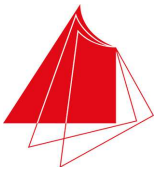


Hochschule Karlsruhe - Technik und Wirtschaft

Fakultät für Informatik und Wirtschaftsinformatik
Fachgebiet Wirtschaftsinformatik

BACHELORTHESIS

Konzeption und Entwicklung eines Szenarioassistenten für ein heterogenes Verbundsystem in der Bildaufklärung

von	Herrn Andre Feuerbach
Arbeitsplatz	Fraunhofer IOSB, Karlsruhe
Erstbetreuer	Prof. Dr. Manfred Seifert
Zweitbetreuer	Prof. Dr. Jürgen Zimmermann
Abgabetermin	31.12.2012

Karlsruhe, 31.12.2012

Vorsitzender des
Prüfungsausschusses

Andre Feuerbach
Marienstr. 61
76137 Karlsruhe

Hiermit versichere ich, dass ich die vorliegende Bachelorthesis selbstständig und ausschließlich unter Verwendung der angegebenen Quellen und sonstigen Hilfsmittel verfasst habe.

Die Arbeit wurde in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde zur Erlangung eines akademischen Grades vorgelegt.

Karlsruhe, den 31.Dezember 2012

Andre Feuerbach

Danksagung

Zunächst möchte ich mich bei Herrn Prof. Dr. Manfred Seifert für die Betreuung dieser Arbeit bedanken. Sein stets hilfreiches Feedback war sehr wertvoll für mich.

Ein ganz besonderer Dank geht an Frank Wiedermann. Neben dem Korrekturlesen möchte ich mich bei Ihm vor allem für die Gespräche und die Unterstützung in der Endphase dieser Arbeit bedanken.

Ein großer Dank für das Korrekturlesen dieser Arbeit geht auch an Marc Hähnel.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ABUL	Automatisierte Bildauswertung am Beispiel UAV Luna
AMFIS	Aufklärung mit mobilen und ortsfesten Sensoren im Verbund
BOSH	Bidirectional-streams Over Synchronous HTTP
CSD	Coalition Shared Data Server
ExBa	Experimentalsystem Bildaufklärung
FORBOT	Forschungs Roboter Plattform
JAXB	Java Architecture for XML Binding
JDBC	Java Database Connectivity
JID	Jabber Identifier
JMS	Java Message Service
JMX	Java Management Extensions
MOM	Message Oriented Middleware
UAV	Unmanned Aerial Vehicle
XEP	XMPP Extension Protocol
XMPP	Extensible Messaging and Presence Protocol

Listingsverzeichnis

1 Einführung

In diesem Kapitel wird eine kurze Einführung in das Thema dieser Arbeit gegeben. Neben der Motivation wird die Aufgabenstellung gezeigt. Mit der Beschreibung des Aufbaus dieser Arbeit schließt dieses Kapitel.

1.1 Motivation

Die Aufklärung und Überwachung in zivilen und militärischen Bereichen muss oftmals in einem Verbund von unterschiedlichen Systemen durchgeführt werden. Das Experimentalsystem Bildaufklärung (ExBa) des Fraunhofer-Instituts für Optronik, Systemtechnik und Bildauswertung IOSB besteht aus unterschiedlichen, komplexen Systemen, welche Unterstützung bei der Lösung von verschiedenen Aufgaben und Problemen in Bereichen der Bildauswertung bieten. Das Gebiet der Bildauswertung am IOSB umfasst die automatisierte und interaktive Gewinnung von Informationen aus Videoaufnahmen und Bildern sowie deren Aufbereitung und Verarbeitung.

Bei der Durchführung von Demonstrationsszenarien für das ExBa werden unterschiedliche Komponenten des Systems gemeinsam genutzt. Neben Robotern, die in der Luft, an Land oder unter Wasser agieren können werden auch stationäre Kameras zur Überwachung eines Gebietes oder Leuchttische zur Visualisierung von Informationen eingesetzt. Um gemeinsam Aufgaben in einem Demonstrationsszenario durchführen zu können, muss dabei jedes Teilsystem für jedes Demonstrationsszenario einzeln konfiguriert und in Betrieb genommen werden. Dies stellt nicht nur einen zeitlichen Aufwand dar. Da die verschiedenen Systeme unterschiedliche Eigenschaften besitzen und auf verschiedenen Technologien basieren wird auch individuelles Wissen über die Konfiguration und Inbetriebnahme der einzelnen Teilsysteme benötigt.

Die Aufgaben, Konfiguration und Inbetriebnahme der Systeme können durch den Einsatz eines geeigneten, mobilen Assistenzsystems vereinfacht werden indem das Assistenzsystem den Anwender bei der Ausführung dieser Aufgaben unterstützt. Anstatt jede Komponente immer wieder einzeln und aufwendig für ein Demonstrationsszenario zu konfigurieren und in Betrieb zu nehmen können diese Vorgänge automatisiert werden.

1.2 Aufgabenstellung

In dieser Arbeit soll eine Android-App für Smartphones und Tablet-PCs entwickelt werden. Dieser mobile Szenarioassistent soll dem Nutzer die Möglichkeit bieten Konfigurationsparameter für die gemeinsame Ausführung eines Demonstrationsszenarios in die Systemkomponenten eines heterogenen Verbundsystems für die Bildaufklärung einzustellen. Dazu sollen die Konfigurationsparameter in Form von Demonstrationsszenarien in der mobilen Anwendung gespeichert

werden und bei Bedarf per Knopfdruck - wie bei einer Fernbedienung - auf die Systemkomponenten des Verbundsystems übertragen werden können. Somit kann vor der Ausführung eines Demonstrationsszenarios eine aufwendige Konfiguration der unterschiedlichen am Demonstrationsszenario beteiligten Systemkomponenten entfallen.

Dem Anwender sollen die Demonstrationsszenarien und die hinterlegten Konfigurationsparameter ansprechend präsentiert werden. Die Kommunikation mit den Teilsystemen des Verbundsystems ExBa soll über WLAN und das Extensible Messaging and Presence Protocol (XMPP) erfolgen. Die Benutzungsoberfläche soll HTML5-basiert und leicht erweiterbar sein. Hierzu soll ein HTML5-Framework für die Erstellung der Oberfläche verwendet werden.

Außerdem soll im Rahmen dieser Arbeit eine Agentenanwendung, die für die Einstellung der Konfigurationsparameter in die Systemkomponenten zuständig ist, entwickelt werden. Diese Agentenanwendung wird auf den Systemkomponenten bzw. Zielplattformen gestartet und reagiert auf XMPP-Nachrichten, welche die Szenariokonfigurationsparameter enthalten. Der Begriff des Agenten wird in dieser Arbeit, wie beim Simple Network Management Protocol (SNMP), im Sinne einer Netzwerkkomponente verstanden, die auf einem entfernten Knoten im Netzwerk ausgeführt wird und von einer zentralen Stelle aus gesteuert werden kann. Die Zielplattformen des Verbundsystems stellen hierbei die entfernten Netzwerkknoten dar. Die Fernsteuerung der Agentenanwendung erfolgt durch den Szenarioassistenten. Neben diesen Teilaufgaben soll ein Demonstrationsszenario und der Austausch von Nachrichten zwischen den Teilsystemen konzipiert und implementiert werden. Anhand der prototypischen Implementierung der Anwendungen soll in dieser Arbeit die Machbarkeit aufgezeigt werden.

1.3 Aufbau der Arbeit

Das zweite Kapitel soll die Grundlagen für das Verständnis dieser Arbeit schaffen. Hierbei wird zunächst beschrieben, was unter dem Begriff Assistenzsysteme zu verstehen ist. Danach werden das ExBa mit seinen Systemkomponenten und verwandte Arbeiten vorgestellt. Im Unterkapitel Kommunikation werden Technologien aufgezeigt, die in dieser Arbeit für den Austausch von Nachrichten eingesetzt werden. Das Kapitel schließt mit dem Unterkapitel Mobile App, in dem ebenfalls Technologien, die bei der Entwicklung des mobilen Szenarioassistenten Verwendung finden, vorgestellt werden.

Im dritten Kapitel wird eine Anforderungsanalyse, welche die Basis für die Erarbeitung der weiteren Konzepte darstellt, durchgeführt. Zunächst wird das Vorgehen zur Ermittlung der Anforderungen beschrieben. Danach wird ein Überblick über die Anwender der Systeme gegeben, um anschließend die Systemarchitektur aufzuzeigen. Nach einer Schnittstellenbeschreibung der Zielplattformen folgt ein Unterkapitel, in dem relevante Fakten und Annahmen für die Konzeption und Implementierung erläutert werden. Abschließend werden ausgewählte, wichtige Anwendungsfälle und Anforderungen dargelegt.

Das vierte Kapitel beschäftigt sich mit den Konzepten, auf denen die Implementierung für den Szenarioassistenten und die Agentenanwendung beruht. Dazu erfolgt zunächst ein Überblick über den Systemaufbau, um danach auf das Konzept für die Kommunikation zwischen den verschiedenen Systemen einzugehen. Nachdem in diesem Kapitel auch die Konzeption des Demonstrationsszenarios stattfindet, werden die Konzepte für die Implementierung des Szenarioassistenten und der Agentenanwendung vorgestellt.

Nachdem im vorangegangenen Kapitel die Konzeption beschrieben wurde, erfolgt im Kapitel 5 die prototypische Implementierung. Zunächst soll in diesem Kapitel die Grundlage für die Implementierung des Szenarioassistenten und der Agentenanwendung geschaffen werden. Dazu wird das Demonstrationsszenario umgesetzt, das vom Szenarioassistenten an die Agentenanwendung gesendet wird. Anschließend wird die Implementierung des Szenarioassistenten aufgezeigt. Das Kapitel endet mit der Implementierung der Agentenanwendung.

Im sechsten Kapitel findet eine Evaluation der Implementierung statt. Hierfür wird zunächst das Ziel und der Aufbau der Evaluation beschrieben. Im Folgenden werden der Ablauf und die Kriterien festgelegt, um dann das Ergebnis vorzustellen.

Im siebten und letzten Kapitel dieser Arbeit werden die erzielten Ergebnisse zusammengefasst. Abschließend folgt ein Ausblick, der als Grundlage für weitere Arbeiten oder mögliche Erweiterungen zu diesem Thema dienen soll.

2 Grundlagen

In diesem Kapitel sollen die Grundlagen für das Verständnis dieser Arbeit geschaffen werden. Hierbei wird zunächst beschrieben was unter dem Begriff Assistenzsysteme zu verstehen ist. Danach werden das ExBa mit seinen Systemkomponenten und verwandte Arbeiten vorgestellt. Im Unterkapitel Kommunikation werden Technologien aufgezeigt, die in dieser Arbeit, für den Austausch von Nachrichten eingesetzt werden. Das Kapitel schließt mit dem Unterkapitel Mobile App in dem weitere Technologien, die bei der Entwicklung des mobilen Szenarioassistenten Verwendung finden, vorgestellt werden.

2.1 Assistenzsysteme

Um Menschen in bestimmten Situationen, wie beispielsweise bei der Verwendung von Produkten oder bei der Bearbeitung von Aufgaben, zu unterstützen werden heutzutage in vielen Bereichen Assistenzsysteme eingesetzt. Auf dem Gebiet der Fahrerassistenzsysteme wird der Mensch beispielsweise durch Abstandswarner, Bremshilfen oder Einparkhilfen unterstützt. Werden mobile Geräte, wie Smartphones oder Tablet-PCs, an Assistenzsysteme angebunden, spricht man von mobilen Assistenzsystemen. Mobile Assistenzsysteme existieren unter anderem in den Bereichen Medizintechnik, Hausautomation oder in der Tourismusbranche.

2.1.1 Experimentalsystem Bildaufklärung

Das ExBa (Abbildung 2.1) bietet Unterstützung für die Lösung von Aufgaben und Problemen im Bereich der Bildauswertung. Dabei besteht das System aus den Systemkomponenten Coalition Shared Data Server (CSD), Automatisierte Bildauswertung am Beispiel UAV Luna (ABUL), Aufklärung mit mobilen und ortsfesten Sensoren im Verbund (AMFIS) und dem Digitalen Lagetisch. Nachfolgend sollen diese verschiedenen Systemkomponenten des Verbundsystems kurz erläutert werden [Fra12].

CSD

Die Systemkomponente CSD stellt eine Datenbasis für die interoperablen Systeme dar. Die Mechanismen für den Zugriff, als auch das zugrundeliegende Datenmodell und die Schnittstellen dieser Datenbank basieren auf Standards. Die wesentlichen Funktionen und Eigenschaften dieser ExBa-Systemkomponente sind die Verwaltung, Abfrage, Überprüfung, Sicherung, Protokollierung und Synchronisierung von Daten im Verbundsystem. Wobei unter Daten Bilder, Videoaufnahmen, Meldungen und Lageinformationen zu verstehen sind.

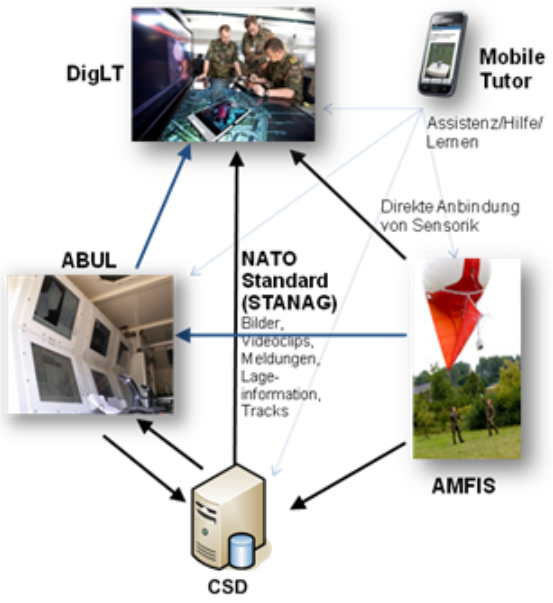


Abbildung 2.1: Das Experimentalsystem Bildaufklärung

ABUL

Die Hauptaufgabe der Integrationsplattform ABUL ist die Überwachung und Aufklärung im Bereich der Sicherheit. Darüber hinaus ist die Unterstützung von Bildauswertern bei länger dauernden Aufgaben im Bereich von Beobachtungs- und Auswertungsaufgaben vorgesehen. Dazu wird die Drohne Luna, die auch bei der Bundeswehr eingesetzt wird, für die Lieferung von Bilddaten genutzt.

AMFIS

Das System AMFIS ist, neben dem Digitalen Lagetisch, eine der ExBa-Systemkomponenten, die als Zielplattform für die Einstellung von Konfigurationsparametern dienen soll. Das System ermöglicht die mobile generische Aufklärung und Überwachung mit Hilfe von Luft-, Land- und Wasserfahrzeugen im Sensornetzwerkverbund. Die zugehörige mobile AMFIS-Bodenkontrollstation (Abbildung 2.2) ist eine Benutzungsschnittstelle und Datenintegrationsplattform für verschiedene stationäre und mobile Sensorträger und deren Sensoren [Bür11].

Ein Sensorträger ist neben Quadrocoptern, Heliumballons oder Domkameras, die Forschungs Roboter Plattform (FORBOT). Der FORBOT (Abbildung 2.3) ist ein sechsrädriger, geländegängiger Landroboter [Rob12]. Zusammen mit den ortsfesten Domkameras Axis-Dome 1 und Axis-Dome 2 stellt der FORBOT die Zielsysteme der AMFIS-Bodenkontrollstation dar, für die im Rahmen dieser Arbeit eine Parametereinstellung erfolgen soll. Die Domkameras sind Netzwerk-Kameras, die oftmals in Videoschutzsystemen, wie sie beispielsweise Banken oder Tankstellen besitzen, eingesetzt werden.

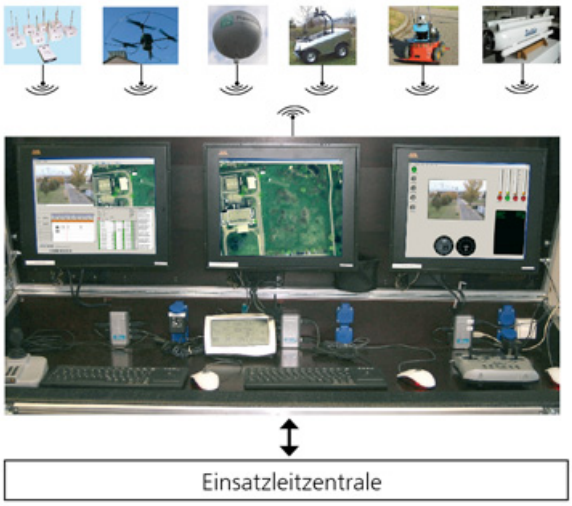


Abbildung 2.2: Die AMFIS-Bodenkontrollstation

Sensorträger können mit verschiedenen Sensoren ausgestattet werden. So besitzt der FORBOT beispielsweise einen GPS-Sensor mit dem die Position des FORBOTs in einem Einsatzgebiet bestimmt werden kann. Weitere Beispiele für Sensoren, mit denen ein Sensorträger ausgestattet werden kann, sind Magnetometer, Barometer und Thermometer.

Die AMFIS-Bodenkontrollstation besteht unter anderem aus dem Piloten-Arbeitsplatz und einer Anwendung für die Lagedarstellung. Der Piloten-Arbeitsplatz ermöglicht die Steuerung der verschiedenen Sensorträger, während die Lagedarstellung eine Übersicht des Einsatzgebiets in Form einer Lagekarte anzeigt. Die Lagekarte zeigt in verkleinerter Darstellung einen Überblick über die Positionen der Sensorträger in einem Einsatzgebiet.

Die typischen Aufgaben und Einsatzbereiche des AMFIS-Systems sind:

- Einsatz bei Naturkatastrophen
- Aufklärung im urbanen Umfeld
- Militärischer Einsatz
- Bewachung von Innenräumen (Messehallen, Konferenzen) und Großveranstaltungen
- Lokalisieren und Identifizieren von Personen und Fahrzeugen

Abbildung 2.3: Der Landroboter FORBOT

Digitaler Lagetisch

Der Digitale Lagetisch (Abbildung 2.4) ist ein Multi-Display-Arbeitsplatz, der die interdisziplinäre und kooperative Bearbeitung von Aufgaben im Team unterstützt, welche unterschiedliche Sichten auf ein geografisches Gebiet erfordern. Mit Hilfe des Digitalen Lagetisches kann ein Team in Krisensituationen eine größere grafische Region auf einen Blick analysieren und entsprechende Aktivitäten planen. Der Digitale Lagetisch kann mit Tablet-PCs oder Handgesten bedient werden. Für diese ExBa-Systemkomponente soll ebenfalls eine Parametereinstellung durch den Szenarioassistenten erfolgen.

Typische Aufgaben des Digitalen Lagetisches sind:

- Lageanalyse und Lagedarstellung im Katastrophenfall
- Abruf von Informationen über Verkehr, Wetterlage und Positionen von Einsatzfahrzeugen
- Detailanalyse von Interessensbereichen

Abbildung 2.4: Der Digitale Lagetisch

2.1.2 Verwandte Arbeiten

Manashty *et al.* [Man10] stellen einen Ansatz zur Fernsteuerung einer Smart-Home-Umgebung durch ein Smartphone vor. In diesem mobilen Assistenzsystem werden keine kompletten Konfigurationen für alle angeschlossenen Geräte übertragen, sondern Aufgaben für die einzelnen Geräte durch die Smartphone-Anwendung definiert. Eine Ausführung dieser Aufgaben kann sofort oder zeitgesteuert erfolgen. Außerdem können Regeln für das System festgelegt werden. Diese Regeln koppeln auszuführende Aktionen an Bedingungen. Tritt eine Bedingung ein, werden die entsprechenden Aufgaben bzw. Aktionen von den Geräten ausgeführt.

Die Audi AG bietet mit Ihrem Audi Konfigurator Deutschland ein mobiles Assistenzsystem für die Erstellung einer persönlichen Neuwagenkonfiguration an [Aud12]. Die Anwendung ist für die mobilen Betriebssysteme Android und iOS erhältlich. Der Anwender kann die Konfigurationsparameter in der App abspeichern und an einen Audi-Händler übertragen. Dazu wird in der App ein sogenannter Audi-Code erzeugt, über den der Audi-Händler die zuvor vom Kunden gespeicherte Konfiguration in sein System zur weiteren Bearbeitung laden kann (Abbildung 2.5).

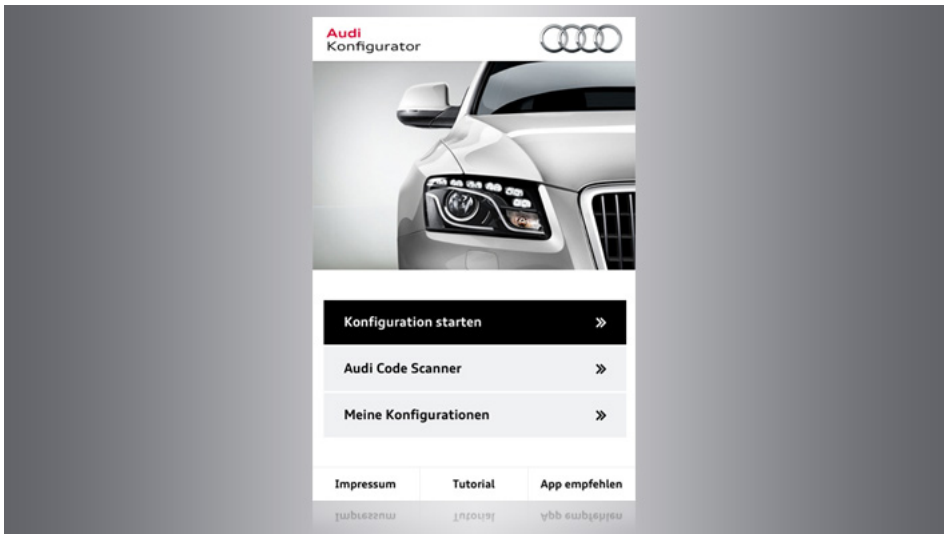


Abbildung 2.5: Audi Konfigurator Deutschland

Die Firma Laerdal entwickelte einen mobilen Szenarioassistenten für die medizinische Ausbildung im Rahmen der Notfallmedizin und der Krankenpflege [Lae12]. VitalSim stellt einen Multiparameter-Simulator für Vitalfunktionen dar und besteht aus einer Basiseinheit und einer mobilen Steuereinheit (Abbildung 2.6). Zusätzlich existiert eine Software mit Szenario- und Berichtsfunktionen.

Die Basiseinheit ist mit Trainingsmodellen, sogenannten Ganzkörper-Patientensimulatoren, verbunden und führt Simulationsszenarien aus (Abbildung 2.7). Die Steuereinheit ist über Funk mit der Basiseinheit verbunden und ermöglicht auf Knopfdruck die Ausführung von Szenariokonfigurationen durch die Basiseinheit.

Szenarien können mit der mitgelieferten Software erstellt werden. Weiterhin können fertige Szenariokonfigurationen für verschiedene Bereiche, wie z.B. Krankenpflegeschulen, Medizinische Hochschulen, Militär und Rettungswesen, bezogen werden.

Die Szenariofunktion des Systems enthält:

- vorprogrammierte Szenarien zur Vereinfachung des Systembetriebs und der Simulationen
- eine Software mit der Szenarien erstellt werden können
- die Möglichkeit, bis zu zehn Szenarien in der Basiseinheit zu speichern

Die Szenarien sind nicht in der Fernsteuerung gespeichert, sondern werden mit einer Software über eine USB-Schnittstelle auf die Basiseinheit übertragen.

Abbildung 2.6: VitalSim Basis- und Steuereinheit

Abbildung 2.7: VitalSim Basiseinheit verbunden mit Trainingsmodell

Mittlerweile existiert das Nachfolgeprodukt SimPad, welches abwärtskompatibel zu VitalSim ist (Abbildung 2.8) und zwei verschiedene Betriebsmodi besitzt. Im automatischen Modus werden vordefinierte Szenarien genutzt und ausgewählt (Abbildung 2.9). Im manuellen Modus erfolgt die Steuerung von Szenarien anhand der Änderung von Parametern im Simulator durch den Anwender selbst.

Abbildung 2.8: Nachfolgeprodukt SimPad

Abbildung 2.9: Automatischer Modus

2.2 Kommunikation

Die Kommunikation zwischen dem Szenarioassistenten, der Agentenanwendung und den Schnittstellen der Zielplattformen basiert auf Nachrichten. Im Folgenden werden die dazu erforderlichen Technologien vorgestellt.

2.2.1 XMPP

Das XMP-Protokoll ist eine offene, XML-basierte Internetprotokollfamilie, die primär für Instant Messaging Anwendungen eingesetzt wird. Weitere Bereiche, in denen XMPP erfolgreich

eingesetzt wird, sind:

- Soziale Netzwerke
- Computerspiele
- Middleware
- Netzwerkmanagement

XMPP wurde der Öffentlichkeit im Jahre 1999 erstmalig von Jeremy Miller unter dem Namen Jabber vorgestellt. Das Ziel von Jabber war es einen offenen Standard für Instant Messaging Anwendungen zu schaffen, da zur damaligen Zeit viele verschiedene proprietäre Protokolle wie ICQ (Homophon für I seek you) von Mirabilis, AIM (AOL Instant Messenger) von AOL und MSN (Microsoft Network) von Microsoft existierten. Wollte ein ICQ-Nutzer mit einem AIM-Nutzer kommunizieren, so war dies nicht möglich, da jede Technologie ihren eigenen Standard nutzte [Ada02].

Im August 2001 gründete die Jabber Gemeinde die Jabber Software Foundation (JSF) aus der im Jahre 2007 die XMPP Standards Foundation (XSF) wurde. Jeremy Miller beantragte bei der Internet Engineering Task Force (IETF) eine Anerkennung des Protokolls als Standard und im Jahre 2004 veröffentlichte die IETF folgende XMPP-Standards:

- RFC 3920: Core
- RFC 3921: Instant Messaging and Presence
- RFC 3922: Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)
- End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)

Um Nachrichten zwischen dem Szenarioassistenten und der Agentenanwendung auszutauschen wird in dieser Arbeit das XMP-Protokoll eingesetzt. XMPP basiert auf einer dezentralen Client-Server-Architektur. Ähnlich wie beim Simple Mail Transport Protocol (SMTP) meldet sich ein Client an seinem Server an und kann dann Nachrichten senden oder empfangen. Die Kommunikation erfolgt nach dem Push-Prinzip: Sendet ein Nutzer eine Nachricht an seinen Server dann übernimmt dieser die Weiterleitung der Nachricht. Dazu wird die Nachricht, sofern der Empfänger nicht am selben Server angemeldet ist, an den XMPP-Server des Empfängers geschickt, der die Nachricht dann zustellt.

Jeder XMPP-Nutzer bzw. jede XMPP-Entität besitzt zur eindeutigen Identifikation und Adressierung einen Jabber Identifier (JID). Ein JID ähnelt im Aufbau einer E-Mail-Adresse:

„benutzer@server.tld/ressource“

Vor dem @-Zeichen steht der Benutzername. Dies ist ein frei wählbarer Name, mit dem sich ein Nutzer am Server registrieren und anmelden kann. Danach folgt die Domäne, unter der der Server des Nutzers erreichbar ist. Nach dem Schrägstrich kann optional eine Ressource angegeben werden.

Das Konzept der Ressourcen ermöglicht die mehrmalige Anmeldung am Server von verschiedenen Orten oder Geräten mit dem gleichen Benutzerkonto. So kann sich ein Anwender mit seinem Smartphone und dem JID „benutzer@server.tld/smartphone“ anmelden, während er sich von seinem Laptop aus mit dem JID „benutzer@server.tld/laptop“ einloggt.

Der Austausch von XMPP-Nachrichten erfolgt über das Transmission Control Protocol (TCP). Dazu baut ein Client für die gesamte Dauer einer Sitzung eine TCP-Verbindung zum Server auf und sendet über einen XML-Datenstrom Nachrichten an den Server. Der Server stellt ebenfalls eine TCP-Verbindung zum Client her. Über eine bestehende Verbindung kann eine XMPP-Entität asynchron eine unbegrenzte Anzahl von Nachrichten senden oder empfangen.

XMPP-Nachrichten sind in der Auszeichnungssprache XML definiert und werden als Stanzas bezeichnet. Hiervon existieren die drei Typen <message>, <presence> und <iq> [SST09]. Stanzas sind, verglichen mit Datenpaketen anderer Netzwerkprotokolle, das Grundelement der Kommunikation über XMPP. Ein <message>-Stanza dient dem Austausch von Textnachrichten, während eine XMPP-Entität ein <presence>-Stanza nutzt, um Verfügbarkeitsinformationen zu senden. Stanzas vom Typ <iq> (Information query) ermöglichen eine Request-Response-Interaktion zwischen XMPP-Entitäten.

Eine wichtige Eigenschaft des Protokolls ist die Möglichkeit der Erweiterung. Mittlerweile existieren eine Fülle von XMPP-Erweiterungen, sogenannte XMPP Extension Protocols (XEP), die den Kern des Protokolls erweitern. Im Folgenden werden wichtige Erweiterungen, die im Rahmen dieser Arbeit genutzt werden, beschrieben.

Mit der Protokollerweiterung „XEP-0045: Multi-User Chat“ können mehrere XMPP-Entitäten untereinander Textnachrichten in einem Chat-Raum austauschen. Listing 2.1 zeigt ein <message>-Stanza, das an einen Multiuserchat gesendet wird. Das Attribut „from“ enthält den JID des Absenders. Im Attribut „to“ wird der Empfänger, also der JID des Multiuserchats, angegeben. Das „type“-Attribut enthält mit dem Wert „groupchat“ den Typ dieses <message>Stanzas. Die an den Multiuserchat zu übermittelnde Textnachricht steht im <body>-Element. Diese Nachricht wird vom XMPP-Server an alle Teilnehmer des Multiuserchats weitergeleitet. Mit der Protokollerweiterung „XEP-0077: In-Band Registration“ kann ein Benutzerkonto bei einem XMPP-Server registriert werden. Die Protokollerweiterung „XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)“ wird für Anwendungen genutzt, welche aufgrund gewisser Restriktionen keine langlebigen TCP-Verbindungen aufbauen können. Dies kann beispielsweise bei Browser-basierten Client-Anwendungen der Fall sein. Dazu wird XMPP über eine HTTP-Verbindung emuliert.

```
<message from="bob@server.tld/Smartphone"
     to="chatroom@conference.server.tld"
     type="groupchat">
  <body>Hallo!</body>
<\message>
```

Listing 2.1: Beispiel für ein <message>-Stanza

2.2.2 ActiveMQ

ActiveMQ ist ein Message Broker der Apache Software Foundation, der die Spezifikation Java Message Service (JMS) in der Version 1.1, für den Austausch von Nachrichten implementiert. ActiveMQ ist unter der Apache Lizenz frei verfügbar und wird auch als Message Oriented Middleware (MOM) bezeichnet. Eigenschaften dieser Softwareplattform sind Hochverfügbarkeit, Performanz, Skalierbarkeit, Zuverlässigkeit und Sicherheit im Bereich des Nachrichtenaustausches [Sny11].

Das vorrangige Ziel von ActiveMQ ist, neben Sprach- und Plattformunabhängigkeit, eine auf Standards basierende, nachrichtenorientierte Anwendungsintegration. Dazu bietet ActiveMQ, neben den bereits erwähnten Eigenschaften:

Konnektivität: ActiveMQ unterstützt eine Vielzahl von Kommunikationsprotokollen.

Client APIs: Die Plattform bietet, neben einem API für die Programmiersprache Java, Programmierschnittstellen für die Sprachen C/C++, C-Sharp, Perl, PHP, Python und Ruby.

Administration: ActiveMQ eröffnet verschiedene Möglichkeiten einen Message Broker zu administrieren. So können Werkzeuge, welche die Spezifikation Java Management Extensions (JMX) implementieren, genutzt werden. Es besteht aber auch die Möglichkeit die eingebaute Webkonsole, die über einen Webbrowser steuerbar ist, zu verwenden.

Ein Anwendungsszenario, in dem der Einsatz von ActiveMQ sinnvoll ist wäre beispielsweise die Integration von Anwendungen in heterogenen Anwendungslandschaften. Dies wird durch die bereits erwähnte Sprach- und Plattformunabhängigkeit und die Vielzahl an Client-APIs ermöglicht. ActiveMQ wird auch genutzt um eine lose Kopplung zwischen Anwendungen zu erreichen.

Bei der Integration von heterogenen Systemen und Anwendungen spielt der Austausch von Nachrichten eine wichtige Rolle [Ric09]. Eine Anwendung, die JMS-Nachrichten sendet oder empfängt, wird als JMS-Client bezeichnet. Hierbei wird zwischen Producer (Sender) und Consumer (Empfänger) unterschieden, wobei JMS-Client-Anwendungen auch beide Eigenschaften zugleich besitzen können.

JMS unterscheidet für den Austausch von Nachrichten zwei Modelle. Beim „Point to point“-Modell werden Nachrichten an eine Queue gesendet. Queues sind virtuelle Kanäle, die wie eine Warteschlange funktionieren. Ein Sender schickt eine Nachricht, die in die Warteschlange eingestellt wird und für einen Empfänger bestimmt ist. Im Gegensatz dazu werden im „Publish/Subscribe“-Modell Nachrichten vom Sender an ein sogenanntes Topic gesendet, bei dem sich beliebig viele Empfänger anmelden können um eine Nachricht zu empfangen. Abbildung 2.10 verdeutlicht diesen Unterschied.

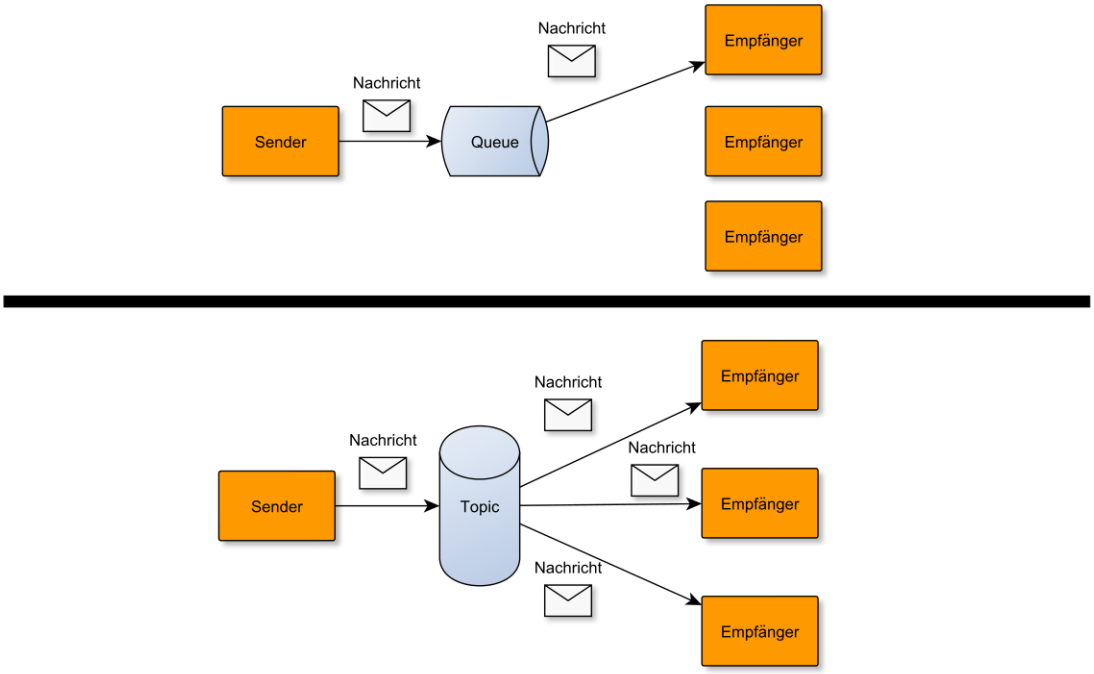


Abbildung 2.10: JMS-Queue und JMS-Topic

JMS ist kein Messaging System. Vielmehr bietet das API eine Abstraktionsebene zu Schnittstellen und Klassen, die Client-Anwendungen für die Kommunikation mit Messaging Systemen benötigen. Dies ist vergleichbar mit Java Database Connectivity (JDBC), dessen API eine Abstraktionsebene für den Zugriff auf relationale Datenbanken bietet.

Für den Austausch von Nachrichten bietet JMS verschiedene Nachrichtentypen. Unterschieden werden diese Nachrichtentypen nach der Art ihres Inhalts:

Message: Der Typ *javax.jms.Message* ist der einfachste Nachrichtentyp und bildet das Basis-Interface für alle anderen Nachrichtentypen. Diese Nachricht besitzt keine Nutzlast und kann für Benachrichtigungen bei bestimmten Ereignissen eingesetzt werden.

BytesMessage: Dieser Nachrichtentyp enthält einen Bytestrom.

TextMessage: Um einfache Textnachrichten oder XML-Strukturen zu senden kann dieser Nachrichtentyp verwendet werden. Er enthält ein Objekt vom Typ „java.lang.String“.

MapMessage: Eine MapMessage enthält ein Objekt vom Typ „java.util.Map“.

ObjectMessage: Eine Objektnachricht enthält ein serialisierbares Java-Objekt und ist somit für den Austausch von Java-Objekten geeignet.

StreamMessage: StreamMessages bestehen aus einem Strom von Java-Primitiven.

Die Kommunikation mit dem Digitalen Lagetisch muss über einen ActiveMQ Message Broker erfolgen, weshalb diese Technologie in der Agentenanwendung eingesetzt wird.

2.3 Mobile App

Mobile Applikationen sind Anwendungsprogramme, die auf mobilen Geräten installiert und ausgeführt werden können. Mobile Geräte werden in Geräteklassen eingeteilt. Neben den Geräteklassen Smartphone und Tablet-PC, für die eine Implementierung des Szenarioassistenten erfolgen soll, existieren weitere Geräteklassen, wie beispielsweise MP3-Player oder E-Reader.

2.3.1 Android

Android ist ein Betriebssystem und eine Software-Plattform für mobile Geräte. Sie entstand aus dem kleinen Startup Android Inc mit dem Ziel ein freies mobiles Betriebssystem zu entwickeln. Das Unternehmen wurde im Jahre 2005 von Google übernommen. Mittlerweile ist die Open Handset Alliance (OHA) Eigentümer und für die weitere Entwicklung des Betriebssystems verantwortlich. Die OHA ist ein Konsortium, das neben Google unter anderem aus den Firmen HTC, Samsung, Sprint und T-Mobile besteht. Das erste auf der Android-Plattform basierende Smartphone wurde mit dem HTC Dream, das auch unter dem Namen T-Mobile G1 bekannt wurde, im Jahre 2008 veröffentlicht. Seitdem wird das Betriebssystem mit jeder neu veröffentlichten Version populärer [Rol11].

Im zweiten Quartal 2012 hatte Android als Betriebssystem für Smartphones einen weltweiten Marktanteil von 68,1%, was einem Absatz von über 107 Millionen Geräten entspricht. Der Konkurrent Apple kommt mit seinem Betriebssystem iOS vergleichsweise auf 16,4% Marktanteil und somit 26 Millionen abgesetzten Geräten [Hei12].

Die Basis der Architektur des Betriebssystems bildet ein Linux-Kernel (Abbildung 2.11). Dieser Kernel stellt unterschiedliche Gerätetreiber wie WiFi-, Kamera-, Tastatur-, Audio- und Bildschirmtreiber zur Verfügung. Darüberhinaus übernimmt der Kernel auch die Steuerung der Energie-, Prozessor- und Speicherverwaltung.

Darauf aufbauend enthält die darüberliegende Schicht der Architektur neben nativen Bibliotheken die Android-Laufzeitumgebung und deren Kernbibliotheken. Die Android-Laufzeitumgebung beinhaltet darüber hinaus die Dalvik Virtual Machine (Dalvik VM). Native Android-Anwendungen werden in Java erstellt. Der Dalvik Cross-Assembler ist für die Übersetzung von Java-Binärdateien (.class) in den Android-eigenen DEX-Bytecode zuständig. Dieser DEX-Bytecode wird anschließend in der Dalvik VM ausgeführt. Die nativen, auf C oder C++ basierenden Bibliotheken, bieten Dienste für die Anwendungsschicht der Android-Architektur. So stellt die WebKit-Bibliothek eine Web Rendering Engine zur Verfügung, die auch bei Browsern wie Safari und Chrome eingesetzt wird. Die Bibliothek SQLite bietet eine SQL-Datenbank, während OpenGL eine 3D-Grafik-Bibliothek darstellt. Die Architekturschicht Application Framework bietet Dienste oder Manager für den Zugriff auf Systemfunktionen. So können beispielsweise Sensoren oder WiFi bei der Entwicklung von Anwendungen genutzt werden. Die oberste Schicht der Architektur bilden die in Java programmierten Anwendungen, die entweder vorinstalliert sind oder in Form einer Android Package Datei (.apk) auf den Geräten installiert werden können [Gar11].

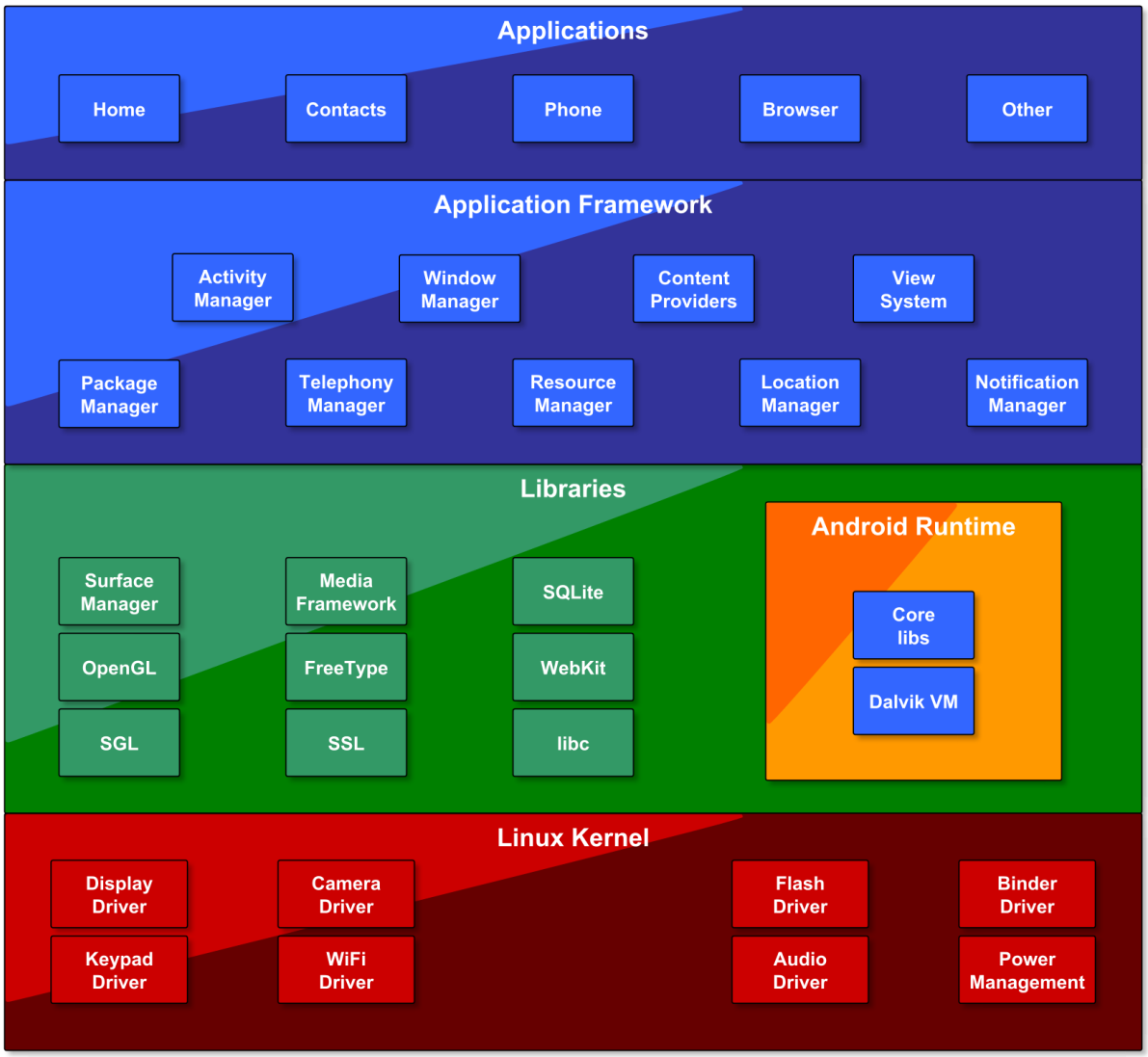


Abbildung 2.11: Architektur Android

Zur Entwicklung von Anwendungen für die Android-Plattform wird das Android SDK benötigt, das viele nützliche Werkzeuge beinhaltet. Neben den benötigten Bibliotheken für die Entwicklung von Anwendungen enthält das Android SDK auch die Dalvik VM zum Emulieren von mobilen Geräten wodurch das Testen von Anwendungen ohne ein Hardwaregerät möglich ist. Zum Debuggen von Anwendungen kann der Dalvik Debug Monitor Service (DDMS) eingesetzt werden. Das Android SDK ist für die Plattformen Windows, Linux und Mac OS X oder auch als Plug-in für die Entwicklungsumgebung Eclipse erhältlich.

Der in dieser Arbeit zu entwickelnde Szenarioassistent soll Android-basiert sein. Bei der Entwicklung von Anwendungen für die Android-Plattform spielt der API-Level eine wichtige Rolle. Mit diesem wird angegeben, auf welchen Geräten eine Anwendung installiert und ausgeführt werden kann. Um eine möglichst hohe Zahl von Android-Geräten zu erreichen sollte hier ein möglichst niedriger API-Level gewählt werden. Für den zu entwickelnden Szenarioassistenten wird der API-Level 10 verwendet. Wie dieser API-Level auf die aktiv genutzten Android-Geräte verteilt ist kann der Tabelle 2.1 entnommen werden [And12a].

Version	API-Level	Bezeichnung	Verteilung in Prozent
1.5	3	Cupcake	0.1
1.6	4	Donut	0.3
2.1	7	Eclair	2.7
2.2	8	Froyo	10.3
2.3 - 2.3.2	9	Gingerbread	0.2
2.3.3 - 2.3.7	10	Gingerbread	50.6
3.1	12	Honeycomb	0.4
3.2	13	Honeycomb	1.2
4.0.3 - 4.0.4	15	Ice Cream Sandwich	27.5
4.1	16	Jelly Bean	5.9
4.2	17	Jelly Bean	0.8

Tabelle 2.1: API-Level-Verteilung der aktiv genutzten Android-Geräte

2.3.2 PhoneGap

PhoneGap ist ein Hybrid-App-Framework für die Entwicklung nativer und mobiler Cross-Plattform-Anwendungen unter Verwendung standardisierter Web-Technologien wie HTML5, CSS und JavaScript. Das Framework ist unter der Apache Lizenz in der Version 2.0 frei verfügbar. Anwendungen, die mit PhoneGap erstellt wurden werden als hybride Anwendungen bezeichnet, da sie den Zugriff auf die Systemfunktionen einer Geräteplattform ermöglichen, aber nicht in der nativen Programmiersprache einer Plattform z.B. Objective-C für iOS erstellt wurden.

In der aktuellen Version 2.2.0 unterstützt das Framework die Plattformen Android, iOS, Blackberry, Windows Phone, Palm, WebOS, Bada und Symbian. Da eine plattformübergreifende Version des Szenarioassistenten nicht vorgesehen ist, beschränkt sich die Implementierung der Anwendung mit PhoneGap vorerst auf die Android-Plattform.

Das Projekt PhoneGap entstand im Jahr 2008 während dem iPhoneDevCamp in San Francisco [War12]. Initiator des Projekts war die kanadische Firma Naitobi, die im Jahr 2011 von der Firma Adobe übernommen wurde. Noch im selben Jahr wurde das Projekt, um die weitere Unabhängigkeit und Offenheit sicherzustellen, an die Apache Software Foundation übergeben, wo im Rahmen des Apache Incubator-Programms auch eine Namensänderung zu Apache Cordova stattfand. PhoneGap wird weiterhin als Apache Cordova-Distribution von Adobe angeboten und erhält Unterstützung von namhaften Firmen wie Microsoft, Google, IBM, RIM und HP [Pho12].

Für die Installation des Smartphone-Szenarioassistenten auf den Android-Geräten des Fraunhofer IOSB wird in dieser Arbeit das Framework PhoneGap eingesetzt. Dazu wird eine Android Package Datei des Projekts erzeugt und für die Ausführung der Anwendung auf den mobilen Endgeräten installiert.

2.3.3 jQuery Mobile

Die Entwicklung mobiler Anwendungen birgt besondere Herausforderungen. Diese Anwendungen werden nicht wie Desktopanwendungen genutzt. Informationen sollen möglichst ohne langes Suchen und auf einen Blick verfügbar sein. Eine Steuerung muss, da Smartphones und Tablet-PCs einen Touchscreen besitzen, mit dem Finger erfolgen. Auch der stetig wachsende Markt verschiedener Smartphones und Tablet-PCs, die unterschiedliche Displaygrößen und -auflösungen bieten, erschweren die Entwicklung von mobilen Anwendungen.

Diesen Herausforderungen entgegenzutreten ist das Ziel der Entwicklung der jQuery Mobile-Bibliothek. jQuery Mobile optimiert die Seiten einer mobilen Anwendung für die verschiedenen Geräte und Betriebssysteme und sorgt ebenfalls für eine korrekte Darstellung der Oberfläche im Hoch- und Querformat. Um die Kompatibilität der Bibliothek zu erhöhen und eine breitere Unterstützung von Geräten zu ermöglichen, nutzt jQuery Mobile den Progressive Enhancement-Ansatz bei dem zunächst nur ein lauffähiges HTML5-Markup erstellt wird, das auch von älteren Geräten ohne Fehler dargestellt werden kann. Abhängig vom Gerät auf dem die Anwendung dann ausgeführt wird, erfolgt clientseitig eine Generierung von zusätzlichen Markup-Elementen, CSS-Klassen und Event-Handlern für die Oberfläche einer Anwendung.

Um bei der Entwicklung des Szenarioassistenten auf bewährte Web-Technologien zu setzen und ein auf HTML5 basierendes Framework zu verwenden wird die Benutzungsoberfläche der Smartphone-Anwendung mit dem Framework jQuery Mobile umgesetzt. jQuery Mobile ist eine freie, umfangreiche und Markup-basierte Softwarebibliothek für die Entwicklung von Oberflächen für mobile Anwendungen. Sie wurde erstmalig im Jahr 2010 vom jQuery-Team um John Resig, unter einer Duallizenz MIT (Massachusetts Institute of Technology license) oder GPL (GNU General public license) veröffentlicht.

Weitere wichtige Eigenschaften der Bibliothek:

- bietet Standard-Oberflächenelemente wie beispielsweise Seiten, Schaltflächen, Listen, Dialoge, und Formularelemente
- ist optimiert für Touchscreen-Oberflächen
- nutzt Asynchronous JavaScript and XML (AJAX) um Inhalte dynamisch nachzuladen
- das visuelle Design ist durch die Verwendung von Themes einheitlich

Die Bibliothek unterstützt den Großteil aller modernen Desktop-, Smartphone-, E-Reader- und Tablet-Plattformen [jQu12].

Um einen Überblick der unterstützten Plattformen zu geben nimmt das jQuery-Team eine Unterteilung der Kompatibilität in verschiedene Kategorien vor. Die A-Kategorie-Plattformen bieten volle Unterstützung, inklusive AJAX und CSS media queries. C-Kategorie-Plattformen sind nicht kompatibel zu jQuery Mobile da sie das CSS und den JavaScript-Code des Frameworks nicht unterstützen. Diese Plattformen können nur den reinen HTML-Inhalt von jQuery Mobile-Anwendungen anzeigen [Fir12].

3 Anforderungsanalyse

In diesem Kapitel wird eine Anforderungsanalyse durchgeführt, welche die Basis für die Erarbeitung der weiteren Konzepte darstellt. Zunächst soll das Vorgehen zur Ermittlung der Anforderungen beschrieben werden. Danach wird ein Überblick über die Anwender des Systems gegeben, um anschließend die Systemarchitektur aufzuzeigen. Nach einer Schnittstellenbeschreibung der Zielplattformen folgt ein Abschnitt, in dem relevante Fakten und Annahmen für die Konzeptionierung und Implementierung erläutert werden. Abschließend werden ausgewählte, wichtige Anwendungsfälle und Anforderungen dargelegt.

Vorgehen bei der Anforderungsanalyse

Um die bereits im Vorfeld dieser Arbeit feststehenden Anforderungen um noch fehlende Anforderungen zu ergänzen, wird eine Anforderungsanalyse durchgeführt. Nachdem alle Anwendungsfälle ermittelt und beschrieben sind erfolgt eine Prüfung der Anwendungsfallabdeckung durch die bereits bestehenden Anforderungen. Falls Anwendungsfälle nicht durch bereits bestehende Anforderungen abgedeckt werden können, werden neue Anforderungen für diese Anwendungsfälle erstellt und die Liste der Anforderungen entsprechend ergänzt.

3.1 Anwender im System

Die potentiellen Anwender des Systems werden in folgende Rollen eingeteilt:

- ExBa-Entwickler
- ExBa-Nutzer
- ExBa-Operator

Der ExBa-Entwickler ist für die Erstellung und Einstellung von Szenariokonfigurationen in der Smartphone-Anwendung verantwortlich. Er besitzt die nötigen Kenntnisse, um im Vorfeld der Ausführung eines Demonstrationsszenarios, eine Szenariokonfiguration mit den entsprechenden Parametern zu erstellen und im Szenarioassistenten zu speichern.

Ein ExBa-Nutzer ist der eigentliche Nutzer des Szenarioassistenten und stellt mit Hilfe der Anwendung die Szenariokonfigurationsparameter für ein Demonstrationsszenario in die ExBa-Teilsysteme AMFIS-Bodenkontrollstation und Digitaler Lagetisch ein. Dazu benötigt der Nutzer keine exakten Kenntnisse über den Inhalt der Szenariokonfigurationen. Die Auswahl eines Demonstrationsszenarios erfolgt anhand von Beschreibungen die im Szenarioassistenten angezeigt werden.

Der ExBa-Operator hat die Aufgabe die Agentenanwendungen auf den Zielplattformen für die Ausführung von Demonstrationsszenarien vorzubereiten. Dazu benötigt er Kenntnisse darüber wie diese entsprechend konfiguriert und anschließend ausgeführt werden müssen. In den Konfigurationsdateien der Agentenanwendungen aktiviert er die Module, also Zielsysteme, die für die Ausführung des geplanten Demonstrationsszenarios benötigt werden.

3.2 Systemarchitektur

Den zentralen Mittelpunkt der Architektur bildet ein XMPP-Server, der die Nachrichtenvermittlung zwischen dem Szenarioassistenten und den Agentenanwendungen auf den Zielplattformen regelt (Abbildung 3.1). Die Kommunikation zwischen diesen beiden Anwendungen erfolgt über den XMPP-Server und unter Verwendung von XMPP-Nachrichten, welche die Nutzlast enthalten. Der Szenarioassistent sendet die Nachrichten über das spezielle Transportprotokoll BOSH, das, wie im Kapitel Grundlagen beschrieben, eine XMPP-Verbindung über eine HTTP-Verbindung emuliert. Die Einstellung der Szenariokonfigurationsparameter in die Zielplattformen Digitaler Lagetisch und AMFIS-Bodenkontrollstation erfolgt über deren Schnittstellen.

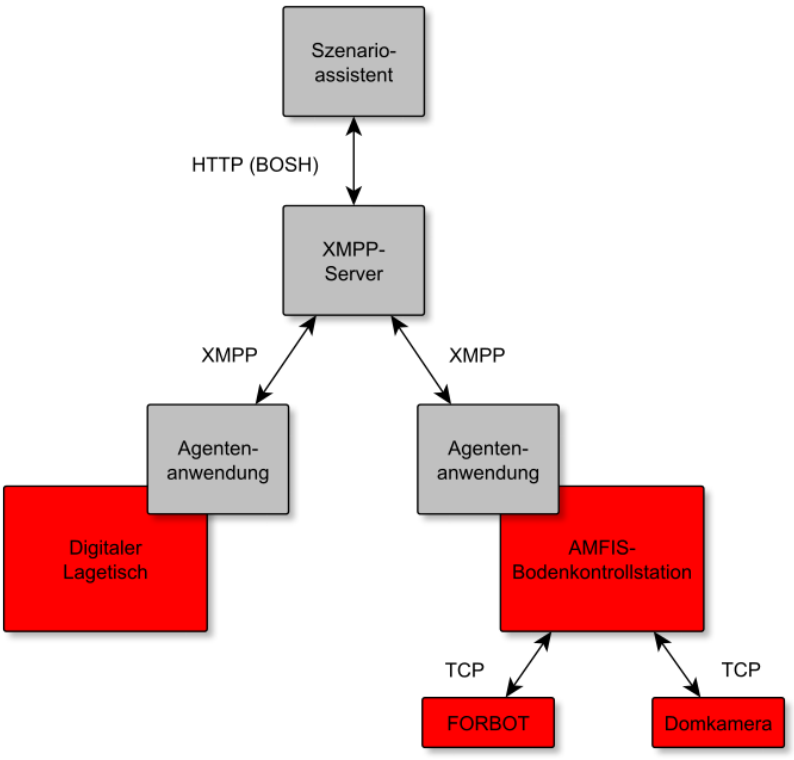


Abbildung 3.1: Systemarchitektur

3.3 Schnittstellen der Zielplattformen

In diesem Abschnitt werden die Schnittstellen der Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch beschrieben.

AMFIS-Bodenkontrollstation

Die Kommunikaton zwischen einer Agentenanwendung und der AMFIS-Bodenkontrollstation erfolgt über den AMFIS-Konnektor. Der AMFIS-Konnektor ist ein Message Broker bei dem sich Client-Anwendungen anmelden, um Nachrichten senden und empfangen zu können. So können beispielsweise Sensordaten, wie die Position eines Sensorträgers empfangen werden während gleichzeitig die Möglichkeit besteht Nachrichten, die Parameter zur Steuerung von Sensorträgern enthalten, an den AMFIS-Konnektor zu senden (Abbildung 3.2).

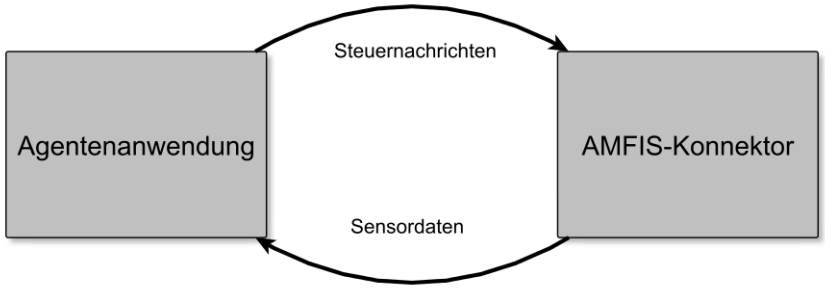


Abbildung 3.2: Kommunikation mit dem AMFIS-Konnektor

Der AMFIS-Konnektor nutzt ein proprietäres Kommunikationsprotokoll, basierend auf XML. Für den Austausch von Nachrichten werden XML-Strings über TCP-Verbindungen gesendet (Listing 3.1).

Nach dem Aufbau einer Verbindung zum AMFIS-Konnektor sendet dieser einen konstanten Strom von Nachrichten die von Client-Anwendungen empfangen und ausgewertet werden können. Zur Entwicklung von Client-Anwendungen für den AMFIS-Konnektor in der Programmiersprache Java existiert die Bibliothek JavaAmfisCom, die eine Programmierschnittstelle mit Methoden für die Kommunikation bereitstellt und somit die Entwicklung von Client-Anwendungen erleichtert da das Kommunikationsprotokoll nicht selbst implementiert werden muss.

```
<message key="string" type="string">
  <parentmessage key="string" />
  <timestamp>YYYY-MM-DD hh:mm:ss</timestamp>
  <originator type="SENSOR|SENSORNODE|USER|SYSTEM">
    <sensornodeid>bigint</sensornodeid>
    <sensorid>bigint</sensorid>
    <userid>bigint</userid>
  </originator>
  <subject type="SENSORNETWORK|SENSORNODE|SENSOR|SYSTEM">
    <sensornetwork>bigint</sensornetwork>
    <sensornodeid>bigint</sensornodeid>
    <sensorid>bigint</sensorid>
  </subject>
  <value>value</value>
</message>
```

Listing 3.1: Aufbau einer AMFIS-Nachricht

Digitaler Lagetisch

Die Schnittstelle des Digitalen Lagetisches bildet eine Message Oriented Middleware. Als Provider wird hierfür ein ActiveMQ Message Broker eingesetzt, der die JMS-Spezifikation implementiert. Zur Kommunikation mit dieser Schnittstelle ist in der Agentenanwendung eine Komponente, die eine Verbindung zum Message Broker aufbaut und eine Nachricht mit den Szenariokonfigurationsparametern für den Digitalen Lagetisch sendet, erforderlich.

3.4 Relevante Fakten und Annahmen

In diesem Abschnitt soll auf Fakten und Annahmen eingegangen werden, die für die weitere Konzeptionierung und somit auch der Implementierung des Szenarioassistenten und der Agentenanwendung wichtig sind.

Demonstrationsszenarien

Ein Demonstrationsszenario, das mit dem Szenarioassistenten gestartet werden kann, ist statisch. Es wird also vorausgesetzt, dass das geografische Gebiet und dessen geografische Koordinaten bekannt sind. Ebenso ist davon auszugehen, dass die Positionen der AMFIS-Zielsysteme FORBOT und Domkamera zu Beginn eines Demonstrationsszenarios bekannt sind.

Steuerung der AMFIS-Zielsysteme

Im Rahmen dieser Arbeit soll zunächst, im Fall der AMFIS-Bodenkontrollstation, nur die Steuerung der AMFIS-Zielsysteme FORBOT und Domkamera mit der Ausführung eines Demonstrationsszenarios ermöglicht werden. Weiterhin ist aus sicherheitstechnischen Gründen eine direkte Steuerung mobiler Sensorträger wie Drohnen oder Landroboter mit der Einstellung von Parametern durch den Szenarioassistenten nicht vorgesehen, da diese Sensorträger keine Objekterkennung besitzen und folglich kollidieren könnten. Vielmehr können die Parameter für diese Sensorträger nur in den Piloten-Arbeitsplatz eingestellt werden. Nachdem die Parameter eingestellt wurden muss die Ausführung und damit auch die Steuerung ein Operator der AMFIS-Bodenkontrollstation am Piloten-Arbeitsplatz übernehmen. Im Gegensatz dazu kann eine Domkamera, da sie ein stationärer Sensorträger ist, erhaltene Parameter in Form von Steuerbefehlen direkt und autonom ausführen.

Architekturumstellung des Digitalen Lagetisches

Da momentan eine Umstellung der Software-Architektur des Digitalen Lagetisches stattfindet, ist im Rahmen dieser Arbeit kein realer Zugriff auf dieses System möglich. Die Einstellung von Szenariokonfigurationsparametern in den Digitalen Lagetisch kann daher nur simuliert werden.

3.5 Anwendungsfälle und Anforderungen

In diesem Abschnitt werden wichtige Anwendungsfälle und Anforderungen für den Szenarioassistenten und die Agentenanwendung aufgezeigt.

Szenarioassistent

Zur besseren Nachverfolgbarkeit wird jeder Anwendungsfall mit der Abkürzung „AF“ und einer Nummer versehen. Anforderungen beginnen mit dem Kürzel „ANF“, ebenfalls folgt eine Nummer. Zunächst wird für den Szenarioassistenten der primäre Anwendungsfall „AF-01: Szenariokonfigurationsparameter auf die Systemkomponenten der Zielplattformen übertragen“, der das typische Anwendungsszenario darstellt, betrachtet. Dieser Anwendungsfall bildet die Basis für das Identifizieren und Erarbeiten weiterer Anwendungsfälle, indem die einzelnen konkreten Ablaufschritte, soweit dies möglich und nötig ist, in weitere Anwendungsfälle heruntergebrochen werden.

Das Ziel dieses Anwendungsfalles ist die möglichst einfache und schnelle Einstellung von Szenariokonfigurationsparametern in die ExBa-Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch. Der ExBa-Nutzer stellt hierbei den primären Akteur des Anwendungsfalles dar, wohingegen der ExBa-Entwickler und der ExBa-Operator als sekundäre Akteure betrachtet werden. Dass die Zielplattformen einsatzbereit sind, wird als Vorbedingung für diesen Anwendungsfall festgelegt. Das Erfolgsergebnis ist die erfolgreiche Einstellung der Szenariokonfigurationsparameter in die Zielplattformen. Die Durchführung eines Demonstrationsszenarios ist das auslösende Ereignis.

Die Ablaufschritte des Anwendungsfalles gestalten sich folgendermaßen. Dabei besitzt jeder Ablaufschritt, der einen weiteren Anwendungsfall darstellt, einen Verweis auf die Beschreibung für diesen neuen Anwendungsfall:

1. ExBa-Entwickler erstellt eine Szenariokonfigurationsdatei (AF-02).
2. ExBa-Entwickler speichert die Szenariokonfiguration im Szenarioassistenten (AF-03).
3. ExBa-Nutzer startet den Szenarioassistenten.
4. ExBa-Nutzer bekommt eine Übersicht der verfügbaren Szenarien angezeigt (AF-04).
5. ExBa-Nutzer wählt das auszuführende Szenario aus der Szenarioübersichtsliste aus (AF-05).
6. ExBa-Nutzer bekommt Detailinformationen zum ausgewählten Szenario angezeigt (AF-06).

   6.1 ExBa-Nutzer lässt sich die Verfügbarkeit der Zielsysteme anzeigen (AF-07).
7. ExBa-Nutzer startet das ausgewählte Szenario (AF-08).

Der erste Ablaufschritt des Anwendungsfalles verweist auf den untergeordneten Anwendungsfall AF-02. Eine vollständige Liste der Anwendungsfallbeschreibungen für den Szenarioassistenten kann im Anhang A.1.1 eingesehen werden. Um die Ermittlung der weiteren Anforderungen zu beschreiben wird zunächst der Anwendungsfall AF-02 aufgezeigt (Tabelle 3.1). Bei der Prüfung der Anwendungsfallabdeckung wird diesem Anwendungsfall, wie in der letzten Tabellenzeile zu erkennen ist, die bereits im Vorfeld dieser Arbeit bestehende Anforderung ANF-01 zugewiesen.

AF-02:	Szenariokonfigurationsdatei erstellen
Ziel:	Erstellung einer Szenariokonfigurationsdatei, welche die notwendigen Parameter für eine Szenarioausführung enthält.
Primäre Akteure:	ExBa-Entwickler
Sekundäre Akteure:	keine
Vorbedingung:	Durchführung eines Demonstrationsszenarios.
Erfolgsergebnis:	Konfigurationsdatei, die im Szenarioassistenten gespeichert werden kann.
Ablauf:	1.) ExBa-Entwickler erstellt eine Konfigurationsdatei. 2.) ExBa-Entwickler trägt Parameter in die Konfigurationsdatei ein.
Anforderung/en:	ANF-01

Tabelle 3.1: Anwendungsfallbeschreibung AF-02

Jedem weiteren Anwendungsfall werden Anforderungen zugewiesen, sodass eine vollständige Abdeckung der Anwendungsfälle durch bereits bestehende oder ergänzende Anforderungen erreicht werden kann. Die dabei neu aufgestellten funktionalen Anforderungen sind der Tabelle 3.2 zu entnehmen. Eine Beschreibung aller Anforderungen erfolgt in der Auflistung der Anforderungen für den Szenarioassistenten im Anhang A.2.1.

ANF-08:	Anzeige der verfügbaren Szenarien in einer Übersichtsliste
ANF-09:	Auswahl eines Szenarios aus der Übersichtsliste
ANF-10:	Anzeige einer Szenariodetailseite zu einem gewählten Szenario
ANF-11:	Verfügbarkeit der Zielplattformen bzw. Zielsysteme auf einer Übersichtsseite anzeigen
ANF-12:	Speichern von Szenariokonfigurationsdateien im Szenarioassistenten

Tabelle 3.2: Zusätzlich aufgestellte Anforderungen für den Szenarioassistenten

Agentenanwendung

Im Rahmen der Anwendungsfallanalyse für die Agentenanwendung wird der primäre Anwendungsfall „AF-09: Szenariokonfigurationsparameter in die Systemkomponenten der Zielplattformen einstellen“ festgelegt.

Das Ziel dieses Anwendungsfalles ist die Einstellung von Szenariokonfigurationsparametern in die Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch, die vom Szenarioassis-

tenten in Form einer Szenariokonfiguration für ein Demonstrationsszenario empfangen werden. Die primären Akteure in diesem Anwendungsszenario sind der ExBa-Operator und die Agentenanwendung selbst. Sekundäre Akteure werden keine festgestellt und die Vorbedingung und das Erfolgsergebnis werden vom primären Anwendungsfall des Szenarioassistenten übernommen.

An dieser Stelle folgt nun der Ablauf für diesen Anwendungsfall:

1. ExBa-Operator konfiguriert die Agentenanwendung für die Zielplattform (AF-10).
2. ExBa-Operator führt Agentenanwendung auf der Zielplattform aus (AF-11).
3. Agentenanwendung baut eine XMPP-Serververbindung auf (AF-12).
4. Agentenanwendung verbindet sich mit der Zielplattform (AF-13).
5. Agentenanwendung sendet Verfügbarkeit der Zielsysteme (AF-14).
6. Agentenanwendung nimmt Szenariokonfiguration entgegen (AF-15).
7. Agentenanwendung wertet Szenariokonfiguration aus (AF-16).
8. Agentenanwendung stellt Parameter aus der Szenariokonfiguration in die Zielsysteme ein (AF-17).

Die vollständige Liste der ermittelten Anwendungsfälle für die Agentenanwendung kann im Anhang A.1.2 eingesehen werden.

Für die Agentenanwendung müssen die bereits bestehenden Anforderungen ebenfalls durch neu aufgestellte Anforderungen ergänzt werden. Eine Übersicht dieser Anforderungen erfolgt in der Tabelle 3.3. Die vollständige Auflistung aller Anforderungen an die Agentenanwendung erfolgt im Anhang A.2.2. Nachdem alle Anwendungsfälle und Anforderungen ermittelt sind wird im folgenden Kapitel die Konzeption aufgezeigt.

ANF-18:	Implementierung einer Komponente für XMPP
ANF-19:	Implementierung einer Komponente für AMFIS
ANF-20:	Implementierung einer Komponente für den Digitalen Lagetisch
ANF-22:	Erstellung einer ausführbaren Java-Archiv-Datei (JAR)

Tabelle 3.3: Zusätzlich aufgestellte Anforderungen der Agentenanwendung

4 Konzeption

Dieses Kapitel beschreibt die Konzepte auf der die Implementierung für den Szenarioassistenten und die Agentenanwendung aufbaut. Dazu erfolgt ein Überblick über den Systemaufbau, um anschließend auf das Konzept für die Kommunikation zwischen den verschiedenen Systemen einzugehen. Nachdem die Konzeptionierung des Demonstrationsszenarios stattfand, werden die Konzepte für die Implementierung des Szenarioassistenten und der Agentenanwendung vorgestellt.

4.1 Systemaufbau

In diesem Unterkapitel soll der Aufbau des gesamten Systems, das aus dem XMPP-Server, dem Szenarioassistenten und der Agentenanwendung besteht, beschrieben werden. Für eine grafische Übersicht des Systemaufbaus wird auf die bereits gezeigte Abbildung 3.1 der Systemarchitektur im Kapitel Anforderungsanalyse verwiesen.

XMPP-Server

Da bereits eine XMPP-Serverinstanz existiert entfällt die Installation und Konfiguration eines XMPP-Servers. Der XMPP-Server stellt die zentrale Kommunikationsinstanz zwischen dem Szenarioassistenten und den Agentenanwendungen auf den Zielplattformen dar. Der Server empfängt XMPP-Nachrichten und stellt sie den Empfängern zu. Zur Kommunikation zwischen den Systemen wird im Server ein Multiuserchat eingerichtet. Der Multiuserchat übernimmt auch eine Überwachungs- und Protokollfunktion. Durch Anmelden eines gewöhnlichen XMPP-Clients am Server kann der Nachrichtenverkehr mitverfolgt und überwacht werden.

Szenarioassistent

Der Szenarioassistent soll dem Anwender die Möglichkeit bieten aus einer Liste von Demonstrationsszenarien eine Auswahl zu treffen. Für die einzelnen Szenarien werden XML-basierte Szenariokonfigurationsdateien hinterlegt. Ist eine Auswahl getroffen wird das Demonstrationsszenario gestartet und die zugehörige XML-Datei in einer XMPP-Nachricht an den XMPP-Server gesendet. Der XMPP-Server liefert die Nachricht an die Agentenanwendungen auf den Zielplattformen aus, die dann die Weiterverarbeitung übernehmen und die Szenariokonfigurationsparameter in die Zielsysteme der Zielplattformen einstellen.

Agentenanwendung

Die Agentenanwendung stellt aus Sicht des Szenarioassistenten die Schnittstelle zu den Ziel-

plattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch dar. Die Aufgabe einer Agentenanwendung, die auf einer Zielplattform ausgeführt wird besteht zunächst darin Szenariokonfigurationsdaten vom Szenarioassistenten über den XMPP-Multiuserchat entgegenzunehmen und auszuwerten. Nach der Auswertung sollen die Parameter in die Zielsysteme der Zielplattform eingestellt werden. Weiterhin soll eine Agentenanwendung die Verfügbarkeit der Zielsysteme auf der Zielplattform prüfen und diese Information für die Auswertung und Visualisierung an den Szenarioassistenten senden. Die Agentenanwendung soll, wie im Kapitel Anforderungsanalyse beschrieben, modular aufgebaut sein und über eine externe XML-Datei konfiguriert werden können. Zur Ausführung der Agentenanwendung auf einer Zielplattform müssen die entsprechenden Module, AMFIS-Bodenkontrollstation mit dem FORBOT und den Domkameras oder Digitaler Lagetisch, in der Konfigurationsdatei parametrisiert und aktiviert werden.

4.2 Kommunikation

Die Kommunikation zwischen dem Szenarioassistenten und den Agentenanwendungen erfolgt durch XMPP-Nachrichten, die an den Multiuserchat des XMPP-Servers gesendet werden (Abbildung 4.1). Die Einstellung der Szenariokonfigurationsparameter durch die Agentenanwendungen in die Zielsysteme erfolgt über die spezifischen Schnittstellen der Zielplattformen. Im Fall der AMFIS-Bodenkontrollstation werden AMFIS-Nachrichten an den AMFIS-Konnektor gesendet. An den ActiveMQ Message Broker der Zielplattform Digitaler Lagetisch wird eine JMS-Nachricht geschickt.

Die Verfügbarkeitsnachrichten der Agentenanwendungen werden, sobald diese auf den Zielplattformen gestartet wurden, kontinuierlich an den Multiuserchat gesendet. Der XMPP-Server leitet diese Nachrichten zur Auswertung an den Szenarioassistenten weiter. Eine Szenariokonfiguration wird vom Szenarioassistenten erst dann an den Multiuserchat gesendet, wenn die Ausführung eines Demonstrationsszenarios in der Anwendung gestartet wurde. Die gesendete Szenariokonfiguration wird anschließend vom XMPP-Server an die Agentenanwendungen auf den Zielplattformen weitergeleitet.

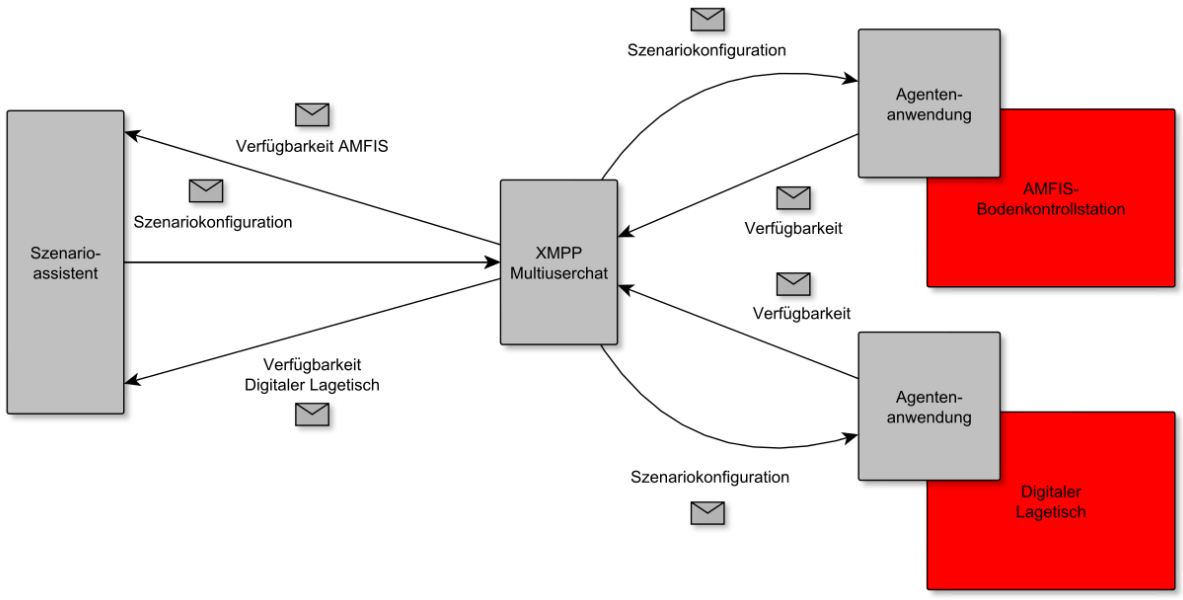


Abbildung 4.1: Kommunikation zwischen den Systemen

4.3 Demonstrationsszenario

Um ein Demonstrationsszenario definieren zu können müssen zunächst einige Eigenschaften des Szenarios festgelegt werden. Das Ziel des Szenarios ist die Überwachung einer großen Liegenschaft. Diese Liegenschaft kann als eingegrenztes Gebiet mit verschiedenen Gebäuden verstanden werden. Im Demonstrationsszenario wird dieses Gebiet durch das Fraunhofer IOSB in Karlsruhe repräsentiert.

Beteiligt an diesem Demonstrationsszenario sind der Digitale Lagetisch und die AMFIS-Zielsysteme FORBOT, Axis-Dome 1 und Axis-Dome 2. Außer dem AMFIS-Zielsystem FORBOT sind mit den beiden Domkameras ausschließlich stationäre Zielsysteme an diesem Demonstrationsszenario beteiligt. Für die Durchführung wird vorausgesetzt, dass alle Zielsysteme einsatzbereit sind. Die Aufgabe der Zielsysteme in diesem Demonstrationsszenario ist die Überwachung und Aufklärung der Lage im Innenbereich der Liegenschaft zur Unterstützung des Sicherheitspersonals vor Ort. Dazu soll sich jede Domkamera auf einen bestimmten Bereich der Liegenschaft ausrichten und das Bildmaterial an die Lagedarstellung der AMFIS-Bodenkontrollstation liefern. Der FORBOT soll den Innenbereich der Liegenschaft erkunden, während der Digitale Lagetisch ebenfalls Bildmaterial zur Liegenschaft anzeigt. Weiterhin wird, wie im Kapitel Anforderungsanalyse beschrieben, angenommen, dass der FORBOT seine Überwachungsaufgabe nicht autonom durchführen kann, da das System keine Objekterkennung besitzt und somit bei eventuell auftretenden Hindernissen im Gelände nicht ausweichen kann und demzufolge kollidieren könnte. Die Parameter für den FORBOT werden in diesem Demonstrationsszenario, wie bereits beschrieben, in das AMFIS-System nur eingestellt und vom AMFIS-Operator am Piloten-Arbeitsplatz der AMFIS-Bodenkontrollstation entsprechend ausgeführt. Alle anderen Systeme sind in der Lage ihre Aufgaben autonom durchzuführen.

4.3.1 Aufgaben

Für die Ausführung dieses Demonstrationsszenarios wird eine XML-basierte Szenariokonfigurationsdatei für den Szenarioassistenten erstellt. Dazu muss zunächst festgelegt werden wie die Aufgaben in diesem Demonstrationsszenario von den beteiligten Zielsystemen mit welchen Steuerbefehlen umgesetzt werden sollen.

FORBOT

Die Zielsysteme der AMFIS-Bodenkontrollstation lassen sich über Nachrichten steuern, die an den AMFIS-Konnektor gesendet werden. Um seine Aufgabe im Demonstrationsszenario durchzuführen ist für den FORBOT der AMFIS-Nachrichtentyp „waypointlist“ von Bedeutung. Dieser Nachrichtentyp enthält eine Liste von geografischen Koordinaten in Form von Wegpunkten. Wurde diese Nachricht an den AMFIS-Konnektor gesendet und ausgewertet, wird die Wegpunktliste im Piloten-Arbeitsplatz der AMFIS-Bodenkontrollstation angezeigt und kann von einem Operator mit dem FORBOT abgefahren werden.

Domkameras Axis-Dome 1 und Axis-Dome 2

Für jede Domkamera wird die Ausrichtung auf einen Bereich der Liegenschaft umgesetzt. Eine

Domkamera kann mit dem AMFIS-Nachrichtentyp „lookat“ gesteuert werden. Als Parameter wird bei diesem Nachrichtentyp eine geografische Koordinate angegeben. Wird dieser Nachrichtentyp an den AMFIS-Konnektor gesendet und anschließend ausgewertet, richtet sich eine Domkamera auf den Bereich, der mit der geografischen Koordinate in der Nachricht angegeben wurde.

Digitaler Lagetisch

Der Digitale Lagetisch soll im Rahmen des Demonstrationsszenarios das geografische Gebiet, in der sich die Liegenschaft befindet, anzeigen. Damit der Digitale Lagetisch eine Lagedarstellung für ein geografisches Gebiet anzeigen kann muss eine Nachricht, die als Parameter die ID des Szenarios enthält, das angezeigt werden soll, an die Schnittstelle des Digitalen Lagetisches gesendet werden.

4.3.2 Datenmodell

Zur Erledigung dieser Aufgaben muss nun ein geeignetes XML-basiertes Datenmodell, das alle Parameter zur Steuerung der einzelnen Zielsysteme enthält, definiert werden (Abbildung 4.2). Zur Beschreibung dieser Szenariokonfigurationsdatei wird die Schemasprache XML Schema eingesetzt. Die zentralen XML-Elemente der Konfigurationsdatei sind neben dem Stammelement <scenarioConfig> die folgenden Kindelemente:

- <metadata>: Soll den Autor und das Erstellungsdatum der Szenariokonfiguration enthalten. Somit kann jederzeit festgestellt werden, wann und von wem die Konfigurationsdatei erstellt wurde.
- <information>: Enthält Informationen zum Demonstrationsszenario. Vorgesehen sind der Name des Szenarios, eine Kurz- und Langbeschreibung, sowie der Ort und die Art des Szenarios.
- <nodes>: Beinhaltet alle am Szenario beteiligten Zielsysteme in Form von <node>-Kindelementen.
- <node>: Dieses Element stellt ein Zielsystem dar und soll einen eindeutigen Bezeichner und die Konfigurationsparameter, die in das Zielsystem übertragen werden, enthalten. Das Element besteht aus den Unterlementen <nodeId>, <operation> und <value>.
- <nodeId>: Enthält den eindeutigen Bezeichner eines Zielsystems und wird für die Kopplung zwischen den Agentenanwendungen und den Zielsystemen benötigt.
- <operation>: Bezeichnet die Aufgabe, die ausgeführt werden soll und entspricht gleichzeitig einer Kennung für einen Steuerbefehl eines Zielsystems.
- <value>: Das Element legt einen Wert für die auszuführende Aufgabe eines Zielsystems fest.

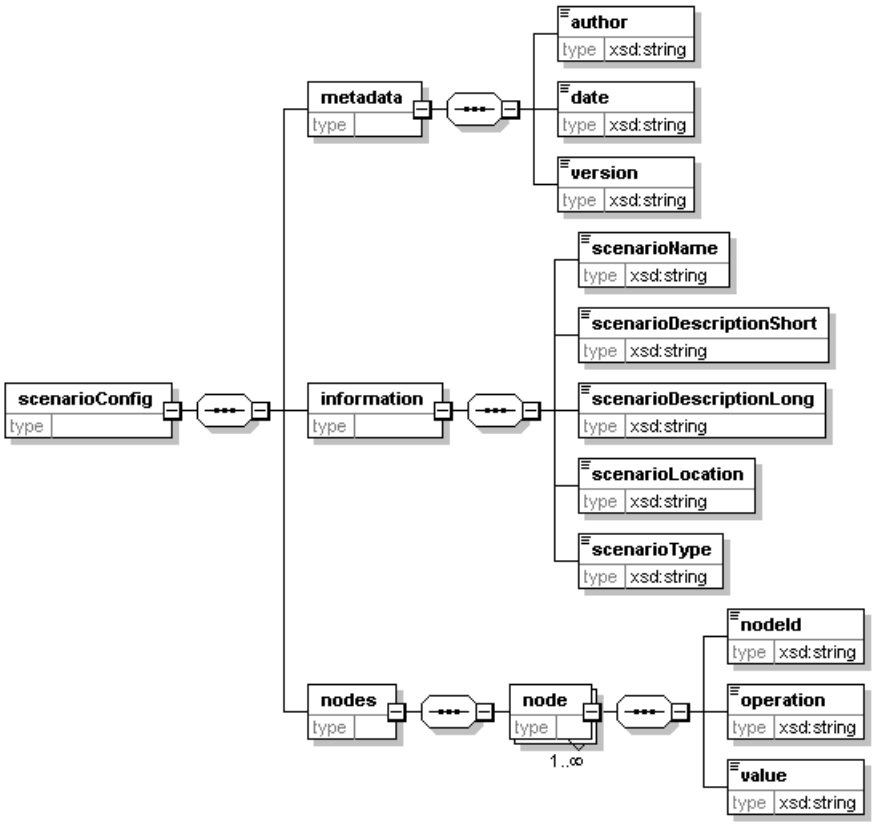


Abbildung 4.2: XML Schema-Datei der Szenariokonfiguration

4.4 Szenarioassistent

In den beiden folgenden Abschnitten findet nun die Konzeptionierung für die Implementierung des Szenarioassistenten statt. Dazu wird zunächst auf die Architektur eingegangen, um anschließend zu zeigen, wie die Oberflächen und Funktionen der Anwendung umgesetzt werden sollen.

4.4.1 Architektur

Der Szenarioassistent soll in einer komponentenbasierten Architektur realisiert werden. Dazu soll der Szenarioassistent aus den Komponenten Präsentation, Anwendungslogik und XMPP-Client bestehen (Abbildung 4.3).

Präsentation

Die Komponente für die Präsentation der Anwendung hat verschiedene Aufgaben. Sie ist für die Darstellung aller Oberflächen und ihrer Elemente zuständig. Weiterhin werden Benutzereingaben, wie die Auswahl eines Demonstrationsszenarios, an die Komponente für die Anwendungslogik weitergeleitet, in welcher anschließend eine Weiterverarbeitung erfolgt. Für die Ausgabe der Zielsystemverfügbarkeit an den Benutzer empfängt die Präsentations-Komponente Daten von der Komponente der Anwendungslogik und visualisiert sie entsprechend in der Benutzungsoberfläche.

Anwendungslogik

Die Komponente der Anwendungslogik hat eine Steuerungs- und Koordinationsfunktion. Wurde ein Demonstrationsszenario ausgewählt und gestartet, erhält die Komponente die Daten der Benutzereingabe (ausgewähltes Szenario) von der Präsentations-Komponente und kann dann die entsprechende Szenariokonfigurationsdatei einlesen und der XMPP-Client-Komponente zur Übertragung an den XMPP-Server übergeben.

XMPP-Client

Der XMPP-Client stellt eine Komponente für die Kommunikation mit dem XMPP-Server dar. Diese Komponente kapselt alle Funktionen, die für die Kommunikation mit dem XMPP-Server notwendig sind.

Funktionen, die der XMPP-Client bieten soll, sind:

- Verbindungsaufbau zum XMPP-Server
- Registrierung und Authentifizierung
- Erstellen eines Multiuserchats
- Betreten eines Multiuserchats
- Senden und Empfangen von XMPP-Nachrichten

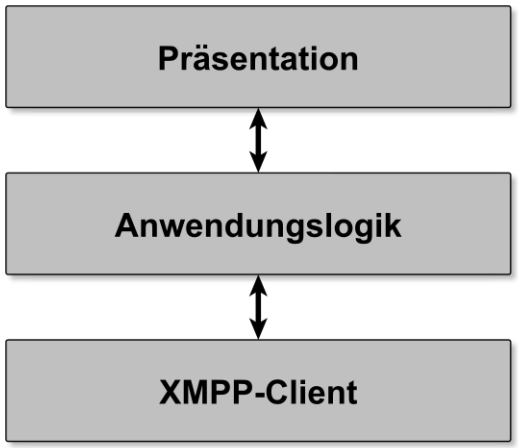


Abbildung 4.3: Architektur des Szenarioassistenten

4.4.2 Oberfläche und Funktionen

Folgende Oberflächenfunktionen ergeben sich aus der Anforderungsanalyse für den Szenarioassistenten:

- Anzeige von Szenarien
- Auswahl eines Szenarios

- Anzeige von Detaildaten zu einem Szenario
- Anzeige der verfügbaren Zielsysteme
- Starten eines Szenarios

Um die Anforderungen an die Benutzungsoberfläche des Szenarioassistenten zu erfüllen, lassen sich die folgenden Oberflächenlayouts feststellen:

- Die Szenarioübersichtsliste zur Anzeige und Auswahl eines Demonstrationsszenarios. Dieses Oberflächenlayout wird auch nach dem Starten der Anwendung angezeigt.
- Die Szenariodetailansicht, welche die Anzeige von Detaildaten zu einem Demonstrationsszenario ermöglicht.
- Die Ansicht für die Anzeige der verfügbaren Zielsysteme.

Für die Oberflächen des Szenarioassistenten wird zunächst ein grundsätzlicher Aufbau der Seiten festgelegt. So soll jedes Oberflächenlayout der Anwendung eine Kopfzeile für Kontextinformationen und einen Bereich für den eigentlichen Seiteninhalt erhalten. Der Aufbau der Oberflächen soll nachfolgend anhand des typischen Anwendungsszenarios „Einstellung von Szenariokonfigurationsparametern in die Zielsysteme der Zielplattformen“ aufgezeigt werden.

Nach dem Starten der Anwendung soll der Benutzer des Szenarioassistenten in der Lage sein, aus einer Auflistung von Szenarien das gewünschte Demonstrationsszenario auszuwählen und dieses zu starten. Dazu bekommt der Benutzer auf der Startseite eine Listenansicht der verfügbaren Szenarien angezeigt (Abbildung 4.4). Die Einträge der Liste enthalten Bilder und Kurzbeschreibungen zu den Szenarien, wobei eine Kurzbeschreibung die Art und den Ort des Demonstrationsszenarios enthalten soll. Die Entwürfe der Oberflächen des Szenarioassistenten bestanden schon im Vorfeld dieser Arbeit. In diesen Entwürfen trägt die Anwendung den Namen „@ssist.Szenario“, welcher im Verlauf der Arbeit wieder verworfen wurde. An dieser Stelle soll nun der Hinweis erfolgen, dass deshalb der Name „Szenarioassistent“ mit dessen Abkürzung „SzenAs+“ schon seit Beginn dieser Arbeit verwendet wird.

Abbildung 4.4: Startseite des Szenarioassistenten

Nachdem der Benutzer ein Demonstrationsszenario ausgewählt hat, wird er auf die zugehörige Detailseite weitergeleitet (Abbildung 4.5). Die Detailansicht enthält neben der Art und dem Ort des Szenarios eine ausführlichere Beschreibung des Demonstrationsszenarios. Weiterhin sind auf der Detailseite Buttons zum Starten eines Szenarios und zur Weiterleitung zur Seite für die Anzeige der verfügbaren Zielsysteme vorgesehen. Wird der Button „Szenario starten“ gedrückt, wird die Szenariokonfigurationsdatei des zuvor ausgewählten Demonstrationsszenarios eingelesen und an die Agentenanwendungen auf den Zielplattformen übertragen. Danach soll der Benutzer wieder zurück auf die Startseite geleitet werden.

Abbildung 4.5: Szenariodetailseite des Szenarioassistenten

Wird in der Detailansicht der Button zur Anzeige der verfügbaren Zielsysteme gedrückt, soll dem Benutzer eine Listenansicht der verfügbaren Zielsysteme angezeigt werden (Abbildung 4.6). Jeder Listeneintrag für ein Zielsystem enthält neben dem Namen auch den Status des Zielsystems. Bei einem verfügbaren System hat der Listeneintrag eine grüne Hintergrundfarbe; bei nicht verfügbaren Systemen wird eine rote Hintergrundfarbe angezeigt.

Abbildung 4.6: Anzeige der verfügbaren Zielsysteme

4.5 Agentenanwendung

In den noch folgenden Abschnitten wird das Konzept für die Implementierung der Agentenanwendung dargelegt.

4.5.1 Architektur

Basis der Architektur der Agentenanwendung bildet eine XMPP-Komponente, welche die gesamte Logik zur Kommunikation mit dem XMPP-Server enthält (Abbildung 4.7). Darauf aufbauend soll die Anwendung je eine Komponente für den Digitalen Lagetisch und die AMFIS-Bodenkontrollstation mit deren Teilsystemen beinhalten. Die oberste Schicht der Anwendung bildet eine Komponente zur Steuerung der gesamten Agentenanwendung. In dieser Komponente werden die Module, die in der Konfigurationsdatei des Agenten festgelegt werden, erzeugt.

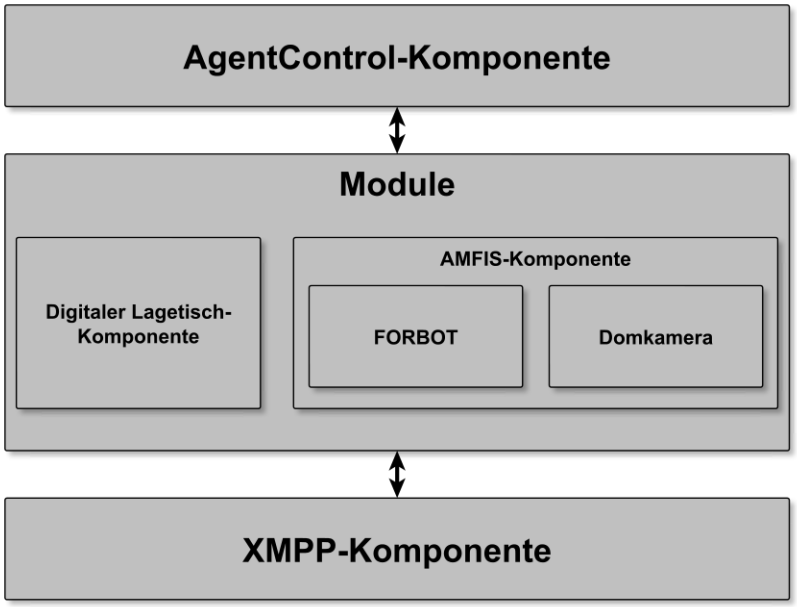


Abbildung 4.7: Architektur der Agentenanwendung

4.5.2 Konfiguration

Um die benötigten Module für die Ausführung eines Demonstrationsszenarios in der Agentenanwendung zu erzeugen, muss dieser vor dem Starten auf der Zielplattform entsprechend konfiguriert werden. Ein Modul stellt im Kontext dieser Arbeit ein Zielsystem einer Zielplattform dar, das an einem Demonstrationsszenario beteiligt ist. Die Konfiguration und Parametrisierung dieser Module soll in der Agentenanwendung über eine externe XML-Datei, die eingelesen und ausgewertet wird, erfolgen. Dazu werden die Module in der XML-Datei eingetragen und bei der Ausführung erzeugt. Abbildung 4.8 zeigt die XML Schema-Datei, welche die Basis der Konfigurationsdatei für die Agentenanwendung darstellt.

Die Konfigurationsdatei soll neben dem Stammelement <agentConfig> ein oder mehrere <module>-Elemente enthalten. Ein <module>-Element enthält Unterelemente zur Identifikation des Zielsystems auf der Zielplattform. Es enthält neben einem Kindelement das angibt ob das Zielsystem auf der Plattform aktiv geschaltet wird, weitere Kindelemente für den Benutzernamen und das Passwort, mit dem das Zielsystem beim XMPP-Server registriert und angemeldet wird.

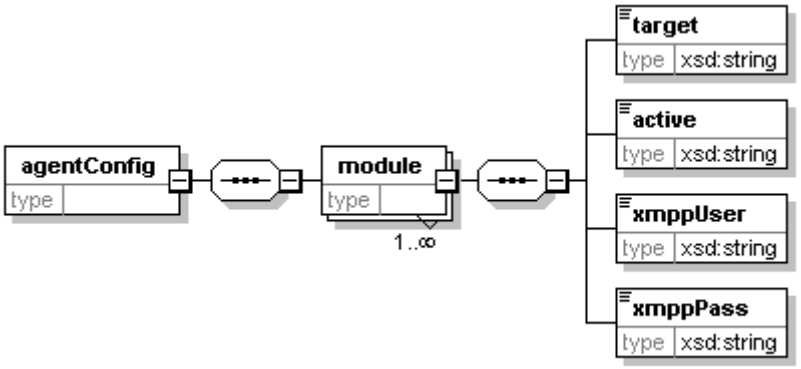


Abbildung 4.8: XML Schema-Datei der Agentenkonfiguration

Um die Verarbeitung von XML-basierten Daten in der Agentenanwendung zu erleichtern wird die Technologie Java Architecture for XML Binding (JAXB) eingesetzt. JAXB ermöglicht die einfache Generierung von Java-Klassen aus XML Schema-Dateien und bietet Funktionen zum Marshalling und Unmarshalling von XML-Daten. In der Version 2.0.3 ist JAXB bereits fester Bestandteil des Java SDK 6.0. Die Verwendung dieses gängigen Standards bietet sich für das Erfassen von XML-basierten Dateien an. Neben der externen Konfigurationsdatei sollen auch die Szenariokonfigurationsdateien in der Agentenanwendung mit JAXB deserialisiert werden.

4.5.3 Komponenten

Komponente AgentControl

Das Einlesen der externen Konfigurationsdatei soll in dieser Komponente erfolgen. Somit stellt sie den Einstiegspunkt der Agentenanwendung dar und ist zugleich auch für die Erzeugung der Module, die zuvor in der Konfigurationsdatei eingetragen, parametrisiert und aktiviert wurden, verantwortlich. Dies hat zur Folge, dass in dieser Komponente neben Funktionen zum Einlesen und Auswerten der Konfigurationsdatei auch die benötigte Programmlogik zur Erzeugung der Module realisiert werden soll.

Komponente AMFIS

Die AMFIS-Komponente bildet die Basis für den Datenaustausch mit der AMFIS-Bodenkontrollstation und deren Zielsystemen. Um eine Verbindung zum AMFIS-Konnektor aufzubauen, existiert, wie im Kapitel Anforderungsanalyse beschrieben, die Bibliothek JavaAmfisCom. Der Funktionsumfang der AMFIS-Komponente wird in der Anforderung ANF-19 genauer beschrieben. In der Agentenanwendung wird für diese Komponente das Package *amfis* erstellt.

Nach dem Aufbau einer Verbindung zum AMFIS-Konnektor wird jeder Client-Anwendung eine AMFIS-Nachricht mit der Beschreibung des Sensornetzes der AMFIS-Bodenkontrollstation zugesendet. Diese Beschreibung ist XML-basiert und enthält Informationen zum Aufbau des Sensornetzes. Ein Sensornetz besteht aus einem oder beliebig vielen Sensornetzwerken, die wiederum aus einem oder beliebig vielen Sensorknoten bestehen. Ein Sensorknoten stellt einen Sensorträger, wie beispielsweise den FORBOT, dar. Diese Sensorknoten bestehen, je nach Ausstattung eines Sensorträgers, aus einem oder mehreren Sensoren. Um diese Struktur in der Agentenanwendung abzubilden wird ein Datenmodell festgelegt, das im Klassendiagramm in der Abbildung 4.9 dargestellt wird. Für dieses Datenmodell wird das Package *model* angelegt.

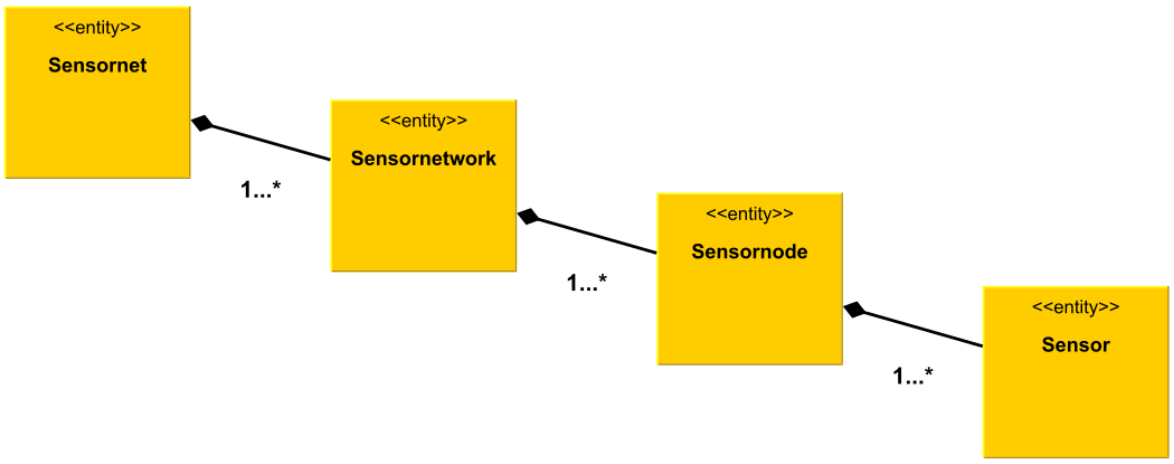


Abbildung 4.9: Klassendiagramm des AMFIS-Datenmodells

Komponente Digitaler Lagetisch

Die Kommunikation mit dem Zielsystem Digitaler Lagetisch wird in dieser Arbeit nur simuliert. Dazu wird ein Message Broker installiert. Dies erfolgt mit dem JMS-Provider ActiveMQ, der auf dem im Rahmen dieser Arbeit benutzten Entwicklungsrechner ausgeführt werden soll. Um Nachrichten an den Message Broker zu senden wird in der Komponente für den Digitalen Lagetisch ein JMS-Client, der für die Kommunikation mit dem Message Broker zuständig ist, implementiert.

Nach der abgeschlossenen Anforderungsanalyse wurden im weiteren Verlauf dieser Arbeit weitere Anforderungen an diese Komponente bekannt, die an dieser Stelle, da sie für die weitere Implementierung wichtig sind, nicht unerwähnt bleiben sollen. So soll eine JMS-Objektnachricht, die als Nutzlast ein Java-Objekt mit der ID des aufzurufenden Szenarios im Digitalen Lagetisch enthält, implementiert werden. Diese JMS-Objektnachricht soll an ein JMS-Topic gesendet werden, bei dem die JMS-Client-Anwendungen des Digitalen Lagetisches angemeldet sind. Weiterhin wird der Erhalt einer gesendeten JMS-Objektnachricht nicht bestätigt. Daraus folgt, da ein JMS-Topic verwendet werden soll, dass in dieser Komponente das „Publish/Subscribe“-Modell umgesetzt wird. Da gesendete JMS-Nachrichten nicht bestätigt werden enthält der zu entwickelnde JMS-Client lediglich einen JMS-Producer.

Weitere Funktionen, die diese Komponente bieten soll, werden in der Anforderung ANF-20 genauer beschrieben. Außerdem wird das Package *diglt* für diese Komponente vorgesehen.

Komponente XMPP

Diese Komponente enthält, wie in ANF-18 beschrieben, alle Funktionen, die für den Datenaustausch mit dem XMPP-Server erforderlich sind. Zur Umsetzung dieser Funktionen wird die Open-Source-Bibliothek Smack verwendet. Zur Kapselung dieser Komponente wird das Package *xmpp* in der Anwendung erstellt.

4.5.4 Zielsystemverfügbarkeit

Der Begriff der Zielsystemverfügbarkeit soll als Status für ein Zielsystem verstanden werden. Dazu wird unterschieden ob für ein Zielsystem die Einstellung von Konfigurationsparametern und somit auch die Beteiligung an einem Demonstrationsszenario möglich ist oder nicht. Die Agentenanwendung soll periodisch die Verfügbarkeit der Zielsysteme an den Multiuserchat des XMPP-Servers senden, sodass diese Information vom Szenarioassistenten empfangen, ausgewertet und an der Benutzungsoberfläche visualisiert werden kann.

Da die Zielplattform Digitaler Lagetisch nur simuliert werden kann, erfolgt hier keine Verfügbarkeitsprüfung. Vielmehr wird angenommen, dass der Digitale Lagetisch grundsätzlich verfügbar ist wenn eine Agentenanwendung darauf ausgeführt wird.

5 Implementierung

Nachdem im vorhergehenden Kapitel die Konzeption beschrieben wurde, erfolgt in diesem Kapitel die prototypische Implementierung der Konzepte. Zunächst soll die Grundlage für die Implementierung des Szenarioassistenten und der Agentenanwendung geschaffen werden. Dazu wird das Demonstrationsszenario umgesetzt, das vom Szenarioassistenten an die Agentenanwendung gesendet wird. Anschließend wird die Implementierung des Szenarioassistenten aufgezeigt. Das Kapitel endet mit der Implementierung der Agentenanwendung. Die kompletten Listings der Auszüge können in den Anhängen B.1 und B.2 eingesehen werden.

5.1 Demonstrationsszenario

In diesem Abschnitt soll aufgezeigt werden, wie die Szenariokonfigurationsdatei zum konzipierten Demonstrationsszenario erstellt wird. Dazu werden zunächst die Wegpunkte für den FORBOT, die nacheinander vom ExBa-Operator am Piloten-Arbeitsplatz angefahren werden sollen, in Form einer Wegpunktliste definiert. Danach werden die Bereiche bestimmt, auf die sich die Domkameras nach dem Einstellen der Parameter ausrichten sollen.

FORBOT

Der Landroboter FORBOT hat im Demonstrationsszenario die Aufgabe, den Innenbereich des Geländes zu erkunden. Zur Umsetzung dieser Aufgabe wird für den FORBOT eine Wegpunktliste in die AMFIS-Bodenkontrollstation übertragen. Um diese Wegpunktliste zu bestimmen, bietet die Lagedarstellung die Möglichkeit eine Wegpunktliste, die aus einem oder beliebig vielen Wegpunkten besteht, zu erzeugen und in eine XML-Datei zu exportieren.

Der AMFIS-Nachrichtentyp „waypointlist“ ermöglicht das Übertragen einer Liste von Wegpunkten, welche die Parameter dieser Nachricht darstellen. Nach dem Übertragen dieser Nachricht können diese Wegpunkte nacheinander vom ExBa-Operator am Piloten-Arbeitsplatz mit einem Joystick angesteuert werden. Die einzelnen Wegpunkte bestehen dabei aus jeweils sechs Parametern: Dem Breiten- und Längengrad, der Höhenänderung relativ zum vorherigen Wegpunkt in Metern, der Geschwindigkeit in Metern pro Sekunde, der Ausrichtung des Sensorträgers am Wegpunkt in Grad und der Verweildauer am Wegpunkt in Sekunden. Alle Parameter, außer der Ausrichtung und der Verweildauer, die ganzzahlig eingetragen werden, sind durch Fließkommazahlen definiert. Für jeden Wegpunkt müssen alle Parameter übertragen werden, wobei die Parameter der Höhe und der Geschwindigkeit sowie der Ausrichtung und der Verweildauer mit dem Wert null belegt sein dürfen. Bei der Definition dieser Parameter für den FORBOT werden nur der Breiten- und Längengrad angegeben, da die Angabe der restlichen Parameter als nicht sinnvoll erachtet wird.

Nachdem die Wegpunktliste in der Benutzungsoberfläche der AMFIS-Bodenkontrollstation definiert ist, wird diese exportiert. Der Wert dieser Wegpunktliste wird anschließend in das <value>-Element der Konfiguration für den FORBOT übernommen. Danach werden die Werte für die Elemente <nodeId> und <operation> eingetragen (Listing 5.1).

```
<node>
  <nodeId>amfis-48</nodeId>
  <operation>waypointlist</operation>
  <value>49.015861;8.426263;0.0;0.0;0;0;... </value>
</node>
```

Listing 5.1: Konfiguration für den FORBOT

Axis-Dome 1

Um die Parameter für die Konfiguration der Domkamera Axis-Dome 1 zu bestimmen wird die Domkamera am AMFIS-Piloten-Arbeitsplatz auf den gewünschten Ausrichtungsbereich gesteuert. Hier wird darauf geachtet, dass die Domkamera auf den Eingangsbereich des Fraunhofer IOSB ausgerichtet ist (Abbildung 5.1).

Abbildung 5.1: Ausrichtung der Domkamera Axis-Dome 1

Die für diese Ausrichtung der Domkamera benötigten Parameter für den AMFIS-Nachrichtentyp „lookat“ sind der Breitengrad, der Längengrad und die Höhe in Metern. Nachdem die Domkamera ausgerichtet ist können diese Parameter im Piloten-Arbeitsplatz abgelesen und in das <value>-Element der Konfiguration für die Domkamera Axis-Dome 1 übernommen werden. Weiterhin wird der eindeutige Bezeichner und die Aufgabe festgelegt. Listing 5.2 zeigt das <node>-Element der Konfiguration für die Domkamera Axis-Dome 1.

```
<node>
  <nodeId>amfis-35</nodeId>
  <operation>lookat</operation>
  <value>49.01534;8.426347;0</value>
</node>
```

Listing 5.2: Konfiguration der Domkamera Axis-Dome 1

Axis-Dome 2

Mit dem gleichen Ansatz wie bei der Domkamera Axis-Dome 1 werden für die zweite Domkamera die Parameter für das <value>-Element bestimmt. Um eine bessere Überwachung des Geländes zu gewährleisten wird für die zweite Domkamera ein anderer Ausrichtungsbereich gewählt (Abbildung 5.2).

Abbildung 5.2: Ausrichtung der Domkamera Axis-Dome 2

Die Domkamera Axis-Dome 2 soll sich in diesem Demonstrationsszenario auf den Wareneingangsbereich des Fraunhofer IOSB ausrichten und Bildmaterial liefern. Die Konfiguration für die Domkamera Axis-Dome 2 wird dazu wie in Listing 5.3 dargestellt definiert.

```
<node>
  <nodeId>amfis-36</nodeId>
  <operation>lookat</operation>
  <value>49.01563;8.426079;0</value>
</node>
```

Listing 5.3: Konfiguration der Domkamera Axis-Dome 2

Digitaler Lagetisch

Für den Digitalen Lagetisch und dessen Konfiguration wird das in Listing 5.4 gezeigte <node>-Element definiert. Das Element enthält im Kindelement <nodeId> den eindeutigen Bezeichner für dieses Zielsystem. Im <operation>-Element wird die Aufgabe „requestScenario“ festgelegt, die den Aufruf eines Szenarios im Digitalen Lagetisch beschreibt. Das <value>-Element beinhaltet die ID des aufzurufenden Szenarios im Digitalen Lagetisch.

```
<node>
  <nodeId>diglt-1</nodeId>
  <operation>requestScenario</operation>
  <value>1</value>
</node>
```

Listing 5.4: Konfiguration des Digitalen Lagetisches

5.2 Szenarioassistent

Aus den Anforderungen für die Implementierung des Szenarioassistenten ergeben sich verschiedene Rahmenbedingungen für die eingesetzte Entwicklungsumgebung. So soll die Oberfläche der Anwendung unter Verwendung eines HTML5-basierten Frameworks erstellt werden. Dazu wird jQuery Mobile eingesetzt. jQuery Mobile bietet für die Erstellung von mobilen Webseiten verschiedene Oberflächen-Komponenten, die auf HTML5, JavaScript und CSS basieren und bei der Erstellung der Oberfläche des Szenarioassistenten genutzt werden.

Die gesamte Anwendungslogik des Szenarioassistenten wird in JavaScript umgesetzt. Um den Szenarioassistenten, der vollständig auf Web-Technologien basiert, nativ auf einem Smartphone oder Tablet-PC installieren zu können, wird das Framework PhoneGap genutzt. Mit PhoneGap kann der Szenarioassistent zu einer nativen App kompiliert und anschließend installiert werden.

Um eine PhoneGap-Anwendung für Android-Geräte zu entwickeln muss außerdem die integrierte Entwicklungsumgebung Eclipse mit dem Android Development Tool Plug-in und ein Android SDK installiert werden.

Da die benötigten Funktionen für den Austausch von Daten mit dem XMPP-Server ebenfalls in JavaScript implementiert werden müssen, wird die auf JavaScript basierende XMPP-Bibliothek Strophe.js verwendet. Mit JavaScript ist der Aufbau von langlebigen TCP-Verbindungen nicht möglich, sodass für die Kommunikation mit dem XMPP-Server die XMPP-Protokollerweiterung BOSH eingesetzt wird. Dazu nutzt BOSH die Long Polling-Technik [Mof10].

5.2.1 Architektur

Die konzipierte Architektur des Szenarioassistenten soll neben Komponenten für die Präsentation und die Anwendungslogik eine Komponente zur Kapselung aller Funktionen für die Kommunikation mit dem XMPP-Server beinhalten. Um diese Architektur umzusetzen werden zuerst verschiedene JavaScript- und HTML-Dateien in der Struktur des zuvor angelegten PhoneGap-Projekts erzeugt.

Komponente Präsentation

Für diese Komponente werden HTML-Seiten, die für die Darstellung der Oberfläche zuständig sind, erstellt. Die Datei *index.html* beinhaltet das HMTL-Markup der Seite, die nach dem Starten der App angezeigt wird. Auf dieser Seite ist die Oberfläche mit der Listenansicht der Demonstrationsszenarien definiert. Zur Anzeige und Definition der Inhalte für die Seite, welche die Detailinformationen zum umgesetzten Demonstrationsszenario anzeigt, wird die Datei *scenarioDetailsPageOne.html* erstellt. Die Oberfläche für die Anzeige der verfügbaren Zielsysteme wird in der Datei *deviceStatusPage.html* realisiert.

Komponente Anwendungslogik

Die Komponente für die Anwendungslogik des Szenarioassistenten wird in der JavaScript-Datei *app.js* umgesetzt. Da diese Komponente für die Steuerung der Anwendungslogik verantwortlich ist, werden in dieser Datei verschiedene Event-Handler registriert.

Komponente XMPP-Client

In der JavaScript-Datei *xmppmanager.js* werden alle Funktionen, die für die Kommunikation mit dem XMPP-Server nötig sind, gekapselt. Dazu ist in dieser Datei das JavaScript-Objekt *XmppConnection* definiert, das die Bibliothek Strophe.js, unter anderem für die Erzeugung einer XMPP-Verbindung, nutzt. Weiterhin sind auch verschiedene Event-Handler und JavaScript-Funktionen enthalten.

Initialisierung der PhoneGap-Anwendung

PhoneGap-Anwendungen werden in der WebView-Komponente der Plattform, auf welcher sie installiert werden, angezeigt. Eine WebView ist ein Anzeigeelement für die Darstellung von Webseiten. Im Fall der Android-Plattform lädt PhoneGap das HTML5, JavaScript und CSS in den Container *android.webkit.WebView* und die Ausführung wird dabei von der darunter liegenden WebKit Rendering Engine übernommen. Das Listing 5.5 zeigt die Initialisierung in der Klasse *MainAcitvity* des Szenarioassistenten. Dazu wird die Klasse *org.apache.cordova.DroidGap*, die von PhoneGap zur Verfügung gestellt wird, importiert. Die ursprüngliche Erweiterung durch die Klasse *android.app.Activity* muss mit der neu eingebundenen Klasse ersetzt werden. Damit in der *WebView* die Startseite des Szenarioassistenten geladen wird, muss die Methode *setContentView* durch die Methode *super.loadUrl*, welche als Argument die zu ladende Startseite übergeben bekommt, ersetzt werden [And12b].

```
import org.apache.cordova.DroidGap;
import android.os.Bundle;

public class MainActivity extends DroidGap {

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        super.setIntegerProperty("loadUrlTimeoutValue", 60000);
        super.loadUrl("file:///android_asset/www/index.html");
    }

}
```

Listing 5.5: Initialisierung der PhoneGap-Anwendung

5.2.2 Oberfläche und Funktionen

Die Umsetzung der Oberfläche des Szenarioassistenten erfolgt ausschließlich unter Verwendung von UI-Komponenten der JavaScript-Bibliothek jQuery Mobile. Eingebunden wird die Bibliothek im Kopfbereich des HTML5-Markups der Startseite. Listing 5.6 zeigt einen Auszug der Datei *index.html*, in welcher die Bibliothek eingebunden und initialisiert wird. Da jQuery Mobile auf der JavaScript-Bibliothek jQuery aufbaut muss darauf geachtet werden, dass die JavaScript-Datei von jQuery vor der JavaScript-Datei von jQuery Mobile aufgeführt wird.

```
...
<head>
  ...
    <link rel="stylesheet" href="css/jquery.mobile.structure.css" />
    <link rel="stylesheet" href="css/jquery.mobile.theme.css" />
    <script type="text/javascript" src="js/lib/jquery.js"></script>
    <script type="text/javascript" src="js/lib/jquery.mobile.js"></script>
    ...
</head>
...
```

Listing 5.6: Einbinden der JavaScript-Bibliothek jQuery Mobile

Anhand der Benutzung des Szenarioassistenten für die Einstellung eines Demonstrationsszenarios in die Zielplattformen sollen nun die mit jQuery Mobile umgesetzten Oberflächen gezeigt werden.

Wird die Anwendung gestartet erscheint zunächst die Startseite (Abbildung 5.3). Damit der ExBa-Nutzer jederzeit sehen kann, ob überhaupt eine Verbindung zum XMPP-Server hergestellt werden konnte, wird der Verbindungsstatus zum XMPP-Server angezeigt. Danach erfolgt eine Auflistung der verfügbaren Szenarien. Jeder Listeneintrag für ein Szenario enthält die Art, den Ort und ein Bild des Szenarios. Um das Aussehen der Listenansicht zu verdeutlichen wurden zusätzlich zum ersten Eintrag für das im Rahmen dieser Arbeit entwickelte Demonstrationsszenario, beispielhaft zwei weitere Listeneinträge für Szenarien erstellt.

Wählt der ExBa-Nutzer ein Demonstrationsszenario durch Antippen des Listeneintrags aus, wird er auf die zugehörige Detailseite des Szenarios weitergeleitet. Auf dieser Seite werden die Detailinformationen zum ausgewählten Szenario angezeigt (Abbildung 5.4). Außerdem enthält die Seite zwei Buttons. Wird der Button „Zielsysteme“ gedrückt, erscheint die Übersichtseite für die Anzeige der verfügbaren Zielsysteme. Diese Anzeige erfolgt ebenfalls in einer Listenansicht (Abbildung 5.5). Jeder Listeneintrag zeigt durch seine Hintergrundfarbe und ein Icon an, ob das System verfügbar ist oder nicht. Ein roter Hintergrund signalisiert, dass das System nicht verfügbar ist, während ein grüner Hintergrund das System als verfügbar markiert. Der Button „Szenario starten“ veranlasst die Ausführung des Demonstrationsszenarios und somit das Senden der entsprechenden Szenariokonfigurationsdatei an den Multiuserchat des XMPP-Servers. Anschließend wird der ExBa-Nutzer wieder zurück zur Startseite der Anwendung weitergeleitet.

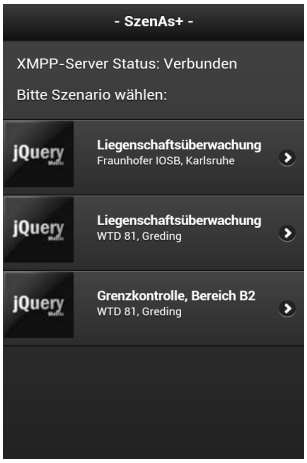


Abbildung 5.3: Startseite

Abbildung 5.4: Detailseite

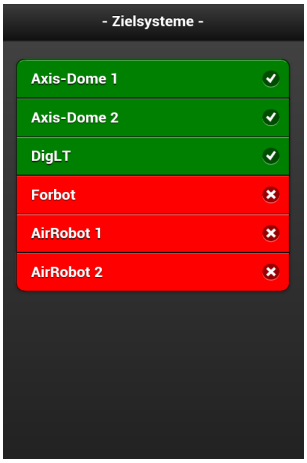


Abbildung 5.5: Zielsysteme

Nachdem die implementierte Oberfläche des Szenarioassistenten gezeigt wurde soll nun die Implementierung ausgewählter Funktionen folgen. Zunächst soll beschrieben werden, wie das Senden der Szenariokonfigurationsdatei in einer XMPP-Nachricht über das XMP-Protokoll für den Szenarioassistenten umgesetzt wird. Um XMPP-Nachrichten senden zu können muss zunächst eine Verbindung aufgebaut werden. Wird der Szenarioassistent gestartet erfolgt automatisch ein Verbindungsaufbau zum XMPP-Server. Dazu wird die Möglichkeit genutzt, jQuery-Funktionen an DOM-Events zu binden. Die Funktion *doConnect* (Listing 5.7) in der Datei *xmppmanager.js* löst nach Initialisierung der Anwendung das Event „connect“ aus und übergibt den JID und das Passwort für die Anmeldung beim XMPP-Server an die Funktion, welche dieses Event behandelt.

```
function doConnect() {

   $(document).trigger('connect', {
       jid : XmppConnection.jid,
       password : XmppConnection.password
   });

}
```

Listing 5.7: Funktion doConnect

Die Funktion, die an dieses Event gebunden wurde, instanziiert zunächst ein *Strophe.Connection*-Objekt und startet den Verbindungsprozess. Das Listing 5.8 zeigt einen Auszug der Funktion für den Aufbau einer Verbindung zum XMPP-Server. Wurde eine Verbindung hergestellt, wird das Event „connected“ ausgelöst.

```
$(document).bind('connect', function (ev, data) {
    var conn = new Strophe.Connection(XmppConnection.url);

    conn.connect(data.jid, data.password, function (status) {
        if (status === Strophe.Status.CONNECTED) {
          $('#connectionState').html("XMPP-Server Status: Verbunden");
            $(document).trigger('connected');
        }

  ...

});
```

Listing 5.8: Auszug Event-Handler „connect“

Listing 5.9 zeigt die Implementierung des Verhaltens, welches an das „connected“-Event gekoppelt wurde. An dieser Stelle wird nun ein <presence>-Stanza zur Signalisierung der Anwesenheit an den XMPP-Server gesendet. Um einen Multiuserchat im XMPP-Server einzurichten wird anschließend das Strophe-Plug-in für die XMPP-Protokollerweiterung des Multiuserchats initialisiert. Mit dem Senden eines weiteren <presence>-Stanzas an den Multiuserchat wird diesem anschließend beigetreten. Um XMPP-Nachrichten vom Multiuserchat zu empfangen wird abschließend ein Event-Handler für eintreffende XMPP-Nachrichten registriert.

```
$(document).bind('connected', function () {

  XmppConnection.connection.send($pres().c('priority').t('-1'));

  XmppConnection.connection.muc.init(XmppConnection.connection);

  XmppConnection.connection.muc.createInstantRoom(XmppConnection.room,
      createRoomSuccess, createRoomError);

  XmppConnection.connection.send($pres({to: XmppConnection.room + "/" +
      XmppConnection.nickname
  }).c("x", {xmlns: XmppConnection.NS_MUC}).c("history", {maxchars: '0'}));

  XmppConnection.connection.addHandler(XmppConnection.onGroupchatMessage,
            null, "message", "groupchat");

});
```

Listing 5.9: Event-Handler „connected“

Das eigentliche Senden der XMPP-Nachricht, welche die Szenariokonfiguration enthält, wird in der Datei *app.js* ausgeführt. Dazu wird ein Event-Handler (Listing 5.10) an das „click“-Event des Buttons für das Starten der Szenarioausführung gebunden, welcher auf dem mit Strophe erzeugten *connection*-Objekt die Funktion *send* aufruft und somit die Szenariokonfiguration im <body>-Element der XMPP-Nachricht an den Multiuserchat überträgt.

```
$(".btnStartScenario").live("click", function (event){

  var body = scenarioXML;

  XmppConnection.connection.send($msg({to: XmppConnection.room, type: "groupchat
      "}).c('body').t(scenarioXML));

  alert("Das Szenario wurde gestartet.");

});
```

Listing 5.10: Event-Handler für das Senden der Szenariokonfiguration

Abgesehen von den XMPP-Nachrichten, die für die Kommunikation mit dem XMPP-Server gesendet werden, ist für die Ausführung eines Demonstrationsszenarios durch den Szenarioassistenten nur die Implementierung einer Nachricht zum Senden der Szenariokonfiguration an den Multiuserchat nötig.

Anzeige der Zielsystemverfügbarkeit

Die Anzeige der Zielsystemverfügbarkeit erfolgt im Szenarioassistenten in einer Listenansicht. Zielsystemverfügbarkeitsnachrichten, die der Szenarioassistent von einer Agentenanwendung über den Multiuserchat empfängt, werden von der Funktion *onGroupchatMessage* in der Datei *xmppmanager.js* behandelt. Trifft eine Zielsystemverfügbarkeitsnachricht ein wird sie zunächst geparst. Dies erfolgt in der Funktion *getDeviceStatusDataFromXmlString* (Listing 5.11), die als Parameter die zu parsende Nachricht erwartet und ein JavaScript-Array zurückliefert, das die Zielsysteme in Form von JavaScript-Objekten enthält.

```
function getDeviceStatusDataFromXmlString(xmlString) {

    var doc = new DOMParser().parseFromString(xmlString, 'text/xml');
    var devices = doc.getElementsByTagName('node');
    var deviceNode = {};
    var nodes = [];
    var node;

    for (var i = 0; i < devices.length; i++) {
      node = devices[i];
      deviceNode.nodeId = $(node).find( "nodeId" ).text();
      deviceNode.name = $(node).find( "name" ).text();
      nodes.push(deviceNode);
      deviceNode = {};

    }

    return nodes;

}
```

Listing 5.11: Parsen einer Zielsystemverfügbarkeitsnachricht

Für jedes dieser Zielsysteme wird ein JavaScript-Objekt angelegt, das zwei Eigenschaften besitzt und in einem JavaScript-Array gespeichert wird. Die Eigenschaft *name* enthält den Namen des Zielsystems, während die Eigenschaft *timeoutCounter* eine Zählvariable darstellt, der ein initialer Wert zugewiesen wird.

In der Datei *app.js* wird beim Starten der Anwendung eine Timer-Funktion ausgeführt (Listing 5.12), die periodisch die Funktion *decrementTimeoutCounter* auf dem *XmppConnection*-Objekt aufruft.

```
window.setInterval(XmppConnection.decrementTimeoutCounter, 1000);
```

Listing 5.12: Timer-Funktion für die Funktion *decrementTimeoutCounter*

Die Funktion *decrementTimeoutCounter* (Listing 5.13) in der Datei *xmppmanager.js* iteriert über das JavaScript-Array, in welchem die Zielsysteme bei Ankunft einer Zielsystemverfügbarkeitnachricht gespeichert werden, und dekrementiert die Eigenschaft *timeoutCounter* jedes Zielsystem-Objekts.

```
decrementTimeoutCounter: function () {

      if(XmppConnection.participantsToDisplay.length !== 0) {

        for(var i = 0; i < XmppConnection.participantsToDisplay.length; ++i) {
              if(XmppConnection.participantsToDisplay[i].timeoutCounter < 0) {
                XmppConnection.participantsToDisplay[i].timeoutCounter = 0;
              } else {
                XmppConnection.participantsToDisplay[i].timeoutCounter -= 1;
              }
          }
...
```

Listing 5.13: Funktion *decrementTimeoutCounter*

Es wird ebenfalls eine Timer-Funktion für die Funktion *displayDeviceStatus* gestartet, die für das Darstellen der Zielsysteme in der Listenansicht zuständig ist. In der Funktion *displayDeviceStatus* (Listing 5.14) wird dann über das JavaScript-Array iteriert, um die Eigenschaft *timeoutCounter* der Zielsystem-Objekte zu prüfen. Besitzt diese Eigenschaft einen Wert, der größer als null ist, so wird das Zielsystem in der Listenansicht als verfügbar angezeigt. Ist der Wert null, wird es als nicht verfügbar angezeigt. Dazu erfolgt ein dynamischer Aufbau des Markups für die Listenansicht. Bei jeder neu eintreffenden Zielsystemverfügbarkeitsnachricht wird die Eigenschaft *timeoutCounter* eines Zielsystems in der Funktion *onGroupchatMessage* wieder auf den initialen Wert gesetzt und das Zielsystem wird als verfügbar eingestuft und dementsprechend angezeigt.

```
displayDeviceStatus: function () {

      var devices = ['Forbot', 'Axis-Dome 1', 'Axis-Dome 2', 'AirRobot 1', '
          AirRobot 2', 'DigLT'];

      if(XmppConnection.participantsToDisplay.length !== 0) {

      $("#lv_deviceStatus").html("<ul id='lv_deviceStatus' data-role='listview'
          data-inset='true'></ul>");

      var list = "";

      var devicesOnline = [];

      for(var i = 0; i < devices.length; ++i) {
          for(var j = 0; j < XmppConnection.participantsToDisplay.length; ++j) {
            if(devices[i] === XmppConnection.participantsToDisplay[j].name &&
                XmppConnection.participantsToDisplay[j].timeoutCounter > 0) {
              list += "<li data-icon=\"check\" style=\"background: green;\"><a
                  href=\"#\">" + devices[i] + "</a></li>";
              devicesOnline.push(devices[i]);
            }
          }
        }
      }

      if(devicesOnline !== undefined) {
        for(var i = 0; i < devices.length; ++i) {
          if(devicesOnline.contains(devices[i])) {
            continue;
          } else {
            list += "<li data-icon=\"delete\" style=\"background: red;\"><a href
                =\"#\">" + devices[i] + "</a></li>";
          }
        }
      }

      $("#lv_deviceStatus").html(list);
      $("#lv_deviceStatus").listview("refresh");

}
```

Listing 5.14: Funktion *displayDeviceStatus*

5.3 Agentenanwendung

In diesem Abschnitt soll die Implementierung der Agentenanwendung beschrieben werden. Dazu wird zuerst aufgezeigt, wie die Anforderung für die Konfiguration der Anwendung implementiert wird, um anschließend einen Überblick über die Umsetzung der einzelnen Komponenten in der Agentenanwendung und deren Aufgaben zu geben.

5.3.1 Konfiguration

Die Konfiguration der Agentenanwendung soll, wie im Kapitel Konzeption beschrieben, über eine externe XML-basierte Konfigurationsdatei erfolgen. Dazu wird zunächst eine XML Schema-Datei für die Konfiguration spezifiziert. Aus dieser XML Schema-Datei wird dann ein Instanzdokument der eigentlichen XML-Konfigurationsdatei generiert.

Um eine Objektrepräsentation der Konfigurationsdatei in der Agentenanwendung zu erhalten werden mit dem JAXB Binding Compiler „xjc“, aus der zuvor erstellten XML Schema-Datei, die benötigten Java-Klassen für die Agentenanwendung generiert. Dazu muss beim Aufruf des Binding Compilers nur die Schema-Datei angegeben werden, aus welcher die Klassen generiert werden sollen. Diese Klassen erzeugen dann beim späteren Deserialisierungsprozess in der Agentenanwendung ein Java-Objekt der eingelesenen Konfigurationsdatei. Der Inhalt der Konfigurationsdatei kann anschließend bequem über die Getter-Methoden des Java-Objektes gelesen werden.

Ein XML-Instanzdokument (Listing 5.15) der Agentenkonfiguration besteht aus mindestens einem <module>-Element, das ein Zielsystem darstellt und aus folgenden Kindelementen besteht:

- <target>: Das Zielsystem, das dieses <module>-Element beschreibt. Ein möglicher Wert für dieses Element besteht immer aus einem Präfix der Zielplattform, „amfis“ oder „diglt“, gefolgt von einem Bindestrich-Minus und der ID des Zielsystems auf der Zielplattform.
- <active>: Hier erfolgt die Angabe ob das Zielsystem in der Agentenanwendung aktiv geschaltet werden soll. Mögliche Werte sind „true“ oder „false“.
- <xmppUser>: Der Benutzername mit dem das Zielsystem am XMPP-Server angemeldet werden soll.
- <xmppPass>: Das Passwort mit dem das Zielsystem am XMPP-Server angemeldet werden soll.

```
...
<module>
  <target>diglt-1</target>
  <active>true</active>
  <xmppUser>DigLT</xmppUser>
  <xmppPass>diglt1234</xmppPass>
</module>
...
```

Listing 5.15: Auszug aus einem XML-Instanzdokument einer Agentenkonfiguration

5.3.2 Komponenten

Komponente AgentControl

Die Komponente AgentControl ist der Einstiegspunkt der Agentenanwendung. Für diese Komponente wird das Entwurfsmuster Factory Pattern zur Erzeugung von Modulen in der Anwendung umgesetzt. In der Abbildung 5.6 sind alle Klassen dieser Komponente abgebildet. Aus Gründen der Übersichtlichkeit wird auf die Darstellung der Attribute in den einzelnen Klassen verzichtet.

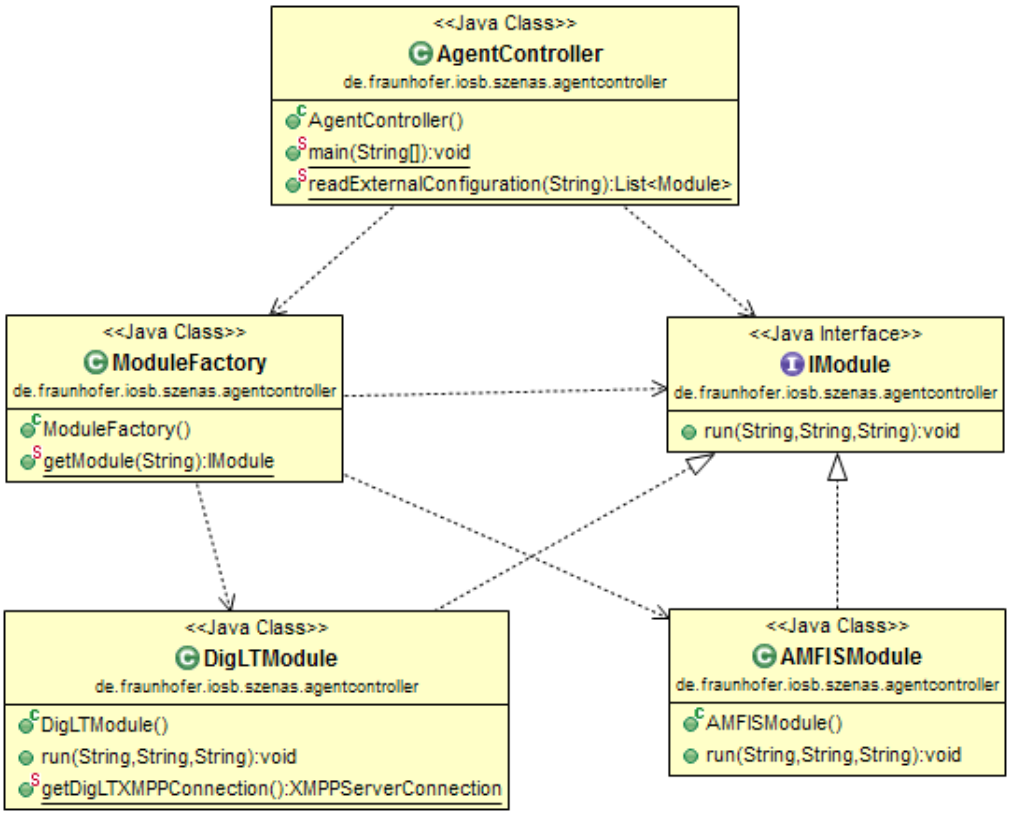


Abbildung 5.6: Klassendiagramm der Komponente AgentControl

Ein Modul der Agentenanwendung muss das Java-Interface *IModule* und dessen Methode *run* implementieren (Listing 5.16). Diese Methode erwartet zur Modulerzeugung drei Parameter: Das Zielsystem, für das ein Modul erzeugt werden soll, den XMPP-Benutzernamen und das XMPP-Passwort. Diese Parameter werden für jedes zu aktivierende Zielsystem aus der externen Konfigurationsdatei eingelesen und beim Aufruf der Methode übergeben.

```
public interface IModule {

  void run(String target, String xmppUser, String xmppPass);

}
```

Listing 5.16: Interface IModule

Nach dem Starten der Agentenanwendung wird in der Klasse *AgentController* die externe Konfigurationsdatei eingelesen. Dies erfolgt durch den Aufruf der Methode *readExternalConfiguration*, deren Rückgabetyp eine Liste von *Module*-Objekten ist, welche die Parameter für die Modulerzeugung enthalten. Anschließend wird über diese Objekte iteriert und die entsprechenden Module unter Verwendung der Klasse *Module Factory* erzeugt. Der folgende Programmabschnitt (Listing 5.17) der Methode *main* zeigt beispielhaft die Instanziierung des Moduls Digitaler Lagetisch.

Ist das Modul für den Digitalen Lagetisch in der externen Konfigurationsdatei der Agentenanwendung aktiviert, dann wird in diesem Abschnitt zuerst eine XMPP-Verbindung unter Verwendung der XMPP-Komponente mit dem JID „DigLT-Status@server.tld/DigLT-Status“ erstellt und einem Timer-Objekt übergeben. Der angemeldete JID sendet nun periodisch die Verfügbarkeit des Digitalen Lagetisches an den Multiuserchat. Anschließend wird unter Verwendung der Klasse *ModuleFactory* das Modul für den Digitalen Lagetisch instanziiert und mit Aufruf der Methode *run* gestartet.

```
for(Module module : readExternalConfiguration(path)) {
  if(module.getActive().equals("true") && module.getTarget().startsWith("diglt")
      ) {

    Timer digLTTimer = new Timer();

    digLTTimer.schedule(new DigLTStatusTask(new XMPPServerConnection(Constants.
        DIGLT_XMPP_STATUS_BOT_USERNAME, Constants.DIGLT_XMPP_STATUS_BOT_PASSWORD
        ,
            Constants.DIGLT_XMPP_STATUS_BOT_NICKNAME, Constants.
                DIGLT_XMPP_STATUS_BOT_RESOURCE)), 5000/* delay ms*/, 5000/*
                period ms*/);

    IModule digLTModule = ModuleFactory.getModule(module.getTarget());
    digLTModule.run(module.getTarget(), module.getXmppUser(), module.getXmppPass
        ());
...
```

Listing 5.17: Instanziierung des Moduls Digitaler Lagetisch

Die Klasse *DigLTModule* repräsentiert das eigentliche Modul für den Digitalen Lagetisch. Nach Instanziierung und Aufruf der Methode *run* wird in dieser Klasse eine XMPP-Verbindung für das Modul erzeugt (Listing 5.18), sodass dieses Zielsystem am Multiuserchat des XMPP-Servers angemeldet ist und eine XMPP-Nachricht, welche die Szenariokonfiguration enthält, empfangen kann.

```
...
@Override
public void run(String target, String xmppUser, String xmppPass) {

  digLTConnection = new XMPPServerConnection(xmppUser, xmppPass, xmppUser,
      xmppUser);

  digLTConnection.login();
  digLTConnection.addPacketListener(null);
...
```

Listing 5.18: Instanziierung einer XMPP-Verbindung für das Modul Digitaler Lagetisch

Sollen AMFIS-Module in der Anwendung erzeugt werden, erfolgt in der Klasse *AgentController* zuerst die Initialisierung einer Verbindung zum AMFIS-Konnektor (Listing 5.19). Dies ist nötig, um die Verfügbarkeit der Zielsysteme zu prüfen, weil dazu die Nachricht mit der Beschreibung des Sensornetzes und dessen Sensorträgern empfangen werden muss. Für das Senden der Zielsystemverfügbarkeitsnachrichten wird dann eine XMPP-Verbindung für den JID „AMFIS-Status@server.tld/AMFIS-Status“ erstellt. Für diese Verbindung wird ebenfalls ein Timer-Objekt erzeugt, um periodisch die Zielsystemverfügbarkeitsnachrichten mit dem angemeldeten JID an den Multiuserchat zu senden.

```
...
if(!isAmfisConnectionInitialized) {
  AmfisConnection amfisConnection = AmfisConnection.getInstance();
  amfisConnection.connect();
  amfisConnection.addListener();

  Timer timer = new Timer();
  timer.schedule(new DeviceStatusTask(), 5000/* delay ms*/, 1000/* period ms*/);
  timer.schedule(new AmfisDeviceStatusTask(new XMPPServerConnection( Constants.
      AMFIS_XMPP_STATUS_BOT_USERNAME, Constants.AMFIS_XMPP_STATUS_BOT_PASSWORD,
      Constants.AMFIS_XMPP_STATUS_BOT_NICKNAME, Constants.
      AMFIS_XMPP_STATUS_BOT_RESOURCE)), 5000/* delay ms*/, 5000/* period ms*/);

  isAmfisConnectionInitialized = true;

  }

  IModule amfisModule = ModuleFactory.getModule(module.getTarget());
  amfisModule.run(module.getTarget(), module.getXmppUser(), module.getXmppPass()
      );
...
```

Listing 5.19: Instanziierung eines Moduls für ein AMFIS-Zielsystem

AMFIS-Module werden in der Agentenanwendung durch die Klasse *AmfisModule* repräsentiert. In dieser Klasse wird ein *SensorModel*-Objekt erzeugt, welches ein Zielsystem und somit einen Sensorträger der AMFIS-Bodenkontrollstation darstellt. Das *SensorModel*-Objekt besitzt eine Referenz auf eine XMPP-Verbindung. Im Konstruktoraufruf für die Erzeugung des Objekts wird für die Instanziierung der XMPP-Verbindung eine neue XMPP-Verbindung erstellt und übergeben (Listing 5.20), sodass dieses Zielsystem am Multiuserchat angemeldet ist und eine Szenariokonfiguration empfangen kann.

```
...
@Override
public void run(String target, String xmppUser, String xmppPass) {
  Integer sensornodeId = Integer.valueOf(target.substring(6));

  SensornodeModel sensornodeModel = new SensornodeModel(sensornodeId, xmppUser,
      xmppPass, xmppUser, xmppUser, new XMPPServerConnection(xmppUser, xmppPass,
       xmppUser, xmppUser));

  sensornodeModel.getConnection().login();
  sensornodeModel.getConnection().addPacketListener(sensornodeModel);
...
```

Listing 5.20: Instanziierung einer XMPP-Verbindung für ein AMFIS-Zielsystem

Komponente AMFIS

Die AMFIS-Komponente setzt alle Funktionen für die Kommunikation mit dem AMFIS-Konnektor um. Dazu nutzt sie die Programmierschnittstelle der JavaAmfisCom-Bibliothek, die Methoden für den Auf- und Abbau einer Verbindung zum AMFIS-Konnektor und zum Senden und Empfangen von Nachrichten bereitstellt. Um eine Verbindung zum AMFIS-Konnektor herzustellen wird zunächst die Klasse *AmfisConnection* realisiert. Für diese Klasse wird das Entwurfsmuster Singleton eingesetzt, da die gesamte Kommunikation mit dem AMFIS-Konnektor über eine einzige Instanz eines *AmfisConnection*-Objekts erfolgen soll. Für den Verbindungsaufbau wird in der Methode *connect* der Klasse *AmfisConnection* zunächst ein *ConnectorClient*-Objekt instanziiert und darauf die Methode *connect* aufgerufen (Listing 5.21), sodass eine Verbindung zum AMFIS-Konnektor hergestellt wird und die kontinuierlich gesendeten Nachrichten des AMFIS-Konnektors empfangen werden können.

```
...
client = new ConnectorClient();
client.connect(Constants.AMFIS_CONNECTOR_HOST,Constants.AMFIS_CONNECTOR_PORT);
...
```

Listing 5.21: Verbindungsaufbau zum AMFIS-Konnektor

Um AMFIS-Nachrichten zu empfangen und auszuwerten wird in der Klasse *AmfisConnection* die innere Klasse *AmfisConnectionListener* umgesetzt, welche das Java-Interface *java.awt.event.ActionListener* implementieren und die Listener-Methode *actionPerformed* überschreiben muss. Sendet der AMFIS-Konnektor eine Nachricht, wird ein *ActionEvent* ausgelöst und die Methode *actionPerformed* aufgerufen, der als Argument ein *ActionEvent*-Objekt übergeben wird (Listing 5.22). Die Methode *getActionCommand*, die auf dem *ActionEvent*-Objekt aufgerufen wird, liefert eine Kennung in Form einer String-Zeichenkette, mit der unterschieden wird um welche AMFIS-Nachricht es sich handelt. Dies ist wichtig, weil der AMFIS-Konnektor wie bereits erwähnt, beim ersten Verbindungsaufbau jeder neu angemeldeten Client-Anwendung eine Nachricht schickt, die das Sensornetz mit den verschiedenen angemeldeten Sensorträgern der AMFIS-Bodenkontrollstation beschreibt. Diese Nachricht wird durch die Kennung „getSensorweb“ identifiziert, bei allen anderen AMFIS-Nachrichten lautet die Kennung „getMessage“.

Um die Informationen der Nachricht, welche die Beschreibung des Sensornetzes enthält, in der Agentenanwendung vorzuhalten, wird, wie in der Konzeption für die Agentenanwendung beschrieben, ein Datenmodell implementiert auf das die Nachricht beim Einlesen abgebildet werden kann. Listing 5.22 zeigt wie diese Nachricht eingelesen wird.

```
...
@Override
public void actionPerformed(ActionEvent event) {

  String cmd = event.getActionCommand();

  SensorwebModel sensorwebModel = new SensorwebModel();

  if("getSensorweb".equals(cmd)) {

    sensorweb = ((ConnectionReader) event.getSource()).getSensorweb();
```

```
      for (int i = 0; i < sensorweb.getSensornetworkCount(); ++i) {

        sensornetwork = sensorweb.getSensornetwork(i);

        sensorwebModel.getSensornetworks().add(new SensornetworkModel(
            sensornetwork.getId(), sensornetwork.getName(),
                sensornetwork.getDesc(), String.valueOf(sensornetwork.
                    getConnection()), sensornetwork.getAddress()));

        for(int j = 0; j < sensornetwork.getSensornodeCount(); ++j) {

          sensornode = sensornetwork.getSensornode(j);

          deviceStatus = new DeviceStatus(sensornode.getId(), sensornode.getName
              (), sensornode.getMobile());

          deviceStatusList.add(deviceStatus);

          sensorwebModel.getSensornetworks().get(i).getSensornodes().add(new
              SensornodeModel(sensornode.getId(), sensornode.getName(),
                  sensornode.getAlias(), sensornode.getMobile(), sensornode.
                      getPosition(), sensornode.getTimeout()));

            for(int k = 0; k < sensornode.getSensorCount(); ++k) {

              sensor = sensornode.getSensor(k);

              sensorwebModel.getSensornetworks().get(i).getSensornodes().get(j).
                  getSensors().add(new SensorModel(sensor.getId(), sensor.
                  getPriority(),
                              sensor.getClazz(), sensor.getManufacturer(),
                                sensor.getDesc(), sensor.getThreshold_min(),
                                  sensor.getThreshold_max()));
...
```

Listing 5.22: Einlesen der Beschreibung des Sensornetzes

Komponente Digitaler Lagetisch

In dieser Komponente wird ein JMS-Client für die Kommunikation mit dem ActiveMQ Message Broker des Digitalen Lagetisches realisiert. Diese Klasse baut zunächst in der Methode *start* eine Verbindung zum Message Broker des Digitalen Lagetisches auf, um anschließend eine JMS-Session zu erzeugen (Listing 5.23). Nachdem die JMS-Session erzeugt ist wird das Topic „SCENARIO_EXECUTION_TOPIC“ erstellt, an das die Nachricht für die Anforderung einer Szenarioausführung gesendet werden kann. Für das Senden der JMS-Objektnachricht an das Topic wird in der Klasse *DigLTJMSClient* ein Objekt vom Typ *javax.jms.MessageProducer* erzeugt.

Das Senden der Nachricht erfolgt in der Methode *requestScenarioExecution*, die als Parameter die ID eines Szenarios erwartet. In dieser Methode wird dann eine JMS-Objektnachricht mit der ID des Szenarios erstellt und an das Topic für die Ausführung des Szenarios gesendet.

```
public class DigLTJMSClient {
...
public void start() throws JMSException {
    connectionFactory = new ActiveMQConnectionFactory(Constants.DIGLT_BROKER_URL
        );

    connection = connectionFactory.createConnection();
    connection.start();

    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

    executionTopic = session.createTopic("SCENARIO_EXECUTION_TOPIC");
    scenarioExecutionProducer = session.createProducer(executionTopic);

                    scenarioExecutionProducer.setDeliveryMode(DeliveryMode.
                        NON_PERSISTENT);
...
public void requestScenarioExecution(int scenarioId) {
...
    executionObjectMsg = session.createObjectMessage(new ScenarioExecutionMsg(
        scenarioId));
      this.scenarioExecutionProducer.send(executionObjectMsg);
...
  }
...
}
```

Listing 5.23: Auszug der Klasse DigLTJMSClient

Komponente XMPP

Diese Komponente bildet die Basis für die gesamte Kommunikation zwischen einer Agentenanwendung auf einer Zielplattform und dem XMPP-Server. Sie kapselt alle Funktionen, um mit dem XMPP-Server zu kommunizieren. Dazu werden zunächst Methoden zum Anmelden und Einloggen beim XMPP-Server erstellt. Um den Multiuserchat des XMPP-Servers zu nutzen werden weiterhin Methoden zum Erstellen und Betreten eines Multiuserchats umgesetzt. Außerdem werden Methoden für das Senden und Empfangen von XMPP-Nachrichten benötigt und umgesetzt. Für die Implementierung all dieser Funktionen wurde die frei verfügbare Java-XMPP-Bibliothek Smack in der Version 3.2.2 eingesetzt. Das komplette Listing der Klasse *XMPPServerConnection* kann im Anhang B.2 eingesehen werden.

5.3.3 Nachrichten

Die Agentenanwendung hat die Aufgabe Informationen zur Verfügbarkeit der Zielsysteme an den Szenarioassistenten zu senden. Aus diesem Grund werden XML-basierte Nachrichten definiert, die im <body>-Element von XMPP-Nachrichten an den Multiuserchat gesendet und dort vom Szenarioassistenten empfangen werden können. Da die Agentenanwendung modular aufgebaut sein soll, werden für alle Zielplattformen, für die Module in der Agentenanwendung erzeugt werden können, eigene Nachrichten definiert, die sich nur durch das XML-Stammelement in der Nachricht unterscheiden.

Nachfolgend soll am Beispiel der Zielplattform AMFIS-Bodenkontrollstation der Aufbau dieser Nachrichten erläutert werden. Die Nachricht basiert auf einem XML-String und besteht aus dem Stammelement <AMFISDeviceStatus> (Listing 5.24). Für jedes aktuell verfügbare Zielsystem enthält das Stammelement <node>-Kindelemente. Ein <node>-Element besteht aus den folgenden Kindelementen:

<nodeId>: Bezeichnet die ID des Zielsystems

<name>: Bezeichnet den Namen des Zielsystems

Eine Nachricht, welche die Zielsystemverfügbarkeit der Zielplattform Digitaler Lagetisch enthält, besitzt grundsätzlich denselben Aufbau und unterscheidet sich nur vom Stammelement, das bei dieser Nachricht <DigLTDeviceStatus> lautet.

```
<AMFISDeviceStatus>
    <node>
    <nodeId>amfis-51</nodeId>
    <name>FORBOT</name>
  </node>
  <node>
    <nodeId>amfis-35</nodeId>
    <name>Axis-Dome 1</name>
  </node>
</AMFISDeviceStatus>
```

Listing 5.24: Zielsystemverfügbarkeitsnachricht der AMFIS-Bodenkontrollstation

JMS-Objektnachricht

Damit ein Szenario in der Lagedarstellung des Digitalen Lagetisches angezeigt werden kann, wird die ID des aufzurufenden Szenarios an das dafür zuständige Topic des Message Brokers gesendet. Für die Übertragung wird eine JMS-Objektnachricht implementiert. JMS-Objektnachrichten enthalten als Nutzlast einfache Java-Bean-Klassen, die für den Netzwerktransport das Java-Interface *Serializable* implementieren müssen [Sny11]. Das Listing 5.25 zeigt einen Auszug der Java-Bean-Klasse, mit der die ID eines Szenarios an den Message Broker gesendet werden kann.

```
public class ScenarioExecutionMsg implements Serializable {

  private static final long serialVersionUID = 8142969352178374969L;
  int scenarioId;

  public ScenarioExecutionMsg(int scenarioId) {
    this.scenarioId = scenarioId;
  }

  public int getScenarioId() {
    return scenarioId;
  }

  public void setScenarioId(int scenarioId) {
    this.scenarioId = scenarioId;
  }
...
```

Listing 5.25: JMS-Objektnachricht

5.3.4 Einstellen der Konfigurationsparameter

Erhält die XMPP-Verbindung eines Zielsystems eine Szenariokonfiguration, muss zunächst die Szenariokonfigurationsdatei eingelesen und ausgewertet werden. Um dies zu erleichtern wurde dafür ebenfalls JAXB genutzt. Aus der XML Schema-Datei *scenarioConfig.xsd* werden Klassen für die Agentenanwendung generiert, sodass eine eintreffende Szenariokonfiguration in eine Objektrepräsentation deserialisiert werden kann.

Jede XMPP-Verbindung eines Zielsystems verfügt über eine Listener-Methode für eintreffende XMPP-Nachrichten. Trifft eine XMPP-Nachricht mit einer Szenariokonfiguration ein, erfolgt in der Methode *addPacketListener* in der Klasse *XMPPServerConnection* zunächst der Deserialisierungsprozess (Listing 5.26). Danach wird über alle <node>-Elemente der Szenariokonfiguration iteriert um das <node>-Element zu finden, das für dieses Zielsystem bestimmt ist. Aus diesem Element wird anschließend das <operation>-Element und das <value>-Element eingelesen.

Für AMFIS-Zielsysteme sind die Werte „lookat“ und „waypointlist“ von Bedeutung. Das <value>-Element enthält die Parameter. Entsprechend des Werts im <operation>-Element, ist „lookat“ für die XMPP-Verbindung einer Domkamera bestimmt und „waypointlist“ für die XMPP-Verbindung des FORBOTs. Anhand des Werts wird dann der AMFIS-Nachrichtentyp mit den Parametern erzeugt und an den AMFIS-Konnektor gesendet. Somit sind die Parameter für ein Zielsystem in die AMFIS-Zielplattform eingestellt.

```
...
  ScenarioConfig scenarioConfig = XMLProcessor .getInstance().
      unmarshallScenarioconfigurationFromString(message.getBody());

  for(Node node : scenarioConfig.getNodes().getNode()) {

    nodeId = Integer.valueOf(node.getNodeId().substring(6));

    if(nodeId == sensornode.getSensornodeId()) {
      ...
      String operation = node.getOperation();

      if("lookat".equals(operation)) {
        sensornode.setEncodeBehavior(new EncodeLookAt());
        sensornode.performEncode(nodeId, node.getValue());
      } else if("waypointlist".equals(operation)) {
        sensornode.setEncodeBehavior(new EncodeWaypointlist());
        sensornode.performEncode(nodeId, node.getValue());
...
```

Listing 5.26: Auszug der Methode *addPacketListener* für AMFIS-Zielsysteme

Für den Digitalen Lagetisch wurde für den Aufruf eines Szenarios im <operation>-Element der Wert „scenarioRequest“ definiert. Der Wert des <value>-Elements enthält die ID des Szenarios, das im Digitalen Lagetisch aufgerufen werden soll. Nachdem die ID des aufzurufenden Szenarios eingelesen ist, wird eine Instanz der Klasse *DigLTJMSClient* erzeugt (Listing 5.27). Auf dieser Instanz wird dann die Methode *requestScenarioExecution* aufgerufen. Die Methode bekommt als Argument die ID, welche zuvor aus der Szenariokonfiguration eingelesen wurde, übergeben. Mit diesem Methodenaufruf wird der Parameter in das Zielsystem Digitaler Lagetisch eingestellt.

```
...
if("requestScenario".equals(operation))) {

  DigLTJMSClient client = new DigLTJMSClient();

  client.start();

  client.requestScenarioExecution(Integer.parseInt(digLTScenarioId));

}
...
```

Listing 5.27: Auszug der Methode *addPacketListener* für das Zielsystem Digitaler Lagetisch

5.3.5 Zielsystemverfügbarkeitsprüfung für die AMFIS-Bodenkontrollstation

Die Nachricht mit der Beschreibung des AMFIS-Sensornetzes, die der AMFIS-Konnektor nach dem Verbindungsaufbau sendet, wird genutzt, um die Prüfung der verfügbaren Zielsysteme in der AMFIS-Bodenkontrollstation umzusetzen. Erhält die Agentenanwendung diese Nachricht, wird für jeden Sensorträger, der in dieser Beschreibung aufgeführt ist, ein *DeviceStatus*-Objekt erzeugt. In diesen Objekten werden die Informationen zur Verfügbarkeit von Sensorträgern gespeichert. Alle *DeviceStatus*-Objekte werden in einer Java-ArrayList abgelegt.

Um nun die Prüfung der Zielsystemverfügbarkeit für die Zielsysteme der AMFIS-Bodenkontrollstation zu ermöglichen müssen die kontinuierlich gesendeten Nachrichten des AMFIS-Konnektors ausgewertet werden, da die Schnittstelle keine Möglichkeit bietet die Verfügbarkeit von Sensorträgern abzufragen. Bei der Auswertung der Nachrichten muss zwischen mobilen und stationären Sensorträgern unterschieden werden, da für deren Verfügbarkeitsprüfung unterschiedliche Bedingungen gelten.

Die Grundlage der Verfügbarkeitsprüfung für einen Sensorträger ist der Eingang einer AMFIS-Nachricht vom Typ „position“. Bei stationären Sensorträgern, wie den Domkameras, reicht der Eingang einer Positionsnachricht aus, um den Status des Sensorträgers als verfügbar und steuerbar einzustufen. Bei mobilen Sensorträgern, wie dem FORBOT, muss zusätzlich eine Prüfung der Positionswerte, die in der Nachricht enthalten sind, erfolgen. Sind alle Positionswerte ungleich null, kann dieser Sensorträger ebenfalls als verfügbar und steuerbar eingestuft werden. Listing 5.28 zeigt den Algorithmus der die Verfügbarkeit der Sensorträger ermittelt.

Trifft die Positionsnachricht eines Sensorträgers ein, wird über die angelegte Java-ArrayList iteriert und das Attribut *timeoutCounter* für dessen *DeviceStatus*-Objekt auf den initialen Wert gesetzt. Dies erfolgt mit dem Aufruf der Methode *resetTimeoutCounter* auf dem entsprechenden *DeviceStatus*-Objekt. Weiterhin wird mit dem Aufruf der Methode *setControllable* gesetzt ob der Sensorträger steuerbar ist.

Um das Senden dieser Informationen zu ermöglichen, wurde, wie im Abschnitt Nachrichten beschrieben, eine Zielsystemverfügbarkeitsnachricht definiert. Zum periodischen Senden dieser Nachricht wird bei Initialisierung von AMFIS-Modulen in der Agentenanwendung ein Timer gestartet, der diese Nachricht an den Multiuserchat sendet.

```
...
  if ("getMessage".equals(cmd)) {
...
  if(deviceStatusList.size() != 0) {
          for(int i = 0; i < deviceStatusList.size(); ++i) {

            if(deviceStatusList.get(i).getSensornodeId() == msg.getOriginator().
                getSensornodeid()) {
              //reset timeout counter
              deviceStatusList.get(i).resetTimeoutCounter();
              deviceStatusList.get(i).setOnline(true);

                if(msg.getType() == MessageTypeAttributeType.POSITION) {
                  position = msg.getValue().split(";");

                  for(int j = 0; j < position.length; ++j) {
                    if(!position[j].equals("0")) {
                    deviceStatusList.get(i).setPosition(true);
                    break;
...                 deviceStatusList.get(i).setPositionValues(position);
...
        for(int i = 0; i < deviceStatusList.size(); ++i) {
          if((deviceStatusList.get(i).isOnline() && !deviceStatusList.get(i).
              isMobile)) {
          deviceStatusList.get(i).setControllable(true);
          } else if(msg.getType() == MessageTypeAttributeType.POSITION && (
              deviceStatusList.get(i).isOnline() && deviceStatusList.get(i).
              isPosition())) {
            deviceStatusList.get(i).setControllable(true);
               } else if(msg.getType() == MessageTypeAttributeType.POSITION && (
                   deviceStatusList.get(i).isOnline() && !deviceStatusList.get(i
                   ).isPosition())) {
                deviceStatusList.get(i).setControllable(false);
...
```

Listing 5.28: Prüfung der Zielsystemverfügbarkeit für AMFIS-Zielsysteme

Mit dem Aufruf der Methode *decrementTimeoutCounter* dekrementiert ein weiterer, zuvor gestarteter, Timer sekündlich das Attribut *timeoutCounter* aller *DeviceStatus*-Objekte (Listing 5.29). Erreicht das Attribut den Wert null, wird das Attribut *isControllable* des Sensorträgers auf „*false*“ gesetzt. Somit ist das Zielsystem nicht mehr steuerbar und wird als nicht verfügbar eingestuft.

```
public class DeviceStatusTask extends TimerTask {

  ArrayList<DeviceStatus> devicesList;

  public void run() {

    devicesList = AmfisConnection.getInstance().getDeviceStatusList();

    for(int i = 0; i < devicesList.size(); ++i) {
      if(devicesList.get(i).isOnline) {
        devicesList.get(i).decrementTimeoutCounter();
...
```

Listing 5.29: Auszug der Klasse *DeviceStatusTask*

Vor dem Absenden der Zielsystemverfügbarkeitsnachricht wird in der Klasse *AmfisDeviceStatusTask* (Listing 5.30) über die Java-*ArrayList*, welche die *DeviceStatus*-Objekte enthält, iteriert, um die Zielsystemverfügbarkeitsnachricht mit den enthaltenen verfügbaren Sensorträgern zu erstellen und an den Multiuserchat zu senden.

```
public class AmfisDeviceStatusTask extends TimerTask {

  XMPPServerConnection connection;

  ArrayList<DeviceStatus> devicesList;

  String statusXML = "";

  public AmfisDeviceStatusTask(XMPPServerConnection connection) {
    this.connection = connection;
    this.connection.login();
  }

  public void run() {

    devicesList = AmfisConnection.getInstance().getDeviceStatusList();

    statusXML += "\n<AMFISDeviceStatus>";

    for (int i = 0; i < devicesList.size(); ++i) {
      if (devicesList.get(i).isControllable()) {
        statusXML += "\n<node>\n<nodeId>amfis-" + devicesList.get(i).
            sensornodeId + "</nodeId>";
        statusXML += "\n<name>" + devicesList.get(i).deviceName + "</name>";
        statusXML += "\n</node>";
      }
    }

    statusXML += "\n</AMFISDeviceStatus>";

    this.connection.sendMessage(statusXML);

    statusXML = "";

  }

}
```

Listing 5.30: Klasse *AmfisDeviceStatusTask*

6 Evaluation

In diesem Kapitel findet eine Evaluation der Implementierung statt. Hierfür werden zunächst das Ziel und der Aufbau der Evaluation beschrieben. Im Anschluss werden der Ablauf und die Kriterien festgelegt, um dann das Ergebnis aufzuzeigen.

6.1 Ziel und Aufbau

Das Ziel dieser Evaluation ist ein Funktionsnachweis anhand des Anwendungsszenarios - Einstellung von Szenariokonfigurationsparametern in die Zielsysteme - für den Szenarioassistenten und die Agentenanwendung. Die Grundlage für diesen Funktionsnachweis bildet die Szenariokonfigurationsdatei des konzipierten und implementierten Demonstrationsszenarios.

Um die Ausführung des Demonstrationsszenarios zu ermöglichen sind einige vorbereitende Maßnahmen nötig. Der Chat-Client Psi wird eingesetzt, um jederzeit nachvollziehen zu können welche Entitäten am Multiuserchat angemeldet sind und welche XMPP-Nachrichten an den Multiuserchat gesendet werden. Somit kann der gesamte Nachrichtenverkehr zwischen den Agentenanwendungen und dem Szenarioassistenten im Chat-Fenster der Benutzungsoberfläche des Chat-Clients protokolliert und überwacht werden. Weiterhin muss die Erreichbarkeit der am Demonstrationsszenario beteiligten Zielsysteme gewährleistet werden, sodass die Parameter der Szenariokonfigurationsdatei, die der Szenarioassistent übertragen soll, auch von den Zielsystemen empfangen werden können.

Für die AMFIS-Bodenkontrollstation bedeutet dies, dass die Anwendung AMFIS-Konnektor gestartet und die Zielsysteme FORBOT, Axis-Dome 1 und Axis-Dome 2 am AMFIS-Konnektor angemeldet werden müssen. Der AMFIS-Piloten-Arbeitsplatz wird benötigt um die Kamerabilder der beiden Domkameras am Bildschirm anzeigen und verfolgen zu können. Ebenso wird die Anwendung für die Lagedarstellung gestartet, sodass die eingestellte Wegpunktliste und die aktuelle Position des FORBOTs angezeigt wird.

Aufgrund der Tatsache, dass der Digitale Lagetisch nur simuliert werden kann, soll auf dem im Rahmen dieser Arbeit verwendeten Entwicklungsrechner eine lokale Instanz eines ActiveMQ Message Brokers gestartet werden. In diesen ActiveMQ Message Broker wird die Nachricht für den Aufruf eines Szenarios im Digitalen Lagetisch eingestellt. Weiterhin wird eine Agentenanwendung für den Digitalen Lagetisch konfiguriert. Dazu wird ein <module>-Element für den Digitalen Lagetisch in die externe XML-Konfigurationsdatei eingetragen und parametrisiert. Anschließend muss die Agentenanwendung auf dem Entwicklungsrechner ausgeführt werden, sodass der Empfang einer Szenariokonfiguration möglich ist.

Auf der Zielplattform AMFIS-Bodenkontrollstation soll ebenfalls eine Instanz der Agentenanwendung ausgeführt werden. Es wird, wie bei der Simulation für den Digitalen Lagetisch, eine externe Konfigurationsdatei erstellt. Die beteiligten Zielsysteme der Zielplattform AMFIS-

Bodenkontrollstation sind der FORBOT und die beiden Domkameras Axis-Dome 1 und Axis-Dome 2. Diese werden in Form von <module>-Elementen in die Konfiguration eingetragen.

6.2 Ablauf und Kriterien

In diesem Abschnitt wird zunächst der Ablauf der Evaluierung beschrieben. Nachdem alle Zielsysteme aktiviert und einsatzbereit sind wird der Szenarioassistent gestartet. Danach werden die Agentenanwendungen ausgeführt, sodass die benötigten Module erzeugt werden und deren Anmeldung am Multiuserchat des XMPP-Servers stattfindet (Abbildung 6.1). Im Anschluss daran folgt die Auswahl und der Start des Demonstrationsszenarios im Szenarioassistenten.

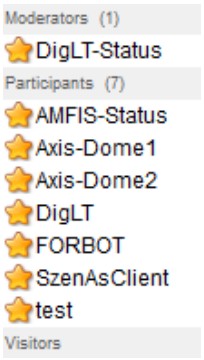


Abbildung 6.1: Teilnehmer im Multiuserchat

Für die unterschiedlichen Zielsysteme wurden im Vorfeld der Szenarioausführung hinreichende Kriterien für die erfolgreiche Einstellung von Konfigurationsparametern festgelegt. So sollen die Domkameras, weil sie direkt steuerbar sind, mit Einstellung der Parameter ihre Aufgaben sofort ausführen und sich auf die Bereiche, die mit den geografischen Koordinaten in der Szenariokonfigurationsdatei angegeben wurden, ausrichten. Für den FORBOT soll die eingestellte Wegpunktliste im Piloten-Arbeitsplatz angezeigt werden. Bei der Simulation des Digitalen Lagetisches wird ein Topic mit der empfangenen JMS-Objektnachricht im ActiveMQ Message Broker angelegt.

6.3 Ergebnis

Mit dem Starten der Szenarioausführung im Szenarioassistenten wird zunächst geprüft, ob die entsprechende Konfigurationsdatei an den Multiuserchat gesendet wird. Da die Chat-Sitzung mit Psi beobachtet werden kann, konnte die erfolgreiche Übertragung im Chat-Fenster von Psi nachvollzogen werden. Die Abbildung 6.2 zeigt die gesendete Szenariokonfigurationsdatei in der Benutzungsoberfläche des Chat-Clients.

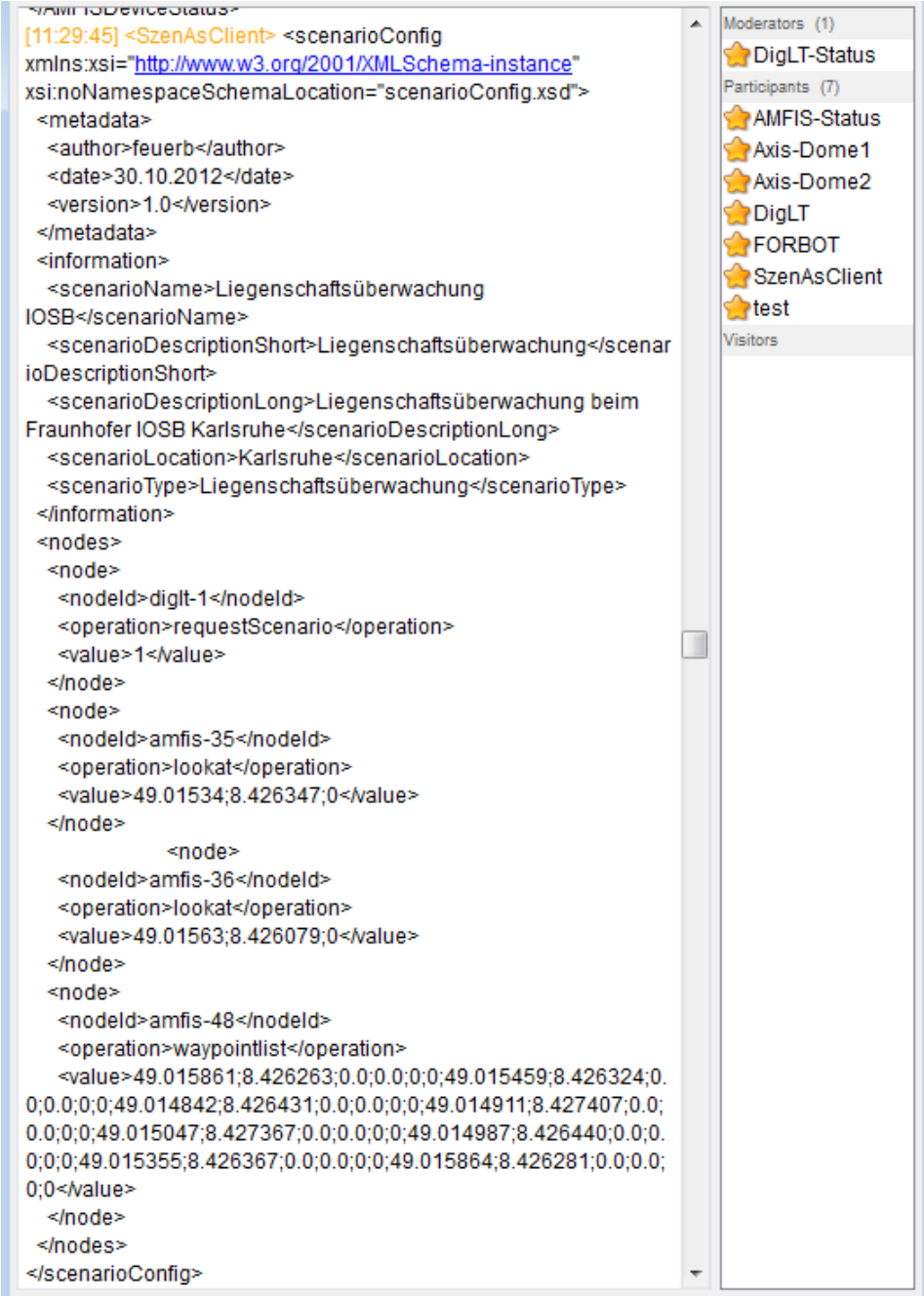


Abbildung 6.2: Gesendete Szenariokonfigurationsdatei

Die Ausrichtung der Domkameras wird im Piloten-Arbeitsplatz überprüft. Um die korrekte Einstellung der Parameter für die Domkameras nach der Szenarioausführung kontrollieren zu können, wurde für jede Domkamera ein Screenshot der Ausrichtung vor der Szenarioausführung gemacht. Nach der Ausführung des Demonstrationsszenarios erfolgt ein Vergleich der Kamerabilder. Die Abbildung 6.3 zeigt die Ausrichtung der Domkamera Axis-Dome 1 vor der Ausführung. Vergleichend dazu zeigt die Abbildung 6.4 das Bild der Kamera, nachdem die Einstellung der Parameter für dieses Zielsystem erfolgt ist. Der Nachweis für die Domkamera Axis-Dome 2 wird in derselben Weise durchgeführt. Abbildung 6.5 zeigt die ursprüngliche Ausrichtung der Domkamera. Abbildung 6.6 zeigt die Ausrichtung, nachdem die Konfigurationsparameter für dieses Zielsystem ebenfalls eingestellt sind.

Abbildung 6.3: Axis-Dome 1 vorher

Abbildung 6.4: Axis-Dome 1 nachher

Abbildung 6.5: Axis-Dome 2 vorher

Abbildung 6.6: Axis-Dome 2 nachher

Wie im Abschnitt Kriterien erwähnt soll die eingestellte Wegpunktliste für den FORBOT am Piloten-Arbeitsplatz angezeigt werden. Auch für den FORBOT wird im Vorfeld ein Screenshot des Kamerabildes im Piloten-Arbeitsplatz erstellt (Abbildung 6.7). In dieser Abbildung ist im rechten Bereich das noch leere Fenster für die einzustellende Wegpunktliste zu sehen. Nach dem Starten der Szenarioausführung im Szenarioassistenten wird die eingestellte Wegpunktliste im Fenster für die Anzeige von Wegpunktlisten im Piloten-Arbeitsplatz angezeigt (Abbildung 6.8).

Abbildung 6.7: FORBOT vor Parametereinstellung

Abbildung 6.8: FORBOT nach Parametereinstellung

Queues und Topics, die in einem ActiveMQ Message Broker angelegt sind, können über eine Webkonsole eingesehen werden. Diese Möglichkeit wird genutzt um nachprüfen zu können ob die JMS-Objektnachricht, nach der Ausführung des Demonstrationsszenarios im Szenarioassistenten, an den ActiveMQ Message Broker zugestellt ist. Vor Ausführung des Demonstrationsszenarios wurden zunächst alle Topics des gestarteten ActiveMQ Message Brokers manuell in der Webkonsole gelöscht (Abbildung 6.9).

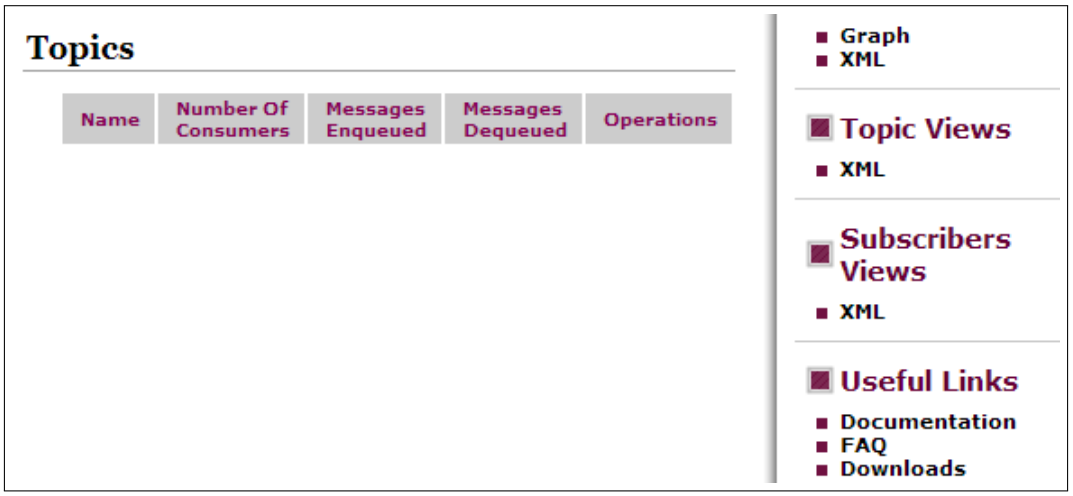
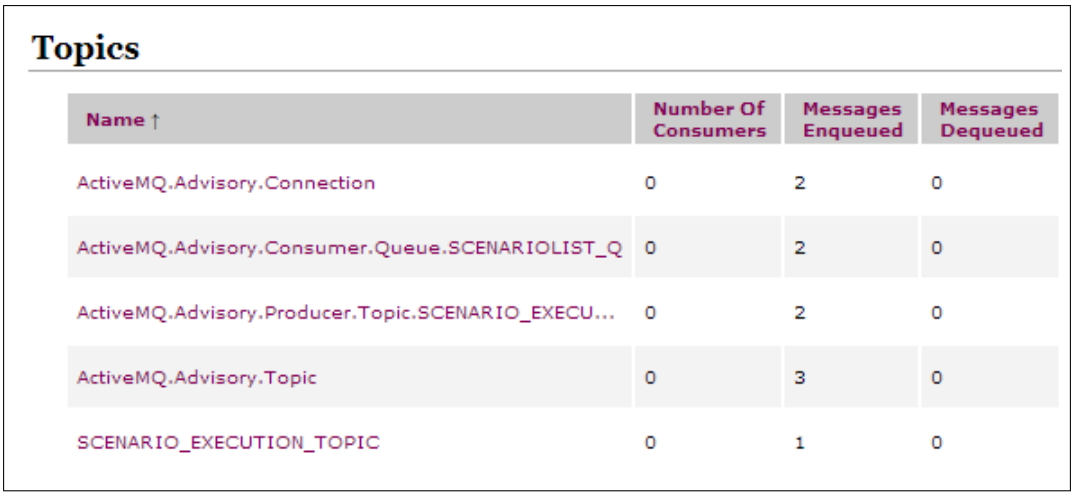


Abbildung 6.9: Webkonsole vor der Szenarioausführung

Mit dem Ausführen des Demonstrationsszenarios und der anschließenden Zustellung der JMS-Objektnachricht ist das neue Topic „SCENARIO_EXECUTION_TOPIC“ im ActiveMQ Message Broker angelegt. In der Spalte „Messages Enqueued“ der Webkonsole ist zu erkennen, dass die Nachricht für den Digitalen Lagetisch in das Topic eingestellt ist (Abbildung 6.10). Somit erfolgte nach der Ausführung des Demonstrationsszenarios auch für das Zielsystem Digitaler Lagetisch die Parametereinstellung.



Topics

Name ↑	Number Of Consumers	Messages Enqueued	Messages Dequeued
ActiveMQ.Advisory.Connection	0	2	0
ActiveMQ.Advisory.Consumer.Queue.SCENARIOLIST_Q	0	2	0
ActiveMQ.Advisory.Producer.Topic.SCENARIO_EXECU...	0	2	0
ActiveMQ.Advisory.Topic	0	3	0
SCENARIO_EXECUTION_TOPIC	0	1	0

Abbildung 6.10: Webkonsole nach der Szenarioausführung

7 Zusammenfassung und Ausblick

Im letzten Kapitel dieser Arbeit werden die erzielten Ergebnisse zusammengefasst. Abschließend folgt ein Ausblick, der als Grundlage für weitere Arbeiten oder mögliche Erweiterungen dienen soll.

7.1 Zusammenfassung

Im Rahmen dieser Arbeit wurde das System ExBa um einen mobilen Szenarioassistenten erweitert. Eine Assistenzfunktion wurde dahingehend umgesetzt, dass zur Durchführung von Demonstrationsszenarien für das ExBa kein individuelles Wissen für die Konfiguration der beteiligten Systemkomponenten benötigt wird und außerdem die aufwendige Konfiguration, die normalerweise im Vorfeld der Durchführung erfolgen muss, entfällt. Ein Demonstrationsszenario kann ganz einfach im Szenarioassistenten gestartet werden. Weiterhin bietet der Szenarioassistent Assistenzinformationen zur Verfügbarkeit der Systeme. In der Oberfläche wird angezeigt ob ein Zielsystem für die Durchführung eines Demonstrationsszenarios verfügbar ist und somit eine Einstellung von Konfigurationsparametern zur Steuerung des Zielsystems überhaupt möglich ist.

In weiteren Teilaufgaben dieser Arbeit wurde eine Agentenanwendung in der Programmiersprache Java, ein Demonstrationsszenario und die auf Nachrichten basierende Kommunikation zwischen den einzelnen Anwendungen implementiert. Neben der Kommunikation zwischen dem Szenarioassistenten und der Agentenanwendung über das XMP-Protokoll wurde der Austausch von Nachrichten zwischen der Agentenanwendung und den ExBa-Systemkomponenten realisiert. Dazu wurde zunächst das benötigte Wissen zum weiteren Verständnis dieser Arbeit vermittelt. Es wurde eine Anforderungsanalyse durchgeführt, in der die Anwender der Systeme und die Systemarchitektur aufgezeigt wurden. Weiterhin wurden Anwendungsfälle und Anforderungen für die einzelnen Anwendungen identifiziert. Darauf aufbauend erfolgte dann die Konzeption, welche die Grundlage für die weitere Implementierung bildete. Abschließend wurde eine Evaluation durchgeführt.

Zur Umsetzung des Demonstrationsszenarios mussten zunächst einige Eigenschaften festgelegt werden. Es wurden die beteiligten Systeme und deren Aufgaben im Demonstrationsszenario definiert. Da die Konfigurationsdatei zum entworfenen Demonstrationsszenario XML-basiert sein sollte wurde mit der Schemasprache XML Schema ein Datenmodell entworfen.

Der mobile Szenarioassistent wurde, basierend auf den Web-Technologien HTML, CSS und JavaScript, für das mobile Betriebssystem Android umgesetzt. Dazu wurde das Android SDK und das Framework PhoneGap verwendet. Mit Phonegap wurde die implementierte Offline-Web-App in eine native Android-App gehüllt. Die einfach und intuitiv zu bedienende Benutzungsoberfläche wurde mit dem auf HTML5-basierenden Framework jQuery Mobile um-

gesetzt. Die Logik der Anwendung wurde komplett in JavaScript erstellt. Für die Kommunikation der App mit der Agentenanwendung wurde die JavaScript-XMPP-Bibliothek Strophe.js mit diversen Plug-ins eingesetzt. Zur Kommunikation mit einem Multiuserchat musste die XMPP-Protokollerweiterung „XEP-0045: Multi-User Chat“ umgesetzt werden. Dazu wurde das Strophe-Plug-in *strophe.muc.js* genutzt. Die XMPP-Protokollerweiterung zur Registrierung eines XMPP-Benutzerkontos „XEP-0077: In-Band Registration“ konnte durch Einbinden des Strophe-Plug-ins *strophe.register.js* implementiert werden. Da mit JavaScript keine langlebigen TCP-Verbindungen möglich sind musste die XMPP-Protokollerweiterung „XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)“ genutzt werden, wofür kein zusätzliches Strophe-Plug-in benötigt wurde.

Die implementierte Agentenanwendung kann auf den Systemkomponenten bzw. Zielplattformen des ExBa ausgeführt werden. Die Zielplattformen und die an einem Demonstrationsszenario beteiligten Zielsysteme können in der Agentenanwendung über eine externe XML-basierte Konfigurationsdatei aktiviert und parametrisiert werden. Diese Konfigurationsdatei und die vom Szenarioassistenten in XMPP-Nachrichten gesendeten Szenariokonfigurationen werden in der Agentenanwendung mit JAXB deserialisiert. Um XMPP-Nachrichten empfangen und senden zu können wurde eine XMPP-Komponente mit der Bibliothek Smack realisiert. Die Zielplattform Digitaler Lagetisch konnte im Rahmen dieser Arbeit nur simuliert werden. Dazu wurde der JMS-Provider ActiveMQ eingesetzt. Um eine Nachricht zur Einstellung von Szenariokonfigurationsparametern in den Digitalen Lagetisch an diesen Message Broker zu senden, wurden ein JMS-Client und eine JMS-Objektnachricht implementiert. Für die Kommunikation mit der Schnittstelle der AMFIS-Bodenkontrollstation wurde eine Komponente unter Verwendung der Bibliothek JavaAmfisCom implementiert. Diese ermöglicht das Erzeugen und Senden von AMFIS-Nachrichten, um Szenariokonfigurationsparameter in die Zielsysteme der AMFIS-Bodenkontrollstation einzustellen. Weiterhin wurde mit der Implementierung der Komponente auch der Empfang von AMFIS-Nachrichten zur Ermittlung der Zielsystemverfügbarkeit ermöglicht. Zum Senden der Zielsystemverfügbarkeit an den Szenarioassistenten wurde ein eigener XML-basierter Nachrichtentyp erstellt.

7.2 Ausblick

Das in dieser Arbeit entwickelte System bietet eine Grundlage für dessen Weiterentwicklung oder zukünftige Arbeiten. So bietet sich zunächst die Anbindung der weiteren ExBa-Systemkomponenten ABUL und CSD an. Es könnten beispielsweise Daten in Form von Kamerabildern der Domkameras oder des FORBOTs an den CSD gesendet und darin abgespeichert werden.

Die Assistenzfunktion des Szenarioassistenten kann dahingehend erweitert werden, dass weitere Assistenzinformationen in der App angezeigt werden. Dazu leitet die Agentenanwendung vom AMFIS-Konnektor empfangene Positionsnachrichten des FORBOTs und der Domkameras an den Szenarioassistenten weiter. Es bietet sich an, diese Informationen im Szenarioassistenten in einer Kartenansicht zu visualisieren. So erhält der Benutzer jederzeit einen Überblick über die Positionen der Sensorträger im Einsatzgebiet. Außerdem kann der GPS-Sensor des mobilen Geräts genutzt werden, um zusätzlich zu den Positionen der verschiedenen Sensorträger die eigene Position in der Kartenansicht anzuzeigen.

Die XMPP-Protokollerweiterung BOSH kann im Szenarioassistenten zukünftig durch Verwendung des WebSocket-Protokolls ersetzt werden. Diese Möglichkeit war zum Zeitpunkt dieser Arbeit noch nicht in der JavaScript-XMPP-Bibliothek Strophe.js implementiert. Durch diesen Ansatz wird eine bessere Performanz bei der Netzwerkkommunikation erreicht. Vom Autor der Bibliothek existiert bereits ein Internet-Draft bei der IETF. Es ist allerdings noch nicht genau absehbar wann der Einsatz des WebSocket-Protokolls in Verbindung mit XMPP möglich ist [Mof12].

Durch das eingesetzte Cross-Plattform-Framework PhoneGap ist grundsätzlich eine Portierung des Szenarioassistenten auf weitere mobile Betriebssystemplattformen wie iOs, Blackberry, Windows Phone, Palm, WebOS, Bada und Symbian möglich. Da PhoneGap aber nur in der Theorie eine „Write once, run everywhere“-Lösung bietet, müssen dazu noch weitere Anpassungen erfolgen [Spi11].

Literaturverzeichnis

[Ada02] Adams, D.: Programming Jabber: Extending XML Messaging. O'Reilly Media, 2002

[And12a] Android Developers: Platform Versions. http://developer.android.com/about/dashboards/index.html, 2012

[And12b] Android Developers: WebView. http://developer.android.com/reference/android/webkit/WebView.html, 2012

[Aud12] Audi: Audi Konfigurator Deutschland. http://www.audi.de/de/brand/de/erlebniswelt/audi_multimedial/audi_apps/modelle/audi_konfigurator_deutschland.html, 2012

[Bür11] Bürkle, A. u.a.: Autonomous geo-referenced aerial reconnaissance for instantaneous applications. International Journal on Advances in Systems and Measurements, Vol. 4, No. 3 & 4, 2011

[Fir12] Firtman, M.: jQuery Mobile: Up and Running. O'Reilly Media, 2012

[Fra12] Fraunhofer: Fraunhofer IOSB. http://www.iosb.fraunhofer.de, 2012

[Gar11] Gargenta, M.: Learning Android. O'Reilly Media, 2011

[Hei12] Heise: Marktforscher: Über 100 Millionen Androiden ausgeliefert. http://www.heise.de/newsticker/meldung/Marktforscher-Ueber-100-Millionen-Androiden-ausgeliefert-1659638.html, 2012

[Mof12] Moffit, J. u.a.: An XMPP Sub-protocol for WebSocket. http://tools.ietf.org/html/draft-moffitt-xmpp-over-websocket-01, 2012

[jQu12] jQuery Mobile: Mobile Graded Browser Support. http://jquerymobile.com/gbs/, 2012

[Lae12] Laerdal: VitalSim. http://www.laerdal.com/de/doc/247/VitalSim, 2012

[Mof10] Moffit, J.: Professional XMPP Programming with JavaScript and jQuery (Wrox Programmer to Programmer). Wrox, 2010

[Man10] Manashty, A.R. u.a.: A Scenario-Based Mobile Application for Robot-Assisted Smart Digital Homes. International Journal of Computer Science and Information Security (IJCSIS), Vol. 8, No. 5, 2010

[Pho12] PhoneGap: PhoneGap, Cordova, and what's in a name? `http://phonegap.com/`, 2012

[Ric09] Richards, M. u.a.: Java Message Service. 2nd Edition, O'Reilly Media, 2009

[Rob12] Roboterwerk: FORBOT: Outdoor Roboter-Plattform für Forschung u. Wissenschaft. `http://www.roboterwerk.de/`, 2012

[Rol11] Rollins, M.: The Business of Android Apps Development: Making and Marketing Apps that Succeed. Apress, 2011

[SST09] Saint-Andre, P. u.a.: XMPP: The Definitive Guide. O'Reilly Media, 2009

[Sny11] Snyder, B. u.a.: ActiveMQ in Action. Manning Publications, 2011

[Spi11] Spiering, M.: HTML5-Apps für iPhone und Android. 2. aktualisierte Auflage, Franzis, 2011

[War12] Wargo, J.M.: PhoneGap Essentials: Building Cross-Platform Mobile Apps. Addison-Wesley Professional, 2012

Anhang

A.1 Anwendungsfälle

A.1.1 Anwendungsfälle des Szenarioassistenten

AF-01:	Szenariokonfigurationsparameter auf die Systemkomponenten der Zielplattformen übertragen
Ziel:	Die möglichst einfache und schnelle Einstellung von Szenariokonfigurationsparametern in die ExBa-Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	ExBa-Entwickler, ExBa-Operator
Vorbedingung:	Die Zielplattformen und Zielsysteme sind einsatzbereit
Erfolgsergebnis:	Die Szenariokonfigurationsparameter sind eingestellt
Ablauf:	1.) ExBa-Entwickler erstellt eine Szenariokonfigurationsdatei (AF-02). 2.) ExBa-Entwickler speichert die Szenariokonfiguration im Szenarioassistenten (AF- 03). 3.) ExBa-Nutzer startet den Szenarioassistenten. 4.) ExBa-Nutzer bekommt eine Übersicht der verfügbaren Szenarien angezeigt (AF-04). 5.) ExBa-Nutzer wählt das auszuführende Szenario aus der Szenarioübersichtsliste aus (AF-05). 5.) ExBa-Nutzer bekommt eine Übersicht der verfügbaren Szenarien angezeigt (AF-04). 6.) ExBa-Nutzer bekommt Detailinformationen zum ausgewählten Szenario angezeigt (AF-06). 6.1.) ExBa-Nutzer lässt sich die Verfügbarkeit der Zielsysteme anzeigen (AF-07). 7.) ExBa-Nutzer startet das ausgewählte Szenario (AF-08).

Tabelle A.1: Anwendungsfallbeschreibung AF-01

AF-02:	Szenariokonfigurationsdatei erstellen
Ziel:	Erstellung einer Szenariokonfigurationsdatei, welche die notwendigen Parameter für eine Szenarioausführung enthält
Primäre Akteure:	ExBa-Entwickler
Sekundäre Akteure:	keine
Vorbedingung:	Durchführung eines Demonstrationsszenarios
Erfolgsergebnis:	Konfigurationsdatei, die im Szenarioassistenten gespeichert werden kann
Ablauf:	1.) ExBa-Entwickler erstellt eine Konfigurationsdatei. 2.) ExBa-Entwickler trägt Parameter in die Konfigurationsdatei ein.
Anforderung/en:	ANF-01

Tabelle A.2: Anwendungsfallbeschreibung AF-02

AF-03:	Szenariokonfigurationsdatei speichern
Ziel:	Speichern einer Szenariokonfigurationsdatei im Szenarioassistenten
Primäre Akteure:	ExBa-Entwickler
Sekundäre Akteure:	keine
Vorbedingung:	Die Szenariokonfigurationsdatei wurde erstellt
Erfolgsergebnis:	Die Szenariokonfigurationsdatei ist im Szenarioassistenten gespeichert
Ablauf:	1.) ExBa-Entwickler speichert die Konfiguration im Szenarioassistenten.
Anforderung/en:	ANF-12

Tabelle A.3: Anwendungsfallbeschreibung AF-03

AF-04:	Demonstrationsszenarien anzeigen
Ziel:	Der ExBa-Nutzer bekommt die verfügbaren Demonstrationsszenarien angezeigt
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	keine
Vorbedingung:	Demonstrationsszenarien sind im Szenarioassistenten gespeichert
Erfolgsergebnis:	Demonstrationsszenarien werden angezeigt
Ablauf:	1.) Demonstrationsszenarien werden in einer Übersichtsliste angezeigt.
Anforderung/en:	ANF-08

Tabelle A.4: Anwendungsfallbeschreibung AF-04

AF-05:	Demonstrationsszenario auswählen
Ziel:	Die Auswahl eines Demonstrationsszenarios
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	keine
Vorbedingung:	Demonstrationsszenarien werden in der Übersichtsliste angezeigt
Erfolgsergebnis:	Weiterleitung auf die Detailseite des Demonstrationsszenarios
Ablauf:	1.) Der ExBa-Nutzer wählt das auszuführende Demonstrationsszenario in der Übersichtsliste aus.
Anforderung/en:	ANF-09

Tabelle A.5: Anwendungsfallbeschreibung AF-05

AF-06:	Szenariodetailinformationen anzeigen
Ziel:	Die Anzeige von Detailinformationen zu einem gewählten Demonstrationsszenario
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	keine
Vorbedingung:	Demonstrationsszenario wurde ausgewählt
Erfolgsergebnis:	Die Detailseite des Demonstrationsszenarios wird angezeigt
Ablauf:	1.) Der Anwender wählt das auszuführende Demonstrationsszenario in der Übersichtsliste aus.
Anforderung/en:	ANF-10

Tabelle A.6: Anwendungsfallbeschreibung AF-06

AF-07:	Zielsystemverfügbarkeit anzeigen
Ziel:	Die Anzeige der verfügbaren Zielsysteme auf den Zielplattformen
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	Agentenanwendungen auf den Zielplattformen
Vorbedingung:	keine
Erfolgsergebnis:	Die Verfügbarkeit der Zielsysteme wird angezeigt
Ablauf:	1.) ExBa-Nutzer betätigt auf der Szenariodetailseite den Button „Zielsysteme“.
Anforderung/en:	ANF-11

Tabelle A.7: Anwendungsfallbeschreibung AF-07

AF-08:	Szenariokonfigurationsdatei übertragen
Ziel:	Die Szenariokonfigurationsdatei des ausgewählten Demonstrationsszenarios wird auf die Zielplattformen übertragen
Primäre Akteure:	ExBa-Nutzer
Sekundäre Akteure:	keine
Vorbedingung:	Ein Demonstrationsszenario ist ausgewählt
Erfolgsergebnis:	Die Szenariokonfigurationsdatei ist in die Agentenanwendungen auf den Zielplattformen übertragen
Ablauf:	1.) Die Szenariokonfigurationsdatei wird eingelesen. 2.) Die Szenariokonfigurationsdatei wird an die Agentenanwendungen auf den Zielplattformen gesendet.
Anforderung/en:	ANF-03, ANF-06, ANF-07

Tabelle A.8: Anwendungsfallbeschreibung AF-08

A.1.2 Anwendungsfälle der Agentenanwendung

AF-09:	Szenariokonfigurationsparameter in die Systemkomponenten der Zielplattformen einstellen
Ziel:	Einstellen von Szenariokonfigurationsparametern in die ExBa-Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch
Primäre Akteure:	ExBa-Operator, Agentenanwendung
Sekundäre Akteure:	keine
Vorbedingung:	Die Zielplattformen und Zielsysteme sind einsatzbereit
Erfolgsergebnis:	Die Szenariokonfigurationsparameter sind eingestellt
Ablauf:	1.) ExBa-Operator konfiguriert Agentanwendung für die Zielplattform (AF-10). 2.) ExBa-Operator führt Agentenanwendung auf der Zielplattform aus (AF-11). 3.) Agentenanwendung baut eine Verbindung zum XMPP-Server auf (AF-12). 4.) Agentanwendung verbindet sich mit der Zielplattform (AF-13). 4.1.) Agentenanwendung sendet Verfügbarkeit der Zielsysteme (AF-14). 5.) Agentenanwendung nimmt Szenariokonfigurationsdatei entgegen (AF-15). 6.) Agentenanwendung wertet Szenariokonfigurationsdatei aus (AF-16). 6.1.) Agentenanwendung stellt Parameter aus der Szenariokonfigurationsdatei in die Zielsysteme ein. (AF-17).

Tabelle A.9: Anwendungsfallbeschreibung AF-09

AF-10:	Agentenanwendung konfigurieren
Ziel:	Die Konfiguration der Agentenanwendung für eine Zielplattform
Primäre Akteure:	ExBa-Operator
Sekundäre Akteure:	keine
Vorbedingung:	keine
Erfolgsergebnis:	Am Demonstrationsszenario beteiligte Zielsysteme sind auf der Zielplattform aktiviert
Ablauf:	1.) Der ExBa-Operator trägt die Parameter für die Module, die auf der Zielplattform aktiviert werden sollen in die Konfigurationsdatei der Agentenanwendung ein.
Anforderung/en:	ANF-13, ANF-14

Tabelle A.10: Anwendungsfallbeschreibung AF-10

AF-11:	Agentenanwendung ausführen
Ziel:	Die Ausführung der Agentenanwendung auf einer Zielplattform
Primäre Akteure:	ExBa-Operator
Sekundäre Akteure:	keine
Vorbedingung:	Die Agentenanwendung ist konfiguriert
Erfolgsergebnis:	Die Agentenanwendung ist auf der Zielplattform ausgeführt
Ablauf:	1.) Die Agentenanwendung wird auf der Zielplattform gestartet.
Anforderung/en:	ANF-13, ANF-17, ANF-22

Tabelle A.11: Anwendungsfallbeschreibung AF-11

AF-12:	XMPP-Serververbindung aufbauen
Ziel:	Die Agentenanwendung stellt eine Verbindung zum XMPP-Server her
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	XMPP-Server
Vorbedingung:	Die Agentenanwendung ist auf der Zielplattform ausgeführt
Erfolgsergebnis:	Es besteht eine Verbindung zum XMPP-Server
Ablauf:	1.) Für die in der Konfiguration aktiv geschalteten Zielsysteme baut die Agentenanwendung XMPP-Serververbindungen auf. 2.) Registrierung und Authentifizierung am XMPP-Server. 3.) Betreten des Multiuserchats.
Anforderung/en:	ANF-18

Tabelle A.12: Anwendungsfallbeschreibung AF-12

AF-13:	Verbindung zur Zielplattform aufbauen
Ziel:	Agentenanwendung verbindet sich mit der Schnittstelle der Zielplattform, um Szenariokonfigurationsparameter zu senden und Verfügbarkeitsinformationen zu den Zielsystemen abzufragen
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	Zielplattform
Vorbedingung:	Agentenanwendung ist auf der Zielplattform ausgeführt
Erfolgsergebnis:	Es besteht eine Verbindung zur Schnittstelle der Zielplattform
Ablauf:	1.) Agentenanwendung verbindet sich mit der in der Konfiguration aktiv geschalteten Zielplattform.
Anforderung/en:	ANF-19, ANF-20

Tabelle A.13: Anwendungsfallbeschreibung AF-13

AF-14:	Verfügbarkeit der Zielsysteme senden
Ziel:	Agentenanwendung stellt periodisch Informationen zur Verfügbarkeit der Zielsysteme im Multiuserchat bereit
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	Zielplattform
Vorbedingung:	Agentenanwendung ist mit der Zielplattform verbunden
Erfolgsergebnis:	Verfügbarkeitsinformationen sind an den Multiuserchat gesendet
Ablauf:	1.) Agentenanwendung prüft welche Zielsysteme auf der Zielplattform verfügbar sind. 2.) Agentenanwendung sendet die Information an den Multiuserchat.
Anforderung/en:	ANF-18, ANF-19, ANF-20

Tabelle A.14: Anwendungsfallbeschreibung AF-14

AF-15:	Szenariokonfigurationsdatei entgegennehmen
Ziel:	Empfangen einer Szenariokonfigurationsdatei im Multiuserchat
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	Szenarioassistent, XMPP-Server
Vorbedingung:	Szenariokonfigurationsdatei wurde vom Szenarioassistenten an den Multiuserchat gesendet
Erfolgsergebnis:	Die Szenariokonfigurationsdatei ist empfangen
Ablauf:	1.) Agentenanwendung nimmt die Szenariokonfigurationsdatei im Multiuserchat entgegen.
Anforderung/en:	ANF-18

Tabelle A.15: Anwendungsfallbeschreibung AF-15

AF-16:	Szenariokonfigurationsdatei auswerten
Ziel:	Auswertung der XML-basierten Szenariokonfigurationsdatei, um die enthaltenen Parameter an die jeweiligen Zielsysteme zu übertragen
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	keine
Vorbedingung:	Szenariokonfigurationsdatei wurde vom Multiuserchat erhalten
Erfolgsergebnis:	Die Konfigurationsparameter für die Zielsysteme sind ausgewertet und bereit zum Senden
Ablauf:	1.) Agentenanwendung wertet die Szenariokonfigurationsdatei aus. 2.) Agentenanwendung bereitet die enthaltenen Parameter für die Einstellung in die Zielplattform bzw. Zielsysteme vor.
Anforderung/en:	ANF-15, ANF-21

Tabelle A.16: Anwendungsfallbeschreibung AF-16

AF-17:	Zielsystemparameter in Zielplattform einstellen
Ziel:	Übermittlung der Parameter an die Zielplattform bzw. deren Zielsysteme
Primäre Akteure:	Agentenanwendung
Sekundäre Akteure:	Zielplattformen, Zielsysteme
Vorbedingung:	Szenariokonfigurationsdatei wurde ausgewertet
Erfolgsergebnis:	Die Konfigurationsparameter sind in die Zielplattformen bzw. Zielsysteme übertragen
Ablauf:	1.) Agentenanwendung erstellt Nachricht/en mit den Konfigurationsparametern. 2.) Agentenanwendung sendet die Nachricht/en an die Zielplattform.
Anforderung/en:	ANF-19, ANF-20

Tabelle A.17: Anwendungsfallbeschreibung AF-17

A.2 Anforderungen

A.2.1 Anforderungen an den Szenarioassistenten

Bestehende Anforderungen aus Spezifikation

Funktionale Anforderungen

ANF-01:	XML-basiertes Datenmodell zur Erfassung von Szenariokonfigurationen
Beschreibung:	Szenariokonfigurationsdateien sollen im Datenformat XML gespeichert werden. Für die Erfassung einer Szenariokonfigurationsdatei muss ein XML-Datenmodell erstellt werden.

Tabelle A.18: Anforderung ANF-01

ANF-02:	Inhalte zu den Szenarien sollen in HTML5 gespeichert werden, sodass die spätere Editierbarkeit vereinfacht ist
Beschreibung:	Die Szenarieninhalte sollen im Szenarioassistenten leicht zu überarbeiten und speicherbar sein. Dazu sollen moderne Android-fähige HTML5-Frameworks wie beispielsweise Sencha Touch 2 oder jQuery Mobile genutzt werden.

Tabelle A.19: Anforderung ANF-02

ANF-03:	Szenariodaten werden in XML über XMPP gesendet
Beschreibung:	Die Szenariokonfigurationen werden in XML-Dateien an den XMPP-Server gesendet.

Tabelle A.20: Anforderung ANF-03

Nichtfunktionale Anforderungen

ANF-04:	Android Plattform, mind. API-Level 10 (Version 2.3.3), aber auch auf 3.0, 4.0 und 4.1
Beschreibung:	Dies sind die Android-Versionen, auf denen der Szenarioassistent lauffähig sein soll.

Tabelle A.21: Anforderung ANF-04

ANF-05:	Lauffähig auf den Android Geräten des Fraunhofer IOSB (Google Nexus S, HTC Desire, Samsung Galaxy Tab)
Beschreibung:	Dies sind die Android-Geräte auf denen der Szenarioassistent lauffähig sein soll.

Tabelle A.22: Anforderung ANF-05

ANF-06:	Kommunikation über WLAN mit dem zentralen XMPP-Server zum Austausch der Szenariodaten
Beschreibung:	Der Szenarioassistent muss über WLAN auf den zentralen XMPP-Server zugreifen.

Tabelle A.23: Anforderung ANF-06

Zusätzlich aufgestellte Anforderungen

Funktionale Anforderungen

ANF-07:	XMPP-Komponente
Beschreibung:	Der Szenarioassistent muss für die Kommunikation mit den Agentenanwendungen auf den Zielplattformen eine XMPP-Komponente mit folgenden Funktionen enthalten: - Verbindungsaufbau - Registrierung und Authentifizierung - Erstellen eines Multiuserchats - Betreten eines Multiuserchats - Senden und Empfangen von XMPP-Nachrichten

Tabelle A.24: Anforderung ANF-07

ANF-08:	Anzeige der verfügbaren Demonstrationsszenarien in einer Übersichtsliste
Beschreibung:	Der ExBa-Nutzer soll eine Übersichtsliste der auswählbaren Demonstrationsszenarien angezeigt bekommen. Die Liste dient der Auswahl eines Demonstrationsszenarios.

Tabelle A.25: Anforderung ANF-08

ANF-09:	Auswahl eines Demonstrationsszenarios aus der Übersichtsliste
Beschreibung:	Der ExBa-Nutzer muss ein Demonstrationsszenario aus der Übersichtsliste auswählen können.

Tabelle A.26: Anforderung ANF-09

ANF-10:	Anzeige einer Szenariodetailseite zu einem gewählten Demonstrationsszenario
Beschreibung:	Der ExBa-Nutzer soll Szenariodetailinformationen zu einem ausgewählten Demonstrationsszenario angezeigt bekommen.

Tabelle A.27: Anforderung ANF-10

ANF-11:	Verfügbarkeit der Zielplattformen bzw. Zielsysteme in einer Übersichtsseite anzeigen
Beschreibung:	Der ExBa-Nutzer soll eine Übersicht der aktuell verfügbaren Zielsysteme erhalten.

Tabelle A.28: Anforderung ANF-11

ANF-12:	Speichern von Szenariokonfigurationsdateien im Szenarioassistenten
Beschreibung:	Der ExBa-Entwickler soll die Möglichkeit haben, erstellte Szenariokonfigurationsdateien im Szenarioassistenten zu speichern.

Tabelle A.29: Anforderung ANF-12

A.2.2 Anforderungen an die Agentenanwendung

Bestehende Anforderungen aus Spezifikation

Funktionale Anforderungen

ANF-13:	Modularisierte Struktur: Die Agentenanwendung enthält pro Zielsystem ein spezifisches Modul, das sich um die spezifische Einstellung der Konfigurationsparameter auf dem Zielsystem kümmert
Beschreibung:	Die Agentenanwendung muss die Einstellung von Konfigurationsparametern in die verschiedenen Zielsysteme (FORBOT, Domkamera, Digitaler Lagetisch) ermöglichen. Da der Digitale Lagetisch nicht aus Teilsystemen besteht, stellt er eine Zielplattform und ein Zielsystem dar.

Tabelle A.30: Anforderung ANF-13

ANF-14:	Aktive Module können parametrisiert werden (Konfigurationsdatei)
Beschreibung:	In der Konfigurationsdatei der Agentenanwendung soll die Zielplattform (AMFIS oder Digitaler Lagetisch), auf welcher die Agentenanwendung ausgeführt wird, aktiv geschaltet werden können.

Tabelle A.31: Anforderung ANF-14

ANF-15:	Nutzung von Technologien zur schnellen Serialisierung und Deserialisierung von XML-Datenströmen (z.B. JAXB, XStream)
Beschreibung:	Um die Verarbeitung von XML-Dateien zu vereinfachen soll eine Technologie für die Serialisierung und Deserialisierung eingesetzt werden.

Tabelle A.32: Anforderung ANF-15

ANF-16:	Java-basiert
Beschreibung:	Die Agentenanwendung soll in der Programmiersprache Java realisiert werden.

Tabelle A.33: Anforderung ANF-16

ANF-17:	Auf den Zielplattformen AMFIS-Bodenkontrollstation und Digitaler Lagetisch installierbar
Beschreibung:	Die Agentensoftware muss auf den beiden unterschiedlichen Zielplattformen ausführbar sein.

Tabelle A.34: Anforderung ANF-17

Zusätzlich aufgestellte Anforderungen

Funktionale Anforderungen

ANF-18:	Implementierung einer Komponente für XMPP
Beschreibung:	Die Agentenanwendung benötigt eine XMPP-Komponente mit den folgenden Funktionen: - Verbindungsaufbau - Registrierung und Authentifizierung - Erstellen eines Multiuserchats - Betreten eines Multiuserchats - Senden und Empfangen von XMPP-Nachrichten

Tabelle A.35: Anforderung ANF-18

ANF-19:	Implementierung einer Komponente für AMFIS
Beschreibung:	Die Agentenanwendung muss eine AMFIS-Komponente mit den folgenden Funktionen enthalten: - Aufbau einer Verbindung zum AMFIS-Konnektor - Erstellen und Senden von AMFIS-Nachrichten - Empfangen und Auswerten von AMFIS-Nachrichten - Abfragen der Verfügbarkeit der AMFIS-Zielsysteme - Senden der Verfügbarkeit an den Multiuserchat

Tabelle A.36: Anforderung ANF-19

ANF-20:	Implementierung einer Komponente für den Digitalen Lagetisch
Beschreibung:	Die Agentenanwendung muss eine Komponente zur Kommunikation mit dem Digitalen Lagetisch enthalten. Folgende Funktionen sind in dieser Komponente vorgesehen: - Aufbau einer Verbindung zum Message Broker - Senden einer Nachricht mit den Parametern - Senden der Verfügbarkeit an den Multiuserchat

Tabelle A.37: Anforderung ANF-20

ANF-21:	Auswertung von Szenariokonfigurationsdateien
Beschreibung:	Eine empfangene Szenariokonfigurationsdatei enthält unterschiedliche Konfigurationsparameter für die verschiedenen Zielplattformen und deren Zielsysteme. Diese Konfiurationsparameter müssen ausgewertet und für die Einstellung auf den Zielplattformen weiterverarbeitet werden.

Tabelle A.38: Anforderung ANF-21

ANF-22:	Erstellung einer ausführbaren Java-Archiv-Datei (JAR)
Beschreibung:	Um das Ausführen der Agentenanwendung auf einer Zielplattform zu ermöglichen muss eine Java-Archiv-Datei von der Agentenanwendung erstellt werden. Diese Datei kann dann auf einer Zielplattform ausgeführt werden.

Tabelle A.39: Anforderung ANF-22

B.1 Listings des Szenarioassistenten

Die beigefügte CD-ROM enthält das vollständige Projekt.

```
<?xml version="1.0" encoding="UTF-8"?>
<scenarioConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="scenarioConfig.xsd">
  <metadata>
    <author>feuerb</author>
    <date>30.10.2012</date>
    <version>1.0</version>
  </metadata>
  <information>
    <scenarioName>Liegenschaftsüberwachung IOSB</scenarioName>
    <scenarioDescriptionShort>Liegenschaftsüberwachung
    </scenarioDescriptionShort>
    <scenarioDescriptionLong>Liegenschaftsüberwachung beim Fraunhofer IOSB
         Karlsruhe
    </scenarioDescriptionLong>
    <scenarioLocation>Karlsruhe</scenarioLocation>
    <scenarioType>Liegenschaftsüberwachung</scenarioType>
  </information>
  <nodes>
    <node>
      <nodeId>diglt-1</nodeId>
      <operation>requestScenario</operation>
      <value>1</value>
    </node>
    <node>
      <nodeId>amfis-35</nodeId>
      <operation>lookat</operation>
      <value>49.01534;8.426347;0</value>
    </node>
    <node>
      <nodeId>amfis-36</nodeId>
      <operation>lookat</operation>
      <value>49.01563;8.426079;0</value>
    </node>
    <node>
      <nodeId>amfis-48</nodeId>
      <operation>waypointlist</operation>
      <value>
        49.015861;8.426263;0.0;0.0;0;0;
        49.015459;8.426324;0.0;0.0;0;0;
        49.014842;8.426431;0.0;0.0;0;0;
        49.014911;8.427407;0.0;0.0;0;0;
        49.015047;8.427367;0.0;0.0;0;0;
        49.014987;8.426440;0.0;0.0;0;0;
        49.015355;8.426367;0.0;0.0;0;0;
        49.015864;8.426281;0.0;0.0;0;0
      </value>
    </node>
  </nodes>
</scenarioConfig>
```

Listing B.1: Szenariokonfigurationsdatei *scenarioOne.xml*

```
<!DOCTYPE html>

<html>

<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
        maximum-scale=1.0;" />
    <link rel="stylesheet"  href="css/jquery.mobile.structure.css" />
    <link rel="stylesheet" href="css/jquery.mobile.theme.css" />
    <link rel="stylesheet" href="css/custom.css" />
    <script>
      var userAgent = navigator.userAgent + '';
      if (userAgent.indexOf('iPhone') > -1) {
          document.write('<script src="js/lib/cordova-iphone.js"></sc' + 'ript>'
              );
          var mobile_system = 'iphone';
      } else if (userAgent.indexOf('Android') > -1) {
          document.write('<script type="text/javascript" charset="utf-8" src="js
              /lib/cordova-2.2.0.js"></sc' + 'ript>');
          var mobile_system = 'android';
      } else {
          var mobile_system = '';
      }
    </script>
    <script src="js/lib/jquery.js"></script>
    <script src="js/lib/strophe.js"></script>
    <script src="js/lib/strophe.muc.js"></script>
    <script src="js/lib/strophe.register.js"></script>
    <script src="js/app/xmppmanager.js"></script>
    <script src="js/app/app.js"></script>
    <script src="js/app/bootstrap.js"></script>
    <script src="js/lib/jquery.mobile.js"></script>
</head>

<body>
    <div data-role="page" id="mainPage" data-theme="a">
      <div data-role="header" >
      <h1>- SzenAs+ -</h1>
    </div>

    <div data-role="content">
    <div id="connectionState">
      </div>
    <div id="chooseScenario" style="padding-bottom: 2em; padding-top: 1em;" >
      Bitte Szenario wählen:
      </div>

      <ul data-role="listview" id="lv_scenarios" >
          <li id="scenarioListItemOne" class="scenarioListItem">
            <a href="scenarioDetailsPageOne.html" >
        <img src="images/jquery.jpg" />
        <h2>Liegenschaftsüberwachung</h2><p>Fraunhofer IOSB, Karlsruhe</p>
            </a>
        </li>

          <li id="scenarioListItemTwo" class="scenarioListItem">
            <a href="scenarioDetailsPageTwo.html" >
            <img src="images/jquery.jpg" />
```

```
            <h2>Liegenschaftsüberwachung</h3><p>WTD 81, Greding</p>
            </a>
        </li>

          <li id="scenarioListItemThree" class="scenarioListItem">
        <a href="scenarioDetailsPageThree.html" >
          <img src="images/jquery.jpg" />
          <h2>Grenzkontrolle, Bereich B2</h3><p>WTD 81, Greding</p>
            </a>
          </li>
      </ul>
    </div>
  </div>
</body>
</html>
```

Listing B.2: Datei *index.html*

```
var XmppConnection = {
    connection: null,
    room: "szenasplusroomconfigured@conference.server.tld",
    nickname: "SzenAsClient",
    jid: "szenasclient@server.tld/SzenAsClient",
    NS_MUC: "http://jabber.org/protocol/muc",
    url: "http://10.57.15.111:17070/http-bind/",
    domain: "server.tld",
    deviceStatusDigLT: {},
    deviceStatusAMFIS: {},
    participantsToDisplay: [],
    registrationUsername: "szenasclient",
    password: "szenasclient123",

    log: function (msg) {
        console.log(msg);
    },

    onGroupchatMessage: function (message) {

        var from = $(message).attr('from');
        var room = Strophe.getBareJidFromJid(from);
        var nick = Strophe.getResourceFromJid(from);
        var msg = "";

        if(nick !== null) {
          msg = $(message).children('body').text();

          if(msg.search("<AMFISDeviceStatus") !== -1) {

            var nodes = getDeviceStatusDataFromXmlString(msg);

            if(XmppConnection.participantsToDisplay.length === 0) {
              for (var i = 0; i < nodes.length; i++) {

                XmppConnection.deviceStatusAMFIS.name = nodes[i].name;
                XmppConnection.deviceStatusAMFIS.timeoutCounter = 15;
                    XmppConnection.participantsToDisplay.push(XmppConnection.
                        deviceStatusAMFIS);
                    XmppConnection.deviceStatusAMFIS = {};
```

```
            }
        }

        for(var i = 0; i < nodes.length; ++i) {

          var found = false;

          for(var j = 0; j < XmppConnection.participantsToDisplay.length; ++
              j) {

            if(nodes[i].name === XmppConnection.participantsToDisplay[j].
                name) {
                found = true;
                XmppConnection.participantsToDisplay[j].timeoutCounter = 15;
              }

          }

          if(!found) {
            XmppConnection.deviceStatusAMFIS.name = nodes[i].name;
            XmppConnection.deviceStatusAMFIS.timeoutCounter = 15;
            XmppConnection.participantsToDisplay.push(XmppConnection.
                deviceStatusAMFIS);
            XmppConnection.deviceStatusAMFIS = {};
          }

        }

      }

      if(msg.search("<DigLTDeviceStatus") !== -1) {

        var nodes = getDeviceStatusDataFromXmlString(msg);

        if(XmppConnection.participantsToDisplay.length === 0) {

          for (var i = 0; i < nodes.length; i++) {

            XmppConnection.deviceStatusDigLT.name = nodes[i].name;
            XmppConnection.deviceStatusDigLT.timeoutCounter = 15;
                XmppConnection.participantsToDisplay.push(XmppConnection.
                    deviceStatusDigLT);
                XmppConnection.deviceStatusDigLT = {};

            }
        }

        for(var i = 0; i < nodes.length; ++i) {

          var found = false;

          for(var j = 0; j < XmppConnection.participantsToDisplay.length; ++
              j) {

            if(nodes[i].name === XmppConnection.participantsToDisplay[j].
                name) {
                found = true;
                XmppConnection.participantsToDisplay[j].timeoutCounter = 15;
              }
```

```
            }

            if(!found) {
              XmppConnection.deviceStatusDigLT.name = nodes[i].name;
              XmppConnection.deviceStatusDigLT.timeoutCounter = 15;
              XmppConnection.participantsToDisplay.push(XmppConnection.
                  deviceStatusDigLT);
              XmppConnection.deviceStatusDigLT = {};
            }

          }

        }

      }

      return true;

  },

  sendPresence: function () {
    if(XmppConnection.connection !== null) {
      XmppConnection.connection.send($pres().c('priority').t('-1'));
    XmppConnection.connection.send(
        $pres({to: XmppConnection.room + "/" + XmppConnection.nickname
        }).c("x", {xmlns: XmppConnection.NS_MUC}).c("history", {maxchars: '0'
            }));
    }
  },

  decrementTimeoutCounter: function () {

    if(XmppConnection.participantsToDisplay.length !== 0) {

      for(var i = 0; i < XmppConnection.participantsToDisplay.length; ++i) {
            if(XmppConnection.participantsToDisplay[i].timeoutCounter < 0) {
              XmppConnection.participantsToDisplay[i].timeoutCounter = 0;
            } else {
              XmppConnection.participantsToDisplay[i].timeoutCounter -= 1;
            }

          }

    }
  },

  displayDeviceStatus: function () {

    var devices = ['Forbot', 'Axis-Dome 1', 'Axis-Dome 2', 'AirRobot 1', '
        AirRobot 2', 'DigLT'];

    if(XmppConnection.participantsToDisplay.length !== 0) {

    $("#lv_deviceStatus").html("<ul id='lv_deviceStatus' data-role='listview'
        data-inset='true'></ul>");

    var list = "";

    var devicesOnline = [];
```

```
      for(var i = 0; i < devices.length; ++i) {
          for(var j = 0; j < XmppConnection.participantsToDisplay.length; ++j) {
            if(devices[i] === XmppConnection.participantsToDisplay[j].name &&
                XmppConnection.participantsToDisplay[j].timeoutCounter > 0) {
              list += "<li data-icon=\"check\" style=\"background: green;\"><a
                  href=\"#\">" + devices[i] + "</a></li>";
              devicesOnline.push(devices[i]);
            }
          }
        }
      }

      if(devicesOnline !== undefined) {
        for(var i = 0; i < devices.length; ++i) {
          if(devicesOnline.contains(devices[i])) {
            continue;
          } else {
            list += "<li data-icon=\"delete\" style=\"background: red;\"><a href
                =\"#\">" + devices[i] + "</a></li>";
          }
        }
      }

      $("#lv_deviceStatus").html(list);
      $("#lv_deviceStatus").listview("refresh");

    }

};

$(document).bind('register', function () {

   var conn = new Strophe.Connection(XmppConnection.url);

   var register = function (status) {
  if (status === Strophe.Status.CONNECTED) {
        $('#connectionState').html("XMPP-Server Status: Verbunden");
        $(document).trigger('connected');
    } else if (status === Strophe.Status.REGISTER) {
        conn.register.fields.username = XmppConnection.registrationUsername;
        conn.register.fields.password = XmppConnection.password;
        conn.register.submit();
    } else if (status === Strophe.Status.REGISTERED) {
        doConnect();
    }

  };

  conn.register.connect(XmppConnection.domain, register, 60, 1);

  XmppConnection.connection = conn;

});

$(document).bind('connect', function (ev, data) {
    var conn = new Strophe.Connection(XmppConnection.url);

    // conn.rawInput = function (data) { console.log('RECV: ' + data); };
  //conn.rawOutput = function (data) { console.log('SENT: ' + data); };
```

```
    // Strophe.log = function (lvl, msg) { console.log("LOG: " + msg); };
    conn.connect(data.jid, data.password, function (status) {
        if (status === Strophe.Status.CONNECTED) {
          $('#connectionState').html("XMPP-Server Status: Verbunden");
            $(document).trigger('connected');
        } else if (status === Strophe.Status.DISCONNECTED) {
          $('#connectionState').html("XMPP-Server Status: Verbindung beendet");
            doConnect();
        }
        else if (status === Strophe.Status.ERROR) {
          $('#connectionState').html("XMPP-Server Status: Fehler");
        } else if (status === Strophe.Status.CONNECTING) {
          $('#connectionState').html("XMPP-Server Status: Verbinde");
        } else if (status === Strophe.Status.CONNFAIL) {
          $('#connectionState').html("XMPP-Server Status: Verbindung
              unterbrochen");
        } else if (status === Strophe.Status.AUTHENTICATING) {
          $('#connectionState').html("XMPP-Server Status: Authentifiziere");
        } else if (status === Strophe.Status.AUTHFAIL) {
          $('#connectionState').html("XMPP-Server Status: Authentifizierung
              fehlgeschlagen");
            $(document).trigger('register');
        } else if (status === Strophe.Status.ATTACHED) {
          $('#connectionState').html("XMPP-Server Status: Session hergestellt");
        }
    });

    XmppConnection.connection = conn;

});

$(document).bind('connected', function () {

  XmppConnection.connection.send($pres().c('priority').t('-1'));

  XmppConnection.connection.muc.init(XmppConnection.connection);

  XmppConnection.connection.muc.createInstantRoom(XmppConnection.room,
      createRoomSuccess, createRoomError);

  XmppConnection.connection.send($pres({to: XmppConnection.room + "/" +
      XmppConnection.nickname
  }).c("x", {xmlns: XmppConnection.NS_MUC}).c("history", {maxchars: '0'}));
  XmppConnection.connection.addHandler(XmppConnection.onGroupchatMessage,
            null, "message", "groupchat");

});

$(document).bind('disconnected', function () {
    // remove dead connection object
    XmppConnection.connection = null;
    doConnect();

});

function doConnect() {

   $(document).trigger('connect', {
       jid : XmppConnection.jid,
```

```
        password : XmppConnection.password
        });

}
function getDeviceStatusDataFromXmlString(xmlString) {

    // Parse the XML string into a XMLDocument
    var doc = new DOMParser().parseFromString(xmlString, 'text/xml');

    // Find the sensornode nodes
    var devices = doc.getElementsByTagName('node');
    var deviceNode = {};
    var nodes = [];
    var node;

    // Loop through them and save their text content into an array
    for (var i = 0; i < devices.length; i++) {
      node = devices[i];
      deviceNode.nodeId = $(node).find( "nodeId" ).text();
      deviceNode.name = $(node).find( "name" ).text();
      nodes.push(deviceNode);
      deviceNode = {};

    }

    return nodes;

}

function createRoomSuccess() {
  //alert('room created');
}

function createRoomError() {
  //alert('room was not created');
}


Array.prototype.contains = function(obj) {
    var i = this.length;
    while (i--) {
        if (this[i] == obj) {
            return true;
        }
    }
    return false;
}
```

Listing B.3: Datei *xmppmanager.js*

```
var startApp = function() {
    //define the application
  var SzenAsPlus = {};

  (function(app){

    // set a few variables which can be used within the app
    var scenarioXML;
    var filename;
    var fileScenarioOne = "scenarioOne.xml";
    var fileScenarioTwo = "scenarioTwo.xml";
    var fileScenarioThree = "scenarioThree.xml";

    app.init = function(){
      app.bindings();
      window.setInterval(XmppConnection.sendPresence, 5000);
      window.setInterval(XmppConnection.decrementTimeoutCounter, 1000);
      window.setInterval(XmppConnection.displayDeviceStatus, 2000);
    }

    app.bindings = function(){
      // set up binding

      $(".scenarioListItem").live("click", function (event)
      {
        event.preventDefault();

        if(this.id === "scenarioListItemOne") {
          loadScenario(fileScenarioOne);
        }
        else if(this.id === "scenarioListItemTwo") {
          loadScenario(fileScenarioTwo);
        }
        else if(this.id === "scenarioListItemThree") {
          loadScenario(fileScenarioThree);
        }

      });

      $(".btnStartScenario").live("click", function (event)
      {

          var body = scenarioXML;
          XmppConnection.connection.send($msg({to: XmppConnection.room, type: "
              groupchat"}).c('body').t(scenarioXML));
          alert("Das Szenario wurde gestartet.");
          $.mobile.changePage("index.html");

      });

    }

    function loadScenario(file) {
      filename = file;
      loadXMLFile(filename);
    };

    function loadXMLFile(filename) {
       $.ajax({   url: "xml/" + filename,
```

```
            type: 'GET',
            dataType: 'xml',
            success: function(result) {
              scenarioXML = new XMLSerializer().serializeToString(result);
            },
            error: function() {
                    alert("Szenariokonfiguration konnte nicht geladen werden.");
            }
                    })

    };

    app.init();

  })(SzenAsPlus);

};
```

Listing B.4: Datei *app.js*

B.2 Listings der Agentenanwendung

Die beigefügte CD-ROM enthält das vollständige Projekt.

```
<?xml version="1.0" encoding="UTF-8"?>
<agentConfig xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    noNamespaceSchemaLocation="agentConfig.xsd">
  <module>
    <target>diglt-1</target>
    <active>true</active>
    <xmppUser>DigLT</xmppUser>
    <xmppPass>diglt1234</xmppPass>
  </module>
</agentConfig>
```

Listing B.5: XML-Instanzdokument einer Agentenkonfiguration

```
package de.fraunhofer.iosb.szenas.agentcontrol;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileDescriptor;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.io.UnsupportedEncodingException;
import java.net.URISyntaxException;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;
import java.util.Timer;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;

import org.jivesoftware.smack.XMPPException;

import de.fraunhofer.iosb.szenas.amfis.AmfisConnection;
import de.fraunhofer.iosb.szenas.amfis.DeviceStatusTask;
import de.fraunhofer.iosb.szenas.amfis.AmfisDeviceStatusTask;
import de.fraunhofer.iosb.szenas.diglt.DigLTStatusTask;
import de.fraunhofer.iosb.szenas.jaxb.AgentConfig;
import de.fraunhofer.iosb.szenas.jaxb.AgentConfig.Module;
import de.fraunhofer.iosb.szenas.model.SensornodeModel;
import de.fraunhofer.iosb.szenas.util.Constants;
import de.fraunhofer.iosb.szenas.xmpp.XMPPServerConnection;


public class AgentController {

  static AgentConfig agentConfig = null;
  static int sensornodeId;
  static boolean isDigLTActive = false;
  static boolean isSensornodeModuleActive = false;
  static boolean isScenarioRunning = false;
```

```
static List<Module> modules;
static String xmppUser;
static ArrayList<SensornodeModel> scenarioSensornodes;
static SensornodeModel sensornodeModel;
static boolean isAmfisConnectionInitialized = false;


public static void main(String[] args) throws XMPPException {

  System.out.print("Bitte den Pfad zur Konfigurationsdatei eingeben: ");

  String path = null;

  BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

  try {
    path = br.readLine();
  } catch (IOException e) {
    System.out.println("Pfad konnte nicht eingelesen werden.");
    e.printStackTrace();
  }

  try {

  for(Module module : readExternalConfiguration(path)) {
    if(module.getActive().equals("true") && module.getTarget().startsWith("
        diglt")) {

      //create a diglt module
      System.out.println("Erzeuge DigLT Modul...");

      Timer digLTTimer = new Timer();

          digLTTimer.schedule(new DigLTStatusTask(new XMPPServerConnection(
          Constants.DIGLT_XMPP_STATUS_BOT_USERNAME, Constants.
              DIGLT_XMPP_STATUS_BOT_PASSWORD,
          Constants.DIGLT_XMPP_STATUS_BOT_NICKNAME, Constants.
              DIGLT_XMPP_STATUS_BOT_RESOURCE)), 5000/* delay ms*/, 5000/*
              period ms*/);

      IModule digLTModule = ModuleFactory.getModule(module.getTarget());
      digLTModule.run(module.getTarget(), module.getXmppUser(), module.
          getXmppPass());


    } else if(module.getActive().equals("true") && module.getTarget().
        startsWith("amfis")){

      //create an amfis connection if not initialized and start timer task
      if(!isAmfisConnectionInitialized) {
        AmfisConnection amfisConnection = AmfisConnection.getInstance();
        amfisConnection.connect();
        amfisConnection.addListener();
        Timer timer = new Timer();
        timer.schedule(new DeviceStatusTask(), 5000/* delay ms*/, 1000/*
            period ms*/);
        timer.schedule(new AmfisDeviceStatusTask(new XMPPServerConnection(
                    Constants.AMFIS_XMPP_STATUS_BOT_USERNAME, Constants.
                        AMFIS_XMPP_STATUS_BOT_PASSWORD,
```

```
                    Constants.AMFIS_XMPP_STATUS_BOT_NICKNAME, Constants.
                        AMFIS_XMPP_STATUS_BOT_RESOURCE)), 5000/* delay ms*/,
                        5000/* period ms*/);
          isAmfisConnectionInitialized = true;
          System.out.println("AMFIS-Konnektor-Verbindung hergestellt und Timer
              gestartet...");
        }

        //create an amfis module
        System.out.println("Erzeuge AMFIS-Modul...");

        IModule amfisModule = ModuleFactory.getModule(module.getTarget());
        amfisModule.run(module.getTarget(), module.getXmppUser(), module.
            getXmppPass());

      }

    }

    } catch (NullPointerException e) {
      System.out.println("Konfigurationsdatei konnte nicht gelesen werden.");
    }

    System.out.println();
    System.out.println("Zum Beenden Eingabetaste druecken.");
    System.out.println();

    try {
      System.in.read();
      System.exit(0);
    } catch (IOException e) {
      e.printStackTrace();
    }

}



public static void setIsScenarioRunning(boolean value) {
  isScenarioRunning = value;
}

public static boolean getIsScenarioRunning() {
  return isScenarioRunning;
}

public static ArrayList<SensornodeModel> getScenarioSensornodes() {
  return scenarioSensornodes;
}

public static SensornodeModel getSensornodeModelBySensornodeId(int
    sensornodeId) {

  for(int i = 0; i < scenarioSensornodes.size(); ++i) {
    if(sensornodeId == scenarioSensornodes.get(i).getSensornodeId()) {
      return scenarioSensornodes.get(i);
    }
  }

  return null;
```

```
  }

  public static List<Module> readExternalConfiguration(String path) throws
      NullPointerException{
    try {

      File file = new File(path + "\\agentConfig.xml");

      JAXBContext jaxbContext = JAXBContext
          .newInstance(AgentConfig.class);

      Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

      agentConfig = (AgentConfig) jaxbUnmarshaller
          .unmarshal(file);

    } catch (JAXBException e) {
    }

      modules = agentConfig.getModules();

    return modules;

  }

}
```

Listing B.6: Klasse *AgentController*

```
package de.fraunhofer.iosb.szenas.agentcontrol;

import de.fraunhofer.iosb.szenas.xmpp.XMPPServerConnection;

public class DigLTModule implements IModule {

  static XMPPServerConnection digLTConnection;

  @Override
  public void run(String target, String xmppUser, String xmppPass) {

    digLTConnection = new XMPPServerConnection(xmppUser, xmppPass, xmppUser,
        xmppUser);

    digLTConnection.login();
    digLTConnection.addPacketListener(null);

  }

  public static XMPPServerConnection getDigLTXMPPConnection() {
    return digLTConnection;
  }

}
```

Listing B.7: Klasse *DigLTModule*

```
package de.fraunhofer.iosb.szenas.agentcontrol;

import de.fraunhofer.iosb.szenas.model.SensornodeModel;
import de.fraunhofer.iosb.szenas.xmpp.XMPPServerConnection;


public class AmfisModule implements IModule {

  @Override
  public void run(String target, String xmppUser, String xmppPass) {

    Integer sensornodeId = Integer.valueOf(target.substring(6));

    SensornodeModel sensornodeModel = new SensornodeModel(sensornodeId, xmppUser
        ,
    xmppPass, xmppUser, xmppUser,
      new XMPPServerConnection(xmppUser, xmppPass,
          xmppUser, xmppUser));

    sensornodeModel.getConnection().login();
    sensornodeModel.getConnection().addPacketListener(sensornodeModel);

  }

}
```

Listing B.8: Klasse *AmfisModule*

```
package de.fraunhofer.iosb.szenas.amfis;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.IOException;
import java.net.UnknownHostException;
import java.util.ArrayList;

import de.fraunhofer.iosb.amfis.amfisCom.ConnectionReader;
import de.fraunhofer.iosb.amfis.amfisCom.ConnectorClient;
import de.fraunhofer.iosb.amfis.amfisCom.castor.message.Message;
import de.fraunhofer.iosb.amfis.amfisCom.castor.message.types.
    MessageTypeAttributeType;
import de.fraunhofer.iosb.amfis.amfisCom.castor.sensorweb.Sensor;
import de.fraunhofer.iosb.amfis.amfisCom.castor.sensorweb.Sensornetwork;
import de.fraunhofer.iosb.amfis.amfisCom.castor.sensorweb.Sensornode;
import de.fraunhofer.iosb.amfis.amfisCom.castor.sensorweb.Sensorweb;
import de.fraunhofer.iosb.szenas.model.SensorModel;
import de.fraunhofer.iosb.szenas.model.SensornetworkModel;
import de.fraunhofer.iosb.szenas.model.SensornodeModel;
import de.fraunhofer.iosb.szenas.model.SensorwebModel;
import de.fraunhofer.iosb.szenas.util.Constants;

//@Singleton
public final class AmfisConnection {

  ConnectorClient client;
  ConnectionReader reader;

  Sensorweb sensorweb;
  Sensornetwork sensornetwork;
  Sensornode sensornode;
  Sensor sensor;
  Message msg;

  ArrayList<SensornodeModel> scenarioSensornodes;
  ArrayList<DeviceStatus> deviceStatusList = new ArrayList<DeviceStatus>();

  DeviceStatus deviceStatus;

  String[] position;

  private static AmfisConnection instance;

  private AmfisConnection() {
  }

  public synchronized static AmfisConnection getInstance() {
    if (instance == null) {
      instance = new AmfisConnection();
    }
    return instance;
  }

  public ArrayList<DeviceStatus> getDeviceStatusList() {
    return deviceStatusList;
  }
```

```
public void setScenarioSensornodes(ArrayList<SensornodeModel>
    scenarioSensornodes) {
  this.scenarioSensornodes = scenarioSensornodes;
}

public void connect() {
  try {
    client = new ConnectorClient();
    client.connect(Constants.AMFIS_CONNECTOR_HOST, Constants.
        AMFIS_CONNECTOR_PORT);
  } catch (UnknownHostException e) {
    System.out.println("Keine Verbindung zum AMFIS-Konnektor moeglich.");
  } catch (IOException e) {
    System.out.println("Keine Verbindung zum AMFIS-Konnektor moeglich.");
  }
}

public void addListener() {
  client.addActionListener(new AmfisConnectionListener());
}

public void sendAmfisMessage(Message msg) {

    client.sendMessage(msg);

}

private class AmfisConnectionListener implements ActionListener {

  @Override
  public void actionPerformed(ActionEvent event) {

    String cmd = event.getActionCommand();

    SensorwebModel sensorwebModel = new SensorwebModel();

    if ("getSensorweb".equals(cmd)) {

      sensorweb = ((ConnectionReader) event.getSource()).getSensorweb();

        for (int i = 0; i < sensorweb.getSensornetworkCount(); ++i) {

          sensornetwork = sensorweb.getSensornetwork(i);

          sensorwebModel.getSensornetworks().add(new SensornetworkModel(
              sensornetwork.getId(), sensornetwork.getName(),
              sensornetwork.getDesc(), String.valueOf(sensornetwork.
                  getConnection()), sensornetwork.getAddress()));

          for (int j = 0; j < sensornetwork.getSensornodeCount(); ++j) {

            sensornode = sensornetwork.getSensornode(j);

            deviceStatus = new DeviceStatus(sensornode.getId(), sensornode.
                getName(), sensornode.getMobile());

            deviceStatusList.add(deviceStatus);

            sensorwebModel.getSensornetworks().get(i).getSensornodes().add(new
                SensornodeModel(sensornode.getId(), sensornode.getName(),
```

```
                    sensornode.getAlias(), sensornode.getMobile(), sensornode.
                        getPosition(), sensornode.getTimeout()));

                for (int k = 0; k < sensornode.getSensorCount(); ++k) {

                  sensor = sensornode.getSensor(k);

                  sensorwebModel.getSensornetworks().get(i).getSensornodes().get
                      (j).getSensors().add(new SensorModel(sensor.getId(),
                      sensor.getPriority(),
                              sensor.getClazz(), sensor.getManufacturer(),
                                sensor.getDesc(), sensor.getThreshold_min(),
                                  sensor.getThreshold_max()));

                }

            }

          }

      }


if ("getMessage".equals(cmd)) {

        msg = ((ConnectionReader) event.getSource()).getMessage();

        if(deviceStatusList.size() != 0) {
          for(int i = 0; i < deviceStatusList.size(); ++i) {

            if(deviceStatusList.get(i).getSensornodeId() == msg.getOriginator().
                getSensornodeid()) {
              //reset timeout counter
              deviceStatusList.get(i).resetTimeoutCounter();
              deviceStatusList.get(i).setOnline(true);

                if(msg.getType() == MessageTypeAttributeType.POSITION) {
                  position = msg.getValue().split(";");

                  for(int j = 0; j < position.length; ++j) {
                    if(!position[j].equals("0")) {
                    deviceStatusList.get(i).setPosition(true);
                    break;
                    }

                  }

                  deviceStatusList.get(i).setPositionValues(position);

                }

            }

        }

          for(int i = 0; i < deviceStatusList.size(); ++i) {
            if((deviceStatusList.get(i).isOnline() && !deviceStatusList.get(i).
                isMobile)) {
              deviceStatusList.get(i).setControllable(true);
              //Controllable and not mobile
```

```
            } else if(msg.getType() == MessageTypeAttributeType.POSITION && (
                deviceStatusList.get(i).isOnline() && deviceStatusList.get(i).
                isPosition())) {
              deviceStatusList.get(i).setControllable(true);
              //Controllable and mobile
              } else if(msg.getType() == MessageTypeAttributeType.POSITION && (
                  deviceStatusList.get(i).isOnline() && !deviceStatusList.get(i)
                  .isPosition())) {
                deviceStatusList.get(i).setControllable(false);
              }

          }

        }

      }

    }

  }

}
```

Listing B.9: Klasse *AmfisConnection*

```
package de.fraunhofer.iosb.szenas.diglt;

import java.util.ArrayList;
import java.util.UUID;

import javax.jms.Connection;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.MessageProducer;
import javax.jms.ObjectMessage;
import javax.jms.Session;
import javax.jms.TextMessage;

import org.apache.activemq.ActiveMQConnection;
import org.apache.activemq.ActiveMQConnectionFactory;

import de.fraunhofer.iosb.szenas.agentcontrol.AgentController;
import de.fraunhofer.iosb.szenas.util.Constants;

public class DigLTJMSClient {

  private ActiveMQConnectionFactory connectionFactory;

  private Connection connection;

  private Session session;

  private MessageProducer scenarioExecutionProducer;
  private MessageProducer scenariolistProducer;
```

```
  private MessageConsumer scenariolistConsumer;

  private Destination executionTopic;
  private Destination listTopic;
  private Destination receiveListQueue;

  private ObjectMessage executionObjectMsg;

  public void start() throws JMSException {

    //org.apache.log4j.BasicConfigurator.configure();

    connectionFactory = new ActiveMQConnectionFactory(Constants.DIGLT_BROKER_URL
        );

    connection = connectionFactory.createConnection();
    connection.start();

    session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);

    executionTopic = session.createTopic("SCENARIO_EXECUTION_TOPIC");
    scenarioExecutionProducer = session.createProducer(executionTopic);
    scenarioExecutionProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

    listTopic = session.createTopic("SCENARIOLIST_TOPIC");
    scenariolistProducer = session.createProducer(listTopic);
    scenariolistProducer.setDeliveryMode(DeliveryMode.NON_PERSISTENT);

    receiveListQueue = session.createQueue("SCENARIOLIST_Q");
    scenariolistConsumer = session.createConsumer(receiveListQueue);
    scenariolistConsumer.setMessageListener(new MessageListener() {

      @Override
      public void onMessage(Message msg) {

        if (msg instanceof ObjectMessage) {
          ObjectMessage om = (ObjectMessage)msg;
          Scenariolist list;
          try {
            list = (Scenariolist) om.getObject();
            ArrayList<Scenario> scenarios = list.getScenariolist();
            System.out.println("Scenariolist:\n");

            for(Scenario scenario : scenarios) {
              System.out.println("Scenario id: " + scenario.getScenarioId()
                  + " Scenario Ort: " + scenario.getScenarioLocation());
            }
          } catch (JMSException e) {
            System.out.println("Objektnachricht konnte nicht deserialisiert
                werden.");
          }
        }
      }
    });
  }

  public void stop() throws JMSException {
    scenarioExecutionProducer.close();
    scenariolistProducer.close();
    scenariolistConsumer.close();
```

```
    session.close();
    connection.close();
  }

  public void requestScenarioExecution(int scenarioId) {

    System.out.println("Szenarioausführung für id "
        + scenarioId + " angefordert.");

    try {
      executionObjectMsg = session.createObjectMessage(new ScenarioExecutionMsg(
          scenarioId));
    } catch (JMSException e) {
      System.out.println("Objektnachricht für die Szenarioanforderung konnte
          nicht erstellt werden.");
    }

    try {
      this.scenarioExecutionProducer.send(executionObjectMsg);
    } catch (JMSException e) {
      System.out.println("Objektnachricht für die Szenarioanforderung konnte
          nicht gesendet werden.");
      e.printStackTrace();
    }

  }

  public void requestScenariolist() throws JMSException {

    System.out.println("Fordere Szenarioliste an...");

    TextMessage txtMessage = session.createTextMessage();

    this.scenariolistProducer.send(txtMessage);

  }


}
```

Listing B.10: Klasse *DigLTJMSClient*

```
package de.fraunhofer.iosb.szenas.xmpp;

import javax.jms.JMSException;

import org.jivesoftware.smack.ConnectionConfiguration;
import org.jivesoftware.smack.PacketListener;
import org.jivesoftware.smack.SASLAuthentication;
import org.jivesoftware.smack.SmackConfiguration;
import org.jivesoftware.smack.XMPPConnection;
import org.jivesoftware.smack.XMPPException;
import org.jivesoftware.smack.packet.Message;
import org.jivesoftware.smack.packet.Packet;
import org.jivesoftware.smack.packet.Presence;
import org.jivesoftware.smackx.Form;
import org.jivesoftware.smackx.muc.DiscussionHistory;
import org.jivesoftware.smackx.muc.MultiUserChat;
```

```
import de.fraunhofer.iosb.szenas.diglt.DigLTJMSClient;
import de.fraunhofer.iosb.szenas.jaxb.ScenarioConfig;
import de.fraunhofer.iosb.szenas.jaxb.ScenarioConfig.Nodes.Node;
import de.fraunhofer.iosb.szenas.model.EncodeLookAt;
import de.fraunhofer.iosb.szenas.model.EncodeWaypointlist;
import de.fraunhofer.iosb.szenas.model.SensornodeModel;
import de.fraunhofer.iosb.szenas.util.Constants;
import de.fraunhofer.iosb.szenas.util.XMLProcessor;

public class XMPPServerConnection {

  XMPPConnection connection;
  MultiUserChat muc;

  String room;
  String nick;
  String username;
  String password;
  String resource;
  int nodeId;
  String digLTScenarioId;
  String operation;

  public XMPPServerConnection(String pUsername, String pPassword, String pNick,
      String pResource) {
    this.room = Constants.XMPP_MUC;
    this.username = pUsername;
    this.password = pPassword;
    this.nick = pNick;
    this.resource = pResource;
  }

  @SuppressWarnings("static-access")
  public void login() {

    ConnectionConfiguration config = new ConnectionConfiguration(
        Constants.XMPP_SERVER_URL, Constants.XMPP_SERVER_PORT);
    this.connection = new XMPPConnection(config);

    try {
            this.connection.connect();

    } catch (XMPPException e1) {
      System.out.println("Fehler bei Aufbau der XMPP-Verbindung.");
    }

    SASLAuthentication.supportSASLMechanism("PLAIN", 0);

    try {

          this.connection.login(username, password, resource);

          connection.sendPacket(new Presence(Presence.Type.available,
                      "Bereit...", -1, Presence.Mode.available));

    } catch (XMPPException xe) {
      //If first login failed, try to create an account and then login
      //Close and recreate connection first
          connection.disconnect();
```

```
        connection = new XMPPConnection(config);

        try {

            connection.connect();
            connection.getAccountManager().createAccount(username, password);
            connection.login(username, password, resource);
            connection.sendPacket(new Presence(Presence.Type.available,
                        "Ready...", -1, Presence.Mode.available));
            System.out.println("XMPP-Konto erstellt und eingeloggt...");

        } catch (XMPPException e1) {
            System.out.println("Fehler beim XMPP-Konto erstellen und einloggen
                .");
        }

    }


this.muc = new MultiUserChat(connection, Constants.XMPP_MUC);

DiscussionHistory history = new DiscussionHistory();
  history.setMaxStanzas(0);

try {

           this.muc.getRoomInfo(connection, Constants.XMPP_MUC);

    } catch (XMPPException e1) {

      try {
        this.muc.create(username);
    this.muc.sendConfigurationForm(new Form(Form.TYPE_SUBMIT));
  } catch (XMPPException e) {
    System.out.println("Fehler bei Erzeugung des Chatrooms...");
  }

      try {
    this.muc.join(username,"", history, SmackConfiguration.
        getPacketReplyTimeout());
  } catch (XMPPException e) {
    System.out.println("Fehler beim Betreten des Chatrooms...");
  }

    }

try {
  this.muc.join(username, "", history, SmackConfiguration.
      getPacketReplyTimeout());
} catch (XMPPException e) {
  System.out.println("Fehler beim Betreten des Chatrooms...");
}

try {
    this.muc.sendConfigurationForm(new Form(Form.TYPE_SUBMIT));

    } catch (XMPPException e1) {

            System.out.println("Chatroom wurde bereits konfiguriert...");
```

```
      }

}

public void addPacketListener(final SensornodeModel sensornode) {
  if(sensornode != null) {
  //Add a message listener for the running AMFIS instance
  sensornode.getConnection().muc.addMessageListener(new PacketListener() {
    @Override
    public void processPacket(Packet packet) {
      Message message = (Message) packet;

      if(message.getBody().contains("scenarioConfig")){

        ScenarioConfig scenarioConfig = XMLProcessor
              .getInstance().unmarshallScenarioconfigurationFromString(
                    message.getBody());

        for(Node node : scenarioConfig.getNodes().getNode()) {

          nodeId = Integer.valueOf(node.getNodeId().substring(6));

          if(nodeId == sensornode.getSensornodeId()) {
            String msg = "\nKonfiguration erhalten!\n";
                 msg += "Sende Nachricht für Operation: " + node.getOperation
                     () + "\n";
                 msg += "mit Wert: " + node.getValue() + "\n";

            sensornode.getConnection().sendMessage(msg);

              String operation = node.getOperation();

              if("lookat".equals(operation)) {
                sensornode.setEncodeBehavior(new EncodeLookAt());
                sensornode.performEncode(nodeId, node.getValue());
              } else if("waypointlist".equals(operation)) {
                sensornode.setEncodeBehavior(new EncodeWaypointlist());
                sensornode.performEncode(nodeId, node.getValue());
              }

          }
        }

      }

    }
  });
  } else {
    //Add a message listener for the running DigLT instance
    this.muc.addMessageListener(new PacketListener() {

      @Override
      public void processPacket(Packet packet) {
        Message message = (Message) packet;

        if(message.getBody().contains("scenarioConfig")){

          ScenarioConfig scenarioConfig = XMLProcessor
                .getInstance().unmarshallScenarioconfigurationFromString(
                      message.getBody());
```

```
            for(Node node : scenarioConfig.getNodes().getNode()) {
              if(node.getNodeId().startsWith("diglt")) {
                digLTScenarioId = node.getValue();
                operation = node.getOperation();
                break;
              }
            }

            if(operation.equals("requestScenario")) {

                sendMessage("Konfiguration erhalten...Szenarioanforderung fuer
                    ID: " + digLTScenarioId);

                DigLTJMSClient client = new DigLTJMSClient();
                try {
                  client.start();
                  client.requestScenarioExecution(Integer.parseInt(
                      digLTScenarioId));
                  try {
                    //Wait
                    Thread.sleep(3000);
                  } catch (InterruptedException e) {
                    e.printStackTrace();
                  }
                  client.stop();
                } catch (JMSException e) {
                  System.out.println("Anforderung der Szenarioausfuehrung
                      gescheitert...");
                }
            }

            operation = "";

          }

        }
      });
    }


  }

  public void sendMessage(String message) {
    try {
      muc.sendMessage(message);
    } catch (XMPPException e) {
      System.out.println("Nachricht konnte nicht an den Chatroom gesendet werden
          .");
    }
  }

}
```

Listing B.11: Klasse *XMPPServerConnection*

```java
package de.fraunhofer.iosb.szenas.diglt;

import java.io.Serializable;

public class ScenarioExecutionMsg implements Serializable {

  /**
   *
   */
  private static final long serialVersionUID = 8142969352178374969L;

  int scenarioId;

  public ScenarioExecutionMsg(int scenarioId) {
    this.scenarioId = scenarioId;
  }

  public int getScenarioId() {
    return scenarioId;
  }

  public void setScenarioId(int scenarioId) {
    this.scenarioId = scenarioId;
  }

  @Override
  public String toString() {
    return "ScenarioExecutionMsg [scenarioId=" + scenarioId + "]";
  }

}
```

Listing B.12: Klasse *ScenarioExecutionMsg*

```java
package de.fraunhofer.iosb.szenas.amfis;

import java.util.ArrayList;
import java.util.TimerTask;

public class DeviceStatusTask extends TimerTask {

  ArrayList<DeviceStatus> devicesList;

  public void run() {

    devicesList = AmfisConnection.getInstance().getDeviceStatusList();

    for(int i = 0; i < devicesList.size(); ++i) {
      if(devicesList.get(i).isOnline) {
        devicesList.get(i).decrementTimeoutCounter();
      }
    }

  }

}
```

Listing B.13: Klasse *DeviceStatusTask*