Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

AW2 - SoSe 2009

André Goldflam

Architekturen intelligenter Wohnungen

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

André Goldflam

Thema

Architekturen intelligenter Wohnungen

Stichworte

Ubiquitous Computing, Ambient Living, Software Architecture

Kurzzusammenfassung

In dieser Ausarbeitung werden drei Kategorien Architekturen intelligenter Wohnungen jeweils anhand einer Beispielarchitektur vorgestellt sowie bewertet

Inhaltsverzeichnis

1 Einleitung **4**

2 Entwicklung, Standards und Konferenzen **4**

3 Architekturen **5**
3.1 Zentral: iROS 6
3.2 Dezentral: Personal Home Server 9
3.3 Workflowbasiert: ubiHome 12

4 Zusammenfassung und Ausblick **14**

Literatur **16**

1 Einleitung

Im Rahmen des „Ambient Living Place“-Projektes an der HAW-Hamburg wird eine intelligente Wohnung ausgebaut. Es wurden bereits verschiedene Arbeiten in diesem Bereich angefertigt (z.B. Dreyer (2009)). Weiterhin arbeiten zurzeit ca. 10 Studenten in Rahmen von Bachelor- und Masterarbeiten an diesem Forschungsprojekt. Innerhalb dieser Ausarbeitung sollen Architekturen für intelligente Wohnungen vorgestellt und bewertet werden.

2 Entwicklung, Standards und Konferenzen

Mark Weiser beschrieb 1991 in seinem Artikel Weiser (1991) erstmals die Verknüpfung von unsichtbaren Computern und alltäglichen Gebrauchsgegenständen. In Harter u. a. (1999) wurde ein Sensorenframework vorgeschlagen, welches die Position von Personen innerhalb eines Gebäudes überwacht und so Anwendungen ermöglicht, auf Bewegungen von Benutzern zu reagieren, z.B. einen Bildschirm einzuschalten, wenn sie nahe an diesen herantreten. Im Folgenden sollen Entwicklungen im Bereich der Kommunikations-Middleware dargestellt werden.

LON (Local Operating Network) stammt aus dem Bereich der Gebäudeautomatisierung und wurde 1990 von der Firma Echelon [1] entwickelt. LON ist ein Feldbus und ermöglicht die dezentralisierte Automatisierung in Netzwerken. Für die Kommunikation wird ein eigenes Kommunikationsprotokoll namens LonTalk eingesetzt [Echelon-Corporation].

Der HAVi (Home Audio Video Interoperability) 1.0-Standard wurde 1999 von einem großen Konsortium, bestehend unter anderem aus Grundig, Philips, Sony, Thompson und weiteren, spezifiziert. HAVi arbeitet als ein verteiltes System und ermöglicht über die vorgegebenen Standards die Entwicklung von Anwendungen, welche auf die Dienste der einzelnen Geräte aufbauen [HAVi-Initiative]. Da die aktuellsten Veröffentlichungen auf der Webseite der Initiative [2] aus dem Jahr 2004 sind, ist allerdings anzunehmen, dass dieser Standard nicht weitergepflegt wird.

UPnP (Universal Plug and Play) wurde ursprünglich von Microsoft entwickelt. 1999 wurde das sogenannte UPnP Forum [3] gegründet, ein Zusammenschluss aus Firmen der Unterhaltungsindustrie, und spezifiziert seitdem die UPnP-Standards . UPnP dient zum Auffinden und Veröffentlichen von Diensten. Dabei setzt es auf ein peer-to-peer-Netzwerk durch TCP/IP auf.

[1] http://www.echelon.com
[2] http://www.havi.org
[3] http://www.upnp.org

Nachdem ein Dienst gefunden wurde, wird ein Kommunikationskanal direkt zwischen zwei Geräten aufgebaut UPnP-Forum. Im Gegensatz zu HAVi erscheinen bei UPnP Forum ständig neue Standards und Spezifikationen bezüglich neuer Geräte wie z.B. Medienservern.

OSGi (Open Services Gateway Initiative) wird von der OSGi-Alliance [4] (ehemals die Open Services Gateway Initiative) spezifiziert. Die Gründung erfolgte ebenso 1999. OSGi ist ein auf Java basierendes Framework, welches auf dem Komponentenmodell aufbaut. Eine Komponente besitzt dabei einen eigenen Namens- und Datenbereich, die Kommunikation zwischen Komponenten erfolgt über das Framework. Ein weitere zentrale Aufgabe des Frameworks ist die Verwaltung von Komponenten, diese können dynamisch installiert, gestartet und wieder gestoppt werden OSGI-Alliance (2007).

Im Rahmen des hier vorgestellten Forschungsbereiches sind drei internationale Konferenzen von Interesse:

TRIDENT: Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities[5], erstmals 2005 abgehalten.

ICOST: International Conference On Smart homes and health Telematic[6], erstmals 2003 abgehalten.

UbiComp: International Conference on Ubiquitous Computing[7], erstmals 1999 abgehalten.

3 Architekturen

Die untersuchten Architekturen lassen sich in folgende Kategorien einteilen:

Zentral: Die Kommunikation zwischen Diensten wird über eine zentrale Instanz gesteuert.

Dezentral: Die Dienste sowie die Steuerungsanwendung sind dezentralen organisiert.

Workflowbasiert: Das System erzeugt zur Erfüllung eines Benutzerkommandos eine Folge von Dienstaufrufen, die entsprechend miteinander verknüpft sind.

Von dieser Einteilung unabhängig unterscheiden sich die Architekturen in der Art und Weise, wie Dienste beschrieben werden, wie Dienste miteinander kommunizieren und wie durch eine Zusammenstellung von verschiedenen Diensten Anwendungen für ein intelligente Haus realisiert werden können. Die Bewertung einer Architektur soll folgende Punkte berücksichtigen:

[4] http://www.osgi.org
[5] http://www.tridentcom.org
[6] http://www.icost-conference.org/
[7] http://www.ubicomp.org

Integration: Beschreibt, wie bereits bestehende Anwendungen und Hardwarekomponenten in die Architektur integriert werden können.

Dienste: Schildert, wie Dienste beschrieben und verwaltet werden.

Kontext: Stellt die Beschreibung und Verarbeitung der Kontextinformationen dar.

Benutzerschnittstellen: Erläutert, wie die grafische Oberfläche für den Benutzer erstellt wird.

Orchestrierung: Beschreibt, wie und ob das System die Verkettung von Basisdiensten zu komplexeren Diensten gestattet.

Im Folgenden sollen nun Forschungsarbeiten aus dem Bereich des „Ambient Living“ sowie deren Architekturen beschrieben und am Ende nach den vorgestellten Kriterien bewertet werden. Zusätzlich erfolgt abschließend eine Auflistung von Architekturen, welche ähnliche Merkmale aufweisen.

3.1 Zentral: iROS

Da zu diesen Zeitpunkt an der HAW Hamburg im Rahmen des „Ambient Living Place“-Projektes bereits Arbeiten auf dem von Terry Winograd vorgestellten Event Heap basieren, soll das an der Stanford Universität entwickelte System hier trotz seines Alters vorgestellt werden.

Das Projekt „System Support for Interactive Workspaces“ sollte hauptsächlich zur Unterstützung von Expertenteams dienen. Dabei beschränkt sich das System nicht nur auf den Bereich der Kollaboration, sondern auch auf die Steuerung der Geräte innerhalb des Konferenzraumes, z.B. das Ein- und Ausschalten des Beamers (vgl.. Winograd (2001),S. 2).

Systemaufbau

Abbildung 3.1 zeigt den Aufbau des Systems. Im Folgenden sollen die einzelnen Schichten näher betrachtet werden.

Communication Infrastructure: Event Heap Der Event Heap soll für „Interactive Workspaces“ die Funktionalität bereitstellen, die eine Event Queue bei traditionellen Einzelnbenutzer-GUIs zur Verfügung stellt (vgl. Brad Johanson (2001), S.1). Als Entwurfsprinzip wird eine Blackboard-Architektur verwendet. Anwendungen können Events eines Typs auf das Board setzen oder nach Ereignissen suchen, welche bestimmte Kriterien erfüllen. Die Anwendungen kommunizieren dabei nicht direkt miteinander, sondern nur über den Event Heap. Wenn eine kommunizierende Anwendung kurzfristig ausfallen sollte, bedeutet dies

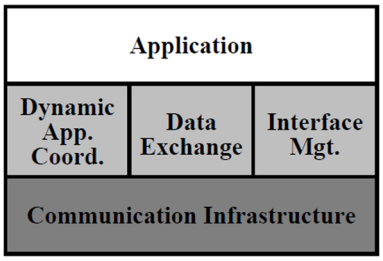

Abbildung 1: Architektur (Quelle: Winograd (2001),S. 3)

keinen zwangsläufigen Zusammenbruch des Kommunikationspartners, da nur dessen Nachrichten ausbleiben. Sollte ein Event nicht konsumiert werden, wird er nach einer gewissen Zeit verworfen (vgl. Brad Johanson (2001), S.3). Da der Event ein zentraler Bestandteil der Kommunikationsschicht darstellt, soll dieser im Folgenden genauer betrachtet werden.

Event Der Event Heap basiert auf TSpaces von IBM (P. Wyckoff und Ford), welches eine Implementation von Tuple Spaces ist. Diese wurden erstmals von Nicholas Carriero beschrieben. Ein Event beinhaltet eine Liste aus mehreren Feldern, wobei ein Feld durch fest definierte Attribute beschrieben wird. Eine Anwendung kann einen Event auf den Event Heap legen. Zusätzlich ist es einer Anwendung möglich, über ein TemlateEvent den Event Heap nach bestimmten Events zu durchsuchen. Das TemplateEvent ist eine Schablone, in der die Suchkriterien eingetragen sind. Sie enthält reservierte Felder für die konkreten Werte, die ein Suchergebnis zurückliefern würde. Wird bei der Suche über ein TemplateEvent ein passendes Event gefunden und zurückgegeben, kann dieses entweder gelöscht oder auf dem Event Heap belassen werden. Über gewisse Standard-Felder können z.B. Eventtyp, Quelle, Zielperson oder Zielgruppe angegeben werden (Eine Tabelle der Felder ist in Brad Johanson (2001) auf S.5 zu finden). Über ein Feld kann für ein Event festgelegt werden, wie oft es verbraucht werden darf. Ist ein Event unendlich oft konsumierbar, wird über einen Mechanismus verhindert, dass derselbe Event wiederholt über eine Suche gefunden wird (vgl. Brad Johanson (2001), S.4 - S.6).

Data Exchange: Data Heap Die Heterogenität der Anwendungen (und der Geräte, auf denen sie laufen) erschwert den Austausch von Daten, da die Datenformate in der Regel nicht von allen Teilnehmern verstanden werden. Ebenso heterogen sind die Datenquellen, die von einfachen Dateizugriffen auf File-Servern bis zum Live-Streaming einer Kamera reichen können. Um den Austausch von Daten zwischen Anwendungen zu ermöglichen, soll das System den Zugriff auf Daten typ- und ortstransparent gestalten (vgl. Winograd (2001), S. 4).

Der Data-Heap setzt dynamisch nutzbare Datentransformatoren ein, um Original-Daten in ein für die Zielanwendung lesbares Format zu transformieren. Zusätzlich entkoppelt er die Speicherung und die Verwaltung der Daten. Anfragen an den Data Heap (wie z.B. das Speichern oder Suchen) werden über den Event Heap weitergeleitet. Nachdem der Event Heap die Vermittlung beendet hat, werden die Daten direkt zwischen Anwendung und Data Heap ausgetauscht. Er vermittelt zwischen Anwendungen, die Daten austauschen wollen, indem er Protokoll-Inkompatibilitäten ausgleicht (durch die zur Verfügungstellung eines Java-Protocol-Handlers oder die Nutzung von Standards wie z.B. WebDAV). Die vorgestellten Mechanismen bieten den nötigen Grad an Orts- und Typtransparenz. Die Originaldaten werden dabei innerhalb des Data Heaps verwaltet (vgl. Winograd (2001), S. 6).

Interface Management System: ICRafter Sobald ein Dienst sich am Event Heap anmeldet, wird über Java Reflection automatisch eine XML-Dienstbeschreibung generiert. Wenn nun eine Anwendung die Anfrage für eine Benutzerschnittstelle für einen bestimmten Dienst stellt, liefert sie verschiedene Meta- und Kontextinformationen (wie z.B. unterstützte Sprachen, Größe des Displays, etc.) an ICrafter mit. Als Ergebnis erhält die Anwendung die generierte Schnittstelle (z.B. in HTML) zurück. Befehle des Benutzers werden dabei als Events über den Event Heap an den Dienst gesendet (vgl. Winograd (2001), S. 6 - 7).

Dynamic Application Coordination Diese Komponente erlaubt es Anwendungen des „Interactive Workspaces“, miteinander zu kommunizieren, inklusive Anwendungen, welche nicht ursprünglich dafür ausgelegt worden sind. In dem hier vorgestellten System müssen die Anwendungen miteinander keine „Verhandlung“ durchführen (wie z.B. bei RPCs), da die Events selbstbeschreibend sind, die Kommunikation anonymisiert abläuft und automatische Nachrichtenkonverter eine Integration erleichtern. Zusätzlich können Anwendungen sich auch für Events registrieren, die nicht ursprünglich für sie gedacht waren (vgl. Winograd (2001), S. 4 - 5).

Bewertung Die Integration von Anwendungen gestaltet sich mit IROS einfach, da deren Schnittstellen als Dienst über den Event Heap angebunden werden können. Zusätzlich erleichtert der Data Heap den Datenaustausch zwischen verschiedenen Anwendungen. Der Aufbau eines Events ist sehr übersichtlich und leicht zu implementieren. Die Dienstbeschreibungen werden implizit über die in dem Event enthaltenen Informationen gegeben, wobei der Typ des Events den Namen des Dienstes identifiziert. Innerhalb des Systems Sensoren können ihre Daten einfach an den Event Heap und so den Kontext für andere Anwendungen liefern, allerdings existiert keine spezielle Systemkomponente, die für die Aufbereitung und Verwaltung des Kontextes zuständig ist. Bei vielen Sensoren und hohen Aktualisierungsraten kann es hier möglicherweise zu Performanceengpässen kommen. Die Benutzerschnittstelle für die Nutzung

eines Dienstes wird über ICrafter generiert. Die Orchestrierung muss innerhalb einer eigens implementierten Anwendung erfolgen.

Ähnliche Arbeiten

- Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home Rodden u. a. (2004)

3.2 Dezentral: Personal Home Server

In Nakajima und Satoh (2006) argumentieren die Autoren, dass aktuelle Middleware nicht ausreichend die spontane Kollaboration in einer kontextbezogenen Form unterstützt. Daher wird ein sogenannter „personal home server“ (PHS) vorgeschlagen, welcher die Fernsteuerung von Diensten ermöglichen soll. Dabei besitzt jede Person einen eigenen „Home Server“ in Form eines mobilen Endgerätes (z.B. ein PDA). Da jede Person ein eigenes Eingabegerät besitzt, werden auf diesem entsprechend die persönlichen Einstellungen gespeichert. Als Einsatzgebiete sollen dabei nicht nur das eigene Heim, sondern z.B. auch Parkplätze oder Flughäfen dienen (vgl. Nakajima und Satoh (2006), S. 380).

Systemaufbau

Im Folgenden soll der Personal Home Server vorgestellt werden. Das hier beschriebene System unterstützt ausschließlich UPnP-basierte Geräte. Anschließend soll eine Übersicht des Ablaufes dargestellt und darauffolgend die einzelnen Schritte detaillierter betrachtet werden.

Personal Home Server (PHS) Abbildung 2 zeigt die Architektur eines PHS. Als Basis wird das bereits erwähnte OSGi genutzt. Zur Implementierung des UPnP-Protokolls sind die Komponenten UPnP, SSDP[8], GENA [9], SOAP (Simple Object Access Protocol), sowie Http Server und Http Client notwendig. Die Komponente Database speichert Informationen über Geräte, welche sich in der Nähe des Benutzers befinden. Preference-Inference reichert die Kommandos des Benutzers mit Kontextinformationen an. In-Forwarder erhält Kommandos in einem

[8] Simple Service Discovery Protocol: Dient zur Suche nach UPnP-Geräten
[9] General Event Notification Architecture: Ermöglicht die Abonnierung von Events für UPnP-Geräten

Abbildung 2: Architekturübersicht (Quelle Nakajima und Satoh (2006), S. 382)

kodierten URL-Format und wandelt diese in ein SOAP-Format um. Entsprechend übermittelt die Out-Forwarder-Komponente ein SOAP-Kommando an ein Zielgerät. Jini/SOAP ermöglicht die Anbindung von Java-Anwendungen, da SOAP-Nachrichten in einen RMI-Aufruf umgewandelt werden können. Die Jini/SOAP-Komponente soll an dieser Stelle nur die Integrationsfähigkeit der Architektur unter Beweis stellen. Die UI-Komponente generiert eine HTML-Benutzerschnittstelle, welche auf der Dienstbeschreibung basiert (vgl. Nakajima und Satoh (2006), S. 382-383).

Ablaufsübersicht Ein PHS sammelt über die SSDP-Komponente Informationen über Geräte in der Nähe des Benutzers und speichert sie dann über die Database-Komponente ab. Anschließend generiert die UI-Komponente ein Präsentationsdokument. Ein Anzeigegerät in der Nähe des Benutzers erhält dieses Dokument und stellt dieses dar. Das Präsentationsdokument enthält die URL-kodierten Kommandos, durch welche der Benutzer bestimmte Aktionen auslösen kann. Das Präsentationsdokument wird durch die Preference-Inference-Komponenten hinsichtlich des Benutzers angepasst. Löst der Nutzer nun eine Aktion aus, indem er z.B. auf einen Link drückt, wird das Kommando an den PHS über die In-Forwarder-Komponenten geschickt, verarbeitet und später wieder über die Out-Forwarder-Komponente an das Zielgerät geschickt (vgl. Nakajima und Satoh (2006), S. 382).

Auffinden eines Dienstes Betreffende Heimgeräte senden in einem Zeitintervall Nachrichten, um einem PHS ihre Existenz mitzuteilen. Zusätzlich wird durch den PHS selbst eine zyklische Suche durchgeführt, auf die alle in Reichweite befindlichen Geräte antworten. Die Dienstbeschreibungen werden in der Database-Komponente gespeichert. Meldet sich ein Gerät nicht mehr, wird seine Dienstbeschreibung nach einer bestimmten Zeit verworfen. Da alle Geräte über einen PHS gesteuert werden können, aber z.B. nur die Inhaber des Gerätes das Steuerungs-Recht dazu haben sollen, soll an dieser Stelle ein Sicherheitsmechanismus eingeführt werden, auf den in diesem Zusammenhang nicht näher eingegangen werden soll (vgl. Nakajima und Satoh (2006), S. 384).

Beschreibung eines Dienstes Die Dienstbeschreibung wird durch ein RDF-Dokument [10] spezifiziert. Zur Identifizierung des Zielgerätes werden dabei dessen Attribute genutzt. Würde beispielsweise eine URL „function=light&?type=ceiling“ enthalten, würden die Servicebeschreibungen nach dem function- und dem type-Tag durchsucht und auf die vorgegebenen Werte geprüft werden (vgl. Nakajima und Satoh (2006), S. 384 - S. 385).

Kontextanreicherung von Kommandos Durch die Preference-Inference-Komponente werden Kommandos in Abhängigkeit des Benutzerkontextes umgewandelt. Zum Beispiel würde eine URL welche „?function=TV“ enthält, das Kommando expandieren und bestimmte Zuordnungsregeln, welche in der Kontext-Datenbank der Preference-Inference-Komponente gespeichert sind, auf das expandierte Kommando anwenden. In Bezug auf das eben genannte Beispiel könnte eine Regel folgendermaßen lauten: $Function(x, TV) \rightarrow Location(x, living-room), Support(x, Premiere)$. Dabei individualisiert die Preference-Inference-Komponente das ursprüngliche Kommando, indem sie Kontextinformationen nutzt. Als Ergebnis würde „?function=TV?support=BS?locations=living-room“ zurückgeliefert werden. Als Regel-Engine wird SWI-Prolog [11] eingesetzt und über JPL [12] angesteuert. Die Regeln müssen dabei durch den Benutzer selbst erstellt werden (vgl. Nakajima und Satoh (2006), S. 386).

Generierung der Benutzerschnittstelle Die UI-Komponente generiert automatisch ein HTML-Dokument, welches die Steuerung der sich in Reichweite befindlichen Geräte ermöglicht. Dabei werden die schon vorher beschriebenen URLs als Link in dem HTML-Dokument platziert. Sollte der PHS selbst keine Anzeigemöglichkeit besitzen, wird nach einem entsprechenden Gerät gesucht. Das gewünschte Anzeige-Gerät fragt dann das Präsentationsdokument an und stellt es in einem Webbrowser dar (vgl. Nakajima und Satoh (2006), S. 386 - S. 387).

Ähnliche Arbeiten

1. A generic UPnP architecture for ambient intelligence meeting rooms and a control point allowing for integrated 2D and 3D interaction (vgl. Shirehjini (2005))
2. A generic mobile agent framework for ambient intelligence(vgl. Lee u. a. (2008))

[10] Resource Description Framework: http://www.w3.org/RDF/
[11] http://www.swi-prolog.org/
[12] http://www.swi-prolog.org/packages/jpl/java$_api/index.html$

Bewertung Solange Dienste UPnP unterstützen oder Jini unterstützen, stellt sich die Integration sehr einfach dar. Für andere Anwendungen müssen entsprechende Adapter implementiert werden. Die Dienstbeschreibungen im RDF-Format sind semantisch deutlich aussagekräftiger als die in iROS. Die Kontextanreicherung über die vorgestellte Expandierung der Benutzerkommandos durch Prolog erscheint kompliziert und schlecht erweiterbar, da der Benutzer selbst diese Regeln aufstellen muss. Zusätzlich werden an dieser Stelle nur persönliche Einstellungen beachtet und Kontextinformationen von Sensoren allgemein vernachlässigt. Interessant an dem System ist allerdings, dass der PHS auch zur Steuerung von Geräten in anderen Wohnungen als der eigenen problemlos eingesetzt werden kann. Die Benutzerschnittstelle für das Bedienen der Dienste wird automatisch generiert und für den Benutzer personalisiert. Eine Orchestrierung von Diensten ist nicht möglich.

3.3 Workflowbasiert: ubiHome

Ha u. a. (2007) beschreibt das Design sowie dessen Implementation eines intelligenten Hauses. Die Infrastruktur schließt dabei die Integration von Sensoren, Endgeräte, darauf aufbauende Dienste und weitere ubiquitäre Ressourcen ein. Kommandos werden dabei ausgeführt, indem beteiligte Dienste über einen Planungsmechanismus in einem Ausführungsplan zusammengestellt und anschließend ausgeführt werden.

Systemaufbau

Abbildung 3 zeigt die Architektur des ubiHome-Systems. Im Folgenden soll eine kurze Ablaufübersicht den Einstieg erleichtern und anschließend sollen die einzelnen Komponenten beschrieben werden.

Ablaufsübersicht Als Kommunikations-Basis werden semantische Web Services eingesetzt und die verschiedenen Ressourcen einheitlich über diese angebunden. Die Beschreibung bezüglich Fertigkeiten und Schnittstellen eines Dienstes wird in OWL-S[13] (Web Ontology Language for Web Services) definiert. Die Sensordaten sollen in der sogenannten Service Knowledge Registry (SKR) gehalten werden und stellen die Wissensbasis für einen Service Agent (SA) dar. Der SA kann über diese Wissensbasis einen Workflow-Plan zur Ausführung einer bestimmten Benutzeranfrage erstellen und über SOAP[14] (Simple Object Access Protocol) mit den beteiligten Geräten und Diensten kommunizieren, um den Plan umzusetzen (vgl. Ha u. a. (2007), S.1).

[13] http://www.w3.org/Submission/OWL-S/
[14] http://www.w3.org/TR/soap12-part1/

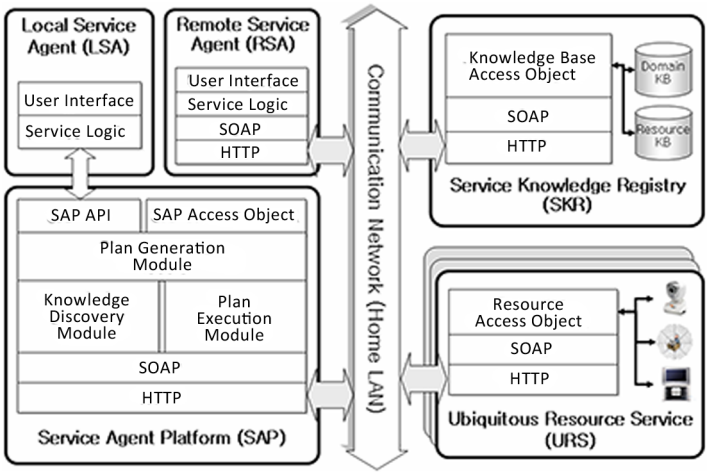

Abbildung 3: Architekturübersicht

Service Agent (SA) Es werden lokale (LSA) und entfernte (RSA) Service Agents unterschieden. Die Implementierung eines LSA setzt direkt auf die SAP (Service Agent Platform) auf und wird von Geräten genutzt, welche selbst genügend Rechenkapazität besitzen, um die SAP lokal verfügbar zu halten. Der RSA wird auf einem von dem SAP getrennten System implementiert. Dabei kommuniziert der RSA über das SAP access object per Web Service. Beide SAs besitzen eine Benutzerschnittstelle sowie die Service-Logik. Wird über die Benutzerschnittstelle eine Aktion ausgelöst, enkodiert die Service-Logic-Schicht das Kommando in ein OWL-S Request-Profil und sendet dieses entweder direkt durch die SAP API (LSA) oder das bereits erwähnte SAP access object (RSA) (vgl. Ha u. a. (2007), S.2).

Service Agent Platform (SAP) Die oberste Schicht dient der Kommunikation mit den Services Agents. Das Plan Generation Module erhält von der oberen Schicht das OWL-S Request-Profil und erhält über das darunter liegende Knowledge Discovery Module das benötigte Wissen, um den geforderten Dienst zu planen. Dabei durchsucht das Knowledge Discovery Module zum einen den Ubiquitous Resource Service (URS) und zum anderen die SKR. Die Suche dient zum Auffinden kontextbezogener Daten und möglicher Dienste, die bei der Umsetzung der Aktion als relevant eingestuft werden. Nach Erhalt dieser Informationen wird ein Ausführungsplan für die Aktion erstellt. Dabei kommen Planungsmethoden aus dem Bereich der künstlichen Intelligenz zum Einsatz. Zuletzt führt das „Plan Execution Module“ den Plan aus, indem die URS per SOAP angesprochen werden können (vgl. Ha u. a. (2007), S.2).

Ubiquitous Resource Service (URS) Ein URS stellt die konkrete Implementation eines Web Services für eine ubiquitäres Gerät wie z.B. eine Licht- oder eine Klimaanlage dar. Jeder URS besitzt ein eigenes „resource access object“, über welches ein Gerät oder eine Gruppe von Geräten angesprochen werden kann, z.B. eine Gruppe von RFID-Lesern welche den Standort des Benutzers überwachen (vgl. Ha u. a. (2007), S.3).

Service Knowledge Registry (SKR) Die SKR enthält zwei verschiedene Typen von Wissen: Domain- und Gerätewissen. Das Domainwissen enthält das Wissen über die Zusammenstellungen von atomaren Diensten und speichert zusätzlich verarbeitete Dienstbeschreibungen ab, indem es deren IOPEs („Input“, „Output“, „Precondition“ und „Effects“) bestimmt. Das Gerätewissen beschreibt die atomaren Dienste, die ein Gerät ausführen kann (vgl. Ha u. a. (2007), S.3).

Bewertung Externe Anwendungen müssen über einen eigens implementierten semantischen Webservice integriert werden. Da die Dienstbeschreibungen in OWL-S definiert sind und von deren Qualität die Verträglichkeit mit den anderen Komponenten abhängig ist, müssen diese mit entsprechender Sorgfalt erstellt und bereits vorhandene Services und deren Beschreibungen bekannt sein. Der Vorteil liegt dabei dann in der deutlich mächtigeren Aussagekraft einer Dienstbeschreibung, durch welche überhaupt erst der hohe Grad an Orchestrierung erreicht werden kann. Für die Kontextdaten gilt dieselbe Sorgfalt bezüglich deren Dienstbeschreibungen. Die Benutzerschnittstelle wird bei dieser Architektur innerhalb der LSA und RSA implementiert und kann nicht durch eine Komponente generiert werden.

Ähnliche Arbeiten

1. iFlat - Eine dienstorientierte Architektur für intelligente Räume (vgl. Stegelmeier u. a. (2009))
2. Context-aware based mobile service for ubiquitous home (vgl. **?**)
3. Contextual information management using contract: based workflow (vgl. Murthy und Krishnamurthy (2005))

4 Zusammenfassung und Ausblick

Es wurden verschiedene Architekturen vorgestellt und deren Stärken bzw. Schwächen bezüglich der vorgestellten Merkmale aufgezeigt. Ziel der weiteren Beschäftigung mit dieser Thematik ist es nun, die Stärken unter Vermeidung der Schwächen der einzelnen Architekturen

zusammenzuführen. Dabei soll ein hoher Abstraktionsgrad der jeweiligen Schichten eine optimale Unterstützung anzubindender Systeme bieten. Ein mögliches Szenario könnte folgendermaßen aussehen:

Im Rahmen einer intelligenten Wohnung scheint ein zentrales System vertretbar zu sein, da viele der verschiedenen Komponenten fest in die Umgebung eingebunden sind (z.B. die Sensoren), außerdem kann davon ausgegangen werden, dass alle Komponenten zu jeder Zeit miteinander vernetzt sind. Die Kommunikationsschicht sollte asynchrone sowie synchrone Kommunikation unterstützen. Die asynchrone Kommunikation könnte über den Event Heap realisiert werden und dadurch die aufgezeigten Vorteile bezüglich der Integration von Fremdanwendungen mit einschließen. Die Service- und Kontextbeschreibungen sollten zum einen in OWL-S und zum anderen in einem einfachen Format, z.B. in XML, angegeben werden können. Durch die Unterstützung eines einfachen Formates könnten Fremdanwendungen leichter integriert werden, die nicht auf semantisch starke Kontexte angewiesen sind. Die Möglichkeit der Orchestrierung ist dadurch weiterhin gegeben. Benutzerschnittstellen sollen generiert, aber auch über durch die Anwendungen selbst zur Verfügung gestellt werden können.

Inwieweit so eine Architektur realisierbar ist, soll im Rahmen des Moduls „Projekt 2“ geklärt werden.

Literatur

[Brad Johanson 2001] BRAD JOHANSON, Pat Hanrahan Terry W.: The Event Heap: An Enabling Infrastructure for Interactive Workspaces. (2001)

[Dreyer 2009] DREYER, Markus: *Ein nutzeradaptierendes agentenbasiertes TV System als Teil eines intelligenten Hauses*, Diplomarbeit, 2009

[Echelon-Corporation] ECHELON-CORPORATION: *LonWorks Technology Overview*. – URL http://www.echelon.com/communities/developers/lonworks/default.htm

[Guo und Chung 2008] GUO, Xi ; CHUNG, Paul W. H.: The architecture of a web service-based remote control service system. In: *iiWAS '08: Proceedings of the 10th International Conference on Information Integration and Web-based Applications & Services*. New York, NY, USA : ACM, 2008, S. 555–558. – ISBN 978-1-60558-349-5

[Ha u. a. 2007] HA, Young-Guk ; SOHN, Joo-Chan ; CHO, Young-Jo: ubiHome: An Infrastructure for Ubiquitous Home Network Services, June 2007, S. 1–6

[Harter u. a. 1999] HARTER, Andy ; HOPPER, Andy ; STEGGLES, Pete ; WARD, Andy ; WEBSTER, Paul: The anatomy of a context-aware application. In: *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA : ACM, 1999, S. 59–68. – ISBN 1-58113-142-9

[HAVi-Initiative] HAVI-INITIATIVE: *WhitePaper HAVi*

[Lee u. a. 2008] LEE, Yung-Chuan ; KHORASANI, Elham S. ; RAHIMI, Shahram ; GUPTA, Bidyut: A generic mobile agent framework for ambient intelligence. In: *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*. New York, NY, USA : ACM, 2008, S. 1866–1871. – ISBN 978-1-59593-753-7

[Murthy und Krishnamurthy 2005] MURTHY, V. K. ; KRISHNAMURTHY, E. V.: Contextual information management using contract: based workflow. In: *CF '05: Proceedings of the 2nd conference on Computing frontiers*. New York, NY, USA : ACM, 2005, S. 236–245. – ISBN 1-59593-019-1

[Nakajima und Satoh 2006] NAKAJIMA, Tatsuo ; SATOH, Ichiro: A software infrastructure for supporting spontaneous and personalized interaction in home computing environments. In: *Personal Ubiquitous Comput.* 10 (2006), Nr. 6, S. 379–391. – ISSN 1617-4909

[OSGI-Alliance 2007] OSGI-ALLIANCE: *OSGi Service Platform, Core Specification Release 4, Version 4.1*. 2007

[P. Wyckoff und Ford] P. WYCKOFF, T. J. L. ; FORD, D. A.: TSpaces. In: *IBM Systems Journal 37(3)*. – URL http://www.almaden.ibm.com/cs/TSpaces

[Rodden u. a. 2004] RODDEN, Tom ; CRABTREE, Andy ; HEMMINGS, Terry ; KOLEVA, Boriana ; HUMBLE, Jan ; , Karl-Petter ; HANSSON, Pär: Between the dazzle of a new building and its eventual corpse: assembling the ubiquitous home. In: *DIS '04: Proceedings of the 5th conference on Designing interactive systems*. New York, NY, USA : ACM, 2004, S. 71–80. – ISBN 1-58113-787-7

[Shirehjini 2005] SHIREHJINI, Ali A. N.: A generic UPnP architecture for ambient intelligence meeting rooms and a control point allowing for integrated 2D and 3D interaction. In: *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*. New York, NY, USA : ACM, 2005, S. 207–212. – ISBN 1-59593-304-2

[Stegelmeier u. a. 2009] STEGELMEIER, Sven ; WENDT, Piotr ; LUCK, Kai von: iFlat - Eine dienstorientierte Architektur für intelligente Räume, 2009

[UPnP-Forum] UPNP-FORUM: *Understanding UPnP: A White Paper*

[Weiser 1991] WEISER, Mark: The Computer for the Twenty-First Century. In: *Scientific American* (1991)

[Winograd 2001] WINOGRAD: System Support for Interactive Workspaces. (2001)