at the same time.
To infer the right state, the device has to be in online mode. Then, the data token will be fetched from $Data_{received}$ and evaluated in applying the operation $analyseUbiTVdata$. This operation checks the received information and determines which TV set should be activated or deactivated. The decision is made as follows:

- If $first_{dataReceived}(d_r) = north$ then activate TV_1 because the inhabitant is in the lounge area and deactivate TV_2.
- If $first_{dataReceived}(d_r) = south$ then activate TV_2 because the inhabitant is in the sleeping area and deactivate TV_1.
- If $first_{dataReceived}(d_r) = middle$ then deactivate both TV sets, because the inhabitant is out of sight of both TV sets.
- If none of the previous conditions hold, then deactivate both TV sets, because the inhabitant is either in the bathroom or has left the apartment.

According to this decision, the activation state will be changed in firing the transitions *change status of* TV_1 and *change status of* TV_2. The operation $isActiveToken(x) = tt$ forces the transition to remove the token representing the activation state from the place TV_1 resp. TV_2, in order to assign a new token to the place representing the activation state. The token showing the on/off state stays untouched.

On/Off State	**Activation State**	**What is the meaning?**
on	activate	The television set shows the current tv program.
on	deactivate	The screen of the television set is switched off. The sound of the current tv program can be heard.
off	activate	The television set is switched off. Nothing shown and heard.
off	deactivate	The television set is switched off. Nothing shown and heard.

Table 7.1: Possible states of a television set in the model of the Location-Based Screen

7.1.2.12.1 Location-Based Screen in Offline Mode

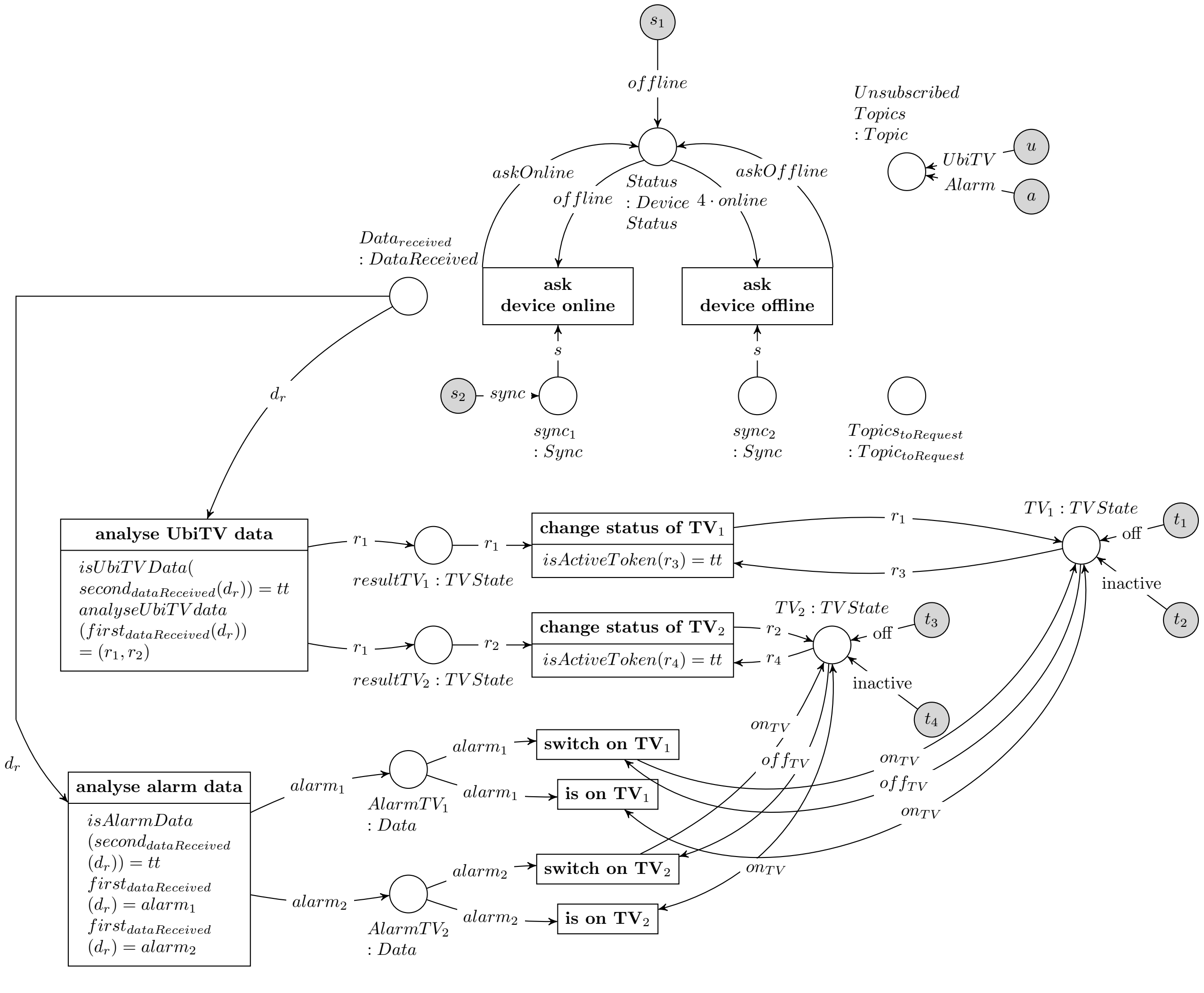


Figure 7.18: Net $locationBasedScreen_{offline}$

7.1.2.12.2 Location-Based Screen in Online Mode

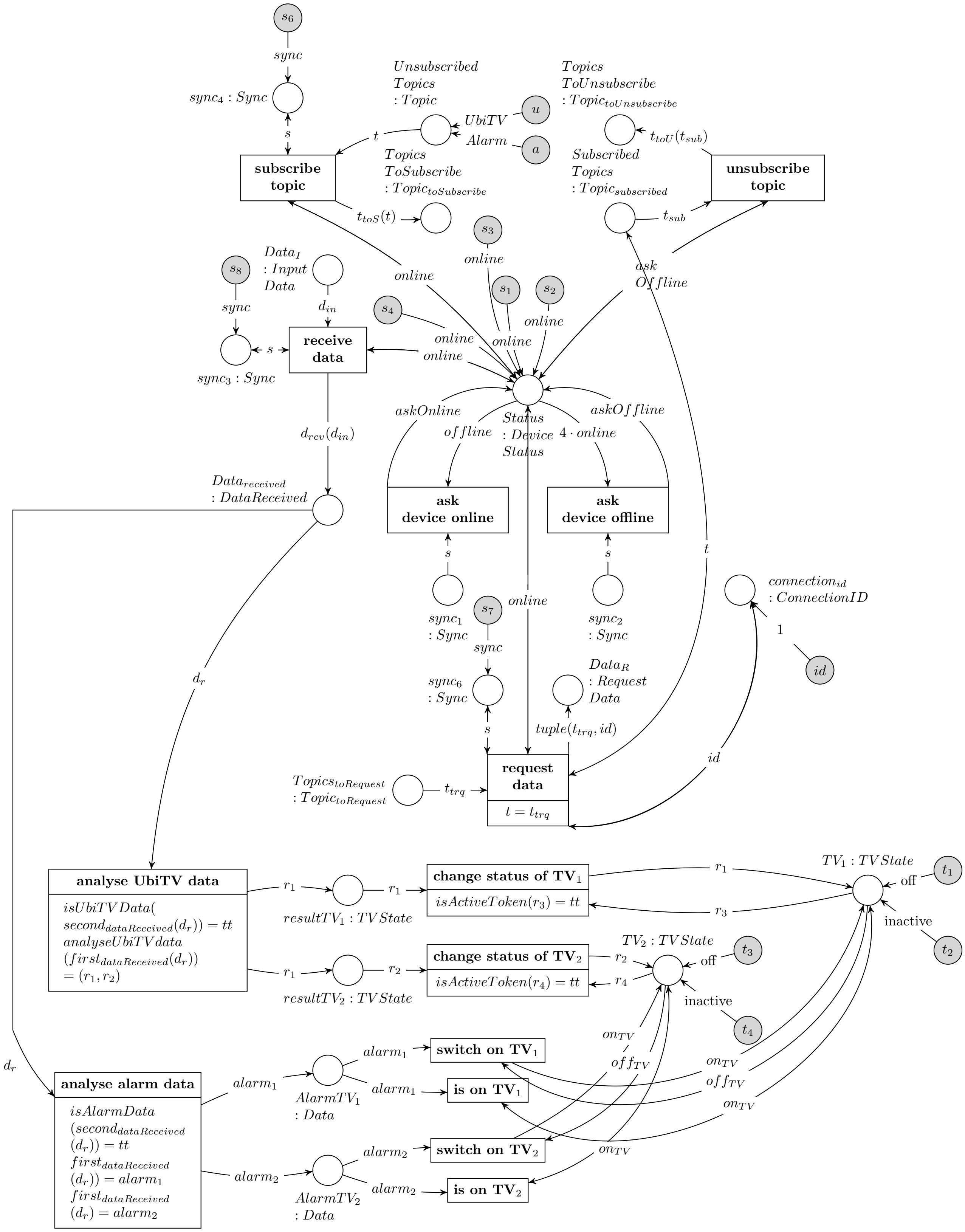


Figure 7.19: Net $locationBasedScreen_{online}$

7.1.2.12.3 Revisiting the Requirements Towards the Model of the Location-Based Screen

In section 4.2.2.11 requirements are set up, which should be fulfilled by the model of the Location-Based Screen. The following section revises those requirements.

$\mathbf{r_{screen_1}}$: Each place $TV_1 : TVState$ and $TV_2 : TVState$ represent one of both TV sets in the Living Place apartment. TV_1 represents the TV set in the lounge area, whereas TV_2 stands for the TV set in the sleeping area.
Thus, the requirement r_{screen_1} is met.

$\mathbf{r_{screen_2}}$: The tokens *UbiTV* and *Alarm* initially assigned to the place $UnsubscribedTopics : Topic$ represent both topics to which the Location-Based Screen is able to subscribe resp. unsubscribe. In firing the transitions *subscribe topic* resp. *unsubscribe topic*, the object net is able to initiate the subscription resp. unsubscription of a topic. Consequently, the requirement r_{screen_2} is met.

$\mathbf{r_{screen_3}}$: After subscribing to both topics, the object net $locationBasedScreen_{online}$ is able to receive data regarding both topics. The transition *analyse UbiTV data* collects all data token belonging to the topic *UbiTV* and processes them.
The transition *analyse alarm data* fires, when a token belonging to the topic *Alarm* is assigned to the place $Data_{received} : DataReceived$. Then, it is able to process this token.
The processing of different data belonging to different topics is done seperately, thus the requirement r_{screen_3} is fulfilled.

$\mathbf{r_{screen_4}}$: The transition *analyse alarm data* is able to fire if a token of the topic *Alarm* is available on the place $Data_{received} : DataReceived$ and processes this token with the help of the transition conditions as follows:

- $isAlarmData(second_{dataReceived}(d_r)) = tt$
  This condition checks for tokens belonging to the topic *Alarm*.
- $first_{dataReceived}(d_r)) = alarm_1$ and $first_{dataReceived}(d_r) = alarm_2$
  Both conditions evaluates the first component of the token which is a triple. This token will be casted to the correct type : *Data* and assigned to the places $AlarmTV_1 : Data$ resp. $AlarmTV_2 : Data$. Afterwards either the transition *switch on* TV_i or *is on* TV_i is able to fire. They are inverse to each other in order to either replace the token off assigned to the place TV_i or replace the token *on* $(i \in \{1, 2\})$.

Because of the model of the Location-Based Screen evaluates the on/off state for each TV set identical, they are always switched on at the same time, resp. switched off at the same time.
Hence, the requirement r_{screen_4} is fulfilled.

$\mathbf{r_{screen_5}}$: The transition condition $analyseUbiTVdata(first_{dataReceived}(d_r)) = (r_1, r_2)$ evaluates the data according to the following conditions (as presented in the formal description of this operation in 11.1):

- (active, inactive)
  if $x = north$
- (inactive, active)
  if $x = south$
- (inactive, inactive)
  if $x = middle$
- (inactive, inactive)
  else

The resulting tuple describes the new activation state, which will be assigned for each TV state. In firing the transitions *change status of* TV_i $(i \in \{1, 2\})$ and checking the internal transition conditions $isActiveToken(x) = tt$, which is responsible that only the token with *active* or *inactive* will be used, the activation state is able to be changed for each TV set.
As a result, the requirement r_{screen_5} is fulfilled.

$\mathbf{r_{screen_6}}$: In table 7.1, the possible states of both TV sets, which are regarded in the model of the Location-Based Screen, is illustrated. The states are assigned as tokens to the places $TV_1 : TVState$ and $TV_2 : TVState$, and follow the rules given in this table.
Hence, the model fulfils the requirement r_{screen_6}.

$\mathbf{r_{screen_7}}$: The model of the Location-Based Screen is a receiver, therefore the requirement r_{screen_7} is met.

$\mathbf{r_{screen_8}}$: The inverse AHOI rules $SwitchOffDisplay_{user}$ and $SwitchOnDisplay_{user}$ are provided by the model of Living Place system to enable the inhabitant to switch off resp. on both TV sets in the apartment. Additionally, each AHOI rule has a corresponding AHLI rule $ruleSwitchOffDisplay_{display}$ resp. $ruleSwitchOnDisplay_{display}$ presented in section 7.1.3.5.7. The AHOI rules are illustrated in explicit in 7.1.4.10.
As a result, the requirement r_{screen_8} is fulfilled.

To summarise, the model of the Location-Based Screen fulfils all requirements set up in section 4.2.2.11.

7.1.2.13 Ambient Light

The Living Place project will be equipped with an Ambient Light, which will be presented in the following and has the following states:

- **bright**: Which means, the light is switched on and will shine bright, which is the default mode of the lamp.
- **dimmed**: The light is switched on in deimmed mode. This happens when the Alarm Clock 2.0 signals, that the user should be woken up. Then the Ambient Light will be switched on by the Living Place system in dimmed mode.
- **off**: The light is switched off.

The model of the light is described by a receiving device, whose individual part represents the state, if the associated lamp in the apartment is turned on resp. turned off. The receiving device is subscribed to the topic *Light*, therefore it receives all messages sent to this topic. An example for an information belonging to this topic is: $(bright, Light, 123)$ or $(off, Light, 861)$. After receiving messages, the device interpretes the message in firing the transition *analyse data and change light* and executing the operation $analyseLightData(d_r) = l_1$. This means, the message contains a triple representing the data and the operation takes the first element of the triple and checks its state. Afterwards this state will be assigned as token to the place $Light : LightState$ representing a certain lamp with its own state (lamp is switched on or lamp is turned off). Additionally, if the topic of the incoming message is *Alarm*, the token *dimmed* will be returned in order to change the state of the light into dimmed mode, when an alarm is provoked within the Living Place system.
Only one token is able to be assigned to the place *Light* at one time, because the token assigned to this place will be swaped with the new one generated by the operation *analyseLightData*.
Initially the light is switched off which is Displayed in assigning the token off to the place $Light : LightState$.

The visual models of the device nets $light_{offline}$ and $light_{online}$ are illustrated in figure 7.20 and figure 7.21.

7.1.2.13.1 Revisiting the Requirements Towards the Model of the Ambient Light

In this section, the requirements set up for the model of the Ambient Light will be revised regarding the fulfilment of each requirement by the object net $light_{online}$ (resp. $light_{offline}$ for the Ambient Light device in offline mode. For processing messages, the device needs to be in online mode).

$\mathbf{r_{light_1}}$: One token is assigned to the place $Light : LightState$. Both, the place and the token assigned to it, represent the light in the Living Place apartment. The token signals the current state of the light. The following states of the light are possible:

- off : The light is switched off.
- $bright$: The light is switched on and shines brightly.
- $dimmed$: The light is switched on, but it is dimmed.

Because the model contains the representation of the current status of the light, the requirement r_{light_1} is fulfilled.

$\mathbf{r_{light_2}}$: Initially, the tokens $Light$ and $Alarm$ are assigned to the place $UnsubscribedTopics : Topic$ representing the topics, to which the Ambient Light is able to get subscribed resp. unsubscribed. Hence, the requirement r_{light_2} is met.

$\mathbf{r_{light_3}}$: The transition *analyse data and change light* checks its transition condition when firing. The condition $analyseLightData(d_r) = l_1$ is evaluated according to the following conditions (see 11.1:

- $dimmed$
  if $second_{dataReceived_A}(x) = Alarm$
- $first_{dataReceived_A}(x)$
  else

That means, every data belonging to the topic $Alarm$ ensures that the light is switched on in dimmed mode. If a token belonging to the topic $Light$ is received, it sets the state of the light according to the first component in this data triple.
The conditions set up by the requirement r_{light_3} are modelled, thus this requirement is fulfilled.

$\mathbf{r_{light_4}}$: The object net representing the Ambient Light is modelled as a receiver. Therefore, the requirement r_{light_4} is met.

$\mathbf{r_{light_5}}$: The Living Place system provides the AHOI rules $SwitchOffLight_{user,bright}$, $SwitchOff-Light_{user,dimmed}$ and $SwitchOnLight_{user}$ in order to enable the inhabitant to change the state of the Ambient Light. Each AHOI rule has a corresponding AHLI rule $ruleSwitchOff_{Light,bright}$, $ruleSwitch-Off_{Light,dimmed}$ resp. $ruleSwitchOn_{Light}$ in order to transform the object net $light_{online}$. The AHOI rules are presented in 7.1.4.11, the corresponding AHLI rules in 7.79, 7.80 and 7.1.3.5.9.
Because of the availability of these AHOI and AHLI rules, the requirement r_{light_5} is fulfilled.

Finally, all requirements set for the model of the Ambient Light are fulfilled by the model.

7.1.2.13.2 Ambient Light in Offline Mode

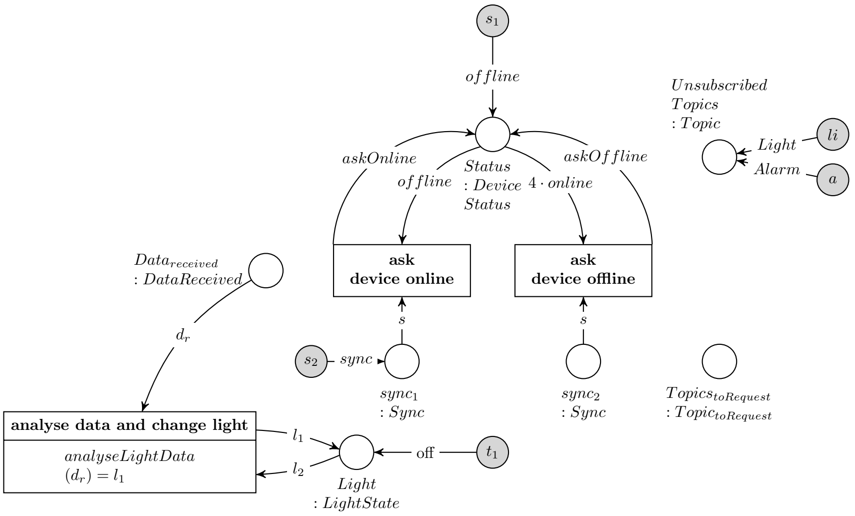


Figure 7.20: Net $light_{offline}$

7.1.2.13.3 Ambient Light in Online Mode

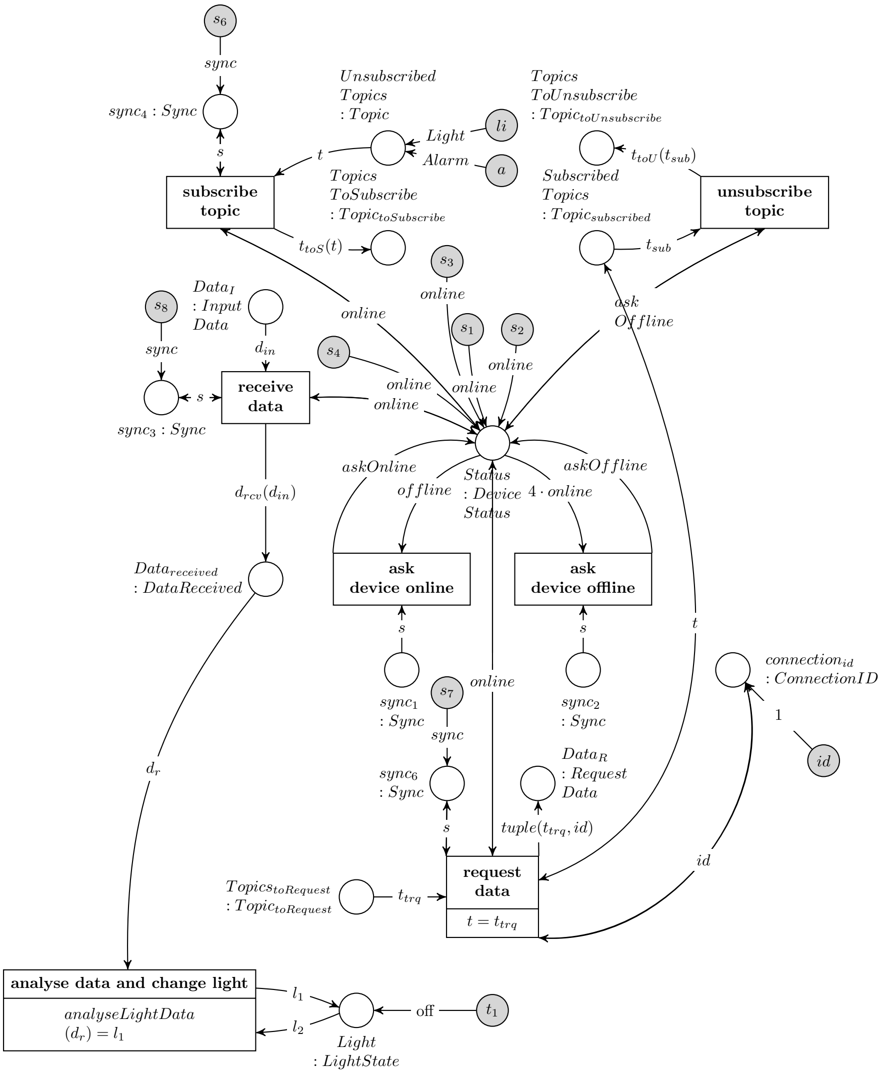


Figure 7.21: Net $light_{online}$

7.1.2.14 Weather Information System

The Weather Information System in the Living Place system will be modelled as a sending device which generates weather data from a given set of weather data. The given set of weather data $WeatherData$ is defined in the Σ_{OL}-algebra A_{OL} of the Object Level (see 11.1). Currently it contains:

- sun,
- rain,
- storm,
- snow,
- wind,
- hail

The weather data can be generated in firing the transition *create weather data*. Then, the varibale w will be assigned with one element from the collection $Weather$. Additionally, a timestamp needs to be given defining the beginning of this weather condition in assigning the variable t. Then, this specivic data will be converted into the global type *Data* in evaluating the term *weatherToData*(t, w) and afterwards, extended by the specific topic *Weather*. The resulting tuple matches the type *DataToSend* and will be assigned as token to the place $Data_{toSend}$: *DataToSend*.

7.1.2.14.1 Revisiting the Requirements Towards the Model of the Weather Information System

In 4.2.2.9 the requirements regarding the Weather Information System were presented. In this section, each requirement will be revised regarding its fulfilment by the model of the device.

$\mathbf{r_{weather_1}}$: When firing the transition *create weather data*, the variable *time* representing the timestamp of the begin of this weather and the variable w representing the type of weather forecast need to be assigned with an element out of the corresponding carrier set. *time* : $Timestamp$ and $w : Weather$. The operation *weatherToData*$(time, w)$ converts the input values to the general format : *Data*. Afterwards the tuple containing the newly generated weather information and the topic *Weather* will be assigned to the place $Data_{toSend}$: *DataToSend*. The currently available set of weather data is enumerated in last section 7.1.2.14.
The firing of the transition simulates the getting of a new weather forecast and therefore represents the receiving of data over the internet. Hence, the requirement $r_{weather_1}$ is met.

$\mathbf{r_{weather_2}}$: The resulting weather information generated after firing the transition *create weather data*, is a tuple including the weather forecast and a timestamp referring to the begin of this event.
As a result, the requirement $r_{weather_2}$ is fulfilled.

$\mathbf{r_{weather_3}}$: In firing the transition *send data*, the device net extends the currently processed token by the specific connection identfifcation of this device. Then, the resulting data will be set to the place $Data_O$: *OutputData*. It satisfies the general data format, therefore the token is now able to get processed the Message Oriented Middleware.
Consequently, the requirement $r_{weather_3}$ will be fulfilled.

$\mathbf{r_{weather_4}}$: The Weather Information System is modeled as a sender, thus this requirement $r_{weather_4}$ is fulfilled.

7.1.2.14.2 Weather Information System in Offline Mode

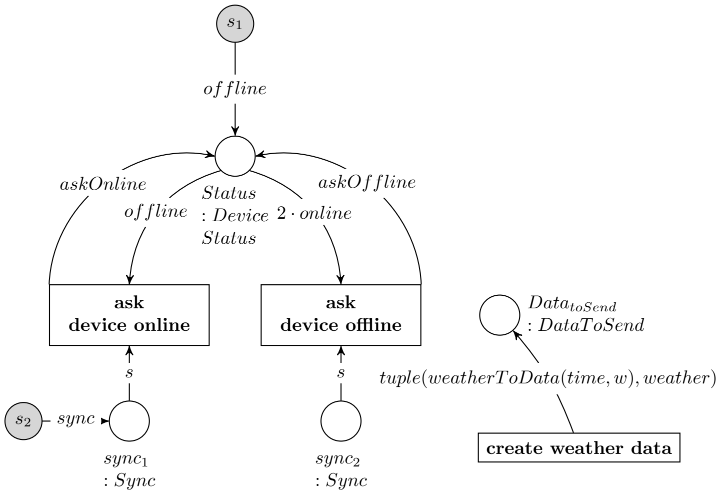


Figure 7.22: Net $weatherinformationystem_{offline}$

7.1.2.14.3 Weather Information System in Online Mode

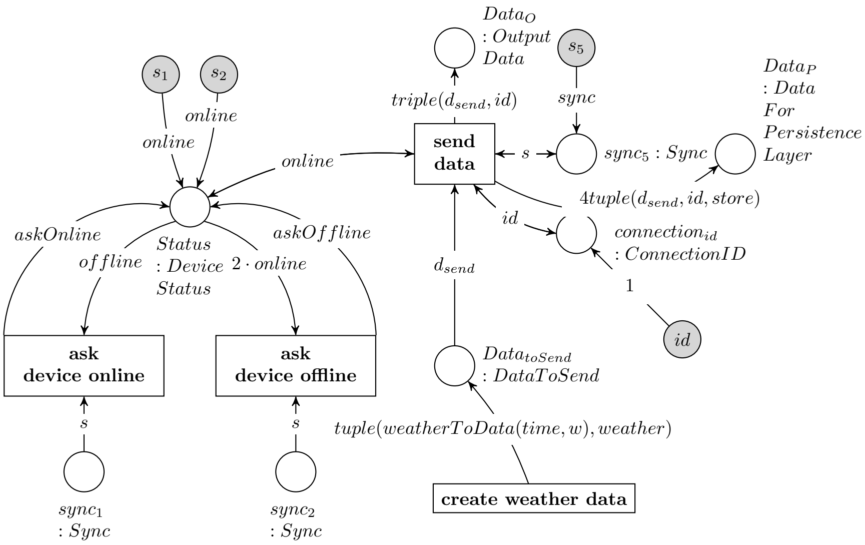


Figure 7.23: Net $weatherinformationystem_{online}$

7.1.2.15 Traffic Service

Similar to the Weather Information System (see 7.1.2.14), the Traffic Servcie is a sending component which generates data from the following set of traffic information, which are represented by elements of the carrier set of the type $TrafficData$ in the Σ_{OL}-algebra A_{OL}, see 11.1:

- jam,
- accident,
- free,
- roadwork

The traffic data will be produced in firing the transition *create traffic data*. The resulting data tuple containing the timestamp describing the start of the event and a description of this traffic situation, will be converted to match the type *DataToSend* in applying the operation $trafficToData$. Afterwards, the resulting data can be asssigned as token to the place $Data_{toSend}$ from where the information will be processed further to the message broker.

7.1.2.15.1 Traffic Service in Offline Mode

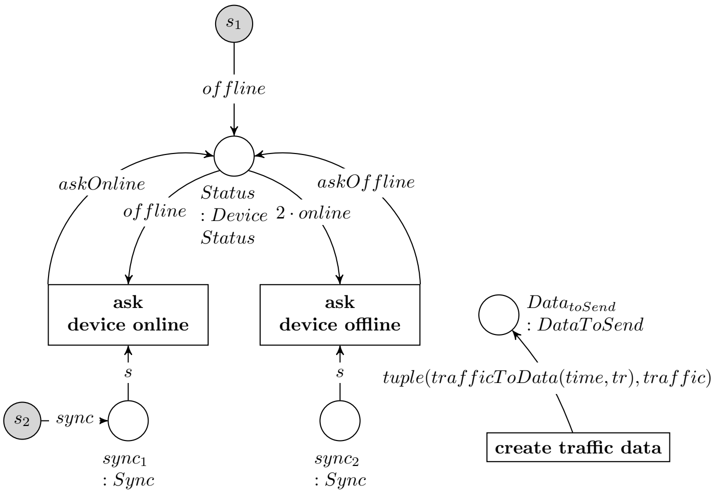


Figure 7.24: Net $trafficService_{offline}$

7.1.2.15.2 Traffic Service in Online Mode

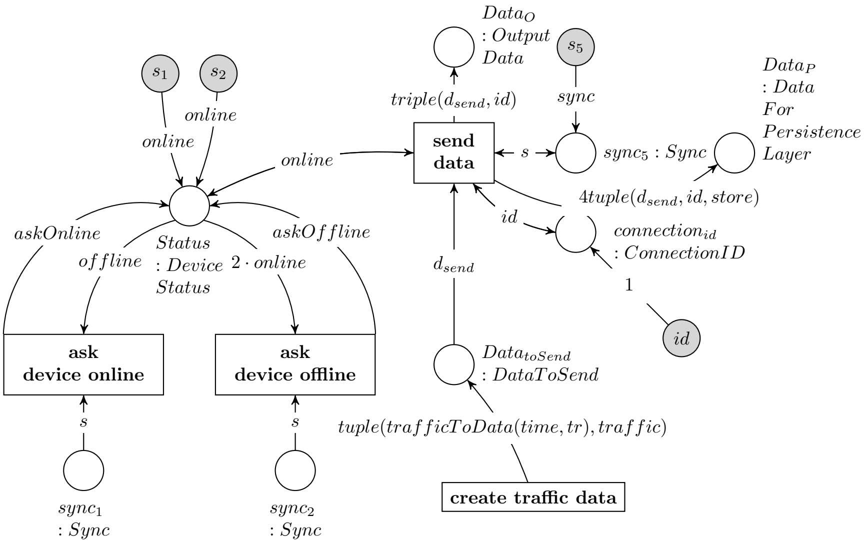


Figure 7.25: Net $trafficService_{online}$

7.1.2.15.3 Revisiting the Requirements Towards the Model of the Traffic Service

In 4.2.2.9 the requirements regarding the Traffic Service were presented. In this section, each requirement will be revised regarding its fulfilment by the model of the device. The requirements regarding the Traffic Service are similar to the ones set up for the Weather Information System. Consequently the fulfilment of each requirement is analog.

$\mathbf{r_{traffic_1}}$: When firing the transition *create traffic data*, the variable *time* representing the timestamp of the begin of this traffic situation and the variable *tr* representing the traffic situation need to be assigned with an element out of the corresponding carrier set. $time : Timestamp$ and $tr : Traffic$. The

operation $trafficToData(t, w)$ converts the input values to the general format : $Data$. Afterwards the tuple containing the newly generated traffic information and the topic *Traffic* will be assigned to the place $Data_{toSend}$: $DataToSend$. The currently available set of traffic data is enumerated in last section 7.1.2.15.
The firing of the transition simulates the getting of a new traffic forecast and therefore represents the receiving of data over the internet. Hence, the requirement $r_{traffic_1}$ is met.

$\mathbf{r_{traffic_2}}$: The resulting traffic information generated after firing the transition *create traffic data* is a tuple including the traffic forecast and a timestamp referring to the begin of this event.
As a result, the requirement $r_{traffic_2}$ is fulfilled.

$\mathbf{r_{traffic_3}}$: In firing the transition *send data*, the device net extends the currently processed token by the specific connection identfifcation of this device. Then, the resulting data will be set to the place $Data_O$: $OutputData$. It satisfies the general data format, therefore the token is now able to get processed by the Message Oriented Middleware.
Consequently, the requirement $r_{traffic_3}$ will be fulfilled.

$\mathbf{r_{traffic_4}}$: The Traffic Service is modeled as a sender, thus this requirement $r_{traffic_4}$ is fulfilled.

7.1.2.16 Revisiting the Requirement Towards all Devices

The requirement r_{ubio_3} is demanded to the models of all devices (see 4.1.2). In this section, it will be revised regarding its fulfilment by all models presented in this section 7.1.2. This requirement states, that the prevailing context of the Living Place will be taken into account by the communicating devices within their processing steps.

$\mathbf{r_{ubio_3}}$: Each model of a device presented in this section is able to process the prevailing context in a different manner, but each device reacts on the current context resp. on the change of the context. Consequently, the requirement r_{ubio_3} is satisfied by the models of all devices.

7.1.3 System Level

In the following considerations of this chapter, the models of the System Level are defined as $AHOI_{AHLI}$ nets[5], i.e. the model of the internal system behaviour of the Message Oriented Middleware consisting of the model of the internal system behaviour of the **Message Broker**, **Context Interpreter** as well as **Persistence Layer**.

Additionally, the System Level contains a model of the internal system behaviour of the so called **User Operation Processing Unit** that is defined within this chapter.

Figure 7.26 represents the overall model of the System Level in initial mode as one huge $AHOI_{AHLI}$ net containing all these mentioned models of internal system behaviours that are contained in the System Level, so that figure 7.26 gives an overview of all those models that are presented in detail in the coming sections of this chapter.

Section 7.1.3.1 deals with the model of the internal system behaviour of the Message Broker, in section 7.53 the model of the internal system behaviour of the Context Interpreter is defined, section 7.1.3.3 introduces the model of the internal system behaviour of the Persistence Layer and in section 7.1.3.2 the model of the internal system behaviour of the User Operation Processing Unit is elaborated.

[5] For the definition of $AHOI_{AHLI}$ nets see section 6.3.

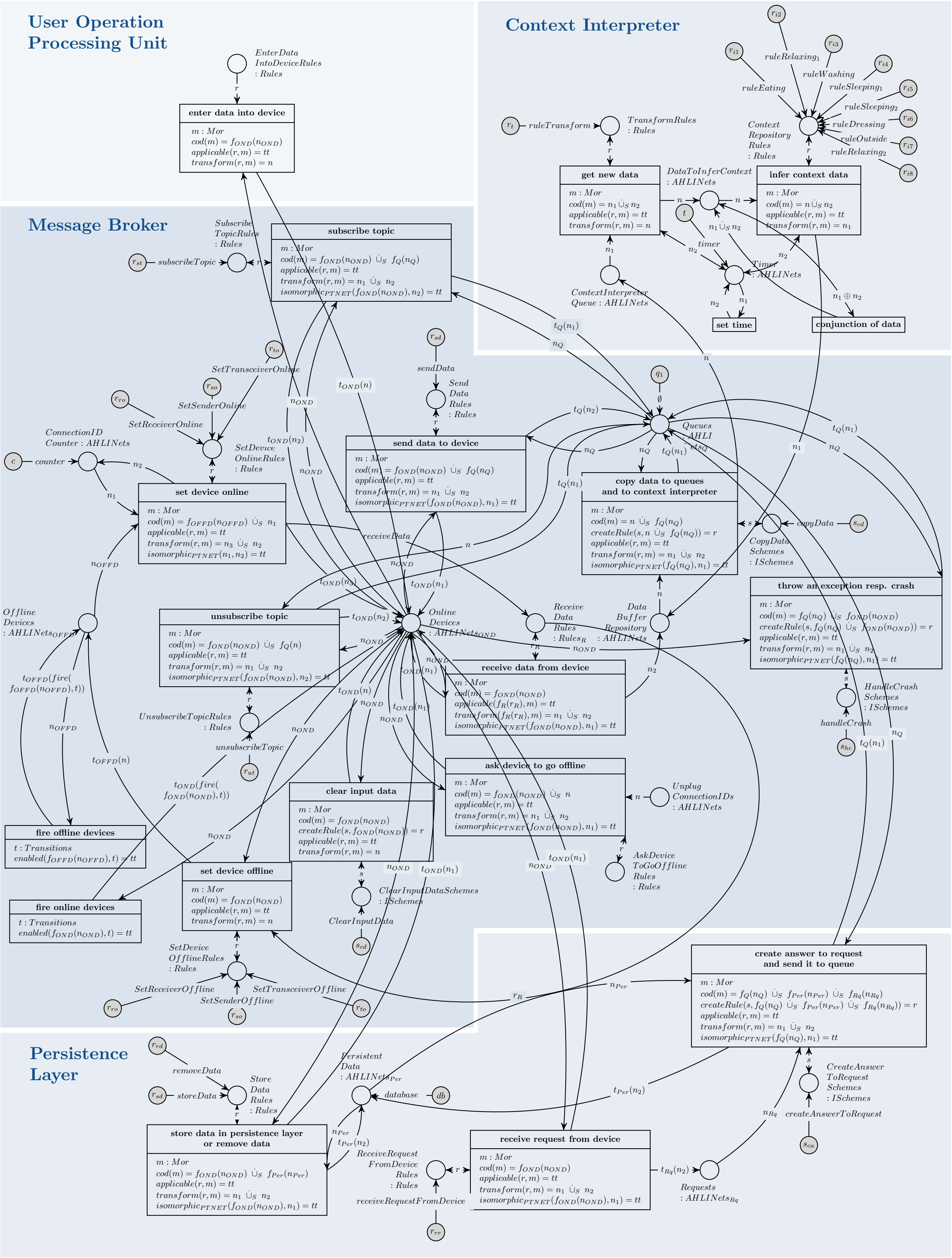

User Operation Processing Unit
Context Interpreter
Message Broker
Persistence Layer
EnterData IntoDeviceRules : Rules
enter data into device
m : Mor
cod(m) = f_OND(n_OND)
applicable(r, m) = tt
transform(r, m) = n
ruleTransform
TransformRules : Rules
get new data
m : Mor
cod(m) = n_1 ∪̇_S n_2
applicable(r, m) = tt
transform(r, m) = n
DataToInferContext : AHLINets
infer context data
m : Mor
cod(m) = n ∪̇_S n_2
applicable(r, m) = tt
transform(r, m) = n_1
ruleRelaxing_1
ruleWashing
ruleEating
ruleSleeping_1
ruleSleeping_2
ruleDressing
ruleOutside
ruleRelaxing_2
Context Repository Rules : Rules
timer
Timer : AHLINets
ContextInterpreter Queue : AHLINets
set time
conjunction of data
n_1 ⊕ n_2
Subscribe TopicRules : Rules
subscribeTopic
subscribe topic
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S f_Q(n_Q)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_2) = tt
SetTransceiverOnline
SetSenderOnline
SetReceiverOnline
sendData
Send Data Rules : Rules
send data to device
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S f_Q(n_Q)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_1) = tt
Queues : AHLINets_Q
ConnectionID Counter : AHLINets
counter
SetDevice OnlineRules : Rules
set device online
m : Mor
cod(m) = f_OFFD(n_OFFD) ∪̇_S n_1
applicable(r, m) = tt
transform(r, m) = n_3 ∪̇_S n_2
isomorphic_PTNET(n_1, n_2) = tt
copy data to queues and to context interpreter
m : Mor
cod(m) = n ∪̇_S f_Q(n_Q)
createRule(s, n ∪̇_S f_Q(n_Q)) = r
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_Q(n_Q), n_1) = tt
copyData
CopyData Schemes : ISchemes
receiveData
Offline Devices : AHLINets_OFFD
Online Devices : AHLINets_OND
unsubscribe topic
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S f_Q(n)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_2) = tt
Receive Data Rules : Rules_R
Data Buffer Repository : AHLINets
throw an exception resp. crash
m : Mor
cod(m) = f_Q(n_Q) ∪̇_S f_OND(n_OND)
createRule(s, f_Q(n_Q) ∪̇_S f_OND(n_OND)) = r
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_Q(n_Q), n_1) = tt
receive data from device
m : Mor
cod(m) = f_OND(n_OND)
applicable(f_R(r_R), m) = tt
transform(f_R(r_R), m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_1) = tt
HandleCrash Schemes : ISchemes
handleCrash
t_OFFD(fire(f_OFFD(n_OFFD), t))
UnsubscribeTopicRules : Rules
unsubscribeTopic
t_OND(fire(f_OND(n_OND), t))
t_OFFD(n)
ask device to go offline
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S n
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_1) = tt
Unplug ConnectionIDs : AHLINets
clear input data
m : Mor
cod(m) = f_OND(n_OND)
createRule(s, f_OND(n_OND)) = r
applicable(r, m) = tt
transform(r, m) = n
fire offline devices
t : Transitions
enabled(f_OFFD(n_OFFD), t) = tt
AskDevice ToGoOffline Rules : Rules
fire online devices
t : Transitions
enabled(f_OND(n_OND), t) = tt
set device offline
m : Mor
cod(m) = f_OND(n_OND)
applicable(r, m) = tt
transform(r, m) = n
ClearInputDataSchemes : ISchemes
ClearInputData
SetDevice OfflineRules : Rules
SetReceiverOffline
SetSenderOffline
SetTransceiverOffline
create answer to request and send it to queue
m : Mor
cod(m) = f_Q(n_Q) ∪̇_S f_Per(n_Per) ∪̇_S f_Rq(n_Rq)
createRule(s, f_Q(n_Q) ∪̇_S f_Per(n_Per) ∪̇_S f_Rq(n_Rq)) = r
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_Q(n_Q), n_1) = tt
CreateAnswer ToRequest Schemes : ISchemes
createAnswerToRequest
removeData
storeData
Store Data Rules : Rules
Persistent Data : AHLINets_Per
database
store data in persistence layer or remove data
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S f_Per(n_Per)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_1) = tt
ReceiveRequest FromDevice Rules : Rules
receiveRequestFromDevice
receive request from device
m : Mor
cod(m) = f_OND(n_OND)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_1) = tt
Requests : AHLINets_Rq
t_Rq(n_2)
t_Per(n_2)
n_Per
n_Rq
r_R
t_Q(n_1)
t_Q(n_2)
n_Q
t_OND(n)
t_OND(n_1)
t_OND(n_2)
n_OND
n_OFFD
n_1
n_2
n
r
s
t
∅
c
q_1
r_t
r_st
r_to
r_so
r_ro
r_sd
r_ut
r_rd
r_rr
s_cd
s_hc
s_ca
db
r_i1
r_i2
r_i3
r_i4
r_i5
r_i6
r_i7
r_i8

Figure 7.26: Overall Model of the System Level in Initial Mode as $AHOI_{AHLI}$ Net

7.1.3.1 Message Broker

The message broker is the central component of the Message Oriented Middleware. It organises the connecting resp. disconnecting of all devices, it handles the process of subscribing resp. unsubscribing of topics for each device. But its main task is to contol the communication within the Living Place system, since it receives all messages sent by sending devices, processes them in checking the corresponding topic and copies and distributes the message to all receiving devices which are subscribed to the specific topic. Consequently, the message broker provides and handles an indirect connection between all devices in online mode connected with the Living Place system.
The message broker is introduced in detail in section 2.2.1.3.

7.1.3.1.1 Subnets For Fire Transitions

The transitions *fire offline devices* and *fire online devices* enable the object net assigned as tokens to the places $OfflineDevices : AHLINets_{OFFD}$ resp. *OnlineDevices* : $AHLINets_{OND}$ to fire the therein defined transitions.[6]

Consider, that some transitions within the object nets are able to fire in parallel, for that there are several more fire transitions defined regarding to the places $OfflineDevices$ resp. *OnlineDevices*, which enables the parallel firing of transitions of the object nets. These fire transitions are omitted in the visual presentation, however are implicitly formally defined, due to a better overview. An example for a transition which enables the parallel firing of two transitions within an object net assigned as token to *OnlineDevices* is given in image 7.28.

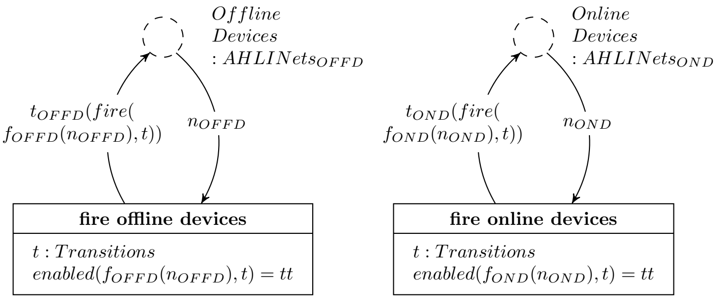

$Offline$ $Devices$ $: AHLINets_{OFFD}$
$t_{OFFD}(fire(f_{OFFD}(n_{OFFD}), t))$
n_{OFFD}
fire offline devices
$t : Transitions$
$enabled(f_{OFFD}(n_{OFFD}), t) = tt$
$Online$ $Devices$ $: AHLINets_{OND}$
$t_{OND}(fire(f_{OND}(n_{OND}), t))$
n_{OND}
fire online devices
$t : Transitions$
$enabled(f_{OND}(n_{OND}), t) = tt$

Figure 7.27: Fire Transitions in System Level

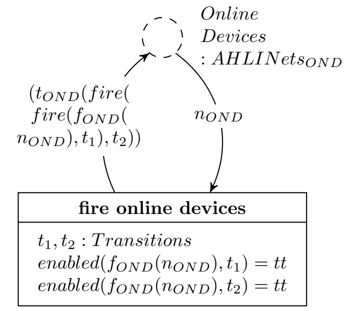

$Online$ $Devices$ $: AHLINets_{OND}$
$(t_{OND}(fire(fire(f_{OND}(n_{OND}), t_1), t_2))$
n_{OND}
fire online devices
$t_1, t_2 : Transitions$
$enabled(f_{OND}(n_{OND}), t_1) = tt$
$enabled(f_{OND}(n_{OND}), t_2) = tt$

Figure 7.28: Fire Transition Enabling a Parallel Firing of Two Transitions in an Object Net

7.1.3.1.2 Subnet for Setting Devices Online

If a device wants to be set into online mode, a token *askOnline* is assigned to the place *Status* : *DeviceStatus* within the device net. Only the message broker is able to set the device into online mode. For that the message broker contains two places $OfflineDevices : AHLINets_{OFFD}$ and *OnlineDevices* : $AHLINets_{OND}$. Both places contain device nets as tokens. $OfflineDevices$ contains all devices which

[6] Compare to [HEM05].

are in offline mode, that means their whole net contains one of the template nets which are described in sections 7.1.2.3, 7.1.2.1 and 7.1.2.2. Additionally the status of the device net will be set to *offline* or *askOnline* in assigning the corresponding token to the place *Status* : *DeviceStatus*. Furthermore when setting a device into online mode, the message broker creates a new connection identification number and assigns a token with this number to the place $connection_{id}$: *ConnectionID* within the device net.

In contrast, the place *OnlineDevices* in the message broker contains all online devices which will use one of the templates of an online device as presented in sections 7.1.2.6, 7.1.2.4 and 7.1.2.5. Their internal status is set to *online* (resp. $askOffline$). Initially no online resp. offline device is available.

For setting a device online, whose net is assigned as a token to the place *OfflineDevices :* $AHLINets_{OFFD}$, the message broker applies one of the three rules which are set as tokens to the place *SetOnline* : *Rules* in firing the transition *set device online*. Each rule sets a certain type of device into online mode: *SetTransceiverOnline* sets a transceiver into online mode, *SetReceiverOnline* a receiver and *SetSenderOnline* a sender. All three rules excludes each other, so that no conflicts are able to occur in applying the wrong rule to a device.

The transition takes a token from the place $OfflineDevices$, applies a rule to the token and puts the transformed token to the place *OnlineDevices* and additionally puts a predefined token *receiveData* to the place *ReceiveDateRules* : $Rules_R$. In addition, in setting a device into online mode, its net gets an individual connection identification which will be assigned as token to the place $connection_{id}$: *ConnectionID* within the device net. The global counter, which represents an unique connection identification for each device, is provided by the net *counter* which is assigned to the place $ConnectionIDCounter$: $AHLINets$. Everytime a device is set into online mode, it gets the number which is stored within this net by assigning the token *id* to *Counter* : $ConnectionID_{messageBroker}$ in setting a token with this number to the place $connection_{id}$ within the device net. After that the global counter is increased by one so that each device gets another connection identification. The rule *receiveData* is shown in figure 7.33.

If a place is painted as dashed, it means, that more elements are connected with this place as illustrated in the current subnet.

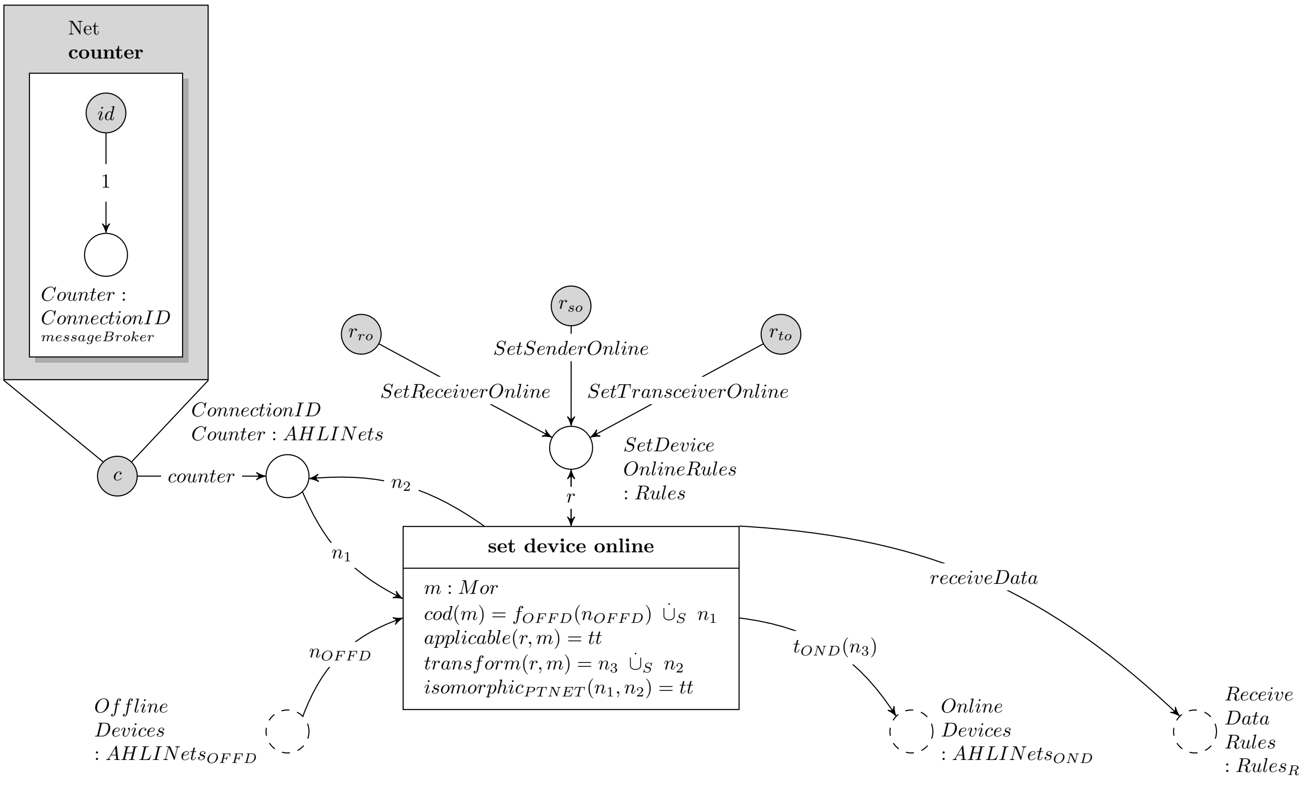


Figure 7.29: Subnet $setDeviceOnline_{messageBroker}$

Rule *SetSenderOnline* for Setting a Sender Into Online Mode

The rule *SetSenderOnline* sets a sender into online mode in adding all the missing places, transitions, tokens and assigning a unique connection identification to the place *ConnectionID*. The NAC prevents the rule to be applied to transceivers, therefore it only can be applied to a sender.

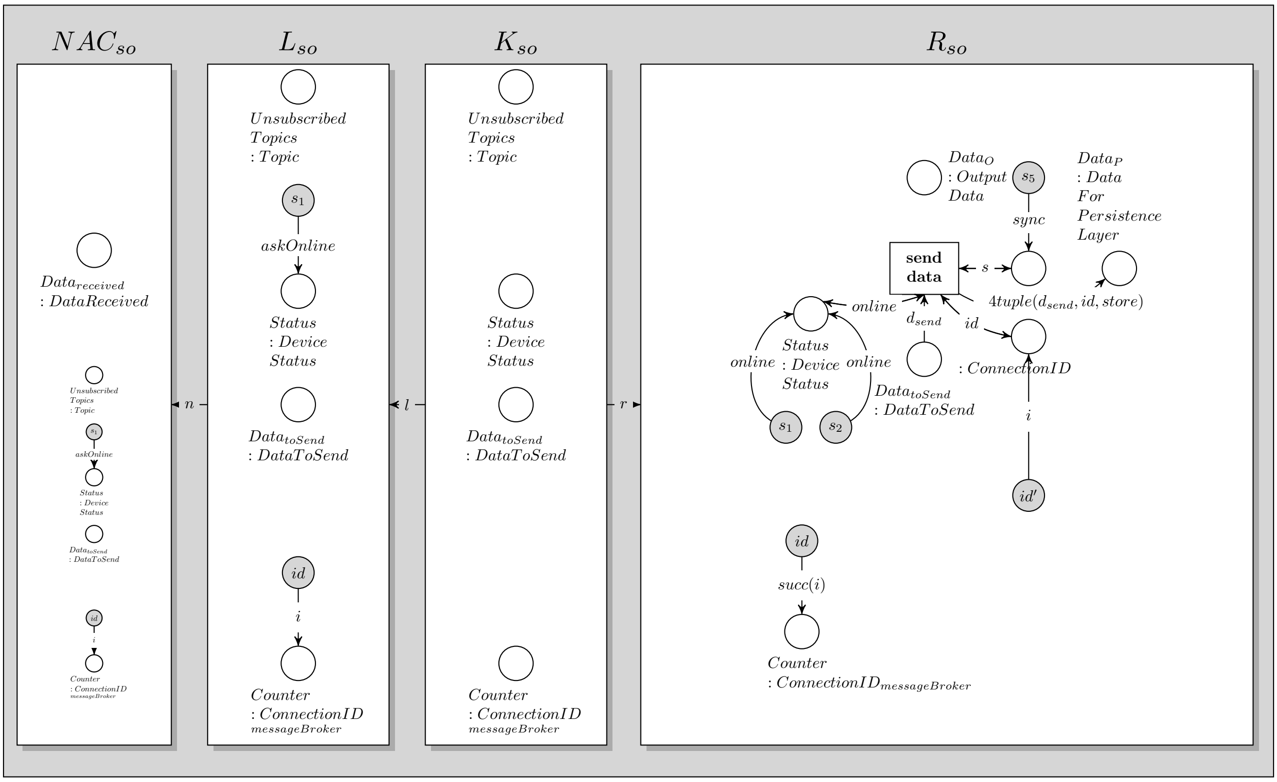

NAC_{so}
L_{so}
K_{so}
R_{so}
$Data_{received}$: DataReceived
Unsubscribed Topics : Topic
s_1
askOnline
Status : Device Status
$Data_{toSend}$: DataToSend
id
i
Counter : ConnectionID messageBroker
n
l
r
$Data_O$: Output Data
s_5
sync
$Data_P$: Data For Persistence Layer
send data
s
$4tuple(d_{send}, id, store)$
online
d_{send}
id
: ConnectionID
online
online
s_1
s_2
i
id'
id
succ(i)
Counter : $ConnectionID_{messageBroker}$

Figure 7.30: Rule *SetSenderOnline*

Rule *SetTransceiverOnline* for Setting a Transceiver Into Online Mode

The rule *SetTransceiverOnline* sets a transceiver online in adding all the missing places, transitions and tokens. This rule can only be applied to tokens representing a transceiver in offline mode.

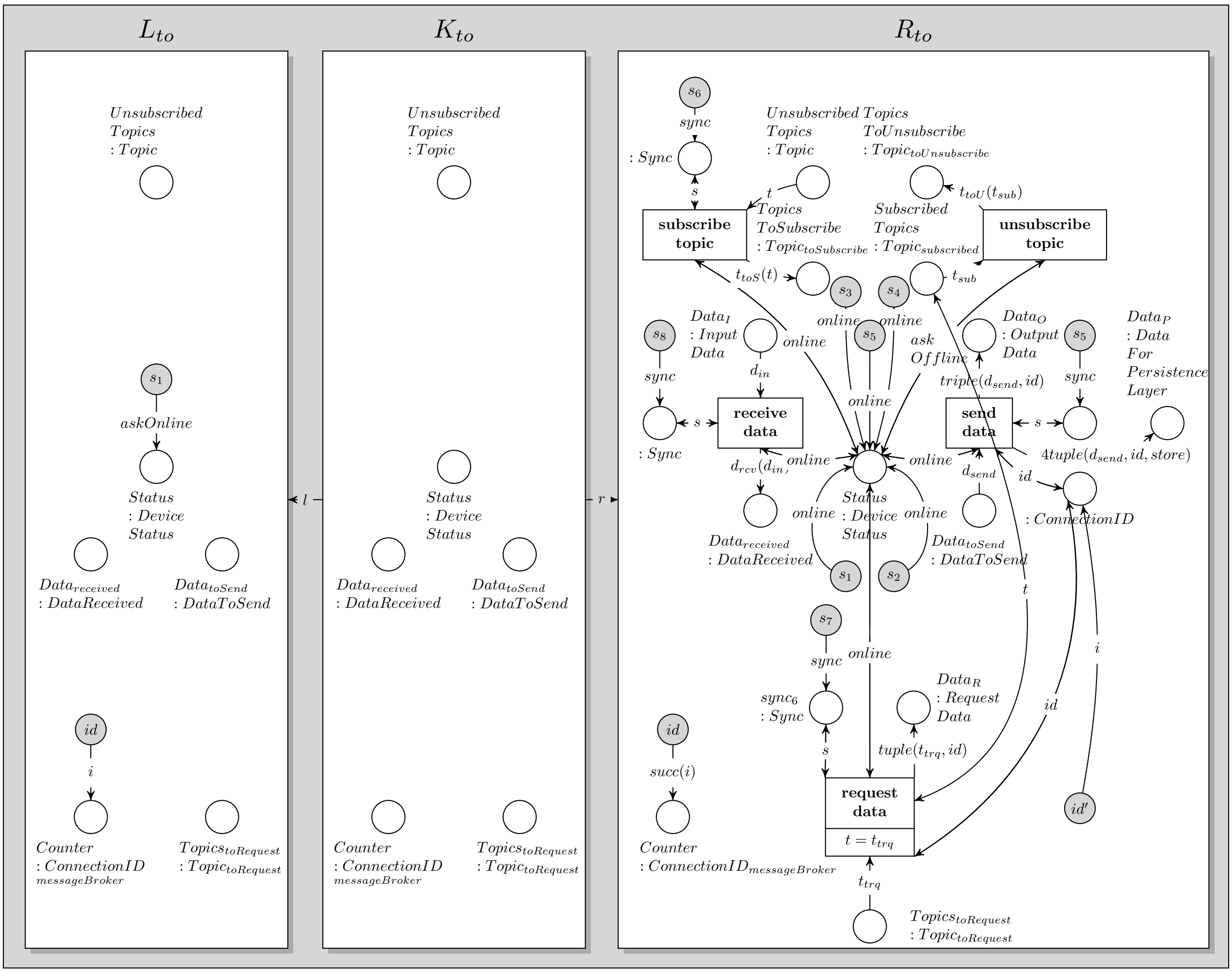


Figure 7.31: Rule *SetTransceiverOnline*

Rule *SetReceiverOnline* for Setting a Receiver Into Online Mode

In applying the rule *SetReceiverOnline*, a receiver will be set into online mode in adding all the missing places, transitions and tokens. The NAC prevents the rule to be applied to transceivers, therefore it only can be applied to a receiver.

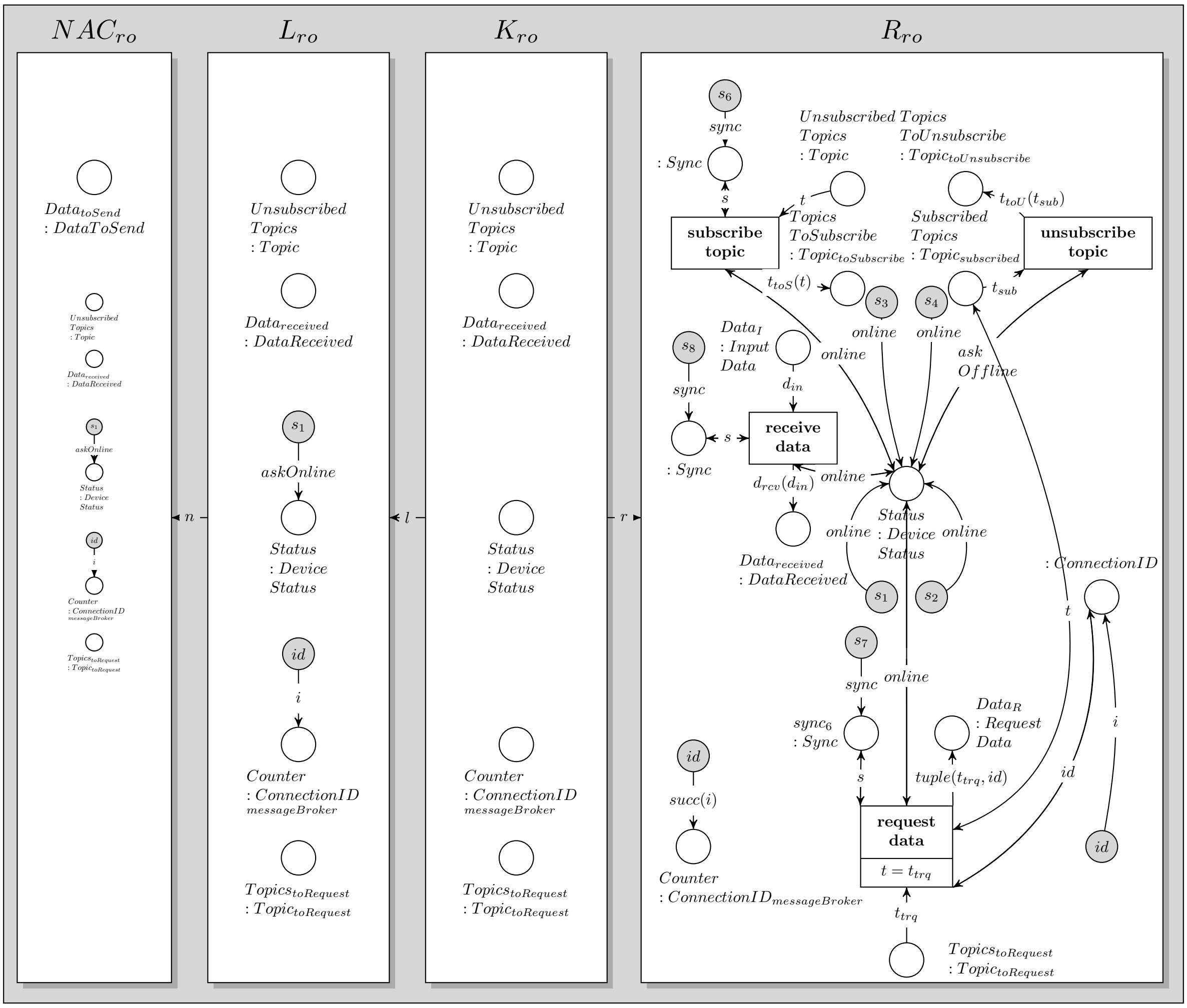


Figure 7.32: Rule *SetReceiverOnline*

7.1.3.1.3 Subnet for Receiving Data From The Devices

Each sending device, i.e. transceiver and sender, is able to create new messages for the Message Oriented Middleware. If a device wants to send a new message, a message token will be put on its place $Data_O$: *OutputData*. The token is a tuple with the following pieces of information:

- An information,
- the name of the topic where the new information belongs to and
- the own device id.

The message broker collects all tokens from the places $Data_O$ belonging to the device nets assigned to the place *OnlineDevices* : $AHLINets_{OND}$ of the message broker. In firing the transition *receive data from device* the rule *receive data* will be applied to a device net from *OnlineDevices* and creates a new net *dataBuffer* which consists of a place : *GlobalData*. Additionally it copies the message token from $Data_O$ of the device net to : *GlobalData* of the new net. After applying the rule, both nets will be split and the device will be put back to the place *OnlineDevices* and the new net will be sent to the place *DataBufferRepository* : *AHLINets*.

The place $DataBufferRepository$ represents a temporary buffer containing all new messages waiting for being processed by the message broker.

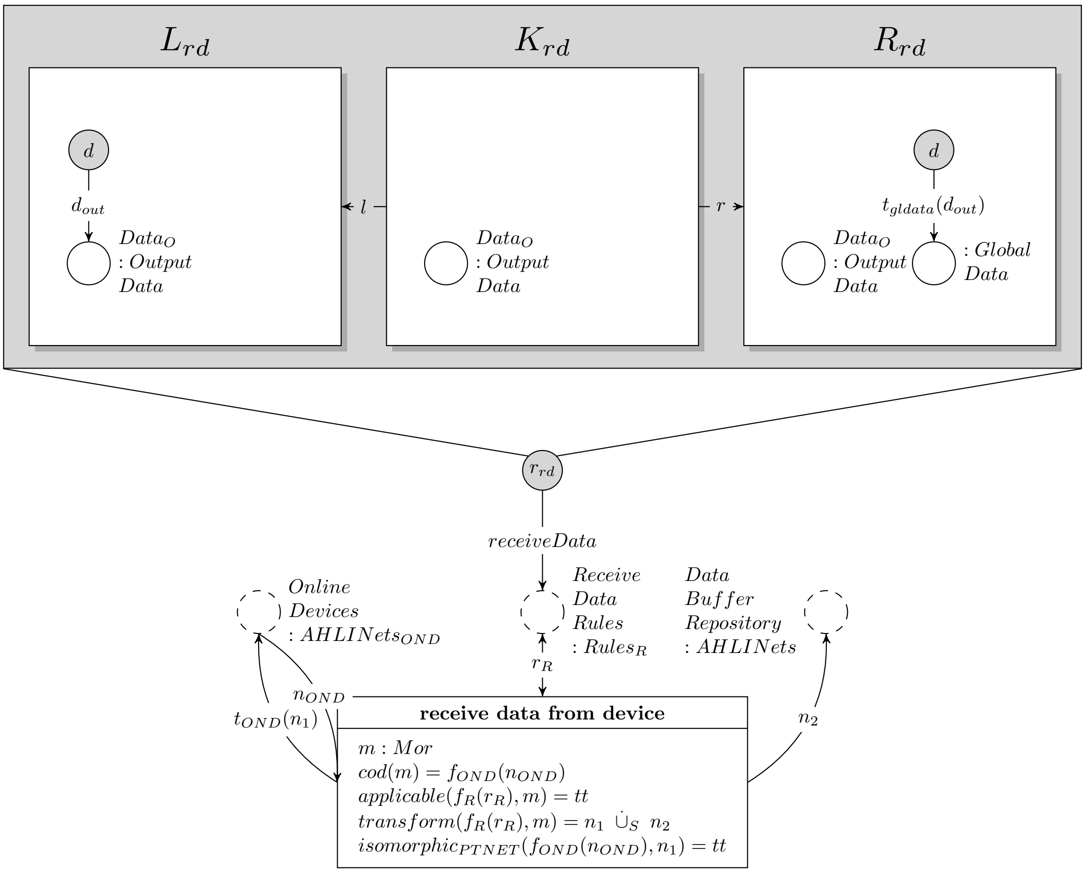

L_{rd}
K_{rd}
R_{rd}
d
d_{out}
$Data_O$: $Output$ $Data$
l
$Data_O$: $Output$ $Data$
r
d
$t_{gldata}(d_{out})$
$Data_O$: $Output$ $Data$
: $Global$ $Data$
r_{rd}
$receiveData$
$Online$ $Devices$: $AHLINets_{OND}$
$Receive$ $Data$ $Rules$: $Rules_R$
$Data$ $Buffer$ $Repository$: $AHLINets$
r_R
n_{OND}
$t_{OND}(n_1)$
n_2
receive data from device
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(f_R(r_R), m) = tt$
$transform(f_R(r_R), m) = n_1 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$

Figure 7.33: Subnet $receiveDataFromDevice_{messageBroker}$

7.1.3.1.4 Subnet for Distribution of Data

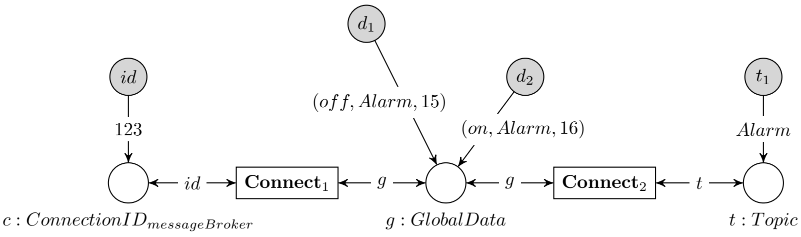


Figure 7.34: Example for a net on the place $Queues$

Because of the message broker distributing all incoming data to the devices connected to the whole system, it should know which device is subscribed to which topics. This information is deposit in the place $Queues : AHLINets_Q$. An example for a net which is situated on the place $Queues$ is illustrated in the image 7.34.

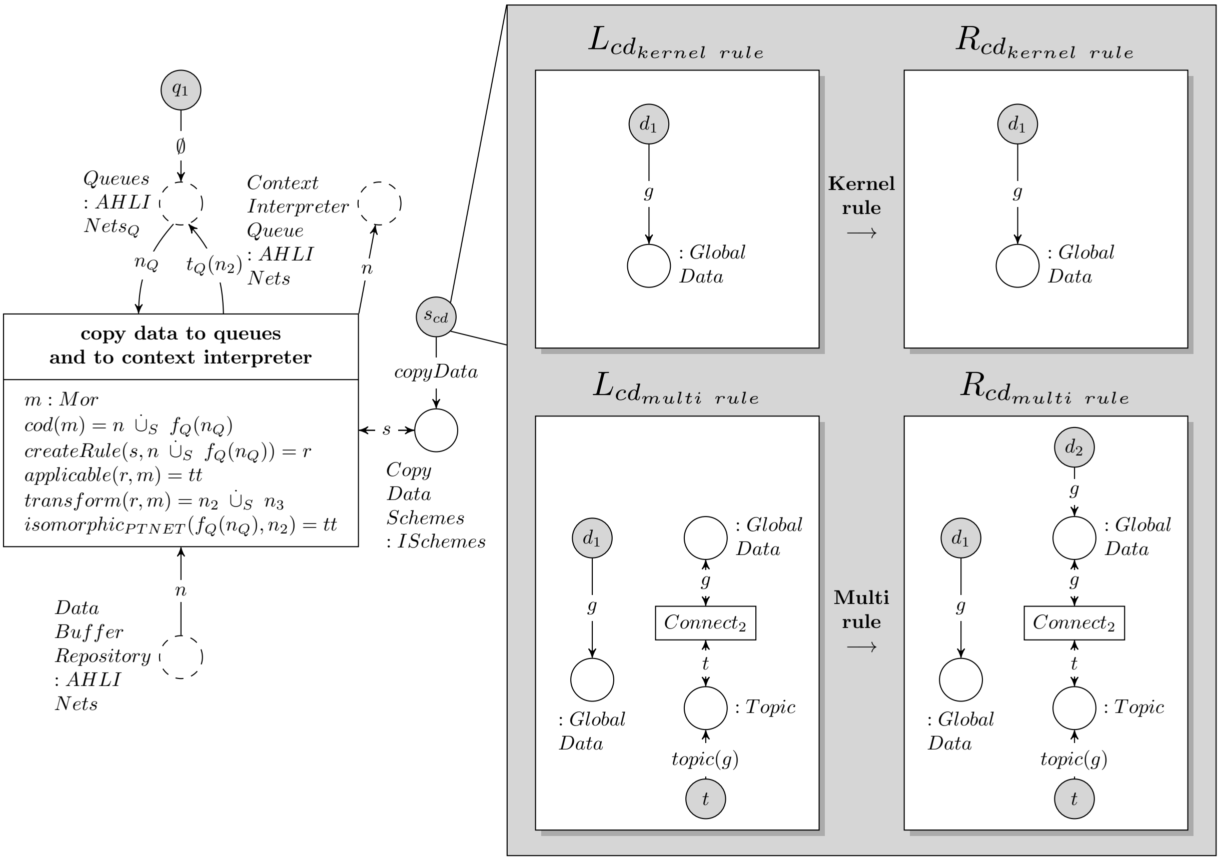
The token assigned to the place $Queues$ can contain more disjoint subnets as described in 7.34. Each subnet contains the information, which device is subscribed to which topic. Because of each subnet creating one connection between the device identification of the individual device and one topic, more than one subnet can exist for each device.

Between the places $: ConnectionID_{messageBroker}$ and $: Topic$ the place $: GlobalData$ exist to which current messages, which are adressed to the specific device and belong to the certain topic, are assigned.

After the message broker has received data from one or more connected devices in online mode, the data will be copied from the temporary data buffer to the place $ContextInterpreterQueue : AHLINets$

which represents the input data queue of the context interpreter (see image 7.53. Furthermore, the information will be copied to each place : *GlobalData* which is connected to the suitable topic in the net which is assigned to the place *Queues*.
When the transition *copy data to queues and to database* fires, the net from the place *Queues*, a token from the place $DataBufferRepository$: $AHLINets$ will be transformed in applying the amalgamated rule generated out of the interaction scheme *CopyData* to the input nets. The output of the transformation will be a modified queue net and a token with the new data which will be placed to $ContextInterpreterQueue$.

The initial marking of this subnet is, that a token is assigned to the place *Queues* which initially contains an empty net, which means that no device is subscribed to any topic. A token including the interaction scheme *copyData* is assigned to the place *CopyDataSchemes* : *ISchemes*.



q_1
$\emptyset$
Queues : AHLI Nets$_Q$
n_Q
$t_Q(n_2)$
Context Interpreter Queue : AHLI Nets
n
copy data to queues and to context interpreter
$m : Mor$
$cod(m) = n \dot{\cup}_S \; f_Q(n_Q)$
$createRule(s, n \dot{\cup}_S \; f_Q(n_Q)) = r$
$applicable(r, m) = tt$
$transform(r, m) = n_2 \dot{\cup}_S \; n_3$
$isomorphic_{PTNET}(f_Q(n_Q), n_2) = tt$
n
Data Buffer Repository : AHLI Nets
s_{cd}
copyData
s
Copy Data Schemes : ISchemes
$L_{cd_{kernel\ rule}}$
$R_{cd_{kernel\ rule}}$
d_1
g
: Global Data
Kernel rule $\longrightarrow$
$L_{cd_{multi\ rule}}$
$R_{cd_{multi\ rule}}$
d_1
d_2
g
: Global Data
$Connect_2$
t
: Topic
topic(g)
t
Multi rule $\longrightarrow$

Figure 7.35: Subnet $copyDataToQueuesAndToContextInterpreter_{messageBroker}$

7.1.3.1.5 Subnet for Sending Data to The Devices

After distributing data to the queues of each subscribed device, the message broker sends the data to the device according to the net assigned as token to the place *Queues* : $AHLINets_Q$. The transition *send data to device* takes a device from *OnlineDevices* : $AHLINets_{OND}$ and the net from *Queues* and applies the rule *sendData* to deploy new data to the device. The rule moves the data token which was assigned to : *GlobalData* from the queue to the input data buffer of the device $Data_I$: *InputData*. Afterwards the modified device net is put back to its place *OnlineDevices* and the modified queue to its corresponding place *Queues*.
In the initial marking, no online device is available.

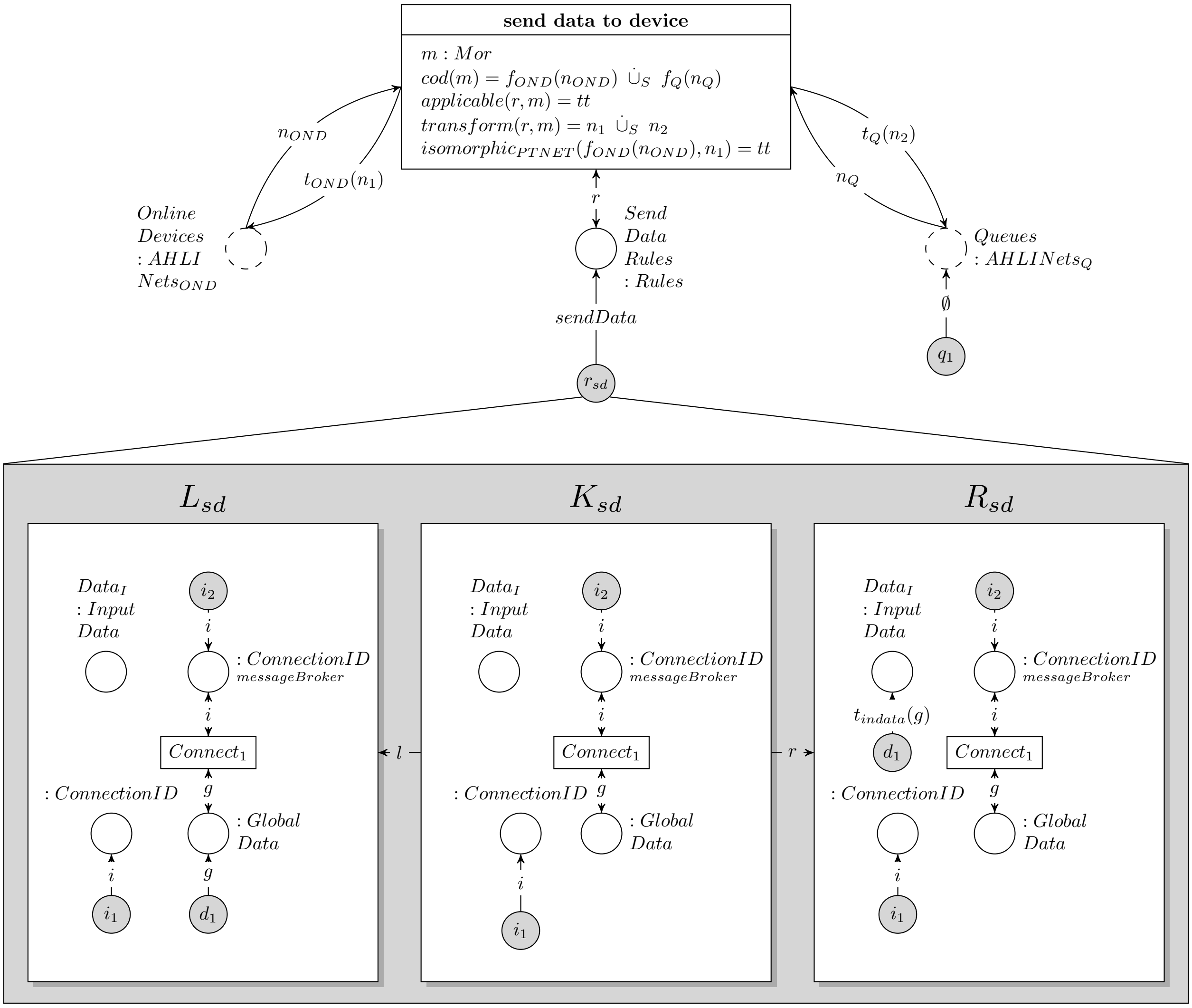


Figure 7.36: Subnet $SendDataToDevice_{messageBroker}$

7.1.3.1.6 Subnet for Preparing a Device to go Into Offline Mode

If a device should go into offline mode, it needs a token set to the place : *Sync* which is connected to the transition *ask device offline*. Only afterwards the device is able to change its status from *online* to $askOffline$ because of the transition *ask device offline* is enabled and can be fired to set a token $askOffline$ to the place *Status* : *DeviceStatus*.
The message broker is able to assign a token *sync* to the correct place within the device net in applying a rule $AskGoOffline_i$ ($i \in \{2, 4, 5\}$) to a device net which is assigned as token to the place *OnlineDevices*. The user initiates the deactivation of the device in assigning a token to the place $UnplugConnectionIDs$: $AHLI$ in applying the AHOI rule $UnplugDevice_{user,nConnectionID}$ as described in 7.82. An example of a token with the value *123* set to *UnplugConnectionIDs* is illustrated in the following image. This means, the user wants the device with the connection identification *123* to be unplugged.

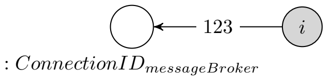


Figure 7.37: Example for the Object Net $nConnectionID_{id}$ With $id = 123$

Afterwards the transition *ask device to go offline* is enabled (see figure 7.38, if a token is assigned to the place *OnlineDevices* and a rule $AskGoOffline_i$ can be applied if the connection identification mentioned within the token in $UnplugConnectionIDs$ is also available in one net assigned as to-

ken to $OnlineDevices$. The rule $AskGoOffline_i$ sets the necessary token $sync$ to the according place within the device net. After performing this task, the modified device net will be put back to the place $OnlineDevices$, but the token taken from $UnplugConnectionIDs$ will be deleted.
The initial marking of this subnet is empty, except from the rule $AskGoOffline_i$ assigned as token to the place $AskDeviceToGoOffline : Rules$.

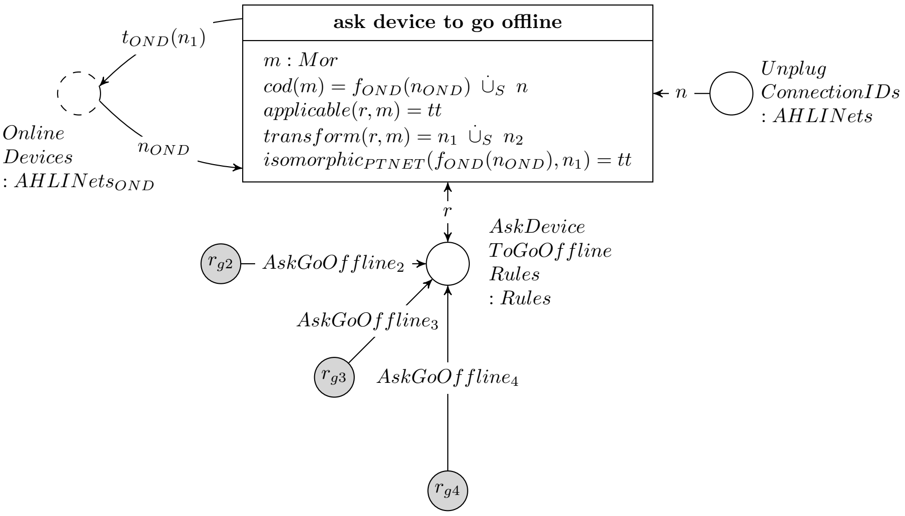

ask device to go offline
$m : Mor$
$cod(m) = f_{OND}(n_{OND}) \dot\cup_S n$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot\cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
$t_{OND}(n_1)$
n_{OND}
Online Devices : $AHLINets_{OND}$
n
Unplug ConnectionIDs : AHLINets
r
AskDevice ToGoOffline Rules : Rules
r_{g2}
$AskGoOffline_2$
$AskGoOffline_3$
r_{g3}
$AskGoOffline_4$
r_{g4}

Figure 7.38: Subnet $AskDeviceToGoOffline_{messageBroker}$

The following image illustrates the rules $AskGoOffline_2$ for setting a sender into offline mode, $AskGoOffline_4$ for setting a receiver into offline mode and $AskGoOffline_5$ for setting a transceiver into offline mode, where the value of x is to be set to $x \in \{2, 4, 5\}$ depending on the rule used.

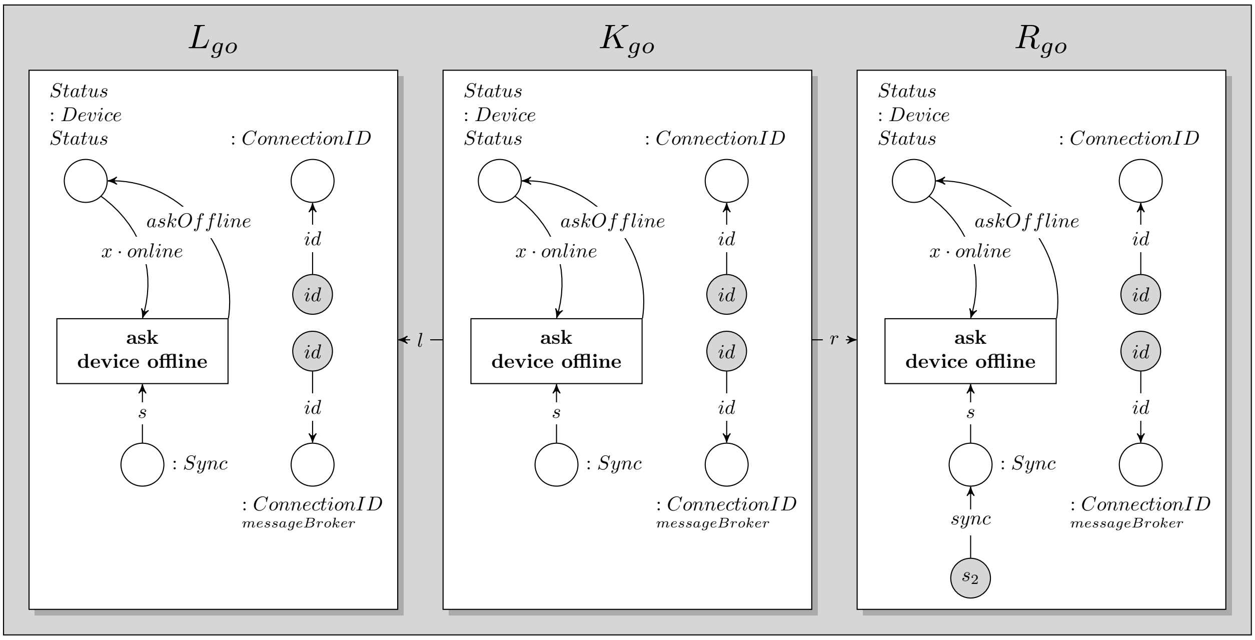


Figure 7.39: Rule $AskGoOffline_x$

7.1.3.1.7 Subnet for Subscribing a Device to a Topic

If a device wants to subscribe itself to a topic, within its device net, it needs a token set to *UnsubscribedTopics : Topic* and to : *Sync* so that it is able to fire the transition *subscribe topic*. The topics assigned as tokens to $UnsubscribedTopics$ are available regarding the needs of the individual device which is represented by this net.
It the transition *subscribe topic* is able to fire, it moves a token from *UnsubscribedTopics* to *TopicsToSubscribe.*

Subsequently, the message broker executes the process of subscribing a device to a certain topic in adding a new subnet to the net assigned as token to the place $Queues : AHLINets_Q$. This can be done in applying the rule *rSubscribe* to a token from *OnlineDevices* and the token from $Queues : AHLI$ while firing the transition *subscribe topic.* As a result, a new subnet as part of all queues is created. Additionally the token representing the topic is moved from *TopicsToSubscribe* to *SubscribedTopics* within the device net. The modified device net resp. the extended queue net are put back to its former places.

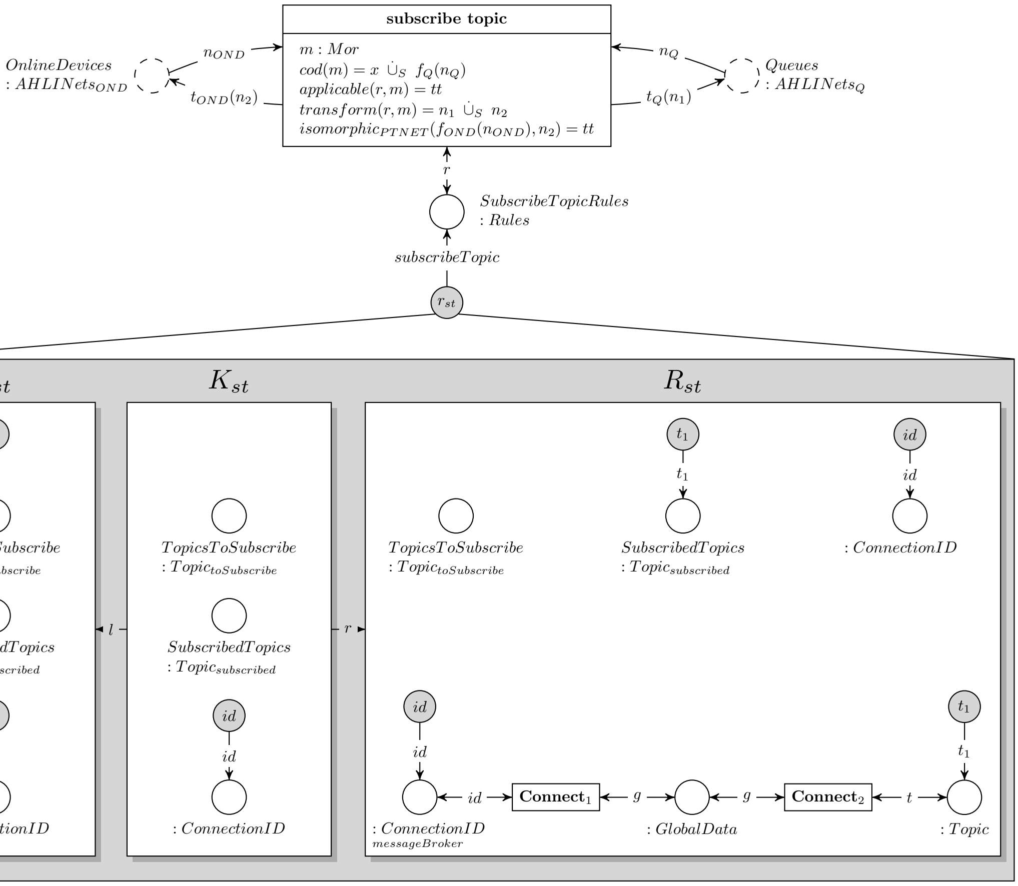


Figure 7.40: Subnet $SubscribeTopic_{messageBroker}$

7.1.3.1.8 Subnet for Unsubscribing a Device from a Topic

Unsubscribing a device from a specified topic is the reverse process to subscribing a topic as described in the last section 7.1.3.1.7 and therefore it is similar, too.
Within the device net a subscribed topic is moved to the place $TopicsToUnsubscribe : Topic_{toUnsubscribe}$ in firing the transition *unsubscribe topic.* Next, the message broker applies the rule *rUnsubscribe* to the device net and to the token assigned to the place $Queues : AHLINets_Q$. The result is a modified device net which is put back to its place $OnlineDevices : AHLINets_{OND}$. Within the device net the topic token is moved from $TopicsToUnsubscribe : Topic_{toUnsubscribe}$ to *UnsubscribedTopics : Topic.* Furthermore the subnet representing the device being subscribed to a specified topic is deleted from the queue net which will be put back to the place $Queues : AHLINets_Q$.

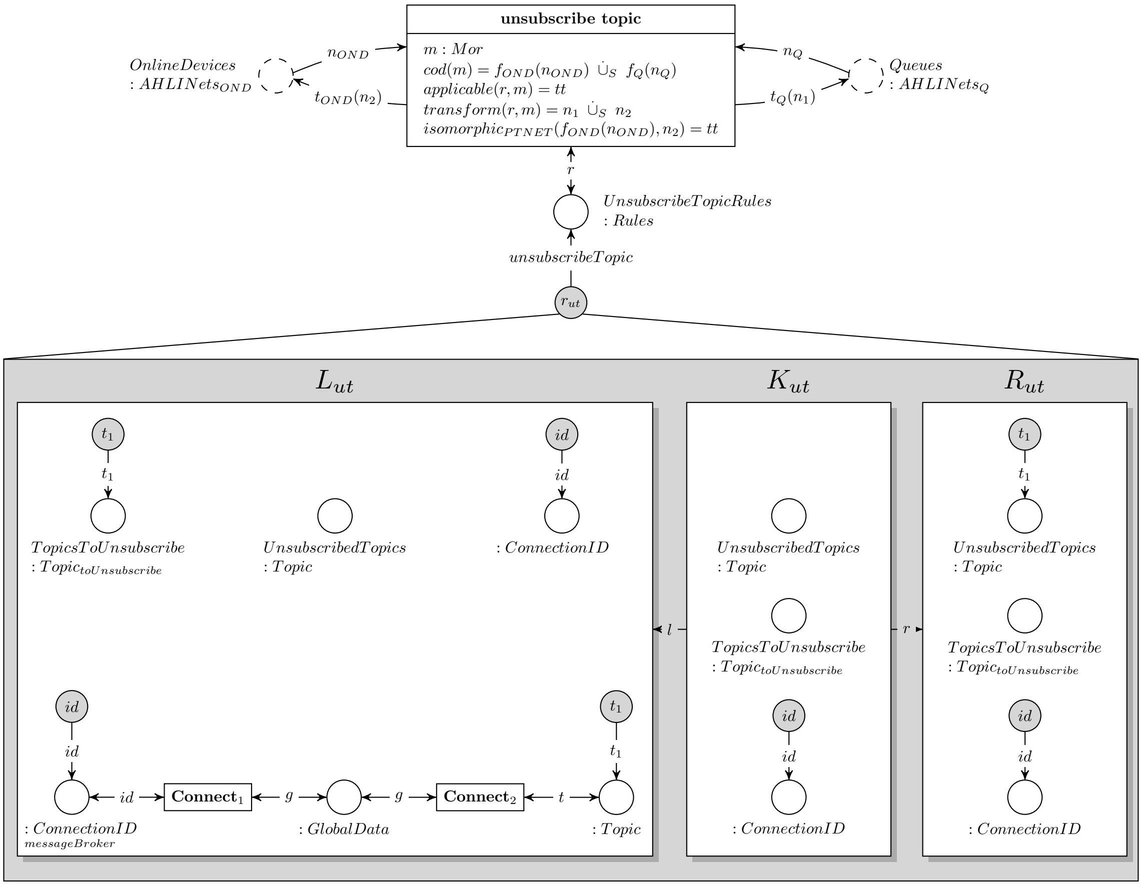

unsubscribe topic
m : Mor
cod(m) = f_OND(n_OND) ∪̇_S f_Q(n_Q)
applicable(r, m) = tt
transform(r, m) = n_1 ∪̇_S n_2
isomorphic_PTNET(f_OND(n_OND), n_2) = tt
OnlineDevices : AHLINets_OND
n_OND
t_OND(n_2)
n_Q
t_Q(n_1)
Queues : AHLINets_Q
r
UnsubscribeTopicRules : Rules
unsubscribeTopic
r_ut
L_ut
K_ut
R_ut
t_1
id
TopicsToUnsubscribe : Topic_toUnsubscribe
UnsubscribedTopics : Topic
: ConnectionID
l
r
Connect_1
Connect_2
g
t
: ConnectionID messageBroker
: GlobalData
: Topic

Figure 7.41: Subnet $UnSubscribeTopic_{messageBroker}$

7.1.3.1.9 Subnet for Setting a Device Offline

If a device net sets a token $askOffline$ to the place $Status$: $DeviceStatus$, it signals a request of going into offline mode to the message broker. But all messages should be processed before the device is able to go into offline mode. In the model shown in the images 7.42 and 7.43 all messages which are currently located on the input buffer of the device $Data_I$: $InputData$ will be deleted in applying the amalgamated rule generated out of the interaction scheme *ClearInputData* first. The transition *clear input data* takes a device net token, deletes all tokens within the device net which are assigned to the place $Data_I$. The amalgamated rule generated out of the interaction scheme as presented in figure 7.42 deletes all tokens on $Data_I$ in one step. (A rule which can be applied several times deleting one token after another is also conceivable. Such a rule will correspond to the second part ($L_{cd_{multirule}} \xleftarrow{l_{multi}} K_{cd_{multirule}} \xrightarrow{r_{multi}} R_{cd_{multirule}}$) of the interaction scheme *ClearInputData*.
Only if no token is set to the place $Data_I$, the device can be set to offline mode. To change the device status the transition *set device offline* needs to be fired in order to apply one of the rules assigned to the place $SetDeviceOfflineRules$: *Rules*. Each of the three rules $SetReceiverOffline$, $SetTransceiverOffline$ and $SetSenderOffline$ sets one type of device into offline mode in deleting places, transitions and tokens from the device net as shown in figures 7.45, 7.44 and 7.46.
After firing the transition *set device offline*, a token *receiveData* is deleted from the place *ReceiveDataRules : Rules*$_R$ and the modified device net is transferred from *OnlineDevices : AHLINets*$_{OND}$ to *OfflineDevices : AHLINets*$_{OFFD}$. The rule *receiveData* which is represented by a token is illustrated in image 7.33.

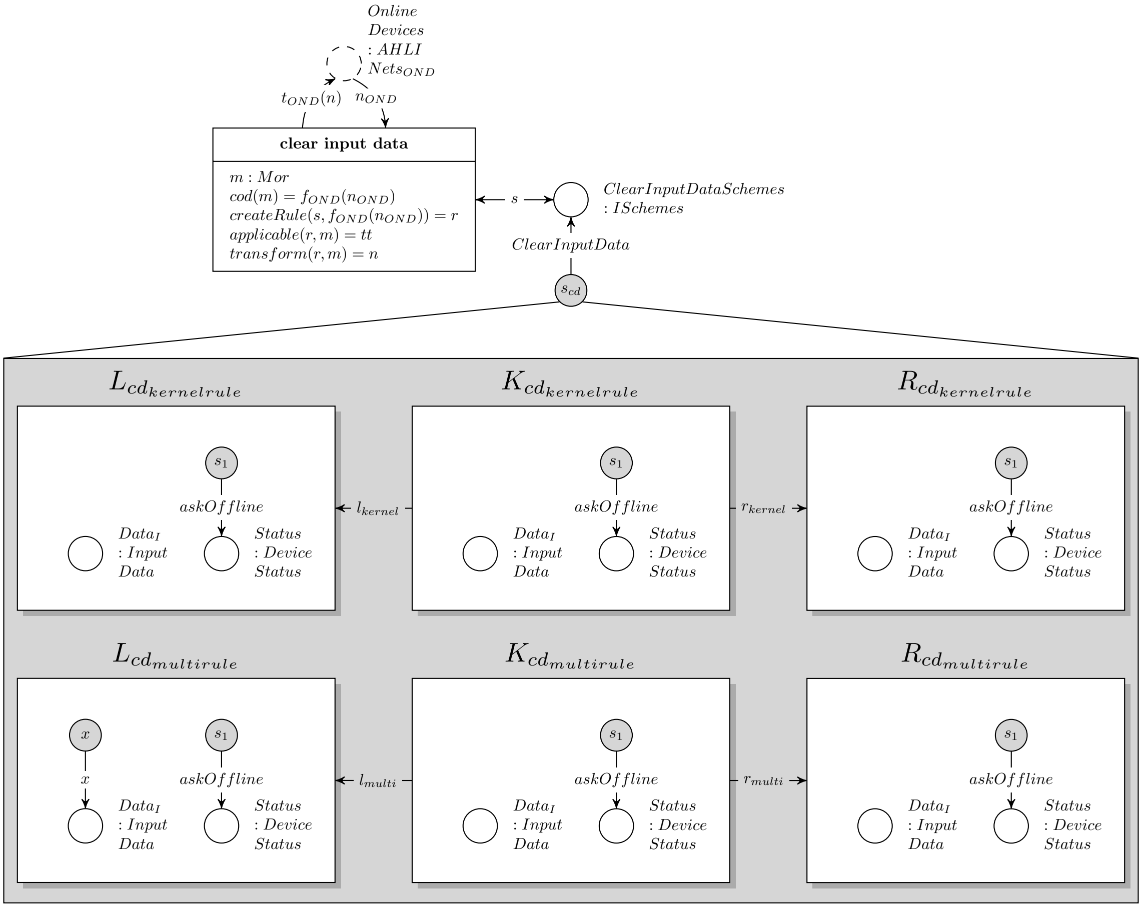


Figure 7.42: Subnet $ClearInputData_{messageBroker}$ with interaction scheme $ClearInputData$

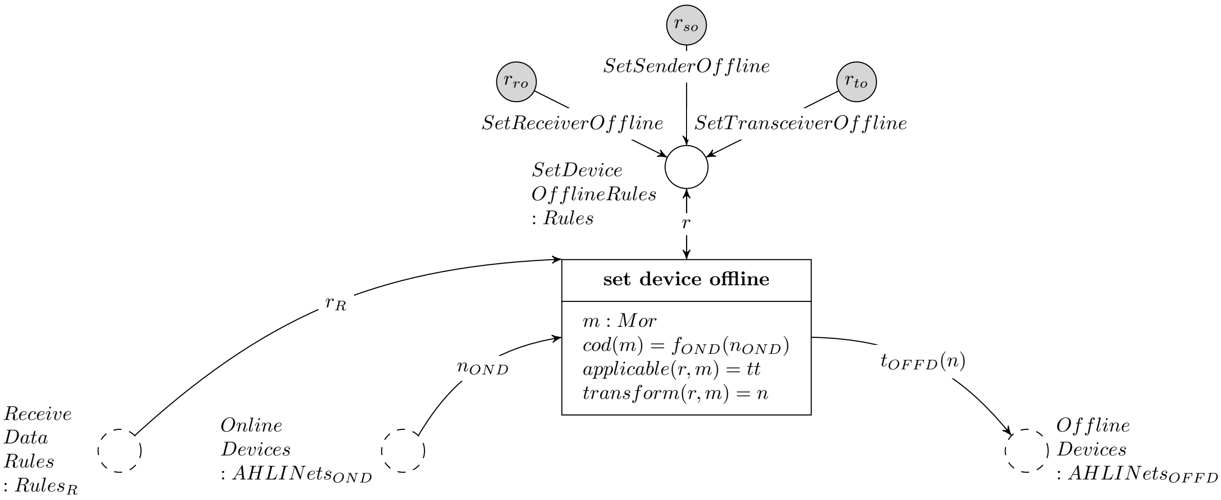


Figure 7.43: Subnet $setDeviceOffline_{messageBroker}$

Rule $SetTransceiverOffline$ for Setting a Transceiver Into Offline Mode

The interaction scheme $SetTransceiverOffline$ (see image 7.44) can be applied, if no token is assigned to the place $Data_I$ anymore. Then, many places, transitions and tokens will be deleted from the net in order to stop the device and prevent any processing. Additonally, the device status is set to $offline$, that means that the device is in offline mode.

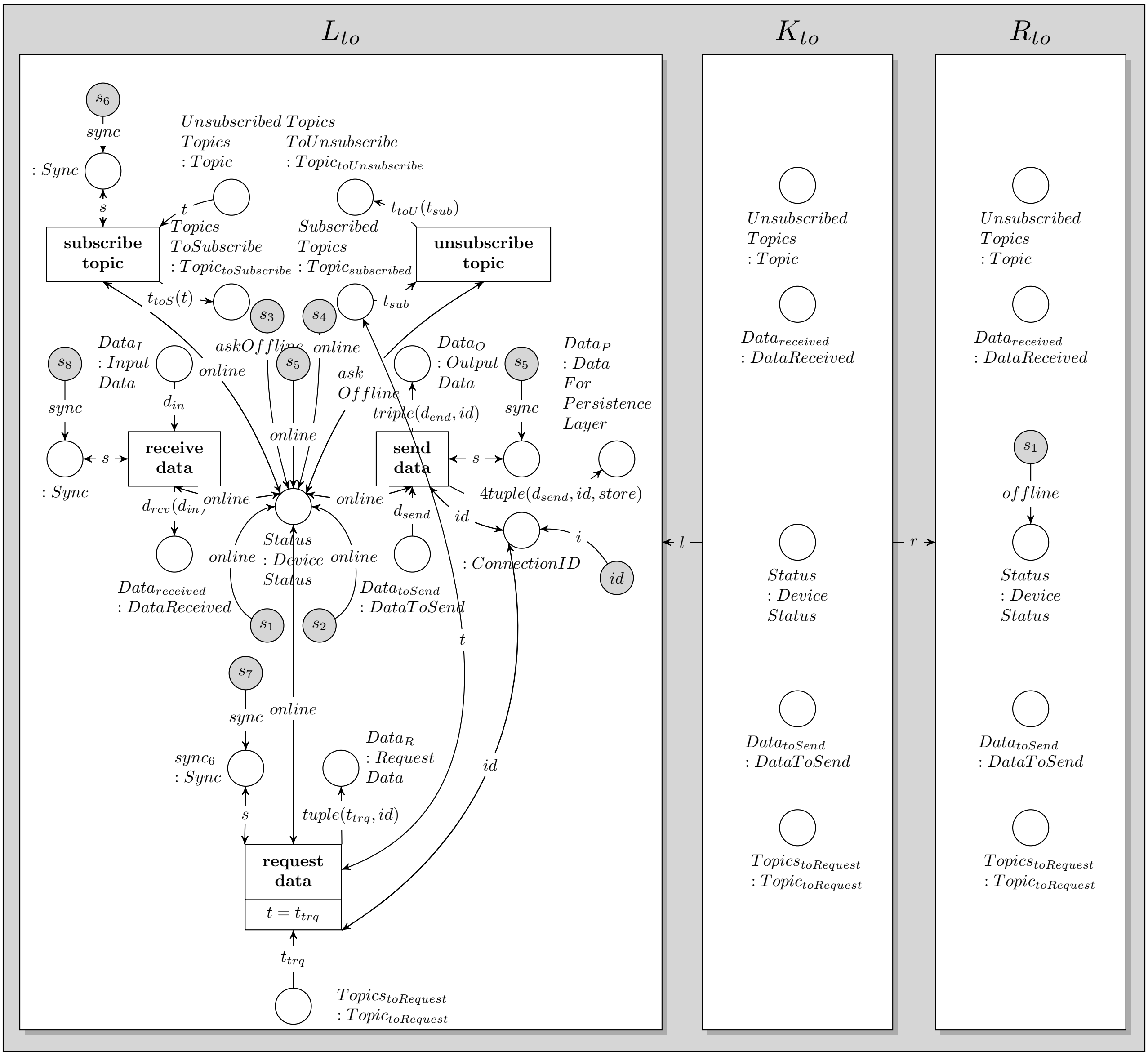


Figure 7.44: Rule $SetTransceiverOffline$

Rule $SetReceiverOffline$ for Setting a Receiver Into Offline Mode

The interaction scheme $SetReceiverOffline$ (see image 7.45) changes the status of a receiver into offline mode in assigning the token $offline$ to the place $Status : DeviceStatus$. Additionally important places, transitions and tokens for enabling the processing of messages are deleted. The rule calculated out of this interaction scheme is only able to change receivers because of the NAC_{ro} no match to a net which contains the place $Data_{toSend} : DataToSend$ can be found. Furthermore, this rule can only be applied if no tokens are assigned to $Data_I : InputData$.

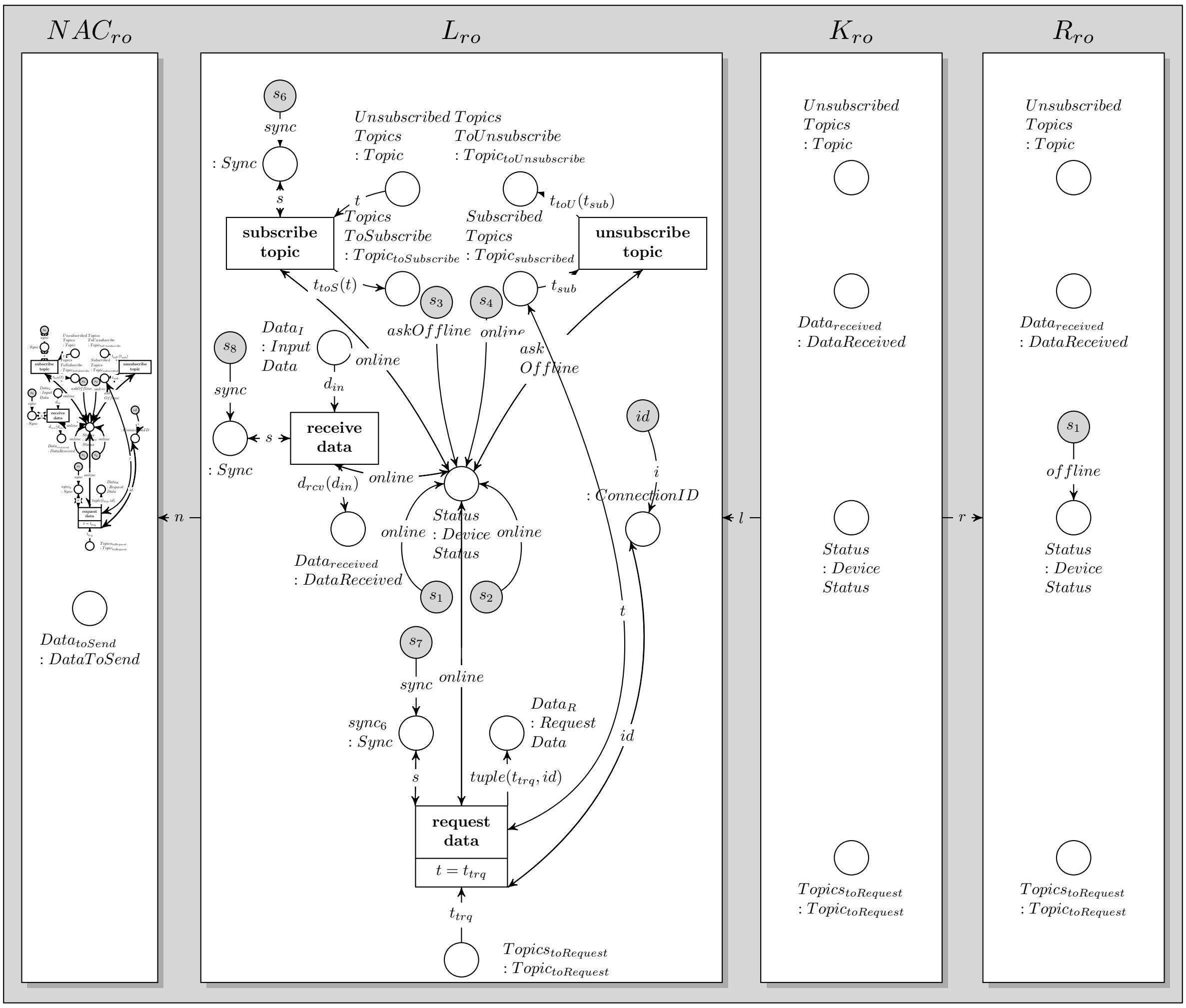


Figure 7.45: Rule $SetReceiverOffline$

Rule $SetSenderOffline$ for Setting a Sender in Offline Mode

The interaction scheme $SetSenderOffline$ (see image 7.46) is similar to the interaction scheme to $SetReceiverOffline$. It is applicable to sender nets which do not have any tokens assigned to the place $Data_I : InputData$. After execution, places, transitions and tokens are deleted from the sender net and its device status is set to $offline$, which represents the offline mode.

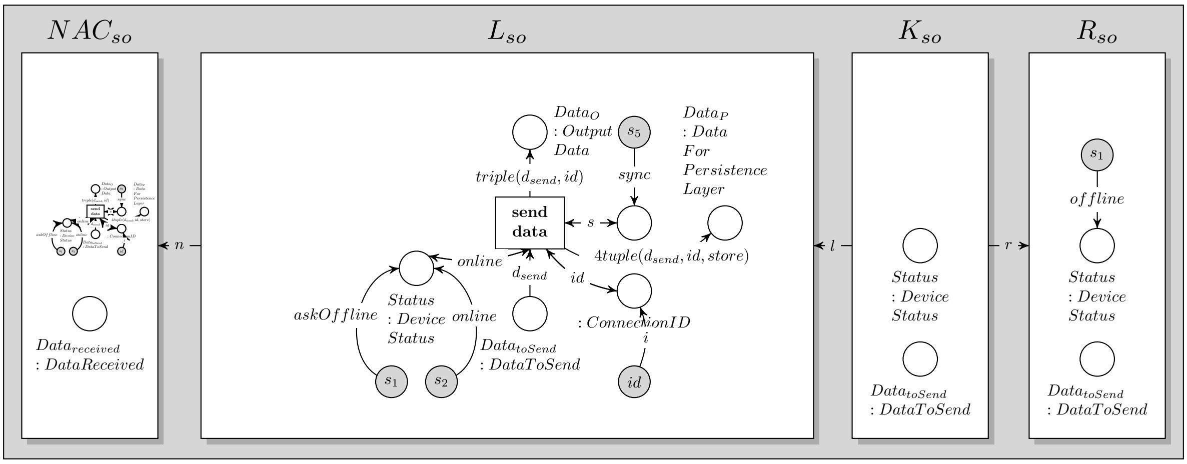


Figure 7.46: Rule $SetSenderOffline$

7.1.3.1.10 Subnet for Handling an Exception or Crash within the Message Broker

When an exception or crash of a certain device occurs, it will be handled in deleting the device and its queues. The user needs to solve such a problem manually so that a defect device should not be kept within the list of online devices, i.e. the problematic device net should not be kept as token assigned to $OnlineDevices : AHLINets_{OND}$. To reassign the repaired device, the user needs to intervene in the system in using the AHOI rule $PluginDevice_{user}$ (see 7.83).

To delete a defective device, the amalgamated rule generated out of the interaction scheme $handleCrash$ is applied. It deletes the device from the place $OnlineDevices : AHLINets_{OND}$ and changes the net from $Queues : AHLINets_{Q}$ in deleting all subnets referring to the specific connection identification of the device.

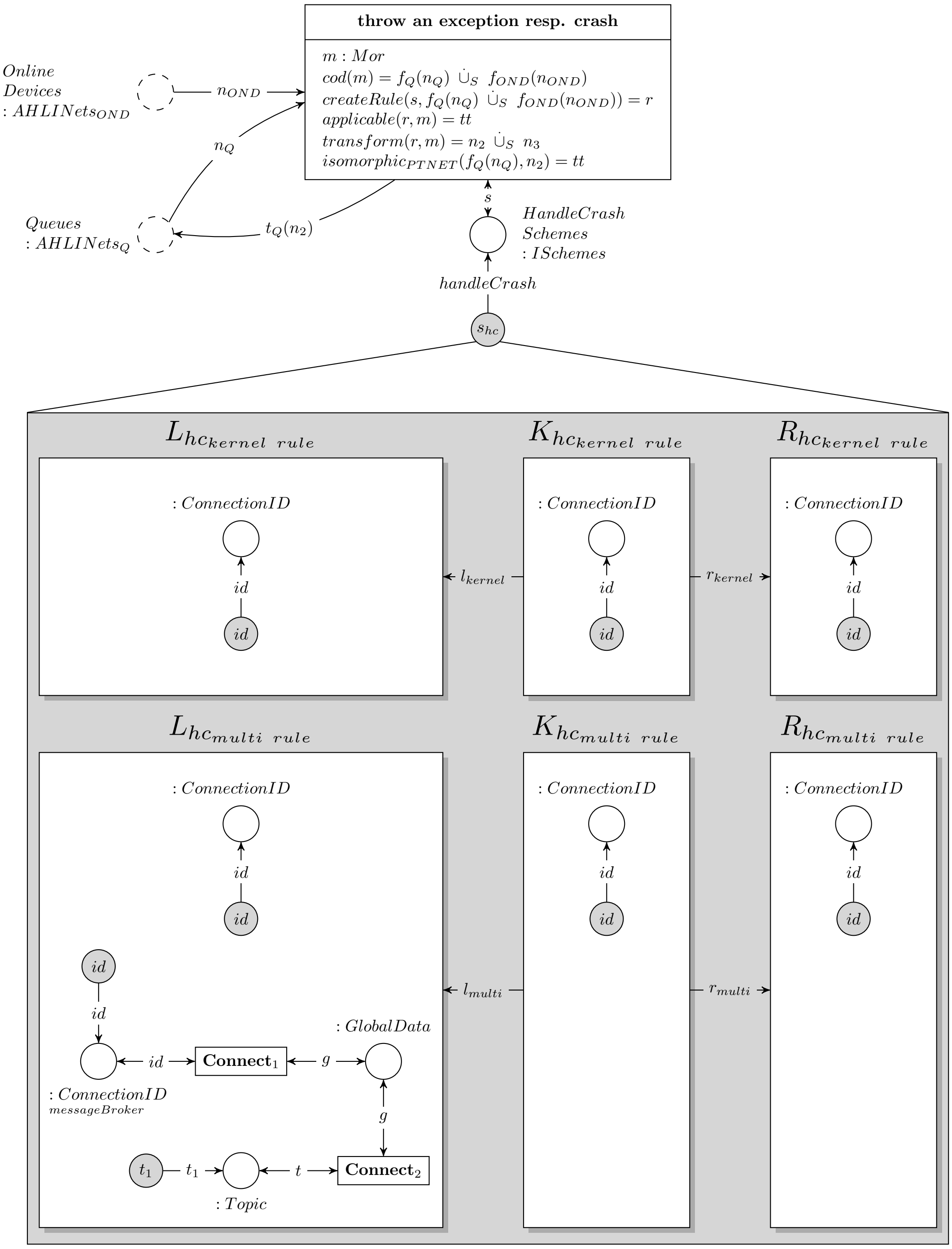


Figure 7.47: Subnet $exceptionRespCrash_{messageBroker}$

7.1.3.1.11 Revisiting the Requirements Towards the Message Broker

In section 4.2.3.1 the requirements set up for the message broker are presented. Furthermore, the Living Place includes the characteristics of a ubiquitous computing system (see 2.1), from which new requirements follow which are mentioned in 4.1. This section deals with the revision of these two kinds of requirements regarding their fulfilment by the model of the message broker.

$\mathbf{r_{mom_{broker_1}}}$: The model of the Living Place system provides user operations $PluginDevice_{user,y}$ (see 7.83) which enable the user to add a new device in offline mode to the Living Place system. This device is assigned as token to the place $OfflineDevices : AHLINets_{OFFD}$. To be set into online mode, the device fires the transition *ask device online* in order to assign a token *askOnline* to its internal place $Status : DeviceStatus$. Afterwards, the message broker is able to fire the transition *set device online* and in this process is able to apply one of the rules *setReceiverOnline*, *setSenderOnline* or *setTransceiverOnline* which corresponds to the type of the device (receiver, sender or transceiver) and which transforms the device net in adding an individual connection id and adding the parts to enable the processing of messages to the device net in order to switch the device into online mode. Furthermore, the device net in online mode will be assigned as token to the place $OnlineDevices : AHLINets_{OND}$.
So, the requirement $r_{mom_{broker_1}}$ is fulfilled.
The transition and its rules are illustrated in 7.29, 7.32, 7.30 and 7.31.

$\mathbf{r_{mom_{broker_2}}}$: A device announces that it wants do get disconnected in assigning the token $askOffline$ to its place $Status : DeviceStatus$. Then, if messages are assigned as tokens to the place $Data_I : InputData$ within the object net of the device, these tokens will be removed in firing the transition *clear input data* and applying a rule generated out of the interaction scheme *clearInputData* (see 7.42). Therefore, the requirement $r_{mom_{broker_2}}$ is met.

$\mathbf{r_{mom_{broker_3}}}$: After a device has announced, that it wants to go into offline mode and if no messages are assigned as tokens to $Data_I : InputData$, the message broker is able to fire the transition *set device offline* and in this process applies one of the rules *setReceiverOffline*, *setSenderOffline* or *setTransceiverOffline*, see 7.43, 7.45, 7.46 and 7.44. For each type of device another rule is provided, e.g. the rule *setReceiverOffline* sets a receiver into offline mode.
No match can be found, if tokens are assigned to $SubscribedTopics : Topic_{subscribed}$ or to *TopicsToSubscribe :* $Topic_{toSubscribe}$, or to $Data_O : OutputData$, or to $Data_P : DataForPersistenceLayer$, or to $Data_R$ *: RequestData.* These tokens have to be processed before the device can be set into offline mode. Then, the part for processing will be removed from the object net of the device and the specific connection id will be deleted. The device will be removed from $OnlineDevices : AHLINets_{OND}$ and its object net in offline mode will be assigned to $OfflineDevices : AHLINets_{OFFD}$.
Consequently, the reqirement $r_{mom_{broker_3}}$ is fulfilled.

$\mathbf{r_{mom_{broker_4}}}$: If a token representing a specific topic is assigned to $TopicsToSubscribe : Topic_{toSubscribe}$ within a device net in online mode, the message broker is able to apply the rule *subscribeTopic* in firing the transition *subscribe topic* (see 7.40). In doing so, the subnet assigned to $Queues : AHLINets_Q$ will be extended by a new structure representing the association between the given topic and the connection id of the device. An example of such an association is given in image 7.34. This subnet standing for an association also represents an internal data buffer for all messages which should be distributed to the device.
In applying the rule *subscribeTopic*, the token assigned to *TopicsToSubscribe* will be removed and assigned to $SubscribedTopics : Topic_{subscribed}$. Therefore a notification of a successful subscription will be provided to the device.
This step satisfies the requirement $r_{mom_{broker_4}}$.

$\mathbf{r_{mom_{broker_5}}}$: In order to announce that a topic should get unsubscribed by the message broker, the device assigns the specifc topic to the place $TopicsToUnsubscribe : Topics_{toUnsubscribe}$. The message broker unsubscribes the topic in applying the rule *unsubscribeTopic* in firing the transition *unsubscribe topic* (see 7.41). Then, the subnet representing the association between topic and connection id of the device will be removed from the object net assigned to $Queues : AHLINets_Q$. Additionally, the topic

assigned as token to $TopicsToUnsubscribe$ will be removed and placed to $UnsubscribedTopics$: $Topic$ within the device net, so that the device is notified.
Consequently, the requirement $r_{mom_{broker_5}}$ is fulfilled.

$\mathbf{r_{mom_{broker_5}}}$: If a device crashs, the message broker is able to fire the transition *throw an exception resp. crash*, which applies the amalgamated rule calculated out of the interaction scheme *handleCrash* to the device assigned as token to *OnlineDevices* : $AHLINets_{OND}$. The rule removes the subnet representing the association between topic and connection id of the device within the object net assigned to *Queues* : $AHLINets_Q$. After firing the transition, the token representing the defective device, will be deleted (see 7.47). Therefore the requirement $r_{mom_{broker_5}}$ is fulfilled.

$\mathbf{r_{mom_{broker_7}}}$: As presented in detail in section 7.1.2, each device contains a place $Data_O$: $OutputData$ which represents the output data buffer for the message broker. When firing the transition *receive data from device* and, in this process, applying one of the rules *receiveData* (see 7.33), the message assigned as token to the place $Data_O$ within the object net of the device, will be collected and stored into a newly created AHLI Net representing an internal data buffer of the message broker. This data buffer will be assigned as token to the place $DataBufferRepository$: $AHLINets$. Hence, the requirement $r_{mom_{broker_7}}$ is met.

$\mathbf{r_{mom_{broker_{8(a)}}}}$: Every time the transition *set device online* fires, a new rule *receiveData* will be assigned as token to the place *ReceiveDataRules* : $Rules_R$. Consequently, for each device in online mode, a rule is assigned to this place, e.g. if five devices are assigned to *OnlineDevices* : $AHLINets_{OND}$, then five rules *receiveData* will be available. (When removing a device, one rule will be removed, too.) Therefore, the transition *receive data* is able to fire in parallel for each device net assigned to *OnlineDevices*. So, the requirement $r_{mom_{broker_8}}$ is met.

$\mathbf{r_{mom_{broker_9}}}$: After infering new 5W1H context data, the Context Interpreter puts this information as token to the place $DataBufferRepository$: $AHLINets$, in order to distribute this information by the message broker, which fires the transition *copy data to queues and to context interpreter*, first (see 7.35). Additionally, the persistence layer assigns its answer to a request directly to the place : *GlobalData* within the corresponding subnet within *Queues* : $AHLINets_Q$, which represents the correct association between the topic of the request and the connection id of the requesting device.
In both cases, the final distributing of data will be done by the message broker which fires the transition *send data to device*. Hence, this requirement $r_{mom_{broker_9}}$ is met.

$\mathbf{r_{mom_{broker_{10}}}}$: Because of the existence of the object net assigned to *Queues* : $AHLINets_Q$ and the possible modifications on this net provoked by the transitions *subscribe topic*, *copy data to queues and to context interpreter* and *send data to device*, the requirement $r_{mom_{broker_{10}}}$ is fulfilled.

$\mathbf{r_{mom_{broker_{11}}}}$: Each disjoint subnet of the object net assigned to *Queues* : $AHLINets_Q$ represents an association between the connection id of a device and a topic. When distributing new messages to the subscribed devices, the rule created out of the interaction scheme *copyData* will be applied when firing the transition *copy data to queues and to context interpreter*. This rule copies the message to the place : *GlobalData* of all disjount subnets, which include the specific topic of the message. Afterwards the transition *send data* fires in order to remove the token assigned to : *GlobalData* and assign it to the place $Data_I$: $InputData$ of the corresponding device net (the device net containing the same connection id). That means, only devices subscribed to the specific topic will receive the message. All other devices do not get this information. Consequently, the requirement $r_{mom_{broker_{11}}}$ is satisfied.

$\mathbf{r_{mom_{broker_{12}}}}$: After firing the transition *copy data to queues and to context interpreter*, the information is copied and forwarded to the place *ContextInterpreterQueue* : $AHLINets$, which stands for the data input buffer of the Context Interpreter (see 7.35). It follows, that the requirement $r_{mom_{broker_{12}}}$ is met.

$\mathbf{r_{mom_{broker_{13}}}}$: The corresponding *AHLI* transformation rules, that are defined as tokens within the $AHOI_{AHLI}$ net of the Message Oriented Middleware are defined in a way, so that only the interface net structure of each device resp. *AHLI* net token of place *OnlineDevices* resp. $OfflineDevice$ is defined within these rules so that only the interface of each device is known to the Middleware. Therefore, the

Middleware completely abstracts from further implementation details of each communicating device. So, requirement $r_{mom_{broker_{13}}}$ is met.

$\mathbf{r_{ubio_2}}$ **and** $\mathbf{r_{ubis_2}}$: The message broker collects data from all sending devices, processes them and distributes them to corresponding receiving devices. Therefore the message broker enables the communication between devices. It controlls the internal states of the devices, e.g.: Should the message broker subscribe a new topic for a device? Does a device request a switch into offline mode?, etc. and therefore guarantees the communication between devices. Because of that, the requirements r_{ubio_2} and r_{ubis_2} are met.

To summarise, all requirements (presented in 4.2.3.1 and the requirements r_{ubio_2} and r_{ubis_2} in 4.1) demanded to the model of the message broker are fulfilled.

7.1.3.2 User Operation Processing Unit - Subnet for Entering User Data to The Corresponding Online Device

In chapter 3 the levels of modelling are presented, which take place on three levels: object, system and User Level. The inhabitant of the Living Place system is only able to interact with the system in applying AHOI rules which will be presented in section 7.1.4. AHOI rules can be applied to the System Level in order to change its behaviour, but no AHOI rule is directly able to change the Object Level, therefore, the user is not able to change the state of a device connected to the system in applying an AHOI rule directly. But the Living Place system contains some devices which provide interfaces for direct user access, like the Daily Planner, the Display in the Multitouch Kitchen Counter, the Alarm Clock 2.0, the Ambient Light and the Location-Based Screen. For that, the model of the Living Place system contains the subnet presented in the image 7.48. It consits of a transition which applies an AHLI rule which is assigned to the place *EnterDataIntoDeviceRules* : *Rules* in order to interact with an online device and fulfil the requirement that the inhabitant should be able to interact with the connected devices, e.g. switching the Alarm Clock 2.0 off or entering new calendar data to the Daily Planner.
The workflow of changing an online device is as follows: The user applies a specific AHOI rule which assigns a corresponding AHLI rule to the place *EnterDataIntoDeviceRules*. Afterwards this rule can be automatically applied to a corresponding device assigned as object net to the place *OnlineDevices* : $AHLINets_{OND}$ in firing the transition *enter data to device* on System Level. As a result, an online device is modified, if a match was found, according to the inhabitants request. Therefore, the user access to connected devices is modelled on all three levels: an AHOI rule which assigns an AHLI rule to a place on System Level which will be assigned to a net on Object Level.
A rule assigned to *EnterDataIntoDeviceRules* will only be applied once. After firing the transition *enter data to device*, this rule will be removed.

Initially no rule is assigned to the place *EnterDataIntoDeviceRules*. The following table enumerates the available AHOI rules and their corresponding AHLI rules for an interaction of the inabitant with the device nets on Object Level:

AHOI rule	**AHLI rule**	**Corresponding Device** (on Object Level)
$ResetAlarmClock2.0_{user}$ (see 7.1.4.5)	$ruleDeleteNextAppointments$ (see 7.63)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleDeleteBedData$ (see 7.64)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleResetContextData$ (see 7.65)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleDeleteWakeUpTime_1$ (see 7.66)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleDeleteWakeUpTime_2$ (see 7.67)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleDeletePersonState$ (see 7.68)	Alarm Clock 2.0

$ResetAlarmClock2.0_{user}$	$ruleResetTimestamp$ (see 7.69)	Alarm Clock 2.0
$ResetAlarmClock2.0_{user}$	$ruleResetSync$ (see 7.70)	Alarm Clock 2.0
$ClearDisplay_{user,d}$ (see 7.1.4.6.1)	$ruleClearDisplay_{d}$ (see 7.1.3.5.2)	Display
$EnterDataToDaily-$ $Planner_{user,t,d}$ (see 7.1.4.7)	$ruleAddData_{planner,t,d}$ (see 7.1.3.5.3)	Daily Planner
$RemoveDataFromDaily-$ $Planner_{user,t,d}$ (see 7.1.4.7)	$ruleRemoveData_{planner,t,d}$ (see 7.1.3.5.4)	Daily Planner
$MeasurePressureIn-$ $Bed_{user,x_1,x_2,x_3,x_4,x_5,x_6}$ (see 7.1.4.8)	$ruleMeasure-$ $Pressure_{bed,x_1,x_2,x_3,x_4,x_5,x_6}$ (see 7.1.3.5.5)	Intelligent Bed
$MeasurePosition_{user,x_1,x_2,x_3,x_4,x_5,x_6}$ (see 7.1.4.9)	$ruleMeasure-$ $Position_{indoor,x_1,x_2,x_3,x_4,x_5,x_6}$ (see 7.1.3.5.6)	Indoor Positioning System
$SwitchOnTV_{user}$ (see 7.1.4.10)	$ruleSwitchOn_{TV}$ (see 7.76)	Location-Based Screen
$SwitchOffTV_{user}$ (see 7.1.4.10)	$ruleSwitchOff_{TV}$ (see 7.77)	Location-Based Screen
$SwitchOnLight_{user}$ (see 7.1.4.11)	$ruleSwitchOn_{Light}$ (see 7.1.3.5.9)	Ambient Light
$SwitchOffLight_{user,bright}$ (see 7.1.4.11)	$ruleSwitchOff_{Light,bright}$ (see 7.79)	Ambient Light
$SwitchOffLight_{user,dimmed}$ (see 7.1.4.11)	$ruleSwitchOff_{Light,dimmed}$ (see 7.80)	Ambient Light

Table 7.2: AHLI rules for Access to an Online Device and Their Corresponding AHOI rules

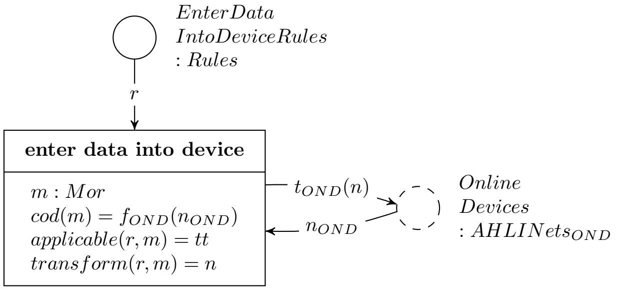


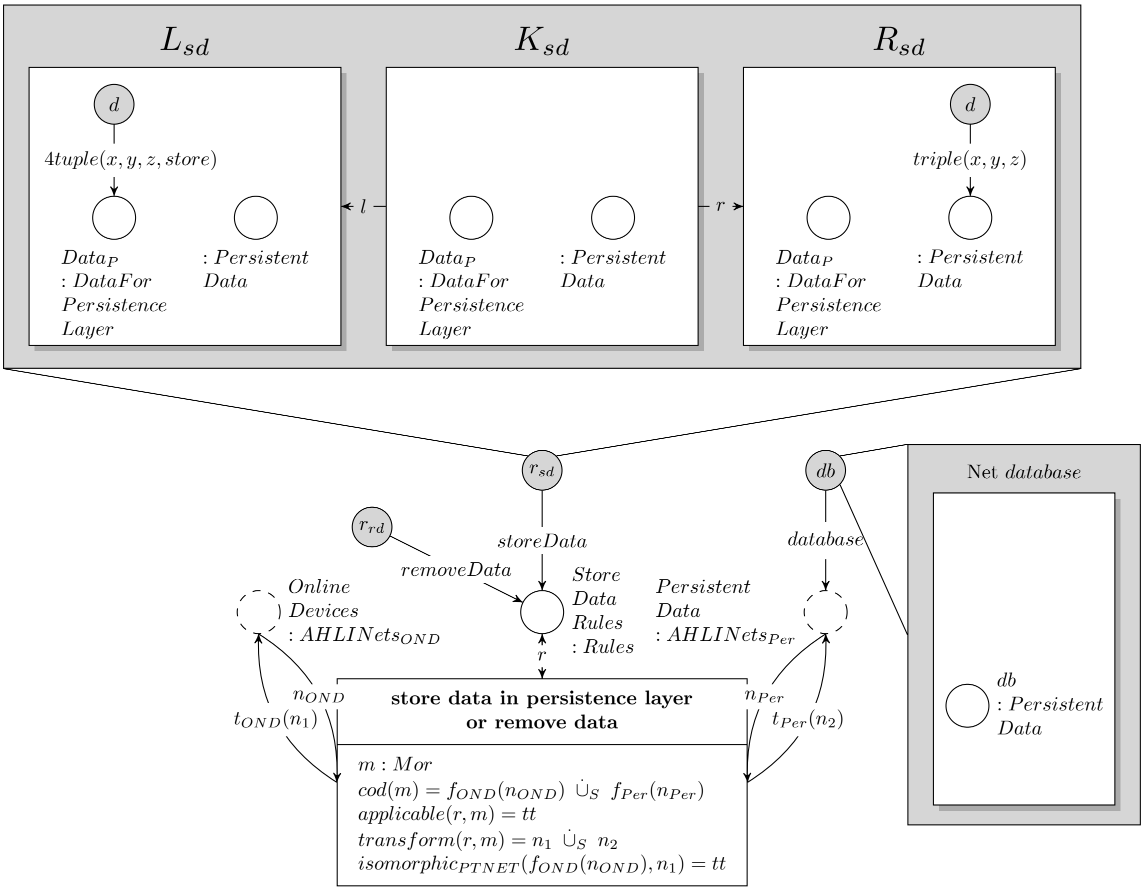
Figure 7.48: Subnet $EnterDataIntoOnlineDevice_{messageBroker}$

7.1.3.3 The Persistence Layer

The persistence layer is part of the message broker and contains information which were exchanged through the Message Oriented Middleware and were stored in a special database. These data can be demanded by connected components. Each device which is able to receive data over the Message Oriented Middleware (i.e. receiver and transceiver) contains a part for requesting information from the persistence layer.[7].
In the following, the parts for the communication with the persistence layer within the Message Oriented Middleware and for storing incoming data will be illustrated.

7.1.3.3.1 Subnet for Storing Data From the Device in the Persistence Layer

Each message which is sent by a device net to the message broker is additionally put to the place $Data_P : DataForPersistenceLayer$ in the device net. The message broker explicitly collects all information, which are assigned as 4-tuple tokens containing *store* as forth element, to this place in all device nets. For that, the rule *storeData* needs to be applied to the nets assigned to $OnlineDevices : AHLINets_{OND}$ and $PersistentData : AHLINets_{Per}$, which copies the data from the local place $Data_P$ of the device net to the place $PersistentData$ of the persistence layer representing a database containing all information ever processed in the Message Oriented Middleware.
In contrast, if a 4-tuple token containing *remove* is assigned to the place $Data_P$, the corresponding data record should be removed from the persistence layer. For deleting data, the AHLI rule *removeData* will be applied when firing the transition *store data in persistence layer or remove data*, if a match could be found.



L_{sd}
K_{sd}
R_{sd}
d
4tuple(x, y, z, store)
$Data_P$: DataFor Persistence Layer
: Persistent Data
l
r
d
triple(x, y, z)
r_{sd}
db
Net database
r_{rd}
storeData
removeData
database
Store Data Rules : Rules
Online Devices : $AHLINets_{OND}$
Persistent Data : $AHLINets_{Per}$
r
n_{OND}
$t_{OND}(n_1)$
n_{Per}
$t_{Per}(n_2)$
store data in persistence layer or remove data
$m : Mor$
$cod(m) = f_{OND}(n_{OND}) \dot\cup_S f_{Per}(n_{Per})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot\cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
db : Persistent Data

Figure 7.49: Subnet $storeDataInPersistenceLayer_{persistenceLayer}$

[7] The transition *request data* (within the device nets of a transceiver of a receiver) is responsible for requesting data from the persistence layer, compare 7.5 and 7.7.

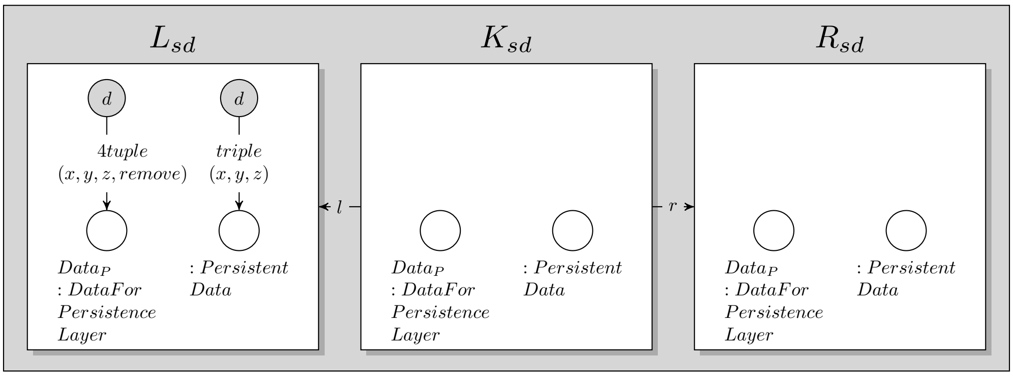


Figure 7.50: Rule *removeData*

7.1.3.3.2 Send a Message to the Persistence Layer and Generate an Answer

If a device requests information from the persistence layer in the Message Oriented Middleware, a token will be assigned to the place $Data_R$: $RequestData$. The persistence layer fetches the request in applying the rule $receiveRequestFromDevice$. For that a requesting device from $OnlineDevices$: $AHLINets_{OND}$ will be taken, and the token containing the request will be deleted and a new place $Request$: $GlobalDataTmp$ will be created to which the request token is assigned. The resulting net will be split and the transformed device net will be put to the place $OnlineDevices$. The newly created net containing the place $Request$: $GlobalDataTmp$ with its token will be assigned to the place $Requests$: $AHLINets_{Rq}$ of the message broker.

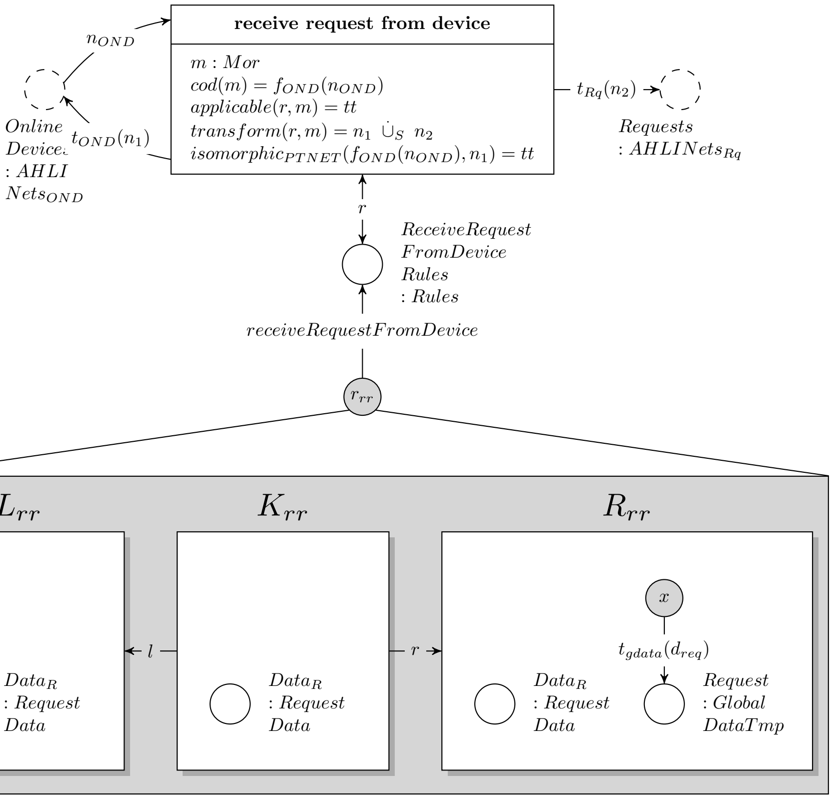

receive request from device
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
n_{OND}
$t_{OND}(n_1)$
$Online$ $Devices$: $AHLI$ $Nets_{OND}$
$t_{Rq}(n_2)$
$Requests$: $AHLINets_{Rq}$
r
$ReceiveRequest$ $FromDevice$ $Rules$: $Rules$
$receiveRequestFromDevice$
r_{rr}
L_{rr}
K_{rr}
R_{rr}
x
d_{req}
$Data_R$: $Request$ $Data$
l
$Data_R$: $Request$ $Data$
r
x
$t_{gdata}(d_{req})$
$Data_R$: $Request$ $Data$
$Request$: $Global$ $DataTmp$

Figure 7.51: Subnet $receiveRequestFromDevice_{messageBroker}$

Afterwards the request will be evaluated in collecting all past data belonging to the specified topic and copying all appropriate information from the database of the persistence layer to the message queue of the requesting device. For that, the rule $createAnswerToRequest$ will be applied to the nets from $Requests$: $AHLINets_{Rq}$, $Queues$: $AHLINets_Q$ and $PersistentData$: $AHLINets_{Per}$. The net assigned to the place $Requests$ will be taken to compare the requested topic with the topics stored in the database of the persistence layer which is represented by the place $PersistentData$. The net previously assigned to *Requests* will be deleted after applying the rule. In contrast, the net from *PersistentData* will stay unchanged. Instead, the subnet belonging to the requesting device id and the specified topic will be extended by all tokens containing past information belonging to the topic. It will be assigned to *Queues*.

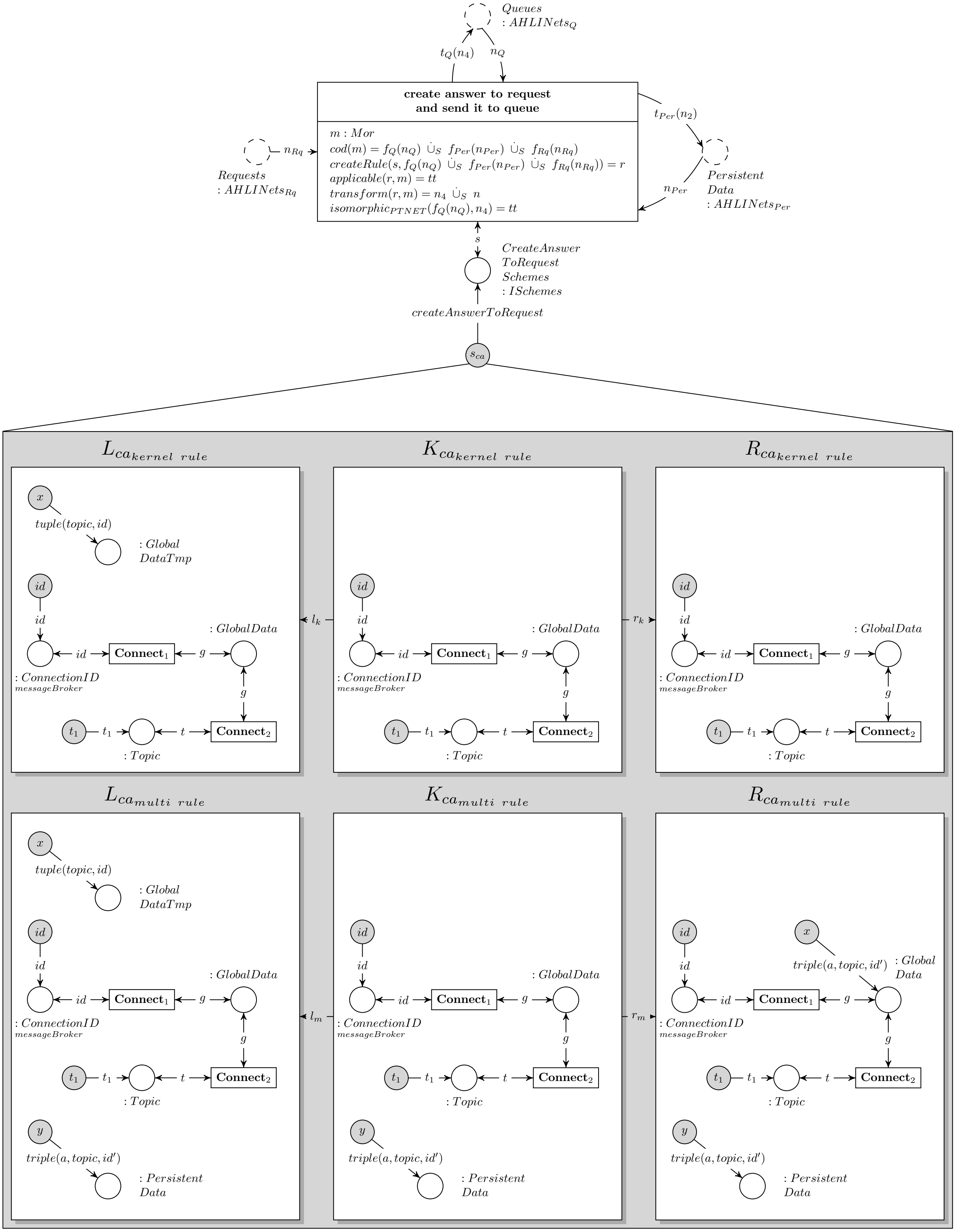


Figure 7.52: Subnet $createAnswerToRequest_{messageBroker}$

7.1.3.3.3 Revisiting the Requirements Towards the Persistence Layer

In section 4.2.3.2 requirements regarding the model of the persistence layer as part of the Message Oriented Middleware were set up. In the following, these requirements will be revised.

$\mathbf{r_{mom_{persistence_1}}}$: As presented in section 7.1.2, each device in online mode contains a place $Data_P$: $DataForPersistenceLayer$. Furthermore, each message provided for the message broker, will be assigned additionally to the place $Data_P$. The token assigned to this place will be copied when the transition *store data in persistence layer or remove data* fires and because of that applies the rule $storeData$. This is a part of the model of the persistence layer, as shown in image 7.51.
So, the requirement $r_{mom_{persistence_1}}$ is fulfilled by the model.

$\mathbf{r_{mom_{persistence_2}}}$: The AHLI net assigned as token to the place $PersistentData : AHLINets_{Per}$ contains the stored data. Therefore, this requirement $r_{mom_{persistence_2}}$ is satisfied.

$\mathbf{r_{mom_{persistence_3}}}$: The rule *removeData* which can be applied when firing the transition *store data in persistence layer or remove data*, deletes a single data record from the data storage. Following, the requirement $r_{mom_{persistence_3}}$ is fulfilled.

$\mathbf{r_{mom_{persistence_4}}}$: Device nets in online mode of the kind receiver or transceiver contain a part for sending requests to the persistence layer. This part consists of: the transition *request data* and the places connected to this transitions: $Data_R$: *RequestData* to which tokens can be assigned, characterising requests ready to be processed by the persistence layer, the place $Topics_{toRequest} : Topic_{toRequest}$ to which tokens representing topics are assigned, the places : *Sync* (with a synchronize token assigned to it), *Status* : *DeviceStatus* and $connection_{id} : ConnectionID$.[8]
After a token is assigned to $Data_R$ within the device net, it can be collected by the persistence layer in firing *receive request from device* and applying the rule $receiveRequestFromDevice$.
As a result, the requirement $r_{mom_{persistence_3}}$ is met.

$\mathbf{r_{mom_{persistence_5}}}$: After receiving a new request, it will be processed in firing the transition *create answer to request and send it to queue.* In doing so, a rule created out of the interaction scheme *createAnswerToRequest* will be applied. It collects all information belonging to the topic which is specified by the request. Because of that, the requirement $r_{mom_{persistence_5}}$ is satisfied.

$\mathbf{r_{mom_{persistence_6}}}$: When firing the transition *create answer to request and send it to queue*, all collected data records belonging to the topic, which is specified by the request, will be assigned to the place : *GlobalData* within the object net assigned to *Queues* : $AHLINets_Q$, which is connected to the given topic and the connection id of the requesting device. Consequently, the place : *GlobalData* pictures a data buffer for the device.
Afterwards, the message broker checks this object net assigned to *Queues* and distributes the data to the corresponding device in firing the transition *send data to device.* To summarise, the persistence layer collects the data and associates them to the correct device, but the sending of the collected data will be performed by the message broker.
As a result, the requirement $r_{mom_{persistence_6}}$ is fulfilled.

[8] Compare 7.5 and 7.7.

7.1.3.4 The Context Interpreter as a Part of the Message Broker

As described in section 2.2.1.5, a part of the Living Place system is the Context Interpreter, a component which collects all incoming data and attempts to infer a context according to the 5W1H model[9]. For the Living Place project, the usage of a Context Interpreter is introduced in [Vos09] and [OV10], but the development of this component is still at its very beginning, so the formal modelling which is presented as visualization below, is still very simple. Its main task is to infer the appropriate context which can be used by the Alarm Clock 2.0 device which is shown in section 7.1.2.7. For that the following rules which create an appropriate 5W1H context are used by the Context Interpreter: *ruleEating* presented in 7.54, $ruleRelaxing_1$ in 7.55, $ruleRelaxing_2$ in 7.56, $ruleWashing$ in 7.57, $ruleSleeping_1$ in 7.58, $ruleSleeping_2$ in 7.59, $ruleDressing$ in 7.60 and *ruleOutside* presented in 7.61.
The transition *conjunction of data* is able to combine two AHLI nets to one net in creating the disjoint union. This is necessary, if a rule needs to aggregate more than one information to infer a new 5W1H context, e.g. the AHOI rule *ruleDressing* is dependent on two kinds of data stored within an AHLI net. Consequently it needs a disjoint union of the necessary two AHLI nets assigned to $DataToInferContext : AHLINets$ as input net in order to infer the correct 5W1H context.
The rule $ruleTransform$ as shown in 7.62 is used to convert the AHLI net within the token assigned to $ContextInterpreterQueue : AHLINets$ of the type $: GlobalData$ which is a triple of $Data \times Topic \times ConnectionID$ to data of the type $: LocalData$ which describes a triple of $Data \times Topic \times Timestamp$. This rule will be applied after new data is put to the place *ContextInterpreterQueue* by the message broker system in order to remove the connection identification and replace it by the current timestamp provided by the place $Time : AHLINets$. The current time is put to the place in firing the transition *set time* and assigning the current time as value to the variable *currentTimeNet*. It is assumed, that the correct timestamp will always be assigned to this place. The general problem of time in AHLI Nets will be discussed in chapter 10.
After converting all incoming data, the transformed token will be assigned to $DataToInferContext : AHLINets$. As next step, one of the rules mentioned can be applied which directly convert the data within the token represented by an AHLI net to 5W1H context data. Each rule will only be applied to new tokens, meaning, the timestamp of the token is less than one minute old, i.e. the timestamps of the token assigned to $DataToInferContext$ is identical with the timestamp assigned to $Timer$. Initially, the timer is set to 0 as visualized in the object net *timer*.
The rules will be described in their corresponding sections. After applying one of these rule the transformed token will be assigned to $DataBufferRepository : AHLINets$ in order to enable the message broker to distribute the resulting context information.

In the model presented below, the main concept of a Context Interpreter is followed in providing a component which creates complex 5W1H context data out of simple data sent by all devices. But in contrast to the ideas presented by Dey, Abword and Sabler in [DAS99] and [SDA99], the Context Interpreter in the Living Place system is currently very simple, not operating on different levels, instead it infers a complex context directly with the help of the mentioned rules.

[9] The 5W1H context is mentioned in section 2.2.1.5 and described in detail in [JW05].

7.1.3.4.1 Infer new Results with the Help of the Context Interpreter

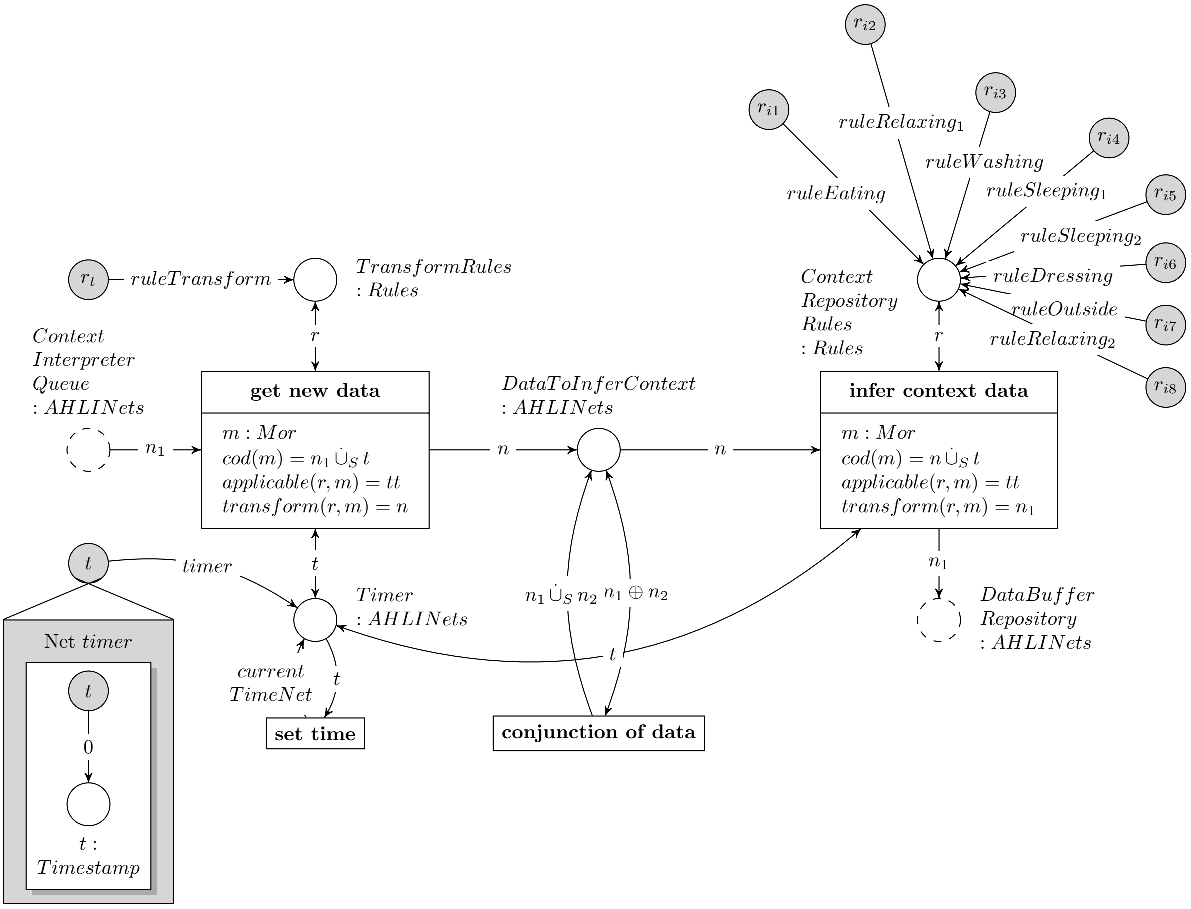


Figure 7.53: Subnet $contextInterpreter_{messageBroker}$

7.1.3.4.2 Rule ruleEating

The rule *ruleEating* converts a data token of the type : *LocalData* with the value $(middle, UbiTV)$ to a 5W1H context with the values:

- who = user
- where = kitchen
- when = undefined
- what = eating
- how = undefined
- why = undefined

Because of the current model of the Living Place system only providing two sensor devices: the pressure sensors in the Intelligent Bed[10] and the Indoor-Positioning-System[11], the context needs to be inferred out of very few data. Additionally the current model of the system is not able to provide all relevant information, like *how*. Because the Living Place system is developed for only one person, the component *who* will be static and always containing the only inhabitant internally named *user*. The resulting context have to be converted to the type : *GlobalData* in order to be able to be sent and processed by the message broker, therefore, the context will be put to the triple: *(<user, kitchen, undefined, eating, undefined, undefined >, Context, 0)*, where *<user, kitchen, undefined, eating, undefined, undefined>* is typed as *Data*, *Context*

[10] See 7.1.2.10.

[11] See 7.1.2.11.

is the name of the topic and 0 is a connection identification. Because of all data of the type : *GlobalData* need a connection identification and the Context Interpreter does not establish any connection to the message broker in the conventional way, it does not contain any specific identification, so the connection identification is filled with 0. The connection identifications of all devices will be greater than 0.

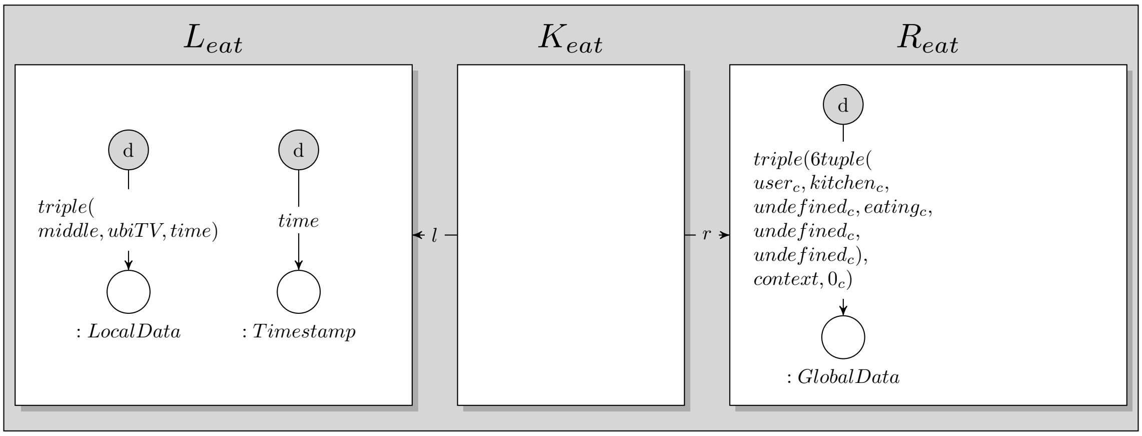


Figure 7.54: Rule *ruleEating*

7.1.3.4.3 Rules ruleRelaxing$_1$ and ruleRelaxing$_2$

Similar to *ruleEating* in 7.54, the rule *ruleRelaxing*$_1$ infers the token of the type : *LocalData* which contains the tuple $(north, UbiTV)$ to the triple *(<user, loungeArea, undefined, relaxing, undefined, undefined>, Context, 0)* containing the 5W1H context, that the only person *user* is situated in the lounge area and is possibly relaxing.
The second rule *ruleRelaxing*$_2$ infers *(<user, sleepingArea, undefined, relaxing, undefined, undefined>, Context, 0)* out of the tokens $(awake, Bed)$.

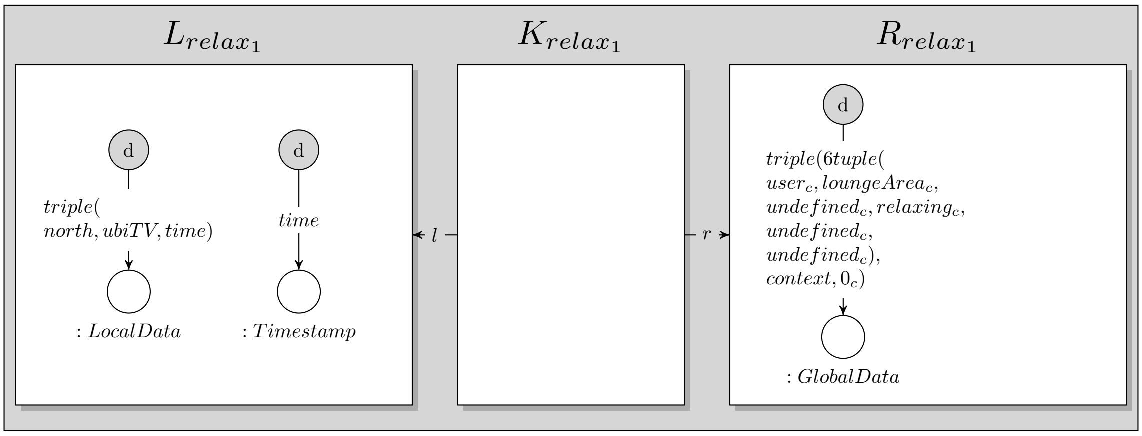


Figure 7.55: Rule *ruleRelaxing*$_1$

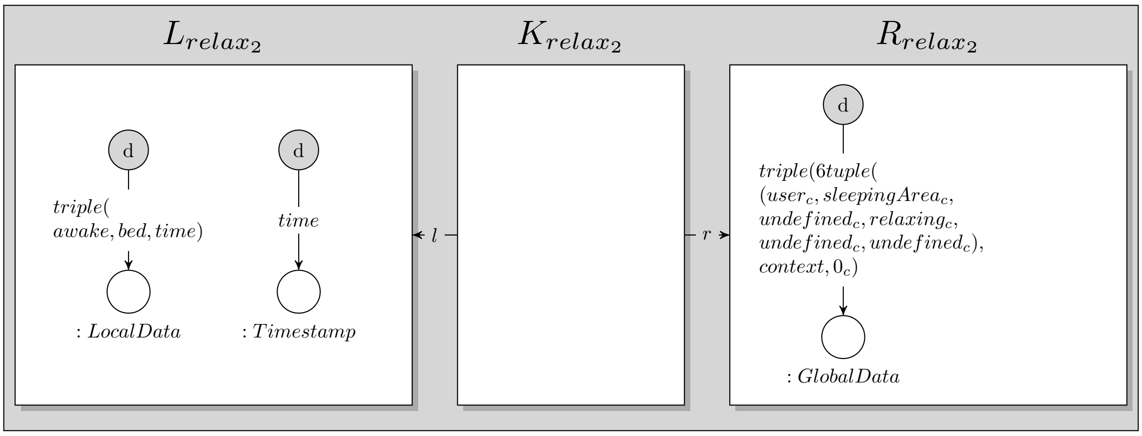


Figure 7.56: Rule $ruleRelaxing_2$

7.1.3.4.4 Rule ruleWashing

The rule $ruleWashing$ transforms a tuple of the kind : *LocalData* to a triple *(<user, bathroom, undefined, washing, undefined, undefined>, Context, 0)* which implies the 5W1H context describing that the person is currently in the bathroom and perhaps washing himself.

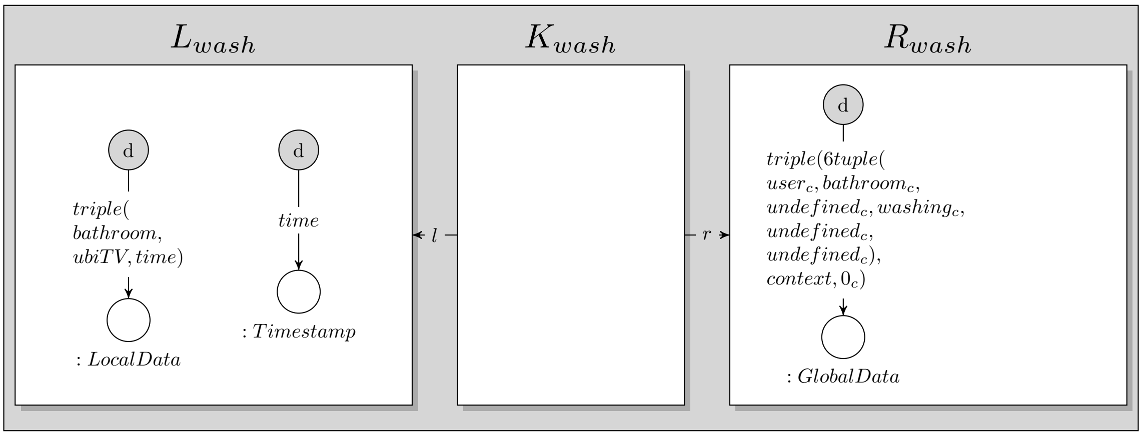


Figure 7.57: Rule $ruleWashing$

7.1.3.4.5 Rule ruleSleeping$_1$ and ruleSleeping$_2$

To recognize a 5W1H context regarding the user being in bed sleeping, the Intelligent Bed sensor device needs to send information regarding the topic *Bed*. A sleeping person is either in deep or in light sleep phase. Therefore the Intelligent Bed sends data regarding both sleeping stages and both stages need to be infered to get an appropriate context. For that, the rules *ruleSleeping*$_1$ and *ruleSleeping*$_2$ are available.
Both rules need a token of the type : *LocalData*, indeed *ruleSleeping*$_1$ needs a token which contains the information ($deepSleep, Bed$) and *ruleSleeping*$_2$ the token providing the information ($lightSleep, Bed$). Both rules infer a triple describing the 5W1H context of the kind : *GlobalData* with: *(<user, sleepingArea, undefined, sleeping, undefined, undefined>, Context, 0)*, where <user, sleepingArea, undefined, sleeping, undefined, undefined> is the 5W1H context, *Context* is the corresponding topic and 0 is a pseudo connection identification provided within messages generated by the Context Interpreter.

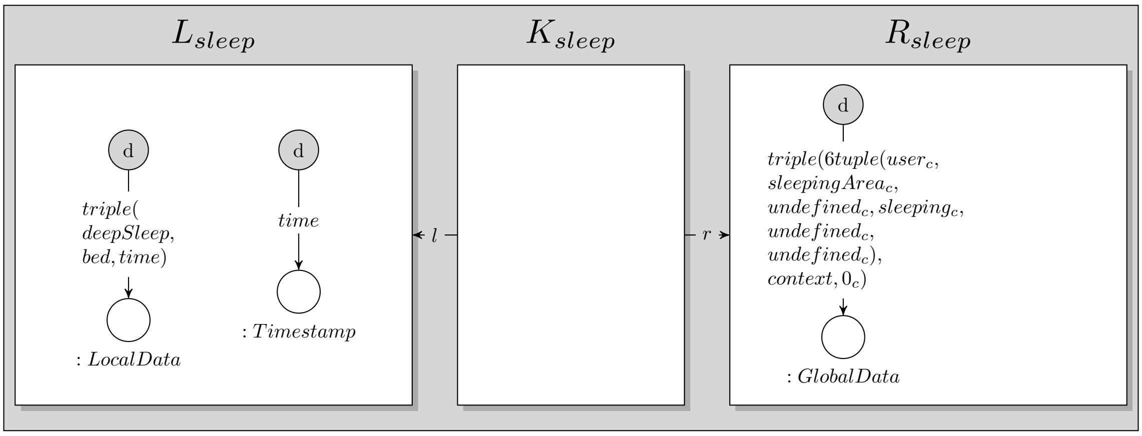


Figure 7.58: Rule $ruleSleeping_1$

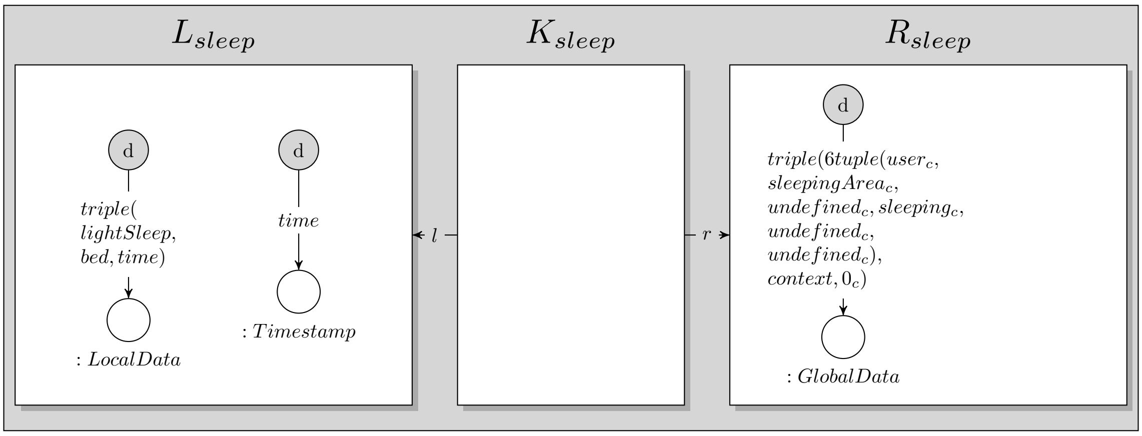


Figure 7.59: Rule $ruleSleeping_2$

7.1.3.4.6 Rule ruleDressing

The rule *ruleDressing* needs both kinds of sensor systems: The Indoor-Positioning-System and the Intelligent Bed. The Indoor-Positioning-System sends data of the topic $UbiTV$ and the Intelligent Bed sends information of the topic *Bed*. Both kinds of information are necessary to recognize a person in the sleeping area who is not sleeping, instead possibly dressing himself. On the one hand, the rule needs a token $(notInBed, Bed)$ assigned to a place of the type : *LocalData* and on the other hand a token $(south, UbiTV)$ assigned to another place of the type : *LocalData*. If this condition is met, a new net consisting of a place : *GlobalData* and a token assigned to this place with the value *(<user, sleepingArea, undefined, dressing, undefined, undefined>, Context, 0)* can be inferred.

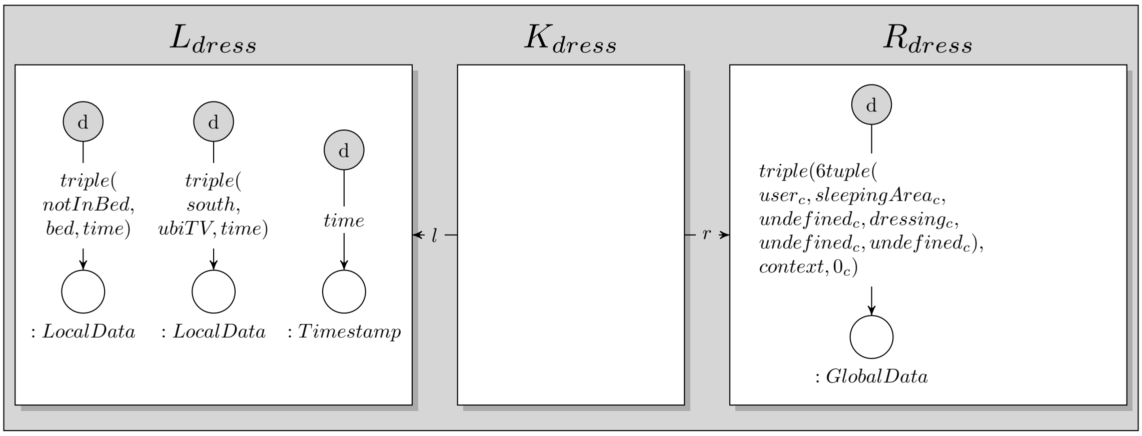


Figure 7.60: Rule $ruleDressing$

7.1.3.4.7 Rule ruleOutside

If the person has left the apartment, the Indoor-Positioning-System is not able to locate the inhabitant in the apartment. Then an information containing the token $(nothing, UbiTV)$ will be distributed to the Context Interpreter. Then the resulting data *(<user, outsideApartment, undefined, undefined, undefined, undefined>, Context, 0)* can be inferred, meaning that no information about the inhabitant is available except that he is not in the apartment.

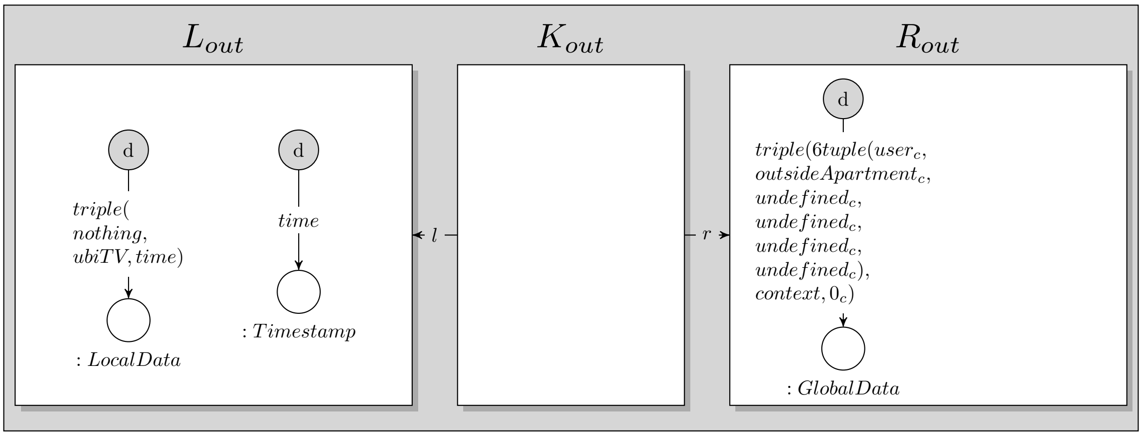

L_{out}
K_{out}
R_{out}
d
triple(nothing, ubiTV, time)
: LocalData
d
time
: Timestamp
l
r
d
triple(6tuple(user_c, outsideApartment_c, undefined_c, undefined_c, undefined_c, undefined_c), context, 0_c)
: GlobalData

Figure 7.61: Rule *ruleOutside*

7.1.3.4.8 Rule ruleTransform

As already mentioned in 7.53, the rule $ruleTransform$ transforms the AHLINet containing a place : *GlobalData* with a token which describes a triple (x, y, z) assigned to this place to a new AHLI net containing a place : *LocalData* with a token assigned to the place which pictures a triple $(x, y, time)$. This rule is necessary for removing the connection identification because it is not relevant anymore but instead it is replaced by the current timestamp, when inferring a new context.

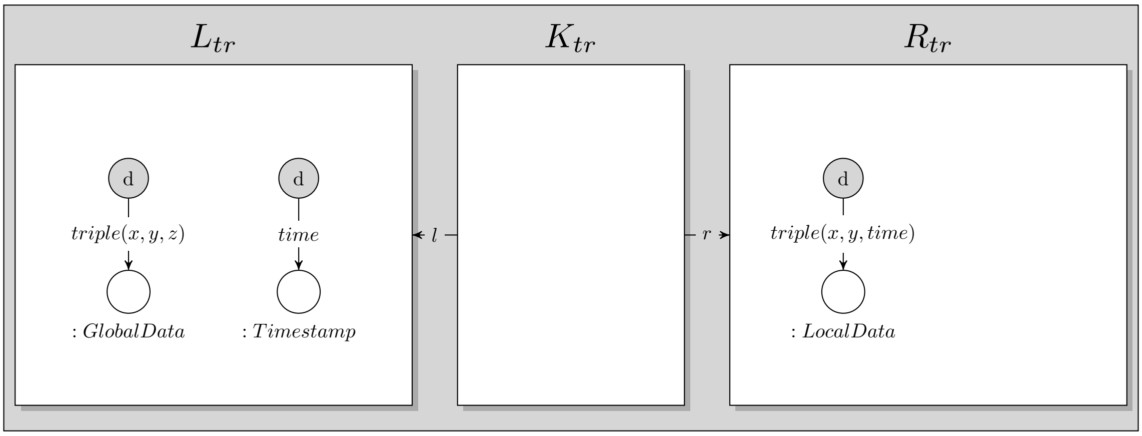


Figure 7.62: Rule $ruleTransform$

7.1.3.4.9 Revisiting the Requirements Towards the Context Interpreter

In this section, the requirements, presented in 4.2.3.3 will be revised regarding their fulfilement by the model of the Context Interpreter. The Context Interpreter is part of the net on System Level which is illustrated in 7.26.

$\mathbf{r_{mom_{ci_1}}}$: Every message sent by the devices to the message broker are directly copied and forwarded to the Context Interpreter in firing the transition *copy data to queues and to context interpreter* within the message broker (see subnet $copyDataToQueuesAndToContextInterpreter_{messageBroker}$ illustrated in 7.35). After firing, the message will be transmitted to the place $ContextInterpreterQueue : AHLINets$.

Therefore, every information will be forwarded to the Context Interpreter, which satisfies the requirement $r_{mom_{ci_1}}$.

$\mathbf{r_{mom_{ci_2}}}$: The model of the Context Interpreter includes an internal timer, whereas the current time is part of the object net assigned to the place $Timer$: $AHLINets$. When firing the transition *get new data*, each message will be transformed in adding the timestamp of the moment of the receipt. Afterwards, when firing *infer context data* and thus applying a rule for infering new information, the current timestamp will be compared with the timestamp set within the message assigned as token to $DataToInferContext$: $AHLINets$. If the timestamps are identical, the token will be used for the inference.
Consequently, only actual data will be taken for the inference process, so the requirement $r_{mom_{ci_2}}$ is met.

$\mathbf{r_{mom_{ci_3}}}$: The rules assigned to the place *ContextRepositoryRules* : *Rules* infer new 5W1H context data. These rules are presented in 7.54, 7.55, 7.56, 7.57, 7.58, 7.59, 7.60 and 7.61. So, the requirement $r_{mom_{ci_3}}$ is fulfilled.

$\mathbf{r_{mom_{ci_4}}}$: Some rules infer a 5W1H context out of an aggregation of different data tokens, e.g. the rule *ruleDressing* infers new data out of combination of two different data. When firing the transition *conjunction of data*, two data will be combined. Thus, this requirement $r_{mom_{ci_4}}$ is met.

$\mathbf{r_{mom_{ci_5}}}$: After infering a new 5W1H context, the resulting message which belongs to the topic *Context* will be assigned as token to the place $DataBufferRepository$: $AHLINets$. This place contains all data which need to be processed by the message broker. Therefore, the new message containing the 5W1H context is ready to get distributed to all devices subscribed to *Context* by the message broker. The requirement $r_{mom_{ci_5}}$ is fulfilled.

To summarise, all requirements set up for the Context Interpreter are fulfilled by the model which is illustrated in image 7.53.

7.1.3.5 AHLI Rules and Interaction Schemes Mediating Between User and Object Level Supporting User Interaction With The Object Level

The Living Place system is modeled on the subsequent levels: data, object, system and User Level. All devices are placed on the Object Level, but in contrast user interaction is only possible on the User Level, therefore the User Level provides rules for user interaction. To perform an action on a device, the System Level supplys AHLI rules to mediate between user and Object Level. Consequently the user is able to change the behaviour of the devices in applying an AHOI rule which uses AHLI rules which change a net on Object Level.

7.1.3.5.1 AHLI Rules for Stopping the Alarm Clock 2.0

To stop the Alarm Clock 2.0, the AHOI rule $StopAlarmClock_{user}$ will be applied which transforms the Alarm Clock 2.0 device on Object Level in using the AHLI rules presented in this section.
Concurrent with stopping, the Alarm Clock 2.0 device will be resetted. For that the following steps are necessary:

- Reset the appointment which is assigned to the place $NextAppointments$: $CalendarData$, in order to remove the active process of waking up resp. reminding. Because the place $NextAppointments$ is only able to contain one token at one time, this token will be removed and a new token $(0, unknown)$ will be assigned in transforming the Alarm Clock 2.0 device with the help of the rule *ruleDeleteNextAppointments* presented in 7.63.

- The bed data token will be resetted in removing the active tokens assigned to the place *Bed* : *Data* from the device net and set an initial token *unknown* to that place in applying the rule *ruleDeleteBedData* (see 7.64).

- The place *Context* : *ContextData* within the net $alarmClock2.0_{online}$ will be cleared in applying an aggregated rule generated out of the interaction scheme *ruleDeleteContextData* (see 7.65) and a new token *(unknown, unknown, unknown)* will be assigned to this place to reset the context data.

- In order to delete the active wake up timestamp, the rule $ruleDeleteWakeUpTime$ will be applied to the Alarm Clock 2.0 device net. Now, the device is not able to remind the user of any appointments any more.
- During the preparation time, the user needs to perform several tasks: end sleeping, dressing, eating and washing. A 4-tuple containing the necessary tasks for the preparation phase for an appointment, will be assigned to the place $Person : PersonState$ in the device net. This token needs to be reset in using the rule *ruleDeletePersonState* (see 7.68), which deletes the current token and assigns the tuple *(undefined, undefined, undefined, undefined)* to this place meaning, that no tasks need to be fulfilled.
- The waiting times because of the current traffic or weather conditions need to be resetted in the Alarm Clock 2.0 device in deleting the token assigned to $AddTime : Timestamp$ and set a new token to this place refering to 0 minutes meaning no extra time assumed because of the weather and traffic conditions. The rule *ruleResetTimestamp* changes this token (see 7.69).
- The synchronise token will be reset in applying the AHLI rule *ruleResetSync*. It removes the token assigned to the place $WakeUpMode : Sync$, which redirects incoming context data for analysing the context information if a token is set to this place, and a new synchronize token will be assigned to the place *RemoveContext* : *Sync*, which removes incoming context data, because the Alarm Clock 2.0 should be inactive after resetting the device net (see 7.70).

The corresponding AHOI rule is illustrated in section 7.1.4.5 and the object net of the Alarm Clock 2.0 in 7.1.2.7.

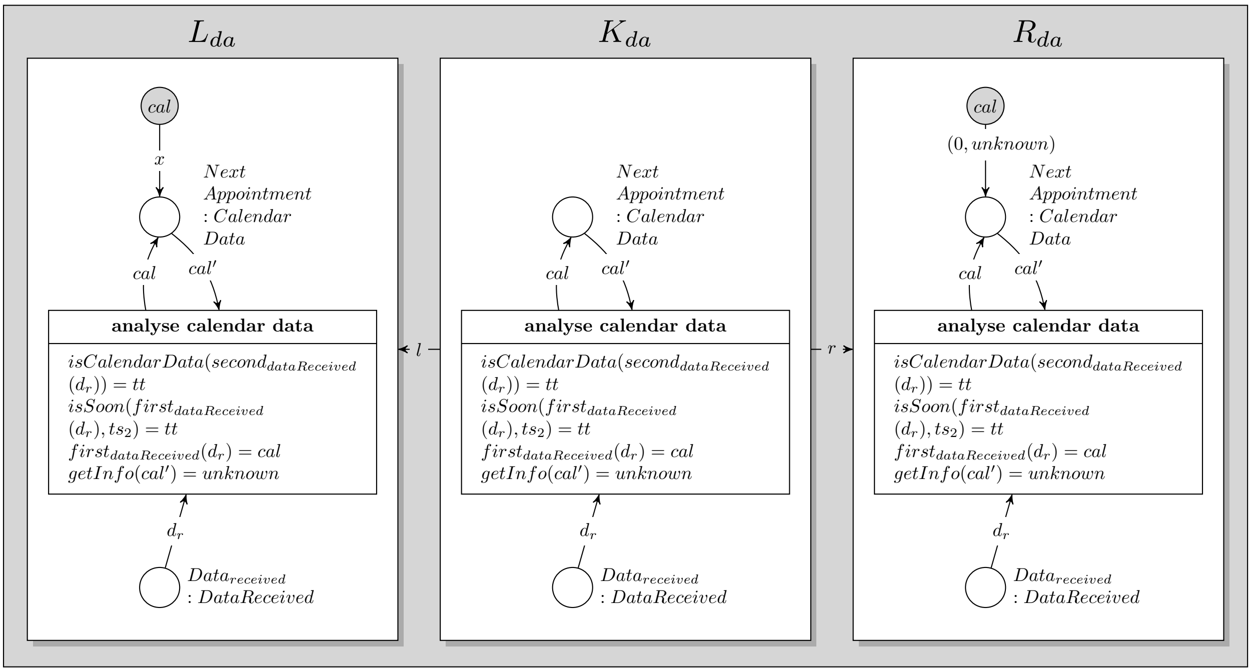

L_{da}
K_{da}
R_{da}
cal
x
Next Appointment : Calendar Data
cal
cal'
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
d_r
$Data_{received}$: DataReceived
l
r
(0, unknown)

Figure 7.63: User interaction rule *ruleDeleteNextAppointments*

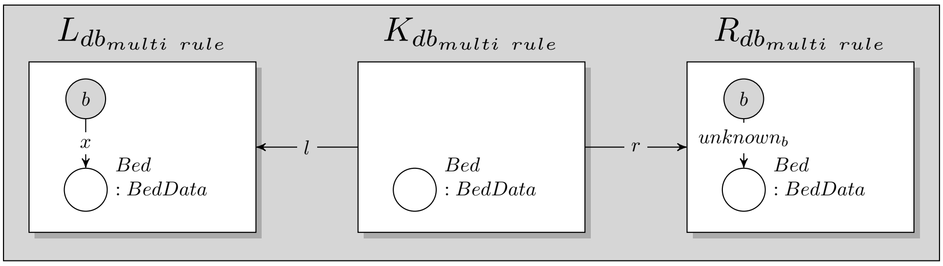


Figure 7.64: User interaction rule *ruleDeleteBedData*

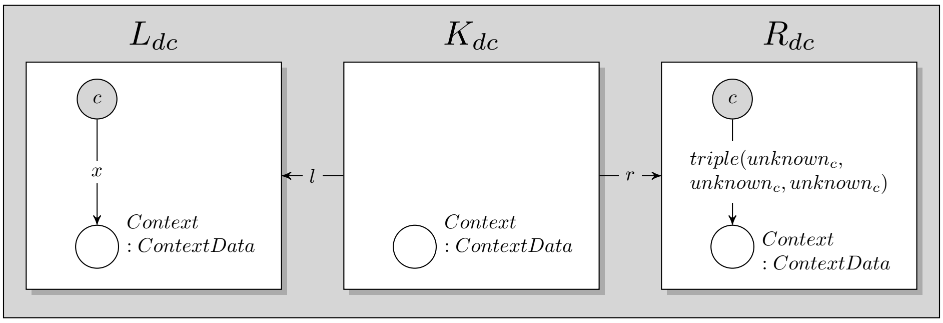


Figure 7.65: User interaction rule *ruleDeleteContextData*

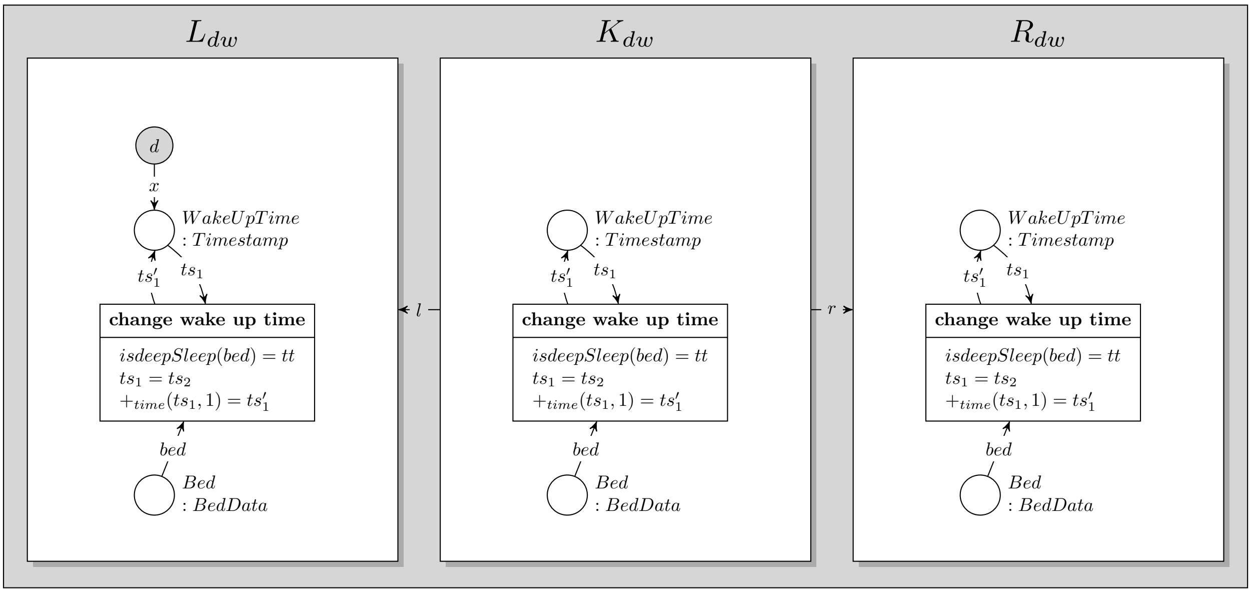


Figure 7.66: User interaction rule $ruleDeleteWakeUpTime_1$

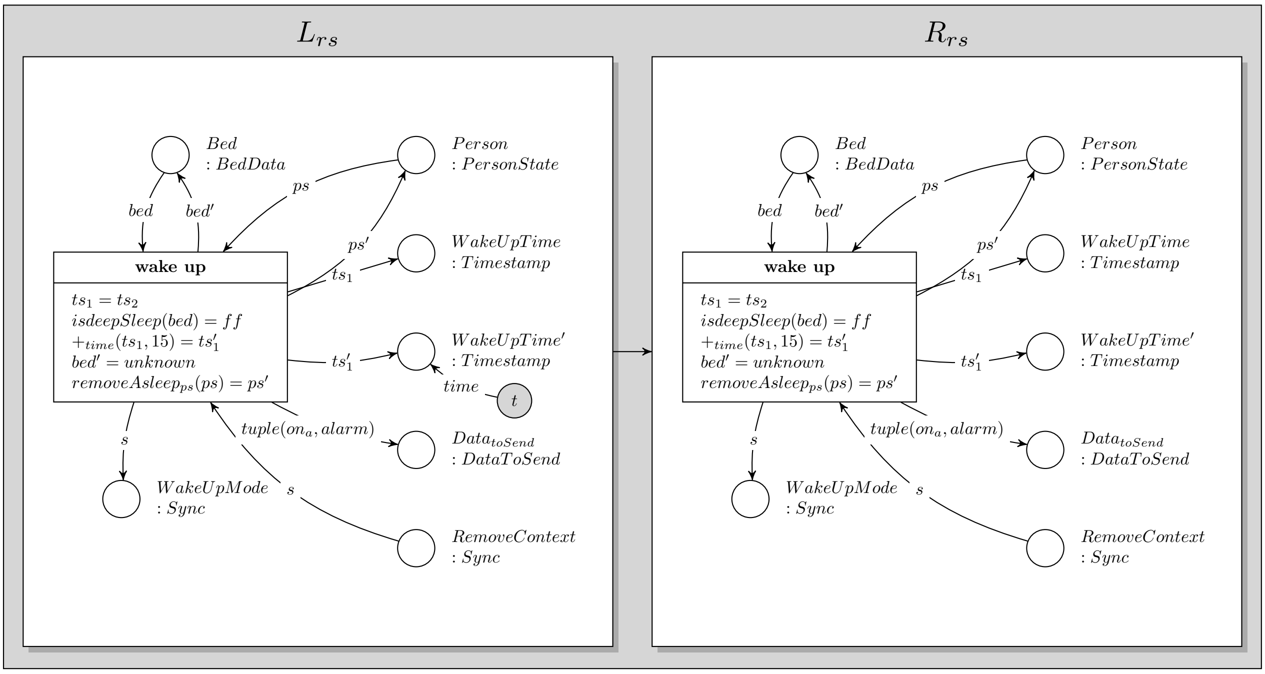


Figure 7.67: Rule $ruleDeleteWakeUpTime_2$

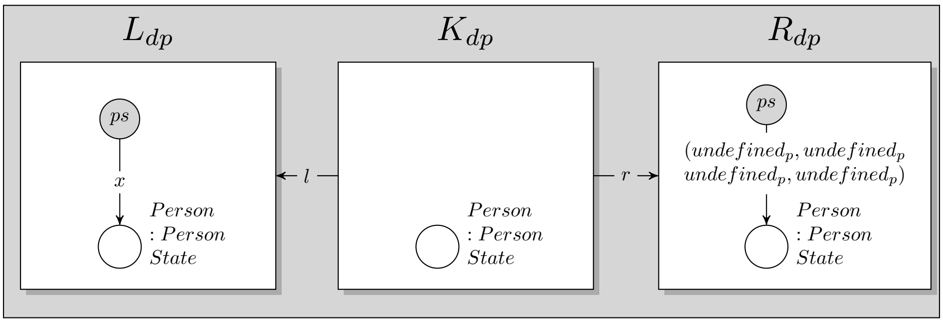


Figure 7.68: User interaction rule *ruleDeletePersonState*

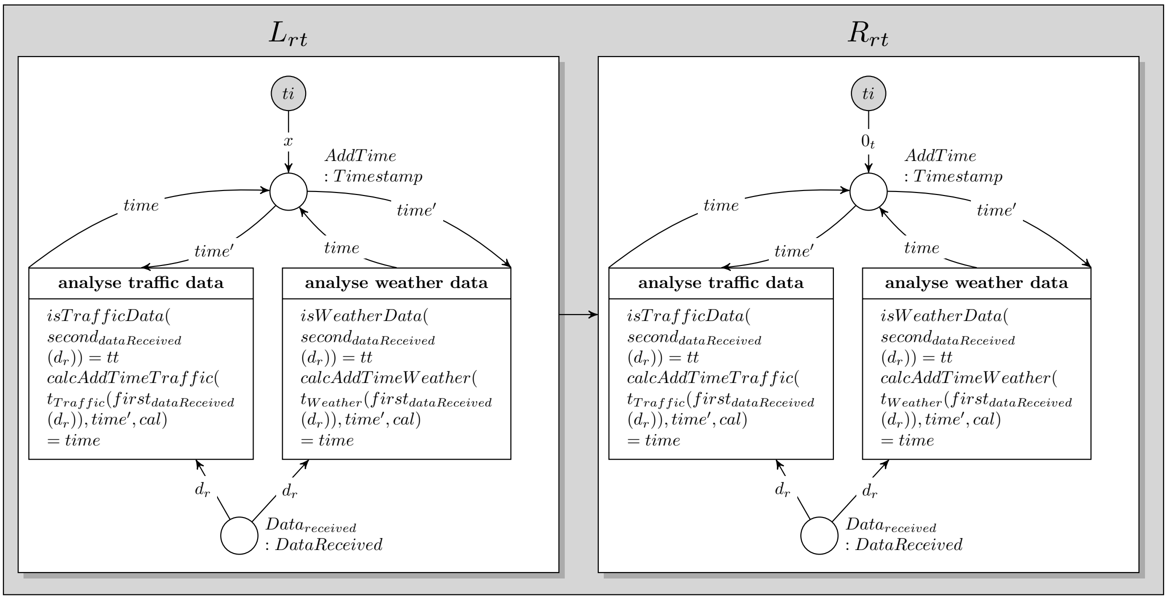


Figure 7.69: Rule *ruleResetTimestamp*

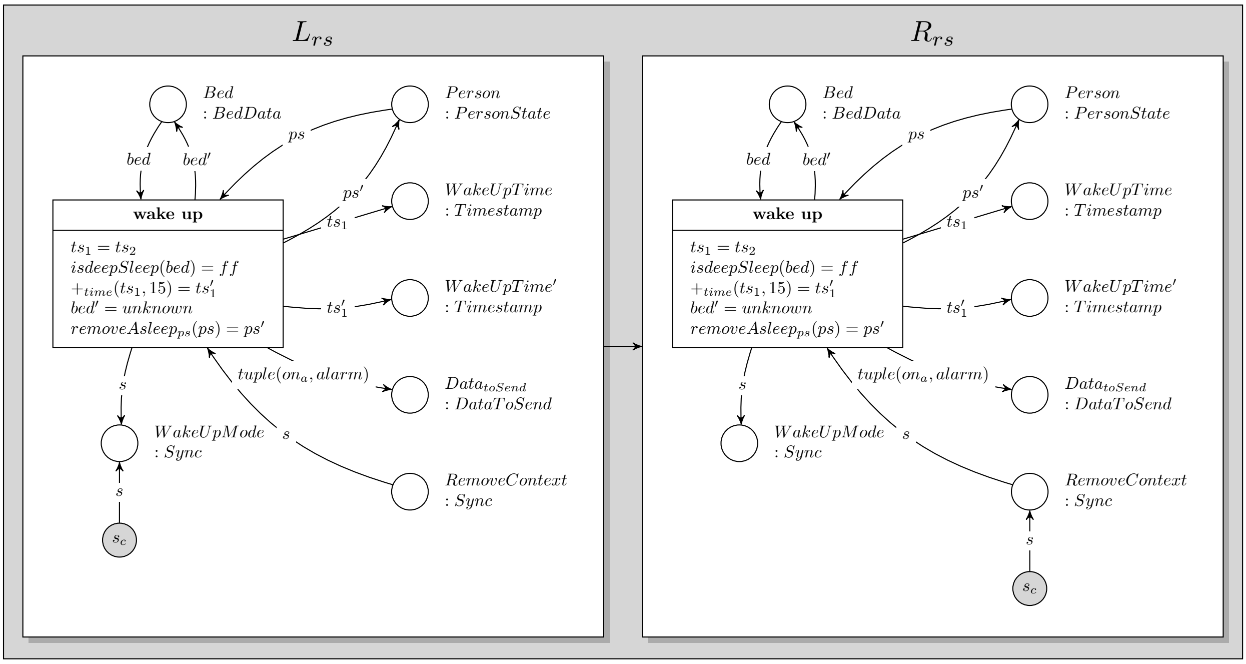


Figure 7.70: Rule *ruleResetSync*

7.1.3.5.2 Clear Display

In 7.1.2.8 the Display device of the Multitouch Kitchen Counter is presented which is required for Displaying calendar data entered by the resident of the Living Place apartment. Additionally it receives and displays all incoming traffic and weather data.
Old data need to be removed manually from the Display in applying an AHOI rule $ClearDisplay_{user,d}$, where d is a the data displayed on the screen (see 7.1.4.6.1). The Living Place system contains an infinite set of AHOI rules dependent on the data d specified by the user. According to each AHOI rule, a corresponding AHLI rule is available, which will be assigned by the AHOI rule to the place *EnterDataIntoDeviceRules* : *Rules*. The corresponding AHLI rule $ruleClearDisplay_d$ removes the given Display data d from the place *display* : *DisplayData* in the device net of the Display in the Multitouch Kitchen Counter in Object Level, which is described in 7.1.2.8, if a match can be found.
Because of an infinite set of AHOI and AHLI rules for removing data, both kinds of rules are presented more general with an indice d, standing for all available AHLI rules and as a consequence additionally for all corresponding AHOI rules
E.g. the AHOI rule $ClearDisplay_{user,rain}$ refers to the AHLI rule $ruleClearDisplay_{rain}$ which removes the data token $rain$ from the place *display* : *DisplayData* within the device net $display_{online}$ of the Display in the Multitouch Kitchen Counter.

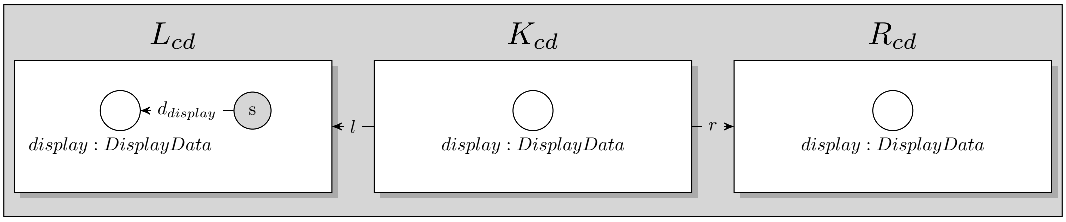


Figure 7.71: Rule $ruleClearDisplay_d$

7.1.3.5.3 Enter Data to Daily Planner

In section 7.1.2.9 the Daily Planner was introduced. It requires calendar data which will be entered by the inhabitant of the Living Place. For that the an infinite set of AHOI transformation rules $EnterDataIntoDailyPlanner_{user,t,d}$, where t describes a timestamp and d is the description of an appointment, are available (see 7.1.4.7). Each AHOI rule assigns a corresponding AHLI transformation rule $ruleAddData_{planner,t,d}$ containing the values (t, d) which should be entered to the Daily Planner. Because of an infinite set of AHOI rules and consequently an infinite set of AHLI rules, a general description of the rule will be taken in the following model: Each rule contains the indices representing the specific rule. Each AHLI rule $ruleAddData_{planner,t,d}$ assigns a token (t, d) to the place *newCalendarData* : *CalendarData* to the device net, if a match was found. After that, the device net on Object Level is able to send the new appointment data to the message broker of the Living Place system. In general, the object net of the Daily Planner is only able to operate, if the AHOI rules $EnterDataIntoDailyPlanner_{user,t,d}$ and their corresponding AHLI rules $ruleAddData_{planner,t,d}$ are available.
E.g. $EnterDataIntoDailyPlanner_{user,12000,Work}$ describes an AHOI rule which assigns the AHLI rule $ruleAddData_{planner,12000,Work}$ to the place *EnterDataIntoDeviceRules* : *Rules* in the system layer, afterwards in applying this AHLI rule, a new token $(12000, Work)$ will be assigned to the place *newCalendarData : CalendarData* within the object net of the Daily Planner.

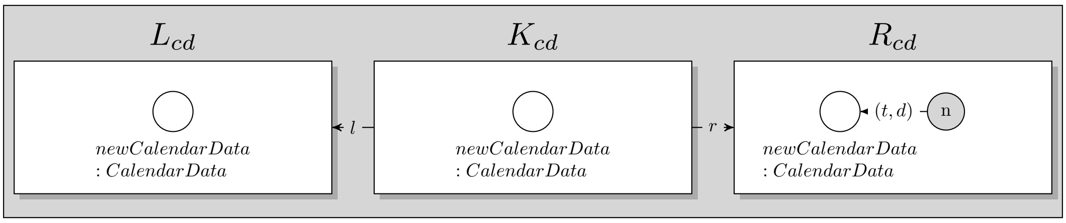


Figure 7.72: Rule $ruleAddData_{planner,t,d}$

7.1.3.5.4 Remove Single Data From Datastore in the Persistence Layer With the Daily Planner

The inverse to the AHLI rule $ruleAddData_{planner,t,d}$ presented in last section 7.1.3.5.3, is the infinite set of AHLI rules presented in this section. The AHLI rule $ruleRemoveData_{system,d,t,id}$ needs a corresponding AHOI rule $DeleteDataFromDailyPlanner_{user,d,t,id}$ (see 7.1.4.7), which assigns this AHLI rule to the place $EnterDataIntoDeviceRules : Rules$. The indices d, t and id, (where d is the description of an appointment, t is the timestamp and id the connection id of the previously stored message (i.e. the connection id of the Daily Planner)), denotes a specific AHOI rule and its corresponding AHLI rule which transforms an object net as follows: The AHLI rule assigns a token $(d, t, id, remove)$ to the place $Data_P : DataForPersistenceLayer$ in the device net of the Daily Planner representing the data to delete as 4-tuple, e.g. $(< 12000, Work >, Calendar, 1, remove)$. The last element of the 4-tuple *remove* signals, that the described data should be removed from the persistence layer. The deletion of data from the persistence layer will be performed when firing the transition *store data in persistence layer or remove data.*

In applying the AHOI rule $DeleteDataFromDailyPlanner_{user,d,t,id}$ the inhabitant of the Living Place is able to delete single data from the persistence layer with the help of $ruleRemoveData_{system,d,t,id}$.

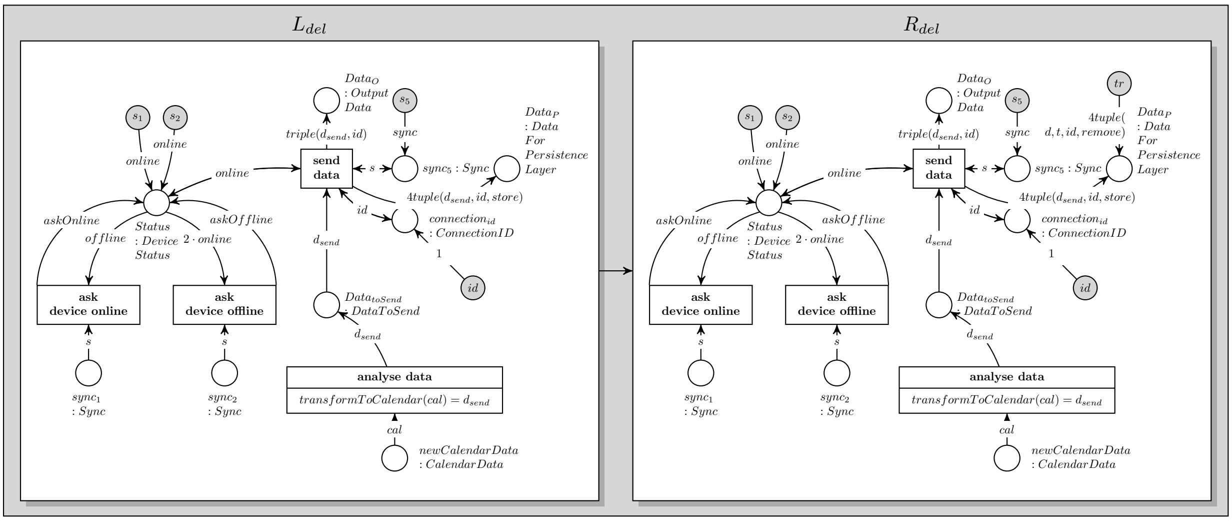


Figure 7.73: Rule $ruleRemoveData_{system,d,t,id}$

7.1.3.5.5 Measure Pressure in the Intelligent Bed

In order to operate, the Intelligent Bed needs data from the six pressure sensors, as described in detail in 7.1.2.10. The pressure data will be entered in using an AHOI rule $MeasurePressureInBed_{user,v_1,v_2,v_3,}$ $_{v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, where $v_i, x_i \in \mathbb{R} \wedge v_i, x_i \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}$ (see 7.1.4.8) which represents all pressure data for all six sensors as input values. The AHOI rule assigns a corresponding AHLI rule $ruleMeasurePressure_{bed,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, where $v_i, x_i \in \mathbb{R} \wedge v_i, x_i \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}$, which takes the pressure data and assigns a token to the place $Sensor : PressureData$ in the AHLI device net $intelligentBed_{online}$ in Object Level, if a possible match was found.
Afterwards the new pressure data is able to be processed by the device net in Object Level.
The indices $v_1, v_2, v_3, v_4, v_5, v_6, x_1, x_2, x_3, x_4, x_5$ and x_6 in the system and its corresponding AHOI rule represent all available rules for each possible v_i, x_i in the given range $v_i, x_i \in \mathbb{R} \wedge x_i \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}$.

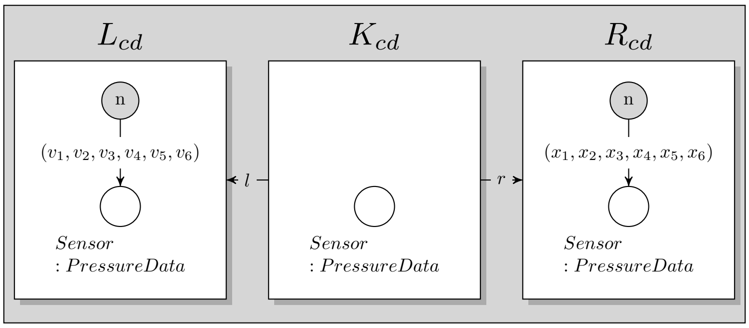


Figure 7.74: Rule $ruleMeasurePressure_{bed,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$

7.1.3.5.6 Measure Position of the User in Living Place Apartment with the Indoor-Positioning-System

Similar to the measurement of pressure sensor data of the Intelligent Bed, the user position for the Indoor-Positioning-System, which is presented in section 7.1.2.11, gets its 3-dimansional position data with the help of an AHOI rule $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, where $v_i = (X_{v_i}, Y_{v_i}, Z_{v_i}), x_i = (X_{x_i}, Y_{x_i}, Z_{x_i}) \wedge X_{v_i}, Y_{v_i}, Z_{v_i}, X_{x_i}, Y_{x_i}, Z_{x_i} \in \mathbb{R} \wedge X_{v_i}, Y_{v_i}, Z_{v_i}, X_{x_i}, Y_{x_i}, Z_{x_i} \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}$ (see 7.1.4.9) and a corresponding AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ containing position data for all six sensors, in order to generate new position data for the device net $indoorPositioningSystem_{online}$ of the Indoor-Positioning-System. The AHLI rule is shown below. Each component in the tuple consists of a triple with 3-dimensional data (X, Y, Z).

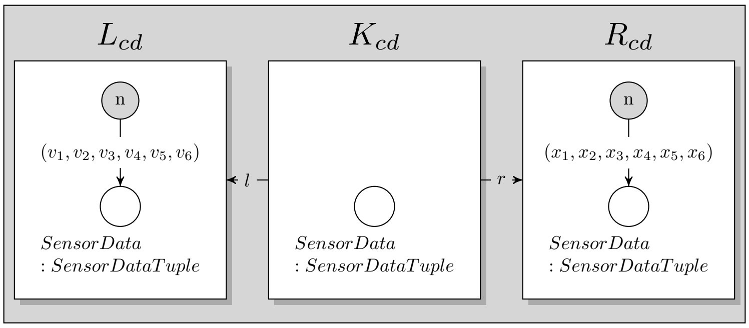

L_{cd}
K_{cd}
R_{cd}
n
$(v_1, v_2, v_3, v_4, v_5, v_6)$
l
r
n
$(x_1, x_2, x_3, x_4, x_5, x_6)$
SensorData : SensorDataTuple
SensorData : SensorDataTuple
SensorData : SensorDataTuple

Figure 7.75: Rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$

7.1.3.5.7 Switch on Location-Based Screen

The Location-Based Screen includes two TV sets which need to be switched on or off by the inhabitant of the Living Place apartment. The device net of the Location-Based Screen in Object Level is presented in section 7.1.2.12.
Both TV sets will be switched on or off simultaneously by the resident, because both TV sets work alternating depending on the current position of the resident: If the resident stays in the lounge area, the TV set in this area is active, whereas the TV set in the sleeping area is active if the person sojourns in the sleeping area. Both TV sets will be inactive, if the person is out of view of both TV sets, i.e. the person is in the middle, the bathroom or outside of the apartment. Being active means, that a TV set shows the TV program, if it is switched on. An inactive TV set does not show any TV program even if it is powered on. If disabled, no TV set will show anything even if the inhabitant stays in the corresponding area. Consequently, both TV sets will be switched on or off simultaneously but will show the TV program alternating or nothing.

To switch on both TV sets, the user is able to apply the AHOI rule $SwitchOnTV_{user}$ for powering on or use $SwitchOffTV_{user}$ for powering off. Both AHOI rules are presented in 7.1.4.10. They apply the corresponding AHLI rules $ruleSwitchOn_{TV}$ resp. $ruleSwitchOff_{TV}$ which change the on/off state of both TV sets simultaneously. The AHLI rules are illustrated in the following images 7.76 and 7.77.

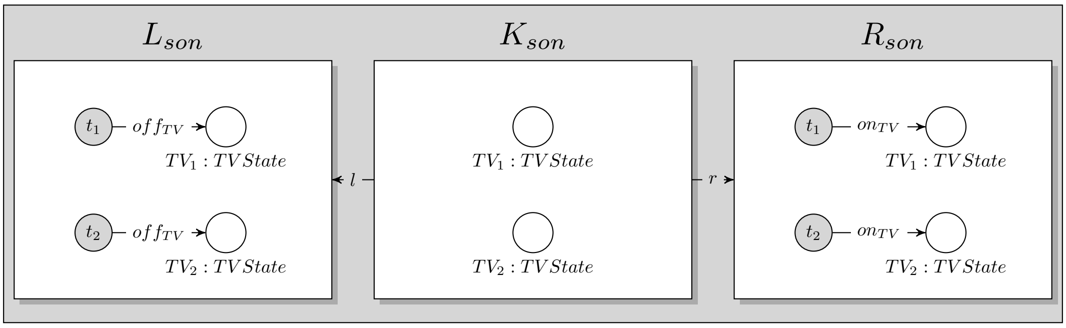


Figure 7.76: Rule $ruleSwitchOn_{TV}$

7.1.3.5.8 Switch off Location-Based Screen

The AHLI rule $ruleSwitchOff_{TV}$ is the opposite to $ruleSwitchOn_{TV}$ and is described in detail in the previous section.

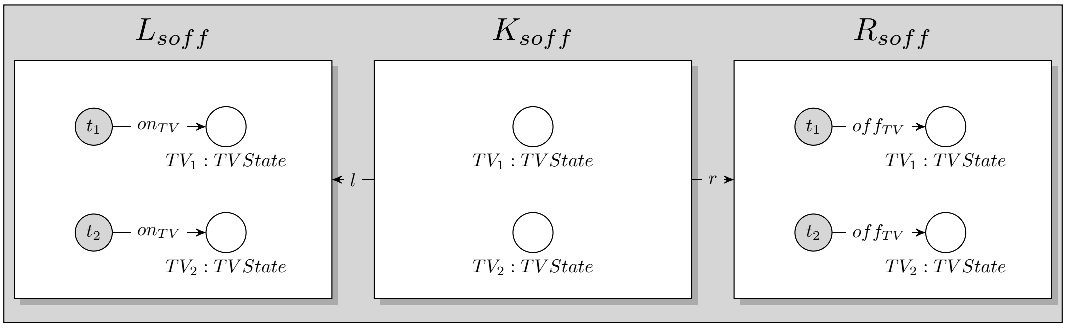


Figure 7.77: Rule $ruleSwitchOff_{TV}$

7.1.3.5.9 Switch on Light

The model of the Ambient Light, as introduced in section 7.1.2.13, needs user interaction to switch on or off the whole light in the Living Place apartment. For this purpose the AHOI rules $SwitchOnLight_{user}$ for switchin on the light and $SwitchOffLight_{user,l}$, where $l \in \{bright, dimmed\}$ for disabling the Ambient Light are available (see 7.1.4.11). All AHOI rules change the state of the light with the help of the AHLI rules $ruleSwitchOn_{Light}$ resp. $ruleSwitchOffLight_l$, where $l \in \{bright, dimmed\}$.

Switching the light on sets a token *bright* to the place $Light : LightState$ in the device net on Object Level, meaning that the light in the apartment is switched on in full brightness. In contrast, the AHLI rule $ruleSwitchOffLight$ replaces any kind of token assigned to the place *Light* by a token referring to the value off representing a disabled light.

Both AHOI rules get an AHLI device net as input parameter, but a match can only be found if the entered device net contains a place of the type *LightState*, which is only possible in the current modelling if the input device net is a net $light_{online}$ as shown in section 7.1.2.13.

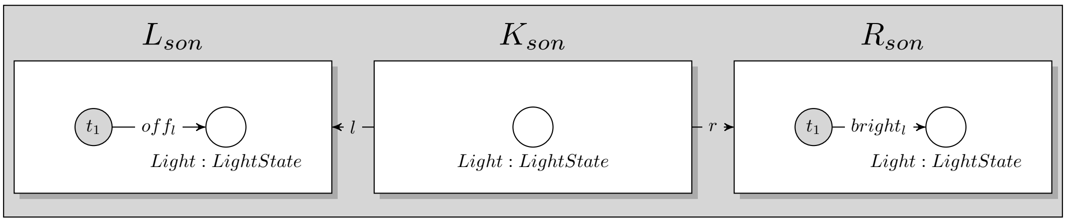


Figure 7.78: Rule $ruleSwitchOn_{Light}$

7.1.3.5.10 Switch off Light

The reverse of the rule $ruleSwitchOn_{Light}$ is the rule $ruleSwitchOffLight_l$ presented below, where $l \in \{bright, dimmed\}$.

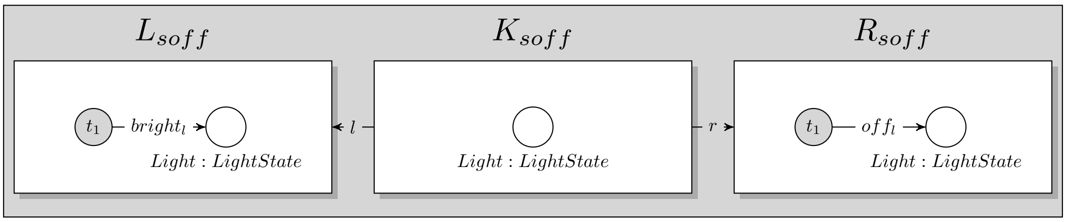


Figure 7.79: Rule $ruleSwitchOff_{Light,bright}$

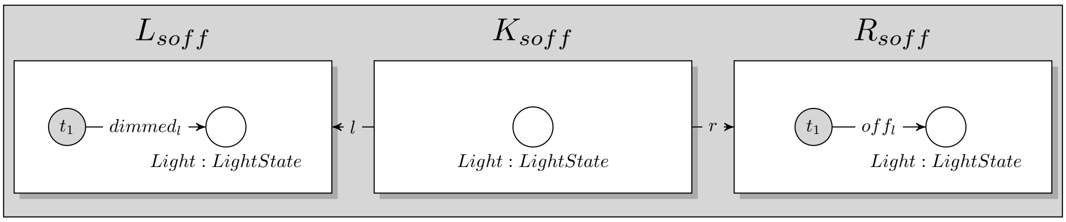


Figure 7.80: Rule $ruleSwitchOff_{Light,dimmed}$

7.1.4 User Level

The User Level provides the possibility for the user to interfere in the procedure of the Living Place system in applying one of the rules mentioned below. As already described, all transformation rules are only able to change the System Level meaning the user is able to intervene in the message broker and beyond change the behaviour of the whole system including the system, object and the data net.

7.1.4.1 Rules for the Administration of the Devices of the Living Place System

The inhabitant of the Living Place apartment is able to administer all devices of the Living Place system in applying one of the following three AHOI rules:

- $SetDeviceOffline_{user}$ for changing the device mode into offline mode.
- $PluginDevice_{user}$ for adding a device in offline mode to the Living Place system. Then, each device will be in offline mode and assigned to the place $OfflineDevices : AHLINets_{OFFD}$ in the net of the message broker on System Level.
- $UnplugDevice_{user}$ for removing a device in offline mode from the whole Living Place system.

The following use case presents the AHOI rules for administering the Living Place system.

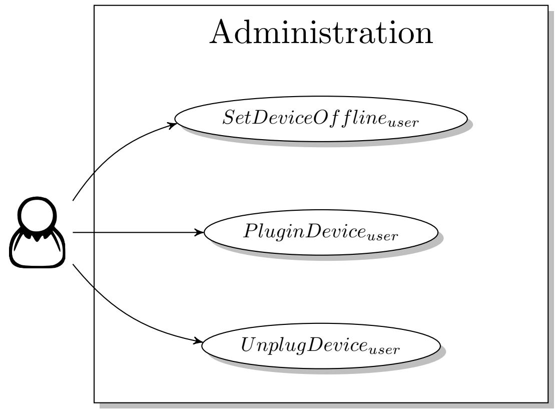


Figure 7.81: Use Case for the Administration of the Living Place System

7.1.4.1.1 User Disconnects Device

For disconnecting a device manually in setting it into offline mode, the user needs to announce the connection identification of the device, which should be set into offline mode, to the message broker to initiate the disconnection of this device. The user applies the AHOI rule $SetDeviceOffline_{user,nConnectionID}$, which contains the indice *nConnectionID* standing for an AHLI net containing the connection id of a device net on Object Level, which should be set into offline mode. The Living Place system contains many concrete AHOI rules $SetDeviceOffline_{user,nConnectionID}$. In this work, a template for an AHOI rule is presented, where *nConnectionID* represents an AHLI net including the connection identification of the device which should be set into offline mode. The rule sets a new token representing the connection identification to remove to the place $UnplugConnectionIDs : AHLINets$. The net which is assigned as token to this place is illustrated in image 7.37. Afterwards the message broker is able to unplug the device in firing the transition *ask device to go offline* described in the subnet 7.38.

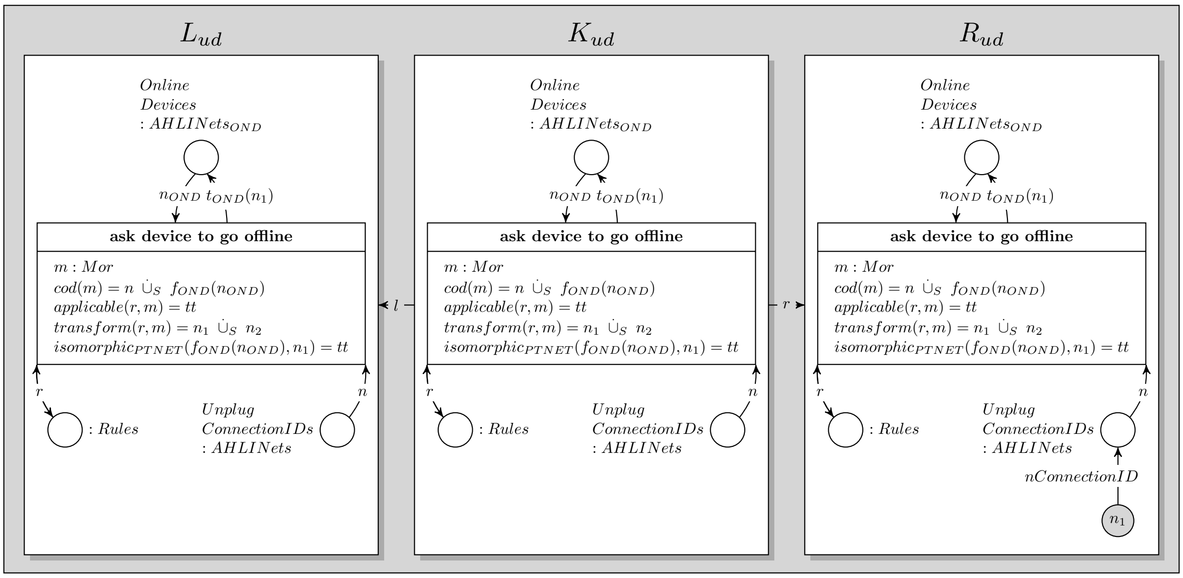

L_{ud}
K_{ud}
R_{ud}
Online Devices : $AHLINets_{OND}$
n_{OND} $t_{OND}(n_1)$
ask device to go offline
$m : Mor$
$cod(m) = n \dot{\cup}_S f_{OND}(n_{OND})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
r
n
: Rules
Unplug ConnectionIDs : AHLINets
l
r
nConnectionID
n_1

Figure 7.82: AHOI rule $SetDeviceOffline_{user,nConnectionID}$

7.1.4.1.2 User Adds a Device

To add a new device to the Living Place system, a new device net will be assigned by the user as token to the place $OfflineDevices : AHLINets_{OFFD}$ in executing the rule $PluginDevice_{user,y}$. The Living Place system includes many AHOI rules of the kind $PluginDevice_{user,y}$, where the indice y represents an AHLI device net. Concrete AHOI rules are for example: $PluginDevice_{user,light_{offline}}$ or $PluginDevice_{user,alarmClock2.0_{offline}}$, etc.
This rule takes the device net in offline mode and assigns it as token to the place $OfflineDevices$:

$AHLINets_{OFFD}$.

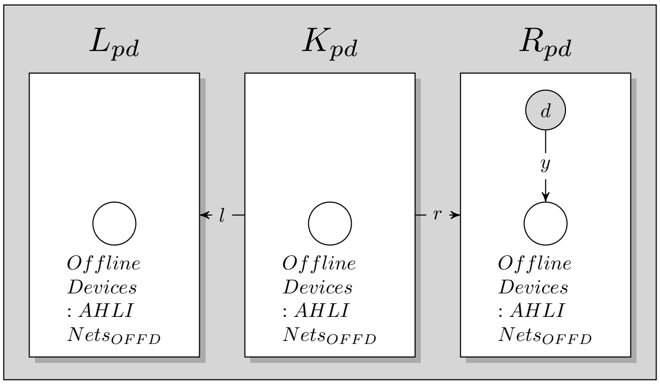

L_{pd}
K_{pd}
R_{pd}
l
r
d
y
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$

Figure 7.83: AHOI rule $PluginDevice_{user,y}$

7.1.4.1.3 User Removes a Device

The reverse to the rule $PluginDevice_{user,y}$ is the following rule $UnplugDevice_{user,y}$. Thereby, a device net in offline mode represented by a token assigned to $OfflineDevices : AHLINets_{OFFD}$ will be removed from this place.

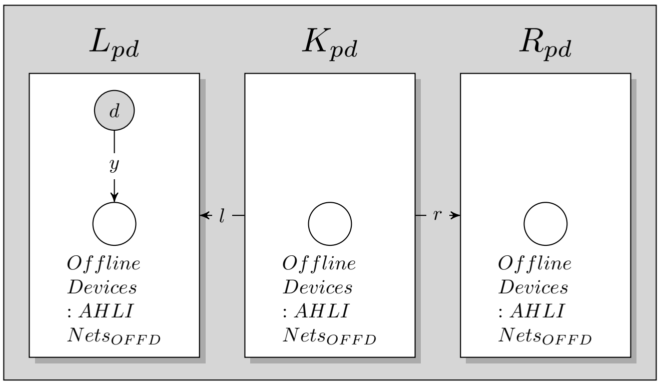

L_{pd}
K_{pd}
R_{pd}
d
y
l
r
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$
$Offline$ $Devices$ $: AHLI$ $Nets_{OFFD}$

Figure 7.84: AHOI rule $UnplugDevice_{user,y}$

7.1.4.2 Interaction Schemes for Setting the Whole Living Place System Offline in Emergency

Another part of the administration of the whole Living Place system is the possibility of setting the whole system in offline mode in case of an emergency. Then two steps are necessary whose interaction schemes are enumerated in the following use case.

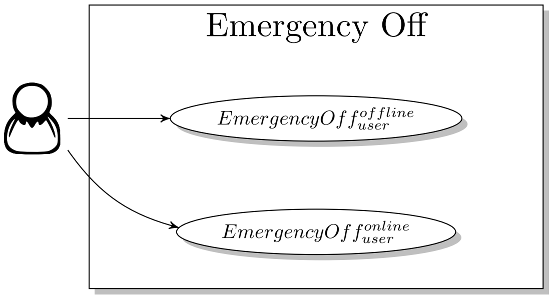

Emergency Off
$EmergencyOff^{offline}_{user}$
$EmergencyOff^{online}_{user}$

Figure 7.85: Use Case for Emergency Off of the Living Place System

7.1.4.2.1 User Interaction Schemes

The subsequent AHOI interaction schemes $EmergencyOff_{user}^{online}$ and $EmergencyOff_{user}^{offline}$ are analogously. In applying the amalgamated rules generated out of these interaction schemes, the user deletes all device nets which are assigned to $OnlineDevices : AHLINets_{OND}$ resp. $OfflineDevices : AHLINets_{OFFD}$.
The rule calculated out of the interaction scheme $EmergencyOff_{user}^{online}$ deletes all device nets in online mode, accordingly all device nets in offline mode will be removed from the message broker in executing a rule calculated out of the interaction scheme $EmergencyOff_{user}^{offline}$

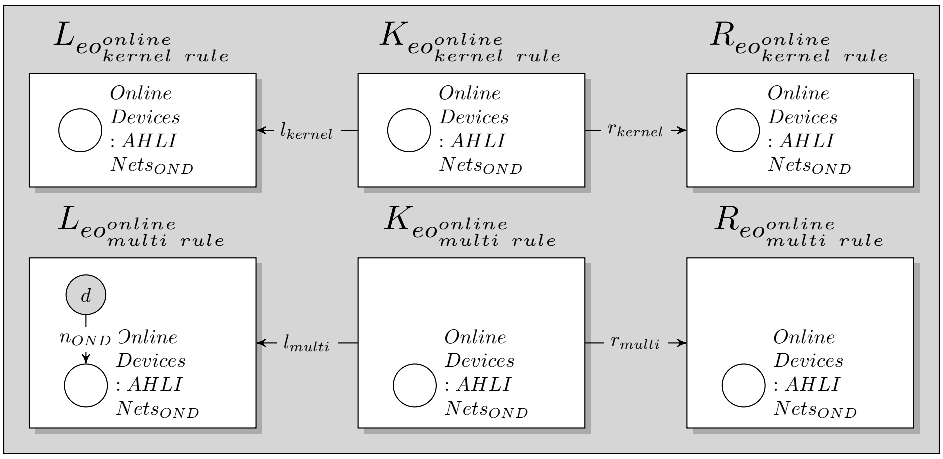

$L_{eo_{kernel\ rule}^{online}}$
$K_{eo_{kernel\ rule}^{online}}$
$R_{eo_{kernel\ rule}^{online}}$
Online Devices : AHLI Nets$_{OND}$
l_{kernel}
r_{kernel}
$L_{eo_{multi\ rule}^{online}}$
$K_{eo_{multi\ rule}^{online}}$
$R_{eo_{multi\ rule}^{online}}$
d
n_{OND}
l_{multi}
r_{multi}

Figure 7.86: User interaction scheme $EmergencyOff_{user}^{online}$

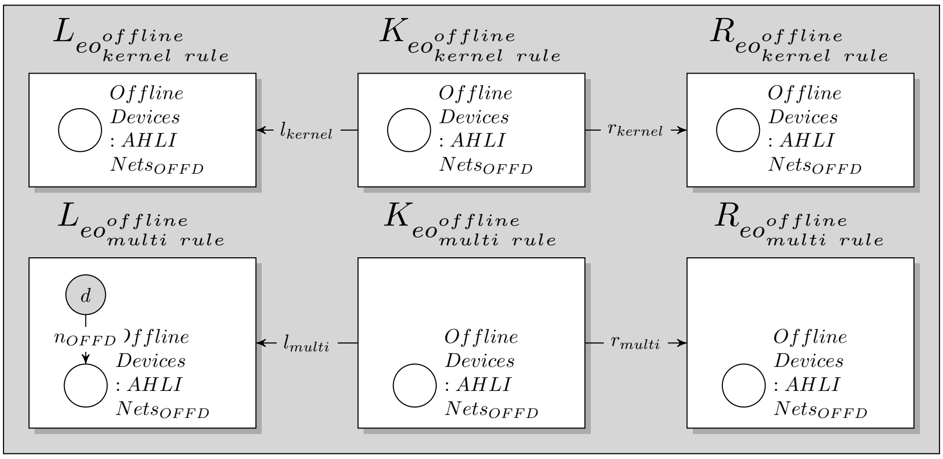

$L_{eo_{kernel\ rule}^{offline}}$
$K_{eo_{kernel\ rule}^{offline}}$
$R_{eo_{kernel\ rule}^{offline}}$
Offline Devices : AHLI Nets$_{OFFD}$
l_{kernel}
r_{kernel}
$L_{eo_{multi\ rule}^{offline}}$
$K_{eo_{multi\ rule}^{offline}}$
$R_{eo_{multi\ rule}^{offline}}$
d
n_{OFFD}
l_{multi}
r_{multi}

Figure 7.87: User interaction scheme $EmergencyOff_{user}^{offline}$

7.1.4.3 Initialising the Whole System

To initialise the whole system, all tokens assigned to the places $Queues : AHLINets_Q$, $Requests : AHLINets_{Rq}$, $ReceiveDataRules : Rules_R$ and $PersistentData : AHLINets_{Per}$ need to be deleted. For that, the following AHOI rules and user interaction schemes need to be applied by the user manually. As input parameter by the user, some rules resp. interaction schemes get the net resp. rule which should be deleted by the rule.

The use case enumerates the AHOI rules and user interaction schemes for initialising the whole Living Place system, which will be introduced in detail in this section.

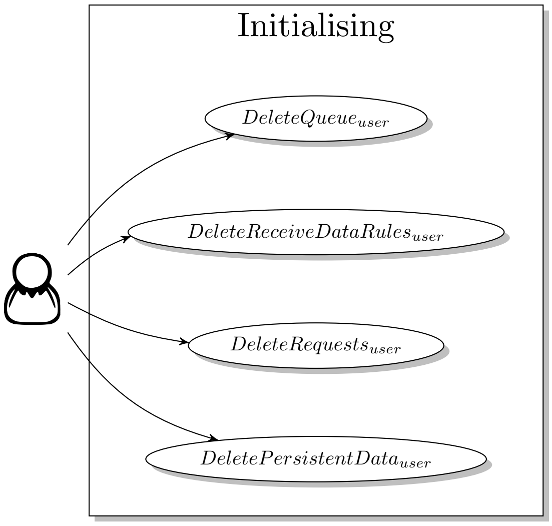

Initialising
$DeleteQueue_{user}$
$DeleteReceiveDataRules_{user}$
$DeleteRequests_{user}$
$DeletePersistentData_{user}$

Figure 7.88: Use Case for the Administration of the Living Place System

7.1.4.3.1 Delete Queues

On part of initialising the whole system is to delete all existing queues in the system net representing the message broker.
Only one token, containing an AHLI net with all available queues representing the connections and subscriptions between message broker and all devices, is assigned to the place $Queues : AHLINets_Q$, so the rule $DeleteQueue_{user}$ needs no input parameter. It resets the only token assigned to the place $Queues$.

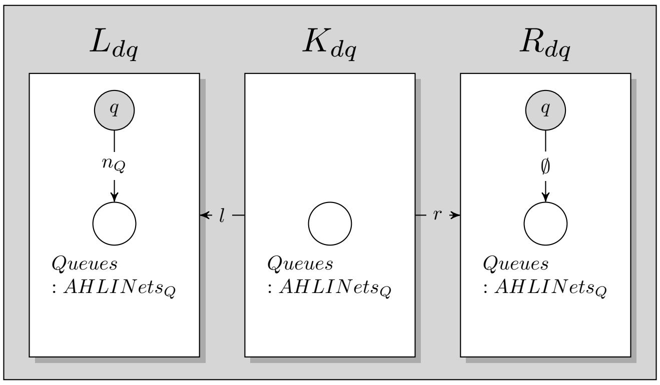


Figure 7.89: AHOI rule $DeleteQueue_{user}$

7.1.4.3.2 Delete All Rules *receiveData*

To delete all receive data rules in one step from the place $ReceiveDataRules : Rules_R$, the user applies the following interaction scheme $DeleteReceiveDataRules_{user}$.

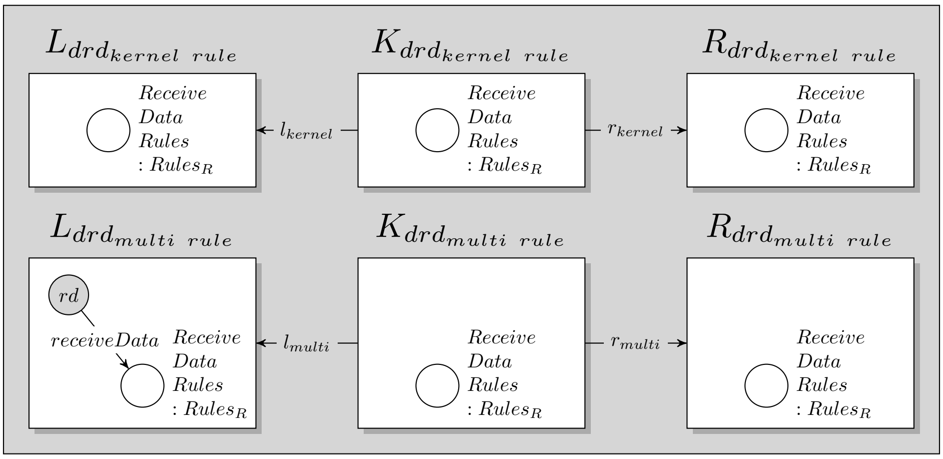


Figure 7.90: User interaction scheme $DeleteReceiveDataRules_{user}$

7.1.4.3.3 Delete Requests

Multiple request can be assigned to the place *Requests* : $AHLINets_{Rq}$ in the message broker net which have to be removed to reset the whole Living Place system. To delete all requests from the place *Requests*, the user applies the interaction scheme $DeleteRequests_{user}$.

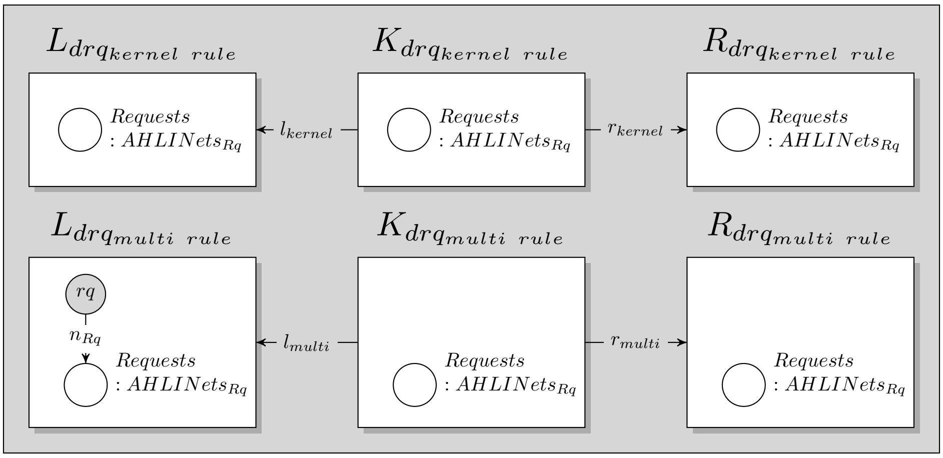


Figure 7.91: AHOI rule $DeleteRequests_{user}$

7.1.4.3.4 Delete all Data From Persistent Dataset Assigned to PersistentData

Only one token is assigned to the place *PersistentData* : $AHLINets_{Per}$ in the part of the system net representing the persistence layer, therefore the rule $DeletePersistentData_{user}$ needs no input parameter and deletes the token assigned to the place *PersistentData* and assigns a new token *emptyDataSet* to this place. The net *emptyDataSet* describes an empty dataset resp. a dataset in initial mode. The net *emptyDataSet* is defined in the Σ_{SL}-Algebra A_{SL} in 11.3.

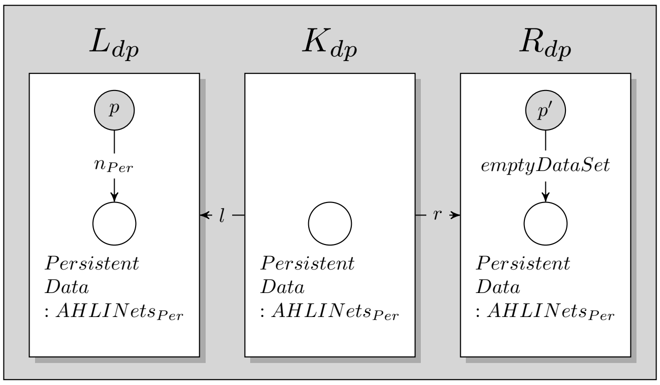

L_{dp}
K_{dp}
R_{dp}
p
n_{Per}
p'
$emptyDataSet$
l
r
$Persistent$ $Data$ $: AHLINets_{Per}$
$Persistent$ $Data$ $: AHLINets_{Per}$
$Persistent$ $Data$ $: AHLINets_{Per}$

Figure 7.92: AHOI rule $DeletePersistentData_{user}$

7.1.4.4 Revisiting the Requirements Towards the User Operations Causing a Change of the Internal State of the Message Oriented Middleware

In section 4.2.4.1 the requirements for all user operations are presented, which cause a change of the internal state of the Message Oriented Middleware. I.e. those requirements refer to the AHOI rules which directly access the System Level in order to change its internal state. In this section, these requirements will be revised regarding their fulfilment by the AHOI rules and AHOI interaction schemes presented in sections 7.1.4.1, 7.1.4.2 and 7.1.4.3. In addition, the Living Place system includes the characteristics of a ubiquitous computing system, as presented in 2.1. These characteristics lead to requirements which have to be fulfilled by the model of the Living Place system, which are mentioned in section 4.1.

$\mathbf{r_{mom_{user_1}}}$: The AHOI rule $SetDeviceOffline_{user,nConnectionID}$ assigns an object net containing a specific connection id to the place $UnplugConnectionIDs : AHLINets$. Afterwards the Message Oriented Middleware is able to take this connection id and sets the corresponding device into offline mode. Therefore, this requirement $r_{mom_{user_1}}$ is fulfilled.

$\mathbf{r_{mom_{user_2}}}$: The inhabitant is able to add a new device in applying a concrete AHOI rule of the kind $PluginDevice_{user,y}$. Then, the given device in offline mode will be assigned as token to the place $OfflineDevices : AHLINets_{OFFD}$, meaning that this device is now added to the Living Place system. Thus, the requirement $r_{mom_{user_2}}$ is met by this AHOI rule.

$\mathbf{r_{mom_{user_3}}}$: The opposite to the previous rule is the rule $UnplugDevice_{user,y}$ which removes the given device from the place $OfflineDevices : AHLINets_{OFFD}$, if it is assigned to this place, so, this device is removed from the Living Place system. Consequently, the requirement $r_{mom_{user_3}}$ is fulfilled.

$\mathbf{r_{mom_{user_4}}}$: The user interaction schemes $EmergencyOff^{online}_{user}$ and $EmergencyOff^{offline}_{user}$ are given for setting the Living Place into offline mode in case of an emergency. They delete all devices assigned to the places $OnlineDevices : AHLINets_{OND}$ resp. $OfflineDevices : AHLINets_{OFFD}$. Hence, this requirement $r_{mom_{user_4}}$ is fulfilled by the models of both interaction schemes.

$\mathbf{r_{mom_{user_5}}}$: To initialise the whole Living Place system, the following rules and interaction schemes are provided:

- $DeleteQueue_{user}$
  This rule resets the object net containing all available queues which represent the connections between the devices and the topics to which they are subscribed and which contains an internal data buffer for all messages which should be distributed to the corresponding devices. The new object net assigned as token to the place $Queues : AHLINets_Q$ is empty.

- $DeleteReceiveDataRules_{user}$
  All rules $receiveData$ will be removed from the place $ReceiveDataRules : Rules_R$. Each rule $receiveData$ enables the parallel receiving of data from all devices.

- $DeleteRequests_{user}$
  Each device is able to request past data from the persistence layer. An initialisation of the Living Place system removes all requests in applying the amalgamated rule created out of this interaction scheme.

- $DeletePersistentData_{user}$
  The object net assigned as token to the place $PersistentData : AHLINets_{Per}$ (within the persistence layer) contains all stored data processed by the message broker until now. This rule resets the object net representing the internal database with an object net containing no data.

All conditions of this $r_{mom_{user_5}}$ are fulfilled by the AHOI rules and user interaction schemes presented in chapter 7.1.4.3.

$\mathbf{r_{ubio_1}}$, $\mathbf{r_{ubis_1}}$ **and** $\mathbf{r_{ubiu_1}}$: The AHOI rules $SetDeviceOffline_{user,nConnectionID}$, $PluginDevice_{user,y}$ and $UnplugDevice_{user,y}$ are provided which enable the connection of devices to resp. disconnection of devices from the Living Place system. Each device is assigned as an own token either to the place $OfflineDevices : AHLINets_{OFFD}$ or to the place $OnlineDevices : AHLINets_{OND}$, hence they are independent from each other and will not influence each other when a device will be added to or removed from the the Living Place system. Consequently, these requirements r_{ubio_1}, r_{ubio_1} and r_{ubiu_1} are satisfied.

As a result, all requirements demanded to the AHOI rules and user interaction schemes causing a change of the internal state of the Message Oriented Middleware, are fulfilled.

7.1.4.5 Rule for Interaction With the Alarm Clock 2.0

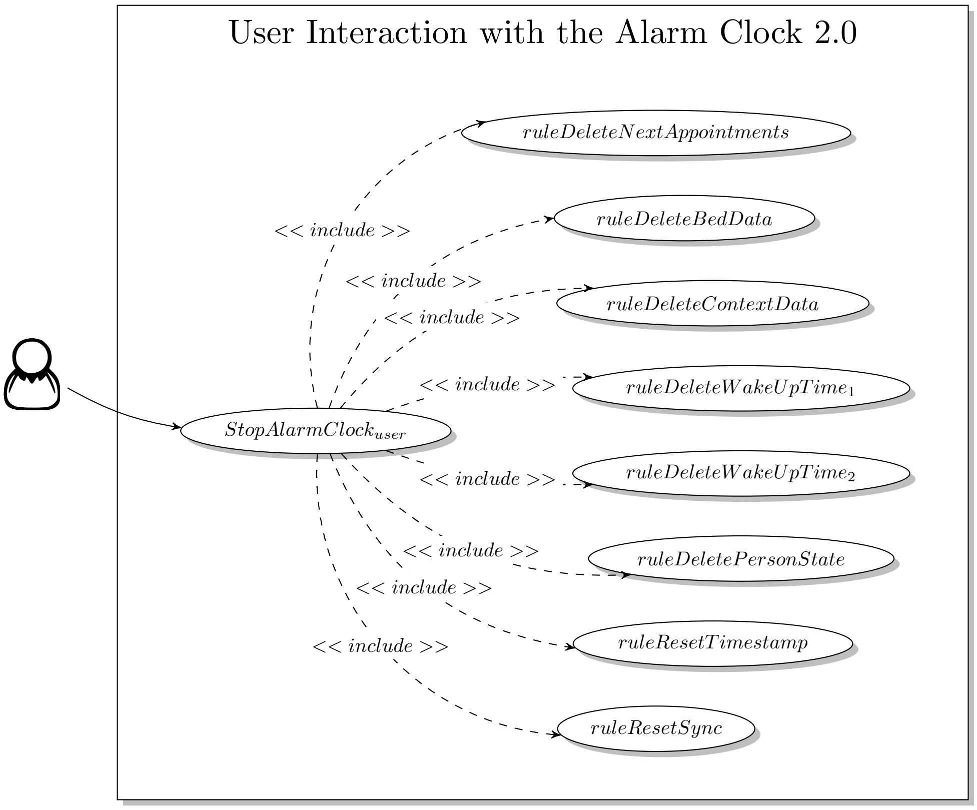


Figure 7.93: Use Case for the Interaction with the Alarm Clock 2.0

The Living Place intends that the inhabitant should have the full control over the whole system. Even if the Alarm Clock 2.0 should remind the user independently of coming appointments and tasks the user has to perform before attending the appointment, the inhabitant will be able to stop the Alarm Clock 2.0 from reminding. The AHOI rule $StopAlarmClock_{user}$ will be provided to stop the Alarm Clock 2.0.

It transforms the device net $alarmClock2.0_{online}$ in applying the following eight AHLI rules to stop the device:

- $ruleDeleteNextAppointments$, see 7.63,
- $ruleDeleteBedData$, see 7.64,
- $ruleDeleteContextData$, see 7.65,
- $ruleDeleteWakeUpTime_1$, see 7.66,
- $ruleDeleteWakeUpTime_2$, see 7.67,
- $ruleDeletePersonState$, see 7.68,
- $ruleResetTimestamp$, see 7.69 and
- $ruleResetSync$, see 7.70.

The use case presents the user interaction for stopping the Alarm Clock 2.0 (compare 7.1.2.7). The following figure illustrates the AHOI transformation rule $StopAlarmClock_{user}$.

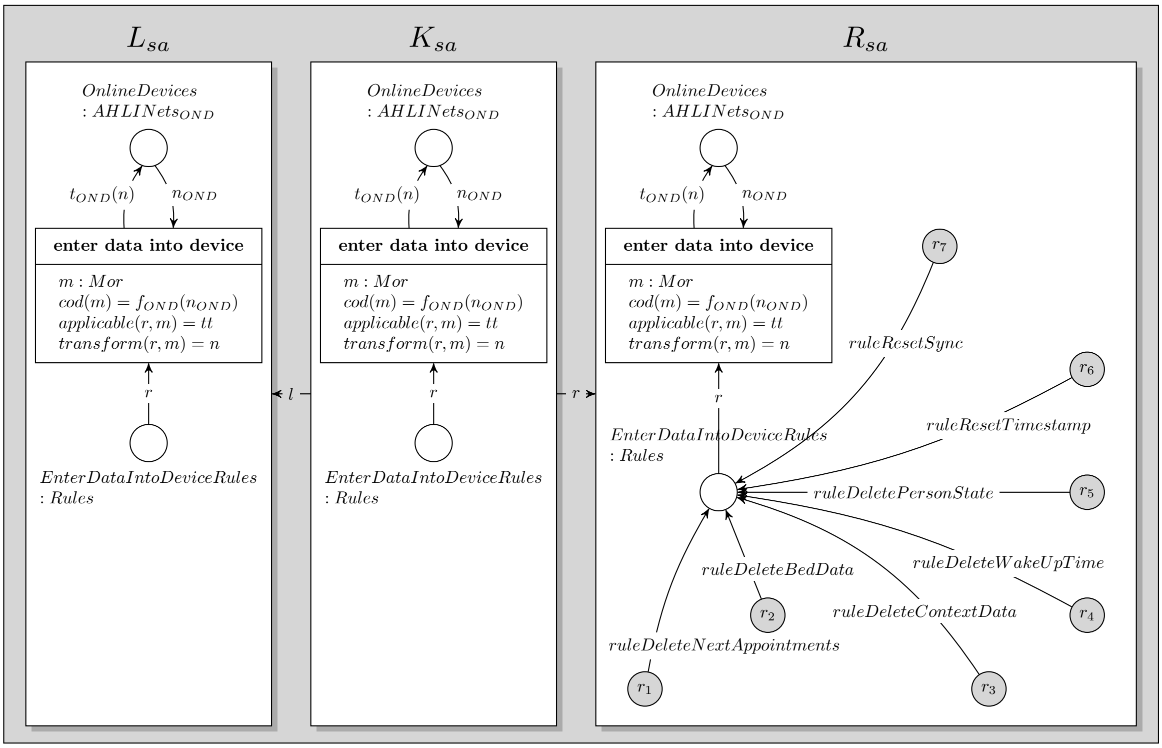


Figure 7.94: AHOI rule $StopAlarmClock_{user}$

7.1.4.5.1 Revisiting the Requirement Towards the User Operations Causing a Change of the Alarm Clock 2.0 Device

In section 4.2.4.2.1 a requirement is presented, which should be fulfilled by the user operation presented in this section 7.1.4.5. In the following this requirement will be revised in order to proove its fulfilment.

$\mathbf{r_{alarm_{user_1}}}$: The AHOI rule $StopAlarmClock_{user}$ is provided in order to stop the Alarm Clock 2.0 device assigned as token to the place $OnlineDevices : AHLINets_{OND}$ within the Message Oriented Middleware on System Level. This AHOI rule assigns the given rules to the place $EnterDataIntoDeviceRules : Rules$:

- $ruleDeleteNextAppointments$
  Removes the currently selected appointment assigned to $NextAppointment : CalendarData$.

- $ruleDeleteBedData$
  Replaces the token assigned to Bed : $BedData$ by *unknown*. So, the internally stored sleeping phase of the user is resetted.

- $ruleDeleteContextData$
  Replaces the token assigned to $Context$: $ContextData$ by the initial triple *(unknown, unknown, unknown)*, which means, that no specific context is recognized, right now.

- $ruleDeleteWakeUpTime_1$
  If a wake up time is assigned to $WakeUpTime$: $Timestamp$, then this token will be removed.

- $ruleDeleteWakeUpTime_2$
  If a wake up time is assigned to $WakeUpTime'$: $Timestamp$, then this token will be removed.

- $ruleDeletePersonState$
  The internal task list assigned as token to $Person$: $PersonState$ will be replaced by the token *(undefined, undefined, undefined, undefined).*

- $ruleResetTimestamp$
  The timestamp containing the additional waiting time because of bad weather resp. traffic conditions will be removed and the initial value of 0 will be assigned to the corresponding place $AddTime$: $Timestamp$.

- $ruleResetSync$
  If available, the synchronise token will be removed from $WakeUpMode$: $Sync$, which represents the processing mode of the Alarm Clock 2.0 device and a new token will be assigned to $RemoveContext$: $Sync$ representing the idle mode of the Alarm Clock 2.0 device, which enables the device to remove all incoming context data.

Because of the availability of these user and these AHLI rules, the requirement $r_{alarm_{user_1}}$ is fulfilled by the model.

7.1.4.6 Rules for Direct User Interaction With the Display in the Multitouch Kitchen Counter

To interact with the Display device (see 7.1.2.8) the model supplies the following AHOI rules presented in the use case and described in detail in this section.

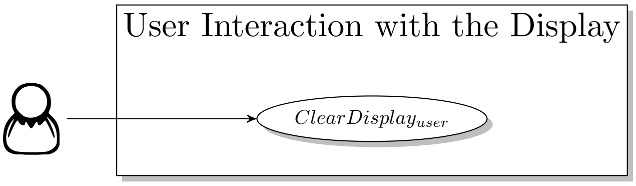


Figure 7.95: Use Case for the Interaction with the Display in the Multitouch Kitchen Counter

7.1.4.6.1 Clear Display

The Living Place will offer the opportunity that the user is able to remove selected data records from the Display of the Multitouch Kitchen Counter. The AHOI rule $ClearDisplay_{user,d}$ is provided for deleting a single data record from the Display. The AHOI resp. AHLI rule are presented as template, where the indice d should be replaced by a concrete data referring to a concrete data which should be removed. Because of a possibly infinte set of data which could be shown by the Display device, an infinite set of AHOI rules with their corresponding AHLI rules are available.

The AHOI rule assigns its corresponding AHLI rule $ruleClearDisplay_d$ to the place *EnterDataIntoDeviceRules : Rules* within the system net. The AHLI rule removes the data d from the place $Display$: $Data$ within the object net of the Display in online mode. The template of the corresponding AHLI rule is presented in 7.1.3.5.2.

7.1.4.6.2 Revisiting the Requirement Towards the User Operations Causing a Change of the Display

In section 4.2.4.2.2 one requirement $r_{Display_{user_1}}$ is presented, which should be fulfilled by the user operation which consists of a user and its corresponding AHLI rule introduced in this section 7.1.4.6.1. In the following this requirement will be revised regarding its fulfilment.

$\mathbf{r_{Display_{user_1}}}$: The AHOI rule $ClearDisplay_{user,d}$ and its corresponding AHLI rule $ruleClearDisplay_d$ removes a given data record from the place $display : DisplayData$ within the device net of the Display, which should be assigned as token to the place $OnlineDevice : AHLINets_{OND}$ within the net of the message broker.
Therefore, this requirement $r_{Display_{user_1}}$, which is demanded to the model, is met.

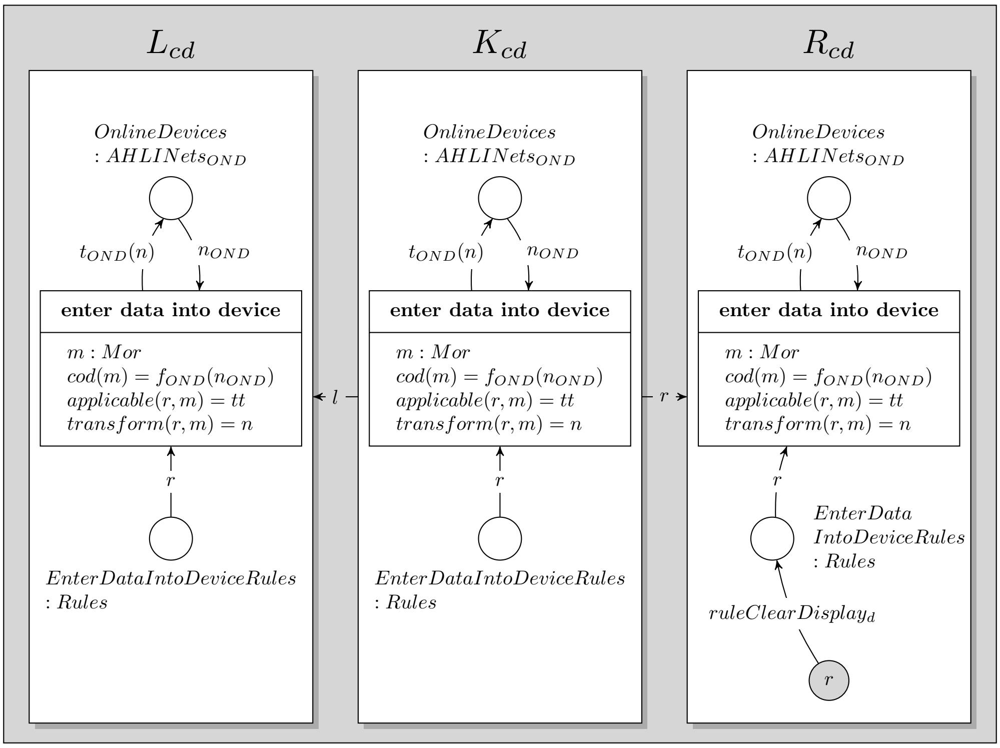


Figure 7.96: AHOI rule $ClearDisplay_{user,d}$

7.1.4.7 Rules for Direct User Input Into the Daily Planner

Calendar data for the Daily Planner (see 7.1.2.9) will be entered by the resident in applying the AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$, where t describes a concrete timestamp and d is a concrete description of an appointment. Because of an infinite set of possibilities, the AHOI rule presented in this section is only a template for all possible concrete AHOI rules.
The AHOI rule assigns the AHLI rule $ruleAddData_{planner,t,d}$, as presented in 7.1.3.5.3, to the place *EnterDataIntoDeviceRules : Rules* in the system net of the Message Oriented Middleware.

To remove data from the Daily Planner, the AHOI rule $RemoveDataFromDailyPlanner_{user,t,d}$ is available which assigns an AHLI rule $ruleRemoveData_{planner,t,d}$ to the place $EnterDataIntoDeviceRules$. The AHLI rule assigns a token to be removed to the place $Data_P$ within the device net of the Daily Planner in order initiate the deletion of of stored information from the persistence layer.

In the following image the use case regarding the interaction of the user with the Daily Planner is presented below, the AHOI rules $EnterDataIntoDailyPlanner_{user,t,d}$ and *RemoveDataFromDaily-Planner*$_{user,t,d}$ are shown.

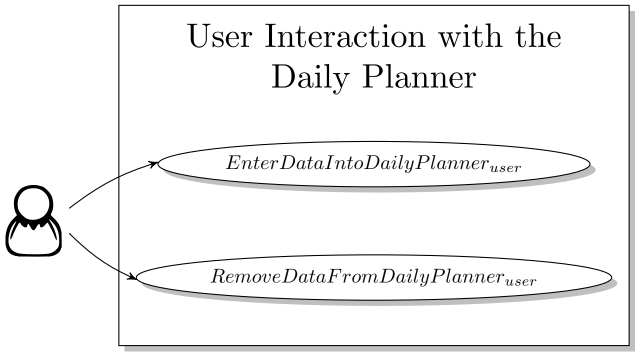

User Interaction with the Daily Planner
$EnterDataIntoDailyPlanner_{user}$
$RemoveDataFromDailyPlanner_{user}$

Figure 7.97: Use Case for the Interaction with the Daily Planner in the Multitouch Kitchen Counter

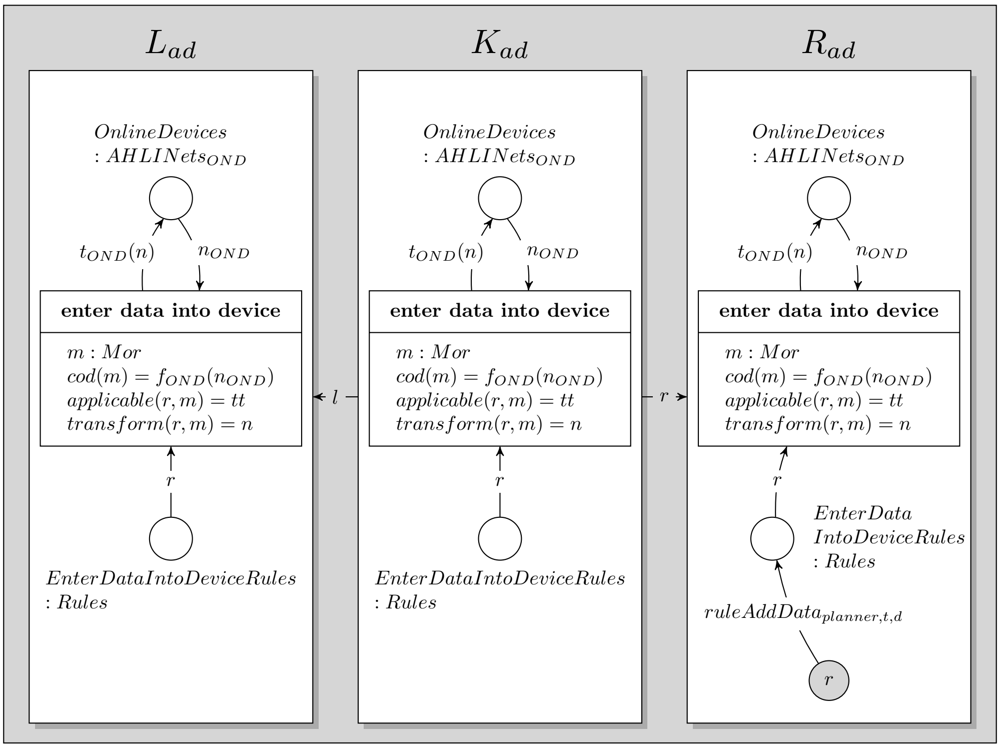


Figure 7.98: AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$

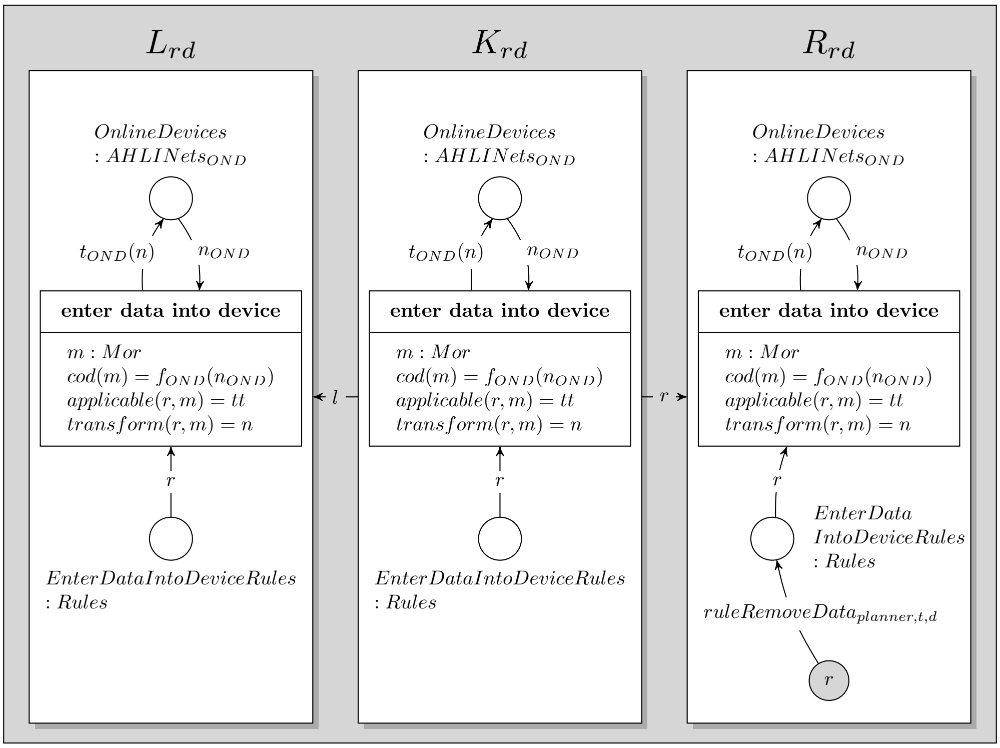


Figure 7.99: AHOI rule $RemoveDataFromDailyPlanner_{user,t,d}$

7.1.4.7.1 Revisiting the Requirements Towards the User Operations Causing a Change of the Daily Planner

In this section, the requirements set up in section 4.2.4.2.3 will be revised regarding its fulfilment by the model of the AHOI rules and its corresponding AHLI rules illustrated in this section.

$\mathbf{r_{planner_{user_1}}}$: The AHOI transformation rule $EnterDataIntoDailyPlanner_{user,t,d}$ assigns an AHLI rule $ruleAddData_{planner,t,d}$ to the place $EnterDataIntoDeviceRules : Rules$ within the net on System Level. The indices t and d represent the new appointment data, where t denotes the timestamp of this appointment and d its description. After applying the AHOI rule, $ruleAddData_{planner,t,d}$ can be applied directly to the device net of the Daily Planner in firing the transition *enter data into device* within the net on System Level, which assigns the new appointment as tuple (t, d) to the place $newCalendarData : CalendarData$ within the device net. From this place, the new appointment will be processed and sent to the Message Oriented Middleware.
Consequently, the requirement $r_{planner_{user_1}}$ is fulfilled by the AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$ and by the AHLI rule $ruleAddData_{planner,t,d}$.

$\mathbf{r_{planner_{user_2}}}$: Similar to the rules for entering new data, the AHOI transformation rules $RemoveDataFromDailyPlanner_{user,t,d}$ and the corresponding AHLI rules $ruleRemoveData_{planner,t,d}$ are provided in order to remove data. The AHOI rule assigns the AHLI rule to the place $EnterDataIntoDeviceRules : Rules$ so that the AHLI rule can be applied to the device net of the Daily Planner. The AHLI rule assigns a token $(d, t, id, remove)$ in order to announce to the persistence layer of the Message Oriented Middleware that the given calendar data should be removed from the Living Place system.
As a result, this requirement $r_{planner_{user_2}}$ is met.

7.1.4.8 Rule MeasurePressureInBed$_{\text{user}}$ for Input of Pressure Data in the Intelligent Bed

The user communicates indirectly with the bed pressure sensors because the user interacts with the bed and thereby provides new stimuli to the pressure sensors. As described in 7.1.2.10, the Intelligent Bed needs pressure data which will be assigned as tokens to the place $Sensor : PressureData$. The data consists of a 6-tuple containing the data for each of the six sensors. Similar to the AHOI resp. AHLI rule presented in last section 7.1.4.6.1, the indices in this AHOI rule represent concrete values. In the following, the template for a concrete AHOI rule will be presented.

New measurement data will be set in applying an AHOI rule $MeasurePressureInBed_{user,v_1,v_2,v_3,v_4,v_5,v_6,}$ $_{x_1,x_2,x_3,x_4,x_5,x_6}$, where $v_i, x_i \in \mathbb{R} \wedge v_i, x_i \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}$, as illustrated in image 7.101. The values x_i represent the current pressure data of the Intelligent Bed, whereas all v_i stand for the last measured pressure data. The AHOI rule assigns an AHLI rule $ruleMeasurePressure_{bed,v_1,v_2,v_3,v_4,v_5,v_6,x_1,}$ $_{x_2,x_3,x_4,x_5,x_6}$ to the place $EnterDataIntoDeviceRules$: $Rules$ for transforming the device net which is presented in 7.1.3.5.5.

The following picture presents the use case showing the interaction between the user and the Intelligent Bed.

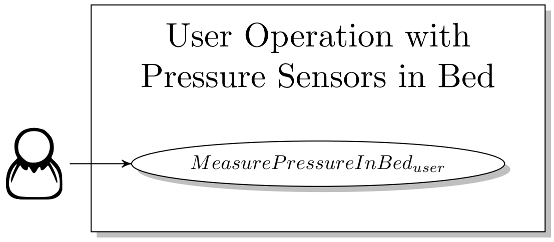


Figure 7.100: Use Case for the User Operation with the Pressure Sensors in the Intelligent Bed

7.1.4.8.1 Revisiting the Requirement Towards the User Operations Causing a Change of the Intelligent Bed

In section 4.2.2.7 one requirement for the user operation causing a change on the Intelligent Bed is presented, which will be revised in the following regarding its fulfilment.

$\mathbf{r_{bed_{user_1}}}$: The AHOI rules $MeasurePressureInBed_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ and the corresponding AHLI rules $ruleMeasurePressure_{bed,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ are provided for entering new pressure data to the device net of the Intelligent Bed. Because of that, the requirement $r_{bed_{user_1}}$ is met.

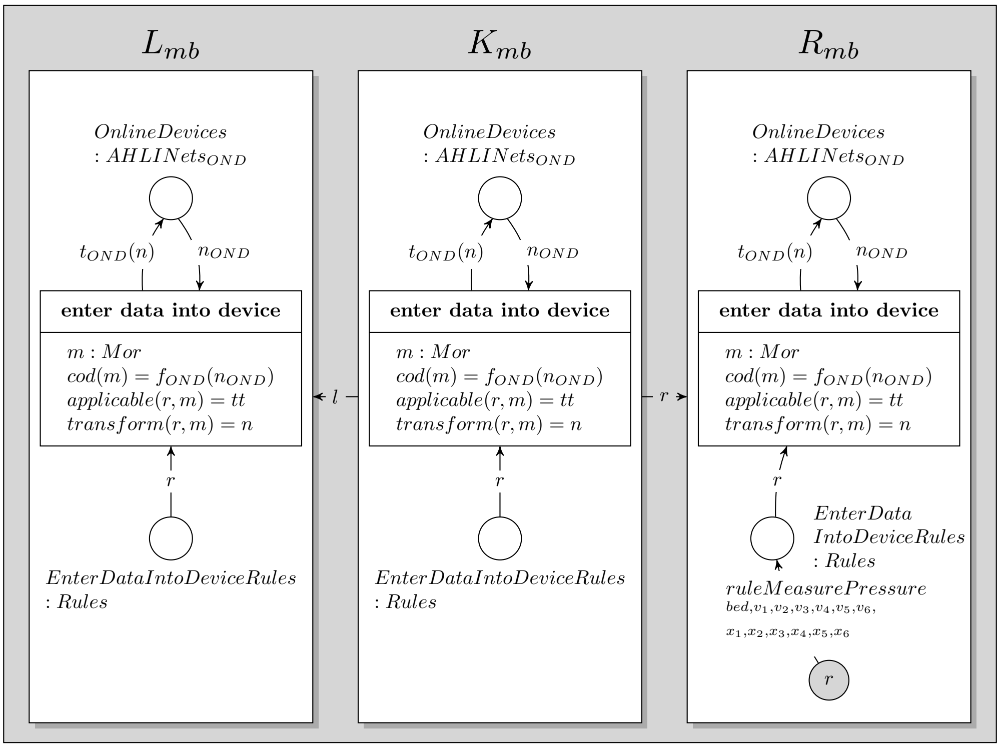


Figure 7.101: AHOI rule $MeasurePressureInBed_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$

7.1.4.9 Rule MeasurePosition$_{\mathbf{user}}$ for Input of Current User Position in the Living Place apartment

Similar to the pressure sensors of the Intelligent Bed as presented in last section, the Indoor-Positioning-System (see 7.1.2.11) gets the current position of the inhabitant, which will be measured automatically and therefore indirectly entered by the user.
In the model of the Living Place system the position data will be entered as 3-dimensional value (X,Y,Z) with the help of an AHOI rule $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$, where $v_i = (X_{v_i}, Y_{v_i}, Z_{v_i}), x_i = (X_{x_i}, Y_{x_i}, Z_{x_i}) \wedge X_{v_i}, Y_{v_i}, Z_{v_i}, X_{x_i}, Y_{x_i}, Z_{x_i} \in \mathbb{R} \wedge X_{v_i}, Y_{v_i}, Z_{v_i}, X_{x_i}, Y_{x_i}, Z_{x_i} \in [0.0, ..., 10.0], i \in \{1,2,3,4,5,6\}$, which is presented below. The first image shows the use case about entering position measurement data to the Living Place system. The second image presents the AHOI rule, which assigns the AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ (see 7.1.3.5.6) to the place $EnterDataIntoDeviceRules : Rules$.

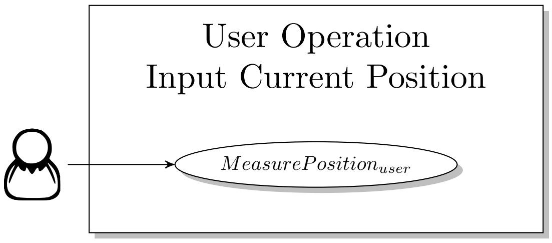


Figure 7.102: Use Case for the Operation regarding the Input of Current User Position

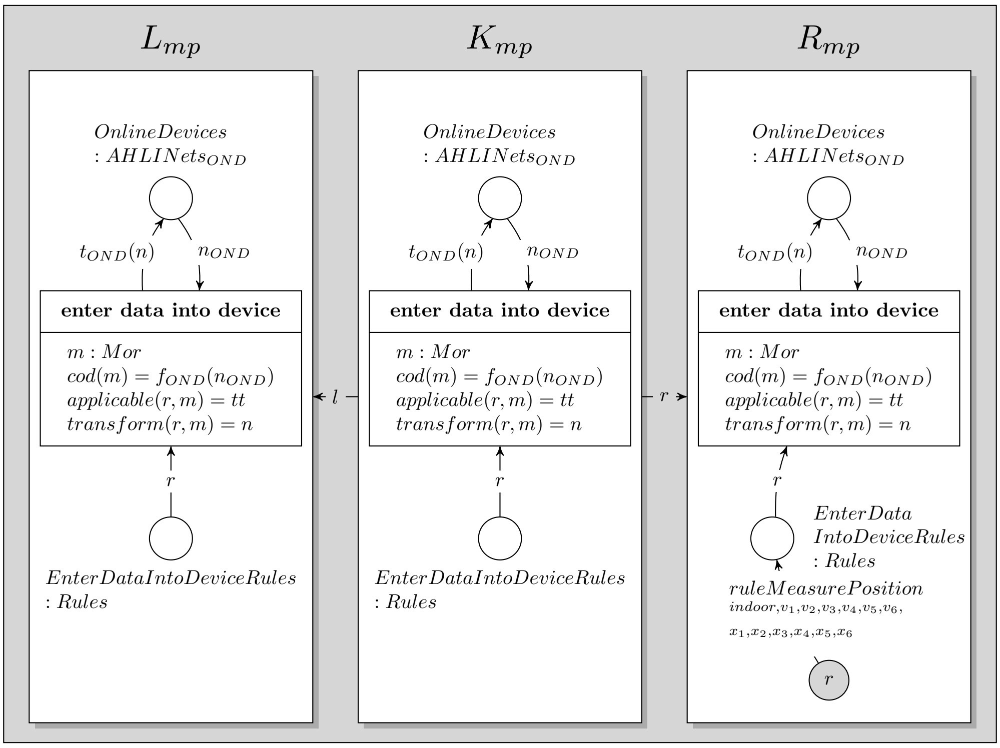


Figure 7.103: AHOI rule $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$

7.1.4.9.1 Revisiting the Requirement Towards the User Operations Causing a Change on the Indoor-Positioning-System

In section 4.2.2.8 one requirement for the user operation causing a change on the Intelligent Bed is presented, which will be revised in the following regarding its fulfilment.

$\mathbf{r_{indoor_{user_1}}}$: The AHOI rules $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ and the corresponding AHLI rules $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ are provided for entering new position data to the device net of the Indoor-Positioning-System. Therefore, the requirement $r_{indoor_{user_1}}$ is satisfied.

7.1.4.10 Rule for Switching the Location-Based Screen On or Off

Both TV sets as part of the Location-Based Screen (see 7.1.2.12) are able to be switched on and off simultaneously by the resident of the Living Place apartment. Both TV sets are showing their program alternative or both show nothing corresponding to the current position of the inhabitant, i.e. according to the user being in sight of the TV set or not. Therefore both TV sets need to switched on together in order to the Location-Based Screen operate correctly.
The possible interactions of the inhabitant of the Living Place apartment with the Location-Based Screen is presented in the following use case.

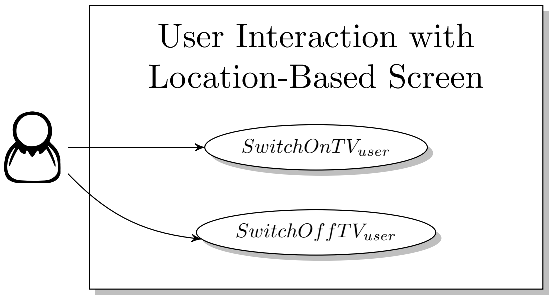


Figure 7.104: Use Case Switching Location-Based Screen on or off

The model of the Living Place system provides two AHOI rules illustrated in the following figures: $SwitchOnTV_{user}$ for switching on both TV sets and $SwitchOffTV_{user}$ the opposite AHOI rule. Each rule assigns the corresponding AHLI rule $ruleSwitchOn_{TV}$ resp. $ruleSwitchOff_{TV}$ to the place *EnterDataIntoDeviceRules : Rules* in the system net in order to transform the device net of the Location-Based Screen in Object Level, if a match can be found. Both AHLI rules are described in 7.76 and 7.77.

7.1.4.10.1 Switch On Location-Based Screen

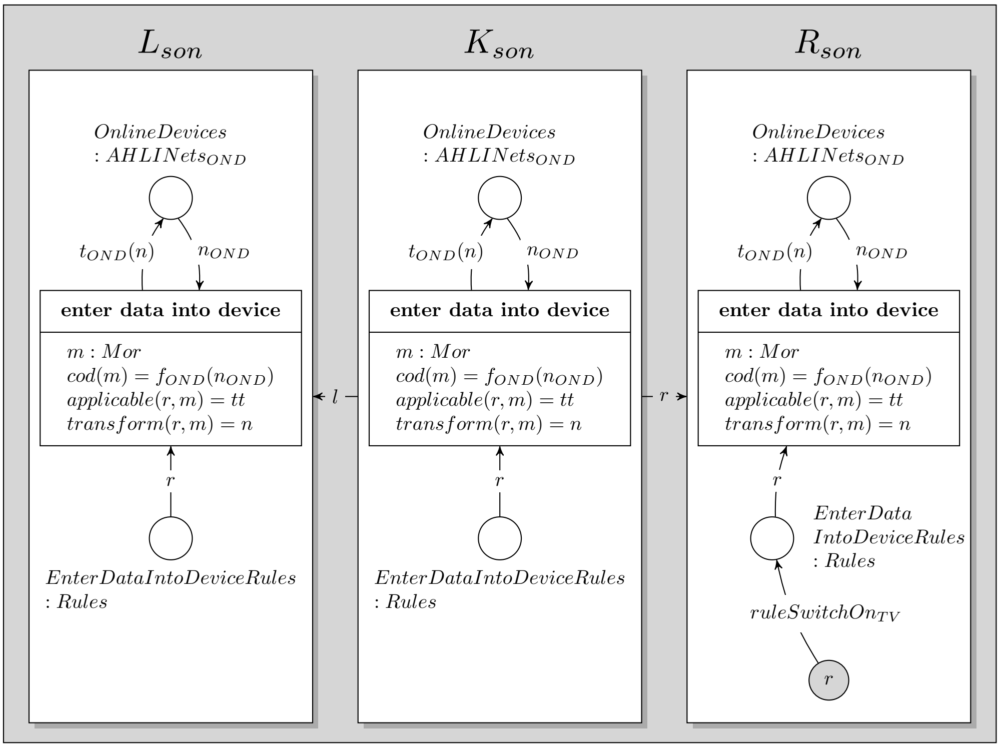


Figure 7.105: AHOI rule $SwitchOnTV_{user}$

7.1.4.10.2 Switch Off Location-Based Screen

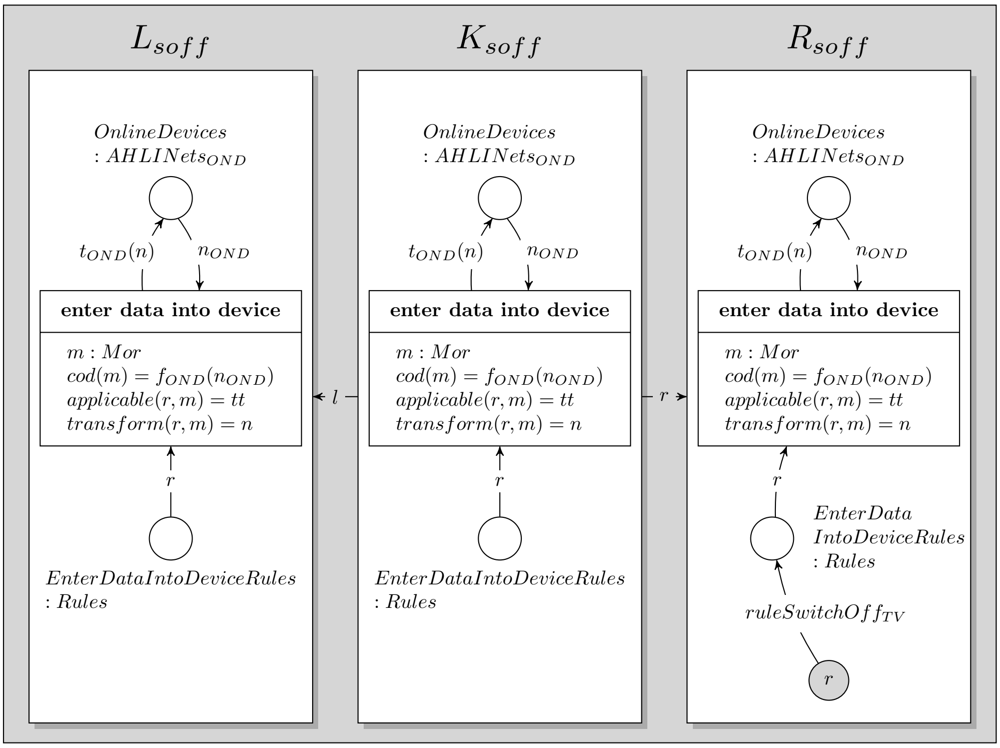


Figure 7.106: AHOI rule $SwitchOffTV_{user}$

7.1.4.10.3 Revisiting the Requirements Towards the User Operations Causing a Change on the Location-Based Screen

In section 4.2.2.11, requirements are presented which should be fulfilled by the models of the user operations described in this section 7.1.4.10. In the following, the requirements will be revised regarding their fulfilments.

$\mathbf{r_{screen_{user_1}}}$: The AHOI rule $SwitchOnTV_{user}$ and its corresponding AHLI rule $ruleSwitchOn_{TV}$ are provided which cause, that the internal on/off state of both TV set represented by a token assigned to each of the places $TV_1 : TVState$ and $TV_2 : TVState$ will be replaced by *on*. This represents the case, that both TV set within the Living Place system are switched on. Consequently, the requirement $r_{screen_{user_1}}$ is fulfilled.

$\mathbf{r_{screen_{user_2}}}$: In contrast, the AHOI rule $SwitchOffTV_{user}$ and its corresponding AHLI rule $ruleSwitchOff_{TV}$ cause the change of both internal TV states, that the token assigned to the places $TV_1 : TVState$ and $TV_2 : TVState$ will be replaced by off. Therefore, the requirement $r_{screen_{user_2}}$ is met by the model of the Living Place system.

7.1.4.11 Rule for Switching the Light On or Off in the Living Place Apartment

In the model of the Living Place system the light, as presented in section 7.1.2.13, is able to be set in the following states: bright, dimmed and off. The inhabitant is able to directly switch the light on into bright mode or switch the light off in the apartment in applying the AHOI rules $SwitchOnLight_{user}$ resp. $SwitchOffLight_{user,bright}$ or $SwitchOffLight_{user,dimmed}$ illustrated in the use case below and in the following pictures 7.108, 7.109 and 7.110.
Each of the AHOI rules have a corresponding AHLI rule $ruleSwitchOn_{Light}$, $ruleSwitchOff_{Light,bright}$ resp. $ruleSwitchOff_{Light,dimmed}$ presented in sections 7.1.3.5.9, 7.79 and 7.80.

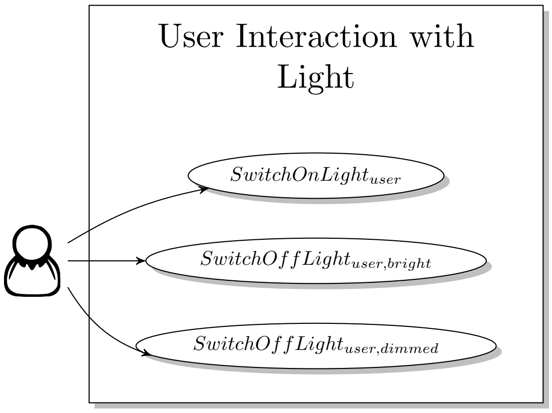

User Interaction with Light
$SwitchOnLight_{user}$
$SwitchOffLight_{user,bright}$
$SwitchOffLight_{user,dimmed}$

Figure 7.107: Use Case Switching Location-Based Screen on or off

7.1.4.11.1 Revisiting the Requirements Towards the User Operations Causing a Change of the Ambient Light

The requirements presented in section 4.2.2.12 should be fulfilled. Therefore, both requirements will be revised in order to check, if they are fulfilled by the models presented in this section 7.1.4.11.

$\mathbf{r_{light_{user_1}}}$: The AHOI rule $SwitchOnLight_{user}$ is provided, which uses the AHLI rule $ruleSwitchOn_{Light}$ for changing the internal state of the Ambient Light within its object net to *on*. The internal state of the Light is represented by the token assigned to the place *Light* : *LightState*. Following, the requirement $r_{light_{user_1}}$ is fulfilled.

$\mathbf{r_{light_{user_2}}}$: For switching the Light off, two AHOI rules are available: $SwitchOffLight_{user,bright}$ and $SwitchOffLight_{user,dimmed}$ and their corresponding AHLI rules $ruleSwitchOff_{Light,bright}$ and $ruleSwitchOff_{Light,dimmed}$. Both rules change the internal state of the Ambient Light represented by a token assigned to the place *Light* : *LightState* within the device net to off. As a consequence, the requirement $r_{light_{user_2}}$ is met.

7.1.4.11.2 Switch On Light

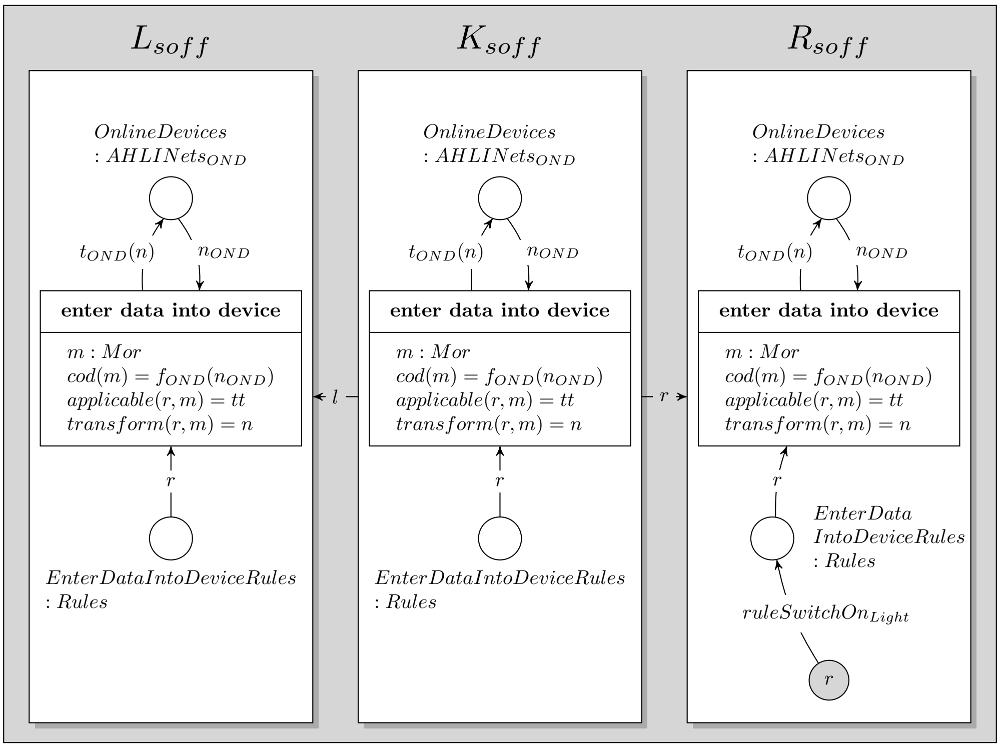


Figure 7.108: AHOI rule $SwitchOnLight_{user}$

7.1.4.11.3 Switch Off Light

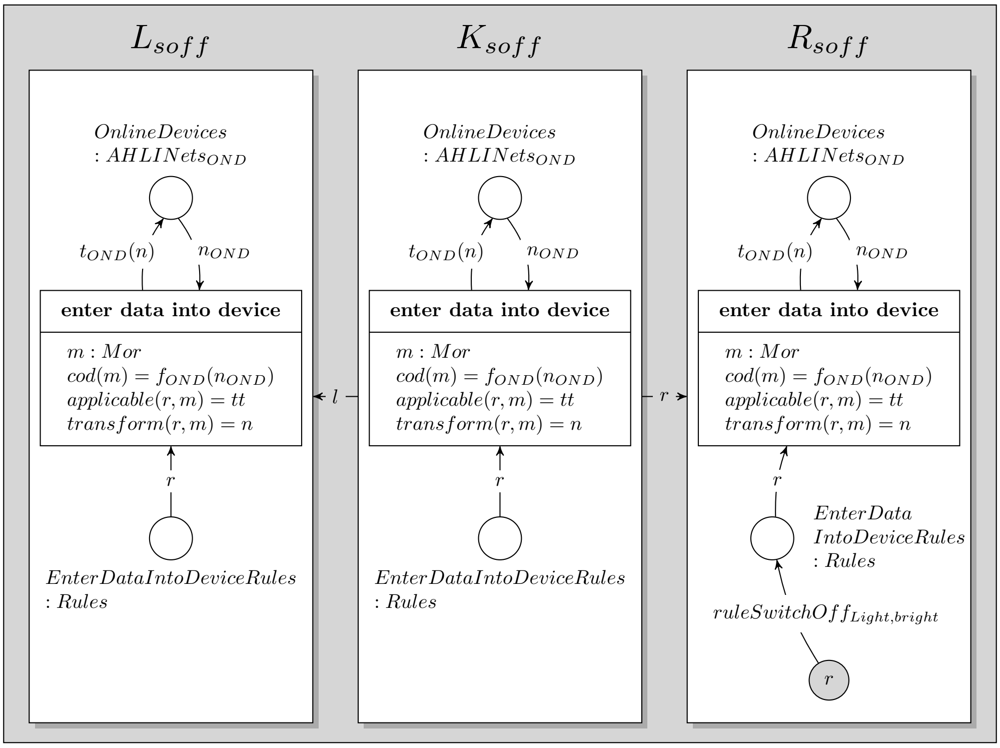


Figure 7.109: AHOI rule $SwitchOffLight_{user,bright}$

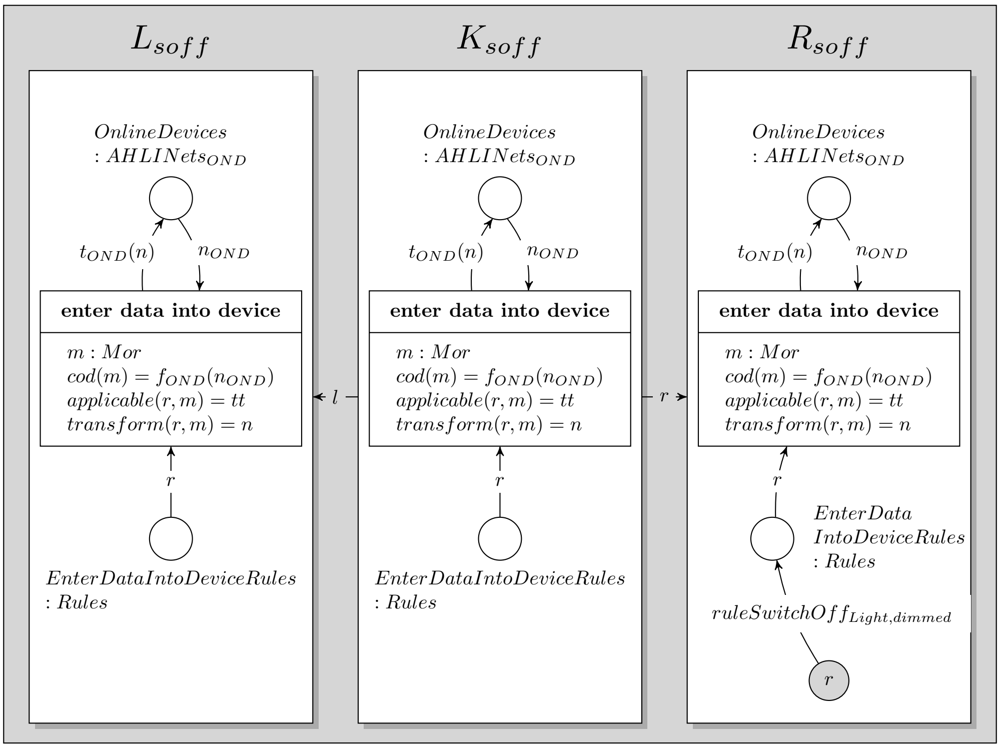


Figure 7.110: AHOI rule $SwitchOffLight_{user,dimmed}$

7.1.4.12 Revisiting the Requirements Towards the User Operations

The following requirements are demanded to the user operations provided by model of the Living Place system, which will be revised in this section regarding its fulfilment by the model.

$\mathbf{r_{ubiu_3}}$: By applying the *AHOI* transformation rules resp. interaction schemes $EmergencyOff^{online}_{user}$, $EmergencyOff^{offline}_{user}$, $DeleteQueue_{user}$, $DeleteReceiveDataRules_{user}$, $DeleteRequests_{user}$ and $DeletePersistentData_{user}$ of figures 7.86, 7.87, 7.89, 7.90, 7.91 and 7.92 on the $AHOI_{AHLI}$ system net, the user is able to take control over the system by shutting it down or restarting resp. initialising the whole system. These rules are defined in a way, so that they are always applicable in the considered situations. Therefore, the user is able to take control over the system at any time. So, requirement r_{ubiu_3} is fulfilled.

$\mathbf{r_{ubiu_2}}$ **and** $\mathbf{r_{ubis_3}}$: In applying the user operations presented in section 7.1.4, the user is able to influence the behaviour of the Living Place system. This leads to the fulfilment of the requirement r_{ubiu_2}. In addition, each AHOI rule resp. user interaction scheme, presented provokes a defined reaction to this operation, e.g. the AHOI rule $PluginDevice_{user,y}$ adds a new device to the Living Place system, or the AHOI rule $SwitchOnLight_{user}$ assigns a rule $ruleSwitchOn_{Light}$ to the place $EnterDataIntoDeviceRules : Rules$ in order to provoke the transformation of the device net of the Ambient Light in online mode, so that the internal state of the light can be switched to *on*. Therefore, each user operation causes a defined reaction of the Living Place system. Consequently, the requirement r_{ubis_3} is met, too.

7.2 Model in a Formal Description

7.2.1 Data Level

The models of the Data Level are represented by the signature Σ_{OL} and Σ_{OL}-Algebra A_{OL} as defined in section 11.1 resp. 11.2.

7.2.1.1 Revisiting the Requirement Towards the Data Level

In section 4.2.1 the requirement r_{data_1} is set up which demands:
The communication in the Living Place system between all connected devices takes place with the help of topics (see 2.2.1.3), hence, the model on the Data Level has to provide a data type for topic with a corresponding carrier set enabling identifiers for all relevant topics.
The Σ_{OL}-Algebra A_{OL} contains the type $A_{OL_{Topic}}$ with a carrier set which enables the generation of identifiers for topics. Hence, this requirement is fulfilled. Additionally, the same algebra includes carrier sets for the data that are typed over these topics. Therefor, requirement r_{ubid_1}, which is put forward in section 4.1, is also fulfilled.

7.2.2 Object Level

The models of the Object Level are defined as $AHLI$ nets as introduced in section 6.1.1. All defined $AHLI$ nets share the same signature Σ_{OL} and Σ_{OL}-Algebra A_{OL} as defined in section 11.1 resp. 11.2.

The explicit formal notation of these $AHLI$ nets is omitted, since the visual representation of the previous section 7.1 is meaningful enough. However. in section 11.5 an exemplary formal notation for $AHLI$ net $device^{offline}_{sender}$ of figure 7.6 is given.

7.2.3 System Level

The models of the System Level are defined resulting into one huge $AHOI_{AHLI}$ net as introduced in section 6.3. The so defined $AHOI_{AHLI}$ net includes signature Σ_{SL} and Σ_{SL}-Algebra A_{SL} as defined in section 11.3 resp. 11.4.

Consider, that all rule places $p \in P$ with $type(p) = Rules$ of this $AHOI_{AHLI}$ net are contextual, i.e. for all transitions $t \in T$ connected with p there exists variable $x \in X$, so that $pre(t)_{|p} = post(t)_{|p}$. Compare to definition of *Higher-Level Net and Rule Systems* in [HEM05].

The explicit formal notation of this $AHOI_{AHLI}$ net is omitted, since the visual representation of the previous section 7.1 is meaningful enough. However. in section 11.5 an exemplary formal notation for $AHLI$ net $device^{offline}_{sender}$ of figure 7.6 is given.

Consider, that as already mentioned in section 6.2.2, the morphsisms l, r of each $AHLI$ transformation rule defined as a token within this $AHOI_{AHLI}$ net are morphisms of the form $l_{\Sigma_{OL}} = r_{\Sigma_{OL}} = id_{\Sigma_{OL}}$ and $l_{A_{OL}} = r_{A_{OL}} = id_{A_{OL}}$ resp. $l_{T_{\Sigma_{OL}}(X)} = r_{T_{\Sigma_{OL}}(X)} = id_{T_{\Sigma_{OL}}(X)}$, where the nets L, K, R share the same signature Σ_{OL} and also the same Σ_{OL}-Algebra A_{OL} resp. the same corresponding term algebra with variables $T_{\Sigma_{OL}}(X)$.[12]

Furthermore, the $AHOI_{AHLI}$ net of the System Level contains the $AHLI$ nets of the Object Level as tokens.

7.2.4 User Level

The explicit formal notations of the $AHOI_{AHLI}$ transformation rules defined in the User Level as part of the topmost reconfigurable $AHOI$ system are omitted, since the visual representations of the previous section 7.1 are meaningful enough. However. in section 11.5 an exemplary formal notation for $AHLI$ net $device_{sender}^{offline}$ of figure 7.6 is given.

Consider, that as already mentioned in section 6.2.2, the morphsisms l, r of each $AHOI_{AHLI}$ transformation rule defined in the User Level as part of the topmost reconfigurable $AHOI$ system, that is intended to be applied to the topmost $AHOI_{AHLI}$ net, are morphisms of the form $l_{\Sigma_{SL}} = r_{\Sigma_{SL}} = id_{\Sigma_{SL}}$ and $l_{A_{SL}} = r_{A_{SL}} = id_{A_{SL}}$ resp. $l_{T_{\Sigma_{SL}}(X)} = r_{T_{\Sigma_{SL}}(X)} = id_{T_{\Sigma_{SL}}(X)}$, where the nets L, K, R share the same signature Σ_{SL} and also the same Σ_{SL}-Algebra A_{SL} resp. the same corresponding term algebra with variables $T_{\Sigma_{SL}}(X)$.[13]

The reconfigurable $AHOI$ system of the User Level is defined as $(N, \mathcal{R})$, where N is the $AHOI_{AHLI}$ system net in initial mode of figure 7.26 and $\mathcal{R}$ is the set of $AHOI_{AHLI}$ transformation rules of the User Level as defined in this chapter.

7.3 Summary

This chapter presents the formal overall model of the Living Place system using the formal modelling techniques of chapter 6.

Within the here presented modelling it is clarified, that the resulting model fulfills requirement $\mathbf{r_{gen}}$ of chapter 4 as well as any other therein defined requirement. Therefore, this model can be considered as an adequate, elegant model of the Living Place system.

By modelling this system that way, it is shown, that the used modelling techniques of Algebraic-High Level Nets with Individual Tokens and the rule-based transformation of such nets following the double pushout approach are expressive enough and therefore suitable to be used as formal tools to model systems of such high complexity in an elegant manner.

Therefore, the question *Are the used formal modelling techniques expressive enough and therefore suitable for modelling the Living Place system?* can be answered with: *Yes*.

In the next chapter 8 the elaborated model is used by considering an exemplary scenario and simulating the therin occuring system behaviours of the Living Place system by using the model.

[12] Refer to sections 11.1 and 11.2 for the definition of signature Σ_{OL} and Σ_{OL}-Algebra A_{OL}.

[13] Refer to sections 11.3 and 11.4 for the definition of signature Σ_{SL} and Σ_{SL}-Algebra A_{SL}.

Chapter 8
Simulation of a Test Scenario

A simulation of a scenario of the Living Place system with all components presented in the modelling in chapter 7 will be illustrated in the current chapter. The scenario which will be simulated will contain the following steps the Living Place system:

1. At the beginning of the scenario, the user attaches all devices except the Alarm Clock 2.0 and the Display in the Multitouch Kitchen Counter to the Living Place system. The devices which will be plugged in are:

   - Daily Planner
   - Intelligent Bed
   - Indoor-Positioning-System
   - Location-Based Screen
   - Weather Information Service
   - Traffic Information Service
   - Ambient Light

   Afterwards the devices need to be set into online mode and, if necessary, all topics will be subscribed automatically and requests for old data will be sent to the persistence layer which is a part of the Message Oriented Middleware.

2. As next step the inhabitant enters a new appointment to the Daily Planner with the help of the Multitouch Kitchen Counter.
3. Now the Alarm Clock 2.0 and the Display in the Multitouch Kitchen Counter will be plugged in to the Living Place system. Both devices request all past data belonging to the topic “Calendar” from the persistence layer.
4. After going to sleep, the Intelligent Bed sends information about the sleeping phases the user is in. Additionally the Alarm Clock 2.0 analyses the calendar data the user has entered and checks current weather and traffic conditions to infer an appropriate wake up time. Then, the Alarm Clock 2.0 compares its internal wake up time with the current timestamp and the sleeping phase. If all conditions are fulfilled, an alarm will be provoked, which illuminates the Ambient Light in dimmed mode and switches on the TV set of the Location-Based Screen.
5. After getting up because of the alarm, the person enters the bathroom to wash. The Living Place apartment will recognize this and removes the task from the internal list.
6. As next step, after 20 minutes, the person goes to the sleeping room to dress. Meanwhile the TV set of the sleeping area will be activated automatically. Because the inhabitant does not fulfil the last task, the Alarm Clock 2.0 tries to remind the user of having breakfast regularily.
7. Before leaving the apartment, the user switches off the whole Living Place system.

All steps of the scenario mentioned, will be presented in detail in this chapter.

8.1 Initial Mode of the Living Place System

Before starting the scenario the Message Oriented Middleware of the Living Place system is set to initial mode, which is illustrated in image 7.26.

In the part representing the message broker, no devices are assigned as tokens to the places $OnlineDevices : AHLINets_{OND}$ or $OfflineDevices : AHLINets_{OFFD}$ in the system net, meaning that no device is connected to the Living Place system. The counter for the connection identifications is initially set to 1. In addition no topics are subscribed, hence no subnets representing available queues connections are existing in the AHLI net assigned to the place $Queues : AHLINets$. At startup no messages were sent, therefore no data are assigned to the place : $GlobalData$ within the object net $DataBuffer_1$.

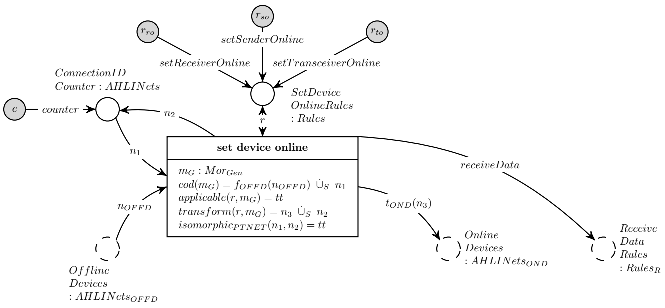
The part of the persistence layer within the Message Oriented Middleware in initial mode contains the object net *database* with one place $db : PersistentData$ without any tokens assigned to it.

The Context Interpreter contains no data. In the whole system net of the Message Oriented Middleware, all necessary rules are available and assigned to its corresponding places.[1][2]

8.2 Attach Devices to the Message Oriented Middleware of the Living Place System

As first step, the inhabitant attaches all devices to the Message Oriented Middleware of the Living Place system. The process of attaching is always done in the sequence presented exemplarily for the device net of the Ambient Light.

1. Extract from system net *System* in initial mode (see 7.29):


r_{ro}
r_{so}
r_{to}
setSenderOnline
setReceiverOnline
setTransceiverOnline
ConnectionID Counter : AHLINets
c
counter
n_2
SetDevice OnlineRules : Rules
r
n_1
set device online
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OFFD}(n_{OFFD}) \dot{\cup}_S n_1$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_3 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(n_1, n_2) = tt$
receiveData
n_{OFFD}
$t_{OND}(n_3)$
Offline Devices : $AHLINets_{OFFD}$
Online Devices : $AHLINets_{OND}$
Receive Data Rules : $Rules_R$

No device is assigned to the places $OfflineDevices : AHLINets_{OFFD}$ resp. $OnlineDevices : AHLINets_{OND}$ in the system net of the part refering to the message broker.

The net *counter* in initial mode is set to the place $ConnectionIDCounter : AHLINets$. Initially the counter is set to 1, whose object net is shown below:

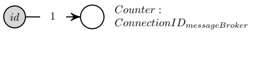
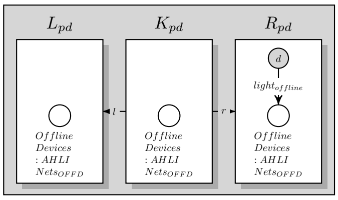


Figure 8.1: Attach a Device

2. Apply AHOI rule $PluginDevice_{user,light_{offline}}$ to the system net *System* (see 7.83):


L_{pd}
K_{pd}
R_{pd}
d
$light_{offline}$
l
r
Offline Devices : AHLI $Nets_{OFFD}$
Offline Devices : AHLI $Nets_{OFFD}$
Offline Devices : AHLI $Nets_{OFFD}$

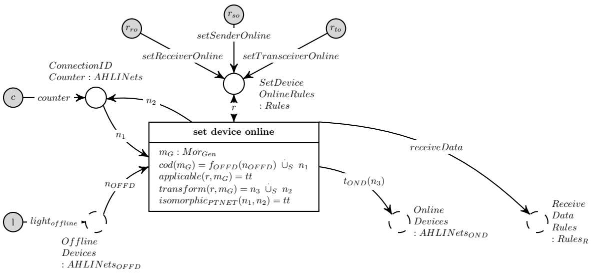
The inhabitant executes the AHOI rule $PluginDevice_{user,light_{offline}}$ first, which provides a concrete AHLI net $light_{offline}$ of the device net representing the Ambient Light in offline mode which should be plugged in. Now, the device net $light_{offline}$ will be added to the Message Oriented Middleware.

Figure 8.2: Attach a Device

[1] In this chapter mainly extracts of nets are presented. Dashed places have the meaning that additional parts are connected to this place but are not mentioned in the specific picture because it shows only a subnet.

[2] All devices added to the system are situated on $OnlineDevices : AHLINets_{OND}$ or $OfflineDevices : AHLINets_{OFFD}$. Connected to these places are fire transitions as described in 7.1.3.1.1. Therefore, the device nets are able to fire. The firing of transitions within object nets triggered by the fire transitions in system level, will not be presented within this scenario in explicit.

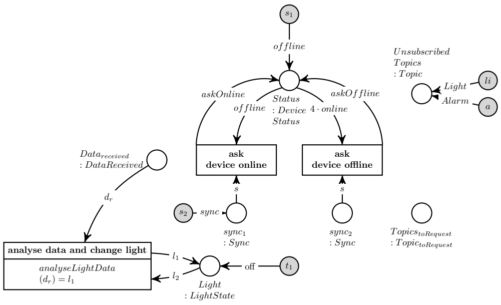
3. System net $System'$ extended by a new offline device $light_{offline}$ assigned to the place $OfflineDevices : AHLINets_{OFFD}$ (see 7.29):



After applying the AHOI rule $PluginDevice_{user,light_{offline}}$ with the indice $light_{offline}$, which is the device net of the Ambient Light in offline mode, a new device $light_{offline}$ is assigned to the place $OfflineDevices : AHLINets_{OFFD}$ in the system net of the part refering to the message broker.

Figure 8.3: Attach a Device

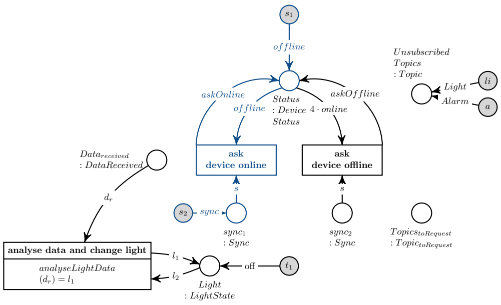
4. Device net $light_{offline}$ which is assigned to the place $OfflineDevices$ in the system net $System'$ (see 7.20):



The device net $light_{offline}$, which is part of the Object Level, is assigned to the place $OfflineDevices$ in the system net $System'$. It is initially set to offline mode but is able to request to be set into online mode.

Figure 8.4: Attach a Device

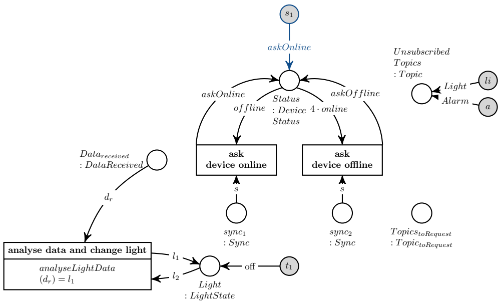
5. Device net $light_{offline}$:



To request the activation into online mode, the transition *ask device online* in the device net fires.

Figure 8.5: Attach a Device

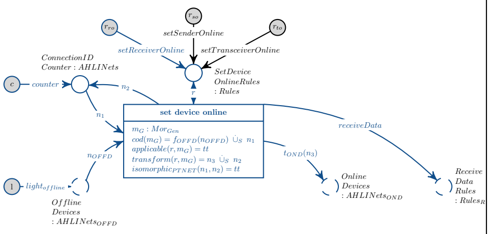
6. Device net $light'_{offline}$:



Afterwards the tokens $sync$ resp. $offline$ are removed from the places $sync_1$ resp. $Status$. Instead a new token $askOnline$ is assigned to the place $Status$.

Figure 8.6: Attach a Device

7. The system net $System'$ is able to apply the rule $setReceiverOnline$ to the device net $light'_{offline}$:

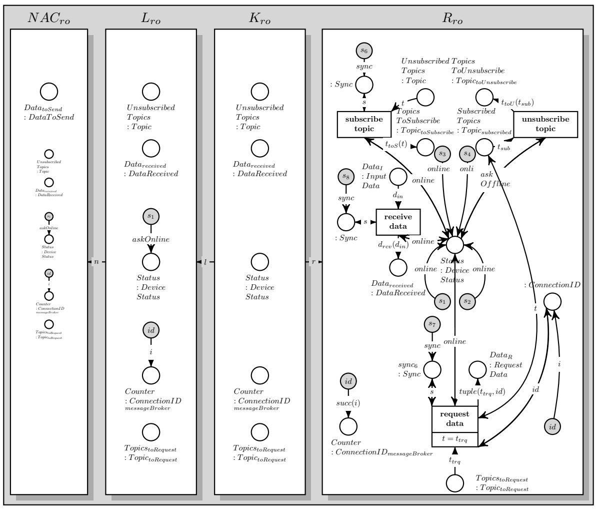


r_{ro}
r_{so}
r_{to}
setSenderOnline
setReceiverOnline
setTransceiverOnline
ConnectionID Counter : AHLINets
SetDevice OnlineRules : Rules
c
counter
n_2
n_1
r
set device online
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OFFD}(n_{OFFD}) \dot{\cup}_S\ n_1$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_3 \dot{\cup}_S\ n_2$
$isomorphic_{PTNET}(n_1, n_2) = tt$
receiveData
n_{OFFD}
$t_{OND}(n_3)$
1
$light_{offline}$
Offline Devices : $AHLINets_{OFFD}$
Online Devices : $AHLINets_{OND}$
Receive Data Rules : $Rules_R$

The object net $light'_{offline}$ is a receiving device. Because of the modification of the object net in last steps 5. and 6., the system net is able to apply the rule $setReceiverOnline$ in firing the transition *set device online* to transforms the device net completely into a the device net of the Ambient Light in online mode.

Figure 8.7: Attach a Device

8. AHLI rule $setReceiverOnline$ which will be applied to the device net $light_{offline}$ and the object net $counter$ when firing *set device online* in order to change the device status from offline to online (see 7.32):

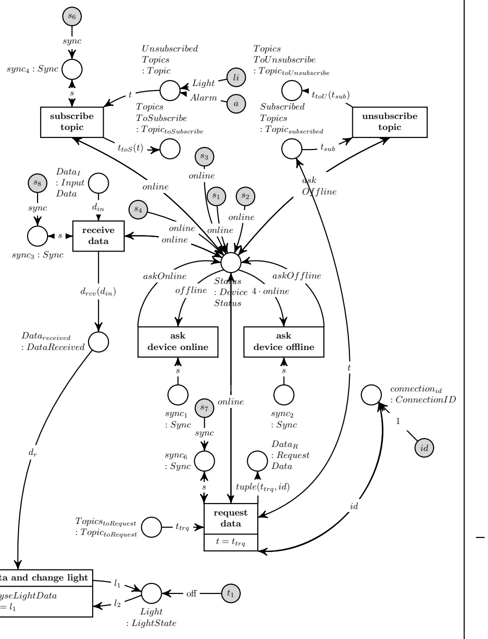


The rule $setReceiverOnline$ will be applied to the device net $light'_{offline}$ in order to change its status into online mode and furthermore the AHLI rule transforms the object net $counter$. The variables i mentioned on the left-hand-side L_{ro} and on the right-hand-side R_{ro} in this rule will be substituted, where i is represented by the match morphism $m = eval(X)$, with $i \in X$ will be depicted as 1, i.e. $eval_X(i) = 1$. Then, the connection identification provided by the object net $counter$ and which will replace i, will be increased from 1 to 2 in executing this rule.
The counter gets the current connection identification which will be increased with every device set into online mode, i.e. everytime the transition *set device online* fires.

The rules $setSenderOnline$ (see 7.30) and $setTransceiverOnline$ (see 7.31) operate similar to $setReceiverOnline$.

Figure 8.8: Attach a Device

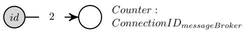
9. Device net $light_{online}$ after transformation by the rule *setTransceiverOnline* (see 7.21):



The object net $light_{online}$ is the result of the transformation with the rule *setReceiverOnline*.

Figure 8.9: Attach a Device

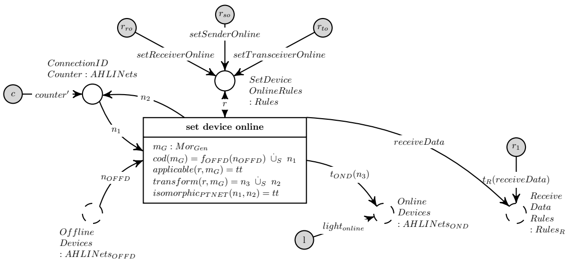
10. Object net $counter'$ after its transformation by the rule *setTransceiverOnline*:



After applying the rule *setReceiverOnline*, the token 1 assigned to *Counter* is removed and replaced by $succ(1) = 2$.

Figure 8.10: Attach a Device

11. The system net $System''$ after firing *set device online* **in order to set the device net into online mode**:



r_{ro}
r_{so}
r_{to}
setSenderOnline
setReceiverOnline
setTransceiverOnline
ConnectionID Counter : AHLINets
c
counter'
n_2
r
SetDevice OnlineRules : Rules
n_1
set device online
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OFFD}(n_{OFFD}) \cup_S n_1$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_3 \cup_S n_2$
$isomorphic_{PTNET}(n_1, n_2) = tt$
n_{OFFD}
receiveData
r_1
$t_{OND}(n_3)$
$t_R(receiveData)$
Online Devices : $AHLINets_{OND}$
Receive Data Rules : $Rules_R$
$light_{online}$
1
Offline Devices : $AHLINets_{OFFD}$

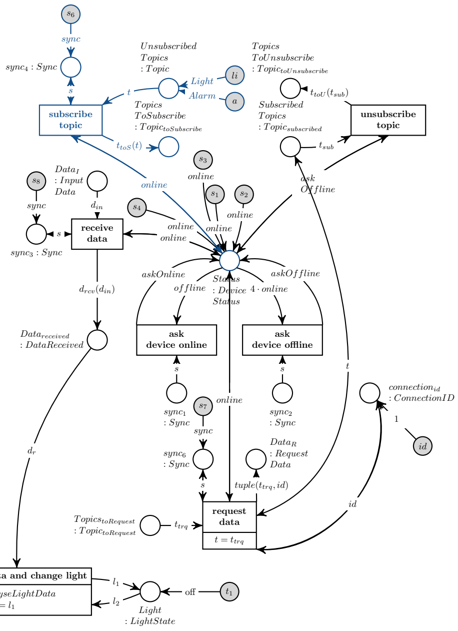
After firing the transition *set device online*, the object net *counter'* is modified as shown in 10. and put back to its place *ConnectionIDCounter*. The device net which was assigned to *OfflineDevices* is removed. Instead the transformed object net $light_{online}$ is assigned to *OnlineDevices*. Additionally a new token *receiveData* is set to the place *ReceiveDataRules*, which is regulates the parallel receiving of data for each device assigned to *OnlineDevices*.

Figure 8.11: Attach a Device

The object net representing the Light was plugged in with the help of the AHOI rule $PluginDevice_{user}$ which was initiated by the inhabitant of the Living Place apartment. The device net was able to request the change into online mode. Afterwards the message broker as part of the Message Oriented Middleware system net automatically sets the device into online mode. For all kinds of devices, these steps are identical, no matter if they are senders, receivers or transceivers. All other devices will be attached to the Living Place system analogously, therefore it will not be presented in explicit for all other device nets.

For receiving devices, i.e. receivers and transceivers, the process of plugging-in contains the subscription of all necessary topics. Afterwards it is possible to send requests to the persistence layer in order to get all past data which were stored, if necessary.
The first steps will be presented in explicit for the Ambient Light in the following part. For all other devices, the process of subscribing topics is analogously, hence it is not shown for each device in explicit.

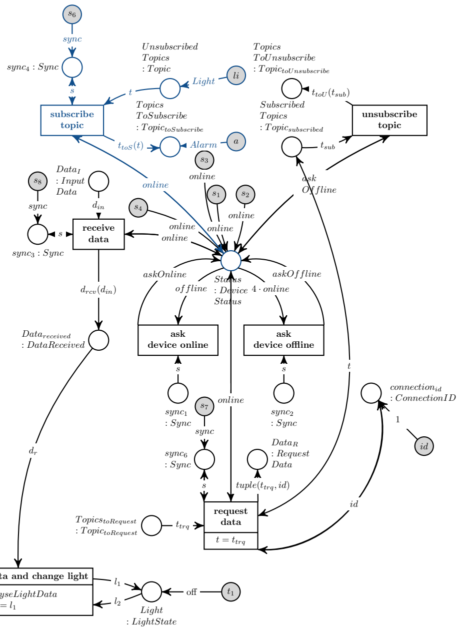
12. Device net $light_{online}$, ready to request subscription of topics:


subscribe topic
unsubscribe topic
receive data
ask device online
ask device offline
request data
analyse data and change light
Unsubscribed Topics : Topic
Topics ToUnsubscribe : Topic_toUnsubscribe
Topics ToSubscribe : Topic_toSubscribe
Subscribed Topics : Topic_subscribed
Light
Alarm
Status : Device Status
online
offline
askOnline
askOffline
sync
Data_received : DataReceived
connection_id : ConnectionID
Topics_toRequest : Topic_toRequest
Light : LightState
off

As next step, the net $light'_{online}$ is able to alert the message broker of new topics the device wants to be subscribed in firing the transition *subscribe topic*.

Figure 8.12: Attach a Device, Subscribe Topic

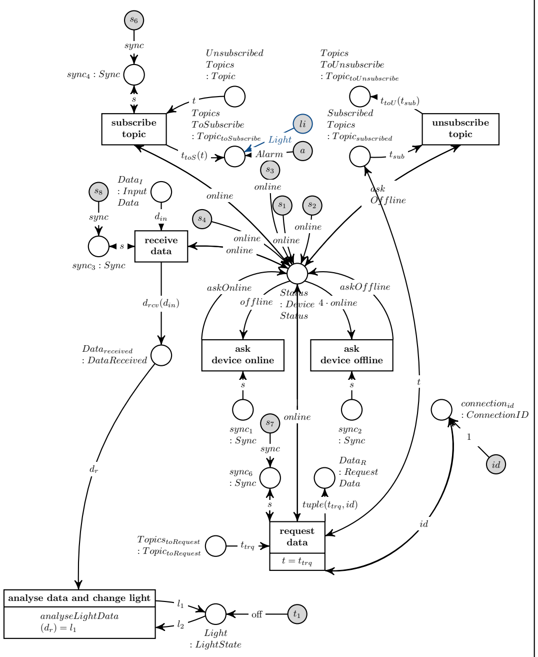
13. Device net $light'_{online}$, ready to request subscription of the second topic:


subscribe topic
unsubscribe topic
receive data
ask device online
ask device offline
request data
analyse data and change light
Unsubscribed Topics : Topic
Topics ToUnsubscribe : Topic_toUnsubscribe
Topics ToSubscribe : Topic_toSubscribe
Subscribed Topics : Topic_subscribed
Light
Alarm
Status : Device Status
online
offline
askOnline
askOffline
sync
Data_received : DataReceived
connection_id : ConnectionID
Topics_toRequest : Topic_toRequest
Light : LightState
off

One of both topics *Alarm* is transferred to the place *TopicsToSubscribe*. The transition *subscribe topic* still is enabled.

Figure 8.13: Attach a Device, Subscribe Topic

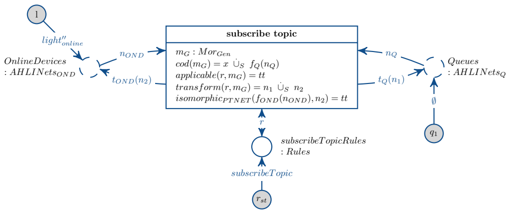
14. Device net $light''_{online}$, with both topics on $TopicsToSubscribe$:



After shifting the second topic *Light* to the place *TopicsToSubscribe*, the transition *subscribe topic* is deactivated. The message broker on System Level should now subscribe the device to all requested topics in the Living Place system.
The device net $light''_{online}$ representing the Ambient Light should be subscribed to the topics: “Alarm” and “Light”.

Figure 8.14: Attach a Device, Subscribe Topic

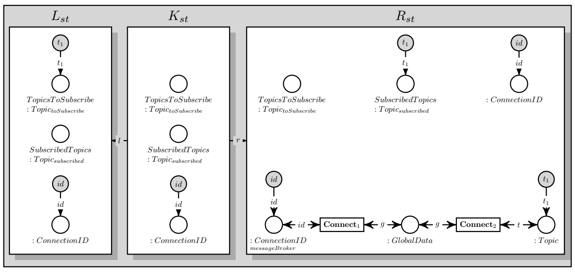
15. Extract from the system net $System''$ for subscribing a device to a topic:


subscribe topic
$m_G : Mor_{Gen}$
$cod(m_G) = x \cup_S f_Q(n_Q)$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_1 \cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_2) = tt$
$light''_{online}$
OnlineDevices : $AHLINets_{OND}$
Queues : $AHLINets_Q$
subscribeTopicRules : Rules
subscribeTopic
n_{OND}
$t_{OND}(n_2)$
n_Q
$t_Q(n_1)$
r_{st}
q_1

Once a device assigned to *OnlineDevices* has set tokens to its place *TopicsToSubscribe* as shown in step 14, the rule *subscribeTopic* is able to be applied to the device net in firing the transition *subscribe topic*.

Figure 8.15: Attach a Device, Subscribe Topic

16. AHLI rule $subscribeTopic$ for subscribing a device to a topic:

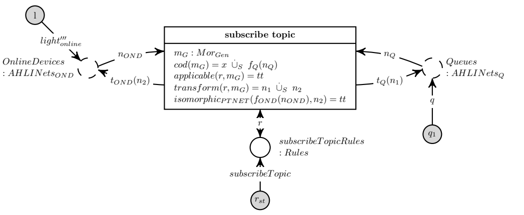

L_{st}
K_{st}
R_{st}
TopicsToSubscribe : $Topic_{toSubscribe}$
SubscribedTopics : $Topic_{subscribed}$
: ConnectionID
: ConnectionID messageBroker
: GlobalData
: Topic
Connect$_1$
Connect$_2$

The AHLI rule *subscribeTopic* will be applied for each topic assigned to *TopicsToSubscribe* within the device net and for each device net.
It transfers a topic assigned to *TopicsToSubscribe* to the place *SubscribedTopics* and creates a new subnet specifying the connection between the device and a certain topic which has to be stored within the Message Oriented Middleware in order to allocate messages correctly regarding its topic and the subscribed devices.

Because of the object net $light''_{online}$ containing two topics to subscribe, this rule will be applied twice.

Figure 8.16: Attach a Device, Subscribe Topic

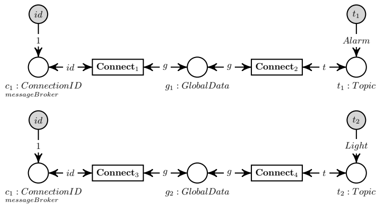
17. System net *System'''* after firing *subscribe topic* twice:



After firing the transition *subscribe topic* of the system net of the Message Oriented Middleware twice, the rule *subscribeTopic* was applied twice for each topic which should be subscribed. Afterwards the device net $light'''_{online}$ was transformed as shown in image 19 and put back to the place *OnlineDevices*. Additionally the token assigned to *Queues* was modified resulting the object net q which is presented in the next figure 18.

Figure 8.17: Attach a Device, Subscribe Topic

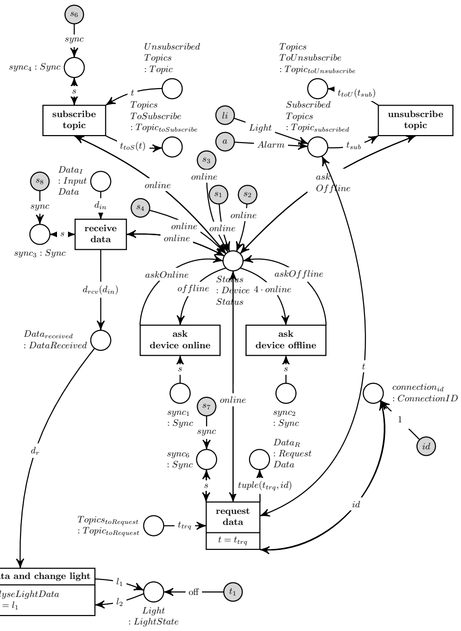
18. Object net q, after firing *subscribe topic* twice:



In the object net q extended after application of the rule *subscribeTopic*, a direct relation between the connection identification of the specific device and each topic was created. For each topic and device, a disjoint subnet will be generated.

Figure 8.18: Attach a Device, Subscribe Topic

19. Device net $light'''_{online}$, after subscription of both topics:



The rule *subscribeTopic* has been applied for each topic assigned to *TopicsToSubscribe* within the device net and for each device net.

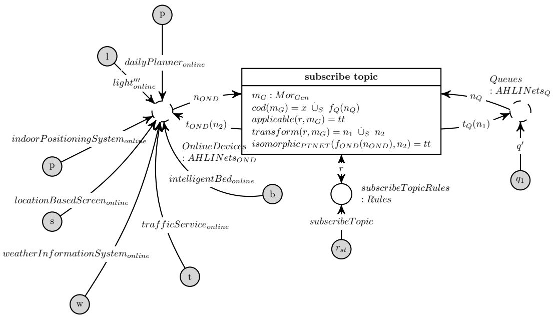
It transfered a topic assigned to *TopicsToSubscribe* to the place *SubscribedTopics* and creates a new subnet specifying the connection between the device and a certain topic which has to be stored within the Message Oriented Middleware in order to allocate messages correctly regarding its topic and the subscribed devices.

Because of the object net $light''_{online}$ contained two topics to subscribe, this rule has been applied twice.

Figure 8.19: Attach a Device, Subscribe Topic

Similar to the device net of the Ambient Light, all other devices can be connected to the Message Oriented Middleware.

20. Extract from the system net $System^4$ after plugging-in all devices except the Alarm Clock 2.0 and the Display in the Multitouch Kitchen Counter and after subscribing their topics:

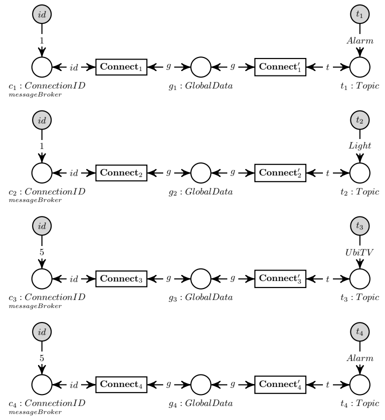

p
l
$dailyPlanner_{online}$
$light'''_{online}$
n_{OND}
$t_{OND}(n_2)$
subscribe topic
$m_G : Mor_{Gen}$
$cod(m_G) = x \dot{\cup}_S \ f_Q(n_Q)$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_1 \dot{\cup}_S \ n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_2) = tt$
Queues
$: AHLINets_Q$
n_Q
$t_Q(n_1)$
q'
q_1
$indoorPositioningSystem_{online}$
p
$OnlineDevices$
$: AHLINets_{OND}$
$intelligentBed_{online}$
b
r
$subscribeTopicRules$
$: Rules$
$subscribeTopic$
r_{st}
$locationBasedScreen_{online}$
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
t
w

All devices except the Alarm Clock 2.0 and the Display in the Multitouch Kitchen Counter are set into online mode and assigned to *OnlineDevices*. Some devices need to be subscribed to topics. This process is similar to the steps presented in 12. - 19. and the resulting device nets are similar to the device net *light'''online* shown in step 19: All topics are assigned as tokens to the place *SubscribedTopics*.
The devices are plugged in in the following order resulting in the mentioned connection ids between device and Message Oriented Middleware:

1. $light'''_{online}$, connection ID 1, see 7.21 or step 19 in the figure above,
2. $dailyPlanner_{online}$, connection ID 2, see 7.13,
3. $intelligentBed_{online}$, connection ID 3, see 7.15,
4. $indoorPositioningSystem_{online}$, connection ID 4, see 7.17,
5. $locationBasedScreen_{online}$, connection ID 5, see 7.19,
6. $weatherInformationSystem_{online}$, connection ID 6, see 7.23,
7. $trafficService_{online}$, connection ID 7, see 7.25.

Figure 8.20: Attach a Device, Subscribe Topic

21. Object net q' contains subnets representing all subscriptions at this stage:



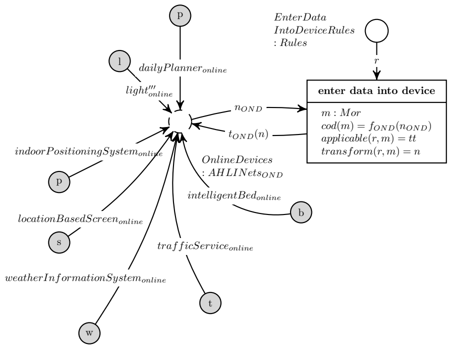
The object net q' contains all subscriptions after attaching all devices, except the Alarm Clock 2.0 and the Display in the Multitouch Kitchen Counter, to the Message Oriented Middleware and subscribing all available topics. The object net q' contains subscriptions for the device nets: $light'''_{online}$ with the connection ID 1 and $locationBasedScreen_{online}$ with the connection id 5.

Figure 8.21: Attach a Device, Subscribe Topic

8.3 The User Enters a new Appointment to the Daily Planner

As second step, the user enters a new appointment to the Daily Planner, in applying the AHOI rule $EnterDataIntoDailyPlanner_{user}$ which adds a new appointment to the Daily Planner device with the help of the AHLI rule $ruleAddData_{planner}$. Afterwards, the newly entered data will be transferred to the persistence layer of the Message Oriented Middleware on System Level.

1. Extract from the system net $System^4$ for entering new data to a device (see 7.48):



To enter a new data to a device net, a AHOI rule have to be applied which puts a AHLI rule to the given place $EnterDataIntoDeviceRules : Rules$ in order to assign the new data token to a certein place in the device net on Object Level. Each AHOI rule for entering data has one corresponding AHLI rule (or more in the case of the AHOI rule $ResetAlarmClock2.0_{user}$), which matches to a special kind of device net.

The following two illustrations show a general AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$ and its general AHLI rule $ruleAddData_{planner,t,d}$ for adding new data (t, d) to the Daily Planner. The indices t and d are placeholders for concrete data which will be transferred as new data to the device net on Object Level. Therfore, an infinite set of AHOI rules and associated AHLI rules exist for adding new information to the Daily Planner.

Figure 8.22: Enter Appointment

2. General AHOI rule $EnterDataIntoDailyPlanner_{user,t,d}$ to enter a new appointment to the Daily Planner (see 7.98):

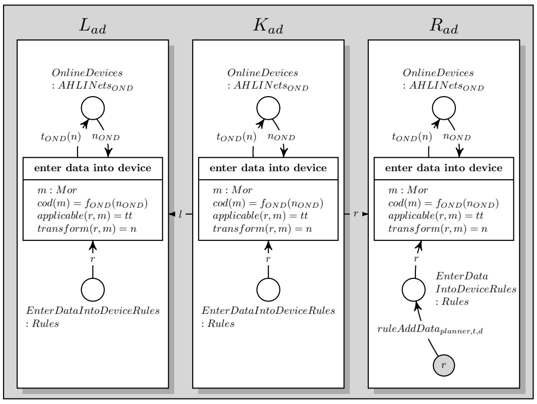


Figure 8.23: Enter Appointment

3. The corresponding general AHLI rule $ruleAddData_{planner,t,d}$ to enter a new appointment to the Daily Planner (see 7.74):

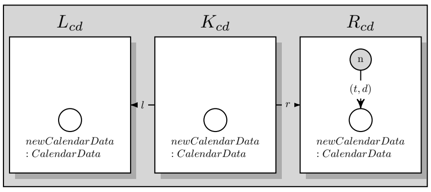
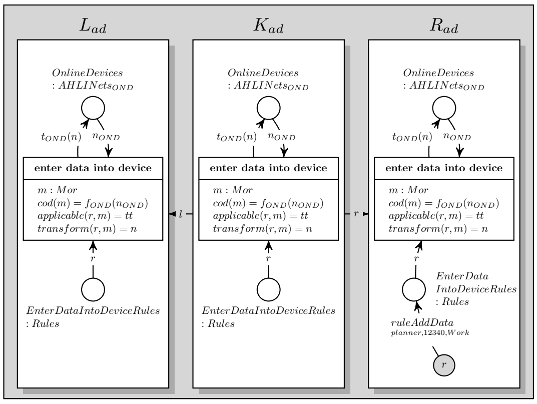


Figure 8.24: Enter Appointment

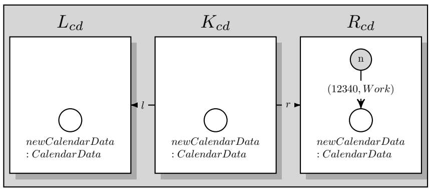
4. Concrete AHOI rule $EnterDataIntoDailyPlanner_{user,12340,Work}$ to enter the appointment "12340, Work" to the Daily Planner:



The concrete AHOI rule $EnterDataIntoDailyPlanner_{user,12340,Work}$ assigns its corresponding AHLI rule $ruleAddData_{planner,12340,Work}$ to the place $EnterDataIntoDeviceRules$ within the system net.

Figure 8.25: Enter Appointment

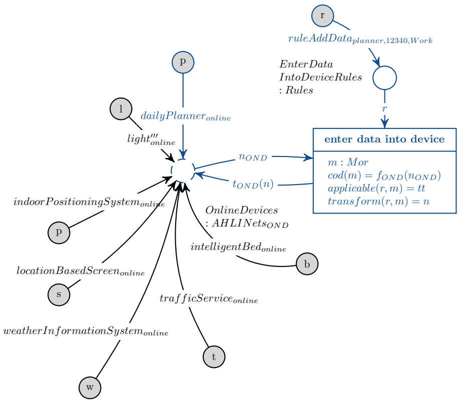
5. Concrete AHLI rule $ruleAddData_{planner,12340,Work}$ to enter the appointment "12340, Work" to the Daily Planner:



The concrete AHLI rule $ruleAddData_{planner,12340,Work}$ assigns the token $(12340, Work)$ to the place $newCalendarData$: $CalendarData$ within the device net $dailyPlanner_{online}$.

Figure 8.26: Enter Appointment

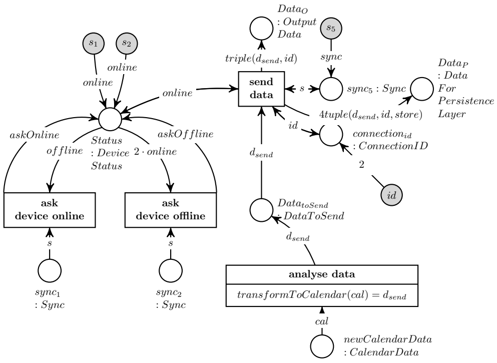
6. Extract from the system net $System^5$ after applying the AHOI rule $EnterDataIntoDailyPlanner_{user,12340,Work}$:



After applying the AHOI rule $EnterDataIntoDailyPlanner_{user,12340,Work}$, a new token refering to the AHLI rule $ruleAddDataplanner, 12340, Work$ is put to the place $EnterDataIntoDeviceRules : Rules$. Now the transition *enter data into device* is able to fire in order to transform the device net of the Daily Planner.

Figure 8.27: Enter Appointment

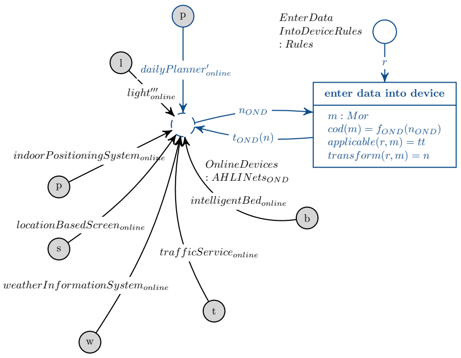
7. Device net $dailyPlanner_{online}$ before entering calendar data (see 7.13):



The image illustrates the object net $dailyPlanner_{online}$ before adding new calendar data to the device net.

Figure 8.28: Enter Appointment

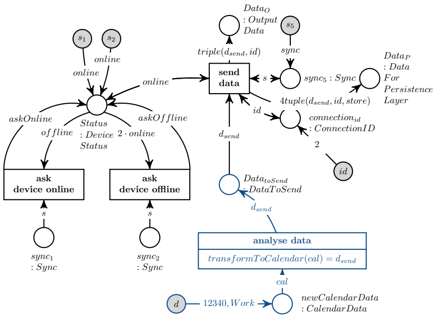
8. Extract from the system net $System^6$ after firing *enter data into device*:


p
EnterData IntoDeviceRules : Rules
r
l
$dailyPlanner'_{online}$
$light'''_{online}$
enter data into device
n_{OND}
$t_{OND}(n)$
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(r, m) = tt$
$transform(r, m) = n$
$indoorPositioningSystem_{online}$
OnlineDevices : $AHLINets_{OND}$
p
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
t
w

After firing the transition *enter data into device*, the rule $ruleAddData_{planner,12340,Work}$ is deleted and the device net of the Daily Planner is modified.

Figure 8.29: Enter Appointment

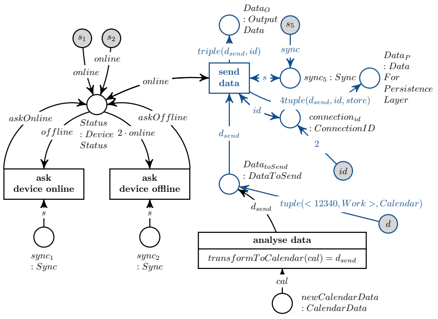
9. Device net $dailyPlanner'_{online}$ after applying the AHLI rules $ruleAddData_{planner,12340,Work}$:



A new calendar data token is assigned to the place *newCalendarData* after applying the rule $ruleAddData_{planner,12340,Work}$. Now, the transition *analyse data* is activated. It executes the operation $transformToCalendar(cal)$ in order to transform the calendar data to a more general data type *DataToSend* consisting of a tuple $Data \times Topic$.

Figure 8.30: Enter Appointment

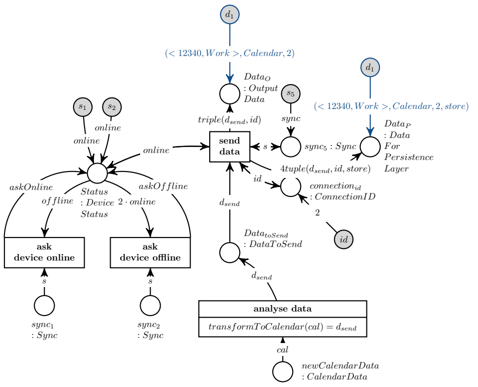
10. Device net $dailyPlanner''_{online}$:



After firing the transition *analyse data* the token assigned to *newCalendarData* is removed and a modified token $(< 12340, Work >, Calendar)$ containing the transformed calendar data and the appropriate topic is assigned to the place $Data_{toSend}$.
The transition *send data* is activated and can be fired in order to send the new appointment to the Message Oriented Middleware.

Figure 8.31: Enter Appointment

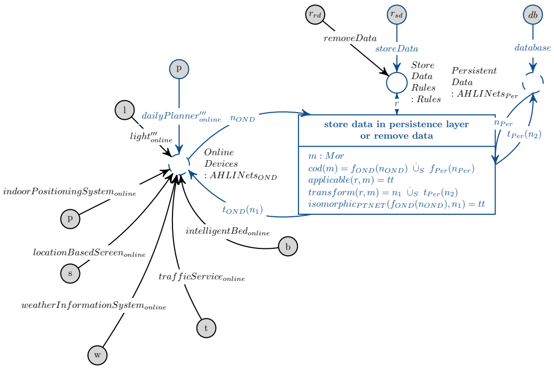
11. Device net $dailyPlanner'''_{online}$:



The transition *send data* has fired and a new token $(< 12340, Work >, Calendar, 2)$ is assigned to $Data_O$ representing the new information, whereas 12340 is the timestamp and *Work* the description of the new appointment, *Calender* is the topic and 2 the connection id of the device. A second token $(< 12340, Work >, Calendar, 2, store)$ is created and assigned to $Data_P$ in order to store the new data within the persistence layer.

Figure 8.32: Enter Appointment

12. Extract from the system net $System^6$ for storing data in the persistence layer (see 7.49):

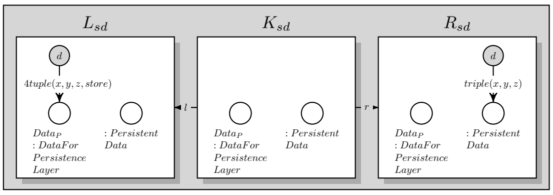

r_{rd}
r_{sd}
db
removeData
storeData
database
Store Data Rules : Rules
Persistent Data : $AHLINets_{Per}$
r
p
l
$dailyPlanner'''_{online}$
n_{OND}
$light'''_{online}$
store data in persistence layer or remove data
$m : Mor$
$cod(m) = f_{OND}(n_{OND}) \dot{\cup}_S f_{Per}(n_{Per})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot{\cup}_S t_{Per}(n_2)$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
n_{Per}
$t_{Per}(n_2)$
Online Devices : $AHLINets_{OND}$
$indoorPositioningSystem_{online}$
p
$t_{OND}(n_1)$
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
t
w

The system net $System^6$ is now able to collect new data from the place $Data_P$ which is provided by the device net $dailyPlanner'''_{online}$ in order to store the data to the internal database of the persistence layer. For that the transition *store data in persistence layer or remove data* is able to fire.

Assigned to the place $StoreDataRules$: $Rules$ are two opposite rules $storeData$ and $removeData$. To distinguish the application of one of both rules, the token assigned to the place $Data_P$: $PersistentData$ within the object net of an online device, is a 4-tuple, consisting of the actual information (e.g. $< 12340, Work >$), the topic (e.g. $Calendar$) and the connection id (e.g. 2). The forth component of the tuple contains a keyword $k \in \{store, remove\}$. It signals, if the data should be stored in applying the rule $storeData$ or if it should be removed in using the rule $removeData$.

Figure 8.33: Enter Appointment, Store Data

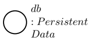
13. AHLI rule *storeData*:



The image illustrates the AHLI rule which is assigned to the place $StoreDateRules$: $Rules$ in n^6. It only stores data, if a token as 4-tuple is assigned to the plave $Data_P$ within the device net, and if the keyword *store* is situated at the last position of the 4-tuple. This indicates, that new data should be stored to the persistence layer with the help of the rule $storeData$ which is set to the place $StoreDataRules$: $Rules$ within the system net.

Figure 8.34: Enter Appointment, Store Data

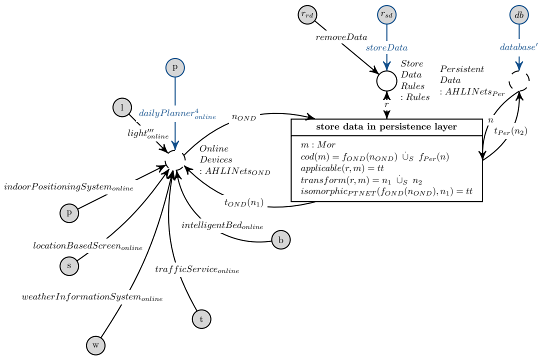
14. Object net *db* representing the database in the persistence layer:



The object net *database* consists of only one place *db* : $PersistentData$ at that stage. During processing, new data will be stored in the persistence layer of the Message Oriented Middleware and assigned as tokens to the place *db* within the object net *database*.

Figure 8.35: Enter Appointment, Store Data

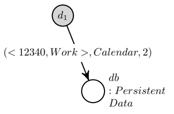
15. Extract from the system net $System^7$ after storing data in the persistence layer:


r_{rd}
r_{sd}
db
removeData
storeData
database′
Store Data Rules : Rules
Persistent Data : AHLINets$_{Per}$
r
p
l
dailyPlanner$^4_{online}$
light‴$_{online}$
n_{OND}
store data in persistence layer
$m : Mor$
$cod(m) = f_{OND}(n_{OND}) \dot{\cup}_S\ f_{Per}(n)$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot{\cup}_S\ n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
n
$t_{Per}(n_2)$
Online Devices : AHLINets$_{OND}$
indoorPositioningSystem$_{online}$
$t_{OND}(n_1)$
p
intelligentBed$_{online}$
b
locationBasedScreen$_{online}$
s
trafficService$_{online}$
weatherInformationSystem$_{online}$
t
w

The transition *store data in persistence layer* was fired and changed the object nets *database′* and *dailyPlanner*^{4} because of the application of the AHLI rule *storeData*.

Figure 8.36: Enter Appointment, Store Data

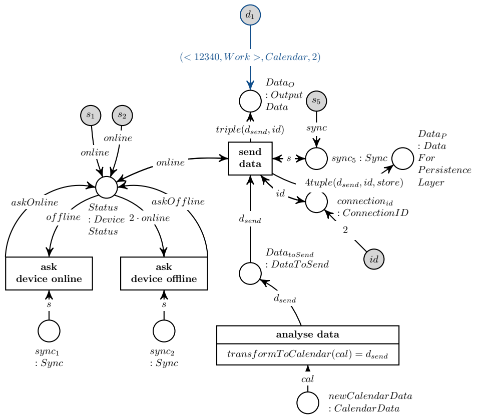
16. Object net *database′* after storing data in the persistence layer:



The object net *database′* is extended by a token $(< 12340, Work >, Calendar, 2)$ representing the first data record in the database of the persistence layer.

Figure 8.37: Enter Appointment, Store Data

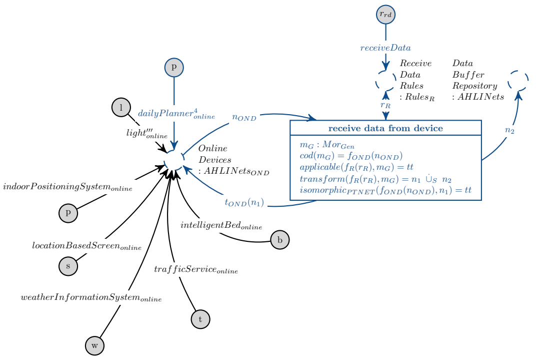
17. Object net $dailyPlanner^4_{online}$ after storing data in the persistence layer:



The device net $dailyPlanner^4_{online}$ was modified after applying the rule *storeData*. The token assigned to $Data_P$ is removed and stored in the persistence layer.
As next step, the output data assigned to $Data_O$ can be processed by the message broker.

Figure 8.38: Enter Appointment, Store Data

18. Extract from system net $System^7$ for receiving data from a device (see 7.33):

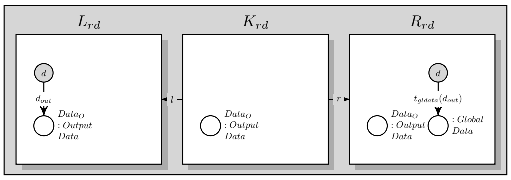


r_{rd}
$receiveData$
$Receive$ $Data$ $Rules$ $: Rules_R$
$Data$ $Buffer$ $Repository$ $: AHLINets$
r_R
p
l
$dailyPlanner^4_{online}$
$light'''_{online}$
n_{OND}
n_2
receive data from device
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OND}(n_{OND})$
$applicable(f_R(r_R), m_G) = tt$
$transform(f_R(r_R), m_G) = n_1 \dot{\cup}_S\ n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
$Online$ $Devices$ $: AHLINets_{OND}$
$indoorPositioningSystem_{online}$
$t_{OND}(n_1)$
p
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
t
w

To receive data from a device, the transition *receive data from device* fires and in this process the rule *receiveData* will be applied to an online device assigned to *OnlineDevices*. A new token which will be assigned to *DataBufferRepository* will be created.

Figure 8.39: Enter Appointment, Receive Data

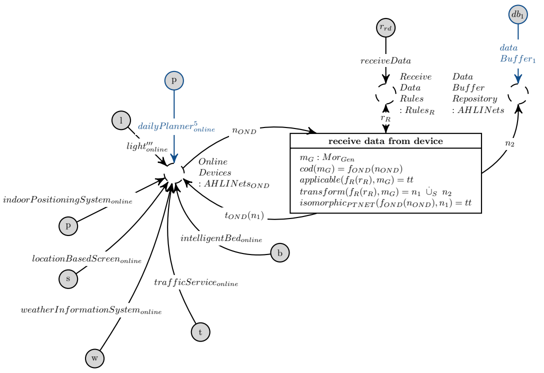
19. AHLI rule *receiveData*:



The AHLI rule *receiveData* copies the token assigned to the place $Data_O : OutputData$ within the device net to a newly created place $: GlobalData$.
As result of the transformation, two disjoint subnets will be created. The one being isomorphic in the category of *PTNET* to the original device net, will be put to *OnlineDevices*. The other will be assigned as new token to *DataBufferRepository*.

Figure 8.40: Enter Appointment, Receive Data

20. Extract from system net $System^8$ after receiving data from a device (see 7.33):



db_1
$data$ $Buffer_1$
r_{rd}
$receiveData$
$Receive$ $Data$ $Rules$ $: Rules_R$
$Data$ $Buffer$ $Repository$ $: AHLINets$
r_R
p
l
$dailyPlanner^5_{online}$
$light'''_{online}$
n_{OND}
n_2
receive data from device
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OND}(n_{OND})$
$applicable(f_R(r_R), m_G) = tt$
$transform(f_R(r_R), m_G) = n_1 \dot{\cup}_S\ n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
$Online$ $Devices$ $: AHLINets_{OND}$
$indoorPositioningSystem_{online}$
$t_{OND}(n_1)$
p
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
t
w

To receive data from a device, the transition *receive data from device* fires and in this process the rule *receiveData* will be applied to an online device assigned to *OnlineDevices* and a new token which will be assigned to *DataBufferRepository* will be created. Both resulting object nets are shown in the following images 21. and 22.

Figure 8.41: Enter Appointment, Receive Data

21. Object net $dataBuffer_1$ after applying the rule *receiveData*:

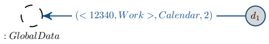


Figure 8.42: Enter Appointment, Receive Data

22. Object net $dailyPlanner^5_{online}$ after applying the rule *receiveData*:

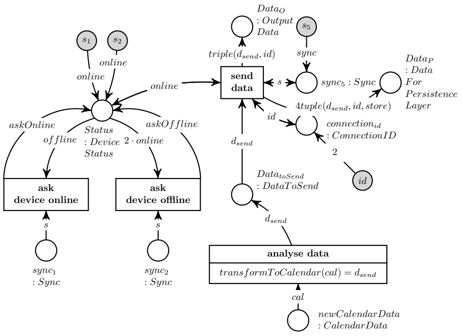
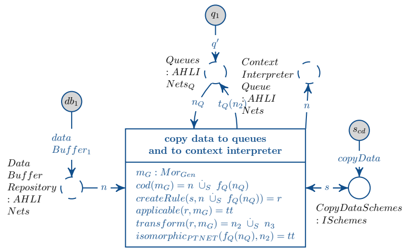


Figure 8.43: Enter Appointment, Receive Data

23. Extract from system net $System^8$ for distributing data (see 7.35):

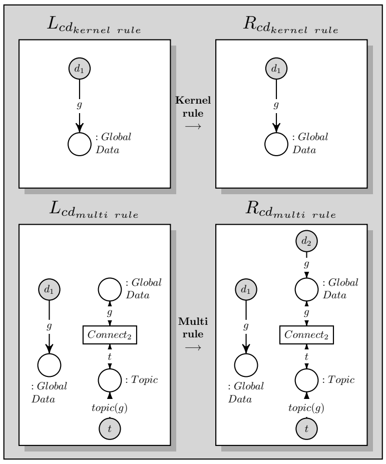

q_1
q'
Queues
: AHLI
$Nets_Q$
Context
Interpreter
Queue
: AHLI
Nets
n_Q
$t_Q(n_2)$
n
db_1
data
$Buffer_1$
Data
Buffer
Repository
: AHLI
Nets
n
copy data to queues
and to context interpreter
$m_G : Mor_{Gen}$
$cod(m_G) = n \dot{\cup}_S \ f_Q(n_Q)$
$createRule(s, n \dot{\cup}_S \ f_Q(n_Q)) = r$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_2 \dot{\cup}_S \ n_3$
$isomorphic_{PTNET}(f_Q(n_Q), n_2) = tt$
s_{cd}
copyData
s
CopyDataSchemes
: ISchemes

To distribute data, the transition *copy data to queues and to context interpreter* will be fired which applies an amalgamated rule generated out of the interaction scheme *copyData* assigned to $CopyDataSchemes : ISchemes$. The generated rule contains the maximum match regarding the object nets $dataBuffer_1$ and q'. Then, the object net $dataBuffer_1$ will be taken and the information represented by this AHLI net will be copied to each subnet in q' assigned to $Queues$, which represents a device subscribed to the specific topic provided by the input token. Additionally the input token from $DataBufferRepository : AHLINets$ will be directly copied to the place $ContextInterpreterQueue$. The processing of the information within the Context Interpreter will be presented in explicit in images 6-13 in part five of the simulation of this scenario in section 8.6.

Figure 8.44: Enter Appointment, Distribute Data

- - →

24. System interaction scheme *copyData*:

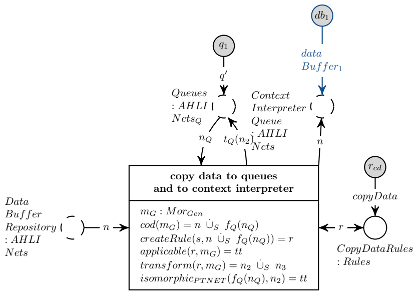


The interaction scheme *copyData* will be used to create a rule with the maximum number of matches which will is able to be applied. The resulting rule gets an information belonging to a certain topic and copies the data token to each subnet representing a device subscribed to the specific topic.

I.e., at this stage, the data triple $(< 12340, Work >, Calendar, 2)$ is assigned to $: GlobalData$. It will be copied to each queue referring to the topic "Calendar". Because of no device, which is plugged in at this moment, is subscribed to this specific topic, no data will be copied. In section 8.4, this rule will be used to copy data.

Figure 8.45: Enter Appointment, Distribute Data

25. Extract from system net $System^8$ after the distribution of data:


db_1
$data\ Buffer_1$
q_1
q'
Queues : AHLI $Nets_Q$
Context Interpreter Queue : AHLI Nets
n_Q
$t_Q(n_2)$
n
r_{cd}
copyData
copy data to queues and to context interpreter
$m_G : Mor_{Gen}$
$cod(m_G) = n \dot{\cup}_S\ f_Q(n_Q)$
$createRule(s, n \dot{\cup}_S\ f_Q(n_Q)) = r$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_2 \dot{\cup}_S\ n_3$
$isomorphic_{PTNET}(f_Q(n_Q), n_2) = tt$
Data Buffer Repository : AHLI Nets
n
r
CopyDataRules : Rules

After finishing the distribution of the information, the token $dataBuffer_1$ formerly assigned to $DataBufferRepository$ is copied to $ContextInterpreterQueue$. As mentioned before, the object net q' stays unchanged.

Figure 8.46: Enter Appointment, Distribute Data

The user enters new data to the Daily Planner as presented in this section. For that, the user inputs the data with the help of the AHOI rule $EnterDataIntoDailyPlanner_{user}$. The AHOI rule adds an appointment to the device net on Object Level with the help of the AHLI rule $ruleAddData_{planner}$ which mediates between both levels. Afterwards, the device net processes the message and provides it once for direct storing it into the persistence layer and one data record available for the processing in the message broker. The system net representing the Message Oriented Middleware will collect both types of data and stores one data record directly to the database of persistence layer. The second data record will be processed, sent to the Context Interpreter and to the devices subscribed to the specific topic. At the moment, presented in this section, no devices are interested in data concerning the topic "Calendar", so no data will be copied for other devices.

8.4 Attach Alarm Clock 2.0 and Display to the Message Oriented Middleware and Send Request

In the next phase, the Alarm Clock 2.0 and the Display of the Multitouch Kitchen Counter are connected to the Living Place system and subscribed to all necessary topics enumerated below: The Alarm Clock 2.0 will be subscribed to the topics (see 7.1.2.7):

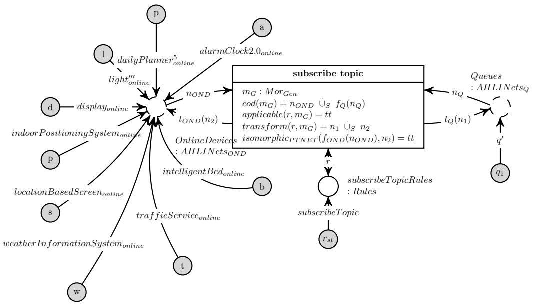
- Weather,
- Traffic,
- Calendar,
- Bed and
- Context.

Whereas the Display will be subscribed to (see 7.1.2.8):

- Weather,
- Traffic and
- Calendar.

The process of plugging-in is analogue to the steps presented in section 8.2 and will not be presented in explicit. The resulting system net $System^9$ and object net q'' are illustrated below in 1. and 2.

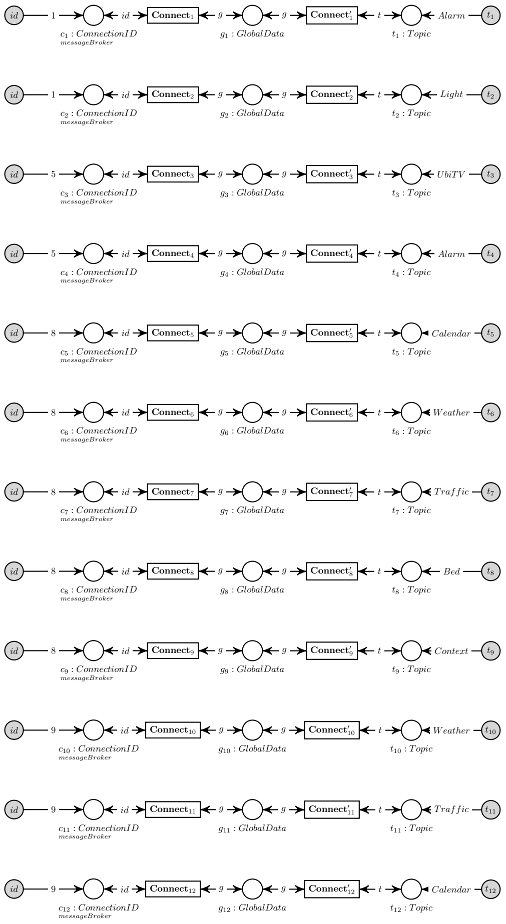
1. Extract from the system net $System^9$ after plugging-in all devices and subscribing their topics:


p
a
l
$dailyPlanner^5_{online}$
$alarmClock2.0_{online}$
$light'''_{online}$
subscribe topic
$m_G : Mor_{Gen}$
$cod(m_G) = n_{OND} \dot\cup_S f_Q(n_Q)$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_1 \dot\cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_2) = tt$
n_{OND}
n_Q
Queues : $AHLINets_Q$
d
$display_{online}$
$t_{OND}(n_2)$
$t_Q(n_1)$
$indoorPositioningSystem_{online}$
OnlineDevices : $AHLINets_{OND}$
q'
p
r
q_1
$intelligentBed_{online}$
b
subscribeTopicRules : Rules
$locationBasedScreen_{online}$
s
subscribeTopic
$trafficService_{online}$
r_{st}
$weatherInformationSystem_{online}$
t
w

All devices are set into online mode and assigned to *OnlineDevices*. A visual description of each device in online mode is given in chaper 7:

1. $light'''_{online}$, connection ID 1,
2. $dailyPlanner_{online}$, connection ID 2,
3. $intelligentBed_{online}$, connection ID 3,
4. $indoorPositioningSystem_{online}$, connection ID 4,
5. $locationBasedScreen_{online}$, connection ID 5,
6. $weatherInformationSystem_{online}$, connection ID 6,
7. $trafficService_{online}$, connection ID 7,
8. $alarmClock2.0_{online}$, connection ID 8,
9. $display_{online}$, connection ID 9.

2. Object net q'' contains subnets representing all subscriptions:


id 1 id $Connect_1$ g g $Connect'_1$ t Alarm t_1
c_1 : ConnectionID messageBroker g_1 : GlobalData t_1 : Topic
id 1 id $Connect_2$ g g $Connect'_2$ t Light t_2
c_2 : ConnectionID messageBroker g_2 : GlobalData t_2 : Topic
id 5 id $Connect_3$ g g $Connect'_3$ t UbiTV t_3
c_3 : ConnectionID messageBroker g_3 : GlobalData t_3 : Topic
id 5 id $Connect_4$ g g $Connect'_4$ t Alarm t_4
c_4 : ConnectionID messageBroker g_4 : GlobalData t_4 : Topic
id 8 id $Connect_5$ g g $Connect'_5$ t Calendar t_5
c_5 : ConnectionID messageBroker g_5 : GlobalData t_5 : Topic
id 8 id $Connect_6$ g g $Connect'_6$ t Weather t_6
c_6 : ConnectionID messageBroker g_6 : GlobalData t_6 : Topic
id 8 id $Connect_7$ g g $Connect'_7$ t Traffic t_7
c_7 : ConnectionID messageBroker g_7 : GlobalData t_7 : Topic
id 8 id $Connect_8$ g g $Connect'_8$ t Bed t_8
c_8 : ConnectionID messageBroker g_8 : GlobalData t_8 : Topic
id 8 id $Connect_9$ g g $Connect'_9$ t Context t_9
c_9 : ConnectionID messageBroker g_9 : GlobalData t_9 : Topic
id 9 id $Connect_{10}$ g g $Connect'_{10}$ t Weather t_{10}
c_{10} : ConnectionID messageBroker g_{10} : GlobalData t_{10} : Topic
id 9 id $Connect_{11}$ g g $Connect'_{11}$ t Traffic t_{11}
c_{11} : ConnectionID messageBroker g_{11} : GlobalData t_{11} : Topic
id 9 id $Connect_{12}$ g g $Connect'_{12}$ t Calendar t_{12}
c_{12} : ConnectionID messageBroker g_{12} : GlobalData t_{12} : Topic

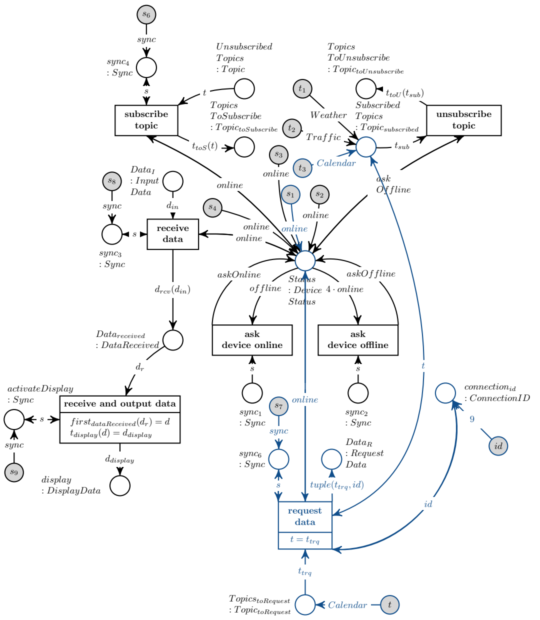
Now, all devices are connected and subscribed to their necessary topics.

Figure 8.47: Attach a Alarm Clock 2.0 and Display

Figure 8.48: Attach a Alarm Clock 2.0 and Display

After subscribing all necessary topics, connected devices in online mode are able to request old data from the persistence layer, if they are either a receiver or a transceiver and if they have at least one token assigned to the place $Topics_{toRequest}$ within their individual device net. Only a few devices need old data from the persistence layer, in the current modeling, the device nets $alarmClock2.0_{online}$ and $display_{online}$ request old data. Both devices request the topic “Calendar” meaning that old calendar data are relevant for both devices. The process of requesting past data by a device on Object Level, the collection of appropriate information within the persistence layer of the Message Oriented Middleware on System Level and the receiving of the data will be described in explicit for the Daily Planner. It is analogue for the Alarm Clock 2.0.

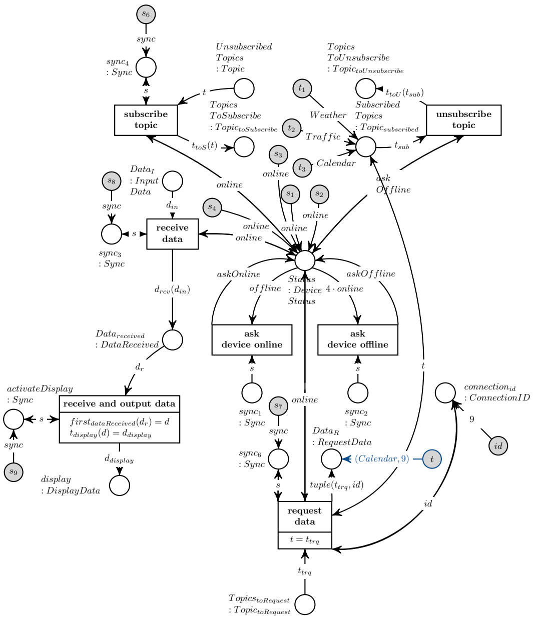
3. Device net $display_{online}$ after plugging-in all devices and subscribing all topics (see 7.11):



After being subscribed to all topics, a request for past data from the persistence layer can be executed. For that, the transition *request data* needs to fire.

Figure 8.49: Request Data From Persistence Layer

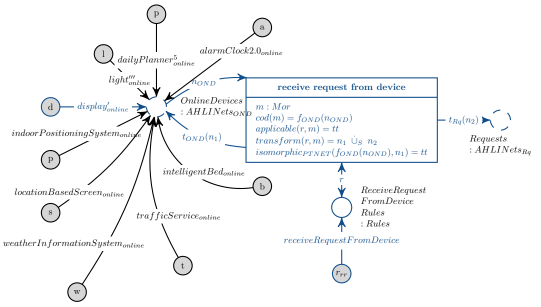
4. Device net $display'_{online}$ ready to request data from the topic "Calendar":



After firing the transition *request data*, a tuple containing the topic and the specific connection identification $(Calendar, 9)$ is transferred to the place $Data_R$.

Figure 8.50: Request Data From Persistence Layer

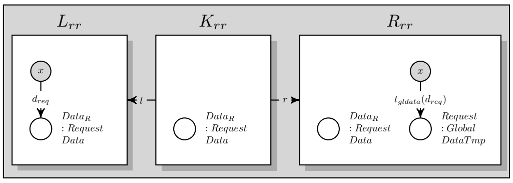
5. Extract from the system net $System^9$ showing part for receiving a request from a device (see 7.51):


receive request from device
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
OnlineDevices : $AHLINets_{OND}$
Requests : $AHLINets_{Rq}$
ReceiveRequest FromDevice Rules : Rules
receiveRequestFromDevice
r_{rr}
$display'_{online}$
$alarmClock2.0_{online}$
$intelligentBed_{online}$
$trafficService_{online}$
$weatherInformationSystem_{online}$
$locationBasedScreen_{online}$
$indoorPositioningSystem_{online}$

To receive a request from a device, the transition *receive request from device* in the system net needs to fire. Then, it applies the AHLI rule *receiveRequestFromDevice*.

Figure 8.51: Request Data From Persistence Layer

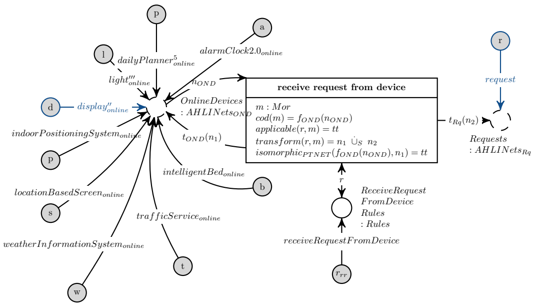
6. AHLI rule $receiveRequestFromDevice$:



The AHLI rule $receiveRequestFromDevice$ removes a token assigned to $Data_R$ in the device net and copies it to a newly created place $Request : GlobalDataTmp$.

Figure 8.52: Request Data From Persistence Layer

7. Extract from the system net $System^{10}$ after receiving a request from a device:


p
l
a
r
$dailyPlanner^5_{online}$
$alarmClock2.0_{online}$
$light'''_{online}$
n_{OND}
request
d
$display''_{online}$
OnlineDevices
: $AHLINets_{OND}$
receive request from device
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(r, m) = tt$
$transform(r, m) = n_1 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
$t_{Rq}(n_2)$
Requests
: $AHLINets_{Rq}$
$indoorPositioningSystem_{online}$
$t_{OND}(n_1)$
p
$intelligentBed_{online}$
b
r
$locationBasedScreen_{online}$
ReceiveRequest
FromDevice
Rules
: Rules
s
$trafficService_{online}$
$weatherInformationSystem_{online}$
receiveRequestFromDevice
t
r_{rr}
w

After the Message Oriented Middleware has collected the request in firing the transition *receive request from device*, the device net $display''_{online}$ is modified as shown in step 9. and a new token is assigned to the place *RequestsAHLINets*, which is presented in step 8.

Figure 8.53: Request Data From Persistence Layer

8. Object net *request* containing the request:

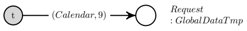
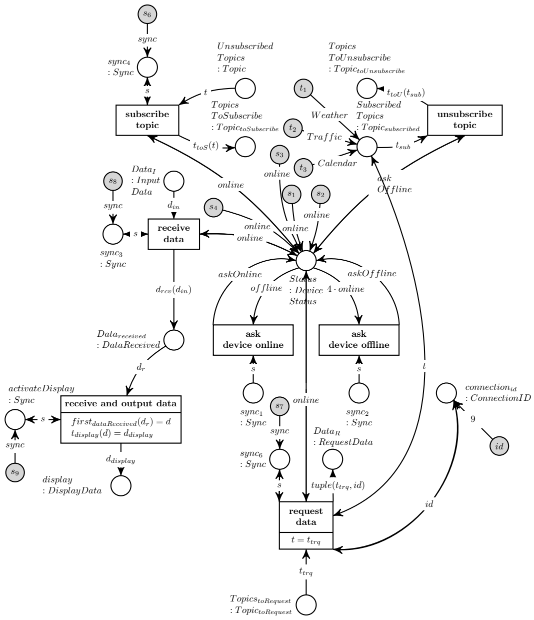


Figure 8.54: Request Data From Persistence Layer

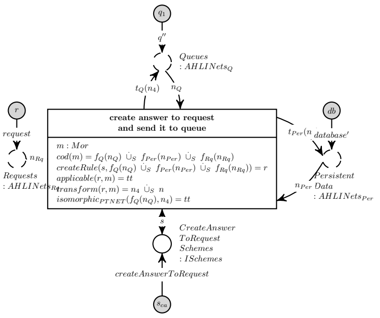
9. Device net $display''_{online}$:



The token containing the request data to the persistence layer is removed.

Figure 8.55: Request Data From Persistence Layer

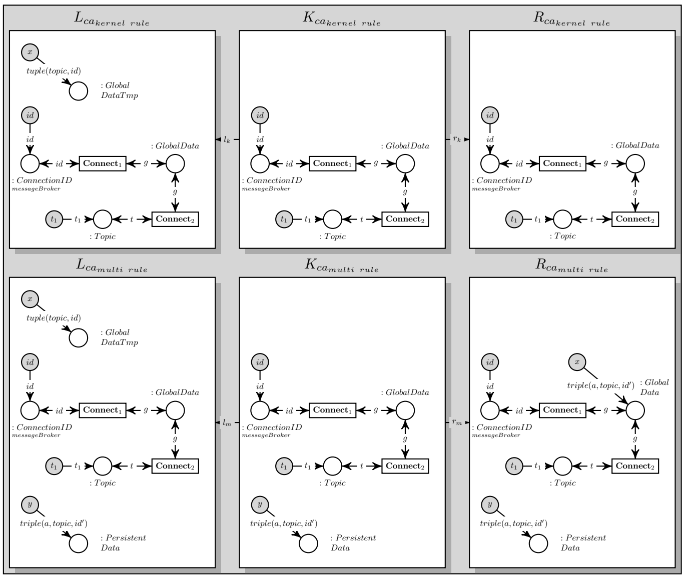
10. Extract from the system net $System^{10}$ showing part for creating an answer to the request (see 7.52):



Within the persistence layer of the Message Oriented Middleware, the request needs to be processed. For that, the transition *create answer to request and send it to queue* will fire in order to apply a rule calculated with the help of the interaction scheme *createAnswerToRequest*. The transition gets the objects nets *request*, q'' and *database'* as input parameters and executes the request in copying all data according to the request to the corresponding queue. The object net *database'* stays unchanged, whereas the processed request will be removed and the object net q'' will be extended by tokens fulfiling the requested data from the persistence layer.

Figure 8.56: Request Data From Persistence Layer, Create Answer

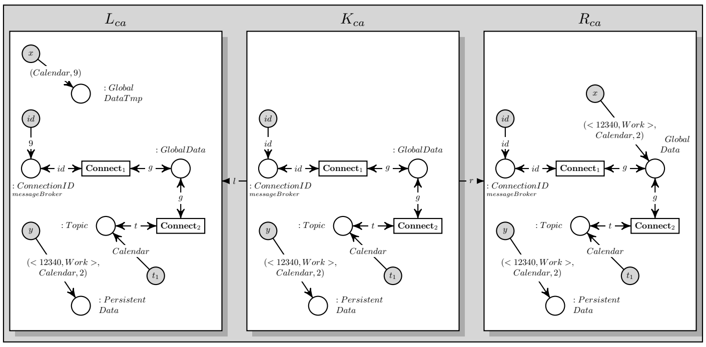
11. Interaction scheme *createAnswerToRequest* (see 7.52):



The interaction scheme *createAnswerToRequest* serves as template for creating a rule which will be applied while firing the transition *create answer to request and send it to queue*. The formal description of the process of creating a rule out of an interaction scheme is presented in chapter 6.

Figure 8.57: Request Data From Persistence Layer, Create Answer

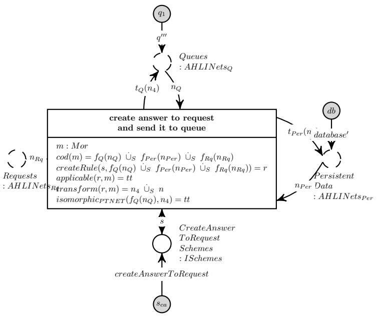
12. Concrete rule r created out of interaction scheme *createAnswerToRequest* and the three object nets *request*, q'' and *database*$'$:



When firing the transition *create answer to request and send it to queue*, the operation *createRules* gets the interaction scheme *createAnswerToRequest* and the disjoint union of the three nets $q'' \dot{\cup}_S \, database' \dot{\cup}_S \, request$ and generates the concrete rule r out of them as described in detail in chapter 6, which will be applied.

Figure 8.58: Request Data From Persistence Layer, Create Answer

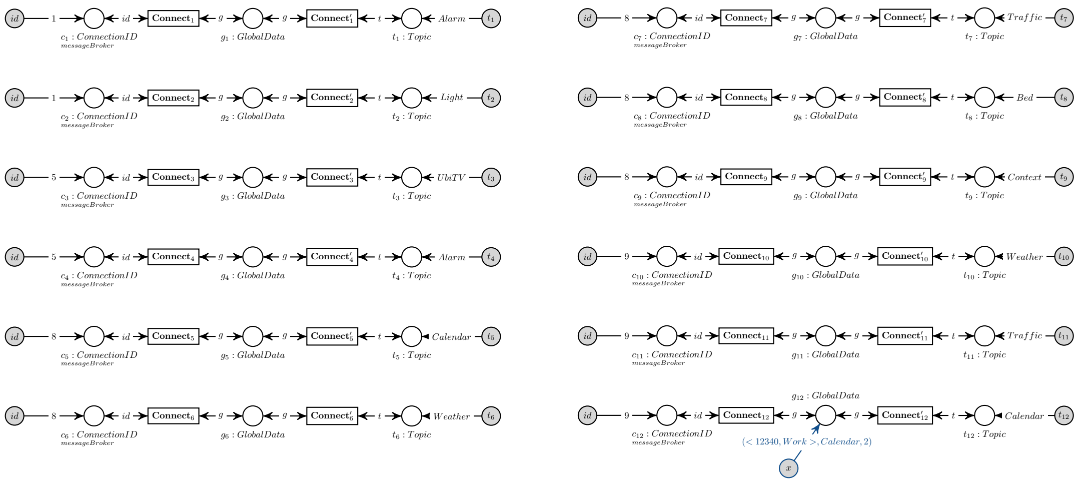
13. Extract from system net $System^{11}$ after firing *create answer to request and send it to queue*:


q_1
q'''
Queues
: $AHLINets_Q$
$t_Q(n_4)$
n_Q
db
$t_{Per}(n$
database$'$
create answer to request and send it to queue
$m : Mor$
$cod(m) = f_Q(n_Q) \dot{\cup}_S \; f_{Per}(n_{Per}) \dot{\cup}_S \; f_{Rq}(n_{Rq})$
$createRule(s, f_Q(n_Q) \dot{\cup}_S \; f_{Per}(n_{Per}) \dot{\cup}_S \; f_{Rq}(n_{Rq})) = r$
$applicable(r, m) = tt$
$transform(r, m) = n_4 \dot{\cup}_S \; n$
$isomorphic_{PTNET}(f_Q(n_Q), n_4) = tt$
n_{Rq}
Requests
: $AHLINets_{Rq}$
Persistent Data
: $AHLINets_{Per}$
n_{Per}
s
CreateAnswer ToRequest Schemes
: ISchemes
createAnswerToRequest
s_{ca}

As a result, the token assigned to *Requests* : *AHLINets* is removed and the object net q''' is transformed as shown in the next illustration.

Figure 8.59: Request Data From Persistence Layer, Create Answer

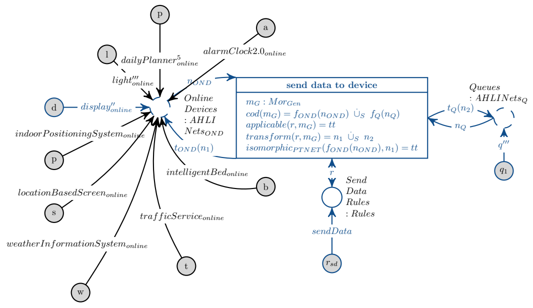
14. Object net q''':



After transforming with the help of the rule r resulting from the interaction scheme *createAnswerToRequest* and the disjoint union of the three nets $q'' \dot{\cup}_S \, database' \dot{\cup}_S \, request$, the resulting object net q''' is extended by one new token assigned to the place g_{12} : *GlobalData*, which represents the message queue regarding the topic *Calendar* of the device with the connection identification 9, i.e. of the Display device.

Figure 8.60: Enter Appointment, Distribute Data

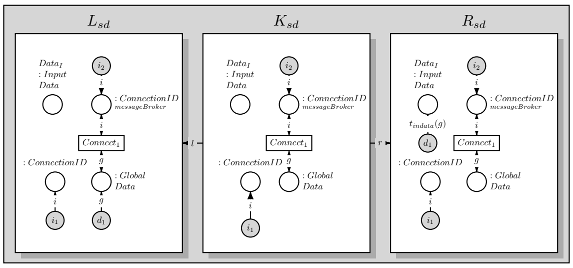
15. Extract from system net $System^{11}$ showing part for sending data to a device (see 7.36):


send data to device
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OND}(n_{OND}) \cup_S f_Q(n_Q)$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_1 \cup_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
Online Devices : AHLI Nets$_{OND}$
Queues : AHLINets$_Q$
Send Data Rules : Rules
sendData
r_{sd}
$display''_{online}$
$dailyPlanner^5_{online}$
$light'''_{online}$
$alarmClock2.0_{online}$
$indoorPositioningSystem_{online}$
$intelligentBed_{online}$
$locationBasedScreen_{online}$
$trafficService_{online}$
$weatherInformationSystem_{online}$
q'''
q_1

To send data to a device, the transition *send data to device* is activated and is able to fire. In doing this, the rule *sendData* described in next illustration is applied.

Figure 8.61: Request Data From Persistence Layer, Send Answer

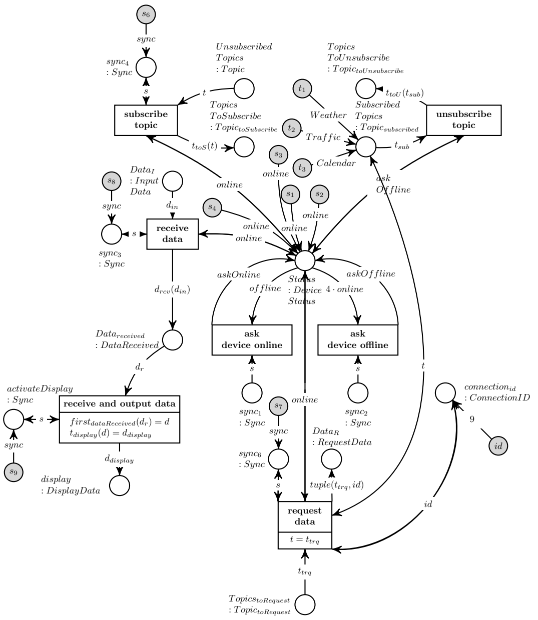
16. Rule *sendData* (see 7.36):



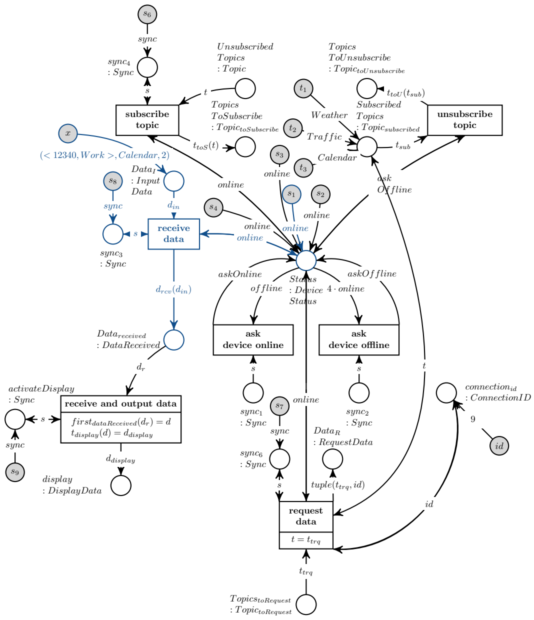
The rule *sendData* gets a disjoint union of a device net, at this stage the device net $display''_{online}$, and the AHLI net assigned to $Queues : AHLINets$, currently it is q'''. When applying, this rule copies the data assigned to the specific message queue belonging to this device to the place $Data_I$: $InputData$ of the device net. $Data_I$ represents the input buffer of the device.

Figure 8.62: Request Data From Persistence Layer, Send Answer

17. Object net $display''_{online}$ before firing *send data to device*:


subscribe topic
unsubscribe topic
receive data
ask device online
ask device offline
receive and output data
$first_{dataReceived}(d_r) = d$
$t_{display}(d) = d_{display}$
request data
$t = t_{trq}$
Unsubscribed Topics : Topic
Topics ToUnsubscribe : $Topic_{toUnsubscribe}$
Topics ToSubscribe : $Topic_{toSubscribe}$
Subscribed Topics : $Topic_{subscribed}$
$Data_I$: Input Data
$Data_{received}$: DataReceived
Status : Device Status
$Data_R$: RequestData
display : DisplayData
$connection_{id}$: ConnectionID
$Topics_{toRequest}$: $Topic_{toRequest}$
activateDisplay : Sync
Weather
Traffic
Calendar

18. Object net $display'''_{online}$ after firing *send data to device*:

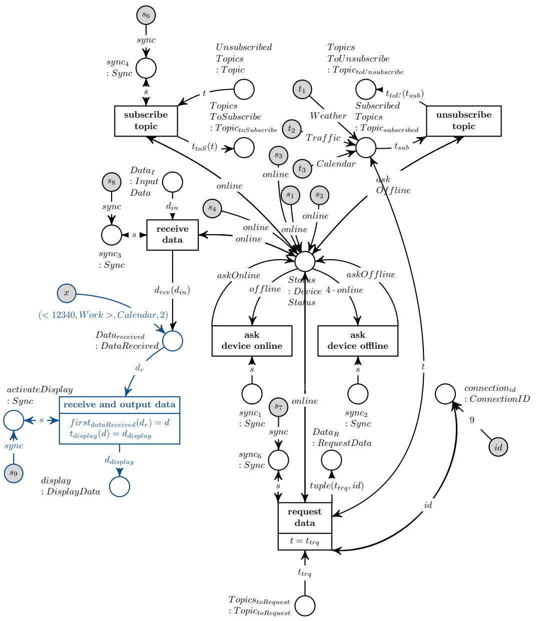


After receiving the new data token on $Data_I$: $InputData$, the transition *receive data* is activated in order to process the information.

Figure 8.63: Request Data From Persistence Layer, Send Answer

Figure 8.64: Request Data From Persistence Layer, Send Answer

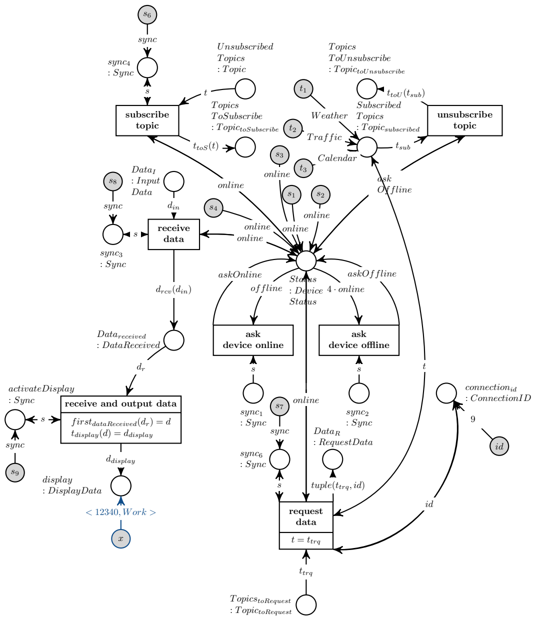
19. Object net $display^4_{online}$ while processing new data:



Afterwards, the data token is able to be processed by the transition *receive and output data*. It executes two operations in order to extract the actual information $< 12340, Work >$ and forwards it to the place $display : DisplayData$.

Figure 8.65: Request Data From Persistence Layer, Send Answer

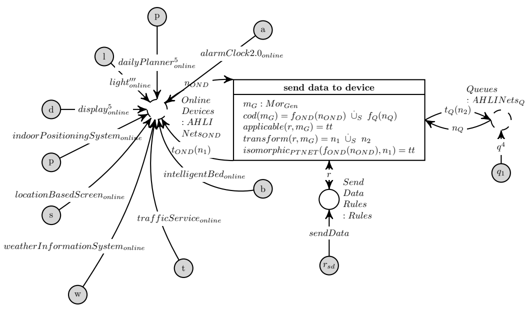
20. Object net $display^5_{online}$ while processing new data:



The data token $< 12340, Work >$ finally arrived at the place $display : DisplayData$, which represents the screen and its currently shown output data of the Display device in the Multitouch Kitchen Counter.

Figure 8.66: Request Data From Persistence Layer, Send Answer

21. Extract from system net $System^{12}$ after processing the requests of the Display device:



After processing the request of the Display device, the system net $System^{12}$ is in the presented state. The device net of the Display was transformed to $display^5_{online}$ and the object net q^4 assigned to $Queues : AHLINets$ is equal to the net q'' shown in figure 2 in this step. The token formerly set to $g_{12} : GlobalData$ was removed after firing the transition *send data to device* in the system net and simultaneously applying the rule *sendData*.

Figure 8.67: Request Data From Persistence Layer

Additionally to the Display device, the Alarm Clock 2.0 needs to send its request for data belonging to the topic “Calendar” to the persistence layer of the Message Oriented Middleware. The processing of the Alarm Clock 2.0’s request is analogue to the steps described in explicit for the Display. Thus, the resulting object net $alarmClock2.0'_{online}$ will be shown in the following illustration:

22. Object net $alarmClock2.0'_{online}$ after processing its requests:

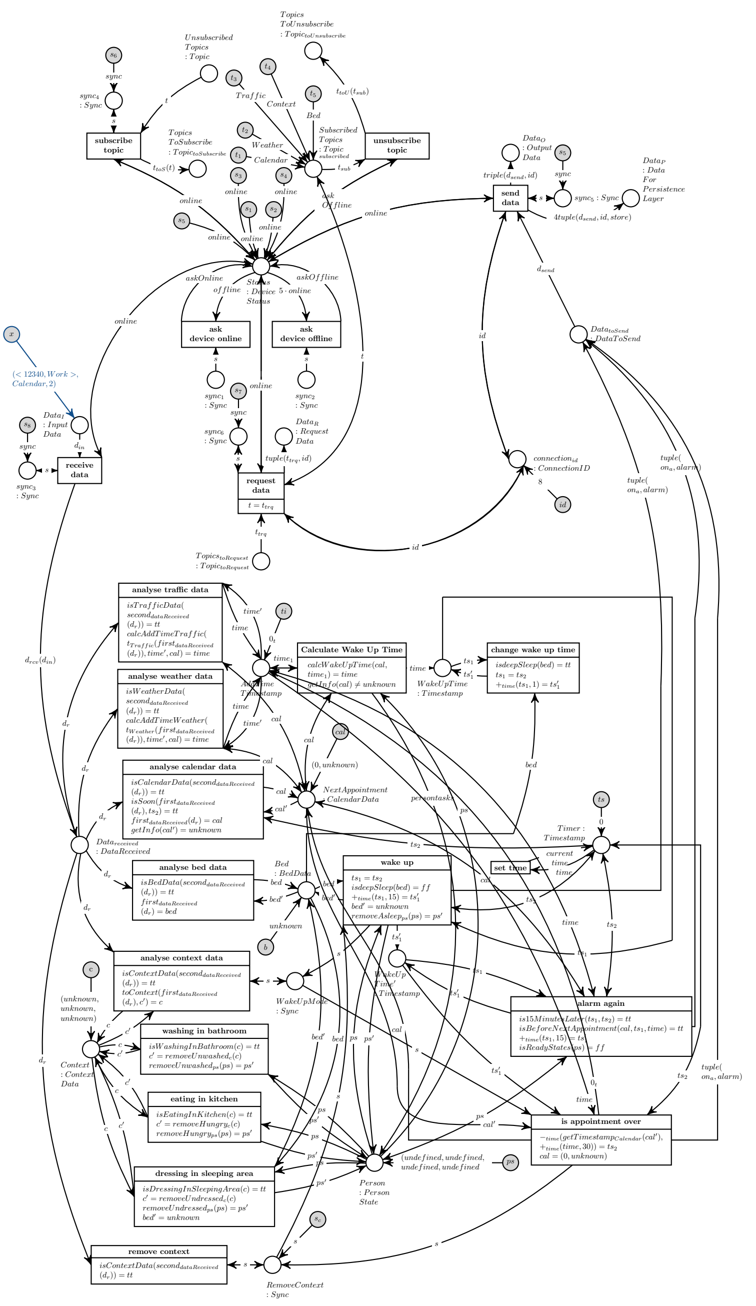
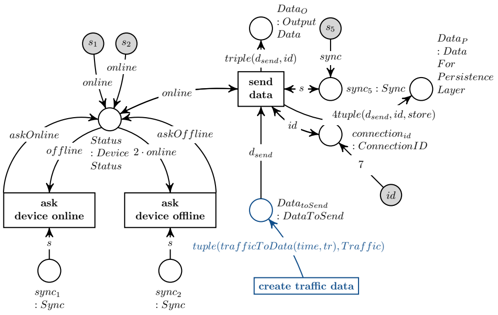


Figure 8.68: Request Data From Persistence Layer

8.5 Wake up the Inhabitant

The user has entered an appointment in step 2 (see 8.3). If the user is in bed sleeping and the appointment gets closer, the Alarm Clock 2.0 needs to infer an appropriate wake up time according to the sleeping phase the inhabitant is in. The Alarm Clock 2.0 should no alert the user in deep sleep phase. Additionally the weather and traffic conditions need to be taken into account for calculating the suitable wake up time. When the inhabitant should be woken up, the Ambient Light will be switched on in dimmed mode and additionally, the TV sets of the Location-Based Screen will be activated and switched on.

1. Object net $trafficService_{online}$ **creates** $(free, 12340)$ (see 7.1.2.15.2):



The device net of the traffic service generates a new token in firing *create traffic data*.

Figure 8.69: Traffic Service Sends $(free, 12340)$

2. Object net $trafficService'_{online}$ **sends data tuple** $(free, 12340)$:

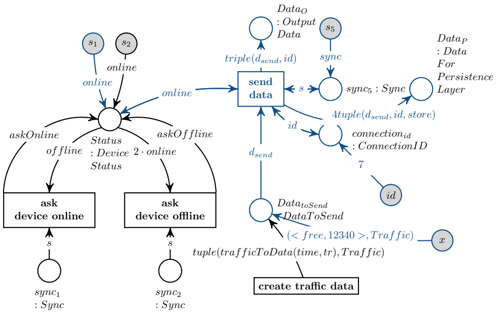
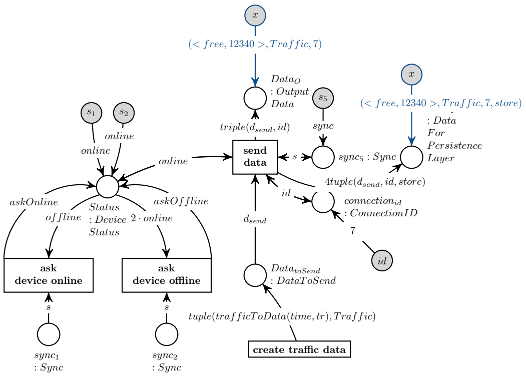

$Data_O$: $Output$ $Data$
s_5
s_1
s_2
$Data_P$: $Data$ For $Persistence$ $Layer$
$triple(d_{send}, id)$
$sync$
$online$
$online$
$online$
send data
s
$sync_5 : Sync$
$4tuple(d_{send}, id, store)$
id
$connection_{id}$: $ConnectionID$
$askOnline$
$askOffline$
$offline$
$Status$: $Device$ $Status$
$2 \cdot online$
d_{send}
7
id
ask device online
ask device offline
$Data_{toSend}$: $DataToSend$
$(< free, 12340 >, Traffic)$
x
s
s
$tuple(trafficToData(time, tr), Traffic)$
create traffic data
$sync_1$: $Sync$
$sync_2$: $Sync$

Figure 8.70: Traffic Service Sends $(free, 12340)$

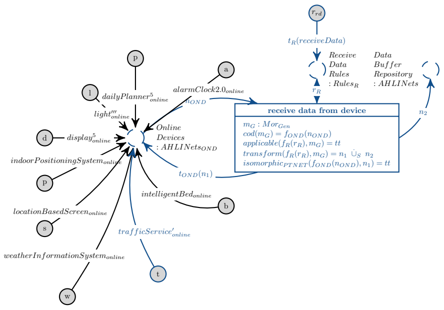
3. Object net $trafficService''_{online}$ **sends data tuple** $(free, 12340)$:


x
$(< free, 12340 >, Traffic, 7)$
x
$Data_O$: $Output$ $Data$
s_5
$(< free, 12340 >, Traffic, 7, store)$
s_1
s_2
: $Data$ For $Persistence$ $Layer$
$triple(d_{send}, id)$
$sync$
$online$
$online$
$online$
send data
s
$sync_5 : Sync$
$4tuple(d_{send}, id, store)$
id
$connection_{id}$: $ConnectionID$
$askOnline$
$askOffline$
$offline$
$Status$: $Device$ $Status$
$2 \cdot online$
d_{send}
7
id
ask device online
ask device offline
$Data_{toSend}$: $DataToSend$
s
s
$tuple(trafficToData(time, tr), Traffic)$
create traffic data
$sync_1$: $Sync$
$sync_2$: $Sync$

After generating a new token $free$, it will be converted and transferred to the places $Data_P$: $DataForPersistenceLayer$ and to $Data_O$: $OutputData$, ready to be processed by the Message Oriented Middleware on System Level.
The new information will be sent to other devices, (the token for that is assigned to $Data_O$), and stored to the persistence layer (for this task, a token is assigned to $Data_P$).

Figure 8.71: Traffic Service Sends $(free, 12340)$

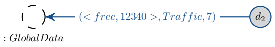
4. Extract from the system net $System^{12}$ **showing part for receiving data**:


r_{rd}
$t_R(receiveData)$
$Receive$ $Data$ $Rules$: $Rules_R$
$Data$ $Buffer$ $Repository$: $AHLINets$
p
a
l
$dailyPlanner^5_{online}$
$alarmClock2.0_{online}$
n_{OND}
$light'''_{online}$
r_R
receive data from device
n_2
$m_G : Mor_{Gen}$
$cod(m_G) = f_{OND}(n_{OND})$
$applicable(f_R(r_R), m_G) = tt$
$transform(f_R(r_R), m_G) = n_1 \dot{\cup}_S n_2$
$isomorphic_{PTNET}(f_{OND}(n_{OND}), n_1) = tt$
d
$display^5_{online}$
$Online$ $Devices$: $AHLINets_{OND}$
$indoorPositioningSystem_{online}$
p
$t_{OND}(n_1)$
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService'_{online}$
$weatherInformationSystem_{online}$
t
w

As next step, the transition *receive data from device* can be fired. It applies the AHLI rule $receiveData$, which is already mentioned in the second step (see section 8.3) in image 19.

Figure 8.72: Traffic Service Sends $(free, 12340)$

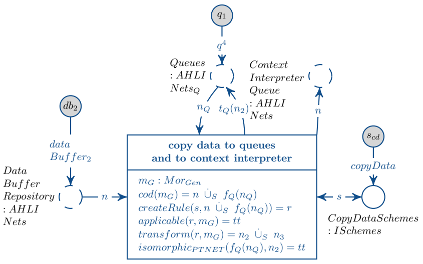
5. Object net $dataBuffer_2$ **assigned to** $DataBufferRepository : AHLINets$:



After the application of the AHLI rule $receiveData$, a new object net $dataBuffer_2$ was created which contains the information to process furter.

Figure 8.73: Traffic Service Sends $(free, 12340)$

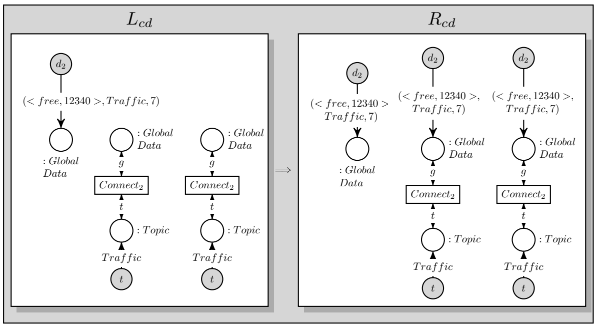
6. Extract from system net $System^{12}$ for distributing data:


q_1
q^4
Queues : AHLI Nets$_Q$
Context Interpreter Queue : AHLI Nets
n_Q
$t_Q(n_2)$
n
db_2
data Buffer$_2$
s_{cd}
copyData
Data Buffer Repository : AHLI Nets
n
s
copy data to queues and to context interpreter
$m_G : Mor_{Gen}$
$cod(m_G) = n \dot\cup_S f_Q(n_Q)$
$createRule(s, n \dot\cup_S f_Q(n_Q)) = r$
$applicable(r, m_G) = tt$
$transform(r, m_G) = n_2 \dot\cup_S n_3$
$isomorphic_{PTNET}(f_Q(n_Q), n_2) = tt$
CopyDataSchemes : ISchemes

In order to distribute data, the transition *copy data to queues and to context interpreter* will fire in order to apply the rule generated out of the interaction scheme *copyData*, which is shown in next image 6. Additionally the data will be sent to the Context Interpreter. The distribution of data to the Context Interpreter will be shown in explicit in a later step.

Figure 8.74: Traffic Service Sends $(free, 12340)$

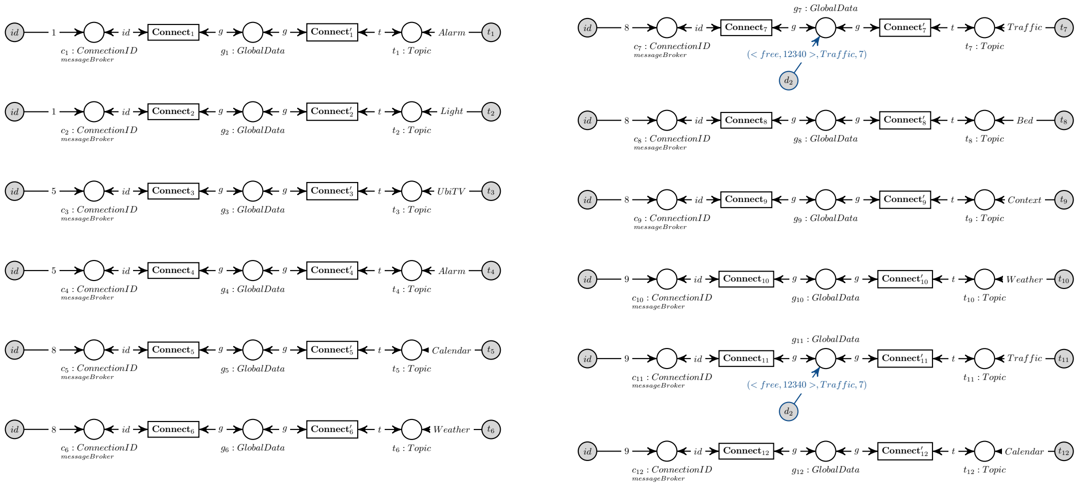
7. Concrete rule r created out of the interaction scheme *copyData* and $dataBuffer_2 \dot\cup_S q^4$:


L_{cd}
R_{cd}
d_2
$(< free, 12340 >, Traffic, 7)$
: Global Data
g
$Connect_2$
t
: Topic
Traffic
t
$\Longrightarrow$
$(< free, 12340 > Traffic, 7)$
$(< free, 12340 >, Traffic, 7)$

The interaction scheme *copyData* generates a rule r in dependence of the disjoint union of the input object nets $dataBuffer_2$ and q^4 according to the formal description given in chapter 6.
The interaction scheme *copyData* is already shown in illustration 24 in the second step of the scenario in section 8.3.

Figure 8.75: Traffic Service Sends $(free, 12340)$

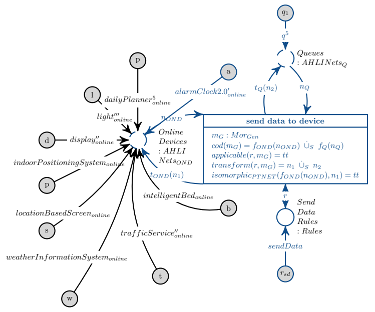
8. Object net q^5:



After executing the transition *copy data to queues and to context interpreter*, the rule r (see image 6.) transformed the object net representing the queues for all connected devices to the given net q^5.

Figure 8.76: Traffic Service Sends $(free, 12340)$

9. Extract from system net $System^{13}$ for sending data to a device:

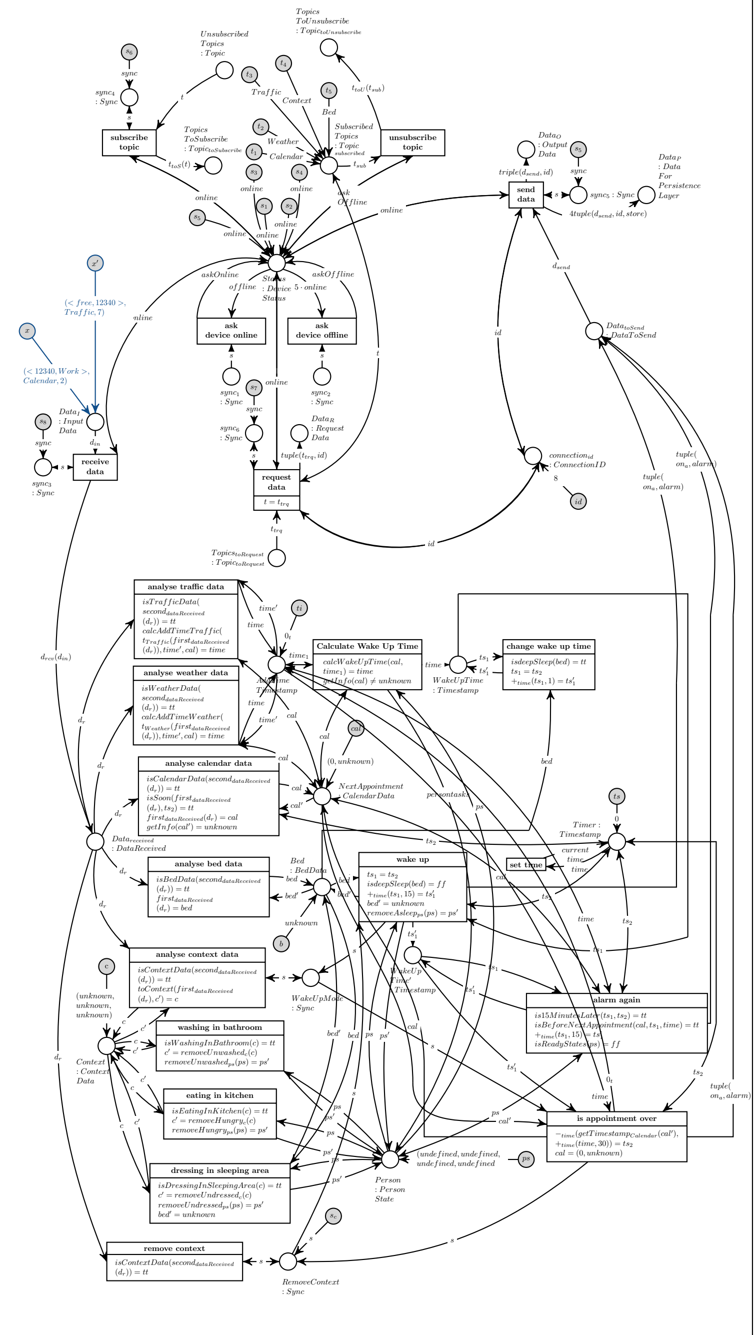


Afterwards, the new data will be sent to all devices, which are subscribed to the corresponding topic. This will be done in firing the transition *send data to device* and in applying the rule *sendData* to the disjoint coproduct of q^5 and the device net of the Alarm Clock 2.0 $alarmClock2.0'_{online}$.

This process is presented in detail in image 15 and 16 in step 3 of this scenario (see section 8.4).

(Because the system has two devices subscribed to the topic "Traffic", this transition is able to fire a second time for the device $display''_{online}$, which will be analogue and therefore will not be presented in explicit.)

10. Object net $alarmClock2.0''_{online}$:

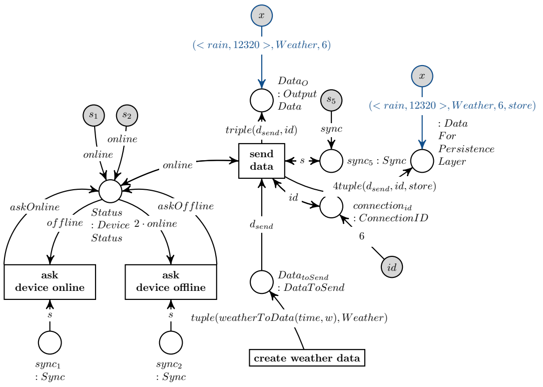


The resulting device net of the Alarm Clock 2.0 after the traffic device has sent its data to the Message Oriented Middleware and the data has been processed and distributed to all devices subscribed to the topic "Traffic" by the message broker.

Figure 8.77: Traffic Service Sends $(free, 12340)$

Figure 8.78: Traffic Service Sends $(free, 12340)$

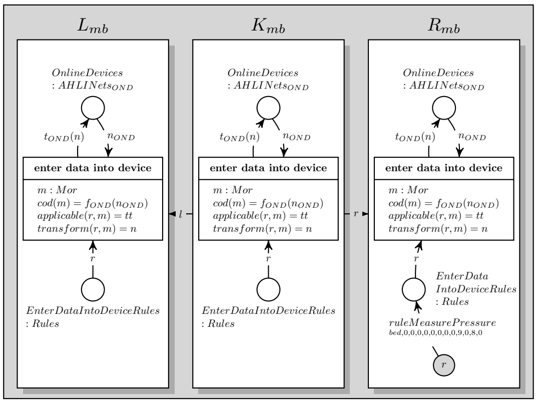
11. Device $weatherInformationSystem_{online}$ creates and processes the information $(rain, 12320)$ (see 7.23):



Similar to the device net of the Traffic Service, the Weather Information System creates a new token in firing the transition *create weather data*. The new token will be processed within this device net until it is transferred to the places $Data_O$ and $Data_P$. The resulting device net of the Weather Information System is illustrated in this image.

Figure 8.79: Weather Information System sends $(rain, 12320)$

12. Concrete AHOI rule $MeasurePressureInBed_{user,0,0,0,0,0,0,0,0,0,9,0,8,0}$ for measuring new data for the Intelligent Bed (For the general rule see 7.101.):



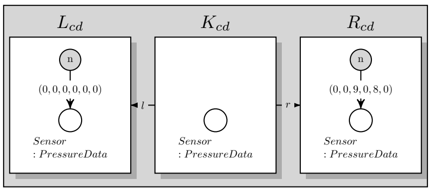
New data for the pressure sensors in the Intelligent Bed will be entered indirectly by the user when he is moving around in the Living Place apartment. For the input of data to the model of the Intelligent Bed device, the AHOI rule $MeasurePressureInBed_{user,0,0,0,0,0,0,0,0,0,9,0,8,0}$, is available, which assigns its corresponding AHLI rule $ruleMeasurePressure_{bed,0,0,0,0,0,0,0,0,0,9,0,8,0}$ to the place $EnterDataIntoDeviceRules : Rules$ within the system net in order to apply the given AHLI rule to the object net of the Intelligent Bed. Each value for x_i represents a new value for each sensor in the Intelligent Bed, where:

- $x_1 = 0$,
- $x_2 = 0$,
- $x_3 = 9$,
- $x_4 = 0$,
- $x_5 = 8$,
- $x_6 = 0$

Each v_i represents an old value representing by the 6-tuple assigned to $Sensor$ with $v_i = 0, i \in \{1, 2, 3, 4, 5, 6\}$.
The corresponding AHLI rule is presented in next image 13, the extract from the system net in image 14.

Figure 8.80: Intelligent Bed sends data

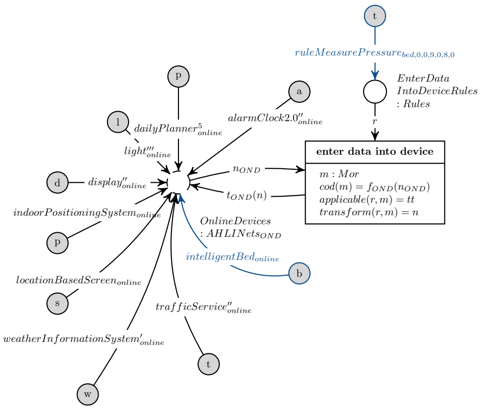
13. Concrete AHLI rule $ruleMeasurePressure_{bed,0,0,0,0,0,0,0,0,0,9,0,8,0}$ for assigning the measurement data to the Intelligent Bed device (For the general rule see 7.74.):



The concrete AHLI rule $ruleMeasurePressure_{bed,0,0,0,0,0,0,0,0,0,9,0,8,0}$ assigns the 6-tuple $(0,0,9,0,8,0)$ containing measurement values for each of the six sensors in the Intelligent Bed to the place $Sensor : PressureData$ within the device net. The device net $intelligentBed'_{online}$ will be shown in image 15., where this rule will be already executed.

Figure 8.81: Intelligent Bed sends data

14. Extract from the system net $System^{14}$ for entering new data:

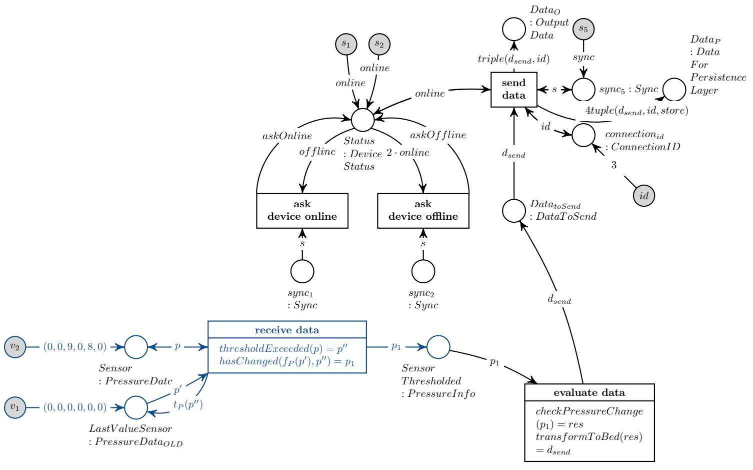


t
$ruleMeasurePressure_{bed,0,0,9,0,8,0}$
$EnterData$
$IntoDeviceRules$
$: Rules$
r
enter data into device
$m : Mor$
$cod(m) = f_{OND}(n_{OND})$
$applicable(r,m) = tt$
$transform(r,m) = n$
p
a
l
$dailyPlanner^5_{online}$
$alarmClock2.0''_{online}$
$light'''_{online}$
d
$display''_{online}$
n_{OND}
$t_{OND}(n)$
$indoorPositioningSystem_{online}$
$OnlineDevices$
$: AHLINets_{OND}$
p
$intelligentBed_{online}$
b
$locationBasedScreen_{online}$
s
$trafficService''_{online}$
$weatherInformationSystem'_{online}$
t
w

The AHLI rule $ruleMeasurePressure_{bed,0,0,9,0,8,0}$ is already assigned to the place $EnterDataIntoDeviceRules$ and is able to be applied to the device net $intelligentBed_{online}$ in firing the transition *enter data into device.*

Figure 8.82: Intelligent Bed sends data

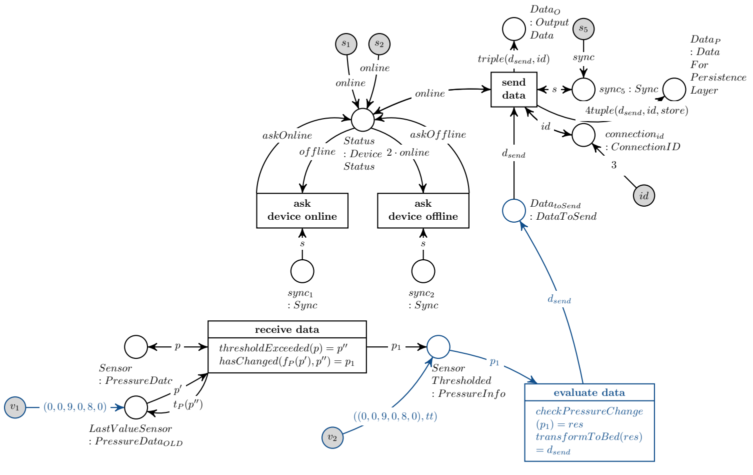
15. Device net $intelligentBed'_{online}$ (see 7.15):



The device net of the Intelligent Bed, after the AHOI and AHLI rule, mentioned in the last illustrations 12. - 14., are applied, contains the new token $(0,0,9,0,8,0)$ assigned to the place $Sensor : PressureData$.
In the next step, the transition *receive data* will be fired.

Figure 8.83: Intelligent Bed sends data

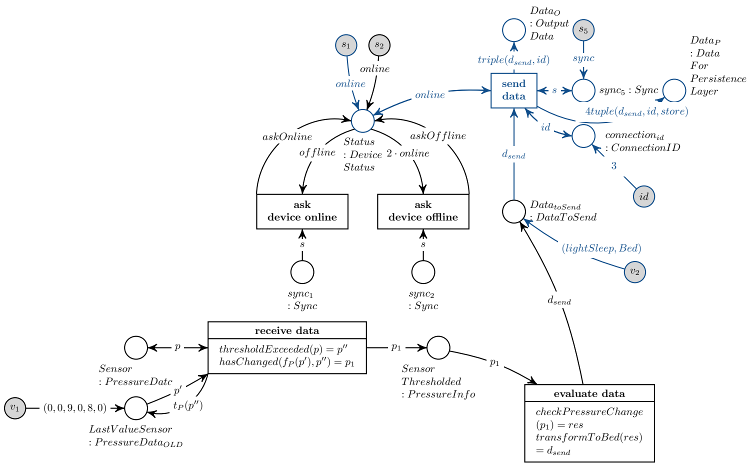
16. Device net $intelligentBed''_{online}$:



The transition *receive data* executes the operations *thresholdExceeded* which returns $p'' = (0, 0, 9, 0, 8, 0)$. The second operation compares the last measured value assigned to $LastValueSensor : PressureData_{OLD}$ (initially it is set to 0 for each x_i) and it finds a difference between the last value token and the new data and therefore returns $((0, 0, 9, 0, 8, 0), tt) = p_1$. The value of p_1 will be forwarded to the place $SensorThresholded : PressureInfo$. Additionally the last value will be exchanged by the new one for the processing of the next incoming pressure data.

Figure 8.84: Intelligent Bed sends data

17. Device net $intelligentBed'''_{online}$:



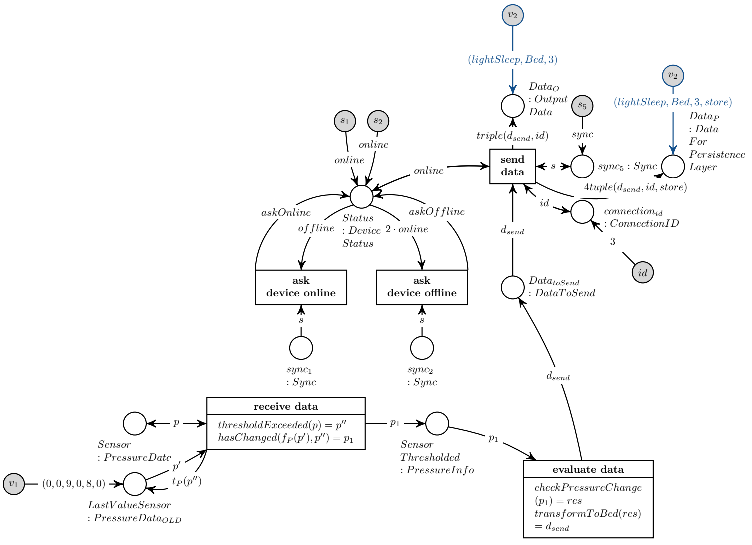
The transition *evaluate data* uses the operation *checkPressureChange* first, which checks the values received from the place *SensorThresholded* and infers one of the following states of the inhabitant of the Living Place apartment:

- *lightSleep*
- *deepSleep*
- *awake*
- *notInBed*

An explicit definition of the operation is given in the Σ_{OL}-algebra A_{OL} of the Object Level in section 11.1.
In the current step, the phase *lightSleep* was infered and the resulting token is set to the place $Data_{toSend}$.

Figure 8.85: Intelligent Bed sends data

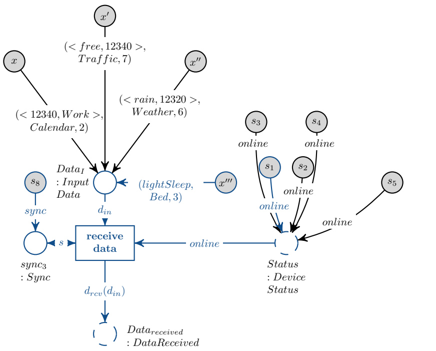
18. Device net $intelligentBed^4_{online}$:


v_2
$(lightSleep, Bed, 3)$
$Data_O$ $: Output$ $Data$
s_5
v_2
$(lightSleep, Bed, 3, store)$
$Data_P$ $: Data$ For $Persistence$ $Layer$
s_1
s_2
$online$
$online$
$triple(d_{send}, id)$
$sync$
$online$
send data
s
$sync_5 : Sync$
$4tuple(d_{send}, id, store)$
$askOnline$
$offline$
$Status$ $: Device$ $Status$
$2 \cdot online$
$askOffline$
id
$connection_{id}$ $: ConnectionID$
d_{send}
3
id
ask device online
ask device offline
$Data_{toSend}$ $: DataToSend$
s
s
$sync_1$ $: Sync$
$sync_2$ $: Sync$
d_{send}
receive data
$thresholdExceeded(p) = p''$
$hasChanged(f_P(p'), p'') = p_1$
p
p_1
p_1
$Sensor$ $: PressureDatc$
$Sensor$ $Thresholded$ $: PressureInfo$
p'
$t_P(p'')$
v_1
$(0, 0, 9, 0, 8, 0)$
$LastValueSensor$ $: PressureData_{OLD}$
evaluate data
$checkPressureChange$ $(p_1) = res$
$transformToBed(res)$ $= d_{send}$

The results from the internal processing steps of the Intelligent Bed device are extended in order to send them to the Message Oriented Middleware. There, the data will be processed in executing the steps: receive data from the device net, copy the data to the queues of devices subscribed to the topic "Bed" and transfer the data to the context interpreter. The only device subscribed to this topic is the Alarm Clock 2.0. As last step, the information will be given to the Alarm Clock 2.0 device in assigning the token to the place $Data_I : InputData$ within its object net.

Figure 8.86: Intelligent Bed sends data

19. Extract from device net $alarmClock2.0'''_{online}$ showing only the part for receiving data:

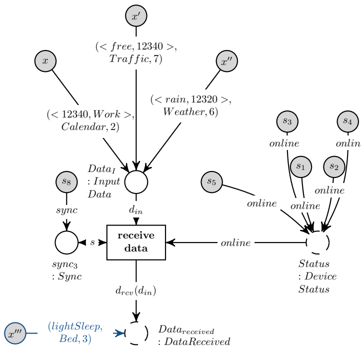

x'
$(< free, 12340 >,$ $Traffic, 7)$
x
x''
$(< rain, 12320 >,$ $Weather, 6)$
$(< 12340, Work >,$ $Calendar, 2)$
s_3
s_4
$online$
$online$
$Data_I$ $: Input$ $Data$
s_8
$(lightSleep,$ $Bed, 3)$
x'''
s_1
s_2
$online$
$online$
s_5
$sync$
d_{in}
$online$
s
receive data
$online$
$sync_3$ $: Sync$
$Status$ $: Device$ $Status$
$d_{rcv}(d_{in})$
$Data_{received}$ $: DataReceived$

This image describes an extract from the object net $alarmClock2.0'''_{online}$ after receiving data regarding the topics $Calendar$, $Traffic$ and Bed.

In the next step, the transition *receive data* will fire in processing the token $(lightSleep, Bed, 3)$ from its predomain.

Figure 8.87: Intelligent Bed sends data

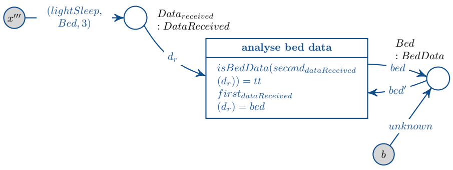
20. Extract from device net $alarmClock2.0^4_{online}$ showing only the part for receiving data:


x'
$(< free, 12340 >,$ $Traffic, 7)$
x
x''
$(< rain, 12320 >,$ $Weather, 6)$
$(< 12340, Work >,$ $Calendar, 2)$
s_3
s_4
$online$
$online$
$Data_I$ $: Input$ $Data$
s_8
s_5
s_1
s_2
$online$
$online$
$online$
$sync$
d_{in}
s
receive data
$online$
$sync_3$ $: Sync$
$Status$ $: Device$ $Status$
$d_{rcv}(d_{in})$
x'''
$(lightSleep,$ $Bed, 3)$
$Data_{received}$ $: DataReceived$

The token $(lightSleep, Bed, 3)$ is transferred to the place $Data_{received}$.

Figure 8.88: Intelligent Bed sends data

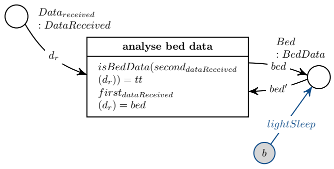
21. Extract from the device net $alarmClock2.0^4_{online}$ showing part for processing data belonging to the topic “Bed”:



The transition *analyse bed data* is now able to fire. For that, it checks, if the token taken from the predomain belongs to the topic “Bed” in using the operation $isBedData$. If this check was successful, the first component of the input triple from $Data_{received}$ will be taken and exchanged with the token assigned to $Bed : BedData$. Initially the token assigned to Bed is set to $unknown$.

Figure 8.89: Intelligent Bed sends data

22. Extract from the device net $alarmClock2.0^5_{online}$ showing part for processing data belonging to the topic “Bed”:



The token assigned to $Bed : BedData$ is replaced by the newly received token $lightSleep$.

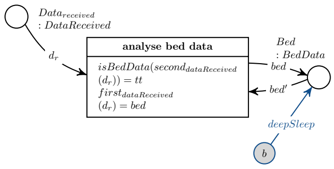
Figure 8.90: Intelligent Bed sends data

23. Summary of the following steps:

The whole night, the Intelligent Bed sends a huge amount of data of recorded pressure data created by the user sleeping in bed. To shorten the scenario up, the Intelligent Bed should send the same data again, that means, the steps 12. - 22. will be repeated with the same preassure values $(0, 0, 9, 0, 8, 0)$. The difference between the current process and the last one is, that the last values received by the device net of the Intelligent Bed and the current ones are identical. So, the resulting phase $deepSleep$ will be inferred by the operation $checkPressureChange$ which will be executed when firing *evaluate data* within the device net of the Intelligent Bed. Finally the triple $(deepSleep, Bed, 3)$ will be transferred to and processed within the device net of the Alarm Clock 2.0.

Figure 8.91: Intelligent Bed sends data

24. Extract from the device net $alarmClock2.0^6_{online}$ showing part for processing data belonging to the topic “Bed”:

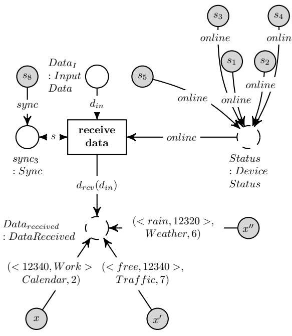


After executing the steps mentioned in 23., the token $deepSleep$ is assigned to $Bed : BedData$.

Figure 8.92: Intelligent Bed sends data

Now, the Alarm Clock 2.0 will fire the transition *set time* sending the current timestamp 12100 in order to start the inference process of an appropriate wake up time. It is assumed, that the correct current time will be distributed by the operation $currenttime$. The general problem of time in AHLI nets resp. AHOI nets is discussed in chapter 10 and will be neglected in this chapter.

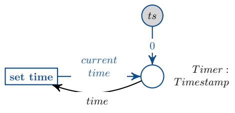
25. Extract from $alarmClock2.0^7_{online}$ after transferring both tokens from $Data_I$ to $Data_{received}$:



After firing the transition *receive data* twice, both tokens are transferred to $Data_{received}$ in order to get processed.

Figure 8.93: Calculate Wake Up Time

26. Extract from $alarmClock2.0^7_{online}$ showing part for generating the current time:

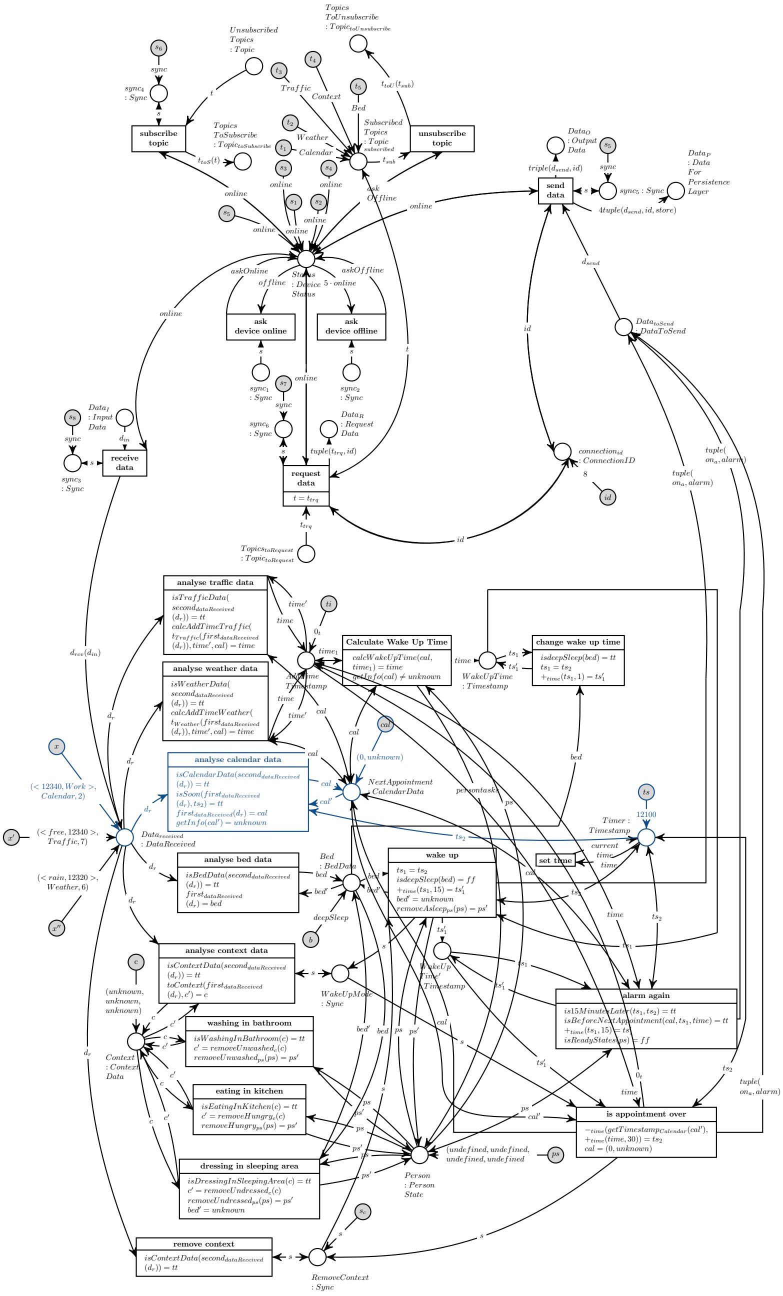


The transition *set time* should now fire in order to replace the token assigned to *Timer : Timestamp* by the current time in executing the operation *currenttime*.

The extract from the device net of the Alarm Clock 2.0 represents the internal time component "TimeModule" of the Alarm Clock 2.0 (for further details see 2.2.2.1). Initially the token assigned to *Timer* is set to 0.

Figure 8.94: Calculate Wake Up Time

27. $alarmClock2.0^8_{online}$ after firing *settime*:

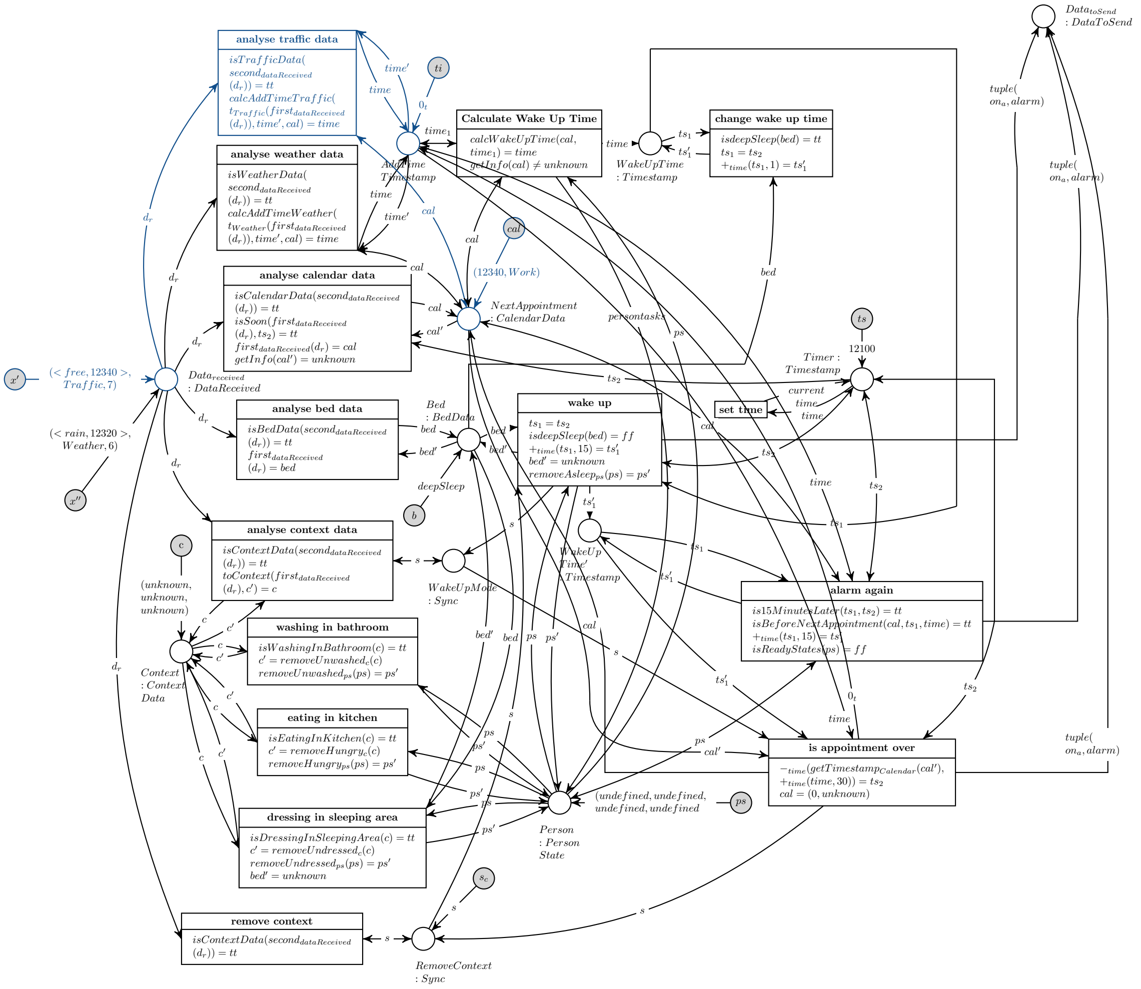


After firing the transition *set time*, the current timestamp 12100 was generated in executing the operation *currenttime*. As next step, the transition *analyse calendar data* will fire in using the token $(< 12340, Work >, Calendar, 2)$ from its predomain, because all conditions of this transition are fulfilled resp. the operations are able to be executed correctly.

For a detailed definition of the operations see the algebra for the Object Level in section 11.1.

Figure 8.95: Calculate Wake Up Time

28. Extract from $alarmClock2.0^{9}_{online}$ showing individual processing part of the device:


analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts'_1$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts'_1$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts'_1$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{received}$: DataReceived
AddTime Timestamp
NextAppointment : CalendarData
Bed : BedData
WakeUpMode : Sync
Context : Context Data
Person : Person State
RemoveContext : Sync
WakeUpTime : Timestamp
WakeUp Time' : Timestamp
Timer : Timestamp
$Data_{toSend}$: DataToSend
x'
$(< free, 12340 >, Traffic, 7)$
x''
$(< rain, 12320 >, Weather, 6)$
ti
0_t
cal
$(12340, Work)$
ts
12100
b
deepSleep
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undefined, undefined
s_c
d_r
time
time'
$time_1$
cal
cal'
bed
bed'
c
c'
s
ps
ps'
ts_1
ts'_1
ts_2
persontasks
current time
$tuple(on_a, alarm)$

After firing the transition *analyse calendar data*, a the token assigned to $NextAppointment$: $CalendarData$ is replaced by the new token $12340, Work$, which represents the next appointment soon to come.

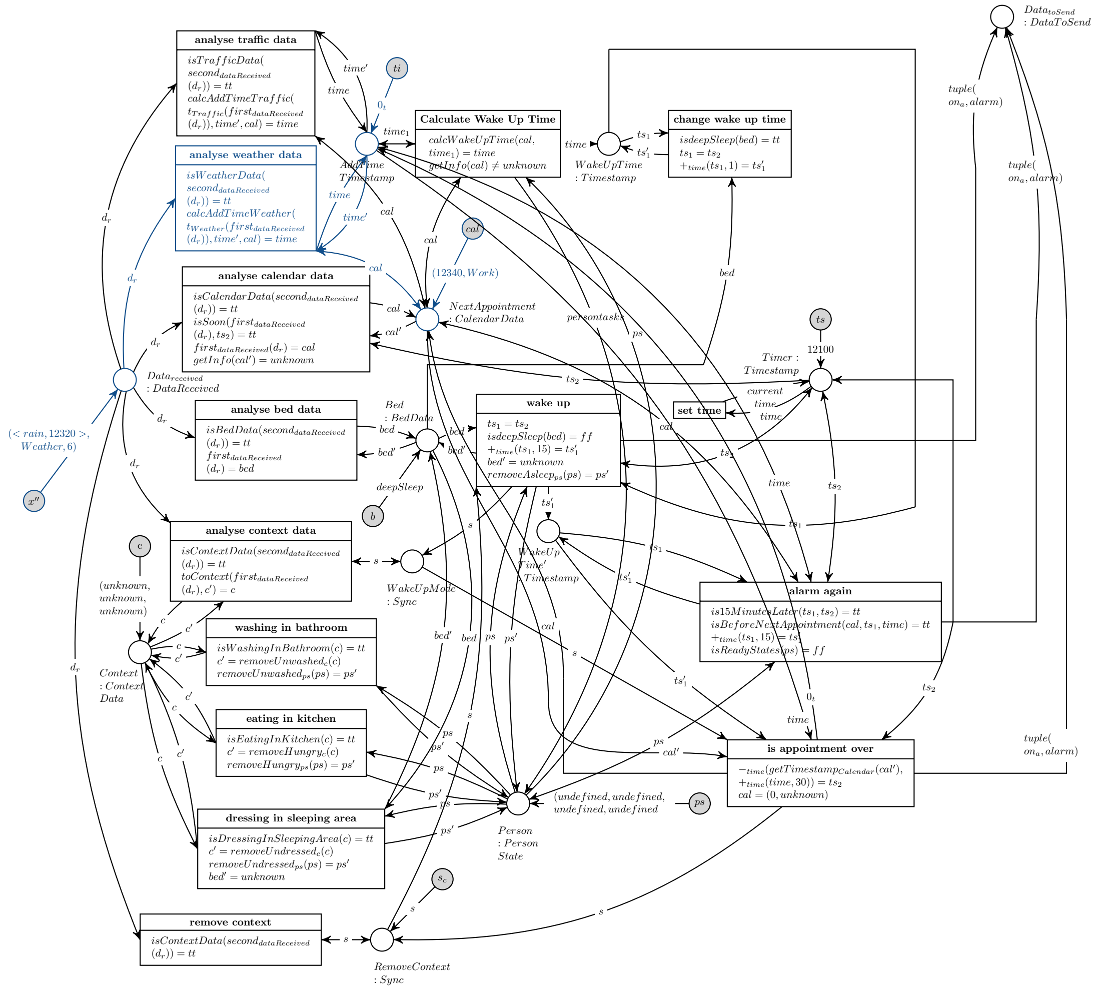
The transition *analyse traffic data* is able to fire in processing the token $(< free, 12340 >, Traffic, 7)$ from the place $Data_{received}$ and possibly adding additional time to the token assigned to $AddTime$: $Timestamp$, if a bad traffic condition is described. This will be infered in using the conditions of this transition:

- $isTrafficData(second_{dataReceived}(d_r)) = tt$: Returns true, because the second component of the token $(< free, 12340 >, Traffic, 7)$ belongs to the topic "Traffic".
- $calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$: Returns no additional time, because even if the event occurs on the way to the appointment (this is the interval between $appointment - 30minutes$ and the timestamp of the appointment), the parameter $free$ signals good traffic conditions.

As a result, the traffic conditions are good and no extra time will be considered.

Figure 8.96: Calculate Wake Up Time

29. Extract from $alarmClock2.0^{10}_{online}$ showing individual processing part of the device:


analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts_1'$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts_1'$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts_1'$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{toSend}$: DataToSend
$Data_{received}$: DataReceived
AddTime : Timestamp
NextAppointment : CalendarData
WakeUpTime : Timestamp
Timer : Timestamp
Bed : BedData
WakeUpMode : Sync
WakeUp Time' : Timestamp
Context : Context Data
Person : Person State
RemoveContext : Sync
ti
0_t
cal
(12340, Work)
ts
12100
$(<rain, 12320>, Weather, 6)$
x''
b
deepSleep
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undefined, undefined)
s_c
$tuple(on_a, alarm)$
d_r
time
time'
$time_1$
ts_1
ts_1'
ts_2
current time
persontasks
bed
bed'
cal'
c
c'
ps'
s

The token assigned to $AddTime : Timestamp$ still contains no additional time after the processing step shown in image 28.

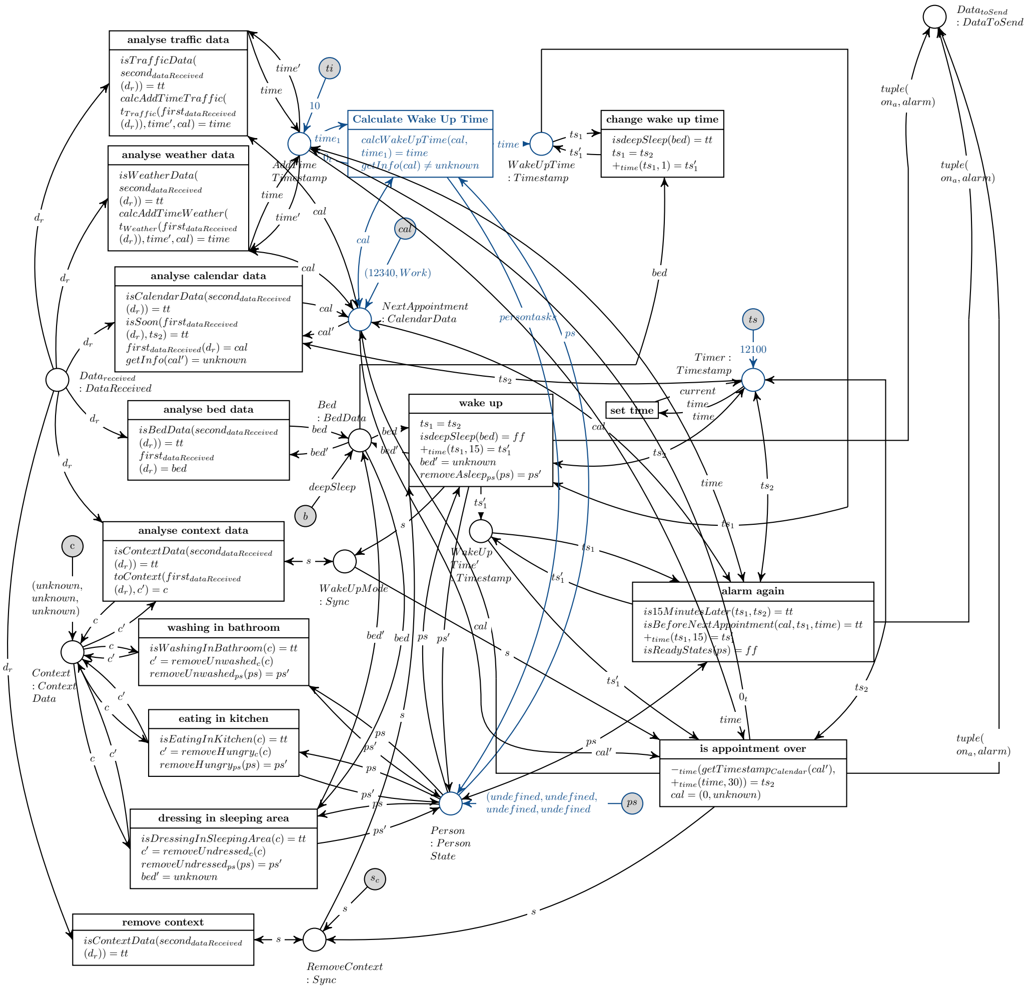
The transition *analyse weather data* is analogue to the transition *analyse traffic data* but processes the last token $(< rain, 12320 >, Weather, 6)$ assigned to the place $Data_{received}$. The conditions will be evaluated as follows:

- $isWeatherData(second_{dataReceived}(d_r)) = tt$: Returns true, because the second component of the token $(< rain, 12320 >, Weather, 6)$ is the topic "Weather".
- $calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$: Retuns an additionaly time of 10 minutes which will be added to the token assigned to $AddTime$, because the event occurs on the way to the appointment and the description $rain$ causes an additional time of 10 minutes.

A detailed definition of the operations are given within the Σ_{OL}-algebra A_{OL} of the Object Level in section 11.1.

Figure 8.97: Calculate Wake Up Time

30. Extract from $alarmClock2.0^{11}_{online}$ showing individual processing part of the device:


analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time', cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time', cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal') = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c') = c
washing in bathroom
isWashingInBathroom(c) = tt
c' = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps'
eating in kitchen
isEatingInKitchen(c) = tt
c' = removeHungry_c(c)
removeHungry_ps(ps) = ps'
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c' = removeUndressed_c(c)
removeUndressed_ps(ps) = ps'
bed' = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts'_1
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts'_1
bed' = unknown
removeAsleep_ps(ps) = ps'
set time
alarm again
is15MinuteLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts'_1
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal'), +_time(time, 30)) = ts_2
cal = (0, unknown)
Data_toSend : DataToSend
Data_received : DataReceived
AddTime Timestamp
WakeUpTime : Timestamp
NextAppointment : CalendarData
Bed : BedData
Timer : Timestamp
WakeUp Time' Timestamp
WakeUpMode : Sync
Context : Context Data
Person : Person State
RemoveContext : Sync
ti
10
time_1
cal
(12340, Work)
ts
12100
b
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undefined, undefined)
s_c
deepSleep
tuple(on_a, alarm)
time'
time
d_r
bed
bed'
ps'
s
ts_1
ts'_1
ts_2
current time
0_t
personstasks

The transition *Calculate Wake Up Time* is able to fire and in doing so, it calculates an appropriate wake up time, but only if the token assigned to $NextAppointment : CalendarData$ is not equal to *unknown*. This condition will be fulfilled and the wake up time will be calculated as follows:

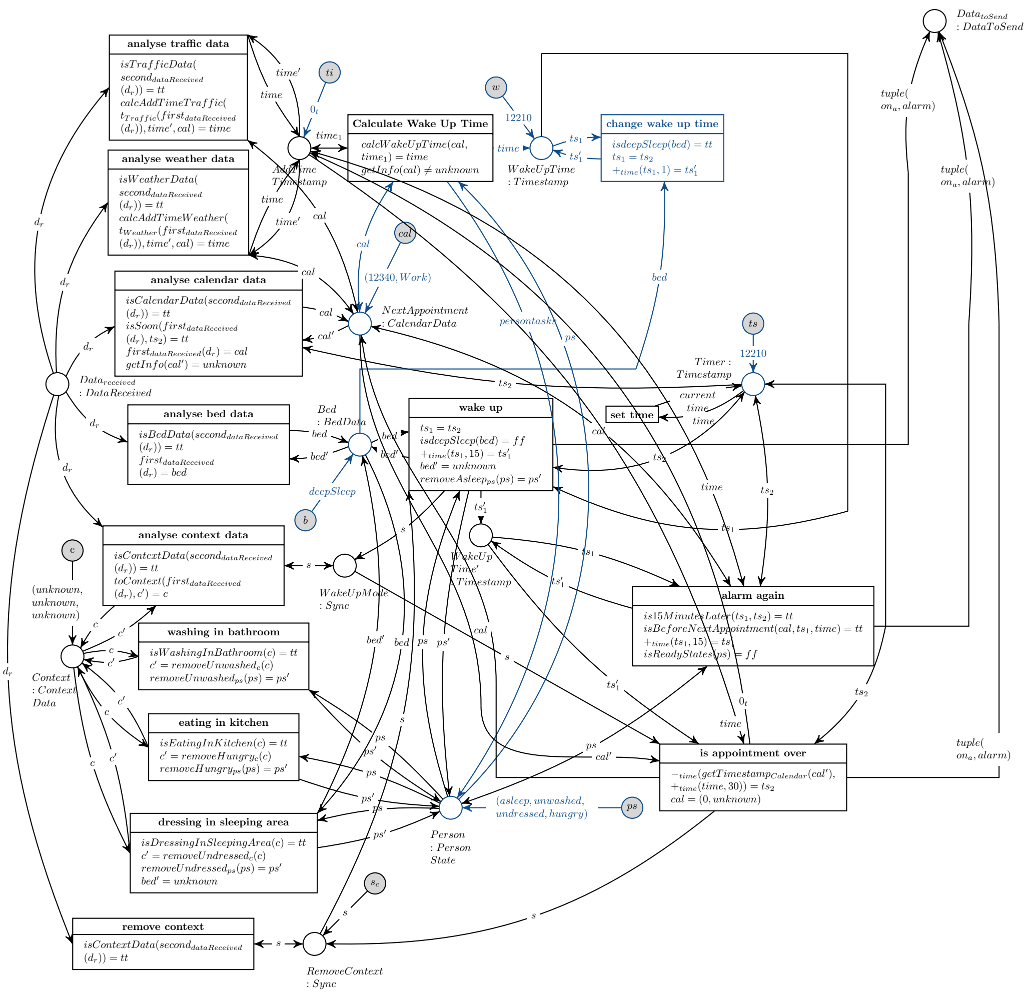
$calculateWakeUpTime(cal, time_1) = time = cal - 120 - time_1 = 12340 - 120 - 10 = 12210$, where

- $cal = 12340$ is the timestamp of the appointment,
- $time_1 = 10$ is the additionaly time and
- 120 is an assumed time for the user to get to the appointment: 60 minutes for preparation (getting up, washing, dressing, eating, see next step in the scenario 8.6), 30 minutes for the way to work and 30 minutes as buffer, if the Alarm Clock 2.0 have to wait until the person finishes the deep sleep phase for an optimal waking up.

The new wake up time will be put to the place $WakeUpTime : Timestamp$. In addition the token assigned to $Person : PersonState$ will be exchanged by the 4-tuple *(asleep, unwashed, undressed, hungry)*, which will be needed in the next step of the scenario in 8.6.
Afterwards the internal timer generates a new timestamp 12210 in firing *set time*. Normally, the timer should generate the current timestamp every minute. To shorten the scenario up, it sets the time needed (analogue to image 26 or 31).

Figure 8.98: Calculate Wake Up Time

31. Extract from $alarmClock2.0^{12}_{online}$ showing individual processing part of the device:


analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts'_1$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts'_1$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{received}$: DataReceived
AddTime : Timestamp
WakeUpTime : Timestamp
NextAppointment : CalendarData
Bed : BedData
WakeUpMode : Sync
WakeUp Time' : Timestamp
Timer : Timestamp
Context : Context Data
Person : Person State
RemoveContext : Sync
$Data_{toSend}$: DataToSend
ti
0_t
w
12210
cal
(12340, Work)
ts
12210
b
deepSleep
c
(unknown, unknown, unknown)
ps
(asleep, unwashed, undressed, hungry)
s_c
d_r
time
time'
$time_1$
ts_1
ts'_1
ts_2
cal'
bed
bed'
personstasks
ps
ps'
c
c'
s
current time
$tuple(on_a, alarm)$

The token assigned to $WakeUpTime : Timestamp$ represents the new wake up time. As next step, the transition *change wake up time* will fire because the token assinged to $Bed : BedData$ is $deepSleep$ signalling that the user is in a deep sleep phase. Consequently, the wake up time will be increase by one minute in order to check next minute again.

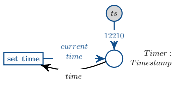
Figure 8.99: Calculate Wake Up Time

32. The Intelligent Bed infers *lightSleep*:

Similar to the steps 12.-22. and 23., the Intelligent Bed measures a new pressure data tuple which differs from the old one but exceeds the threshold and therefore infers the state *lightSleep*. As a result, the token assigned to the place *Bed* : *BedData* within the device net $alarmClock2.0^{12}_{online}$ will be replaced by a token *lightSleep*.

Figure 8.100: Attach a Device

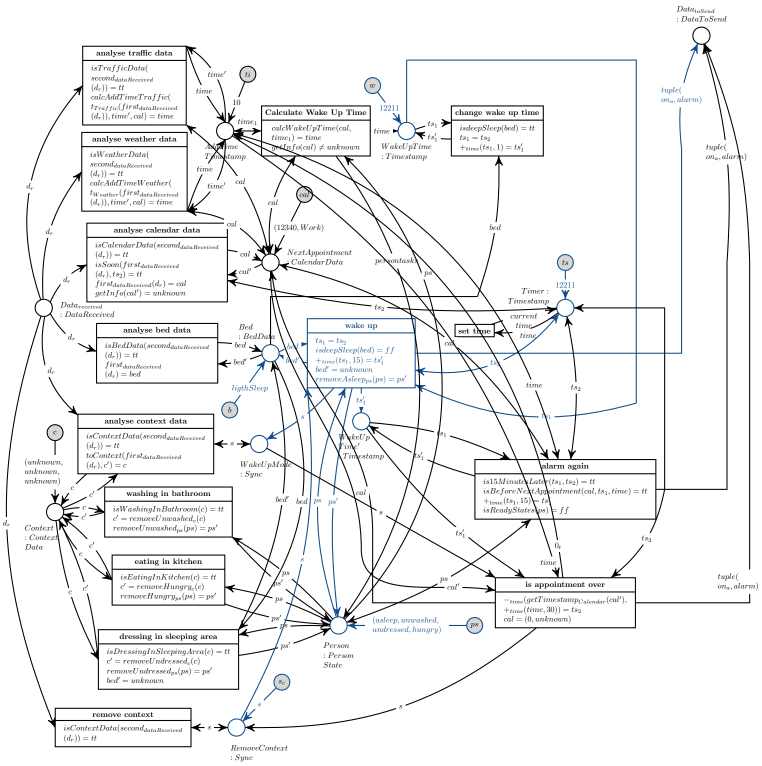
33. The internal timer generates the next timestamp 12211, extract from $alarmClock2.0^{13}$:



The internal timer generates the next timestamp 12211, in firing *set time*.

Figure 8.101: Attach a Device

34. Extract from $alarmClock2.0^{13}_{online}$ showing individual processing part of the device:

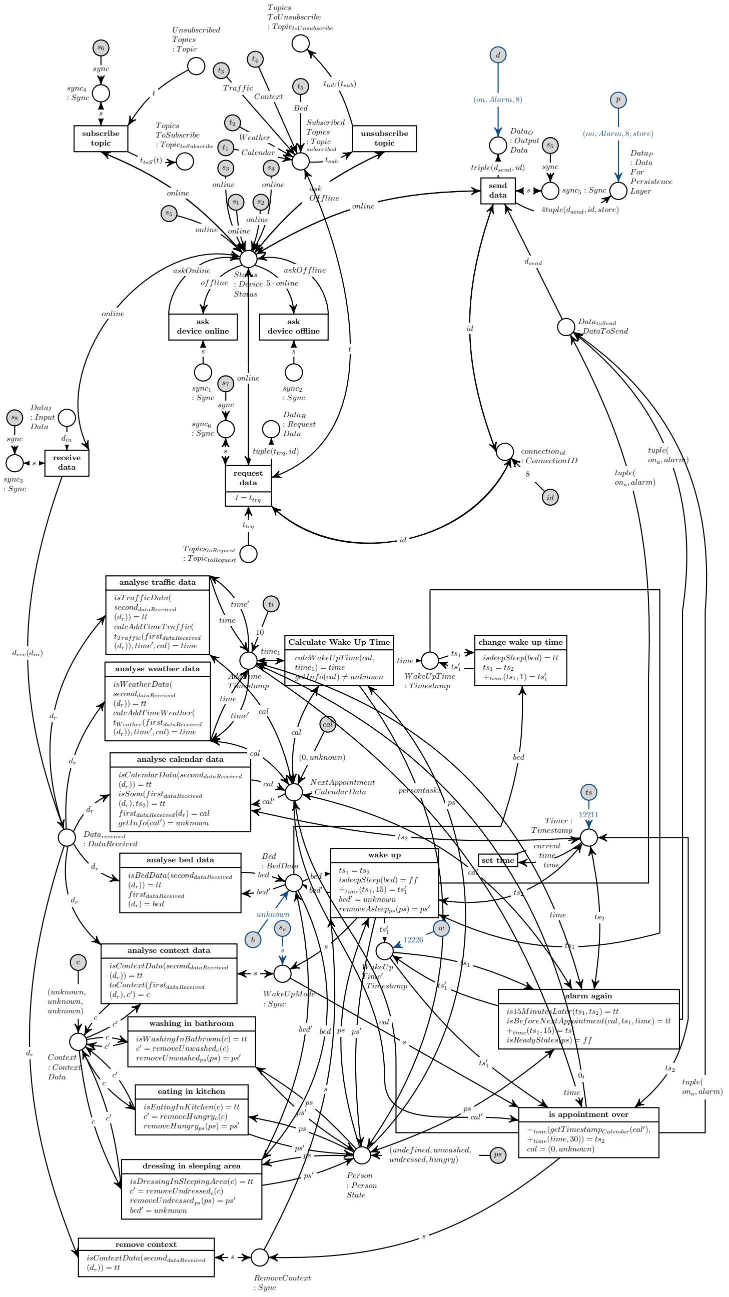

Data_toSend
: DataToSend
analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time′, cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time′, cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal′) = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c′) = c
washing in bathroom
isWashingInBathroom(c) = tt
c′ = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps′
eating in kitchen
isEatingInKitchen(c) = tt
c′ = removeHungry_c(c)
removeHungry_ps(ps) = ps′
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c′ = removeUndressed_c(c)
removeUndressed_ps(ps) = ps′
bed′ = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts′_1
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts′_1
bed′ = unknown
removeAsleep_ps(ps) = ps′
set time
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts_1
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal′), +_time(time, 30)) = ts_2
cal = (0, unknown)
ti
10
AddTime
Timestamp
w
12211
WakeUpTime
: Timestamp
cal
(12340, Work)
NextAppointment
: CalendarData
Data_received
: DataReceived
Bed
: BedData
ligthSleep
b
c
(unknown, unknown, unknown)
Context
: Context
Data
WakeUpMode
: Sync
WakeUp
Time′
: Timestamp
ts
12211
Timer :
Timestamp
current
time
time
Person
: Person
State
(asleep, unwashed, undressed, hungry)
ps
s_c
RemoveContext
: Sync
tuple(on_a, alarm)
persontasks
0_t

Because of the token *lightSleep* assigned to the plave *Bed* : *BedData* and the wake up time is equal to the current timestamp, the transition *wake up* is able to fire. It modifies the current wake up timestamp in adding additional 15 minutes, which will be necessary to remember the user of the tasks, he should fulfil before leaving the Living Place apartment. All open tasks are stored to the place *Person* : *PersonState*. The first component *asleep* will be removed from the task list when the person is woken up by the Alarm Clock 2.0 device. Additionally the synchronize token assigned to *RemoveContext* : *Sync* will be removed, because from now on, no received context data should get removed without being processed. Instead, a new token will be assigned to the place *WakeUpMode* : *Sync* enabling the device to process all new incoming context data. The token *ligthSleep* set to *Bed*.

In order to wake the person up, a token $tuple(on_a, alarm)$ will be created which will be put to $Data_{toSend}$ and will be further processed and transformed within the Alarm Clock 2.0 device net until a token containing this information is assigned to $Data_O$ and another token to $Data_P$.

Figure 8.102: Calculate Wake Up Time

35. Object net $alarmClock2.0^{14}_{online}$:

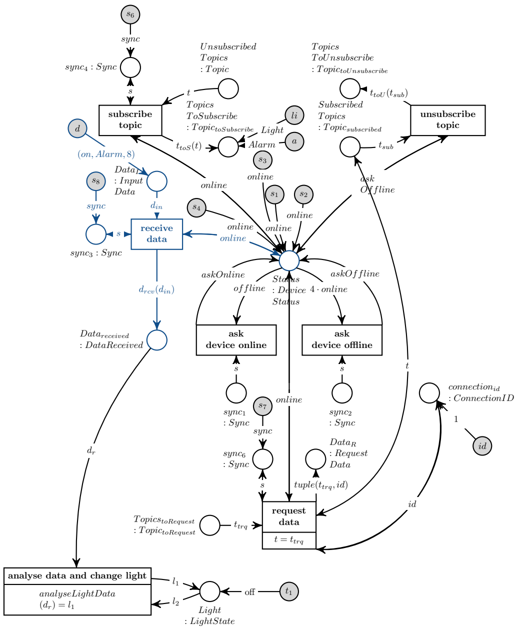

subscribe topic
unsubscribe topic
Unsubscribed Topics : Topic
Topics ToUnsubscribe : Topic_toUnsubscribe
Topics ToSubscribe : Topic_toSubscribe
Subscribed Topics : Topic_subscribed
Traffic
Context
Bed
Weather
Calendar
t_toU(t_sub)
t_toS(t)
t_sub
sync4 : Sync
sync
online
askOnline
askOffline
offline
5 · online
Status : Device Status
ask device online
ask device offline
sync1 : Sync
sync2 : Sync
sync6 : Sync
Data_I : Input Data
d_in
receive data
sync3 : Sync
Data_R : Request Data
tuple(t_trq, id)
request data
t = t_trq
t_trq
Topics_toRequest : Topic_toRequest
(on, Alarm, 8)
(on, Alarm, 8, store)
Data_O : Output Data
triple(d_send, id)
send data
sync5 : Sync
Data_P : Data For Persistence Layer
4tuple(d_send, id, store)
d_send
Data_toSend : DataToSend
id
connection_id : ConnectionID
8
tuple(on_a, alarm)
analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time′, cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time′, cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal′) = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c′) = c
washing in bathroom
isWashingInBathroom(c) = tt
c′ = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps′
eating in kitchen
isEatingInKitchen(c) = tt
c′ = removeHungry_c(c)
removeHungry_ps(ps) = ps′
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c′ = removeUndressed_c(c)
removeUndressed_ps(ps) = ps′
bed′ = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
RemoveContext : Sync
Data_received : DataReceived
d_rcv(d_in)
d_r
Context : Context Data
(unknown, unknown, unknown)
AddTime Timestamp
10
time
time′
time_1
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
WakeUpTime : Timestamp
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts′_1
NextAppointment : CalendarData
(0, unknown)
cal
cal′
Bed : BedData
bed
bed′
unknown
WakeUpMode : Sync
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts′_1
bed′ = unknown
removeAsleep_ps(ps) = ps′
WakeUp Time′ : Timestamp
12226
Timer : Timestamp
12211
current time
set time
personstasks
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts_1
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal′), +_time(time, 30)) = ts_2
cal = (0, unknown)
0_t
Person : Person State
(undefined, unwashed, undressed, hungry)
ps
ps′

The illustration shows the device net $alarmClock2.0^{14}_{online}$ providing the signal $(on, Alarm, 8)$ resp. $(on, Alarm, 8, store)$ to the Message Oriented Middleware.

The Message Oriented Middleware processes the data and distributes it to the device nets $locationBasedScreen_{online}$ and $light'''_{online}$ which are all devices subscribed to the topic “Alarm”. The processing of data on System Level is always analogue to the procedure presented in detail for the device net of the Traffic Service in 3.-10. in this section, and consequently will not be shown in explicit anymore.

Figure 8.103: Calculate Wake Up Time

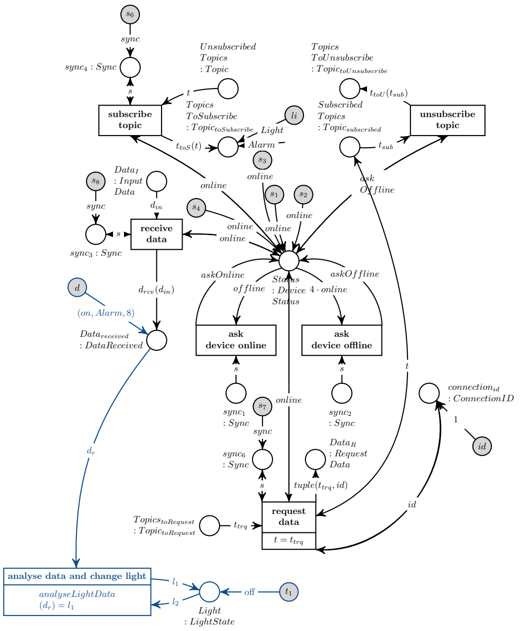
36. Ambient Light receives the token $(on, Alarm, 8)$ (see 7.21):



The token $(on, Alarm, 8)$ is assigned to the place $Data_I$: $InputData$ within the device net of the Ambient Light after finishing the processing of thie data within the Message Oriented Middleware. Now, the data will be collected in firing the transition *receive data*.

Figure 8.104: Signal Waking Up, With Light

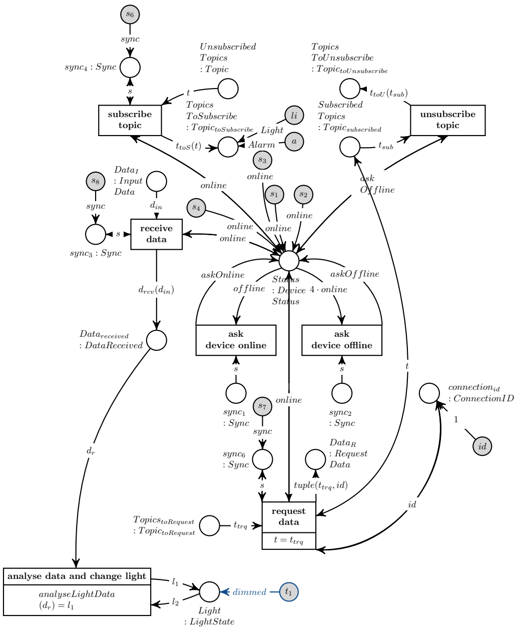
37. Ambient Light processes the token $(on, Alarm, 8)$:



The transition *analyse data and change light* is now able to fire. Its operation $analyseLightData$ checks the incoming data and if it contains on as first component and $Alarm$ as second component, the token assigned to $Light$: $LightState$ will be replaced by a token *dimmed*. If this condition is not met, that means, the topic of this message is “Light”, the value assigned to $Light$: $LightState$ will be replaced by a new state which is represented by the first component of this message.
The operation will be defined within the algebra of the Object Level in section 11.1.

Figure 8.105: Signal Waking Up, With Light

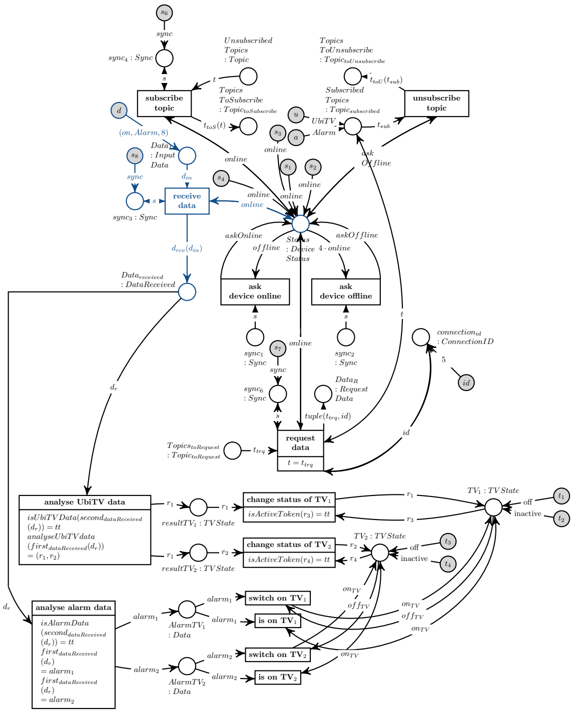
38. Ambient Light processes the token $(on, Alarm, 8)$ (see 7.21):



After firing the transition *analyse data and change light*, the resulting value *dimmed* will be assigned to *Light* : *LightState*, at the same time *off* is removed.

Figure 8.106: Signal Waking Up, With Light

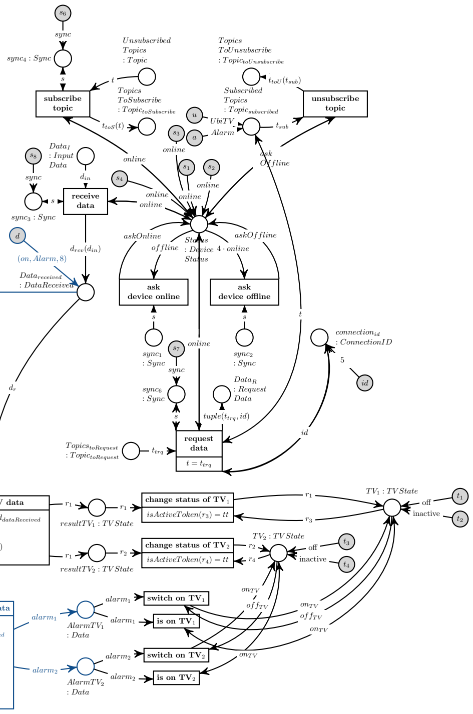
39. Location-Based Screen receives the token $(on, Alarm, 8)$ (see 7.19):



After a token is assigned to the place $Data_I$, the processing is able to start with firing the transition *receive data*. Then, the token will be forwarded to the place $Data_{received}$.

Figure 8.107: Signal Waking Up, With TVs

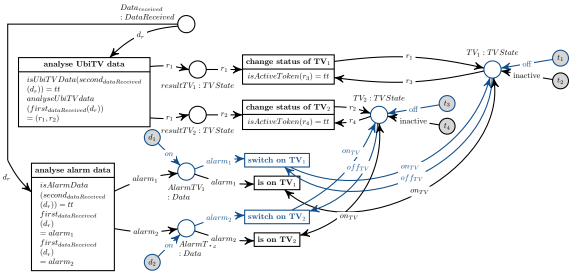
40. Location-Based Screen processes the token $(on, Alarm, 8)$ (see 7.19):



The transition *analyse alarm data* checks, if the received information belongs to the topic "Alarm". Afterwards it takes the first component of the triple, which is *on*, and assigns this value as token to the places $AlarmTV_1 : Data$ resp $AlarmTV_2 : Data$.
If a triple is received which belongs to the topic "Alarm", it always has *on* as first component of the triple. Currently, an information regarding the topic "Alarm" is only created by the Alarm Clock 2.0 device in the model presented in this work. It only sends a data token $(on, Alarm, 8)$.

Figure 8.108: Signal Waking Up, With TVs

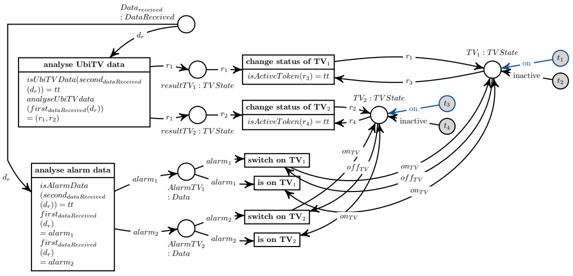
41. Extract from net $locationBasedScreen''_{online}$ showing device specific part:



The TV sets are initially in offline mode, so the transitions *switch on* TV_i are activated $(i \in \{1, 2\})$ in order to change the mode to online.

Figure 8.109: Signal Waking Up, With TVs

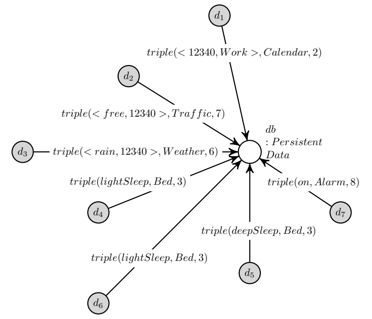
42. Extract from net $locationBasedScreen'''_{online}$ showing device specific part:



Finally both TV sets are switched on which is represented by the token *on* assigned to the places $TV_1 : TVState$ resp. $TV_2 : TVState$.
But none of them will show any image. For that, position data from the Indoor-Positioning-System are necessary in order to show the TV program on one of both TV screens. In order to wake up the inhabitant with the help of the TV set in the sleeping area, the Indoor-Positioning-System will send the position information of the user who stays in bed. Afterwards the Location-Based Screen evaluates the information belonging to the topic "UbiTV" similar to the processing of data belonging to the topic "Alarm" and changes the state of the TV set in the sleeping area to *active*. This will not be presented in explicit, here. The switching on of a certain screen will be shown in the 6th part of the scenario in section 8.7.

Figure 8.110: Signal Waking Up, With TVs

43. Current state of the object net *database″* containing all persistent data:


d_1
$triple(< 12340, Work >, Calendar, 2)$
d_2
$triple(< free, 12340 >, Traffic, 7)$
d_3
$triple(< rain, 12340 >, Weather, 6)$
db
$: PersistentData$
$triple(lightSleep, Bed, 3)$
$triple(on, Alarm, 8)$
d_4
d_7
$triple(deepSleep, Bed, 3)$
$triple(lightSleep, Bed, 3)$
d_5
d_6

At the end of part 4 of the whole scenario, the presented pieces of information were processed by the Message Oriented Middleware and therefore stored within the object net assigned to the place $PersistentData : AHLINets$. This place is situated in the system net describing the Message Oriented Middleware. The token $database''$ assigned to $PersistentData$ represents the persistent database of the Living Place system.

Figure 8.111: Signal Waking Up, With TVs

8.6 Preparation Phase For an Appointment: User is in Bathroom

After the inhabitant has been woken up by the Living Place apartment, he has about one hour for preparing for the appointment, during that time, the user has to fulfil the following tasks:

- washing in bathroom,
- dressing in sleeping area and
- eating in kitchen.

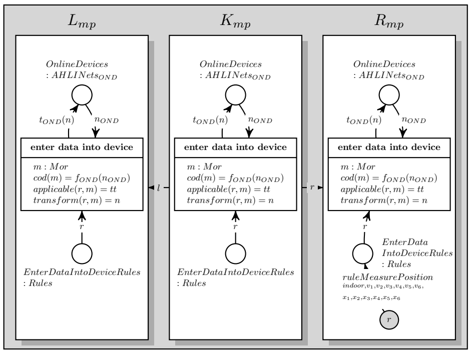
These tasks are internally stored within the Alarm Clock 2.0 device net, so that it is able to check the fulfilment of each task. If an action still remains, the Alarm Clock 2.0 device reminds the user of performing this action in initiating an alarm. This will be done every 15 minutes, in order to ensure enough time to fulfil the specific task.
The process of reminding takes place, until the user should leave the apartment to keep the appointment. To get to the appointment, a duration of 30 minutes is assumed.

In this phase of the scenario, the following steps will be described:

- The inhabitant is in the bathroom of the Living Place apartment. The position of the user will be recognized and interpreted by the Indoor-Positioning-System.
- Afterwards the interpreted position will be passed to the Message Oriented Middleware which distributes the data.
- The Context Interpreter being a part of the Message Oriented Middleware evaluates a 5W1H[3] context out of this information. The resulting data will be distributed.
- The Alarm Clock 2.0 receives and processes the 5W1H context. As a result, the task of washing is fulfilled and the state "unwashed" will be removed from the internal list.

[3] See [JW05].

1. Concrete AHOI rule $MeasurePosition_{user,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ for generating position data for the Indoor-Positioning-System (see 7.103):

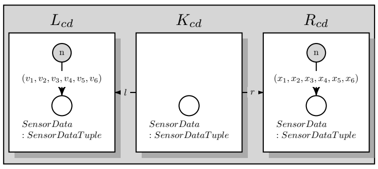


Similar to the measurement of pressure data in the Intelligent Bed (see illustrations 12.-15. in last section 8.6), a concrete AHOI rule out of a huge set of AHOI rules will be applied which assigns its corresponding AHLI rule $ruleMeasurePosition_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$. In the case of adding position data, each v_i, x_i consists of a triple (X_i, Y_i, Z_i) describing the current 3D position coordinate of the user ($i \in \{1, 2, 3, 4, 5, 6\}$).

The indices of the concrete user and its corresponding AHLI rule used in that step, will be: $v_i, x_j = (0, 0, 0)$, for $, j \in \{1, 2, 4, 5, 6\}$ and $x_3 = (9, 6, 0)$. That means, only the third sensor is able to detect the position of the user, because of the inhabitant sojourning in the range of measurement of the third sensor.

Figure 8.112: Send Position *bathroom*

2. Concrete AHLI rule $ruleMeasurePosition$ $_{indoor,v_1,v_2,v_3,v_4,v_5,v_6,x_1,x_2,x_3,x_4,x_5,x_6}$ for generating position data for the Indoor-Positioning-System (see 7.75):

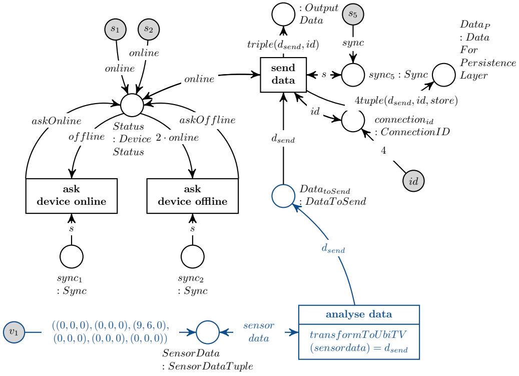

L_{cd}
K_{cd}
R_{cd}
n
$(v_1, v_2, v_3, v_4, v_5, v_6)$
l
r
n
$(x_1, x_2, x_3, x_4, x_5, x_6)$
$SensorData$
$: SensorDataTuple$

The corresponding AHLI rule will be assigned to the place $EnterDataIntoDeviceRules$: $Rules$ which is situated within the system net (see 7.48). This rule will be applied to a device net in online mode in firing the transition *enter data into device*. A match can be found in using $indoorPositioningSystem_{online}$ out of the predomain of this transition.

The assignements are: $v_i, x_j = (0, 0, 0)$, for $i \in \{1, 2, 3, 4, 5, 6\}, j \in \{1, 2, 4, 5, 6\}$ and $x_3 = (9, 6, 0)$.

Figure 8.113: Send Position *bathroom*

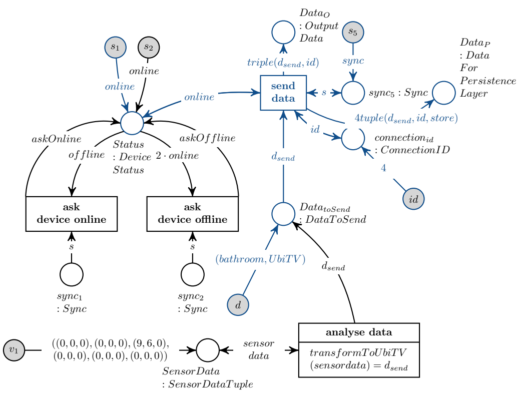
3. Device net $indoorPositioningSystem'_{online}$ after entering new data (see 7.17):



After executing the AHLI rule $ruleMeasurePosition_{indoor,x_1,x_2,x_3,x_4,x_5,x_6}$, $(x_i = (0, 0, 0), i \in \{1, 2, 4, 5, 6\}, x_3 = (9, 6, 0))$, the data token assigned to $SensorData$: $SensorDataTuple$ is replaced by the tuple given in this image. The transition *analyse data* is now activated.

Figure 8.114: Send position *bathroom*

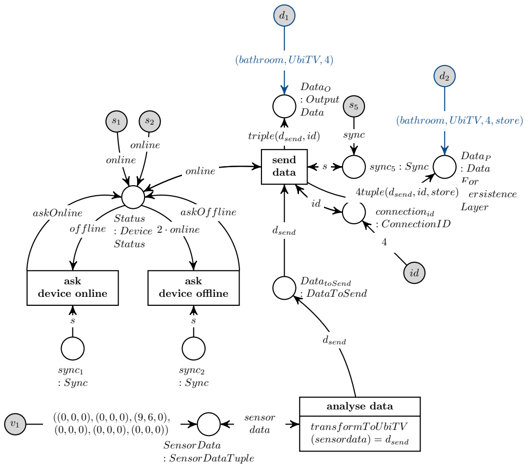
4. Device net $indoorPositioningSystem''_{online}$ while processing data:



After firing the transition *analyse data*, the token $(bathroom, UbiTV)$ was infered by the operation $transformToUbiTV$. A detailed definition if this operation is given in the Σ_{OL}-algebra A_{OL} of the Object Level in 11.1. As next step, the transition *send data* will fire which transforms the token in order to put it to the place $Data_O$ resp. $Data_P$.

Figure 8.115: Send Data *bathroom*

5. Device net $indoorPositioningSystem'''_{online}$ while processing data:

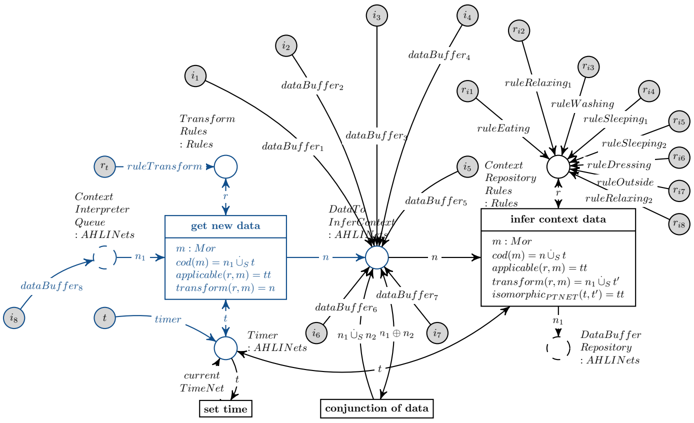


After firing the transition *send data*, a token $(bathroom, UbiTV, 4)$ is assigned to $Data_O$: $OutputData$ which should be distributed by the message broker directly to all devices subscribed to the topic "UbiTV". A second token $(bathroom, UbiTV, 4, store)$ is assigned to $Data_P$: $DataForPersistenceLayer$, which should be stored to the persistence layer of the Message Oriented Middleware represented by the system net.

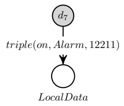
Figure 8.116: Send Data *bathroom*

The information will be collected by the Message Oriented Middleware in firing the transitions *receive data from device*, *copy data to queues and to context interpreter* and *send data to device* which are parts of the system net $System^{15}$ (e.g. see 7.26).
The only device subscribed to this topic "UbiTV" is the Location-Based Screen, which will receive this message. Because of the current position "bathroom", the states of both TV sets represented by the places TV_1 : $TVState$ resp. TV_2 : $TVState$ in the device net of the Location-Based Screen will stay unchanged, that means, both TV sets are still on but inactive.
The transition *copy data to queues and to context interpreter* performs two tasks: copying the new data to the corresponding queue of all devices subscribed to the specific topic. In addition, the information will be passed to the Context Interpreter, which is a part of the Message Oriented Middleware. It infers a new context out of the given data according to a set of rules. The model of the Context Interpreter is presented in detail in 7.53. In the following step, the inference process of an information will be illustrated.

6. Extract from system net $System^{15}$ showing the part of the Context Interpreter (see 7.53):



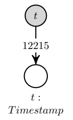
It is assumed, that in the last data processing steps of the Message Oriented Middleware, the Context Interpreter collected the incoming tokens in firing the transition *get new data*. Because of that, all past message token are equiped with an older timestamp and are assigned to the place *DataToInferContext : AHLINets*.
This following illustration shows, as an example, the object net $dataBuffer_7$. It contains the data sent by the Alarm Clock 2.0 $(on, Alarm)$ and the timestamp of the moment, when the Context Interpreter fired the transition *get new data* in order to get this data and provide it for a possible derivation of a new context.



The new object net $dataBuffer_8$, which contains the data token $triple(bathroom, UbiTV, 4)$, will be transformed in firing the transition *get new data*. It applies the rule $ruleTransform$ (see 7.62 or step 12 in next part of the scenario) to the disjoint union of two nets from its predomain. Now, it takes the object nets: $dataBuffer_8 \dot{\cup}_S time$ in order to equip the token with the current timestamp. The Context Interpreter only infers new data which have the same timestamp than the object net $timer$, old data will be ignored.

Figure 8.117: Send Data *bathroom*, Context Interpreter

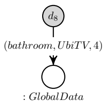
7. Object net $timer$ **containing the current timestampp** 12215:



The current timestamp is assigned to th place t : $Timestamp$ within the object net $timer$. The actual timestamp is now 12215, four minutes after the inhabitant has been woken up by the Living Place system.

Figure 8.118: Send Data *bathroom*, Context Interpreter

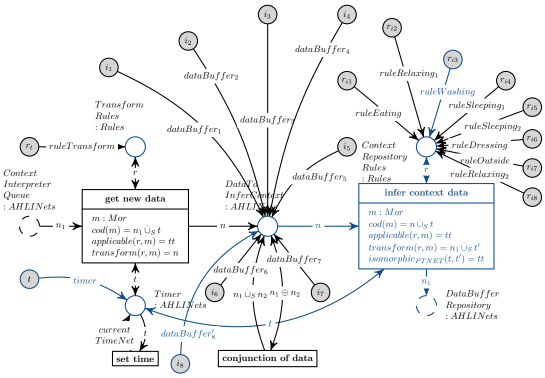
8. Object net $dataBuffer_8$ **containing the new data** $triple(bathroom, UbiTV, 4)$:



The object net $dataBuffer_8$ contains the data token $triple(bathroom, UbiTV, 4)$ which was created and sent by the Indoor-Positioning-System.

Figure 8.119: Send Data *bathroom*, Context Interpreter

9. Extract from system net $System^{16}$ showing the part of the Context Interpreter:



The token is transformed and forwarded to the place $DataToInferContext : AHLINets$. The resulting object net is shown in image 10. Now, the transition *infer context data* is able to fire and concurrently is applyies the rule $ruleWashing$ to the disjoint union of $dataBuffer'_8$ and the object net *timer* containing the current timestamp. The rule is presented in image 11.

Figure 8.120: Send Data *bathroom*, Context Interpreter

10. Object net $dataBuffer'_8$ after firing *get new data*:

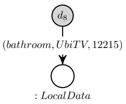


Figure 8.121: Send Data *bathroom*, Context Interpreter

11. AHLI rule $ruleWashing$ (see 7.57):

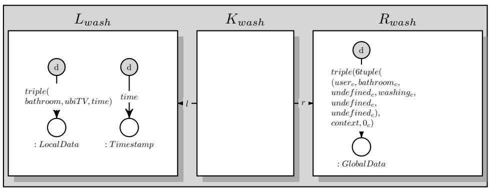
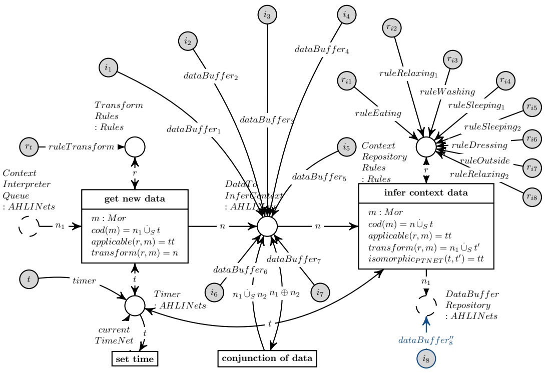

L_wash
K_wash
R_wash
d
triple(bathroom, ubiTV, time)
: LocalData
d
time
: Timestamp
l
r
d
triple(6tuple((user_c, bathroom_c, undefined_c, washing_c, undefined_c, undefined_c), context, 0_c)
: GlobalData

Figure 8.122: Send Data *bathroom*, Context Interpreter

12. Extract from system net $System^{17}$ showing the part of the Context Interpreter:



After firing the transition *infer context data*, a new 5W1H context was infered out of the data $(bathroom, UbiTV)$. The object net containing the 5W1H context is illustrated in the next image.
The Context Interpreter contains a set of rules assigned to $ContextRepositoryRules : Rules$. Each rule is presented in detail in the modelling of the Context Interpreter in 7.53.

Figure 8.123: Send Data *bathroom*, Context Interpreter

13. Object net $dataBuffer''_8$ after firing *infer context data*:

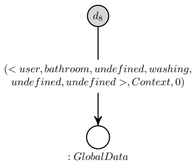
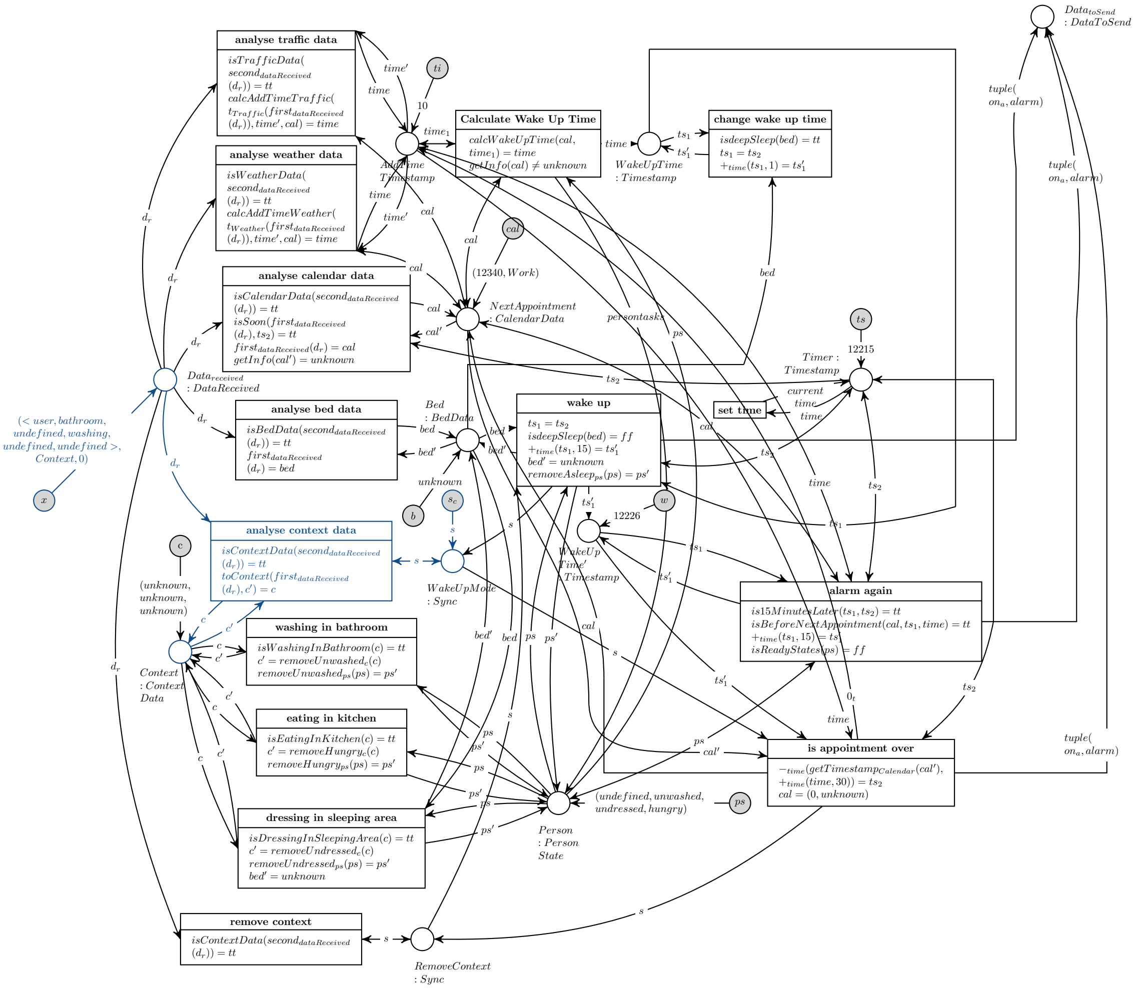

d₈
(< user, bathroom, undefined, washing, undefined, undefined >, Context, 0)
: GlobalData

Figure 8.124: Send Data *bathroom*, Context Interpreter

The 5W1H context information infered by the Context Interpreter will be distributed to all devices subscribed to the topic “Context”. For that, the transitions *copy data to queues and to context interpreter* and *send data to device* will be used, analogue to the processing of information sent by a connected device in online mode.

Currently, only the Alarm Clock 2.0 is subscribed to the topic “Context”, hence the data will only be sent to the net $alarmClock2.0^{15}_{online}$, which receives the data in firing its transition *receive data*. The following illustrations describe the processing of this context data.

14. Extract from $alarmClock2.0^{16}_{online}$ showing individual processing part of the device:

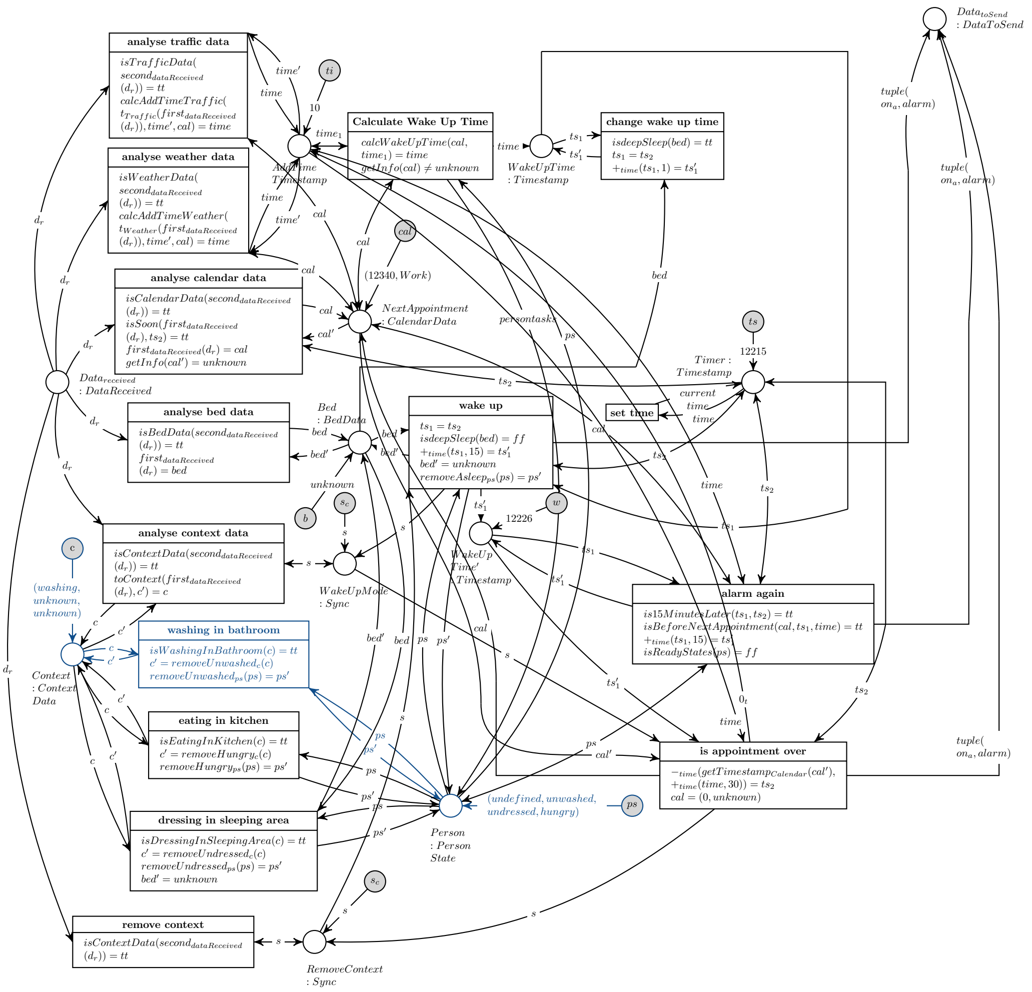

analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts_1'$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts_1'$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts_1'$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{received}$: DataReceived
$(< user, bathroom, undefined, washing, undefined, undefined >, Context, 0)$
x
d_r
AddTime Timestamp
ti
10
$time$
$time'$
$time_1$
cal
$(12340, Work)$
NextAppointment : CalendarData
cal'
Bed : BedData
bed
bed'
unknown
b
s_c
s
WakeUpMode : Sync
c
$(unknown, unknown, unknown)$
Context : Context Data
c'
WakeUpTime : Timestamp
ts_1
ts_1'
ts_2
persontasks
ps
ps'
bed
ts
12215
Timer : Timestamp
current time
time
w
12226
WakeUp Time' : Timestamp
0_t
Person : Person State
$(undefined, unwashed, undressed, hungry)$
ps
RemoveContext : Sync
$Data_{toSend}$: DataToSend
$tuple(on_a, alarm)$

The token $(< user, bathroom, undefined, washing, undefined, undefined >, Context, 0)$ assigned to $Data_{received}$ belongs to the topic “Context”, therefore, the transition *analyse context data* is able to fire. It contains two operations, the first one *isContextData* checks, if the second component of the triple is identical to the topic “Context”. The second operation *toContext* evaluates the input data and possibly changes the token assigned to *Context* : *ContextData* which represents the interpreted contexts, which were not processed until now. Initially this token is set to $(unknown, unknown, unknown)$. After receiving the 5W1H context, that the user is washing in the bathroom, it will be replaced by $(washing, unknown, unknown)$.

A definition of both operations is given within the Σ_{OL}-algebra A_{OL} for the Object Level in 11.1.

Figure 8.125: Alarm Clock 2.0 Processes Data *bathroom*

15. Extract from $alarmClock2.0^{17}_{online}$ showing individual processing part of the device:

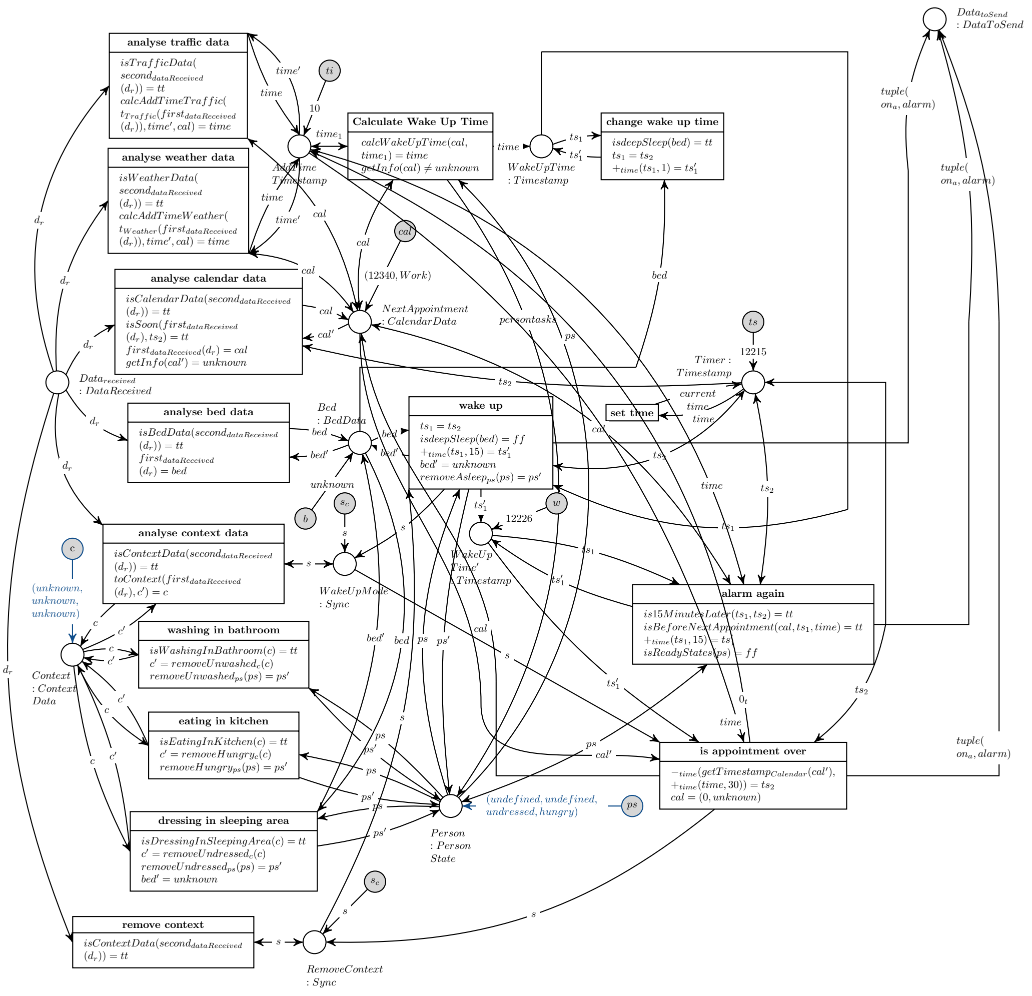

analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts'_1$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts'_1$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts'_1$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{received}$: DataReceived
AddTime Timestamp
ti
10
$time_1$
time
time'
cal
(12340, Work)
NextAppointment : CalendarData
Bed : BedData
bed
bed'
unknown
b
s_c
s
WakeUpMode : Sync
c
(washing, unknown, unknown)
Context : Context Data
c'
d_r
ps
ps'
(undefined, unwashed, undressed, hungry)
Person : Person State
RemoveContext : Sync
WakeUpTime : Timestamp
ts_1
ts'_1
ts_2
WakeUp Time' : Timestamp
12226
w
persontasks
Timer : Timestamp
12215
ts
current time
0_t
$Data_{toSend}$: DataToSend
tuple(on_a, alarm)

After firing the transition *analyse context data*, the token assigned to *Context* : *ContextData* is replaced by the triple (*washing*, *unknown*, *unknown*). The transition *washing in bathroom* will fire next. It checks the first component of the triple, if it is equal to "washing", if true, the first component of this triple will be reset (in executing the operation $removeUnwashed_c$. Additionally, the second component *unwashed* of the 4-tuple set to the place *Person* : *PersonState* will be reset in replacing it with *undefined*.

That means, the user has fulfilled one of the tasks internally stored within the Alarm Clock 2.0 and to whose the inhabitat should be reminded of.

Figure 8.126: Alarm Clock 2.0 Processes Data *bathroom*

16. Extract from $alarmClock2.0^{18}_{online}$ showing individual processing part of the device:


analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time', cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time', cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal') = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c') = c
washing in bathroom
isWashingInBathroom(c) = tt
c' = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps'
eating in kitchen
isEatingInKitchen(c) = tt
c' = removeHungry_c(c)
removeHungry_ps(ps) = ps'
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c' = removeUndressed_c(c)
removeUndressed_ps(ps) = ps'
bed' = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts'_1
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts'_1
bed' = unknown
removeAsleep_ps(ps) = ps'
set time
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal'), +_time(time, 30)) = ts_2
cal = (0, unknown)
Data_received : DataReceived
AddTime : Timestamp
NextAppointment : CalendarData
Bed : BedData
WakeUpMode : Sync
Context : Context Data
Person : Person State
RemoveContext : Sync
WakeUpTime : Timestamp
WakeUp Time' : Timestamp
Timer : Timestamp
Data_toSend : DataToSend
ti
10
cal
(12340, Work)
ts
12215
w
12226
b
unknown
s_c
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undressed, hungry)
tuple(on_a, alarm)
d_r
time
time'
time_1
cal'
bed
bed'
ts_1
ts'_1
ts_2
current time
personstasks
s
c'
0_t

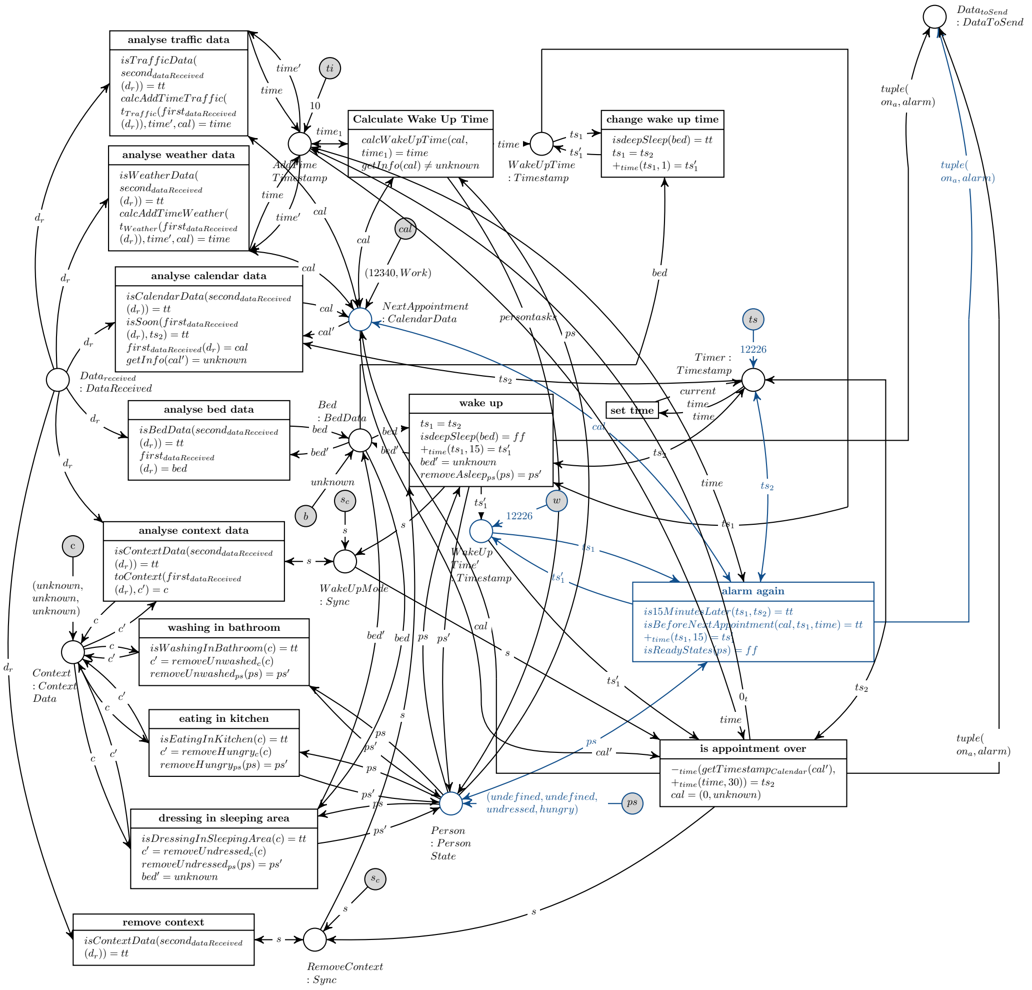
After processing the data ($< \ user, bathroom, undefined, washing, undefined, undefined \ >, Context, 0$), the task *unwashed* is fulfilled and will be removed from the list of unrealised tasks assigned as 4-tuple to the place $Person : PersonState$.

Figure 8.127: Alarm Clock 2.0 Processes Data *bathroom*

8.7 Preparation Phase For an Appointment: User is Dressing in Sleeping Area

The inhabitant stays in the bathroom for 20 minutes. Meanwhile, the Alarm Clock 2.0 checks the internal task list an generates an alarm because there are still actions left to fulfil by the inhabitant. After leaving the bathroom, the user enters the sleeping area to dress. This will be recognized by the Indoor-Location-Based system leading to a deletion of the task *undressed* within the Alarm Clock 2.0. Beyond, the Location-Based Screen automatically activates the TV set in the sleeping area.

1. Extract from $alarmClock2.0^{19}_{online}$ showing individual processing part of the device:


analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time', cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time', cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal') = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c') = c
washing in bathroom
isWashingInBathroom(c) = tt
c' = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps'
eating in kitchen
isEatingInKitchen(c) = tt
c' = removeHungry_c(c)
removeHungry_ps(ps) = ps'
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c' = removeUndressed_c(c)
removeUndressed_ps(ps) = ps'
bed' = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts'_1
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts'_1
bed' = unknown
removeAsleep_ps(ps) = ps'
set time
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts'_1
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal'), +_time(time, 30)) = ts_2
cal = (0, unknown)
Data_received : DataReceived
Data_toSend : DataToSend
AddTime Timestamp
NextAppointment : CalendarData
WakeUpTime : Timestamp
WakeUpTime' : Timestamp
Timer : Timestamp
Bed : BedData
WakeUpMode : Sync
Context : Context Data
Person : Person State
RemoveContext : Sync
ti
10
cal
(12340, Work)
ts
12226
b
unknown
s_c
w
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undressed, hungry)
tuple(on_a, alarm)
d_r
time
time'
time_1
ts_1
ts'_1
ts_2
bed
bed'
current time
persontasks
s
0_t
cal'
ps'

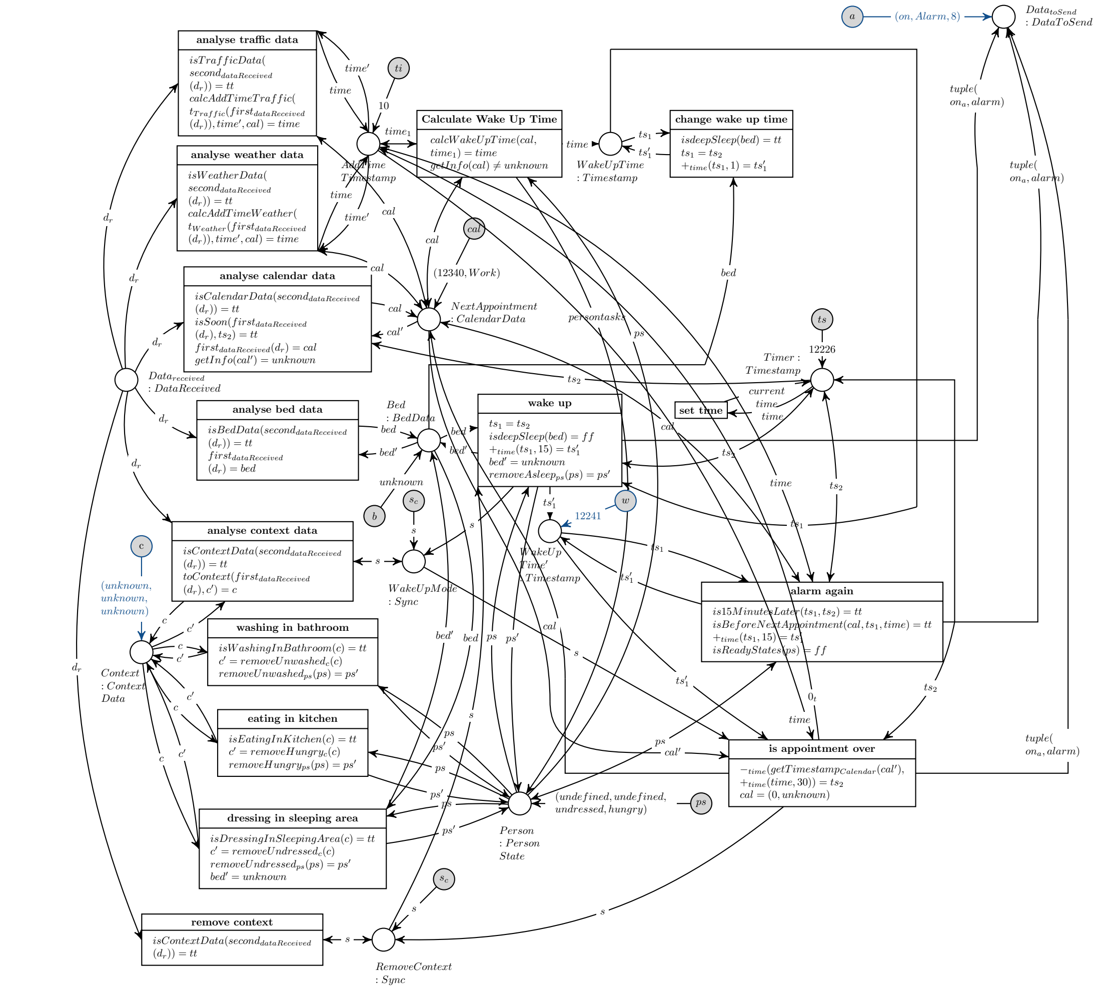
The internal timer has switched to 12226. Now, the transition *alarm again* is able to fire and executing the operations as follows:

- $is15MinutesLater(ts_1, ts_2) = tt$: Returns tt because the timestamps ts_1 and ts_2 are equal and the condition is fulfilled.
- $isBeforeNextAppointment(cal, ts_1, time) = tt$: The actual appointment lies in the future, so this operation returns tt.
- $+_{time}(ts_1, 15) = ts'_1$: The timestamp for the next reminding will be increased by 15 minutes and it will be returned to the place $WakeUpTime'$.
- $isReadyStates(ps) = ff$: As long as there are still tasks to fulfil available ($ps \neq (undefined, undefined, undefined, undefined)$), this operation returns tt.

All conditions of the transition *alarm again* are fulfilled, so a alarm will be initiated in producing the token $(on, Alarm)$.

Figure 8.128: Alarm Clock 2.0 Sends Alarm

2. Extract from $alarmClock2.0^{20}_{online}$ showing individual processing part of the device:



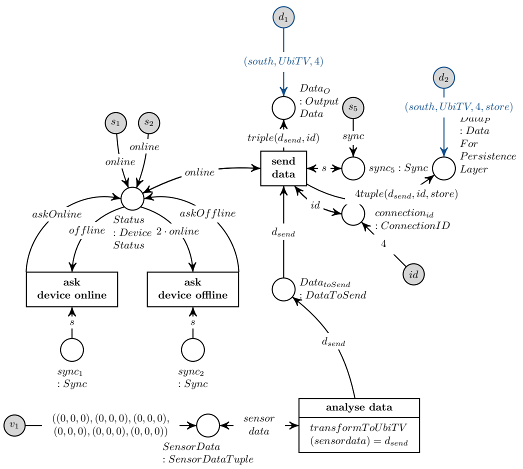
a
(on, Alarm, 8)
Data_toSend : DataToSend
tuple(on_a, alarm)
tuple(on_a, alarm)
tuple(on_a, alarm)
analyse traffic data
isTrafficData(second_dataReceived (d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived (d_r)), time′, cal) = time
analyse weather data
isWeatherData(second_dataReceived (d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived (d_r)), time′, cal) = time
time′
time
ti
10
time_1
AddTime Timestamp
time
time′
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
time
WakeUpTime : Timestamp
ts_1
ts′_1
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts′_1
cal
cal
cal
(12340, Work)
cal
bed
analyse calendar data
isCalendarData(second_dataReceived (d_r)) = tt
isSoon(first_dataReceived (d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal′) = unknown
cal
cal′
NextAppointment : CalendarData
persontasks
ps
ts
12226
Timer : Timestamp
d_r
d_r
d_r
Data_received : DataReceived
ts_2
current time
time
set time
analyse bed data
isBedData(second_dataReceived (d_r)) = tt
first_dataReceived (d_r) = bed
Bed : BedData
bed
bed′
bed
bed′
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts′_1
bed′ = unknown
removeAsleep_ps(ps) = ps′
cal
ts_2
time
ts_2
unknown
b
s_c
s
ts′_1
w
12241
ts_1
d_r
d_r
analyse context data
isContextData(second_dataReceived (d_r)) = tt
toContext(first_dataReceived (d_r), c′) = c
s
WakeUpMode : Sync
WakeUp Time′ : Timestamp
ts_1
ts′_1
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts′_1
isReadyStates(ps) = ff
c
(unknown, unknown, unknown)
c
c′
c
c′
washing in bathroom
isWashingInBathroom(c) = tt
c′ = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps′
Context : Context Data
d_r
c′
c
eating in kitchen
isEatingInKitchen(c) = tt
c′ = removeHungry_c(c)
removeHungry_ps(ps) = ps′
c
c′
bed′
bed
ps
ps′
cal
s
s
ts′_1
0_t
time
ts_2
ps
ps′
ps
ps′
ps
ps′
ps
cal′
is appointment over
−_time(getTimestamp_Calendar(cal′), +_time(time, 30)) = ts_2
cal = (0, unknown)
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c′ = removeUndressed_c(c)
removeUndressed_ps(ps) = ps′
bed′ = unknown
(undefined, undefined, undressed, hungry)
ps
Person : Person State
s_c
s
remove context
isContextData(second_dataReceived (d_r)) = tt
s
RemoveContext : Sync
s

The next timestamp for a possible reminding is increased to 12241 assigned to the place $WakeUpTime'$: $Timestamp$. Furthermore the token $(on, Alarm)$ is set to $Data_{toSend}$ and ready for the next processing steps: The token will be completed and assigned to the places $Data_O$ resp. $Data_P$ within the Alarm Clock 2.0 device net. Aftwerwards it will be collected by the Message Oriented Middleware, analysed in order to distribute the information to all devices subscribed to "Alarm". These devices will produce the defined reaction to this signal.

In the given scenario, only the Ambient Light is able to react to this signal in switching the light on in dimmed mode or the Location-Based Screen reacts to messages belonging to the topic "Alarm" in switching on and activate a TV set, if the person is within its visible range. The processing of messages concerning the topic "Alarm" is presented in the illustrations 36-42 of the forth part of the whole scenario, see 8.5.

Figure 8.129: Alarm Clock 2.0 Sends Alarm

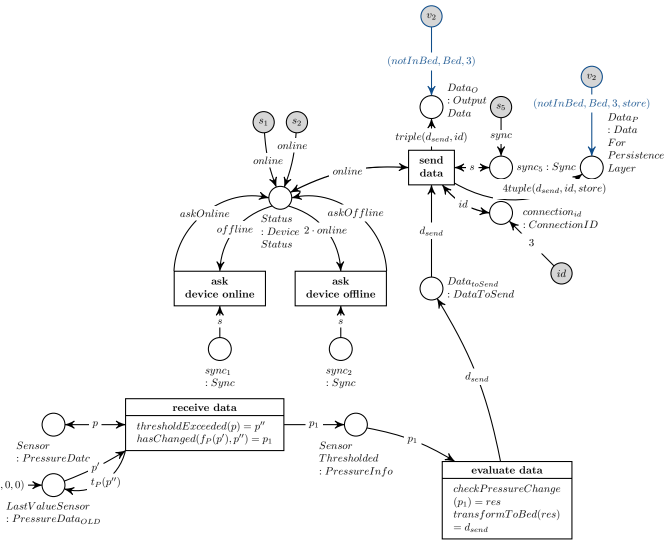
3. Indoor-Positioning-System sends *south*:



As next step, the Indoor-Positioning-System measures values leading to the resulting message $(south, UbiTV, 4)$, as shown in this illustration. The process of entering and processing new measurement values are presented in explicit in images 1-5 in last part of the scenario (see 8.6).

Figure 8.130: Send Data *south*

4. Intelligent Bed sends *notInBed*:



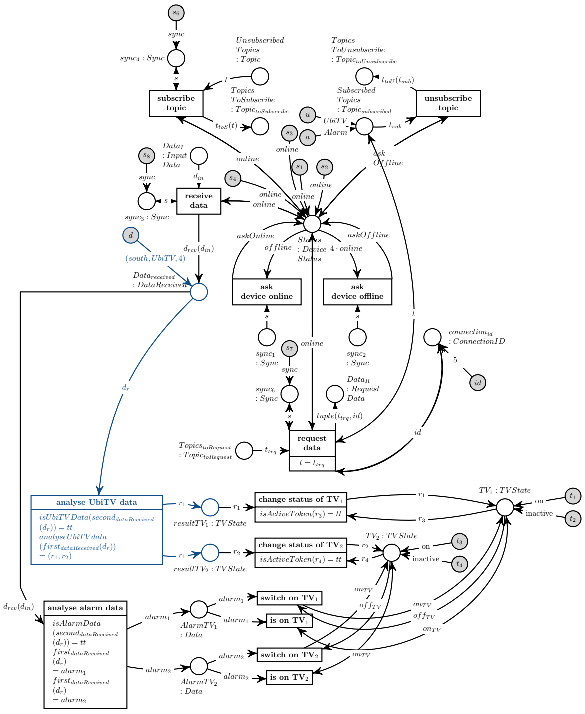
Analogue to the steps presented in 12-18 in the forth part of this simulation of the scenario (see 8.5), the Intelligent Bed recognizes no pressure on its sensors, so the data $notInBed$ will be transmitted to the Message Oriented Middleware. The image shows the state of the device net $intelligentBed^5_{online}$, where the information is evaluated and ready to be accepted and processed by the Message Oriented Middleware.

Figure 8.131: Send Data *notInBed*

Both kinds of information send by the Indoor-Positioning-System and by the Intelligent bed will be processed within the Message Oriented Middleware and distributed to all devices subscribed to the topic “UbiTV” resp. “Bed”. Furthermore, both data will be forwarded to and interpreted by the Context Interpreter.

The Location-Based-Sceen is subscribed to the topic “UbiTV”. The processing of this message will be shown first. Afterwards, the inferece of both information within the Context Interpreter will be described. The last step presented is, that the Alarm Clock 2.0 receives and evaluates the messages sent by the Intelligent Bed and by the Context Interpreter.

5. Location-Based Screen processes the token $(south, UbiTV, 4)$ (see 7.19):



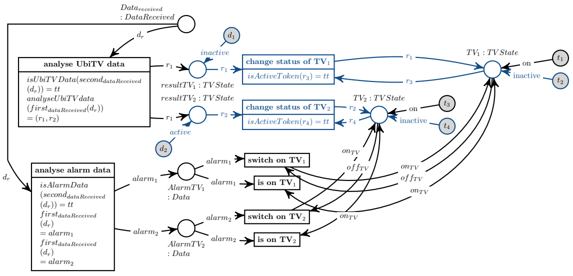
In the image, the token $triple(south, UbiTV, 4)$ has already been passed from the place $Data_I : InputData$ to the place $Data_{received} : DataReceived$ in firing the transition *receive data*. Now, the transition *analyse UbiTV data* is active, which executes its operations as follows:

- $isUbiTVData(second_{dataReceived}(d_r)) = tt$ is fulfilled because the topic of the information triple is equal to "UbiTV".
- $analyseUbiTVdata(first_{dataReceived}(d_r)) = (r_1, r_2)$ returns the tuple $(inactive, active)$ due to the first component of the received data is *south*, which characterises, that the inhabitant is currently in the sleeping area.

As result, the value *inactive* will be transferred to the place $resultTV_1 : TVState$ and the value *active* will be assigned to $resultTV_2 : TVState$.

Figure 8.132: Location-Based Screen receives *south*

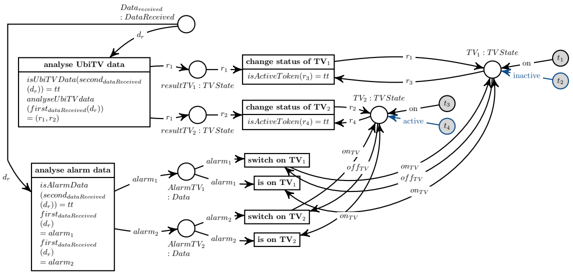
6. Extract from net $locationBasedScreen^5_{online}$ showing device specific part:



As next step, both transitions *change status of* TV_i $(i \in \{1, 2\})$ are able to fire. They take a token, which contains the value *active* or *inactive* from the place $TV_i : TVState$ and replace it by the token, which has been assigned to the corresponding place $resultTV_i$. The outcome of the firing of both transitions is illustrated in the next image.

Figure 8.133: Location-Based Screen receives *south*

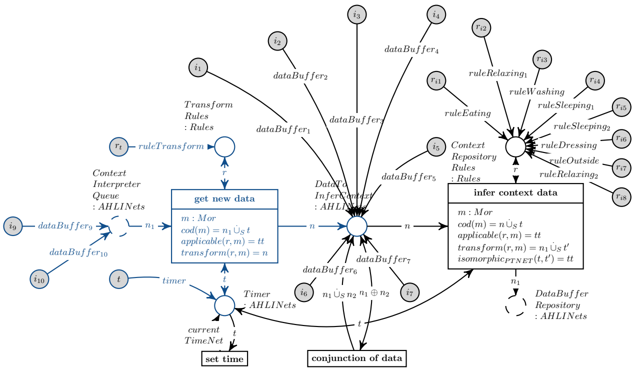
7. Extract from net $locationBasedScreen^6_{online}$ showing device specific part:



Both TV sets are still switched on. Furthermore, the TV set located in the sleeping area is activated because the resident sojourns within its view. The state of the TV set in the sleeping area is represented by the place $TV_2 : TVState$, whereas $TV_1 : TVState$ stands for the state of the TV set situated in the lounge area.

Figure 8.134: Location-Based Screen receives *south*

8. Extract from system net $System^{18}$ showing the part of the Context Interpreter:



Both information sent by the Indoor-Positioning-System and by the Intelligent Bed are assigned to the place $ContextInterpreterQueue : AHLINets$. Each token consists of an AHLI Net which contains the information, as shown in the images 9 and 10. The AHLI net $dataBuffer_9$ contains the data $triple(south, UbiTV, 4)$ assigned as token to the place $: GlobalData$, whereas $dataBuffer_{10}$ contains $triple(notInBed, Bed, 3)$.
Beyond, the AHLI net $timer$ has changed in replacing the timestamp by the current one 12231 as presented in the 11th picture.

The transition *get new data* is now activated and able to fire. It will fire twice in order to process both tokens assigned to $ContextInterpreterQueue$. In doing so, the rule $ruleTransform$ will be applied to $dataBuffer_9 \dot{\cup}_S timer$ resp. $dataBuffer_{10} \dot{\cup}_S timer$.

Figure 8.135: Context Interpreter gets *south* and *notInBed*

9. Object net $dataBuffer_9$:

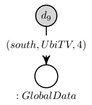


Figure 8.136: Context Interpreter gets *south* and *notInBed*

10. Object net $dataBuffer_{10}$:

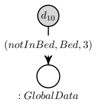


Figure 8.137: Context Interpreter gets *south* and *notInBed*

11. Object net $timer$ containing the current timestampp 12231:

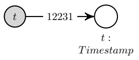
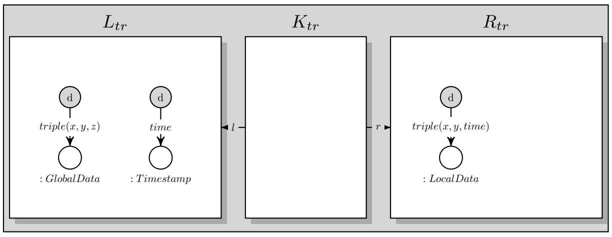


Figure 8.138: Context Interpreter gets *south* and *notInBed*

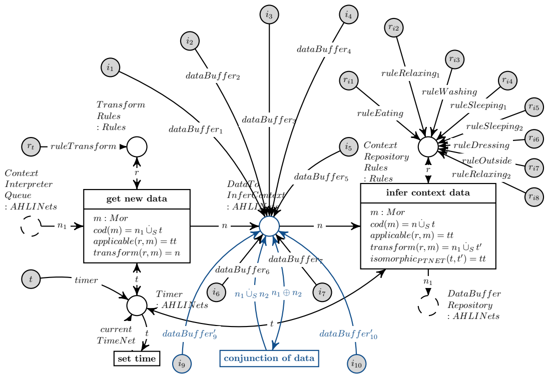
12. AHLI rule $ruleTransform$ of the Context Interpreter (see 7.62):



The rule $ruleTransform$ will be applied to a data record assigned to $ContextInterpreterQueue$: $AHLINets$ and in using the timestamp represented by the AHLI net assigned to $Timer$: $AHLINets$ when the transition *get new data* fires. This rule removes the connection id from the triple containing the message and replaces it by the current time.

Figure 8.139: Context Interpreter gets *south* and *notInBed*

13. Extract from system net $System^{19}$ showing the part of the Context Interpreter:



The transition *conjunction of data* needs to fire in order to combine both object nets $dataBuffer'_9$ and $dataBuffer'_{10}$ so that the rule $ruleDressing$ is able to be applied in the next step.

Figure 8.140: Context Interpreter gets *south* and *notInBed*

14. Object net $dataBuffer'_9$:

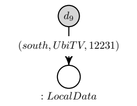


Figure 8.141: Context Interpreter gets *south* and *notInBed*

15. Object net $dataBuffer'_{10}$:

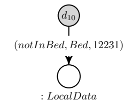
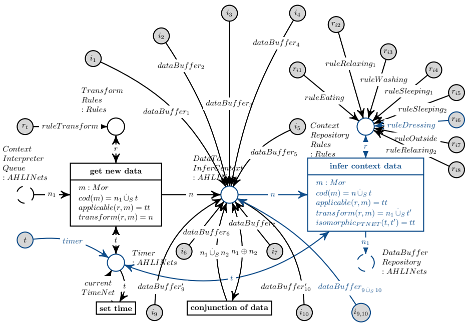


Figure 8.142: Context Interpreter gets *south* and *notInBed*

16. Extract from system net $System^{20}$ showing the part of the Context Interpreter:



Finally, the transition *infer context data* is able to fire and the rule *ruleDressing* can be applied to the object net $dataBuffer_{9 \dot{\cup}_S 10}$ and *timer*. The object net $dataBuffer_{9 \dot{\cup}_S 10}$ is illustrated in 16, the AHLI rule *ruleDressing* in 18. The AHLI rule compares both timestamps being part of the triples assigned to : *LocalData* with the timestamp assigned to : *Timestamp* in order to use new data and no older ones. If all timestamps are equal, the 5W1H context <*user, sleepingArea, undefined, dressing, undefined, undefined*> is inferred.

Figure 8.143: Context Interpreter gets *south* and *notInBed*

17. Object net $dataBuffer_{9 \dot{\cup}_S 10}$:

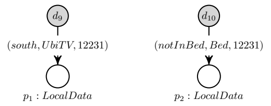


Figure 8.144: Context Interpreter gets *south* and *notInBed*

18. AHLI rule *ruleDressing* (see 7.60):

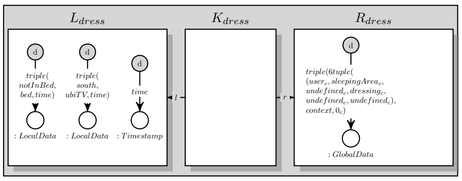
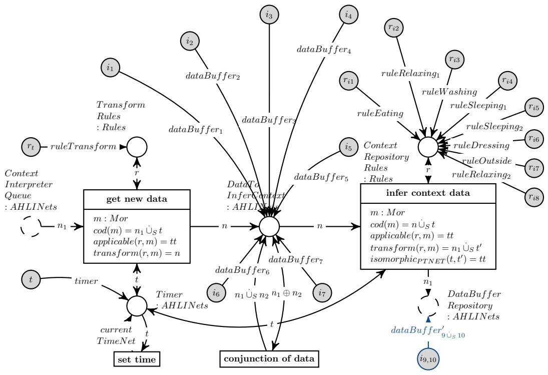

L_{dress}
K_{dress}
R_{dress}
d
triple(notInBed, bed, time)
: LocalData
d
triple(south, ubiTV, time)
: LocalData
d
time
: Timestamp
l
r
d
triple(6tuple((user_c, sleepingArea_c, undefined_c, dressing_c, undefined_c, undefined_c), context, 0_c)
: GlobalData

Figure 8.145: Context Interpreter gets *south* and *notInBed*

19. Extract from system net $System^{21}$ showing the part of the Context Interpreter:



After firing *infer context data*, the object net $dataBuffer'_{9 \dot{\cup}_S 10}$ is assigned as token to $DataBufferRepository : AHLINets$. This object net is visualized in the next picture.

Figure 8.146: Context Interpreter gets *south* and *notInBed*

20. Object net $dataBuffer'_{9 \dot{\cup}_S 10}$:

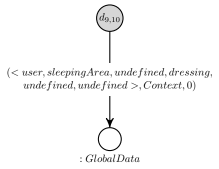
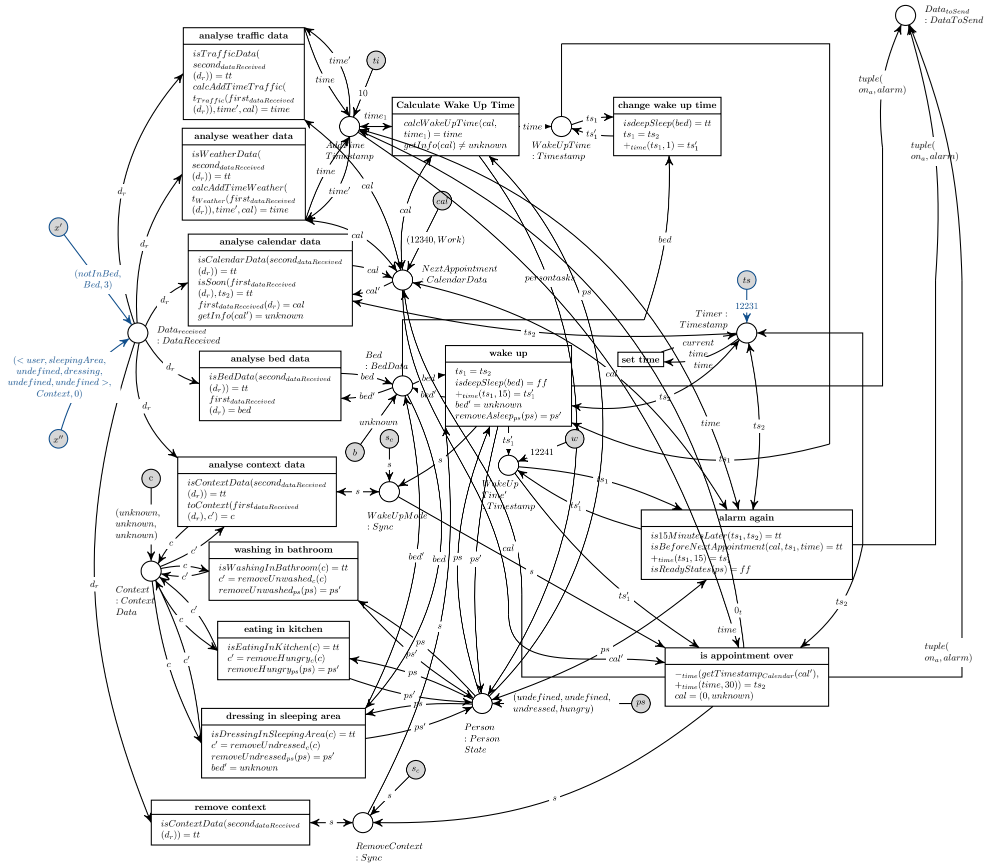

$d_{9,10}$
$(< user, sleepingArea, undefined, dressing, undefined, undefined >, Context, 0)$
$: GlobalData$

Figure 8.147: Context Interpreter gets *south* and *notInBed*

Having infered the new data $(< user, sleepingArea, undefined, dressing, undefined, undefined >, Context, 0)$, the information from the Context Interpreter was processed by the message broker in firing *copy data to queues and to context interpreter* and *send data to device*. The Alarm Clock 2.0 is subscribed to the topics “Bed” and “Context” and, because of that, both information regarding these topics were passed to the Alarm Clock 2.0 device.

21. Extract from $alarmClock2.0^{21}_{online}$ showing individual processing part of the device:


analyse traffic data
isTrafficData(second_dataReceived(d_r)) = tt
calcAddTimeTraffic(t_Traffic(first_dataReceived(d_r)), time′, cal) = time
analyse weather data
isWeatherData(second_dataReceived(d_r)) = tt
calcAddTimeWeather(t_Weather(first_dataReceived(d_r)), time′, cal) = time
analyse calendar data
isCalendarData(second_dataReceived(d_r)) = tt
isSoon(first_dataReceived(d_r), ts_2) = tt
first_dataReceived(d_r) = cal
getInfo(cal′) = unknown
analyse bed data
isBedData(second_dataReceived(d_r)) = tt
first_dataReceived(d_r) = bed
analyse context data
isContextData(second_dataReceived(d_r)) = tt
toContext(first_dataReceived(d_r), c′) = c
washing in bathroom
isWashingInBathroom(c) = tt
c′ = removeUnwashed_c(c)
removeUnwashed_ps(ps) = ps′
eating in kitchen
isEatingInKitchen(c) = tt
c′ = removeHungry_c(c)
removeHungry_ps(ps) = ps′
dressing in sleeping area
isDressingInSleepingArea(c) = tt
c′ = removeUndressed_c(c)
removeUndressed_ps(ps) = ps′
bed′ = unknown
remove context
isContextData(second_dataReceived(d_r)) = tt
Calculate Wake Up Time
calcWakeUpTime(cal, time_1) = time
getInfo(cal) ≠ unknown
change wake up time
isdeepSleep(bed) = tt
ts_1 = ts_2
+_time(ts_1, 1) = ts′_1
wake up
ts_1 = ts_2
isdeepSleep(bed) = ff
+_time(ts_1, 15) = ts′_1
bed′ = unknown
removeAsleep_ps(ps) = ps′
set time
alarm again
is15MinutesLater(ts_1, ts_2) = tt
isBeforeNextAppointment(cal, ts_1, time) = tt
+_time(ts_1, 15) = ts′_1
isReadyStates(ps) = ff
is appointment over
−_time(getTimestamp_Calendar(cal′), +_time(time, 30)) = ts_2
cal = (0, unknown)
Data_received : DataReceived
Data_toSend : DataToSend
AddTime Timestamp
NextAppointment : CalendarData
Bed : BedData
Context : Context Data
WakeUpMode : Sync
RemoveContext : Sync
WakeUpTime : Timestamp
WakeUp Time′ : Timestamp
Timer : Timestamp
Person : Person State
x′
(notInBed, Bed, 3)
(< user, sleepingArea, undefined, dressing, undefined, undefined >, Context, 0)
x″
ti
10
cal
(12340, Work)
b
unknown
s_c
c
(unknown, unknown, unknown)
ts
12231
w
12241
ps
(undefined, undefined, undressed, hungry)
d_r
time
time′
time_1
cal′
bed
bed′
s
c′
ps′
ts_1
ts′_1
ts_2
current time
personstasks
0_t
tuple(on_a, alarm)

The internal timer of the Alarm Clock 2.0 replaced its timestamp with 12231. The Alarm Clock 2.0 device received both data. Now it is able to process both tokens assigned to $Data_{received} : DataReceived$. The context information, that the user is dressing in the sleeping area will be processed analogue to the steps presented in 14-16 in the last part of the scenario (see 8.6):

1. Fire *analyse context data.*
2. The token assigned to $Context : ContextData$ is changed to $(unknown, unknown, dressing)$.
3. Fire *dressing in sleeping area.*
4. The token assigned to $Context : ContextData$ is changed to $(unknown, unknown, unknown)$ and the token assigned to $Person : PersonState$ is changed to $(undefined, undefined, undefined, hungry)$.

In parallel the token $(notInBed, Bed, 3)$ is able to be processed in fire *analyse bed data.* Afterwards, the token assigned to $Bed : BedData$ is replaced by a token $notInBed$. The resulting device net of the Alarm Clock 2.0 after performing all these steps is illustrated in next image.

Figure 8.148: Alarm Clock 2.0 processes information

22. Extract from $alarmClock2.0^{22}_{online}$ showing individual processing part of the device:

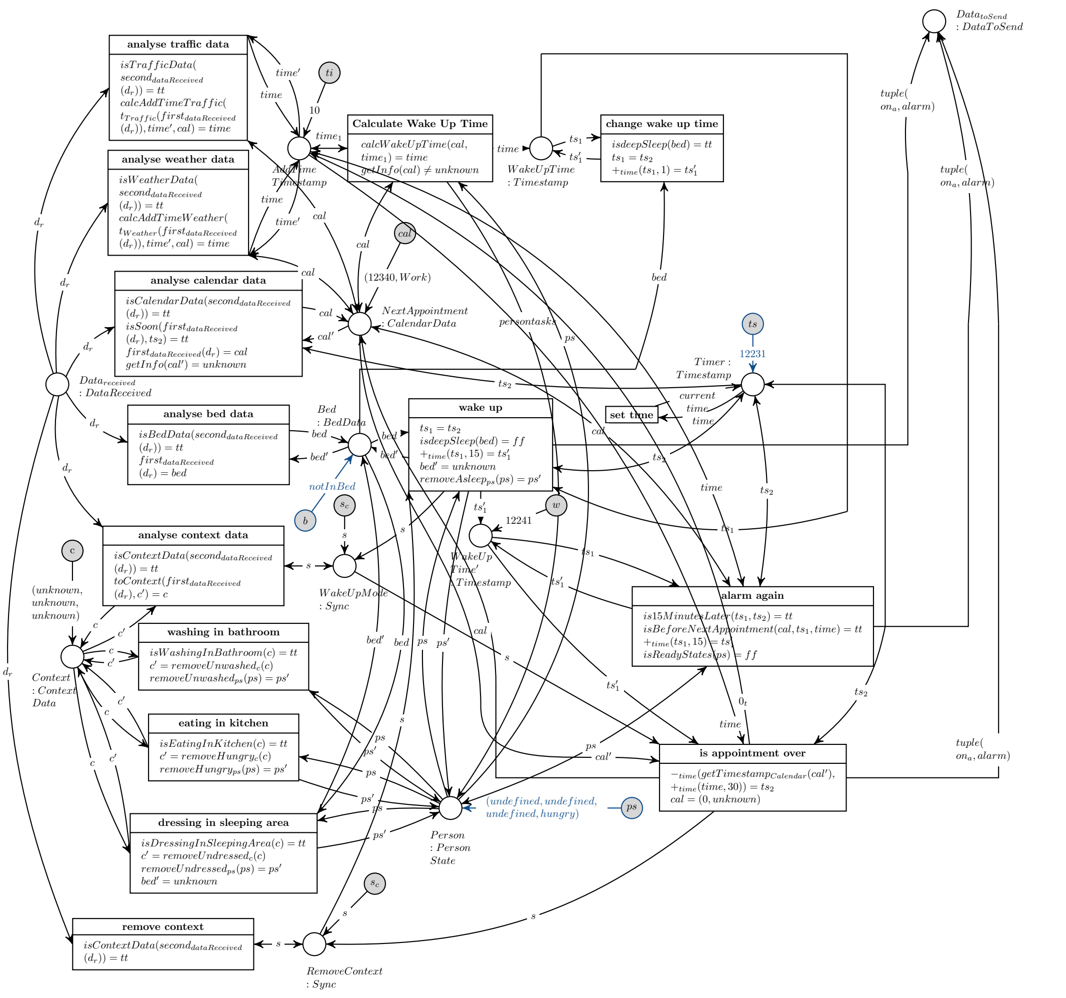

analyse traffic data
$isTrafficData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeTraffic(t_{Traffic}(first_{dataReceived}(d_r)), time', cal) = time$
analyse weather data
$isWeatherData(second_{dataReceived}(d_r)) = tt$
$calcAddTimeWeather(t_{Weather}(first_{dataReceived}(d_r)), time', cal) = time$
analyse calendar data
$isCalendarData(second_{dataReceived}(d_r)) = tt$
$isSoon(first_{dataReceived}(d_r), ts_2) = tt$
$first_{dataReceived}(d_r) = cal$
$getInfo(cal') = unknown$
analyse bed data
$isBedData(second_{dataReceived}(d_r)) = tt$
$first_{dataReceived}(d_r) = bed$
analyse context data
$isContextData(second_{dataReceived}(d_r)) = tt$
$toContext(first_{dataReceived}(d_r), c') = c$
washing in bathroom
$isWashingInBathroom(c) = tt$
$c' = removeUnwashed_c(c)$
$removeUnwashed_{ps}(ps) = ps'$
eating in kitchen
$isEatingInKitchen(c) = tt$
$c' = removeHungry_c(c)$
$removeHungry_{ps}(ps) = ps'$
dressing in sleeping area
$isDressingInSleepingArea(c) = tt$
$c' = removeUndressed_c(c)$
$removeUndressed_{ps}(ps) = ps'$
$bed' = unknown$
remove context
$isContextData(second_{dataReceived}(d_r)) = tt$
Calculate Wake Up Time
$calcWakeUpTime(cal, time_1) = time$
$getInfo(cal) \neq unknown$
change wake up time
$isdeepSleep(bed) = tt$
$ts_1 = ts_2$
$+_{time}(ts_1, 1) = ts'_1$
wake up
$ts_1 = ts_2$
$isdeepSleep(bed) = ff$
$+_{time}(ts_1, 15) = ts'_1$
$bed' = unknown$
$removeAsleep_{ps}(ps) = ps'$
set time
alarm again
$is15MinutesLater(ts_1, ts_2) = tt$
$isBeforeNextAppointment(cal, ts_1, time) = tt$
$+_{time}(ts_1, 15) = ts_1$
$isReadyStates(ps) = ff$
is appointment over
$-_{time}(getTimestamp_{Calendar}(cal'), +_{time}(time, 30)) = ts_2$
$cal = (0, unknown)$
$Data_{received}$: DataReceived
AddTime Timestamp
WakeUpTime : Timestamp
NextAppointment : CalendarData
Bed : BedData
Timer : Timestamp
WakeUp Time' : Timestamp
WakeUpMode : Sync
Context : Context Data
Person : Person State
RemoveContext : Sync
$Data_{toSend}$: DataToSend
ti
10
cal
(12340, Work)
ts
12231
w
12241
b
notInBed
s_c
c
(unknown, unknown, unknown)
ps
(undefined, undefined, undefined, hungry)
$tuple(on_a, alarm)$
d_r
time
time'
$time_1$
cal'
ts_1
ts'_1
ts_2
bed
bed'
persontasks
current time
0_t
c'
ps'
s

Figure 8.149: Alarm Clock 2.0 processed information

The inhabitant does not fulfil all tasks during the preparation time, because he does not have breakfast. Because of that, the Alarm Clock 2.0 reminds the user regularily of the missing task in sending new alarm signals to the Living Place system, until the time for leaving the apartment in order to get to the appointment is reached. The process of generating new data $triple(on, Alarm, 8)$ is shown in the images 1 and 2 in this section.
Another way for the resident to end the constant reminding is the switching off of the Alarm Clock 2.0 (in applying the AHOI rule $StopAlarmClock_{user}$ which is described in 7.1.4.5). The third possibility is presented in the last part of the simulation of this scenario: The user switches the whole Living Place system off and resets it.

8.8 Switch off the whole Living Place system

Only the user is able to switch off the whole Living Place system, normally this is only done in the case of an emergency. For that, a set of AHOI rules need to be applied, whose use case is presented in the first picture. Afterwards the inhabitant will initialise the whole system in applying the AHOI rules, whose use case diagram is shown in the second image.

1. Switching the whole Living Place system off (see 7.1.4.2):

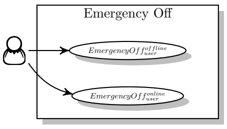

Emergency Off
$EmergencyOff_{user}^{offline}$
$EmergencyOff_{user}^{online}$

Figure 8.150: Use Case: Switch Off in Emergency

--→

2. Initialise the Living Place system (see 7.1.4.3):

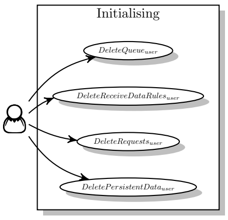
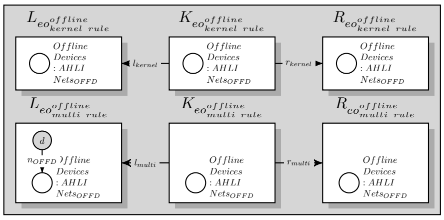

Initialising
$DeleteQueue_{user}$
$DeleteReceiveDataRules_{user}$
$DeleteRequests_{user}$
$DeletePersistentData_{user}$

Figure 8.151: Use Case: Initialise the Whole System

3. User interaction scheme $EmergencyOff_{user}^{offline}$ (see 7.87):

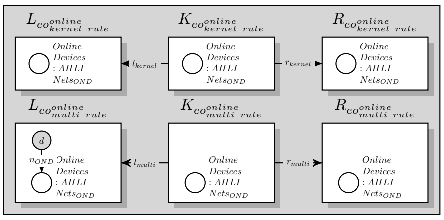

$L_{eo_{kernel\ rule}^{offline}}$
$K_{eo_{kernel\ rule}^{offline}}$
$R_{eo_{kernel\ rule}^{offline}}$
Offline Devices : AHLI Nets$_{OFFD}$
l_{kernel}
r_{kernel}
$L_{eo_{multi\ rule}^{offline}}$
$K_{eo_{multi\ rule}^{offline}}$
$R_{eo_{multi\ rule}^{offline}}$
d
n_{OFFD}
l_{multi}
r_{multi}

In general, an amalgamated rule will be created out of the interaction scheme $EmergencyOff_{user}^{offline}$ based on the maximal match, which deletes all object nets assigned as tokens to the place $OfflineDevices : AHLINets_{OFFD}$ in the system net. However, the current system net $System^{22}$ have no tokens assigned to this place, so that no rule can be created out of this interaction scheme, right now.

Figure 8.152: Emergency Off

--→

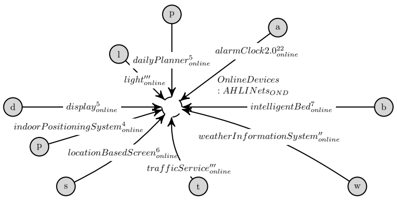
4. User interaction scheme $EmergencyOff_{user}^{online}$ (see 7.86):


$L_{eo_{kernel\ rule}^{online}}$
$K_{eo_{kernel\ rule}^{online}}$
$R_{eo_{kernel\ rule}^{online}}$
Online Devices : AHLI Nets$_{OND}$
l_{kernel}
r_{kernel}
$L_{eo_{multi\ rule}^{online}}$
$K_{eo_{multi\ rule}^{online}}$
$R_{eo_{multi\ rule}^{online}}$
d
n_{OND}
l_{multi}
r_{multi}

In contrast to step 3, a rule can be generated out of the interaction scheme $EmergencyOff_{user}^{online}$ based on the maximal match. It will be applied to the system net $System^{22}$ in order to remove all device in online mode assigned to the place $OnlineDevices : AHLINets_{OND}$.

Figure 8.153: Emergency Off

5. Extract from system net $System^{22}$ **before applying the rule created out of** $EmergencyOff_{user}^{online}$:


p
a
l
$dailyPlanner_{online}^{5}$
$alarmClock2.0_{online}^{22}$
$light_{online}'''$
OnlineDevices : $AHLINets_{OND}$
d
$display_{online}^{5}$
$intelligentBed_{online}^{7}$
b
$indoorPositioningSystem_{online}^{4}$
p
$weatherInformationSystem_{online}''$
$locationBasedScreen_{online}^{6}$
$trafficService_{online}'''$
s
t
w

All devices in online mode are assigned to the place $OnlineDevices$.

Figure 8.154: Emergency Off

--→

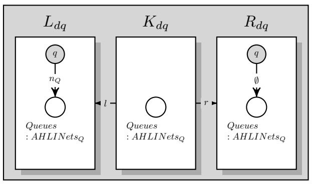
6. Extract from system net $System^{23}$ **after applying the rule created out of** $EmergencyOff_{user}^{online}$:

OnlineDevices : $AHLINets_{OND}$

All object nets are removed from the place $OnlineDevices$ after applying the rule created out of $EmergencyOff_{user}^{online}$. Therfore, no processing of data within the device nets is possible anymore.

Figure 8.155: Emergency Off

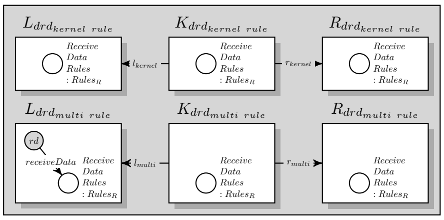
7. AHOI rule $DeleteQueue_{user}$ (see 7.89):



The queue assigned to the place $Queues : AHLINets_Q$ within the system net $System^{23}$ will be reset. In the current system net, the object net q^6 will be replaced (which is equal to the net q'' in image 2 in the third part of the scenario (see 8.4)) by an empty object net.

Figure 8.156: Initialise: Delete Queues

8. User interaction scheme $DeleteReceiveDataRules_{user}$ (see 7.90):



Out of the user interaction scheme, based on the maximal match, $deleteReceiveDataRules_{user}$ a AHOI rule will be created which deletes all rules assigned as tokens to the place $ReceiveDataRules : Rules_R$. Every time a device net was set into online mode, a rule $receiveData$ was put to this place, hence nine rules are currently still available.

Figure 8.157: Initialise: Delete All Rules $receiveData$

9. Extract from system net $System^{24}$ **before applying rule created out of** $DeleteReceiveDataRules_{user}$:

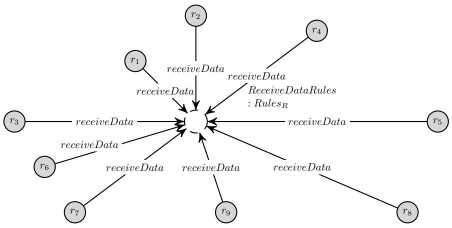


Figure 8.158: Initialise: Delete All Rules $receiveData$

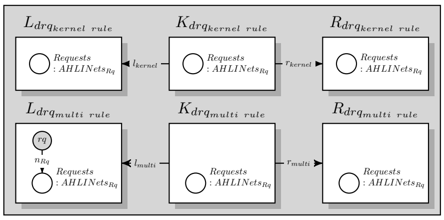
10. Extract from system net $System^{25}$ **after applying rule created out of** $DeleteReceiveDataRules_{user}$:

$ReceiveDataRules$ $: Rules_R$

All tokens are removed from the place $ReceiveDataRules$.

Figure 8.159: Initialise: Delete All Rules $receiveData$

11. System interaction scheme $DeleteRequests_{user}$ (see 7.91):

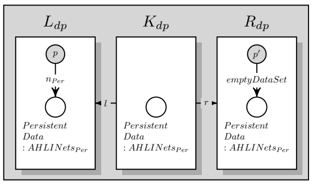


Currently, no requests are processed by the persistence layer of the system net $System^{26}$, so no rule can be created out of this interaction scheme.

Figure 8.160: Initialise: Delete Requests

- - →

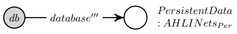
12. AHOI rule $DeletePersistentData_{user}$ (see 7.92):



The token $database'''$ assigned to the place $PersistentData : AHLINets_{Per}$ within the system net $System^{26}$ will be reset in applying the AHLI rule $DeletePersistentData_{user}$.

Figure 8.161: Initialise: Reset Persistent Database

13. Extract from system net $System^{26}$ **and the object net** $database'''$ **before applying the AHOI rule** $DeletePersistentData_{user}$:



Before applying the AHOI rule $DeletePersistentData_{user}$, the object net $database'''$ is assigned as token to the place $PersistentData : AHLINets_{Per}$. The object net $database'''$ is presented below:

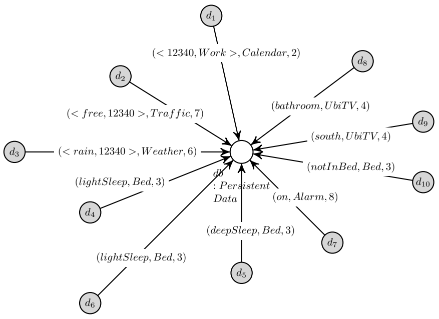
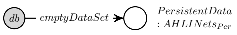


Figure 8.162: Initialise: Reset Persistent Database

- - →

14. Extract from system net $System^{27}$ **after applying the AHOI rule** $deletePersistentData_{user}$:



After applying the AHOI rule $DeletePersistentData_{user}$, the object net $database'''$ is replaced by the net $emptyDataSet$, which is described below:

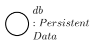


Figure 8.163: Initialise: Reset Persistent Database

8.9 Summary

Within this chapter a possible scenario that can occur in the Living Place and therefore can be processed by the Living Place system is considered by using the model of chapter 7 for a simulation of the considered scenario. Therefore, since the model of chapter 7 is a model of the internal system behaviour of the Living Place system, on the basis of this model it can be reconstructed, how the system behaves within the presented scenario. Thus, this chapter shows up one possibility of how the obtained formal model of the Living Place system can be used to help us to improve our understanding of this system.

Chapter 9

Analysis

In chapter 7 a model of the system of the Living Place is formally defined.

In this chapter it is checked whether this formal overall model of the Living Place system fulfils the following requirements of chapter 4 by formally analyzing this model using formal analysis techniques:

- $\mathbf{r_{mom_{broker_{8(a)}}}}$ The message broker can receive information from different devices in parallel. (Compare to 4.2.3.1)
- $\mathbf{r_{mom_{broker_{8(b)}}}}$ The message broker can **not** receive information from the same device in parallel. (Compare to 4.2.3.1)
- $\mathbf{r_{mom_{user_6}}}$ Those changes of the internal state of the system of the Living Place, which are caused by the execution of the user operations *plugin* device and *unplug* device regarding two different devices, can proceed in parallel. (Compare to 4.2.4.1)
- $\mathbf{r_{user}}$ The change of the internal state of the system of the Living Place, which is caused by the execution of the user operation *enter appointment* and the processing of a previous user operation can proceed in parallel. (Compare to 4.2.4.2)

.

Consider, that as mentioned in chapter 3 the overall model of the Living Place system consists of several submodels that are classified into the following four levels: Data Level, Object Level, System Level and User Level. Thus, the above mentioned requirements towards the overall model of the Living Place system become requirements towards those submodels, which represent the system behaviours that are contained in the requirements, so that by analyzing the overall model, these submodels are analyzed, i.e. in particular with reference to the mentioned requirements the following submodels of the System- resp. User Level are analyzed in this chapter:

- **System Level**: $\mathbf{r_{mom_{broker_{8(a)}}}}, \mathbf{r_{mom_{broker_{8(b)}}}}$ → the model of the internal system behaviour of the message broker (compare to figure 7.26); $\mathbf{r_{user}}$ → the model of the internal system behaviour of the user-operation processing unit (compare to figure 7.26)
- **User Level**: $\mathbf{r_{mom_{user_6}}}$ → the model of the user operation *plugin* device (compare to figure 7.83); $\mathbf{r_{mom_{user_6}}}$ → the model of the user operation *unplug* device (compare to figure 7.84); $\mathbf{r_{user}}$ → the model of the user operation *enter appointment* (compare to figure 7.48)

By defining these submodels as $AHOI_{AHLI}$[1] subnets of the system net[2] resp. transformation rules over $AHOI_{AHLI}$ nets[3] in chapter 7 it is clarified, which behaviour of the system is represented by which submodel in what extent, so that with this knowledge in mind, the requirements mentioned above can be reformulated as follows:

- $\mathbf{r_{mom_{broker_{8(a)}}}}$ The twice enabled transition $receive\ data\ from\ device$ of subnet $ReceiveDataFromDevice_{messageBroker}$, using a different token of place $OnlineDevices$ for each token selection, can fire in parallel, under the consideration that each device is represented by one of these tokens and the procedure of receiving information from a device by the message broker is represented by firing the transition $receive\ data\ from\ device$.

[1] For the definition of $AHOI_{AHLI}$ nets see section 6.3.

[2] Compare figure 7.26 for an overview of the system net.

[3] For the definition of transformation rules over $AHOI_{AHLI}$ nets see section 6.4

- $\mathbf{r_{mom_{broker_{8(b)}}}}$ Analogous to $r_{mom_{user_6}}(a)$: The twice enabled transition $receive\ data\ from\ device$ of subnet $ReceiveDataFromDevice_{messageBroker}$, using the same token of place *OnlineDevices* for each token selection, can **not** fire in parallel.

- $\mathbf{r_{mom_{user_6}}}$ The simultaneously on the system net applicable transformation rules $PluginDevice_{user,device1}$ and $UnplugDevice_{user,device2}$ can be applied on this net in parallel, under the consideration that the change of the internal state of the system of the Living Place, which is caused by the execution of the user operation *plugin device* resp. *unplug device* regarding two different devices $device1$ and $device2$ is represented by rule $PluginDevice_{user,device1}$ resp. $UnplugDevice_{user,device2}$ and the procedure of the respective change of the internal system state is represented by the application of the corresponding rule.

- $\mathbf{r_{user}}$ The on the system net applicable transformation rule $EnterDataIntoDailyPlanner_{user,0,test}$ and the simultaneously enabled transition *enter data into device* of subnet *EnterDataIntoDevice*$_{processingUnit}$ can be applied on the system net resp. fired in parallel, under the consideration that the procedure of processing a previous user operation is represented by firing the transition *enter data into device* and the change of the internal state of the system of the Living Place, which is caused by the execution of a certain user operation *enter appointment*, where in this example an appointment of name *test* and timestamp *0* is entered into the system as a representative for all possible such user operations, is represented by the rule $EnterDataIntoDailyPlanner_{user,0,test}$, so that the procedure of this change is represented by its application.

These requirements contain the following three kinds of parallel independence, that need to be considered within the analysis of the submodels:

1. **Parallel Independence of two Direct Transformations**: Two simultaneously on the same net applicable transformation rules can be applied on this net in parallel, if the two so resulting direct transformations are parallel independet to each other.

2. **Parallel Independence of a Firing Step and a Direct Transformation**: An enabled transition and a transformation rule that is simultaneously applicable on the net which contains the enabled transition, can fire resp. be applied on this net in parallel, if the so resulting firing step and direct transformation are parallel independet to each other.

3. **Parallel Independence of two Firing Steps**: Two simultaneously enabled transitions of the same net can fire in parallel, if the two so resulting firing steps are parallel independent to each other.

First, the formal analysis techniques considering these three kinds of parallel independence are described in section 9.1 before they are used within the analysis of the considered submodels in section 9.2.

9.1 Formal Analysis Techniques

In section 9.1.1 the term *parallel independence of two direct transformations* is defined, which is used within the analysis of section 9.2.1. Section 9.1.2 introduces definitions that lead to the terms *parallel independence of a firing step and a direct transformation* as well as *parallel independence of two firing steps*, which are used within the analysis of sections 9.2.2 and 9.2.3.

9.1.1 Parallel Independence of two Direct Transformations

Two direct transformations[4] $G \overset{p_1,m_1}{\Longrightarrow} H_1$ and $G \overset{p_2,m_2}{\Longrightarrow} H_2$ are parallel independent if there exist morphisms $i : L_1 \to D_2$ and $j : L_2 \to D_1$ such that $f_2 \circ i = m_1$ and $f_1 \circ j = m_2$: [EEPT06, Definition 5.9]

$$\begin{tikzcd}
R_1 \arrow[d, "n_1"] & K_1 \arrow[l, "r_1"'] \arrow[r, "l_1"] \arrow[d, "k_1"] & L_1 \arrow[dr, "m_1"] \arrow[drrr, dashed, "i"] & & L_2 \arrow[dl, "m_2"'] \arrow[dlll, dashed, "j"'] & K_2 \arrow[l, "l_2"'] \arrow[r, "r_2"] \arrow[d, "k_2"] & R_2 \arrow[d, "n_2"] \\
H_1 & D_1 \arrow[l, "g_1"'] \arrow[rr, "f_1"] & & G & & D_2 \arrow[ll, "f_2"'] \arrow[r, "g_2"] & H_2
\end{tikzcd}$$

Two direct transformations that are not parallel independent, are called parallel dependent.

[4] Compare to definition 6.2.3 of direct AHLI transformations.

9.1.2 AHLI Transition Rules & Canonical Transformations of AHLI Nets

The following definitions lead to the terms *parallel independence of a firing step and a direct transformation* as well as *parallel independence of two firing steps* by stating in section 9.1.2.3, that every firing step can be simulated by a certain direct transformation, so that the definition of the *parallel independence of two direct transformations* of the former section 9.1.1 can be used when considering the *parallel independence of a firing step and a direct transformation* or the *parallel independence of two firing steps.*

9.1.2.1 AHLI Transition Rule

Given an AHLI net[6] $ANI = (\Sigma, P, T, pre, post, cond, type, A, I, m)$ we define the transition rule for a consistent transition assignment $(t, asg) \in CT_{ANI}$[7], enabled under the token selection $S = (M, m, N, n)$[8], as the rule
$\varrho(t, S, sag) = (L_t \overset{l}{\leftarrow} K_t \overset{r}{\rightarrow} R_t)$ with

- the common fixed AHL net part $AN_t = (\Sigma, P_t, t, pre_t, post_t, type_t, A)$, where $P_t = ENV_P(t), pre_t(t) = pre(t), post_t(t) = post(t), type_t(p) = type(p)$ with $ENV_P(t)$ is the place environment of t,
- $L_t = (AN_t, M, m_t : M \to A \otimes P_t)$, with $m_t(x) = m(x)$,
- $K_t = (AN_t, \emptyset, \emptyset : \emptyset \to (A \times P_t))$,
- $R_t = (AN_t, N, n_t : N \to A \otimes P_t)$, with $n_t(x) = n(x)$,
- l, r being the obvious inclusions on the rule nets.

[MGE+10, Definition 3.13]

9.1.2.2 Canonical Direct Transformation of AHLI Nets

A direct transformation[4] $ANI_1 \overset{\varrho,f}{\Longrightarrow} ANI_2$ by rule $\varrho = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ with l, r being inclusions is called *canonical* if

- f_I is injective
- f_Σ is injective on the set of variables of Σ_{ANI_1}
- the morphisms in the span $(ANI_1 \leftarrow ANI_0 \to ANI_2)$ of the DPO transformation diagram below are inclusions, and
- $I_2 = I_0 \cup (I_R \setminus r(I_K))$.

[MGE+10, Definition 3.14]

$$\begin{tikzcd}
L \arrow[d, "f"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] & R \arrow[d, "f^*"] \\
ANI_1 & ANI_0 \arrow[l] \arrow[r] & ANI_2
\end{tikzcd}$$

9.1.2.3 Equivalence of Canonical Direct Transformations and Firing of AHLI Nets

Theorem 3.1 "*equivalence of canonical direct transformations and firing of AHLI nets*" in [MGE+10] implies, that for each firing step $ANI \overset{t,asg}{\rightarrowtail} ANI'$[8] via token selection $S = (M, m, N, n)$[7] a canonical direct transformation $ANI \overset{\varrho(t,S,asg),f}{\Longrightarrow} ANI'$ via the transition rule $\varrho(t, S, asg)$ matched by the inclusion match $f : L \to ANI$ can be given, so that it is guaranteed that this direct transformation simulates the firing step resulting into the same net.

[5] Compare to definition 6.1.1 of AHLI Net.

[6] Compare to definition 6.1.2.1 of Consistent Transition Assignment.

[7] Compare to definition 6.1.2.2 of Token Selection.

[8] Compare to definition 6.1.2 of Firing of AHLI net.

Thus, the parallel independence of firing steps can be considered as the parallel independence of the corresponding direct transformations.

This result is useful when analysing the parallel independence of two firing steps resp. the parallel independence of a firing step and a direct transformation in section 9.2.

A similar concept to the here considered parallel independence of firing steps is given in [PPE+05] as the concept of concurrency and parallelism in P/T nets.

9.2 Revisiting Some Requirements Towards the Model

In this section the submodels considered above are analysed by using the formal analysis techniques described in section 9.1 to check whether these submodels fulfil the considered four requirements.

9.2.1 Parallel Independence of two Direct Transformations

First it is checked, whether the model of the Living Place system fulfils requirement $r_{mom_{user_6}}$. Therefor, it is analysed whether the two transformation rules $PluginDevice_{user,device1}$ and $UnplugDevice_{user,device2}$, that are simultaneously applicable on the system net as illustrated in figure 9.1, can be applied on this net in parallel by analysing whether the two so resulting direct transformations are parallel independent to each other.

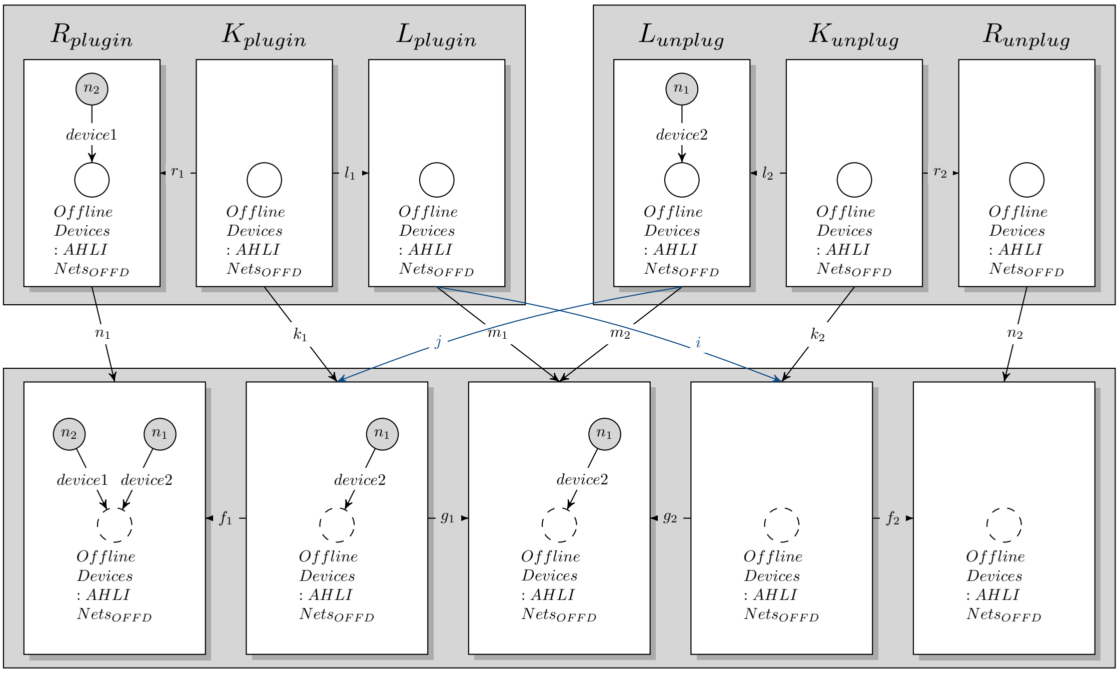


Figure 9.1: Revisiting Requirement $r_{mom_{user_6}}$

For this analysis definition 9.1.1 can be used directly. Consider, that only the necessary subnet of the system net is illustrated in figure 9.1.

Obviously there exist morphisms i and j, such that $g2 \circ i = m_1$ and $g_1 \circ j = m_2$. Thus, the considered two direct transformations are parallel independent and therefor can proceed in parallel. Therefore, the model of the Living Place system fulfils requirement $r_{mom_{user_6}}$.

9.2.2 Parallel Independence of a Firing Step and a Direct Transformation

In this section it is checked, whether the model of the Living Place system fulfils requirement r_{user}. Therefor, it is analysed whether the enabled transition *enter data into device* and the transformation rule $EnterDataIntoDailyPlanner_{user,0,test}$ that is simultaneously applicable on the net which contains the enabled transition as illustrated in figure 9.2, can fire resp. be applied on this net in parallel by analysing whether the so resulting firing step and direct transformation are parallel independent to each other.

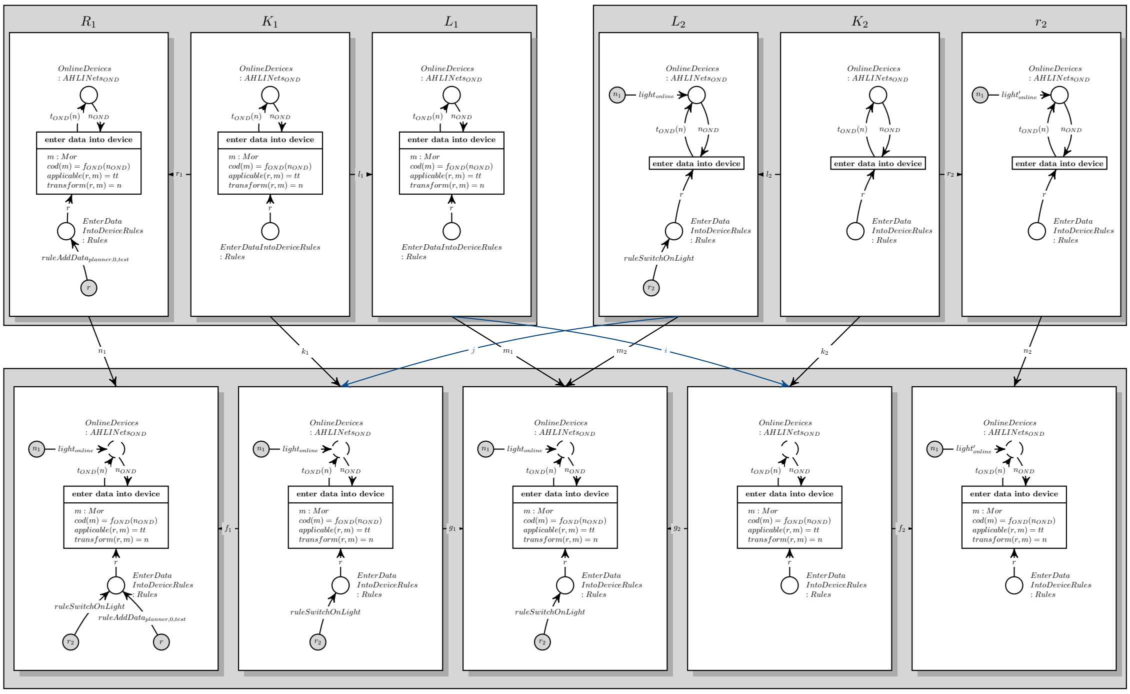


Figure 9.2: Revisiting Requirement r_{user}

For this analysis definition 9.1.1 and also the definitions of section 9.1.2 are used. Consider, that only the necessary subnet of the system net is illustrated in figure 9.2.

As an example that is considered in this analysis, the transition *enter data into device* is enabled under token selection
$(\{r_2, n_1\}, r_2 \mapsto (ruleSwithOnLight, EnterDataIntoDeviceRules); n_1 \mapsto (light_{online}, OnlineDevices), \{n_1\}, n_1 \mapsto (light'_{online}, OnlineDevices))$ and a corresponding enabled consistent transition assignment as a representative for all possibilities of processing a previous user operation, where the previous user operation is represented by the token *ruleSwitchOnLight* in this example.

By using the definitions of section 9.1.2, for the resulting firing step of transition *enter data into device* a canonical direct transformation via the considered transition rule matched by the inclusion match m_2 is given as illustrated in figure 9.2, so that the parallel independence of the resulting firing step and the considered direct transformation, that need to by analysed, can be considered as the parallel independence of the given direct transformation for the firing step and the other considered direct transformation. Therefore, for this analysis definition 9.1.1 can be used now.

Obviously there exist morphisms i and j, such that $g2 \circ i = m_1$ and $g_1 \circ j = m_2$. Thus, the considered two direct transformations are parallel independent and therefor can proceed in parallel. Therefore, the model of the Living Place system fulfils requirement r_{user}.

9.2.3 Parallel Independence of two Firing Steps

In this section it is checked, whether the model of the Living Place system fulfils requirement $r_{mom_{broker_{8(a)}}}$. Therefor, it is analysed whether the twice enabled transition *receive data from device* using a different token of place *OnlineDevices* for each token selection, can fire in parallel by analysing whether the so resulting firing steps are parallel independent to each other.

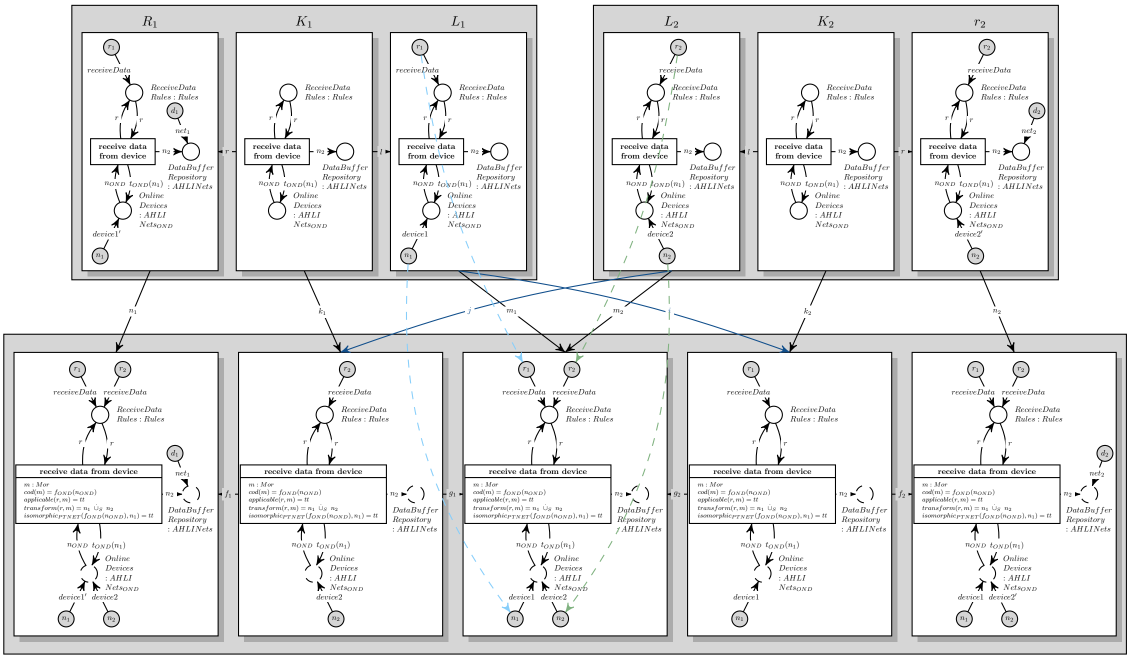


Figure 9.3: Revisiting Requirement $r_{mom_{broker_{8(a)}}}$

For this analysis definition 9.1.1 and also the definitions of section 9.1.2 are used. Consider, that only the necessary subnet of the system net is illustrated in figure 9.3.

As an example that is considered in this analysis, the transition *receive data from device* is enabled under token selection
$(\{r_1, n_1\}, r_1 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device_1, OnlineDevices),$
$\{r_1, n_1, d_1\}, r_1 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device1', OnlineDevices);$
$d_1 \mapsto (net1, DataBufferRepository))$ and a corresponding enabled consistent transition assignment as well as under token selection
$(\{r_2, n_2\}, r_2 \mapsto (receiveData, ReceiveDataRules); n_2 \mapsto (device_2, OnlineDevices),$
$\{r_2, n_2, d_2\}, r_2 \mapsto (receiveData, ReceiveDataRules); n_2 \mapsto (device2', OnlineDevices);$
$d_2 \mapsto (net2, DataBufferRepository))$ and a corresponding enabled consistent transition assignment as a representative for a twice enabled transition *receive data from device* using a different token of place *OnlineDevices* for each token selection.

By using the definitions of section 9.1.2, for the resulting firing steps of transition *receive data from device* canonical direct transformations via the considered transition rules matched by the inclusion matches m_1 resp. m_2 are given as illustrated in figure 9.3, so that the parallel independence of the resulting firing steps, that need to be analysed, can be considered as the parallel independence of the given direct transformations for these firing steps. Therefore, for this analysis definition 9.1.1 can be used now.

Obviously there exist morphisms i and j, such that $g2 \circ i = m_1$ and $g_1 \circ j = m_2$. Thus, the considered two direct transformations are parallel independent and therefor can proceed in parallel. Therefore, the model of the Living Place system fulfils requirement $r_{mom_{broker_{8(a)}}}$.

Furthermore it is checked, whether the model of the Living Place system fulfils requirement $r_{mom_{broker_{8(b)}}}$. Therefor, it is analysed whether the twice enabled transition *receive data from device* using the same token of place *OnlineDevices* for each token selection, can **not** fire in parallel by analysing whether the so resulting firing steps are parallel dependent to each other.

For this analysis definition 9.1.1 and also the definitions of section 9.1.2 are used. Consider, that only the necessary subnet of the system net is illustrated in figure 9.4.

As an example that is considered in this analysis, the transition *receive data from device* is enabled under token selection
$(\{r_1, n_1\}, r_1 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device_1, OnlineDevices),$
$\{r_1, n_1, d_1\}, r_1 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device1', OnlineDevices);$
$d_1 \mapsto (net1, DataBufferRepository))$ and a corresponding enabled consistent transition assignment as well as under token selection
$(\{r_2, n_1\}, r_2 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device_1, OnlineDevices),$
$\{r_2, n_1, d_2\}, r_2 \mapsto (receiveData, ReceiveDataRules); n_1 \mapsto (device1', OnlineDevices);$

$d_2 \mapsto (net1, DataBufferRepository))$ and a corresponding enabled consistent transition assignment as a representative for a twice enabled transition *receive data from device* using the same token of place *OnlineDevices* for each token selection.

By using the definitions of section 9.1.2, for the resulting firing steps of transition *receive data from device* canonical direct transformations via the considered transition rules matched by the inclusion matches m_1 resp. m_2 are given as illustrated in figure 9.4, so that the parallel dependence of the resulting firing steps, that need to be analysed, can be considered as the parallel dependence of the given direct transformations for these firing steps. Therefore, for this analysis definition 9.1.1 can be used now.

Obviously there exist **no** morphisms i and j, such that $g2 \circ i = m_1$ and $g_1 \circ j = m_2$. Thus, the considered two direct transformations are parallel dependent and therefor can **not** proceed in parallel. The one direct transformation deletes a token which is used by the other direct transformation and vice versa, so that both transformations are in a conflict. Therefore, the model of the Living Place system fulfils requirement $r_{mom_{broker_{8(b)}}}$.

Consider, that the firing steps of the same transition never can be parallel independent in general.

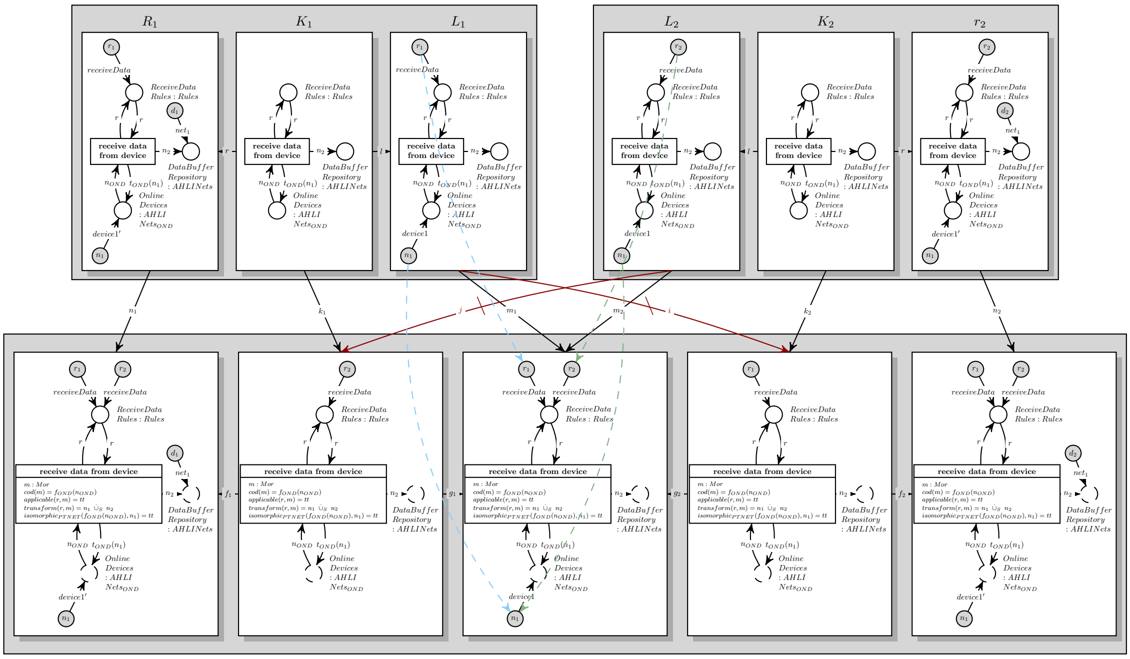


Figure 9.4: Revisiting Requirement $r_{mom_{broker_{8(b)}}}$

9.3 Summary

In this chapter four requirements of chapter 4 towards the model of the Living Place system are considered again, where each of these requirements demands, that certain behaviours can proceed in parallel resp. can not proceed in parallel in this model.

Therefor, the model of the Living Place system is formally analyzed by analyzing whether these behaviours can or can not proceed in parallel in this model using formal analysis techniques.

The used analysis techniques are described and the result of the analysis is given clarifying, that the considered requirements are met by the model.

So, by analysing the formal model of the Living Place system as it is elaborated in chapter 7 using formal analysis techniques, results can be obtained that help us to improve our understanding of this system.

In [EEPT06] a critical pair is defined as a pair of parallel dependent direct transformations $H_1 \overset{p_1,m_1}{\Longleftarrow} G \overset{p_2,m_2}{\Longrightarrow} H_2$ such that $(m_1, m_2) \in \mathcal{E}'$ for the corresponding matches m_1 and m_2 and a given $\mathcal{E}' - \mathcal{M}'$ pair factorization, i.e. (m_1, m_2) are jointly surjective when considering the used theory within this work. Although just one critical pair is considered within this chapter in figure 9.4, it is desirable to consider all critical pairs regarding to certain transformation rules resp. enabled transitions by an automated critical pair analysis to find dependencies and conflicts in general as it is supported for attributed graph transformations by the tool AGG as stated in [Tec]. Consider, that the within chapter 7 defined model of the Living Place contains an infinite set of transformation rules, so that a complete critical pair analysis regarding all these transformation rules is impossible.

Chapter 10

Summary

10.1 Conclusion

This work considers a system of ubiquitous computing an ambient intelligence, the so called Living Place system, and provides a formal model of the internal system behaviour of this system as it is elaborated in chapter 7. Furthermore, the simulation of a scenario in chapter 8 by using this model as well as the presented analysis upon this model and their results in chapter 9 show, how this model can be used to help us to improve our understanding of this system and therefore the nature of this kind of ubiquitous computing systems in general.

By modelling the system of the Living Place in chapter 7 it is clarified, that the used formal modelling techniques as they are presented in chapter 6 are powerful enough in their expressiveness to enable the adequate and elegant modelling of systems of such a complexity as it can be found in the Living Place system.

By using $AHLI$ net transformations as a modelling technique, i.e. transformations with "individual" tokens as markings, instead of approaches with "collective"markings, the marking of a net with individual tokens can be manipulated with rules, i.e. marking-changing rules can be formulated, which is not possible with the collective token approach. This possibility to manipulate the marking of a net with rules is heavily used within the presented model of the Living Place system of chapter 7 within this work, since this feature of the used formal modelling techniques contributes significantly to the expressiveness of these techniques leading into possibly elegant models.

In the following section 10.2 some aspects concerning the modelling of the Living Place system that have been discovered within this work and that are open for research in future work are discussed.

10.2 Future Work

10.2.1 Tool Support

Chapter 7 illustrates the formal model of the Living Place system predominantly in a visual form that consists of several $AHLI$ nets, signatures & algebras, $AHLI$ transformation rules as well as several other formal constructs of chapter 6. As it can be seen from these illustrations, the model is characterized by high complexity. Therefore, it requires a lot of effort to keep all these nets, rules, algebras & signatures consistent with one another. So, a tool support is desirable that enables the visual definition of $AHLI$ nets, i.e. nets with individual tokes, the visual definition of arbitrarily signatures & algebras for these nets, the visual definition of $AHOI$ nets, i.e. nets featuring nets and rules as tokens, the visual definition of $AHLI$ transformation rules, the visual definition of amalgamated rules and simple nested application conditions, the visual definition of reconfigurable $AHLI$ systems as well as the definition of all other formal constructs that are defined in chapter 6 in an appropriate and consistent manner.

Furthermore, such a tool should support the firing and therefore the simulation of such $AHLI$ resp. $AHOI_{AHLI}$[1] nets as well as the application of transformation rules of an arbitrarily reconfigurable $AHLI$ system on such a net as it is exemplarily done in the scenario of chapter 8.

Additionally, such a tool should support analysis techniques that can be automatically applied to these nets and rules, such as the critical pair analysis[2]. In figure 9.4 of chapter 9 a critical pair as part of

[1] See section 6.3 for the definition of $AHOI_{AHLI}$ nets.

[2] Compare to chapter 9

such a critical pair analysis relating to the presented model within this work is given by hand. Performing a complete critical pair analysis on the model of chapter 7 by hand seems to be impossible.

"The Attributed Graph Grammar System" tool, short AGG [Tec], already supports the critical pair analysis. "RON: An editor for Reconfigurable Object Nets" [RON] already supports the visual definition and simulation of so called Reconfigurable Object Nets, which are a simplified version of Algebraic Higher-Order Nets. Furthermore, in [Fis10] an editor for $AHLI$ nets is introduced.

It is part of future work to integrate all the mentioned aspects into one tool.

10.2.2 $\mathcal{E}' - \mathcal{M}'$-Pair Factorization

In chapter 9 it is stated, that it is desirable to be able to perform a critical pair analysis for certain parts of the model of chapter 7.

Therefor, an $\mathcal{E}' - \mathcal{M}'$-pair factorization is needed for category $AHLINets(\Sigma)$ with $\mathcal{E}'$ as the class of jointly epimorphic morphisms. See also [EEPT06, Definition 5.25] and [EEPT06, Definition 6.20] for more details on that subject.

10.2.3 Negative Application Conditions

In section 6.2.6.1 a problem concerning negative application conditions and its satisfiability regarding to a morphism is discussed. Therefor, a new concept of satisfiability regarding to a negative application condition need to be developed.

10.2.4 Distribution of Algebras

In the model of the Living Place system as it is presented in chapter 7, every $AHLI$ object net, as part of the superordinate $AHOI_{AHLI}$ net, shares the same huge signature Σ_{OL} and huge Σ_{OL}-Algebra A_{OL}.[3] It is desirable to define for every $AHLI$ object net an independent algebra and even independent signature due to better readability, since every communicating device as they are defined in chapter 3 is represented by its own $AHLI$ object net, where the signature and algebra of this $AHLI$ net defines the operations and data the corresponding communicating device operates on. However, this seems quite tricky by using the formal techniques of chapter 6, so that every such $AHLI$ object net shares this same signature and algebra.

10.2.5 Modelling of Time

Maybe it is desirable to model the cylce of time as part of the model of the Living Place system as it is presented in chapter 7, since some communicating devices, like the Alarm Clock 2.0, uses timepoints for their operations. The cycle of time is not considered by the model of the Living Place system due to the nature of the used modelling techniques that do not support such modelling of time. In the model of the Living Place system of chapter 7 the current time is represented by an individual token that is assigned to a data element representing the current time on a specific place. Therefore, the current time is part of the current state of the Living Place system but in order to update the current time, this token must be replaced by a god-like hand which can be the user that uses this model. However, this update of the current time is not supported by simulating the model resp. the net itself. Therefore, the concept of Timed Petri Nets[4] is needed to model such a behaviour.

10.2.6 On Restricting the Matches

Within this work several $AHLI$ nets and $AHLI$ transformation rules are defined in chapter 7. To restrict the matches of these rules to these nets, the places of the nets are unambiguously typed, so that a rule can only match on those places as provided.

The restriction of the matches via the typing of the places is working but not elegant, since a variety of conversion operations need to be defined in the corresponding signature & algebra of the nets that contain these places - for every additional type, one conversion operation. See signature Σ_{OL} and Σ_{OL}-Algebra A_{OL} in section 11.1 resp. 11.2 and the nets which uses this algebra in chapter 7 for more details.

It is desirable to introduce an additional formal construct, that allows the restriction of the matches in an appropriate way.

[3] See section 11.1 for the definition of signature Σ_{OL} and 11.2 for the definition of algebra A_{OL}.

[4] Refer to [Wan07] for an introduction to timed petri nets

10.2.7 Instantiation of Transformation Rules

The model of the Living Place system contains an infinite set of transformation rules, since for every possible user operation upon this system there is one transformation rule that represents the change of the internal system behaviour that is caused by this user operation.

Therefore, for example for every possible appointmenet a resident of the Living Place can enter into the Living Place system, a transformation rule is defined. So, there is a transformation rule $EnterAppointment_{user,xyz,0}$ that represents the input of appointment xyz at timepoint 0 into the system, a transformation rule $EnterAppointment_{user,abc,10}$ that represents the input of appointment abc at timepoint 10 into the system, etc., where all these transformation rules follow the same pattern.

Therefor, it is desirable to be able to define only one transformation rule pattern with input parameters - $EnterAppointment(INname : \{w*|w \in \{a,...,z\}\}, time : \mathbb{N})$ - that illustrates the main pattern, so that this rule pattern can be instantiated, i.e. all variables *name* and *time* within this pattern are substituted by the corresponding values resulting into a valid transformation rule. Therefor, a formal notation need to be developed.

10.2.8 Rule Amalgamation

The model of the Living Place system of chapter 7 includes several interaction schemes for rule amalgamation over maximal matching in the context of $AHLI$ nets. In order to be able to use this concept in its full extent in [MGE$^+$10] it is stated, that it need to be shown that the used category of AHLI nets has effective pushouts.

Chapter 11

Appendix

11.1 Signature Σ_{OL}

This section defines signature Σ_{OL} which is included in each $AHLI$ net of the object level, i.e. each $AHLI$ object net that is contained by the superordinated $AHOI_{AHLI}$ net of the system level as illustrated in figure 7.26 as a token.

Therefore, the in the following defined signature is included in each model of the internal system behaviour of a communicating device of the object level as introduced in chapter 7.

Signature Σ_{OL} is defined as follows:

$\Sigma_{\mathbf{OL}} =$

sorts : $Data, Topic, Topic_{toSubscribe}, Topic_{toUnsubscribe}, Topic_{subscribed}, Topic_{toRequest},$
$DeviceStatus, DataReceived, DataToSend, InputData, OutputData, RequestData,$
$DataForPersistenceLayer, ConnectionID, ConnectionID_{messageBroker}, Sync,$
$GlobalData, GlobalDataTmp, SensorReal, SensorData, SensorDataTuple, Bool$
$TVState, LightState, WeatherData, TrafficData, PressureData, PressureInfo,$
$CalendarData, Timestamp, PersonState, LocalData, DisplayData, BedData,$
$PressureData_{OLD}, ContextData, Action, PersonState_{single}, ContextData_{single},$
$5W1HContext, Weather, Traffic$

opns : $tt :\to Bool,$
$ff :\to Bool,$
$online :\to DeviceStatus,$
$offline :\to DeviceStatus,$
$askOnline :\to DeviceStatus,$
$askOffline :\to DeviceStatus,$
$t_{toS} : Topic \to Topic_{toSubscribe},$
$t_{toU} : Topic_{subscribed} \to Topic_{toUnsubscribe},$
$tuple : Topic \times ConnectionID \to RequestData,$
$tuple : Data \times Topic \to DataToSend,$
$triple : Data \times Topic \times ConnectionID \to OutputData,$
$triple : Data \times Topic \times ConnectionID \to GlobalData,$
$triple : Data \times Topic \times Timestamp \to LocalData,$
$triple : ContextData_{single} \times ContextData_{single} \times ContextData_{single} \to ContextData,$
$4tuple : Data \times Topic \times ConnectionID \times Action \to DataForPersistenceLayer,$
$6tuple : 5W1HContext \times 5W1HContext \times 5W1HContext \times 5W1HContext \times$
$5W1HContext \times 5W1HContext \to Data,$
$d_{rcv} : InputData \to DataReceived,$
$transformToUbiTV : SensorDataTuple \to DataToSend,$
$analyseUbiTVdata : Data \to TVState,$
$analyseLightData : DataReceived \to LightState,$
$isActiveToken : TVState \to Bool,$
$weatherToData : Timestamp \times WeatherData \to Data,$
$trafficToData : Timestamp \times TrafficData \to Data,$
$thresholdExceeded : PressureData \to PressureData,$

$$\begin{array}{l}
hasChanged : PressureData \times PressureData \rightarrow PressureInfo,\\
checkPressureChange : PressureInfo \rightarrow BedData,\\
transformToBed : BedData \rightarrow DataToSend,\\
transformToCalendar : CalendarData \rightarrow DataToSend,\\
first_{real} : SensorData \rightarrow SensorReal,\\
second_{real} : SensorData \rightarrow SensorReal,\\
third_{real} : SensorData \rightarrow SensorReal,\\
first_{dataReceived} : DataReceived \rightarrow Data,\\
second_{dataReceived} : DataReceived \rightarrow Data,\\
third_{dataReceived} : DataReceived \rightarrow Data,\\
isTrafficData : Data \rightarrow Bool,\\
isWeatherData : Data \rightarrow Bool,\\
isCalendarData : Data \rightarrow Bool,\\
isBedData : Data \rightarrow Bool,\\
isContextData : Data \rightarrow Bool,\\
isUbiTV Data : Data \rightarrow Bool,\\
isAlarmData : Data \rightarrow Bool,\\
calcAddTimeTraffic : Data \rightarrow Timestamp,\\
calcAddTimeWeather : Data \rightarrow Timestamp,\\
toCalendar : Data \rightarrow CalendarData,\\
getTimestamp_{Calendar} : CalendarData \rightarrow Timestamp,\\
getInfo_{Calendar} : CalendarData \rightarrow Data,\\
isSoon : Data \rightarrow Bool,\\
calcWakeUpTime : CalendarData \times Timestamp \rightarrow Timestamp,\\
toContext : Data \times ContextData \rightarrow ContextData,\\
isdeepSleep : Data \rightarrow Bool,\\
isLightSleep : Data \rightarrow Bool,\\
is15MinutesLater : Timestamp \times Timestamp \rightarrow Bool,\\
isBeforeNextAppointment : CalendarData \times Timestamp \rightarrow Bool,\\
isWashingInBathroom : ContextData \rightarrow Bool,\\
isEatingInKitchen : ContextData \rightarrow Bool,\\
isDressingInSleepingArea : ContextData \rightarrow Bool,\\
isNotInBed : ContextData \rightarrow Bool,\\
hasPersonLeftHome : ContextData \rightarrow Bool,\\
+_{time} : Timestamp \times Timestamp \rightarrow Timestamp,\\
-_{time} : Timestamp \times Timestamp \rightarrow Timestamp,\\
\leq_{time}: Timestamp \times Timestamp \rightarrow Bool,\\
=_{time}: Timestamp \times Timestamp \rightarrow Bool,\\
<_{time} : Timestamp \times Timestamp \rightarrow Bool,\\
removeAsleep_{ps} : PersonState \rightarrow PersonState,\\
removeUnwashed_{ps} : PersonState \rightarrow PersonState,\\
removeHungry_{ps} : PersonState \rightarrow PersonState,\\
removeUndressed_{ps} : PersonState \rightarrow PersonState,\\
removeUnwashed_{c} : ContextData \rightarrow ContextData,\\
removeHungry_{c} : ContextData \rightarrow ContextData,\\
removeUndressed_{c} : ContextData \rightarrow ContextData,\\
isReadyStates : PersonState \rightarrow Bool,\\
alarm :\rightarrow Topic,\\
traffic :\rightarrow Topic,\\
weather :\rightarrow Topic,\\
ubiTV :\rightarrow Topic,\\
calendar :\rightarrow Topic,\\
context :\rightarrow Topic,\\
bed :\rightarrow Topic,\\
on_{TV} :\rightarrow TVState,\\
off_{TV} :\rightarrow TVState,\\
active_{TV} :\rightarrow TVState,\\
inactive_{TV} :\rightarrow TVState,
\end{array}$$

$dimmed_l :\rightarrow LightState$,
$bright_l :\rightarrow LightState$,
$off_l :\rightarrow LightState$,
$unknown_c :\rightarrow ContextData_{single}$,
$undefined_c :\rightarrow 5W1HContext$,
$loungeArea_c :\rightarrow 5W1HContext$,
$sleepingArea_c :\rightarrow 5W1HContext$,
$kitchen_c :\rightarrow 5W1HContext$,
$bathroom_c :\rightarrow 5W1HContext$,
$user_c :\rightarrow 5W1HContext$,
$relaxing_c :\rightarrow 5W1HContext$,
$eating_c :\rightarrow 5W1HContext$,
$washing_c :\rightarrow 5W1HContext$,
$sleeping_c :\rightarrow 5W1HContext$,
$dressing_c :\rightarrow 5W1HContext$,
$0_c :\rightarrow ConnectionID$,
$0_t :\rightarrow Timestamp$,
$north :\rightarrow Data$,
$south :\rightarrow Data$,
$middle :\rightarrow Data$,
$bathroom :\rightarrow Data$,
$nothing :\rightarrow Data$,
$awake :\rightarrow BedData$,
$lightSleep :\rightarrow BedData$,
$deepSleep :\rightarrow BedData$,
$notInBed :\rightarrow BedData$,
$unknown :\rightarrow BedData$,
$undefined_p :\rightarrow PersonState_{single}$,
$on_a :\rightarrow Data$,
$f_P : PressureData_{OLD} \rightarrow PressureData$,
$t_P : PressureData \rightarrow PressureData_{OLD}$,
$first_t : Data \rightarrow Timestamp$,
$second_t : Data \rightarrow Data$,
$f_{display} : DisplayData \rightarrow Data$,
$t_{display} : Data \rightarrow DisplayData$,
$f_{bed} : BedData \rightarrow Data$,
$t_{bed} : Data \rightarrow BedData$,
$t_{display} : Data \rightarrow DisplayData$,
$persontasks :\rightarrow PersonState$,
$askOnline :\rightarrow DeviceStatus$,
$askOffline :\rightarrow DeviceStatus$,
$online :\rightarrow DeviceStatus$,
$offline :\rightarrow DeviceStatus$,
$sync :\rightarrow Sync$,
$remove :\rightarrow Action$,
$store :\rightarrow Action$,
$succ : ConnectionID \rightarrow ConnectionID$,
$t_{gdata} : RequestData \rightarrow GlobalDataTmp$,
$topic : GlobalData \rightarrow Topic$,
$t_{gldata} : OutputData \rightarrow GlobalData$,
$t_{indata} : GlobalData \rightarrow InputData$

vars : $t : Topic$,
$t_{sub} : Topic_{subscribed}$,
$t_{trq} : Topic_{toRequest}$,
$d_{send} : DataToSend$,
$d_{out} : OutputData$,
$d_{in} : InputData$,
$d_r : DataReceived$,

$$\begin{aligned}
&id, i : ConnectionID,\\
&s : Sync,\\
&x, y, z : SensorReal,\\
&sensordata : SensorDataTuple,\\
&r_1, r_2, r_3, r_4 : TVState,\\
&l_1, l_2 : LightState,\\
&w : WeatherData,\\
&tr : TrafficData,\\
&b : Bool,\\
&p, p'' : PressureData, p' : PressureData_{OLD}\\
&p_1, p_2 : PressureInfo,\\
&d : Data,\\
&d_{display} : DisplayData,\\
&cal, cal' : CalendarData,\\
&time, time', time_1, currenttime, ts_1, ts_1', ts_2 : Timestamp,\\
&bed, bed' : Data,\\
&c, c' : ContextData,\\
&alarm_1, alarm_2 : Data,\\
&ps, ps' : PersonState,\\
&g : GlobalData,\\
&d_{req} : RequestData
\end{aligned}$$

Table 11.1: Signature Σ_{OL}

11.2 Σ_{OL}-Algebra A_{OL}

This section defines the Σ_{OL}-Algebra A_{OL} which is included in each $AHLI$ net of the object level, i.e. each $AHLI$ object net that is contained by the superordinated $AHOI_{AHLI}$ net of the system level as illustrated in figure 7.26 as a token.

Therefore, the in the following defined algebra is included in each model of the internal system behaviour of a communicating device of the object level as introduced in chapter 7.

The Σ_{OL}-Algebra A_{OL} is defined as follows:

$\Sigma_{\mathbf{OL}}$-Algebra $\mathbf{A_{OL}}$ =

$$\begin{aligned}
\textbf{sorts}: \quad & A_{OL_{Data}} = \{w^* | w \in \{a, .., z, A, ..., z, 0, ..., 9, <, >\} \cup \{, \}\},\\
& A_{OL_{Topic}} = \{w^* | w \in \{a, .., z, A, ..., z, 0, ..., 9\}\},\\
& A_{OL_{Topic_{toSubscribe}}} = A_{OL_{Topic}},\\
& A_{OL_{Topic_{toUnsubscribe}}} = A_{OL_{Topic}},\\
& A_{OL_{Topic_{subscribed}}} = A_{OL_{Topic}},\\
& A_{OL_{Topic_{toRequest}}} = A_{OL_{Topic}},\\
& A_{OL_{DeviceStatus}} = \{online, offline, askOnline, askOffline\},\\
& A_{OL_{DataReceived}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}},\\
& A_{OL_{DataToSend}} = A_{OL_{Data}} \times A_{OL_{Topic}},\\
& A_{OL_{InputData}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}},\\
& A_{OL_{OutputData}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}},\\
& A_{OL_{RequestData}} = A_{OL_{Topic}} \times A_{OL_{ConnectionID}},\\
& A_{OL_{DataForPersistenceLayer}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}} \times A_{OL_{Action}},\\
& A_{OL_{ConnectionID}} = \mathbb{N},\\
& A_{OL_{ConnectionID_{messageBroker}}} = A_{OL_{ConnectionID}},\\
& A_{OL_{Sync}} = \{sync\},\\
& A_{OL_{GlobalData}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}},\\
& A_{OL_{GlobalDataTmp}} = A_{OL_{RequestData}},\\
& A_{OL_{SensorReal}} = \{x | x \in \mathbb{R} \wedge x \in [0.0, ..., 15.8]\},\\
& A_{OL_{SensorData}} = A_{OL_{SensorReal}} \times A_{OL_{SensorReal}} \times A_{OL_{SensorReal}},\\
& A_{OL_{SensorDataTuple}} = A_{OL_{SensorData}} \times A_{OL_{SensorData}} \times A_{OL_{SensorData}} \times\\
& \qquad A_{OL_{SensorData}} \times A_{OL_{SensorData}} \times A_{OL_{SensorData}},
\end{aligned}$$

$A_{OL_{Bool}} = \{true, false\},$
$A_{OL_{TVState}} = \{on, off, active, inactive\},$
$A_{OL_{LightState}} = \{bright, dimmed, off_A\},$
$A_{OL_{Weather}} = \{sun, rain, storm, snow, wind, hail\},$
$A_{OL_{Traffic}} = \{jam, accident, free, roadwork\},$
$A_{OL_{WeatherData}} = A_{OL_{Weather}} \times A_{OL_{Timestamp}},$
$A_{OL_{TrafficData}} = A_{OL_{Traffic}} \times A_{OL_{Timestamp}},$
$A_{OL_{PressureData}} = \{(x_1, x_2, x_3, x_4, x_5, x_6)|x_i \in \mathbb{R} \wedge x_i \in [0.0, ..., 10.0], i \in \{1, 2, 3, 4, 5, 6\}\},$
$A_{OL_{PressureInfo}} = A_{OL_{Bool}} \times A_{OL_{PressureData}},$
$A_{OL_{CalendarData}} = A_{OL_{Timestamp}} \times A_{OL_{Data}},$
$A_{OL_{Timestamp}} = \mathbb{N},$
$A_{OL_{PersonState_{single}}} = \{asleep, hungry, unwashed, undressed, undefined\},$
$A_{OL_{PersonState}} = A_{OL_{PersonState_{single}}} \times A_{OL_{PersonState_{single}}} \times$
$A_{OL_{PersonState_{single}}} \times A_{OL_{PersonState_{single}}},$
$A_{OL_{LocalData}} = A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{Timestamp}},$
$A_{OL_{DisplayData}} = A_{OL_{Data}},$
$A_{OL_{BedData}} = \{deepSleep, lightSleep, awake, notInBed, unknown\},$
$A_{OL_{PressureData_{OLD}}} = A_{OL_{PressureData}},$
$A_{OL_{ContextData_{single}}} = \{unknown, washing, eating, dressing\},$
$A_{OL_{ContextData}} = A_{OL_{ContextData_{single}}} \times A_{OL_{ContextData_{single}}} \times A_{OL_{ContextData_{single}}},$
$A_{OL_{Action}} = \{store, remove\},$
$A_{OL_{5W1HContext}} = \{undefined, user, relaxing, eating, washing, sleeping, dressing,$
$loungeArea, sleepingArea, kitchen, bathroom, outsideApartment\}$

opns : $tt_{A_{OL}} :\to A_{OL_{Bool}}$ *with*
$tt_{A_{OL}} : true,$
$ff_{A_{OL}} :\to A_{OL_{Bool}}$ *with*
$ff_{A_{OL}} : false,$
$online_{A_{OL}} :\to A_{OL_{DeviceStatus}}$ *with*
$online_{A_{OL}} = online,$
$offline_{A_{OL}} :\to A_{OL_{DeviceStatus}}$ *with*
$offline_{A_{OL}} = offline,$
$askOnline_{A_{OL}} :\to A_{OL_{DeviceStatus}}$ *with*
$askOnline_{A_{OL}} = askOnline,$
$askOffline_{A_{OL}} :\to A_{OL_{DeviceStatus}}$ *with*
$askOffline_{A_{OL}} = askOffline,$
$t_{toS_{A_{OL}}} : A_{OL_{Topic}} \to A_{OL_{Topic_{toSubscribe}}}$ *with*
$t_{toS_{A_{OL}}}(x) = x,$
$t_{toU_{A_{OL}}} : A_{OL_{Topic_{subscribed}}} \to A_{OL_{Topic_{toUnsubscribe}}}$ *with*
$t_{toU_{A_{OL}}}(x) = x,$
$tuple_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Topic}} \to A_{OL_{DataToSend}}$ *with*
$tuple_{A_{OL}}(x, y) = (x, y),$
$tuple_{A_{OL}} : A_{OL_{Topic}} \times A_{OL_{ConnectionID}} \to A_{OL_{RequestData}}$ *with*
$tuple_{A_{OL}}(x, y) = (x, y),$
$triple_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}} \to$
$A_{OL_{OutputData}}$ *with*
$triple_{A_{OL}}(x, y, z) = (x, y, z),$
$triple_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}} \to A_{OL_{GlobalData}}$ *with*
$triple_{A_{OL}}(x, y, z) = (x, y, z),$
$triple_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{Timestamp}} \to A_{OL_{LocalData}}$ *with*
$triple_{A_{OL}}(x, y, z) = (x, y, z),$
$triple_{A_{OL}} : A_{OL_{ContextData_{single}}} \times A_{OL_{ContextData_{single}}} \times A_{OL_{ContextData_{single}}} \to$
$A_{OL_{ContextData}}$ *with*
$triple_{A_{OL}}(x, y, z) = (x, y, z),$
$4tuple_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Topic}} \times A_{OL_{ConnectionID}} \times A_{OL_{Action}} \to$
$A_{OL_{DataForPersistanceLayer}}$ *with*
$4tuple_{A_{OL}}(x_1, x_2, x_3, x_4) = (x_1, x_2, x_3, x_4),$
$6tuple_{A_{OL}} : A_{OL_{5W1HContext}} \times A_{OL_{5W1HContext}} \times A_{OL_{5W1HContext}} \times A_{OL_{5W1HContext}} \times$

$$A_{OL_{5W1HContext}} \times A_{OL_{5W1HContext}} \to A_{OL_{Data}} \; with$$

$$6tuple_{A_{OL}}(x_1, x_2, x_3, x_4, x_5, x_6) =< x_1, x_2, x_3, x_4, x_5, x_6 >,$$

$$d_{rcv_{A_{OL}}} : A_{OL_{InputData}} \to A_{OL_{DataReceived}} \; with$$

$$d_{rcv_{A_{OL}}}(x) = x,$$

$$transformToUbiTV_{A_{OL}} : A_{OL_{SensorDataTuple}} \to A_{OL_{DataToSend}} \; with$$

$$transformToUbiTV_{A_{OL}}((x_1, x_2, x_3, x_4, x_5, x_6) =$$

$$\begin{cases} (south, UbiTV) & \text{if } (0.0 \leq second_{real_{A_{OL}}}(x_1) < 5.5) \\ & \vee(0.0 \leq second_{real_{A_{OL}}}(x_2) < 5.5) \\ (north, UbiTV) & \text{if } (10.0 \leq second_{real_{A_{OL}}}(x_5) \leq 15.8) \\ & \vee(10.0 \leq second_{real_{A_{OL}}}(x_6) \leq 15.8) \\ (middle, UbiTV) & \text{if } ((5.5 \leq second_{real_{A_{OL}}}(x_3) < 10.0) \\ & \wedge(first_{real_{A_{OL}}}(x_3) \leq 8.0)) \\ & \vee((5.5 \leq second_{real_{A_{OL}}}(x_4) < 10.0) \\ & \wedge(first_{real_{A_{OL}}}(x_4) \leq 8.0)) \\ (bathroom, UbiTV) & \text{if } ((5.5 \leq second_{real_{A_{OL}}}(x_3) < 10.0) \\ & \wedge(first_{real_{A_{OL}}}(x_3) > 8.0)) \\ & \vee((5.5 \leq second_{real_{A_{OL}}}(x_4) < 10.0) \\ & \wedge(first_{real_{A_{OL}}}(x_4) > 8.0)) \\ (nothing, UbiTV) & \text{else} \end{cases},$$

$$analyseUbiTVdata_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{TVState}} \; with$$

$$analyseUbiTVdata_{A_{OL}}(x) = \begin{cases} (active, inactive) & \text{if } x = north \\ (inactive, active) & \text{if } x = south \\ (inactive, inactive) & \text{if } x = middle \\ (inactive, inactive) & \text{else} \end{cases},$$

$$analyseLightData_{A_{OL}} : A_{OL_{DataReceived}} \to A_{OL_{LightState}} \; with$$

$$analyseLightData_{A_{OL}}(x) = \begin{cases} dimmed & \text{if } second_{dataReceived_{A_{OL}}}(x) = \\ & Alarm \\ first_{dataReceived_{A_{OL}}}(x) & \text{else} \end{cases},$$

$$isActiveToken_{A_{OL}} : A_{OL_{TVState}} \to A_{OL_{Bool}} \; with$$

$$isActiveToken_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = active \wedge x = inactive \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$$weatherToData_{A_{OL}} : A_{OL_{WeatherData}} \times A_{OL_{Timestamp}} \to A_{OL_{Data}} \; with$$

$$weatherToData_{A_{OL}}(x, y) =< x, y >,$$

$$trafficToData_{A_{OL}} : A_{OL_{TrafficData}} \times A_{OL_{Timestamp}} \to A_{OL_{Data}} \; with$$

$$trafficToData_{A_{OL}}(x, y) =< x, y >,$$

$$thresholdExceeded_{A_{OL}} : A_{OL_{PressureData}} \to A_{OL_{PressureData}} \; with$$

$$thresholdExceeded_{A_{OL}}((x_1, x_2, x_3, x_4, x_5, x_6)),$$

$$= \begin{cases} (x_1', x_2', x_3', x_4', x_5', x_6') \; with & x_i' = 0 \quad \text{if } x_i < 8.0 \\ & x_i' = x_i \quad \text{else} \\ with \; i \in \{1, 2, 3, 4, 5, 6\} & \end{cases},$$

$$hasChanged_{A_{OL}} : A_{OL_{PressureData}} \times A_{OL_{PressureData}} \to A_{OL_{PressureInfo}}$$

$$hasChanged_{A_{OL}}(x_1, x_2) = \begin{cases} (ff_{A_{OL}}, x_2) & \text{if } x_1 = x_2 \\ (tt_{A_{OL}}, x_2) & \text{else} \end{cases},$$

$$checkPressureChange_{A_{OL}} : A_{OL_{PressureInfo}} \to A_{OL_{BedData}} \; with$$

$$checkPressureChange_{A_{OL}}((b, (x_1, x_2, x_3, x_4, x_5, x_6)))$$

$$= \begin{cases} awake & \text{if } x_1 \neq 0 \vee x_2 \neq 0 \\ deepSleep & \text{if } x_3 \neq 0 \wedge b = ff) \\ & \wedge x_5 \neq 0 \wedge b = ff) \\ deepSleep & \text{if } x_4 \neq 0 \wedge b = ff) \\ & \wedge x_6 \neq 0 \wedge b = ff) \\ lightSleep & \text{if } x_3 \neq 0 \wedge b = tt) \\ & \wedge x_5 \neq 0 \wedge b = tt) \\ lightSleep & \text{if } x_4 \neq 0 \wedge b = tt) \\ & \wedge x_6 \neq 0 \wedge b = tt) \\ notInBed & \text{if } x_1 = 0 \wedge x_2 = 0 \\ & \text{if } x_3 = 0 \wedge x_4 = 0 \\ & \text{if } x_5 = 0 \wedge x_6 = 0 \end{cases},$$

$$transformToBed_{A_{OL}} : A_{OL_{BedData}} \to A_{OL_{DataToSend}}\ with$$
$$transformToBed_{A_{OL}}(x) = (x, Bed),$$
$$transformToCalendar_{A_{OL}} : A_{OL_{CalendarData}} \to A_{OL_{DataToSend}}\ with$$
$$transformToCalendar_{A_{OL}}(x,y) = (< x, y >, Calendar),$$
$$first_{real_{A_{OL}}} : A_{OL_{SensorData}} \to A_{OL_{SensorReal}}\ with$$
$$first_{real_{A_{OL}}}((x,y,z)) = x,$$
$$second_{real_{A_{OL}}} : A_{OL_{SensorData}} \to A_{OL_{SensorReal}}\ with$$
$$second_{real_{A_{OL}}}((x,y,z)) = y,$$
$$third_{real_{A_{OL}}} : A_{OL_{SensorData}} \to A_{OL_{SensorReal}}\ with$$
$$third_{real_{A_{OL}}}((x,y,z)) = z,$$
$$first_{dataReceived_{A_{OL}}} : A_{OL_{DataReceived}} \to A_{OL_{Data}}\ with$$
$$first_{dataReceived_{A_{OL}}}((x,y,z)) = x,$$
$$second_{dataReceived_{A_{OL}}} : A_{OL_{DataReceived}} \to A_{OL_{Topic}}\ with$$
$$second_{dataReceived_{A_{OL}}}((x,y,z)) = y,$$
$$third_{dataReceived_{A_{OL}}} : A_{OL_{DataReceived}} \to A_{OL_{ConnectionID}}\ with$$
$$third_{dataReceived_{A_{OL}}}((x,y,z)) = z,$$
$$isTrafficData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isTrafficData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Traffic \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isWeatherData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isWeatherData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Weather \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isCalendarData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}} with$$
$$isCalendarData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Calendar \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isBedData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isBedData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Bed \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isContextData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isContextData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Context \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isUbiTVData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isUbiTVData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = UbiTV \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$isAlarmData_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}\ with$$
$$isAlarmData_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = Alarm \\ ff_{A_{OL}} & \text{else} \end{cases},$$
$$calcAddTimeTraffic_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Time}}\ with$$
$$calcAddTimeTraffic_{A_{OL}}(x,y,z) = \begin{cases} y+30 & \text{if } \mathit{second}_t(x) = jam \wedge \\ & (\mathit{getTimestamp}_{Calendar_{A_{OL}}}(z) \\ & -(30+y) \leq first_t(x) \\ & \wedge first_t(x) \leq \\ & \mathit{getTimestamp}_{Calendar_{A_{OL}}}(z)) \\ y+30 & \text{if } \mathit{second}_t(x) = \mathit{accident} \wedge \\ & (\mathit{getTimestamp}_{Calendar_{A_{OL}}}(z) \\ & -(30+y) \leq first_t(x) \\ & \wedge first_t(x) \leq \\ & \mathit{getTimestamp}_{Calendar_{A_{OL}}}(z)) \\ y+15 & \text{if } \mathit{second}_t(x) = \mathit{roadwork} \wedge \\ & (\mathit{getTimestamp}_{Calendar_{A_{OL}}}(z) \\ & -(30+y) \leq first_t(x) \\ & \wedge first_t(x) \leq \\ & \mathit{getTimestamp}_{Calendar_{A_{OL}}}(z)) \\ y & \text{else} \end{cases},$$
$$calcAddTimeWeather_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Time}}\ with$$

$$
calcAddTimeWeather_{A_{OL}}(x,y,z) = \begin{cases}
y+10 & \begin{array}{l}\text{if } x = rain \wedge \\ (getTimestamp_{Calendar_{A_{OL}}}(z) \\ -(30+y) \leq first_t(x) \\ \wedge first_t(x) \leq \\ getTimestamp_{Calendar_{A_{OL}}}(z))\end{array} \\
y+15 & \begin{array}{l}\text{if } x = storm \wedge \\ (getTimestamp_{Calendar_{A_{OL}}}(z) \\ -(30+y) \leq first_t(x) \\ \wedge first_t(x) \leq \\ getTimestamp_{Calendar_{A_{OL}}}(z))\end{array} \\
y+30 & \begin{array}{l}\text{if } x = snow \wedge \\ (getTimestamp_{Calendar_{A_{OL}}}(z) \\ -(30+y) \leq first_t(x) \\ \wedge first_t(x) \leq \\ getTimestamp_{Calendar_{A_{OL}}}(z))\end{array} \\
y+15 & \begin{array}{l}\text{if } x = hail \wedge \\ (getTimestamp_{Calendar_{A_{OL}}}(z) \\ -(30+y) \leq first_t(x) \\ \wedge first_t(x) \leq \\ getTimestamp_{Calendar_{A_{OL}}}(z))\end{array} \\
y & \text{else}
\end{cases},
$$

$toCalendar_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{CalendarData}}$ *with*
$toCalendar_{A_{OL}}(x) = x,$
$getTimestamp_{Calendar_{A_{OL}}} : A_{OL_{CalendarData}} \to A_{OL_{Timestamp}}$ *with*
$getTimestamp_{Calendar_{A_{OL}}}((x,y)) = x,$
$getInfo_{Calendar_{A_{OL}}} : A_{OL_{CalendarData}} \to A_{OL_{Data}}$ *with*
$getInfo_{Calendar_{A_{OL}}}((x,y)) = y,$
$isSoon_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{Timestamp}} \to A_{Bool}$ *with*
$isSoon_{A_{OL}}(x) =$

$$
\begin{cases}
tt_{A_{OL}} & \begin{array}{l}\text{if } getTimestamp_{CalendarData_{A_{OL}}}(toCalendar(x)) \\ \quad \leq +_{time}(getTimestamp_{Timestamp_{A_{OL}}}(y), 240)\end{array} \\
ff_{A_{OL}} & \text{else}
\end{cases},
$$

$calcWakeUpTime_{A_{OL}} : A_{OL_{CalendarData}} \times A_{OL_{Timestamp}} \to A_{OL_{Timestamp}}$ *with*
$calcWakeUpTime_{A_{OL}}(x,y) = getTimestamp_{Calendar_{A_{OL}}}(x) - (120+y),$
$toContext_{A_{OL}} : A_{OL_{Data}} \times A_{OL_{ContextData}} \to A_{OL_{ContextData}}$ *with*
$toContext_{A_{OL}}(x,(c_1,c_2,c_3)) =$

$$
\begin{cases}
\begin{array}{l}(washing, c_2, c_3) \\ \quad \text{if } first_{dataReceived_{A_{OL}}}(x) =< user, bathroom, undefined, \\ \qquad washing, undefined, undefined > \\
(c_1, eating, c_3) \\ \quad \text{if } first_{dataReceived_{A_{OL}}}(x) =< user, kitchen, undefined, \\ \qquad eating, undefined, undefined > \\
(c_1, c_2, dressing) \\ \quad \text{if } first_{dataReceived_{A_{OL}}}(x) =< user, sleepingArea, undefined, \\ \qquad dressing, undefined, undefined > \\
(c_1, c_2, c_3) \\ \quad \text{else}\end{array}
\end{cases},
$$

$isdeepSleep_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}$ *with*

$$
isdeepSleep_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = deepSleep \\ ff_{A_{OL}} & \text{else} \end{cases},
$$

$isLightSleep_{A_{OL}} : A_{OL_{Data}} \to A_{OL_{Bool}}$ *with*

$$
isLightSleep_{A_{OL}}(x) = \begin{cases} tt_{A_{OL}} & \text{if } x = lightSleep \\ ff_{A_{OL}} & \text{else} \end{cases},
$$

$is15MinutesLater_{A_{OL}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \to A_{OL_{Bool}}$ *with*

$$
is15MinutesLater_{A_{OL}}(x,y) = \begin{cases} tt_{A_{OL}} & \text{if } x \leq_{time} y \\ ff_{A_{OL}} & \text{else} \end{cases},
$$

$isBeforeNextAppointment_{A_{OL}} : A_{OL_{CalendarData}} \times A_{OL_{Timestamp}} \to A_{OL_{Bool}}$ *with*

$$isBeforeNextAppointment_{A_{OL}}(x,y) = \begin{cases} tt_{A_{OL}} & \text{if } y \leq_{time} \\ & getTimestamp_{Calendar_{A_{OL}}}(x) \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$isWashingInBathroom_{A_{OL}} : A_{OL_{ContextData}} \rightarrow A_{OL_{Bool}}$ *with*

$$isWashingInBathroom_{A_{OL}}((c_1,c_2,c_3)) = \begin{cases} tt_{A_{OL}} & \text{if } c_1 = washing \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$isEatingInKitchen_{A_{OL}} : A_{OL_{ContextData}} \rightarrow A_{OL_{Bool}}$ *with*

$$isEatingInKitchen_{A_{OL}}((c_1,c_2,c_3)) \begin{cases} tt_{A_{OL}} & \text{if } c_2 = eating \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$isDressingInSleepingArea_{A_{OL}} : A_{OL_{ContextData}} \rightarrow A_{OL_{Bool}}$ *with*

$$isDressingInSleepingArea_{A_{OL}}((c_1,c_2,c_3)) \begin{cases} tt_{A_{OL}} & \text{if } c_3 = dressing \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$hasPersonLeftHome_{A_{OL}} : A_{OL_{ContextData}} \rightarrow A_{OL_{Bool}}$ *with*

$$hasPersonLeftHome_{A_{OL}}(x) \begin{cases} tt_{A_{OL}} & \text{if } x =< user, outsideApartment, undefined, \\ & undefined, undefined, undefined > \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$+_{time_{A_{OL}}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \rightarrow A_{OL_{Timestamp}}$ *with*

$+_{time_{A_{OL}}}(x,y) = x + y,$

$-_{time_{A_{OL}}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \rightarrow A_{OL_{Timestamp}}$ *with*

$-_{time_{A_{OL}}}(x,y) = x - y,$

$\leq_{time_{A_{OL}}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \rightarrow A_{OL_{Bool}}$ *with*

$$\leq_{time_{A_{OL}}} (x,y) = \begin{cases} tt_{A_{OL}} & \text{if } x \leq y \\ ff_{A_{OL}} & \text{else} \end{cases}$$

$<_{time_{A_{OL}}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \rightarrow A_{OL_{Bool}}$ *with*

$$<_{time_{A_{OL}}} (x,y) = \begin{cases} tt_{A_{OL}} & \text{if } x<y \\ ff_{A_{OL}} & \text{else} \end{cases}$$

$=_{time_{A_{OL}}} : A_{OL_{Timestamp}} \times A_{OL_{Timestamp}} \rightarrow A_{OL_{Bool}}$ *with*

$$=_{time_{A_{OL}}} (x,y) = \begin{cases} tt_{A_{OL}} & \text{if } x = y \\ ff_{A_{OL}} & \text{else} \end{cases},$$

$alarm_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$alarm_{A_{OL}} = Alarm,$

$traffic_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$traffic_{A_{OL}} = Traffic,$

$weather_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$weather_{A_{OL}} = Weather,$

$ubiTV_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$ubiTV_{A_{OL}} = UbiTV,$

$calendar_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$calendar_{A_{OL}} = Calendar,$

$context_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$context_{A_{OL}} = Context,$

$bed_{A_{OL}} :\rightarrow A_{OL_{Topic}}$ *with*

$bed_{A_{OL}} = Bed,$

$on_{TV_{A_{OL}}} :\rightarrow A_{OL_{TVState}}$ *with*

$on_{TV_{A_{OL}}} = on,$

$off_{TV_{A_{OL}}} :\rightarrow A_{OL_{TVState}}$ *with*

$off_{TV_{A_{OL}}} = off,$

$active_{TV_{A_{OL}}} :\rightarrow A_{OL_{TVState}}$ *with*

$active_{TV_{A_{OL}}} = active,$

$inactive_{TV_{A_{OL}}} :\rightarrow A_{OL_{TVState}}$ *with*

$inactive_{TV_{A_{OL}}} = inactive,$

$dimmed_{l_{A_{OL}}} :\rightarrow A_{OL_{LightState}}$ *with*

$dimmed_{l_{A_{OL}}} = dimmed,$

$$\begin{array}{l}
bright_{l_{A_{OL}}} :\rightarrow A_{OL_{LightState}} \ with \\
bright_{l_{A_{OL}}} = bright, \\
off_{l_{A_{OL}}} :\rightarrow A_{OL_{LightState}} \ with \\
off_{l_{A_{OL}}} = off, \\
unknown_{c_{A_{OL}}} :\rightarrow A_{OL_{ContextData_{single}}} \ with \\
unknown_{c_{A_{OL}}} = unknown, \\
undefined_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
undefined_{c_{A_{OL}}} = undefined, \\
loungeArea_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
loungeArea_{c_{A_{OL}}} = loungeArea, \\
sleepingArea_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
sleepingArea_{c_{A_{OL}}} = sleepingArea, \\
kitchen_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
kitchen_{c_{A_{OL}}} = kitchen, \\
bathroom_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
bathroom_{c_{A_{OL}}} = bathroom, \\
user_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
user_{c_{A_{OL}}} = user, \\
relaxing_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
relaxing_{c_{A_{OL}}} = relaxing, \\
eating_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
eating_{c_{A_{OL}}} = eating, \\
washing_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
washing_{c_{A_{OL}}} = washing, \\
sleeping_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
sleeping_{c_{A_{OL}}} = sleeping, \\
dressing_{c_{A_{OL}}} :\rightarrow A_{OL_{5W1HContext}} \ with \\
dressing_{c_{A_{OL}}} = dressing, \\
0_{c_{A_{OL}}} :\rightarrow A_{OL_{ConnectionID}} \ with \\
0_{c_{A_{OL}}} = 0, \\
0_{t_{A_{OL}}} :\rightarrow A_{OL_{Timestamp}} \ with \\
0_{t_{A_{OL}}} = 0, \\
north_{A_{OL}} :\rightarrow A_{OL_{Data}} \ with \\
north_{A_{OL}} = north, \\
south_{A_{OL}} :\rightarrow A_{OL_{Data}} \ with \\
south_{A_{OL}} = south, \\
middle_{A_{OL}} :\rightarrow A_{OL_{Data}} \ with \\
middle_{A_{OL}} = middle, \\
bathroom_{A_{OL}} :\rightarrow A_{OL_{Data}} \ with \\
bathroom_{A_{OL}} = bathroom, \\
nothing_{A_{OL}} :\rightarrow A_{OL_{Data}} \ with \\
nothing_{A_{OL}} = nothing, \\
awake_{A_{OL}} :\rightarrow A_{OL_{BedData}} \ with \\
awake_{A_{OL}} = awake, \\
lightSleep_{A_{OL}} :\rightarrow A_{OL_{BedData}} \ with \\
lightSleep_{A_{OL}} = lightSleep, \\
deepSleep_{A_{OL}} :\rightarrow A_{OL_{BedData}} \ with \\
deepSleep_{A_{OL}} = deepSleep, \\
notInBed_{A_{OL}} :\rightarrow A_{OL_{BedData}} \ with \\
notInBed_{A_{OL}} = notInBed, \\
unknown_{A_{OL}} :\rightarrow A_{OL_{BedData}} \ with \\
unknown_{A_{OL}} = unknown, \\
undefined_{p_{A_{OL}}} :\rightarrow A_{OL_{PersonState_{single}}} \ with \\
undefined_{p_{A_{OL}}} = undefined, \\
on_{a_{A_{OL}}} :\rightarrow A_{OL_{Data}} \ with \\
on_{a_{A_{OL}}} = on,
\end{array}$$

$$\begin{array}{l}
f_{P_{A_{OL}}} : A_{OL_{PressureData_{OLD}}} \to A_{OL_{PressureData}} \ with \\
f_{P_{A_{OL}}}(x) = x, \\
t_{P_{A_{OL}}} : A_{OL_{PressureData}} \to A_{OL_{PressureData_{OLD}}} \ with \\
t_{P_{A_{OL}}}(x) = x, \\
first_{t_{A_{OL}}} : A_{OL_{Data}} \to A_{OL_{Timestamp}} \ with \\
first_{t_{A_{OL}}}(< a, b >) = b, \\
second_{t_{A_{OL}}} : A_{OL_{Data}} \to A_{OL_{Data}} \ with \\
second_{t_{A_{OL}}}(< a, b >) = a, \\
f_{display_{A_{OL}}} : A_{OL_{DisplayData}} \to A_{OL_{Data}} \ with \\
f_{display_{A_{OL}}}(x) = x, \\
t_{display_{A_{OL}}} : A_{OL_{Data}} \to A_{OL_{DisplayData}} \ with \\
t_{display_{A_{OL}}}(x) = x, \\
f_{bed_{A_{OL}}} : A_{OL_{BedData}} \to A_{OL_{Data}} \ with \\
f_{bed_{A_{OL}}}(x) = x, \\
t_{bed_{A_{OL}}} : A_{OL_{Data}} \to A_{OL_{BedData}} \ with \\
t_{bed_{A_{OL}}}(x) = x, \\
t_{display_{A_{OL}}} : A_{OL_{Data}} \to A_{OL_{DisplayData}} \ with \\
t_{display_{A_{OL}}}(x) = x, \\
persontasks_{A_{OL}} :\to A_{OL_{PersonState}} \ with \\
persontasks_{A_{OL}} = (asleep, unwashed, undressed, hungry), \\
removeAsleep_{ps_{A_{OL}}} : A_{OL_{PersonState}} \to A_{OL_{PersonState}} \ with \\
removeAsleep_{ps_{A_{OL}}}((p_1, p_2, p_3, p_4)) = (undefined, p_2, p_3, p_4), \\
removeUnwashed_{ps_{A_{OL}}} : A_{OL_{PersonState}} \to A_{OL_{PersonState}} \ with \\
removeUnwashed_{ps_{A_{OL}}}((p_1, p_2, p_3, p_4)) = (p_1, undefined, p_3, p_4), \\
removeUndressed_{ps_{A_{OL}}} : A_{OL_{PersonState}} \to A_{OL_{PersonState}} \ with \\
removeUndressed_{ps_{A_{OL}}}((p_1, p_2, p_3, p_4)) = (p_1, p_2, undefined, p_4), \\
removeHungry_{ps_{A_{OL}}} : A_{OL_{PersonState}} \to A_{OL_{PersonState}} \ with \\
removeHungry_{ps_{A_{OL}}}((p_1, p_2, p_3, p_4)) = (p_1, p_2, p_3, undefined), \\
removeUnwashed_{c_{A_{OL}}} : A_{OL_{ContextData}} \to A_{OL_{ContextData}} \ with \\
removeUnwashed_{c_{A_{OL}}}((c_1, c_2, c_3)) = (unknown, c_2, c_3), \\
removeHungry_{c_{A_{OL}}} : A_{OL_{ContextData}} \to A_{OL_{ContextData}} \ with \\
removeHungry_{c_{A_{OL}}}((c_1, c_2, c_3)) = (c_1, unknown, c_3), \\
removeUndressed_{c_{A_{OL}}} : A_{OL_{ContextData}} \to A_{OL_{ContextData}} \ with \\
removeUndressed_{c_{A_{OL}}}((c_1, c_2, c_3)) = (c_1, c_2, unknown), \\
isReadyStates_{A_{OL}} : A_{OL_{PersonState}} \to A_{OL_{Bool}} \ with \\
isReadyStates_{A_{OL}}(p) = \begin{cases} tt_{A_{OL}} & \text{if } p = (undefined, undefined, undefined, undefined) \\ ff_{A_{OL}} & \text{else} \end{cases} , \\
askOnline_{A_{OL}} :\to A_{OL_{DeviceStatus}} \ with \\
askOnline_{A_{OL}} = askOnline, \\
askOffline_{A_{OL}} :\to A_{OL_{DeviceStatus}} \ with \\
askOffline_{A_{OL}} = askOffline, \\
online_{A_{OL}} :\to A_{OL_{DeviceStatus}} \ with \\
online_{A_{OL}} = online, \\
offline_{A_{OL}} :\to A_{OL} \ with \\
offline_{A_{OL}} = offline, \\
sync_{A_{OL}} :\to A_{OL_{Sync}} \ with \\
sync_{A_{OL}} = sync, \\
remove_{A_{OL}} :\to A_{OL_{Action}} \ with \\
remove_{A_{OL}} = remove, \\
store_{A_{OL}} :\to A_{OL_{Action}} \ with \\
store_{A_{OL}} = store, \\
succ_{A_{OL}} : A_{OL_{ConnectionID}} \to A_{OL_{ConnectionID}} \ with \\
succ_{A_{OL}}(i) = i + 1, \\
t_{gdata_{A_{OL}}} : A_{OL_{RequestData}} \to A_{OL_{GlobalDataTmp}} \ with \\
t_{gdata_{A_{OL}}}(r) = r, \\
topic_{A_{OL}} : A_{OL_{GlobalData}} \to A_{OL_{Topic}} \ with \\
topic_{A_{OL}}((x, y, z)) = y,,
\end{array}$$

$t_{gldata_{A_{OL}}} : A_{OL_{OutputData}} \to A_{OL_{GlobalData}}$ *with* $t_{gldata_{A_{OL}}}(d) = d,$ $t_{indata_{A_{OL}}} : A_{OL_{GlobalData}} \to A_{OL_{InputData}}$ *with* $t_{indata_{A_{OL}}}(d) = d$

Table 11.2: Σ_{OL}-Algebra A_{OL}

11.3 Signature Σ_{SL}

This section defines signature Σ_{SL} and the following section 11.4 defines the corresponding Σ_{SL}-Algebra A_{SL} that enables an AHLI net equipped with this so defined signature and algebra to include AHLI nets, rules over AHLI nets as well as interaction schemes over AHLI nets as tokens.[1] In comparison to the definition of section 6.3 within this work, AHLI nets of this form are so called $AHOI_{AHLI}$ nets.

The in the following defined signature is included in the overall $AHOI_{AHLI}$ net that represents the model of the internal system behaviour of the Message Oriented Middleware of the system level as introduced in chapter 7 and as illustrated in figure 7.26.

Signature Σ_{SL} is defined as follows:

$\Sigma_{\mathbf{SL}} =$	
sorts :	$Transitions, Places, Bool, AHLINets, AHLINets_{OND}, AHLINets_{OFFD},$ $AHLINets_Q, AHLINets_{Per}, AHLINets_{Rq}, Mor, Rules, Rules_R, ISchemes$
opns :	$tt, ff :\to Bool,$ $enabled : AHLINets \times Transitions \to Bool,$ $fire : AHLINets \times Transitions \to AHLINets,$ $applicable : Rules \times Mor \to Bool,$ $transform : Rules \times Mor \to AHLINets,$ $\dot{\cup}_S : AHLINets \times AHLINets \to AHLINets,$ $cod : Mor \to AHLINets,$ $isomorphic_{PTNET} : AHLINets \times AHLINets \to Bool,$ $createRule : ISchemes \times AHLINets \to Rules,$ $receiveData :\to Rules_R,$ $emptyDataset :\to AHLINets_{Per},$ $t_{OND} : AHLINets \to AHLINets_{OND},$ $f_{OND} : AHLINets_{OND} \to AHLINets,$ $t_{OFFD} : AHLINets \to AHLINets_{OFFD},$ $f_{OFFD} : AHLINets_{OFFD} \to AHLINets,$ $t_Q : AHLINets \to AHLINets_Q,$ $f_Q : AHLINets_Q \to AHLINets,$ $t_{Per} : AHLINets \to AHLINets_{Per},$ $f_{Per} : AHLINets_{Per} \to AHLINets,$ $t_{Rq} : AHLINets \to AHLINets_{Rq},$ $f_{Rq} : AHLINets_{Rq} \to AHLINets,$ $t_R : Rules \to Rules_R,$ $f_R : Rules_R \to Rules$
vars :	$n, n_1, n_2, n_3 : AHLINets,$ $n_{OND} : AHLINets_{OND},$ $n_{OFFD} : AHLINets_{OFFD},$ $n_{Per} : AHLINets_{Per},$ $n_{Rq} : AHLINets_{Rq},$ $n_Q : AHLINets_Q,$ $r : Rules,$ $r_R : Rules_R,$ $m : Mor,$ $t : Transitions,$ $s : ISchemes$

[1] Compare to [HEM05] for signature and algebra definition.

Table 11.3: Signature Σ_{SL}

11.4 Σ_{SL}-Algebra A_{SL}

This section defines the corresponding Σ_{SL}-Algebra A_{SL} to section 11.3 that enables an AHLI net equipped with this so defined signature and algebra to include AHLI nets, rules over AHLI nets as well as interaction schemes over AHLI nets as tokens.[2] In comparison to the definition of section 6.3 within this work, AHLI nets of this form are so called $AHOI_{AHLI}$ nets.

The in the following defined algebra is included in the overall $AHOI_{AHLI}$ net that represents the model of the internal system behaviour of the Message Oriented Middleware of the system level as introduced in chapter 7 and as illustrated in figure 7.26.

Given vocabularies $T_0 = P_0 = I_0 = \mathbb{N}$, where T_0 is the set of possible transitions an AHLI net can contain, P_0 is the set of possible places an AHLI net can contain and I_0 is the set of possible individual tokens an AHLI net can contain, i.e. nets with an inifinite set of places, transitions and individual tokens occurs, the Σ_{SL}-Algebra A_{SL} is defined as follows. Consider, that sets for T_0, P_0, I_0 with such an extent are not necessary for the model presented in this work, since within this work only AHLI nets with finite sets of places, transitions and individual tokens are considered, however they are defined so due to better readability.

$\boldsymbol{\Sigma_{SL}}$**-Algebra** $\mathbf{A_{SL}}$ =

sorts : $A_{SL_{Transitions}} = T_0$,
$A_{SL_{Places}} = P_0$,
$A_{SL_{Bool}} = \{true, false\}$,
$A_{SL_{AHLINets}} =$ *the set of all AHLI nets over* T_0, P_0 *and* I_0
with the signature Σ_{OL}, *i.e.*
$A_{SL_{AHLINets}} = \{ANI | ANI = (\Sigma_{OL}, P, T, pre, post, cond, type, A, I, m)$
$\in Ob^{3}_{\mathbf{AHLINets}(\Sigma_{OL})}, P \subseteq P_0, T \subseteq T_0, I \subseteq I_0\} \cup \{undef\}$,
$A_{SL_{AHLINets_{OND}}} = A_{SL_{AHLINets}}$,
$A_{SL_{AHLINets_{OFFD}}} = A_{SL_{AHLINets}}$,
$A_{SL_{AHLINets_{Q}}} = A_{SL_{AHLINets}}$,
$A_{SL_{AHLINets_{Per}}} = A_{SL_{AHLINets}}$,
$A_{SL_{AHLINets_{Rq}}} = A_{SL_{AHLINets}}$,
$A_{SL_{Mor}} =$ *the set of all AHLI net morphisms for* $A_{SL_{AHLINets}}$
with identities on the signature, i.e.
$A_{SL_{Mor}} = \{f | f : ANI_1 \to ANI_2 \in Mor_{\mathbf{AHLINets}(\Sigma_{OL})}$
with $ANI_1, ANI_2 \in A_{SL_{AHLINets}}, f_\Sigma = id_{\Sigma_{OL}}$ *and* f_P, f_T, f_I *injective*$\}$,
$A_{SL_{Rules}} =$ *the set of all AHLI transformation rules over* $A_{SL_{AHLINets}}$
with possibly a negative application condition[4], *i.e.*
$A_{SL_{Rules}} = \{\varrho | \varrho = (L \xleftarrow{l} K \xrightarrow{r} R)$ *AHLI transformation rule*
with $L, K, R \in A_{SL_{AHLINets}}$ *and injective* **AHLINets** *morphisms* l, r[5]$\} \cup$
$\{r | r = (X \xleftarrow{x} L \xleftarrow{l} K \xrightarrow{r} R)$ *AHLI transformation rule with a negative application condition* $NAC(x)$ *with* $L, K, R, X \in A_{SL_{AHLINets}}$ *and injective* **AHLINets** *morphisms* $l, r, x\}$,
$A_{SL_{Rules_R}} = A_{SL_{Rules}}$,
$A_{SL_{ISchemes}} =$ *the set of all interaction schemes over* $A_{SL_{AHLINets}}$, *i.e*

[2] Compare to [HEM05] for signature and algebra definition.

[3] See section 6.1.4 for the definition of category **AHLINets(Σ)**.

[4] See section 6.2.2 for the definition of AHLI transformation rules and negative application conditions.

[5] Note that the injective **AHLINets** morphisms l, r of the rules not have to be strict on the tokens.

[6] See section 6.1.2.1, 6.1.2.2 and 6.1.2.3 for the definition of enabled consistent transition assignments and token selections.

[7] See section 6.1.2.3 for the definition of firing of *AHLI* nets.

[8] See section 6.2.5 and 6.2.6 for the definition of the applicability of *AHLI* transformation rules with and without negative application conditions.

[9] See section 6.2.3 for the definition of direct *AHLI* net transformations.

[10] See section 6.2.1 for the definition of pushout constructions.

[11] See section 6.7.2 and 6.7.1.2.1 for the definition of functor $\mathcal{V}_{PTNET}$ and P/T net isomorphisms.

[12] See section 6.6 for the definition of interaction schemes and amalgamated rules over maximal weakly disjoint matchings.

$A_{SL_{ISchemes}} = \{is | \forall s_i \in is : s_i : p_0 \to p_i \ with \ p_0, p_i \in A_{SL_{Rules}}\}$ (Compare to section 6.6)

opns : $tt_{A_{SL}} : \to A_{SL_{Bool}} \ with$

$tt_{A_{SL}} = true,$

$ff_{A_{SL}} : \to A_{SL_{Bool}} \ with$

$ff_{A_{SL}} = false,$

$enabled_{A_{SL}} : A_{SL_{AHLINets}} \times A_{SL_{Transitions}} \to A_{SL_{Bool}} \ for$

$ANI = (\Sigma_{OL}, P, T, pre, post, cond, type, A, I, m) \ with$

$$enabled_{A_{SL}}(ANI, t) = \begin{cases} true & \text{if } t \in T, \exists (t, asg) \in CT \text{ and} \\ & \text{token selection } S = (M, m, N, n) \text{ regarding to} \\ & ANI \text{ so that } (t, asg) \text{ is enabled under } S^6 \\ false & \text{else} \end{cases},$$

$fire_{A_{SL}} : A_{SL_{AHLINets}} \times A_{SL_{Transitions}} \to A_{SL_{AHLINets}} \ for$

$ANI = (\Sigma_{OL}, P, T, pre, post, cond, type, A, I, m) \ with$

$$fire_{A_{SL}}(ANI, t) = \begin{cases} (\Sigma_{OL}, P, T, & \text{if } t \in T, \exists (t, asg) \in CT \text{ and} \\ pre, post, cond, & \text{token selection } S = (M, m, N, n) \\ type, A, & \text{regarding to } ANI \text{ so that } (t, asg) \\ I' = (I \setminus M) \cup \ N, & \text{is enabled under } S, \\ m' : I' \to A \otimes P) & \text{i.e. } enabled_{A_{SL}}(ANI, t) = true \\ with \ m' \ as \ defined & \\ in \ section \ 6.1.2.3^7 & \\ & \\ undef & \text{else} \end{cases},$$

$applicable_{A_{SL}} : A_{SL_{Rules}} \times A_{SL_{Mor}} \to A_{SL_{Bool}} \ with$

$$applicable_{A_{SL}}(r, m) = \begin{cases} true & \text{if } r \text{ is applicable at match } m^8 \\ false & \text{else} \end{cases},$$

$transform_{A_{SL}} : A_{SL_{Rules}} \times A_{SL_{Mor}} \to A_{SL_{AHLINets}} \ with$

$$transform_{A_{SL}}(r, m) = \begin{cases} ANI' & \text{if } applicable_{A_{SL}}(r, m) = true \\ undef & \text{else} \end{cases}$$

where for AHLI transformation rule $r : L \xleftarrow{l} K \xrightarrow{r} R$,
match morphism $L \xrightarrow{m} ANI$ and $applicable_{A_{SL}}(r, m) = true$
we have a direct AHLI net transformation ${}^9 ANI \overset{r,m}{\Longrightarrow} ANI'$,

$\dot{\cup}_{S_{A_{SL}}} : A_{SL_{AHLINets}} \times A_{SL_{AHLINets}} \to A_{SL_{AHLINets}}$ defines the structural disjoint union of two given AHLI nets *with*

$\dot{\cup}_{S_{A_{SL}}}(ANI_1, ANI_2) = \underline{if} \ (ANI_1 = undef \vee ANI_2 = undef \vee$
$A_1 \neq A_{OL} \vee A_2 \neq A_{OL}) \ \underline{then} \ undef$
$\underline{else} \ ANI'$, where $ANI' = ANI_1 +_{ANI_0} ANI_2^{10}$ is the pushout object
resp. the gluing of ANI_1 and ANI_2 via $(ANI_0, f : ANI_0 \to ANI_1,$
$g : ANI_0 \to ANI_2)$ with $ANI_0 = (\Sigma_{OL}, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, A_{OL})$
and f, g being **AHLINets** morphisms with $f_\Sigma = id_{\Sigma_{OL}}$ and $f_A = id_{A_{OL}}$,

$cod_{A_{SL}} : A_{SL_{Mor}} \to A_{SL_{AHLINets}} \ with$

$cod_{A_{SL}}(f : ANI_1 \to ANI_2) = ANI_2,$

$isomorphic_{PTNET_{A_{SL}}} : A_{SL_{AHLINets}} \times A_{SL_{AHLINets}} \to A_{SL_{Bool}} \ with$

$$isomorphic_{PTNET_{A_{SL}}}(ANI_1, ANI_2) = \begin{cases} true & \text{if } \mathcal{V}_{PTNET}(ANI_1) \cong \\ & \mathcal{V}_{PTNET}(ANI_2)^{11} \text{ where } \mathcal{V}_{PTNET} \\ & \text{is the forgetful functor} \\ & \text{as defined in section 6.7.2.} \\ false & \text{else} \end{cases},$$

$createRule_{A_{SL}} : A_{SL_{ISchemes}} \times A_{SL_{AHLINets}} \to A_{SL_{Rules}} \ with$

$createRule_{A_{SL}}(is, ANI) = \tilde{p}$, where $\tilde{p}$ is the amalgamated rule over
maximal weakly disjoint matchings on *AHLI* net *ANI* by using interaction scheme is^{12},

$receiveData_{A_{SL}} : \to A_{SL_{Rules_R}} \ with$

$receiveData_{A_{SL}} = receiveData$ as defined in figure 7.33,

$emptyDataset_{A_{SL}} : \to A_{SL_{AHLINets_{Per}}} \ with$

$emptyDataset_{A_{SL}} = database$ as defined in figure 7.49,

$t_{OND_{A_{SL}}} : A_{SL_{AHLINets}} \to A_{SL_{AHLINets_{OND}}} \ with$

$t_{OND_{A_{SL}}}(x) = x,$

$f_{OND_{A_{SL}}} : A_{SL_{AHLINets_{OND}}} \to A_{SL_{AHLINets}} \ with$

$$
\begin{aligned}
&\qquad f_{OND_{A_{SL}}}(x) = x,\\
&t_{OFFD_{A_{SL}}} : A_{SL_{AHLINets}} \to A_{SL_{AHLINets_{OFFD}}} \ with\\
&\qquad t_{OFFD_{A_{SL}}}(x) = x,\\
&f_{OFFD_{A_{SL}}} : A_{SL_{AHLINets_{OFFD}}} \to A_{SL_{AHLINets}} \ with\\
&\qquad f_{OFFD_A}(x) = x,\\
&t_{Q_{A_{SL}}} : A_{SL_{AHLINets}} \to A_{SL_{AHLINets_Q}} \ with\\
&\qquad t_{Q_{A_{SL}}}(x) = x,\\
&f_{Q_{A_{SL}}} : A_{SL_{AHLINets_Q}} \to A_{SL_{AHLINets}} \ with\\
&\qquad f_{Q_A}(x) = x,\\
&t_{Per_{A_{SL}}} : A_{SL_{AHLINets}} \to A_{SL_{AHLINets_{Per}}} \ with\\
&\qquad t_{Per_{A_{SL}}}(x) = x,\\
&f_{Per_{A_{SL}}} : A_{SL_{AHLINets_{Per}}} \to A_{SL_{AHLINets}} \ with\\
&\qquad f_{Per_A}(x) = x,\\
&t_{Rq_{A_{SL}}} : A_{SL_{AHLINets}} \to A_{SL_{AHLINets_{Rq}}} \ with\\
&\qquad t_{Rq_{A_{SL}}}(x) = x,\\
&f_{Rq_{A_{SL}}} : A_{SL_{AHLINets_{Rq}}} \to A_{SL_{AHLINets}} \ with\\
&\qquad f_{Rq_A}(x) = x,\\
&t_{R_{A_{SL}}} : A_{SL_{Rules}} \to A_{SL_{Rules_R}} \ with\\
&\qquad t_{R_{A_{SL}}}(x) = x,\\
&f_{R_{A_{SL}}} : A_{SL_{Rules_R}} \to A_{SL_{Rules}} \ with\\
&\qquad f_{R_A}(x) = x,
\end{aligned}
$$

Table 11.4: Σ_{SL}-Algebra A_{SL}

11.5 Exemplary Formal Notation of an $AHLI$ net

In the following $AHLI$ net $device^{offline}_{sender}$ of figure 7.6 is exemplary formally defined as a representative for the formal notation of all other defined $AHLI$ nets within this work, since chapter 7 only delivers the visual form of the introduced $AHLI$ nets of the presented model.

The formal notation follows definition 6.1.1 of $AHLI$ nets.

$device^{offline}_{sender} = (\Sigma_{OL}$[13]$, P, T, pre, post, cond, type, A_{OL}$[14]$, I, m)$ with:

$$
\begin{aligned}
&P = \{Status, Data_{toSend}, sync_1, sync_2\}\\
&T = \{ask\ device\ online, ask\ device\ offline\}\\
&pre, post : T \to (T_{OP}(X) \otimes P)^{\oplus}\\
&\qquad pre(ask\ device\ online) = (offline, Status) \oplus (s, sync_1)\\
&\qquad pre(ask\ device\ offline) = (online, Status) \oplus (online, Status) \oplus (s, sync_2)\\
&\qquad post(ask\ device\ online) = (askOnline, Status)\\
&\qquad post(ask\ device\ offline) = (askOffline, Status)\\
&cond : T \to P_{fin}(Eqns(s, OP, X)) = \emptyset\\
&type : P \to S\\
&\qquad type(Status) = DeviceStatus\\
&\qquad type(Data_{toSend}) = DataToSend\\
&\qquad type(sync_1) = Sync\\
&\qquad type(sync_2) = Sync\\
&I = \{s_1, s_2\}\\
&m : I \to A \otimes P\\
&\qquad m(s_1) = (offline, Status)\\
&\qquad m(s_2) = (sync, sync_1)
\end{aligned}
$$

[13] See section 11.1 for definition of signature Σ_{OL}.

[14] See section 11.2 for definition of algebra A_{OL}.

List of Figures

List of Tables

Bibliography

[Bar09] Barnkow, Lorenz: *Eine Multitouch-fähige Küchentheke: Im Kontext des Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. Version: 2009

[Bar10] Barnkow, Lorenz: *Ausarbeitung zur Veranstaltung Projekt 1 im Masterstudiengang Informatik SoSe 2010, Eine Multitouch-fähige Küchentheke: Vorbereitende Arbeiten für den Tagesplaner*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-proj1/berichte.html. Version: 2010

[CK04] Christopoulou, Eleni ; Kameas, Achilles: Using Ontologies to Address Key Issues in Ubiquitous Computing Systems / Research Academic Computer Technology Institute. 2004. – Forschungsbericht

[DAS99] Dey, Anind K. ; Abowd, Gregory D. ; Salber, Daniel: A Context-Based Infrastructure for Smart Environments. In: *MANSE'99 Conference* (1999)

[DAS01] Dey, Anind K. ; Abowd, Gregory D. ; Sabler, Daniel: A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications. (2001)

[EBE+09] Ermel, Claudia ; Biermann, Enrico ; Ehrig, Hartmut ; Hoffmann, Kathrin ; Modica, Tony: Modeling Multicasting in Communication Spaces by Reconfigurable High-level Petri Nets / Technical University Berlin. 2009. – Forschungsbericht

[EEPT06] Ehrig, H. ; Ehrig, K. ; Prange, U. ; Taentzer, G.: *Fundamentals of Algebraic Graph Transformation*. Springer-Verlag Berlin Heidelberg 2006, 2006

[Ell09] Ellenberg, Jens: *Wecker 2.0 - Ein Wecker in einem ubicom Haus*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. Version: 2009

[Ell10a] Ellenberg, Jens: *Ausarbeitung Masterprojekt 1, Vorarbeiten für den Wecker 2.0*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-proj1/berichte.html. Version: 2010

[Ell10b] Ellenberg, Jens: *Seminarausarbeitung AW2, Ein Wecker in einem ubicom Haus*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw2/vortraege.html. Version: 2010

[Ell11a] Ellenberg, Jens: *Ausarbeitung Masterprojekt 2, Entwicklung des Wecker 2.0*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/ellenberg.pdf. Version: 2011

[Ell11b] Ellenberg, Jens: *Seminar Ringvorlesung im Masterstudiengang, Klassifizierung von Kontext in einer intelligenten Wohnung*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-seminar/ellenberg/bericht.pdf. Version: 2011

[EMC+01] Ehrig, H. ; Mahr, B. ; Cornelius, F. ; Grosse-Rhode, M. ; Zeitz, P.: *Mathematisch-strukturelle Grundlagen der Informatik*. Bd. 2. Springer-Verlag Berlin Heidelberg 2001, 2001

[Fis10] Fischer, Winzent: *Entwicklung einer Werkzeugumgebung für Algebraische High-Level Netze mit Anwendung auf ein Szenario zur Einsatzsteuerung bei der Berliner Feuerwehr*, Technical University Berlin, Diplomarbeit, 2010

[GBEE10] Golas, Ulrike ; Biermann, Enrico ; Ehrig, Hartmut ; Ermel, Claudia: A Visual Interpreter Semantics for Statecharts Based on Amalgamated Graph Transformation / Technical University Berlin. 2010. – Forschungsbericht

[Gol10] Golas, Ulrike: Multi-Amalgamation in M-Adhesive Categories / Technical University Berlin. 2010. – Forschungsbericht

[Gre01] Greenberg, Saul: Context as a Dynamic Construct. In: *HUMAN-COMPUTER INTERACTION, 2001, Volume 16, pp. 257–268, Lawrence Erlbaum Associates, Inc.* (2001)

[Haa02] Haase, Kim: *Java Message Service API Tutorial*. http://download.oracle.com/javaee/1.3/jms/tutorial/. Version: 2002

[Har09] Hardenack, Frank: *Das intelligente Bett -Interpretation von Schlafphasen als Beispiel für Bodymonitoring im Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. Version: 2009

[Har10a] Hardenack, Frank: *Bodymonitoring in Smart Homes*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/berichte.html. Version: 2010

[Har10b] Hardenack, Frank: *Seminarausarbeitung AW2, - Das intelligente Bett -, Semantische Interpretation von Bodymonitoring-Rohdaten*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw2/vortraege.html. Version: 2010

[Har11] Hardenack, Frank: *Seminarausarbeitung Ringvorlesung (WS 2010/2011), - Das intelligente Bett - Semantische Interpretation auf Basis kapazitiver Sensoren*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-seminar/hardenack/bericht.pdf. Version: 2011

[HBS$^+$02] Hapner, Mark ; Burridge, Rich ; Sharma, Rahul ; Fialli, Joseph ; Stout, Kate: *Java Message Service - Specification Version 1.1*. April 2002

[HEM05] Hoffmann, Kathrin ; Ehrig, Hartmut ; Mossakowski, Till: High-Level Nets with Nets and Rules as Tokens / Technical University Berlin,University of Bremen. 2005. – Forschungsbericht

[Hil05] Hilty, L. et a.: The Precautionary Principle in the Information Society: Effects of Pervasive Computing on Health and Environment. / Report of TA-SWISS, Centre for Technology Assessment, Berne. 2005. – Forschungsbericht

[HK04] Hellenschmidt, Michael ; Kirste, Thomas: A Generic Topology for Ambient Intelligence / Fraunhofer Institute for Computer Graphics. 2004. – Forschungsbericht

[HM03] Hoffmann, Kathrin ; Mossakowski, Till: Algebraic Higher-Order Nets: Graphs and Petri Nets as Tokens. (2003)

[INK06] Ikonen, Veikko ; Niemelä, Marketta ; Kaasinen, Eija: Scenario-Based Design of Ambient Intelligence. In: *Ubiquitous Computing Systems Lecture Notes in Computer Science, LNCS 4239, S. 58* (2006)

[JW05] Jang, Seiie ; Woo, Woontack: 5W1H: Unified User-Centric Context. In: *Proceedings of the 7th International Conference on Ubiquitous Computing* (2005)

[Liv10] *Living Place Hamburg Blog*. http://livingplace.informatik.haw-hamburg.de/blog/. Version: 2010

[Mah04] Mahmoud, Qusay H.: *Middleware for Communications*. John Wiley & Sons, Ltd, 2004

[MGE+10] Modica, Tony ; Gabriel, Karsten ; Ehrig, Hartmut ; Hoffmann, Kathrin ; Shareef, Sarkaft ; Ermel, Claudia ; Golas, Ulrike ; Hermann, Frank ; Biermann, Enrico: Low- and High-Level Petri Nets with Individual Tokens / Technical University Berlin. 2010. – Forschungsbericht

[Mil06] Milner, Robin: Ubiquitous Computing: Shall we Understand It? In: *The Computer Journal Lecture* (2006), March

[OV10] Otto, Kjell ; Voskuhl, Sören: *Projektbericht Sommersemester 2010, Entwicklung einer Architektur für den Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-proj1/berichte.html. Version: 2010

[OV11a] Otto, Kjell ; Voskuhl, Sören: *Projektbericht Wintersemester 10/11, Weiterentwicklung der Architektur des Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/otto-voskuhl.pdf. Version: 2011

[OV11b] Otto, Kjell ; Voskuhl, Sören: *Weiterentwicklung der Architektur des Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master10-11-proj2/otto-voskuhl.pdf. Version: 2011

[PPE+05] Padberg, J. ; Prange, U. ; Ehrig, H. ; Ermel, C. ; Hoffmann, K.: *Petrinetze - Modellierung, Strukturierung und Kompositionalität*. October 2005

[RON] *An editor for Reconfigurable Object Nets*. http://tfs.cs.tu-berlin.de/roneditor/

[RV09] Rahimi, Mohammadali ; Vogt, Matthias: *Aufbau des Living Place Hamburg*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-proj/berichte.html. Version: 2009

[SDA99] Salber, Daniel ; Dey, Anind K. ; Abowd, Gregory D.: The Context Toolkit: Aiding the Development of Context-Enabled Applications. In: *Proceedings of CHI'99, Pittsburgh, PA, May 15-20, 1999* (1999)

[Tec] Technical University Berlin, AGG T.: *AGG*. Technical University Berlin: Technical University Berlin, http://user.cs.tu-berlin.de/~gragra/agg/

[Vos09] Voskuhl, Sören: *Bereitstellung einer Sensorwolke*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/vortraege.html. Version: 2009

[Vos10] Voskuhl, Sören: *Ausarbeitung Anwendungen 2, Architekturen für Context-Aware Systeme*. Hamburg University of Applied Sciences. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw2/vortraege.html. Version: 2010

[Wan07] Wang, Jiacun: *Timed Petri Nets Timed Petri Nets Theory and Application*. Springer-Verlag GmbH, 2007

[Wei91] Weiser, Mark: The Computer for the 21st Century. In: *Scientific Am., reprinted in IEEE Pervasive Computing, 2002, pp. 19-25* (1991)

[YK10] Yuriyama, Madoka ; Kushida, Takayuki: Sensor-Cloud Infrastructure - Physical Sensor Management with Virtualized Sensors on Cloud Computing. In: *IBM* (2010)