Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Projektbericht Wintersemester 10/11

Kjell Otto, Sören Voskuhl

Weiterentwicklung der Architektur des Living Place Hamburg

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Inhaltsverzeichnis

1 Einleitung

Die Entwicklung in der Informatik zeichnet sich durch kontinuierlich kleiner und leistungsfähiger werdende Computer aus. Mit diesem Trend geht einher, dass Informationstechnologien fortwährenden Einzug in das alltägliche Leben des Menschen erhalten. So sind viele Personen zu jeder Zeit von mehreren Computern, wie ihrem Smartphone, dem MP3-Player und ihrem Notebook umgeben. Diese Allgegenwärtigkeit von Computern wurde von Mark Weiser als „Ubiquitous Computing“ bezeichnet (Weiser, 1991). Ein Grundgedanke dieses Begriffs liegt darin, dass Personen bei verschiedensten Aufgaben unterstützt werden und eine als natürlich empfundene Interaktion mit dem Gerät entsteht.
Damit sich eine solche intuitive Mensch-Computer-Interaktion ergibt und eine reale Hilfe für die Person bei der Bearbeitung von Aufgaben entsteht, ist es essentiell, dass der aktuelle Kontext in die Reaktionen eines Systems einbezogen wird. Mit dieser Eigenschaft von Computersystemen befasst sich der Begriff „Context Awareness“ (Abowd u. a., 1999). Hier sammeln die Informationstechnologien Daten über ihre Umgebung, mit dem Ziel einen aktuellen Bezugsrahmen zu erstellen und ihr Verhalten diesem anzupassen. Diese Anpassungsfähigkeit von kontextabhängigen Systemen erhält eine stetig wachsende Bedeutung bei der Einrichtung privater Wohnräume. Während traditionell die elektronischen Geräte einer Wohnung, die Computer des Bewohners und die Automatik des Hauses voneinander getrennte Domänen darstellen, werden diese Bereiche miteinander vernetzt, um mit Hilfe einer sog. „Sensorfusion“ einen im Netzwerk einheitlichen Kontext zu kreieren.
Eine Herausforderung bei einer Vernetzung der Geräte ergibt sich aus der Heterogenität der einzelnen Systeme. Deshalb ist es eine bedeutende Aufgabe bei der Entwicklung von „Context-Aware Systemen“ eine Middleware bereitzustellen, in die verschiedene Systeme integriert werden können, sodass ein Austausch von Diensten und Informationen zwischen allen Komponenten gewährleistet wird. Die hier vorgestellte Projektarbeit wird sich dabei mit verschiedenen Aspekten bei der Bereitstellung einer solchen Middleware befassen.
Das Gebiet der „Intelligent Homes“ wird an der HAW Hamburg seit über drei Jahren unter dem Begriff „Living Place Hamburg“ erforscht. Nachdem im ersten Schritt verschiedene Szenarien eines modernen Lebens in einzelnen Laboren entwickelt wurden, wird seit Anfang 2009 in einem Gebäude der HAW eine real bewohnbare Wohnung aufgebaut. Auf 130m^2 entsteht hier derzeit eine als Loft ausgeprägte Wohnung, in der aus verschiedenen Projekten stammende Technologien und neuartige Interaktionsformen unter authentischen Bedingungen getestet werden können. Ein 3D-Modell dieser Wohnung wird in der Abbildung 1.1

dargestellt. Die in diesem Projekt erreichten Ergebnisse sollen die grundlegenden Elemente für die Architektur des Living Places bilden.

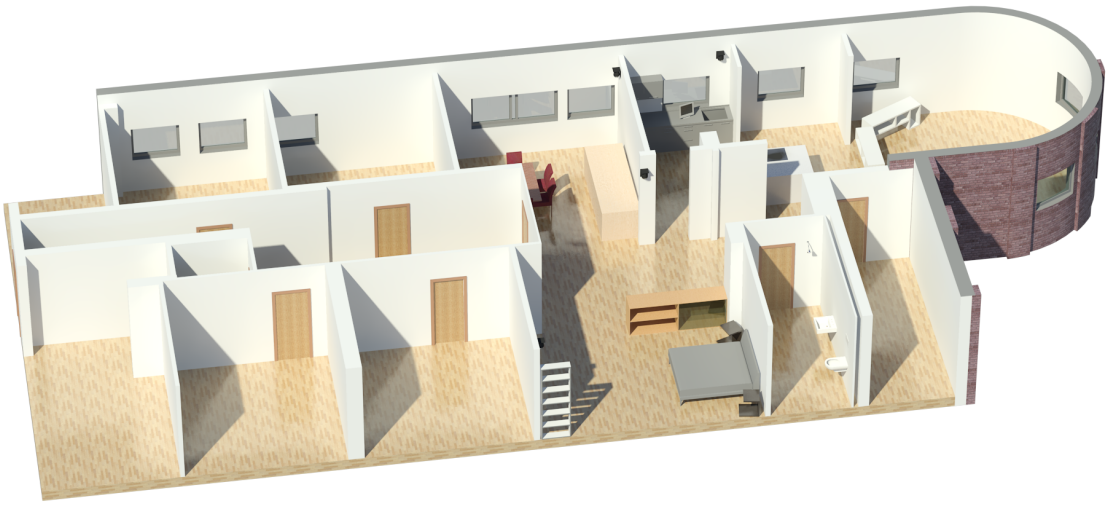

Abbildung 1.1: 3D Rendering des Living Place Hamburg

Ziele

Diese Projektarbeit findet im engen Zusammenhang mit der Veranstaltung „Masterprojekt 1 des Sommersemesters 2010“[1] statt. Einleitend soll in dieser Arbeit die im Ausblick des Projekts 1 erwähnte Persistenzebene entwickelt werden. Hierzu wird es nötig sein, die von der Kommunikationsschnittstelle bereitgestellten Sicherungsmechanismen dahingehend zu überprüfen, ob diese für die gewünschte Nachrichtenspeicherung und einer darauf folgenden Weiterverarbeitung geeignet sind.
Ein weiterer Schwerpunkt dieser Arbeit wird die Realisierung einer Bibliothek sein, die den Zugriff auf den ActiveMQ[2], dem Message-Broker des Living Place, für Anwendungsentwickler erleichtert. Mit Hilfe einer solchen Bibliothek würden diese kein tieferes Verständnis über die Arbeitsweise und Ansteuerung des ActiveMQ benötigen, sondern könnten ihren Fokus vollständig auf die eigene Applikation setzen.
Des Weiteren ist eine Migration der im Projekt 1 in einer Laborumgebung implementierten Szenarien in die reale Wohnumgebung vorgesehen.

[1] http://users.informatik.haw-hamburg.de/ ubicomp/projekte/master2010-proj1/otto_voskuhl.pdf
[2] http://activemq.apache.org/

2 Projektarbeit

2.1 Persistenz

In Otto und Voskuhl (2010) wurde in der Archiktekturbeschreibung für den Living Place Hamburg beabsichtigt, eine eigene Persistenzebene zu implementieren, damit alle über den AcitveMQ versendeten Nachrichten für eine weitere Verarbeitung zur Verfügung stehen. Der Ursprung dieser Entscheidung lag darin, dass die interne Datenbank des ActiveMQ, die sog. „KahaDB“[1], nicht als reiner Nachrichtenspeicher entwickelt wurde. Nach unseren Erkenntnissen dient diese primär dazu, nach einem Ausfall des Systems, alle bisherigen Topics und Queues sowie nicht ausgelieferte Nachrichten wiederherzustellen.
Aus diesem Grund haben wir verschiedene Varianten zur Speicherung der Nachrichten entwickelt und ihre Vor- und Nachteile gegeneinander abgewogen.

JDBC-Datenbank

Zur Persistierung von Nachrichten wird vom ActiveMQ neben der KahaDB außerdem der Einsatz von JDBC-Datenbanken unterstützt. Da sich der Zugriff auf die interne Datenbank als schwierig herausgestellte und wir bereits Erfahrungen im Umgang mit der „Java Database Connectivity“- Schnittstelle gesammelt hatten, wurde der Fokus im ersten Schritt auf den Einsatz einer JDBC-Datenbank gesetzt. Verwendet wurde hierzu das relationale Datenbankverwaltungssystem „MySQL“[2].
Zur Veränderung der Nachrichtenpersistenz des Message Brokers muss die Standardkonfiguration in der Datei actimvemq.xml um folgende Einträge erweitert werden:

```
0
    <persistenceAdapter>
      <jdbcPersistenceAdapter dataSource="#mysql−ds"/>
    </persistenceAdapter>

5   <bean id="mysql−ds" class="org.apache.commons.dbcp.BasicDataSource" destroy−method="close">
      <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
      <property name="url" value="jdbc:mysql://localhost/activemq?relaxAutoCommit=true"/>
      <property name="username" value="root"/>
      <property name="password" value=""/>
10    <property name="maxActive" value="200"/>
```

[1] http://activemq.apache.org/kahadb.html
[2] http://www.mysql.de/

```
    <property name="poolPreparedStatements" value="true"/>
</bean>
```

Nach diesem Schritt muss die Datei „mysql-connector-java-5.0.6-bin.jar“ in den Ordner ./lib/-optional kopiert und eine MySQL-Datenbank mit dem Namen „activemq“ erstellt werden. Ein Neustart des Brokers beinhaltet, dass die Nachrichten nun in die neue Datenbank geschrieben werden.

Bewertung

Ein Vorteil bei der Verwendung einer MySQL-Datenbank hat sich in der Handhabung der persistierten Daten ergeben, da nun alle versendeten Nachrichten an eine Queue oder einem Topic mit einem „Durable Subscriber“ in die Datenbank geschrieben werden und mit Hilfe von SQL-Befehlen abgefragt werden können. Allerdings werden in dieser Variante alle persistierten Daten identisch zum Einsatz einer KahaDB behandelt, was bedeutet, dass automatisch alle ausgelieferten Nachrichten wieder aus dem Nachrichtenspeicher gelöscht werden. Aus diesem Grund wurde die Verwendung einer weiteren Datenbank diskutiert, die unabhängig vom Message Broker arbeitet und über ein Java-Programm alle Nachrichten erhält, die in die interne Persistenzebene des ActiveMQ geschrieben werden.
Da für diese Lösung verschiedene Teilkomponenten entwickelt werden müssten die nur in Verbindung mit der derzeitigen Version des ActiveMQ verwendet werden könnten, haben wir uns gegen diese Variante der Persistierung entschieden.

ActiveMQ Browser

Der „ActiveMQ Browser“[3] ist ein Open Source Projekt, welches eine grafische Oberfläche zur Administration des ActiveMQ anbietet. Hier wird u.a. die Möglichkeit offeriert Topics und Queues zu abonnieren und die gesendeten Nachrichten im Zip-Format zu speichern. Hierzu wird ein entsprechender Ordner auf der Festplatte angelegt und im „ActiveMQ Browser“ als Local Store eingerichtet. Nun kann zu jedem Nachrichtenkanal des ActiveMQ angegeben werden, ob eine Kopie der Nachricht im Local Store gespeichert werden soll. Diese wird als Textdatei angelegt, welche den JMSHeader, den TextMessageBody und die UserProperties enthält.
Da dieses Tool somit einen Mechanismus bereitstellt mit dem die Messages an eine weitere Komponente weitergeleitet werden, wurde die Option in Betracht gezogen, den Code des „ActiveMQ Browser“ so anzupassen, dass die Kopie der Nachrichten direkt in eine Datenbank geschrieben wird.

Bewertung

[3] http://sourceforge.net/projects/activemqbrowser/

Der „ActiveMQ Browser“ birgt den Vorteil, dass er bereits eine vollständig implementierte Persistenz besitzt. Das bedeutet, wenn der Code dieses Tools an die Bedürfnisse der Architektur des Living Place angepasst würde, könnten unabhängig vom Message Broker Kopien der Nachrichten in eine beliebige Datenbank geschrieben werden. Allerdings wäre die Adaption des Tools mit einem sehr hohen Aufwand verbunden gewesen, da es aus mehreren zehntausend Zeilen Code besteht und entsprechend an mehreren Codesegmenten erweitert werden müsste. Ein weiterer negativer Aspekt bei einer Speicherung der Nachrichten durch den „ActiveMQ Browser“ ist, dass jede Queue und jedes Topic durch eine Benutzeraktion an den Local Store gebunden wird. Dies hätte zur Folge, dass jeder neue Nachrichtenkanal manuell in die Persistenzschicht aufgenommen werden müsste. Da diese Vorgehensweise für einen langfristigen Einsatz nur bedingt geeignet ist, haben wir uns gegen den „ActiveMQ Browser“ entschieden.

Persistenz im Living Place Message-Wrapper

Nachdem die bisher vorgestellten Lösungen lediglich mit erheblichen Aufwand realisiert werden könnten, haben wir uns dazu entschieden, den Nachrichtenspeicher direkt in den bereitgestellten Wrapper zu integrieren. Dies hat zur Folge, dass jeder Entwickler, der seine Nachrichten persistieren möchte den Wrapper verwenden muss. Allerdings gehen mit dieser Variante einige Vorteile einher. Zum einen kann nun unabhängig vom ActiveMQ eine geeignete Datenbank gewählt werden und zum anderen müssen keine zusätzlichen Tools eingesetzt werden, sodass sich keine weiteren Abhängigkeiten innerhalb der Architektur ergeben, was die Stabilität des Systems stärkt. Eine detaillierte Betrachtung der Implementierung findet im Abschnitt 2.2 statt.
In einem weiteren Schritt wurden nun verschiedene Datenbanken verglichen, mit dem Ziel ein geeignetes Datenbankmanagementsystem (DBMS) zu wählen. Während unserer Recherche stellte sich schnell heraus, dass uns ein relationales DBMS (RDBMS) vor große Herausforderungen stellen würde, da eine relationale Datenbank aus Tabellen besteht, die einem festen Datenbankschema unterliegen. Dabei erwies es sich als problematisch zu definieren wie die einzelnen Tabellen und deren Spalten aussehen müssten, da nicht endgültig festgelegt ist, in welchem Format die einzelnen Nachrichten an die Kommunikationsschnittstelle gesendet werden. Folglich haben wir weitere DBMS betrachtet und sind auf den Begriff „dokumentenorientierte Datenbanken“ gestoßen. Im Gegensatz zu einem RDBMS werden hier einzelne Dokumente gespeichert, welche entweder aus strukturierten Dateien oder aus „Binary Large Objects“, welche im Sinne eines Datenbankzugriffs unstrukturiert sind, bestehen können. Dabei ist denkbar, dass die strukturierten Dateien aus einer Reihe von Datenfeldern bestehen, die jeweils in Form von Key-Value-Paaren vorliegen. Da diese Eigenschaft einen sehr interessanten Aspekt zur Speicherung der Nachrichten liefert, haben wir uns entschlossen mit „CouchDB“ und „MongoDB“ zwei bekannte Datenbanken im Hinblick auf ihre

Einsetzbarkeit in der Architektur des Living Place Hamburg zu untersuchen. Die Ergebnisse werden im folgenden Abschnitt vorgestellt.

CouchDB

CouchDB[4] ist ein über die „RESTful JSON API“ erreichbarer dokumentenorientierter Datenbankserver der Apache Software Foundation. Die in dieser Datenbank gespeicherten Dokumente stellen Objekte dar, die sog. Felder enthalten. Die Werte dieser Felder können beispielsweise Strings, Nummern oder auch Listen sein. Eine Datenbank dieser Art ist somit eine flache Ansammlung dieser Dokumente, wobei jeder Eintrag über eine eindeutige ID identifiziert werden kann. Darüber hinaus kann CouchDB sowohl offline als auch online verteilt betrieben werden. Dies bedeutet, dass jeder Datenbankknoten ein unabhängiges Replikat der Datenbank besitzt, auf dem Applikationen mit Zugriff auf den vollen Funktionsumfang arbeiten können. Änderungen werden dabei nach einem festgelegten Zeitplan oder einer erneuten Einbindung in das Netzwerk bidirektional repliziert.
Die im Living Place eingesetzten Sensoren werden innerhalb kurzer Zeit eine hohe Anzahl an zu persistierenden Dokumenten generieren. Deshalb ist die Geschwindigkeit in der neue Daten in die Datenbank geschrieben werden können ein bedeutender Aspekt. Folglich haben wir die Performance der CouchDB genauer betrachtet, vgl. (Anderson u. a., 2010).
Die hier vorgestellten Tests stammen aus einer JavaScript Testsuite, welche die Anzahl gespeicherter Dokumente innerhalb von zehn Sekunden zählt und wurde auf einem MacBook Intel Core 2 Duo durchgeführt. Im ersten Schritt wird an dieser Stelle das Einfügen von Daten in einem „Bulk“ untersucht und in der Tabelle 2.1 veranschaulicht.
Aus dem Ergebnis dieses Tests lässt sich schließen, dass bei größer werdenden Paketen eine bessere Performance erzielt wird, da letztlich weniger Schreibvorgänge auf der Datenbank stattfinden. Allerdings müssen zuvor genügend Daten generiert werden, damit die geforderte Paketgröße erreicht wird und eine Speicherung stattfindet. Ein Problem in dieser Vorgehensweise könnte sich darin ergeben, dass ein Dokument erst nach einer gewissen Zeit in der Datenbank gespeichert wird, da die notwendige Bulk-Größe erst zu einem späteren Zeitpunkt erreicht wird. Somit würde die Information nicht in die aktuelle Kontexterfassung anderer Applikationen eingebunden werden.
Eine alternative Methode zur Speicherung der Nachrichten in die CouchDB ist das Einfügen jedes einzelnen Dokumentes. In diesem Fall kommt es jedoch zu einem deutlichen Performance-Verlust, in dem nur noch 257.93 Dokumente pro Sekunde in der Datenbank gespeichert werden können. Es ist davon auszugehen, dass diese Zahl an Dokumenten bei der hohen Anzahl an Sensoren im Living Place sehr schnell überschritten wird, womit die CouchDB für den Einsatz in dieser Architektur nicht geeignet ist.

MongoDB

[4] http://couchdb.apache.org/

Bulk-Größe	Anzahl Dokumente	Dokumente pro Sekunde
100 Docs	4400 Docs	437.37 Docs/Sek
1000 Docs	17000 Docs	1635.40 Docs/Sek
5000 Docs	30000 Docs	2508.15 Docs/Sek
10000 Docs	30000 Docs	2699.54 Docs/Sek

Tabelle 2.1: Einfügen von Bulk-Dokumenten in die CouchDB

Die MongoDB (von „humongous“) weist viele Ähnlichkeiten zur beschriebenen CouchDB auf. Hier handelt es sich ebenfalls um eine dokumentenorientierte Datenbank mit einem schemalosen Objekt-Datenspeicher im JSON-Format. Die Philosophie dieser Datenbank ist dabei sehr Performance-orientiert, was für dieses Projekt einen sehr essentiellen Aspekt darstellt. Einen Performance-Vorteil gegenüber der CouchDB erreicht man hier durch ein sog. „update-in-place“-Verfahren, was beinhaltet, dass die Datenbank keine neue vollständige Kopie des Objektes anlegt, sondern lediglich entsprechende Teile im Dokument ändert. CouchDB hingegen verwendet das „Multiversion Concurrency Control“ (MVCC), welches hohe Kosten verursacht, indem viele verschiedene Versionen eines Objektes in der Datenbank gehalten werden.
Ein weitere Steigerung der Performance wird durch sprachspezifische Datenbanktreiber erreicht. Während die CouchDB REST als Interface zur Datenbank benutzt, werden in der MongoDB native Socket-Protokolle für den Client/Server Zugriff auf die Datenbank verwendet.

Die Ergebnisse eines eigens in Java entwickelten Performance Tests werden in der Tabelle 2.2 demonstriert. Der Test wurde auf einem Mac Pro mit einem Intel Xeon x5472 Prozessor und 10GB RAM ausgeführt. Die Dokumente wurden dabei einzeln eingefügt, sodass keine Bulks zum Einsatz gekommen sind. Das Resultat veranschaulicht eine deutlich bessere Performance gegenüber der CouchDB. Während die CouchDB für 30000 Nachrichten in einem Bulk von 10000 Dokumenten ca. 2,7 Sekunden benötigt, hat die MongoDB 100000 einzelne Dokumente in 1,86 Sekunden eingepflegt.
Aufgrund dieser hohen Leistungsfähigkeit und weiteren positiven Eindrücken, wie beispielsweise einer sehr gut zu bedienenden Konsole, haben wir uns dazu entschieden, die dokumentenorientierte Datenbank MongoDB für die Architektur des Living Place auszuwählen.

Anzahl Dokumente	Dauer
1000 Docs	418 ms
10000 Docs	496 ms
100000 Docs	1.86 sek

Tabelle 2.2: Einfügen von einzelnen Dokumenten in die MongoDB

2.2 Living Place Messaging

Die Projektarbeiten im Living Place Hamburg werden, sobald sie Informationen publizieren oder konsumieren, über den ActiveMQ als Mittelpunkt aller Kommunikationen kommunizieren, vgl. Otto und Voskuhl (2010). Die Architektur ist darauf ausgelegt, möglichst alle Aspekte dieser Kommunikationen zu gewährleisten. Zusätzlich zum Austausch der Nachrichten soll die Möglichkeit bestehen, auch im Nachhinein, bestimmte Kommunikationsverläufe nachzuvollziehen bzw. im Kontext zu untersuchen. Damit alle Beteiligten eine einheitliche Schnittstelle geboten bekommen um ihre Daten auszutauschen, haben wir den sog. ActiveMQ Wrapper entwickelt. Wie in 2.1 beschrieben, werden Nachrichten die im Living Place kommuniziert werden persistiert. Der erste Ansatz war die Persistierung mit in den ActiveMQ zu verlagern. Nachdem dies keine Möglichkeit bot, haben wir uns dazu entschieden, die Persistierung in einer externen Datenbank zu realisieren. Da der ActiveMQ keine Möglichkeit bietet die Nachrichten weiterzuleiten wird das Senden der Daten im ActiveMQ-Wrapper durchgeführt.

2.2.1 Aufgabenstellung des ActiveMQ-Wrappers

Der ActiveMQ-Wrapper soll zwei grundlegende Konzepte im Living Place Hamburg vereinfachen. Die Kommunikation zwischen zwei oder mehr Parteien und die Persistierung der Nachrichten die kommuniziert werden. Anforderungen an den Wrapper sind Performance und Übertragungssicherheit. Die erste Idee zur Architektur des Wrappers war, dass alle benötigten Methoden statisch bleiben und der Entwickler keine Instanzen der Schnittstelle erstellen muss. Dazu wurden für beide Kommunikationspattern, Publisher/Subscriber und Producer/Consumer, Messsagedispatcher entwickelt, die für die Verteilung der Nachrichten auf die entwicklerseitigen Zugriffe verteilt wurden. Probleme entstanden konzeptionell mit der Multithreadingfähigkeit. Dadurch, dass nach dieser Architektur kein Entwickler mehr Instanzen verwenden musste um aus verschiedenen Threads auf den ActiveMQ-Wrapper zuzugreifen, wurden Nachrichten nicht nachvollziehbar von unterschiedlichen Threads empfangen. Dies hat das grundsätzliche Verhalten der vom ActiveMQ zur Verfügung gestellten Funktionalität verändert und wurde aus dem selben Grund verworfen. Nachdem wir

uns von der Idee einer statischen API getrennt hatten, haben wir die Anforderungen mit Instantiierungen realisiert. Recherchen haben ergeben, dass bei einem Zugriff aus mehreren Threads auf die Clientbibliothek des ActiveMQ, jeder Thread eine eigene „javax.jms.Session“ sowie „javax.jms.Connection“ benötigt. In der finalen Version wird bei der Instantiierung der Objekte nun eine eigene Session und Connection für jedes Objekt von der „org.apache.activemq.ActiveMQConnectionFactory“ erzeugt. Mit diesem Ansatz haben wir nach ausführlichen Tests die Probleme im Verhalten mit mehreren Threads lösen können. Der LPActiveMQ-Wrapper ist somit multithreadingfähig. Die Persistenz im Wrapper wurde über ein Singleton-Pattern mit einer geteilten Ressource realisiert, sodass alle Instanzen nur eine Verbindung zur MongoDB verwenden. Das bietet den Vorteil das, sobald nötig, ein „Bulk-Write“ für den Wrapper implementiert werden kann und der Datenbankserver nicht von jedem Client mehrere Verbindungen akzeptieren muss.

2.2.2 Living Place Messaging Wrapper

Die Java-Bibliothek „LivingPlaceActiveMQWrapper“[5][6] vereinfacht die Benutzung des ActiveMQ dadurch, dass sich der Entwickler nicht wie vorher, alle Teile die zur Verwendung benötigt werden selbst instantiieren muss, sondern mit einem Aufruf der API die Infrastruktur gestellt wird, vgl. Listing 2.1. Um die Bibliothek möglichst klein zu halten und die Abhängigkeiten automatisch auflösen zu können, haben wir ein Maven-Projekt erstellt. Ein Target zum Erstellen eines JAR-Archivs wurde bereitgestellt, sodass man bei Erweiterungen keinen Aufwand in diesen Teil der Entwicklung stecken muss.

Die unterschiedlichen Konzepte zur Kommunikation, Publisher/Subscriber und Producer/-Consumer, wurden in Otto und Voskuhl (2010) und The Apache Software Foundation (2010) erläutert und zeigen im Vergleich einen deutlich höheren Aufwand. Die Konfiguration des Wrappers kann über eine Klasse „ConnectionSettings“ verändert werden und bietet dem Entwickler die Möglichkeit sich eine lokale Testumgebung zu instantiieren.

Listing 2.1: Beispielhafte Instantiierung der Wrapperklassen

```
0  /* Example instantiation of the wrapper classes which
    * include message spreading and persistence.
    */
   LPSubscriber s = new LPSubscriber("topicName");
   LPPublisher p = new LPPublisher("topicName");
5
   LPConsumer s = new LPConsumer("queueName");
   LPProducer p = new LPProducer("queueName");
```

Die Entwickler die eine Kommunikation aufbauen wollen, müssen das Topic bzw. die Queue bezeichnen können und sich nicht mehr mit der Server-IP, dem Protokoll oder dem Port des

[5] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Kommunikationsschnittstelle
[6] http://livingplace.informatik.haw-hamburg.de/svn/middleware/

ActiveMQ auseinandersetzen. Die Festlegung der IP-Addressbereiche für den Living Place Hamburg wurde noch nicht vorgenommen. Sobald der ActiveMQ eine feste IP im Living Place bekommen hat, werden die Einstellungen so angepasst, dass es nicht mehr nötig sein wird die Konfigurationsklasse zu verwenden. Für die Kommunikation im Living Place Hamburg auf TCP/IP-Basis soll in Zukunft ausschließlich der ActiveMQ-Wrapper verwendet werden. Im Folgenden wird der Aufbau des Wrappers und die Strukturierung des Codes genauer erläutert, um eventuellen Nachfolgern den Einstieg in den Wrapper zu erleichtern.

LPConsumer / LPProducer

Punkt-zu-Punkt Verbindungen werden im ActiveMQ-Wrapper über das Consumer/Producer-Pattern realisiert. Die Klassen heißen entsprechend „LPConsumer“ und „LPProducer“. Wir haben, um jedem die Wahl des Mechanismus beim konsumieren von Nachrichten selbst zu überlassen, mehrere Möglichkeiten implementiert, um mit der „LPConsumer“-Klasse Nachrichten zu empfangen, vgl. Listing 2.2. Da von anderen Studenten gewünscht war, nicht zwangsweise einen MessageListener implementieren zu müssen, haben wir zwei blockierende Methoden, mit und ohne Timeout entwickelt.

Der „LPProducer“ hingegen hat nur eine Methode zum Versenden der Nachrichten, vgl. Listing 2.3. Wie auch beim „LPConsumer“ werden hier Strings verwendet, um keine Abhängigkeiten zu bestimmten JSON-Bibliotheken zu erzwingen.

Listing 2.2: Consumermethoden zum Empfangen von Nachrichten

```
0   /**
     * Adds a MessageListener to a given Queue in order to receive TextMessages.
     * @param queueName name of the queue to bind the listener to
     * @param listener MessageListener with an implementation of onMessage()
     */
5   public void consume(MessageListener listener) throws JMSException

    /**
     * Receives a TextMessage from the given queueName.
     * @param queueName name of the queue to receive from
10   * @return Message from the queue. null indicates failure of receive, look
     *          at System.err and Logger file for further instructions.
     */
    public String consumeBlocking() throws JMSException

15  /**
     * Receives a TextMessage from the given queueName and returnes after
     * timeoutInMillies if nothing was received.
     * @param queueName name of the queue to receive from
     * @param timeoutInMillies timeout to wait before returning null
20   * @return Message from the queue. null indicates failure or timout of
     *          receive, look at System.err and Logger file for further
     *          instructions.
     */
    public String consumeBlockingWithTimeout(long timeoutInMillies) throws JMSException
```

Listing 2.3: Producermethode zum Versenden von Nachrichten

```
0   /**
     * Send a TextMessage, JSON encoded, with a producer for the given Queue.
```

```
 * @param queueName queue name
 * @param msg message content
 */
public void produce(String msg) {
```

LPSubscriber / LPPublisher

Zur Verteilung von Nachrichten über mehrere Empfänger und Sender wird im Living Place das Publisher/Subscriber-Pattern verwendet. Dieses Konzept wurde von der ActiveMQ-API weiter abstrahiert mit den Klassen „LPSubscriber“ und „LPPublisher“. Sollten sich mehrere Projekte für einen bestimmten Nachrichtentyp interessieren, müssen diese auf ein Topic „subscriben“. Die Verteilung der Informationen und deren Persistierung übernimmt der Wrapper. Auch hier können die Einstellungen über die „ConnectionSettings“ Klasse variiert werden.

Der „LPSubscriber“ unterstützt, ähnlich dem „LPConsumer“, drei Methoden, zwei blockierende und eine zum Einbinden eines MessageListeners, vlg. Listing 2.4.

Listing 2.4: Subscribermethoden zum Empfangen von Nachrichten

```
/**
 * Adds a MessageListener to a given Topic in order to receive TextMessages.
 * @param topicName name of the topic to bind the listener to
 * @param listener MessageListener with an implementation of onMessage()
 */
public void subscribe(MessageListener listener) throws JMSException

/**
 * Receives a TextMessage from the given topicName.
 * @param topicName name of the topic to receive from
 * @return Message from the topic. null indicates failure of receive, look
 *         at System.err and Logger file for further instructions.
 */
public String subscribeBlocking() throws JMSException

/**
 * Receives a TextMessage from the given topicName and returnes after
 * timeoutInMillies if nothing was received.
 * @param topicName name of the topic to receive from
 * @param timeoutInMillies timeout to wait before returning null
 * @return Message from the topic. null indicates failure or timeout of
 *         receive, look at System.err and Logger file for further
 *         instructions.
 */
public String subscribeBlockingWithTimeout(long timeoutInMillies) throws JMSException
```

Auch der „LPPublisher“ ähnelt wieder, um konsistent zu bleiben, dem „LPProducer“ und bietet damit den zweiten Nachrichten verteilenden Automatismus im LPActiveMQWrapper.

Listing 2.5: Publishermethoden zum Versenden von Nachrichten

```
/**
 * Send a TextMessage, JSON encoded, with a producer for the given Queue.
 * @param queueName queue name
 * @param msg message content
 */
public void publish(String msg)
```

Der Push-/ Pull-Mechanismus

Es gibt Anwendungen, bei denen es keinen Sinn macht, Nachrichten periodisch über ein Topic oder eine Queue im Living Place zu verteilen. Ein Beispiel dafür wäre der KalenderAgent, vgl. Barnkow (2010). Beim KalenderAgent ist es nicht sinnvoll alle Termine für einen festzulegenden Zeitraum periodisch zu publizieren. Bei dieser Anwendung ist es notwendig, dass Interessenten die Termindaten von dem KalenderAgent erfragen. Nach dem „Remote-Procedure-Call"-Verfahren haben wir uns entschieden einen solchen Mechanismus in den Wrapper einzubauen. Wir nennen diesen „Push- / Pull-Mechanismus". Der Interessent(„LPPushPullClient") „pusht" eine Aufforderung zum Senden einer Information an die Queue eines Servers(„LPPullPushServer") und „pullt" damit die Informationen auf ein Topic. Der Interessent kann seine Antwort anhand der eigenen ID identifizieren, denn diese sendet der Server beim Generieren der Antwort mit. Dies birgt zwei Vorteile. Erstens müssen Nachrichten nicht unnötig publiziert werden wenn keine Interessenten vorhanden sind und zweitens, können auch andere Anwendungen auf dem Antworttopic des Servers lauschen und eventuell nützliche Informationen weiter verarbeiten. Welche Requests der Server entgegennimmt und wie er auf diese reagiert, haben wir nicht vorgeschrieben und daher müssen diese Informationen von jedem „LPPullpushServer" selbst dokumentiert werden. Eine Seite die im Wiki[7] des Living Place Hamburg alle Nachrichten zentral dokumentiert, wird ständig erweitert und dient der Sammlung dieser Informationen.

Der LPActiveMQWrapper soll diesen Mechanismus möglichst einfach beiden Kommunikationsteilnehmern zur Verfügung stellen. Der Interessent(Client) erstellt sich dabei eine Instanz der Klasse „LPPushPullClient" und gibt dieser bei einer Instantiierung den Namen der Queue auf dem der Server Anfragen entgegennimmt und den Namen des Topics auf dem die Antwort gesendet wird mit. Nach der Instantiierung kann der Client Informationen vom Server „pullen", indem er seinen Request an diesen „pusht". Dazu übergibt er dem Wrapper seine eigene ID, damit dieser die Möglichkeit hat die Antwort auf seine Anfrage aus dem Nachrichtenstrom des Topics zu filtern, vgl. Listing 2.6. Die zweite Möglichkeit besteht darin, alle Nachrichten des Streams über einen „MessageListener" zu abonnieren, als würde man selbst nur einen Subscriber verwenden.

Listing 2.6: LPPushPullClient-Instantiierung und Methoden

```
0  LPPushPullClient client = new LPPushPullClient("queueName", "topicName");

   client.pushPullAndWaitForAnswer("myRequest01", id);
   client.pushPull("myRequest02", listener);
```

Der Entwickler, der Informationen per Request anbieten will, erstellt sich eine Instanz der Klasse „LPPullPushServer" und übergibt dieser auch die Namen von Topic und Queue. Mit

[7] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/ActiveMQ_Messages

diesen Informationen kann der Server beginnen über die pull-Methode Requests zu empfangen. Ist ein Request beim Server eingegangen, kann er anhand der Nachricht eine Antwort generieren und eventuelle Fehlermeldungen an den Client zurücksenden. Die generierte Antwort wird dem Client mit der push-Methode, über das bekannte Topic, zugesandt.

Listing 2.7: LPPullPushServer-Instantiierung und Methoden

```
0  LPPullPushServer server = new LPPullPushServer("queueName", "topicName");

   String  request = server. pull () ;
   //  procees request and generate answer...
   server.push(answer);
```

Für eventuelle Erweiterungen am LPActiveMQWrapper oder Verbesserungsvorschläge sind wir offen und freuen uns über eine Diskussion im Wiki[8].

2.3 Integration eines Indoor-Positioning-System in den LivingPlace

An der HAW Hamburg wird seit ca. zwei Jahren ein Echtzeit-Ortungssystem der Firma Ubisense[9] eingesetzt. Dieses System arbeitet mit sog. „Location-Tags“, die Ultrabreitband-Signale (UWB) an ein Netzwerk von Sensoren senden. Diese im Raum fest verankerten Sensoren verwenden daraufhin die empfangenen Signale, um die genaue Position des Tags zu ermitteln.
Das Lokationssystem von Ubisense wurde bisher in einem ca. 60m^2 großen, offenen Raum eingesetzt und hat sich in verschiedenen Szenarien bewährt. Aufgrund dieser positiven Erfahrungen mit dem System haben wir uns dazu entschieden, es in Zukunft im Living Place einzusetzen. An dieser Stelle war einführend zu analysieren, ob die bisher eingesetzte Anzahl von vier Sensoren im Wohnbereich ausreichen wird. Dabei haben uns zwei entscheidende Aspekte dazu bewogen zwei weitere Sensoren in der Wohnung einzusetzen. Zum einen weist der Living Place eine weitaus kompliziertere Bauweise als das bisherige Labor auf und bietet somit einige Hindernisse für die Sensoren. Ein weiterer Gesichtspunkt war die große Besucheranzahl die in Zukunft den Wohnbereich besichtigen wird. In diesem Zusammenhang könnte es zu einer Reduzierung der Genauigkeit bei der Positionserkennung kommen wenn sich viele Personen in der Wohnung aufhalten, da sich die Ausbreitungsgeschwindigkeit der elektromagnetischen Wellen des Ortungssystems durch Wasser enthaltene Materialien stark vermindert.
Im nächsten Schritt wurde untersucht, wie die sechs Sensoren in der Wohnung positioniert

[8] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Kommunikationsschnittstelle
[9] http://www.ubisense.net/en/

werden sollten, damit möglichst der gesamte Wohnbereich abgedeckt wird. Nach einigen Gesprächen haben wir beschlossen den in Abbildung 2.1 dargestellten Aufbau zu wählen.

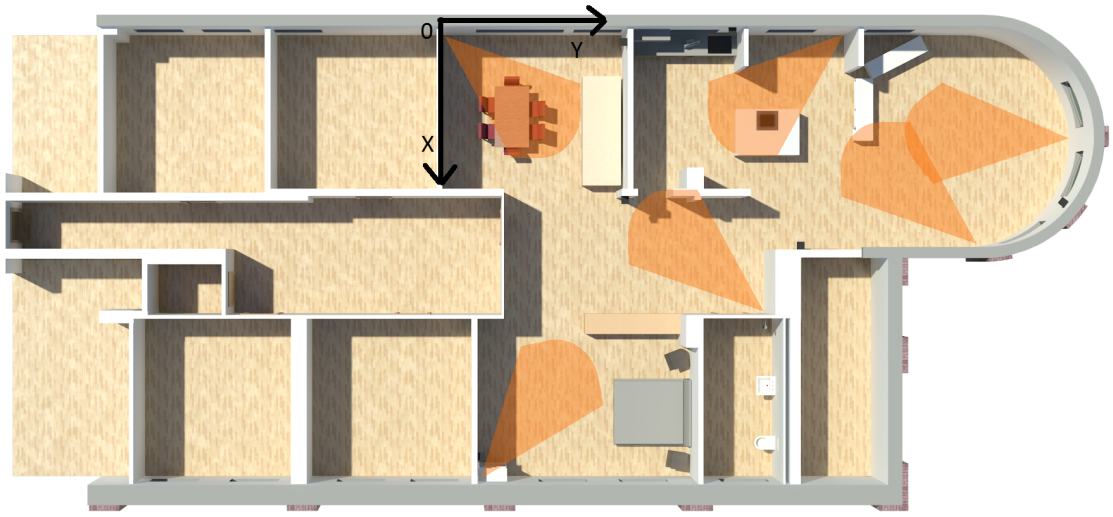

Abbildung 2.1: Übersicht der Positionen der sechs Ubisense Sensoren

Nach der Positionierung der Sensoren ist eine genaue Ausmessung der Koordinaten unerlässlich, da nach unseren Erfahrungen die Präzision des Systems erheblich leidet wenn die Angaben nicht exakt sind. In einem weiteren Schritt wurde nun eine sog. „Cell“ mit der von Ubisense bereitgestellten Software „Location Engine Config“ aufgebaut. Hiermit kann für jeden Sensor neben der X-, Y- und Z-Koordinate in Metern auch der Roll-Pitch-Yaw-Winkel in Grad eingegeben werden. Tabelle 2.3 zeigt beispielhaft wie die Daten für den Sensor in der Rundung aus Abbildung 2.1 gemessen wurden. In diesem Fall wurden die Winkel-Messungen während einer Kalibrierungsphase von der Software selbst gesetzt. Die Koordinaten mussten selbst eingetragen werden und beziehen sich auf einen selbst gewählten Nullpunkt, der unten links am Beginn des Wohnbereichs liegt, siehe Abbildung 2.1.

MAC-Adresse	X-Koordinate	Y-Koordinate	Z-Koordinate	Yaw	Pitch	Roll
00:11:CE:00:18:65	2,84	15,77	2,35	-90	-9,9	0

Tabelle 2.3: Eigenschaften eines Sensors

Nachdem die entsprechenden Messdaten eingetragen sind, muss nachfolgend ein Sensor als „Master“ und „Timing Source“ fungieren. Eine „Timing Source“ stellt die Quelle des Zeitsignals für eine „Cell“ dar. Ein „Master-Sensor“ berechnet die Postionen für eine „Cell“ und koordiniert ein Zeitmultiplexverfahren-Netzwerk (TDMA), was bedeutet, dass in bestimmten

Zeitabschnitten die Daten verschiedener Sender auf einem Kanal übertragen werden. In diesem Fall ist darunter zu verstehen, dass jeder Tag einen angemessenen Zeitplan zugewiesen bekommt, in dem er aktiv ist. In seiner aktiven Zeit sendet der Tag nun Nachrichten die seine Identität beinhalten sowie die UWB-Signale, welche von den Sensoren verwendet werden um die Position zu bestimmen, vgl. (Steggles und Gschwind, 2005). Für jeden Ubisense-Tag lässt sich der Zeitplan individuell an die Anforderungen der Applikation anpassen. So entsteht die Möglichkeit sich schnell bewegende Tags häufiger aktiv werden zu lassen als Tags die sich seltener bewegen. Die Kommunikation erfolgt dabei über einen bidirektionalen RF-Kanal. Ein wichtiger Aspekt in diesem Zusammenhang ist, dass es nur einen „Master“ und eine „Timing Source“ pro „Cell“ geben darf. In unserem Fall wurde ein Sensor ausgewählt, der sowohl die Aufgabe des „Masters“ als auch die der Zeitquelle übernimmt, was allerdings nicht zwingend erforderlich ist und von zwei verschiedenen Sensoren übernommen werden kann.
Da das Ortungssystem an der HAW Hamburg über ein Power over Ethernet (PoE) Verfahren mit Strom versorgt wird, ergibt sich folgend beschriebener Aufbau. Jeder Sensor wird jeweils mit einem Ethernet-Kabel mit dem PoE-Switch und dem „Master-Sensor“ verbunden. Eine Ausnahme bildet dabei nur der „Master-Sensor“ selbst, der lediglich eine Verbindung zu PoE-Switch benötigt. Damit das System in Betrieb genommen werden kann, muss ein DHCP-Server im Netz aktiv sein, damit jeder Sensor eine gültige IP-Adresse erhält und am Netzwerk teilnehmen kann. Sollte jeder Sensor eine grüne Status-LED anzeigen kann das Indoor-Positioning-System die Tags orten und deren Position für weitere Anwendungen zur Verfügung stellen.

2.4 Szenario - Location-based Screen im Living Place

Das Szenario des lokationsbasierten Bildschirms wurde bereits im Projekt 1 für eine Laborumgebung implementiert. Dieser Anwendungsfall beschreibt eine Situation, in der eine Person ihre aktuelle Position innerhalb der Wohnung verlässt, mit der Absicht für eine gewisse Zeit einer anderen Tätigkeit nachzugehen oder um beispielsweise einen Gegenstand für ihre aktuelle Aktivität zu holen. In einem solchen Fall ist es denkbar, dass die Person zwar einen bestimmten Bereich verlässt, jedoch ihre derzeitige Videokonferenz oder den aktuellen Film nicht unterbrechen möchte. Speziell in einer Loftwohnung, in der keine einzelnen Zimmer existieren, sondern Bereiche wie Küche und Arbeitsbereich ineinander übergehen, kann der Fall häufig eintreten, dass durch eine Änderung des Standorts ein anderes Anzeigegerät besser zu sehen ist als das vorherige. Der Living Place bietet aufgrund seiner Bauweise und seinen zwei großen Bildschirmen einen idealen Rahmen für dieses Szenario. Die Fernseher wurden dabei so in der Wohnung aufgestellt, dass sich einer im Schlafbereich und ein anderer im Loungebereich befindet.

Realisierung

Der Living Place besitzt zwei 3D-Fernseher der Firma Samsung mit einer Bildschirmdiagonalen 46"(116 cm). Diese beiden Bildschirme sollen jeweils nur dann das aktuelle Bild zeigen, wenn sich die Person in dessen Bereich befindet.
Zur Positionsbestimmung des Bewohners soll in diesem Szenario das Ubisense-Ortungssystem eingesetzt werden. Zu diesem Zweck wurde eine Applikation geschrieben, welche die Informationen des Indoor-Location-Systems entgegennimmt, interpretiert und die Schlussfolgerung über den ActiveMQ für weitere Komponenten zur Verfügung stellt. Diese Anwendung wurde in der Sprache C# entwickelt, da der Hersteller zur Ansteuerung des Ubisense-Systems „Dynamic Link Library“-Dateien (DLL) anbietet. Im ersten Schritt verbindet sich dieses Programm zur Kommunikationsschnitstelle ActiveMQ und zur angegebenen Ubisense „Cell“. Nach einer erfolgreichen Verbindung mit beiden Systemen werden nun die Positionssignale empfangen. Zur Interpretation dieser Daten wurde der Living Place in drei Bereiche aufgeteilt (Norden, Mitte, Süden). Hierzu wurde vom Nullpunkt aus nur die Y-Koordinate betrachtet, da in diesem Szenario unterschieden werden soll, ob die Person sich vom Nullpunkt aus gesehen im nördlichen oder südlichen Teil der Wohnung aufhält, siehe Abbildung 2.1. Die Tabelle 2.4 zeigt auf, ab welchem Y-Wert sich der Bewohner in welchen Teilbereich aufhält.

Bereich	Y-Koordinate
Norden	10.0 < y < 15.80
Mitte	5.50 < y < 10.0
Süden	0.0 < y < 5.50

Tabelle 2.4: Aufteilung des Living Place in Abhängigkeit zur Y-Koordinate

Sobald der Bewohner seinen aktuellen Standort ändert, wird die neue Koordinate von unserer Anwendung untersucht. Lediglich wenn die neue Position einen Wechsel des Bereichs bedeutet, wird das Versenden einer entsprechenden Nachricht über ein ActiveMQ Topic mit Namen „UbiTV“ initiiert. Sollte die Person im gleichen Bereich bleiben hat dieses für das Szenario keine Auswirkung, weshalb eine Nachricht hier ausbleibt.
Die über den AcitveMQ versendete JSON-Nachricht entspricht dabei dem gewünschten Nachrichtenformat aus dem Projekt 1[10]. Dies bedeutet, sie enthält die aktuelle Versionsummer der Anwendung (1.0), eine eindeutige ID (ClientId beim ActiveMQ + Zufallszahl) und den Teilbereich (z.B. „north“). Mit dem Ziel die Anwendung während ihrer Ausführung beobachten zu können, wurde eine GUI geschrieben, welche den Verbindungsstatus, die aktuellen Positionsdaten und die zuletzt gesendete Nachricht anzeigt. Eine Ansicht dieser GUI wird in der Abbildung 2.2 veranschaulicht.

[10] http://livingplace.informatik.haw-hamburg.de/wiki/index.php/Nachrichtenformat

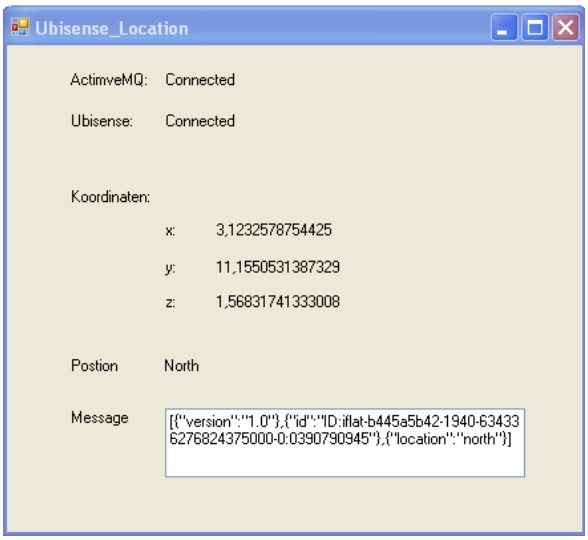

Abbildung 2.2: „LocationGUI“ für den Location-based Screen

Neben der oben beschriebenen Anwendung wurde eine zweite Applikation entwickelt, welche die über den ActiveMQ versendeten Nachrichten verarbeitet. Dieses Programm bietet die Möglichkeit, nach dem Start zu wählen in welchem Teilbereich es sich befindet. Hierdurch lassen sich neue Bildschirme in das bestehende System einbinden ohne den Code zu ändern. Diese Anwendung meldet sich nun am „UbiTV“-Topic an und analysiert alle ankommenden Nachrichten. Sollte der in der Nachricht enthaltene Standort dem beim Start gewählten Bereich gleichen, beginnt das Programm mit der Darstellung des aktuellen Programms.
In einführenden Gesprächen war es geplant die Fernseher ein-und auszuschalten. Allerdings würde es hier bei jedem Wechsel des Bereichs zu einer Verzögerung bei der Darstellung des Bildes kommen, da die Fernseher einige Sekunden benötigen um zu starten. Deshalb haben wir uns dazu entschieden dieses Szenario vorerst weiterhin auf das Fernsehbild, welches über DVB-T im VLC-Player[11] angezeigt wird, zu beschränken. Dabei wird das Bild nicht wie in der Laborumgebung von einem Rechner als Stream bereitgestellt, sondern vom „Elgato EyeTV Netstream DTT“[12]. Dieser Netzwerk Dual-Tuner erlaubt es unverschlüsseltes DVB-T an verschiedene PC’s zu streamen. Diese Komponente muss nicht an einen Computer angeschlossen werden, was den Vorteil birgt, dass dieses Gerät im Hintergrund laufen kann und man somit keinen zusätzlichen PC benötigt der das Fernsehbild in das Netzwerk einspeist, was die Fehleranfälligkeit des Systems reduziert.

[11] http://www.videolan.org/vlc/
[12] http://www.elgato.com/elgato/int/mainmenu/products/tuner/netstreamdtt/product1.de.html

3 Zusammenfassung

Im Projekt 1 wurde deutlich, dass der EventHeap den Anforderungen des Living Place Hamburg nicht genügen wird. Wir haben uns nach ausführlicher Recherche für den ActiveMQ als Kommunikationsmittelpunkt entschieden. Bestehende Anwendungen wurden auf den ActiveMQ übertragen und dieser in der Praxis getestet. Des Weiteren wurde ein einheitliches Nachrichtenformat(JSON) eingeführt, um allen Beteiligten eine gemeinsame, sprachunabhängige Form der Nachrichtenübertragung zu bieten. Eine Architektur für den zentralen Kommunikationspunkt wurde erarbeitet und mit weiteren Abstraktionsleveln verfeinert. Die Basis der Architektur, wie sie im Projekt 1 ausgearbeitet wurde, konnten wir nun im Projekt 2 fertig stellen und bieten eventuellen Nachfolgern damit eine Arbeitsgrundlage.

Die Persistenzebene wurde erneut auf Praxistauglichkeit geprüft und gegen andere Lösungen verglichen. Anhand dieser Untersuchungen haben wir uns gegen die CouchDB entschieden, vgl. Anderson u. a. (2010). Die nun integrierte Lösung mit der MongoDB wurde ausführlich getestet und genügt den Vorstellungen hinsichtlich Performance und Ausfallsicherheit.

Im Projekt 2 wurde ein Wrapper entwickelt, der den Projekten im Living Place Hamburg den Kommunikationsaufwand abnimmt, sodass die Entwickler sich auf das Gestalten und das Verarbeiten ihrer Nachrichten konzentrieren können. Der Wrapper ermöglicht dem Entwickler mit der Instantiierung einzelner Klassen[1] die volle Kommunikation seiner Applikation mit dem Living Place. Ein zusätzlicher Mechanismus um Ergebnisse von bestimmten Anwendungen anzufordern wurde zur Verfügung gestellt, vgl. 2.2.2. Der LPActiveMQWrapper ist nun zur Verwendung im Subversion[2] bereitgestellt und kann von allen Entwicklern genutzt werden, um im Living Place Hamburg zu kommunizieren.

Das Indoor-Positioning-System der Firma Ubisense wurde aus der alten Testumgebung erstmals in den Living Place übertragen und getestet. Die Ergebnisse, die im Living Place erzielt wurden, konnten nach der Parametrisierung des Systems mit den Gegebenheiten der Wandstellung zu einer relativ genauen Lokation der Locationtags geführt werden. Die Wandstellung wurde zunächst nur grob in den Umgebungsplan eingepflegt und wird nach endgültiger Installation des Systems mit Wandhalterungen genau vermessen. Dennoch haben wir die

[1] LPSubscriber, LPPublisher, LPConsumer, LPProducer, LPPullPushServer, LPPushPullClient
[2] http://livingplace.informatik.haw-hamburg.de/svn/middleware/LivingPlaceMessaging/

Positionierung der Ubisense Sensoren untersucht und für jeden Sensor eine geeignete Position erarbeitet. Auch die vom Hersteller angegebene Genauigkeit von 15 cm kann unserer Ansicht nach erreicht werden, vgl. Steggles und Gschwind (2005). Die für das Publizieren der Positionsdaten verantwortliche Software wurde so angepasst, dass sie die Positionen des Tags mit Metainformationen, wie einem Bereich in dem es sich befindet, anreichert. Mit diesen Metainformationen konnte auch das Szenario des Location-Based Screen übertragen werden.

Das Szenario des Location-Based Screen wurde im Living Place auf den neuen Architekturkomponenten und dem dort testweise installierten Ubisense einsatzfähig gemacht. Damit konnte erstmals das Zusammenspiel aller Komponenten in realistischer Umgebung gezeigt werden. Das Szenario ist mit den im Living Place vorgesehen Bildschirmen und der neuen Empfangstechnik(Elgato EyeTV Netstream DTT) realisiert worden und alle dazu nötigen Softwarekomponenten konnten an die neue Umgebung angepasst werden.

Ausblick

Der Ausblick für das Projekt 2 gibt eine Übersicht über die aus dem Projekt 1 umgesetzten Ideen und zeigt Überlegungen für Erweiterungen zukünftiger Nachfolger auf.

Im Ausblick vom Projektbericht 1 haben wir verschiedene Erweiterungen vorgestellt die wir teilweise im Projekt 2 schon realisieren konnten. Der ActiveMQ kann nun in den Living Place verlagert und fest in die Netzwerkinfrastruktur integriert werden. Zunächst wird der ActiveMQ auf einem MacMini installiert der im Kontrollraum an die Stelle geeigneter Hardware positioniert wird. Dieser sollte, sobald vorhanden, mit einem Serverrack ausgetauscht werden. Die Planung für einen „Tunnel“, der den Living Place Hamburg und die Räume 10.80 und 10.81 des Berliner Tor 7 verbindet, sind bereits abgeschlossen, sodass Entwickler der Labore im 10. Stock bald auch den im Living Place integrierten ActiveMQ nutzen können.

Um die realisierten Erweiterungen fertig zu stellen, wollen wir in den folgenden Wochen das Ubisense fest im Living Place installieren und vermessen, um eine möglichst hohe Genauigkeit der Sensorik zu gewährleisten. Die Kabel des Ubisense werden in verzweigenden Kabelsträngen im Unterboden verlegt und mit einem Verteilerswitch ein weiterer Anschlusspunkt für andere Projekte erschlossen. Um die Positionierung von Ubisense Tags abzuschließen werden wir in Zusammenarbeit mit Bastian Karstaedt, vgl. Karstaedt (2010), seine „Location Provider“-Anwendung in Betrieb nehmen.

Literaturverzeichnis

[Abowd u. a. 1999] ABOWD, Gregory D. ; DEY, Anind K. ; BROWN, Peter J. ; DAVIES, Nigel ; SMITH, Mark ; STEGGLES, Pete: Towards a Better Understanding of Context and Context-Awareness. In: *Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*. London, UK : Springer-Verlag, 1999 (HUC '99), S. 304–307. – URL http://portal.acm.org/citation.cfm?id=647985.743843. – ISBN 3-540-66550-1

[Anderson u. a. 2010] ANDERSON, J. C. ; LEHNARDT, Jan ; SLATER, Noah: *CouchDB: The Definitive Guide Time to Relax*. 1st. O'Reilly Media, Inc., 2010. – ISBN 0596155891, 9780596155896

[Barnkow 2010] BARNKOW, Lorenz: Eine Multitouch-fähige Küchentheke: Im Kontext des Living Place Hamburg. (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master09-10-aw1/barnkow/bericht.pdf

[Karstaedt 2010] KARSTAEDT, Bastian: Entwicklung von *Indoor Spatial Services* für Smart Homes auf Basis der Industry Foundation Classes. (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2011-proj2/karstaedt.pdf

[Otto und Voskuhl 2010] OTTO, Kjell ; VOSKUHL, Soeren: Entwicklung einer Architektur für den Living Place Hamburg. (2010). – URL http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2010-proj1/otto_voskuhl.pdf

[Steggles und Gschwind 2005] STEGGLES, Pete ; GSCHWIND, Stephan: THE UBISENSE SMART SPACE PLATFORM. (2005). – URL http://www.pervasive.ifi.lmu.de/adjunct-proceedings/demo/p073-076.pdf

[The Apache Software Foundation 2010] THE APACHE SOFTWARE FOUNDATION: Apache ActiveMQ Architecture. (2010). – URL http://activemq.apache.org/code-overview.html

[Weiser 1991] WEISER, Mark: *The Computer for the 21st Century*. 1991. – URL http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html