Zuwendungsempfänger:	**Förderkennzeichen:**
Universität Koblenz - Landau	01AK049
Vorhabensbezeichnung: MoBiLo: Middlewareplattform zur Unterstützung mobiler Logistiksysteme unter Einsatz aktueller Middleware, die auf Internet- und Open-Source-Technologien basiert	
Laufzeit des Vorhabens: 01.11.2001 bis 30.04.2003	
Berichtszeitraum: 01.11.2001 bis 30.04.2003	

UNIVERSITÄT KOBLENZ · LANDAU

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

Projekt „MoBiLo“

Middlewareplattform

zur Unterstützung

mobiler Logistiksysteme

Abschlussbericht

Universität Koblenz-Landau

Campus Koblenz

Institut für Wirtschafts- und Verwaltungsinformatik

Arbeitsgruppe Prof. U. Frank

die mobilanten GmbH

Maria Trost 23

56070 Koblenz

die mobilanten

Inhaltsverzeichnis

Abbildungen

Abkürzungen

DBMS	Datenbank-Management-System
EJB	Enterprise Java Bean
J2EE	Java2 Enterprise Edition
J2ME	Java2 Micro Edition
JMX	Java Management Extension
KMU	Klein- und mittelständische Unternehmen
OMG	Object Management Group
SOAP	Simple Object Access Protocol
SMS	Short Message System
SNMP	Simple Network Management Protocol
XML	Extensible Markup Language
XML-RPC	XML Remote-Procedure-Call

1. Motivation und Zielsetzung

Unternehmen mit einem hohen Anteil an mobilen Dienstleistungen konnten bisher nur teilweise eine informationstechnische Unterstützung ihrer Geschäftsprozesse realisieren. Der technische Fortschritt im Bereich mobiler Geräte erlaubt nun auch die Entwicklung *mobiler Lösungen,* die eine Verkettung von bisher isolierten Teilschritten zu durchgängigen, medienbruchfreien Prozessen ermöglichen. Jedoch führt die hohe Dynamik des technischen Fortschritts mit seinen kurzen Innovationszyklen zu häufigen, technologiebedingten Anpassungen mobiler Anwendungen (z.B. durch die Einführung neuer Mobilfunkstandards wie GPRS und UMTS).

Eine *Middlewareplattform*[1] als Abstraktionsschicht erlaubt die flexible Adaption innovativer Technologien ohne Modifikationen auf Anwendungsebene. Dies ist mit wirtschaftlichen Vorteilen verbunden, wie z.B. verbessertem Investitionsschutz, Kostenreduktion, kürzeren Entwicklungszeiten und Produktivitätssteigerung. Die Nutzung von Middlewaretechnologien erhöht somit die Flexibilität und Wettbewerbsfähigkeit von Unternehmen.

Ziel des Vorhabens MoBiLo ist die Unterstützung mobiler Logistiksysteme[2] mit Hilfe einer zu konzipierenden Middlewareplattform.

Die hohe Varianz denkbarer Einsatzmöglichkeiten mobiler Lösungen legt eine Fokussierung nahe. Im Forschungsprojekt *FlottHIT*[3], das am Institut für Wirtschafts- und Verwaltungsinformatik der Universität Koblenz-Landau durchgeführt wurde, sind reale Geschäftsprozesse des Kundendienstes in klein- und mittelständischen Unternehmen (KMU) analysiert und Potenziale für eine mobile, informationstechnische Unterstützung aufgezeigt worden. Die dabei für Unternehmen identifizierten wesentlichen Aspekte sind in die Konzipierung der Middlewareplattform eingeflossen.

---

[1] Eine Middlewareplattform integriert heterogene Middlewaretechnologien und stellt sie Anwendungen einheitlich zur Verfügung.

[2] Ein mobiles Logistiksystem unterstützt die Durchführung von logistischen Prozessen unter Einsatz von mobilen Geräten.

[3] FlottHITT: Flottenmanagement im Handwerk durch integrierte Telematikdienste

2. Vorgehensmodell

Um den kurzen Innovationszyklen und der hohen Dynamik im Bereich mobiler Geräte und Technologien gerecht zu werden und daraus resultierende Projektrisiken zu minimieren, wird nach der in [JaBoRu99] vorgestellten Methode *Unified Process* vorgegangen.

Die drei Kernaussagen dieses Ansatzes lauten:

- UseCase-Zentrierung
- Architektur-Zentrierung
- Iteratives und inkrementelles Vorgehen

2.1. Anforderungsanalyse unter Einsatz von UseCase-Analysen

Die Anforderungsanalyse dient der Ermittlung funktionaler und nicht-funktionaler Anforderungen, die an ein mobiles Logistiksystem gestellt werden. Im Vordergrund der fachlichen Anforderungsanalyse stehen die Anwendungsfälle (UseCases). Ein *UseCase* repräsentiert eine Folge von Benutzer/Systeminteraktionen, die auf die Erreichung eines Ergebnisses ausgerichtet sind.

Um ein vollständiges Bild aller Benutzungsformen eines System ermitteln zu können, vor allem der geschäftskritischen Anwendungsfälle, ist ein bestimmtes Wissen über die Bedürfnisse der Anwender sowie der Ansätze, die auf die Optimierung existierender Geschäftsprozesse zielen, erforderlich. Zum Aufbau des für MoBiLo erforderlichen Domänenwissens ist neben Grundlagenliteratur (vgl. u.a. [Gud99]) insbesondere auf die Forschungsergebnisse des Vorhabens *FlottHITT* zurückgegriffen worden (vgl. [JuFr01], [JuHa01], [JuLa01]). Ein wesentlicher Teil der dem Projekt zugrundeliegenden UseCase-Modelle ist auf dieser Basis erstellt worden (siehe 3.2).

Dabei standen zu Anfang nicht die Anforderungen der Middlewareplattform, sondern die Klasse von potenziellen Anwendungen im Fokus.

Dies hat mehrere Gründe:

- Gewährleistung der Nachvollziehbarkeit konkreter Anforderungen. Anforderungen an eine Middleware sind oft fachlich motiviert, aufgrund der hohen Generalität und der Anordnung auf tendenziell tiefen, technischen Schichten aber sehr abstrakt. Ohne konkreten fachlichen Bezug besteht die Gefahr, Aspekte zu betonen, die von den Anwendern der Middleware, d.h. den Anwendungsentwicklern, nicht benötigt werden oder dass Kernanforderungen vernachlässigt werden.
- Das Ziel des Projektes besteht in der Entwicklung einer Middleware, die als Instrument die Entwicklung mobiler Logistiksysteme unterstützt und verschiedene, existierende Middlewaretechnologien vereint. Somit besteht a priori eine starke konzeptuelle Nähe zu den Anwendungen, die mit der Middleware realisiert werden sollen. Der Ansatz lässt sich am ehesten als Frameworkentwicklung begreifen, die Komponenten in einem Architekturrahmen zusammenfasst.

2.2. Architekturzentrierter Entwurf

Die Architektur eines Softwaresystems (*Softwarearchitektur*) beschreibt diejenigen Elemente, die zum Verständnis und der Entwicklung des Systems von besonderer Bedeutung sind. Bei den Elementen kann es sich bspw. um Subsysteme, Komponenten, Schnittstellen, Klassen oder Interaktionen handeln. Den unterschiedlichen Sichtweisen, die aus den verschiedenen Rollen der Projektbeteiligten resultieren, wird in der Softwarearchitektur durch unterschiedliche Perspektiven, die jeweils bestimmte Modellelemente enthalten, Rechnung getragen.

Ausgangspunkt der Erstellung der Softwarearchitektur sind die identifizierten UseCases, die durch diese Architektur unterstützt und möglichst effizient umgesetzt werden können sollen. In den ersten Schritten werden zunächst die UseCases berücksichtigt, die eine besondere Relevanz für die Architektur aufweisen. Weiterhin haben die verwendeten Technologien und Standards einen starken Einfluss auf die Architektur.

Von besonderer Bedeutung bei der Architekturkonzeption sind sog. *Entwurfsmuster*, die als „best practice solution to a common recurring problem" [FlMa02] die Erfahrung und das Wissen um bewährte Lösungskonzepte nutzbar machen und sich positiv auf die Qualität des Architekturentwurfs auswirken.

2.3. Iterative und inkrementelle Vorgehensweise

Bei dieser Vorgehensweise wird in kleinen Schritten, den sog. Iterationen vorangegangen. Jede *Iteration* besteht aus den Phasen des klassischen Wasserfallmodells, d.h. der Planung, Analyse, Design, Implementierung und Test. Hinzu kommt eine Auswertung der Iteration, deren Ergebnisse dann bei der Planung der folgenden Iteration berücksichtigt werden.

Aufgrund des begrenzten Umfangs jeder einzelnen Iteration wird die technische und organisatorische Komplexität gegenüber herkömmlichen Projektmanagementansätzen reduziert. Dies führt zu verbesserter Planbarkeit und zu erleichterter Durchführung des Projektes. Weiterhin können frühzeitig neue technologische Entwicklungen antizipiert werden. Die Vorgehensweise im Rahmen von MoBiLo zeigt *Abbildung 1*.

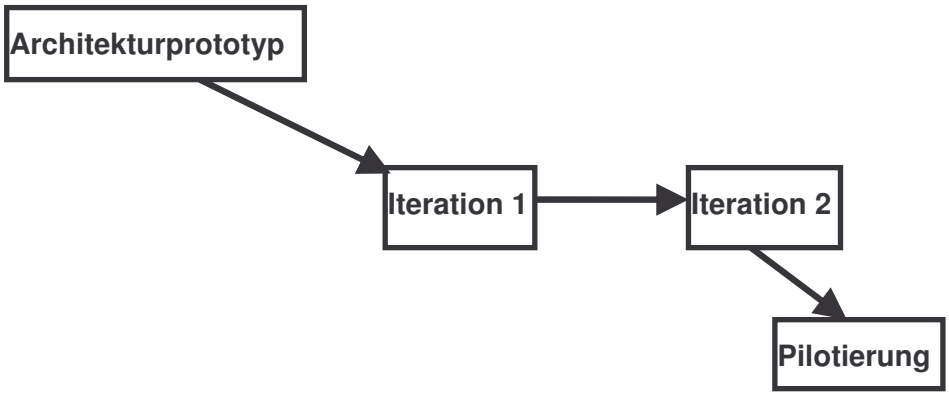

Abbildung 1: MoBiLo-Vorgehensmodell

Ein Ziel des Architekturprototyps ist die Überprüfung wesentlicher Architekturentscheidungen, welche die zentralen technischen Herausforderungen betreffen. Somit kann frühzeitig die Tragfähigkeit eines gewählten Ansatzes überprüft werden. Dabei werden charakteristische UseCases ausgewählt, deren softwaretechnische Umsetzung durch die Middlewareplattform unterstützt werden muss.

Eine abschließende Auswertung des Prototyps liefert wichtige Informationen für die folgenden Schritte. In den nachfolgenden Iterationen wird dann sukzessive der Funktionsumfang erweitert, bis die Zielvorgaben erreicht sind.

Die Pilotierung von Anwendungsprototypen basierend auf der MoBiLo-Plattform stellt die Einsetzbarkeit der Middleware sicher und ermöglicht die Identifikation von Verbesserungs- und Ausbaupotenzialen.

3. Wissenschaftliche und technische Ergebnisse

Im nachfolgenden werden die erzielten Forschungsergebnisse des Vorhabens MoBiLo beschrieben.

3.1. Verwendete Technologien / Produkte

Basis der MoBiLo-Middleware ist die Java2-Plattform, die schon umfangreiche Middleware-funktionalität beinhaltet und die aufgrund der Unabhängigkeit von dem zugrundeliegenden Betriebssystem ein hohes Maß an Portierbarkeit erlaubt. Zum Einsatz kommt serverseitig die *Enterprise Edition* der Java2-Plattform (J2EE)[4] und auf der Clientseite die *Micro Edition* (J2ME)[5] bzw. *PersonalJava* [6]. Der Einsatz von Java schafft insbesondere auf der Seite der mobilen Geräte eine Unabhängigkeit von den zugrundeliegenden Betriebssystemen, so dass diese für das jeweilige Einsatzgebiet ausgewählt werden können.

Der Java2-Platform wurde gegenüber einer CORBA-Basis der Vorzug gegeben, da diese nicht auf allen Betriebssystemen bzw. Geräteplattformen verfügbar ist, speziell auf mobilen Endgeräten wie bspw. Handys, und somit eine Abhängigkeit von einzelnen Betriebssystemen entstanden wäre.

Zentrale Bedeutung kommt der *Extensible Markup Language* (XML)[7] zu, die zur Beschreibung des Datenaustauschs eingesetzt wird. Dadurch können Client und Server stärker entkoppelt werden.

Bei der Auswahl der verwendeten Produkte und Komponenten wurde weitestgehend erfolg-versprechende Open-Source-Software verwendet. Folgende Produkte bzw. Komponenten werden derzeit im Rahmen des Vorhabens eingesetzt:

[4] http://java.sun.com/j2ee/

[5] http://java.sun.com/j2me/

[6] http://java.sun.com/products/personaljava/

[7] [BrMc00], [GoPr99], http://www.w3.org/TR/2000/REC-xml-20001006

Serverseitig:

- Betriebssystem: Linux (Open Source)
- DBMS: MySQL, PostgreSQL (beides Open Source)[8]
- J2EE-konformer Application Server: JBOSS (Open Source)[9]
- Regelmaschine: JESS (kostenlos für Forschungseinsatz)[10]
- Webframework Webwork: (Opens Source)[11]
- McProcessEngine: die mobilanten GmbH (kostenlos für Forschungseinsatz)
- McServiceBroker: die mobilanten GmbH (kostenlos für Forschungseinsatz)

Clientseitig:

- Betriebssysteme: Symbian OS, Pocket PC, Linux
- GUI-Bibliothek: Zaval's LwVCL[12], Eclipse SWT[13] (beides Open Source)
- McServiceBroker: die mobilanten GmbH (kostenlos für Forschungseinsatz)

Werkzeuge:

- Make-Tool: ANT (Open Source)[14]
- JavaDoc-Engine: XDoclet (Open Source) [15]
- Entwicklungsumgebung: Eclipse (Open Source)[16]

---

[8] http://www.mysql.com/ bzw. http://www.de.postgresql.org/

[9] http://www.jboss.org/

[10] http://herzberg.ca.sandia.gov/jess/

[11] http://www.opensymphony.com/webwork/

[12] http://www.zaval.org

[13] http://www.eclipse.org

[14] http://jakarta.apache.org/ant/

[15] http://xdoclet.sourceforge.net/

[16] http://www.eclipse.org

Aufgrund der kurzen Innovationszyklen im Technologiebereich werden in der verbleibenden Projektlaufzeit weitere Technologien bzw. Produkte verfügbar sein. Die iterative Vorgehensweise ermöglicht, neue Technologien bei entsprechender Eignung in die Plattform aufzunehmen oder bestehende zu substituieren.

3.2. UseCase-Modelle

Wie in Abschnitt 2.1 beschrieben, ist die Anforderungsanalyse mit Hilfe von UsesCases erfolgt. Die in der folgenden Abbildung enthaltenen UseCases sind bei der Konzeption der Plattform zugrundegelegt worden.

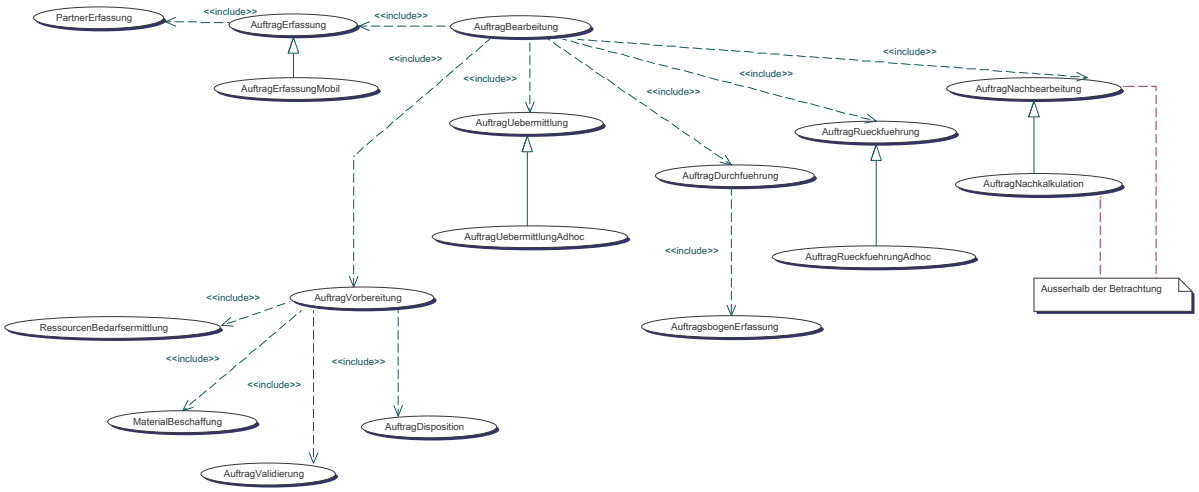

Abbildung 2: UseCase-Modell

3.3. MoBiLo-Architektur

Die Architektur der MoBiLo-Plattform besteht aus mehreren Schichten und besitzt einen komponentenbasierten Aufbau (vgl. [AhAtHa01], [SiStJo02]). Einen generellen Überblick über die Architektur zeigt (

Abbildung 3: MoBiLo-Architektur).

Bei der Konzeption der Plattform wurden den Aspekten Offenheit und Anpassbarkeit besondere Bedeutung zugemessen, um die Wiederverwendbarkeit der Middlewarekomponenten zu erhöhen. Aus diesem Grund sind geeignete Entwurfsmuster ausgewählt worden, die sich in besonderem Maße auf die zuvor genannten Aspekte auswirken (vgl. [FlMa02], [GaHeJoVl95]).

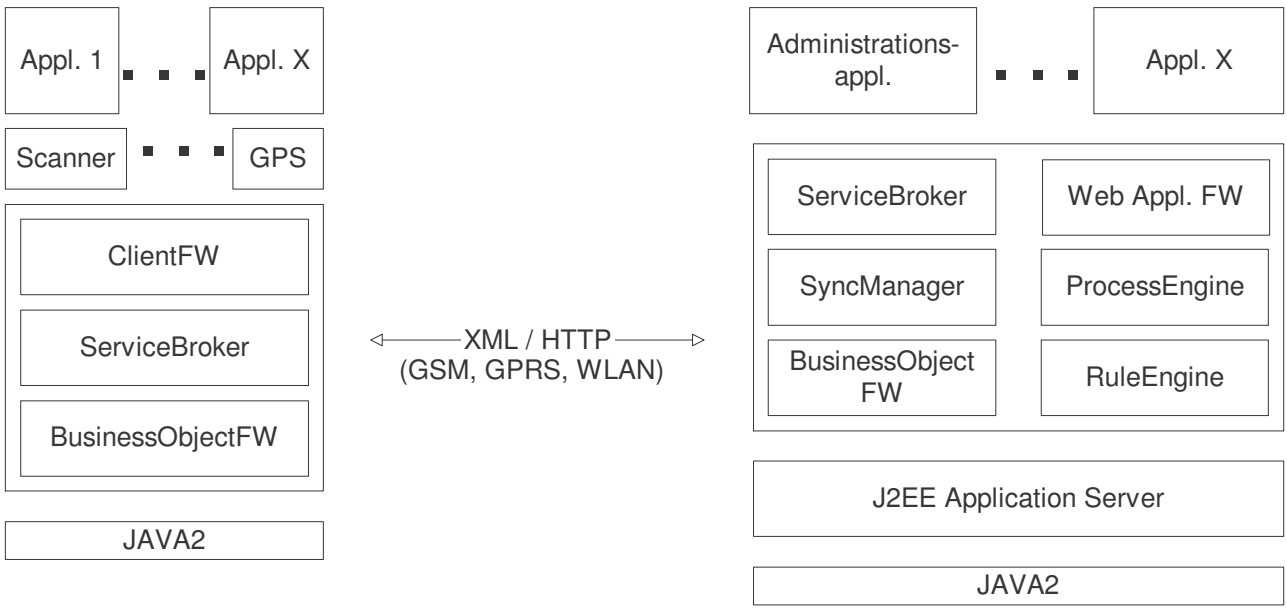

Abbildung 3: MoBiLo-Architektur

3.4. Berücksichtigung von Kontextinformationen

Die Softwareentwicklung für mobile Geräte ist mit einer Reihe von Herausforderungen verbunden, die u.a. aus deren physikalischen Ausmaßen, der Geräteausstattung und dem Akku-Betrieb für den mobilen Einsatz resultieren (vgl. [FrJu01]). Zunächst seien hier die im Vergleich zu herkömmlichen Desktop-Computern geringe CPU-Leistung (33 - 400MHz) und der kleine Hauptspeicher (max. 64MB) zu nennen. Überdies fehlt aktuellen Geräten i.d.R. auch ein persistenter Datenspeicher. Nicht zuletzt stellt die eingeschränkte Benutzerschnittstelle einen wesentlichen Faktor dar. Es existiert i.d.R. keine vollwertige Tastatur und auch das Display ist im Vergleich zu aktuellen Computer-Monitoren/-Displays in seiner Größe deutlich reduziert. Neben den zuvor genannten statischen Aspekten müssen bei der Anwendungsentwicklung auch dynamische Aspekte berücksichtigt werden. Dazu zählen u.a. die schwankende Netzwerkbandbreite bis hin zum völligen Verlust der Kommunikationsverbindung, abnehmende Batterieleistung oder Positionsänderungen.

Beide Aspekte bilden zusammen den *Ausführungskontext*, an den sich die jeweilige mobile Anwendung anpassen können sollte. Traditionelle Middlewareansätze kapseln aber i.d.R. die zugrundeliegenden Technologien vollständig vor Benutzern und Entwicklern. Somit werden keine Möglichkeiten bereitgestellt, anwendungsseitig den aktuellen Ausführungskontext abzufragen oder sogar zu (re)konfigurieren, um die Middleware in ihrer Effizienz und

Leistungsfähigkeit zu steigern. Mit *Reflection* steht ein Konzept bereit, das die Selbstprüfung (*Introspection*) und die eigene Anpassung (*Adaption*) unterstützt [vgl. CaBlMa02].

In der MoBiLo-Middleware ist das Reflection-Konzept zur Umsetzung der Berücksichtigung des Ausführungskontextes (*Context Awareness*) verwendet worden. Nachfolgend wird der verwendete Ansatz beschrieben.

Der Ausführungskontext wird durch statische (`StaticAspect`) und dynamische Aspekte (`DynamicAspect`) beschrieben. Die Unterscheidung ist in Relation zur Laufzeit der Anwendung zu sehen. So ist ein statischer Aspekt zumindest für die Nutzungsdauer einer einzelnen Applikation unveränderlich, während sich ein dynamischer Aspekt zur Anwendungslaufzeit stetig verändern kann. Zu den statischen Aspekten zählen bspw. `Device`, `Application` oder `UserProfile`. Dynamische Aspekte sind bspw. `Position`, `Network` oder `Memory`. Sie werden durch spezielle Sensoren (`AspectMonitor`) überwacht und i.d.R. periodisch aktualisiert. (siehe *Abbildung 4: Aspekte*).

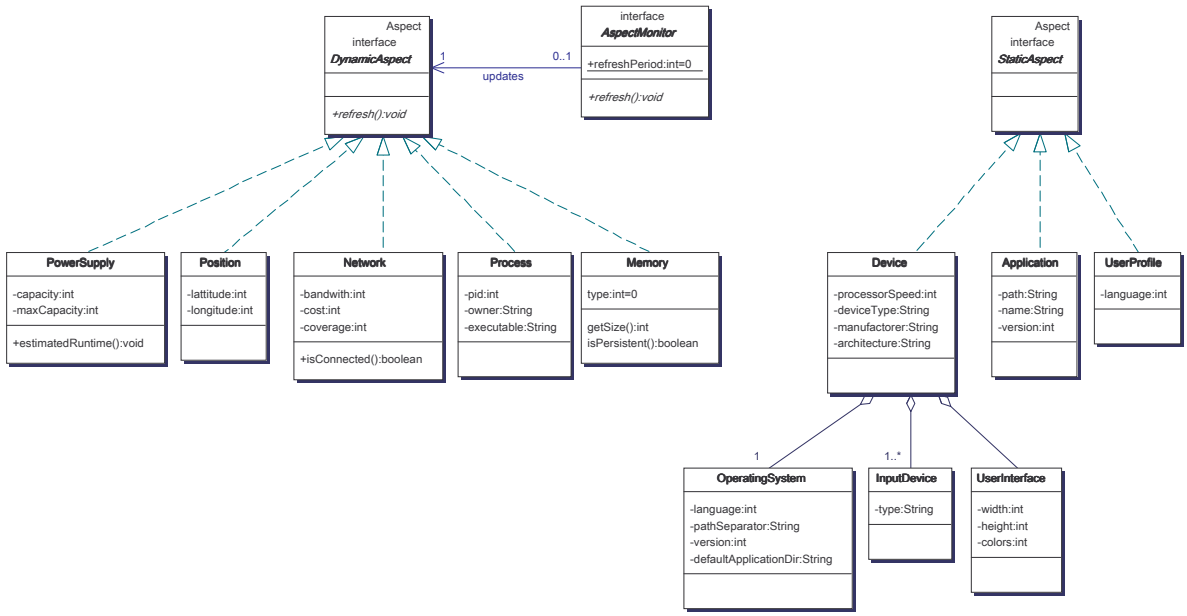

Abbildung 4: Aspekte

Die generellen Typeigenschaften eines Aspektes werden durch einen `AspectType` beschrieben. (*Abbildung 5: Aspekttypen*).

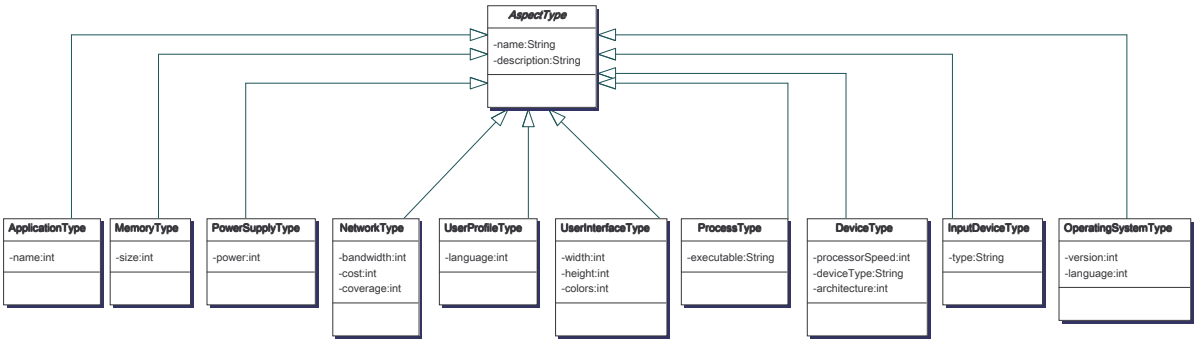

Abbildung 5: Aspekttypen

Aspekte können in einer `Configuration` zusammengefasst werden. Die `SystemConfiguration` repräsentiert die jeweils aktuelle Systemumgebung.

Eine `Condition` formuliert Bedingungen zwischen Aspekten und –typen. So könnte bspw. eine Bedingung definiert werden, die eine bestimmte Displaygröße und eine bestimmte verfügbare Netzbandbreite erfordert. Eine Menge von Bedingungen können zu einer `Policy` zusammengefasst werden, die in mehreren Profilen enthalten sein können. Ein `Profile` setzt einen einzelnen Dienst (`Service`) in Beziehung zu einer oder auch mehreren Policies. Soll nun ein bestimmter Dienst aufgerufen werden, so wird zunächst geprüft, ob für diesen Dienst ein gültiges Profil existiert (siehe *Abbildung 6: Profilermittlung*). Falls mehrere gültige Profile existieren, wird die Auswahlentscheidung an eine Strategieklasse (`ProfileSelectionStrategy`) delegiert, die dann den konkreten Selektionsalgorithmus implementiert. In ein solches Verfahren können nicht-funktionale Anforderungen einfließen, bspw. maximale Energieeinsparung oder größtmögliche Funktionsvielfalt der Middleware. Umgesetzt werden diese Algorithmen in Konkretisierungen der `ProfileSelectionStrategy`.

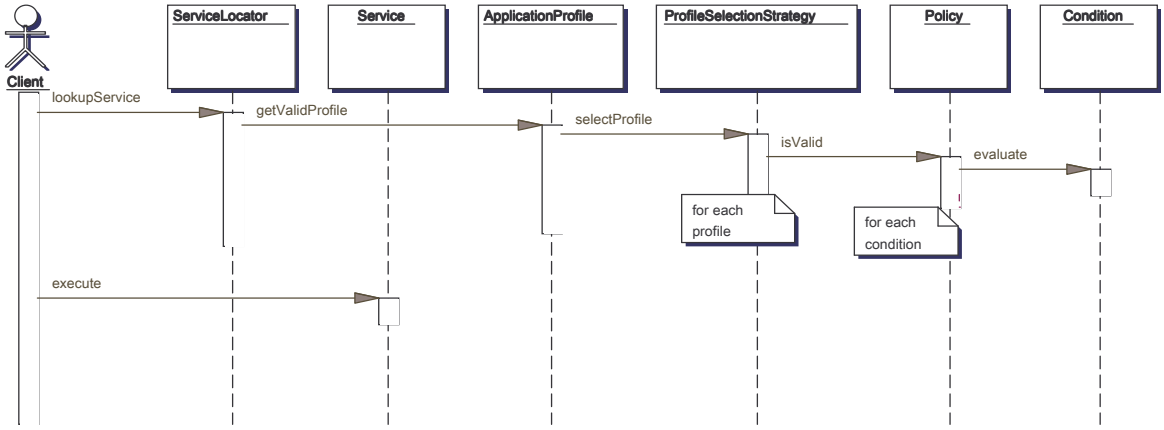

Abbildung 6: Profilermittlung

Die Überprüfung der definierten Profile stellt sicher, dass ein Dienst die für seine erfolgreiche Ausführung erforderliche Systemumgebung vorfindet oder sich ggfs. an diese abweichende Umgebung anpassen kann. So könnte sich ein Dienst zur Darstellung von Bildern an das Gerätedisplay anpassen, in dem dieser die Farbtiefe des Bildes auf die zur Verfügung stehenden Farben abstimmt und somit den Speicherbedarf reduziert. Die für die jeweilige Anwendung definierten Profile werden in einem `ApplicationProfile` zusammengefasst (siehe *Abbildung 7: Berücksichtigung von Kontextinformationen*).

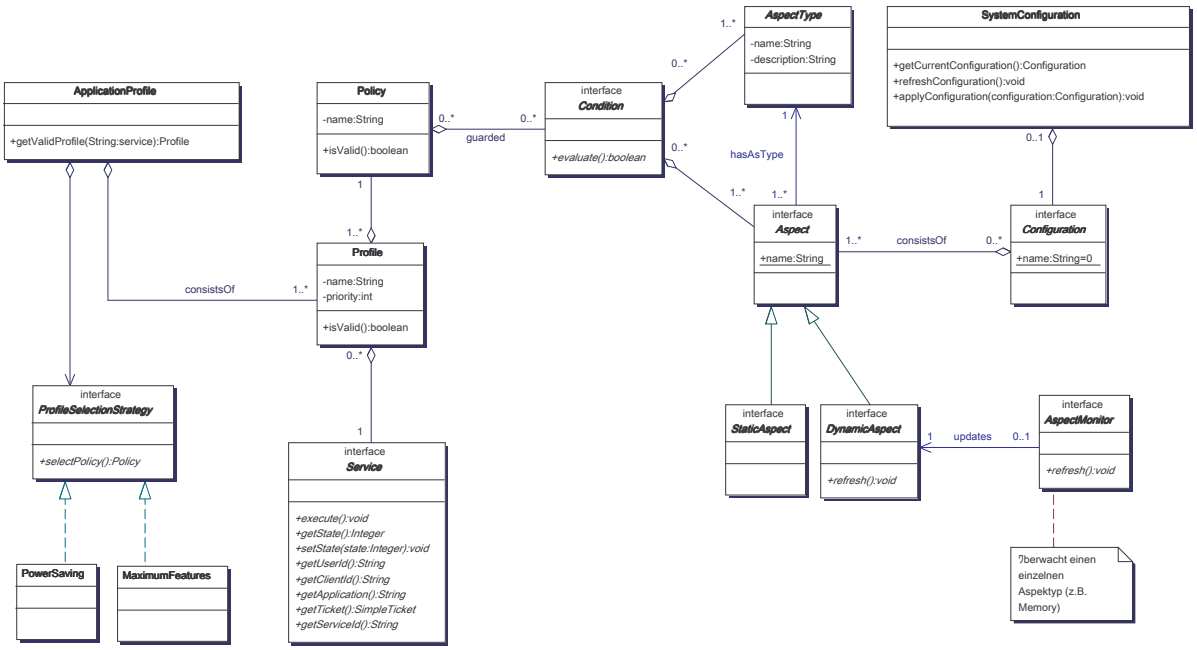

Abbildung 7: Berücksichtigung von Kontextinformationen

Mobiler Programmcode

Das in zuvor beschriebene Konzept der Berücksichtigung von Kontextinformationen erlaubt die Anpassung der Middleware an vorhergesehene Änderungen des Ausführungskontextes. Dennoch können auch Situationen auftreten, die zum Entwurfszeitpunkt nicht vorgesehen worden sind und an die sich die Middleware daher nicht anpassen kann. Für diese Fälle stellt die Technik des *Mobilen Programmcodes*[17] einen geeigneten Ansatz dar (vgl. [CaBlZa01]). Stellt die Middleware einen nicht antizipierten Ausführungskontext fest, so wird die erforderliche Funktionalität – Programmcode – von einer vertrauenswürdigen Stelle bezogen und in die Middleware aufgenommen. Im Falle von MoBiLo bedeutet dies, das einzelne Dienste (Services) nachgeladen werden können (siehe *Abbildung 8: Nachladen von Mobile Code*).

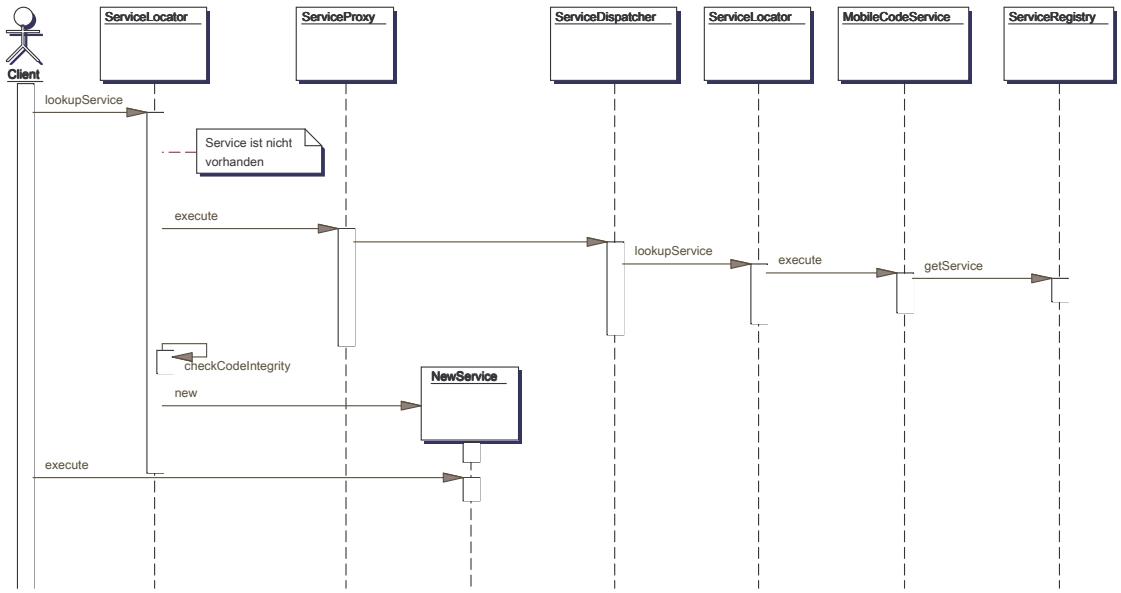

Abbildung 8: Nachladen von Mobile Code

Dabei sind jedoch besondere Herausforderungen zu beachten, die nachfolgend erläutert werden. Es muss zunächst sichergestellt werden, dass Programmcode nur von zulässigen Stellen (Server) nachgeladen werden kann. Dazu müssen diese eindeutig identifizierbar sein

[17] Prominente Vertreter sind Agenten-basierte Systeme.

(Authentifizierung). Der Programmcode sollte signiert sein, damit der der Client auch Authentizität und Integrität des Codes überprüfen kann.

Weiterhin muss die Middleware sicherstellen, dass mobiler Programmcode keine bereits vorhandene Funktionalität beeinträchtigt, da andernfalls Anwendungen in ihrer Funktionsfähigkeit beeinträchtigt werden könnten.

Derzeit befindet sich das zuvor beschriebene Konzept zur Berücksichtigung von Kontextinformationen sowie des Mobile Codes in der Umsetzung.

3.5. Konzipierte und realisierte Middlewarekomponenten

Durch den Kooperationspartner „die mobilanten GmbH" sind bereits ausgearbeitete Konzepte bzw. realisierte Komponenten in das Vorhaben eingebracht worden, speziell zur regelbasierten Vorgangssteuerung, zur Definition und Ausführung von Diensten (Serviceframework) und zur Replikation.

3.5.1. BusinessObject-Framework

Bei der Konzeption der Middlewareplattform wurde der Unterstützung unterschiedlicher Geräteplattformen besondere Bedeutung zugemessen. Insbesondere hinsichtlich der zur Verfügung stehenden Persistenzmechanismen unterscheiden sich die Geräteplattformen erheblich. Um eine hohe Wiederverwendbarkeit bzw. Portierbarkeit der MoBiLo-Plattform zu ermöglichen, ist ein BusinessObject-Framework konzipiert worden, das eine geeignete Abstraktion bietet (siehe *Abbildung 9: BusinessObject-Framework*).

Basis sind `BusinessObjects`, fachlich motivierte Objekte, die auf einer generischen Persistenzschicht aufsetzen (`DataAccessObj`). Der konkrete Persistenzmechanismus wird über spezielle Adaptoren integriert und ist deklarativ per XML konfigurierbar. Z.Zt. ist die Persistenz serverseitig über EJB-Technologie (`EJBAccessObj`) und clientseitig über Dateien (`SimpleAccessObj`) möglich. In den nächsten Iterationen werden weitere Adapter für mobile relationale bzw. objektorientierte Datenbanken hinzukommen. Um die Verwendung von EJBs zu vereinfachen, ist der `ClientContext` eingeführt worden, der das Erzeugen und Auffinden von EJBs ermöglicht.

Die Erzeugung bzw. das Suchen der `BusinessObjects` erfolgt über eine spezielle Factory[18] (`BusinessObjectFactory`), um auch dort eine Kapselung der Persistenz zu erreichen.

Jedes `BusinessObject` besitzt einen systemweit eindeutigen Identifier, der derzeit von dem verwendeten DBMS (`DatabaseIDGenerator`) erzeugt wird. Da es sich bei MoBiLo um eine verteilte Middleware handelt, müssen Maßnahmen zur Sicherung der Datenkonsistenz eingesetzt werden. Dazu besitzt jedes `BusinessObject` einen Versionsstempel, um Konflikte erkennen zu können. Das Verfahren zum Abgleich der verteilten Daten ist in Abschnitt 3.5.4 beschrieben.

[18] vgl. hierzu [GaHeJoVl95]

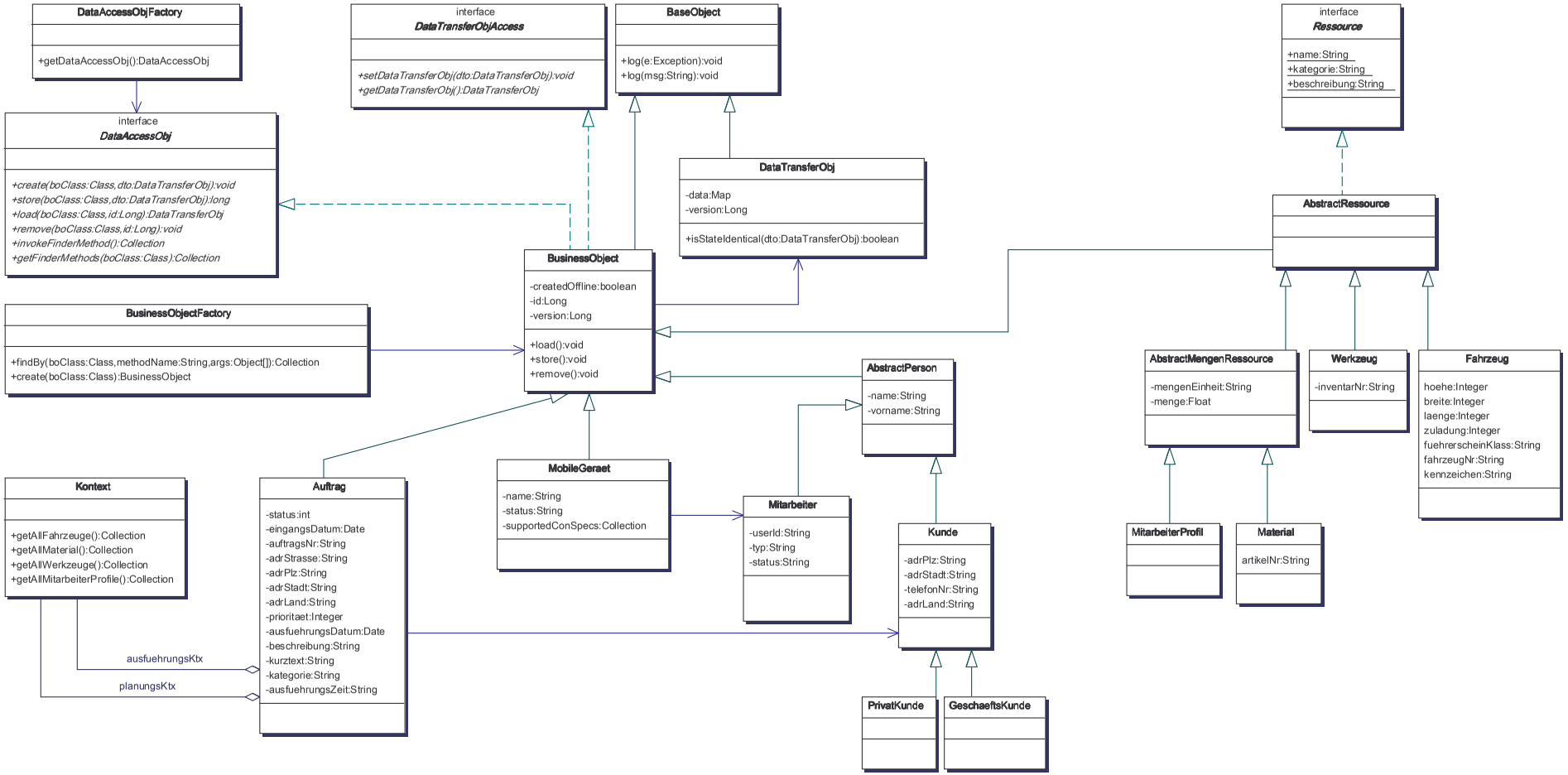

Abbildung 9: BusinessObject-Framework

In einer mehrschichtigen und verteilten Architektur müssen zum Zugriff auf die Persistenzschicht die dazwischenliegenden Schichten und ggfs. Kommunikationsverbindungen genutzt werden. Daraus resultiert ein Overhead für jeden einzelnen Methodenaufruf, da dieser einen Remote-Aufruf, d.h. außerhalb des lokalen Adressraums darstellt. Eine Minimierung derartiger Zugriffe ist daher erforderlich. Mit `DataTransferObj` steht ein generischer Transportcontainer für den Datenaustausch zw. den Schichten zur Verfügung. Jedes `BusinessObject` kann den eigenen Zustand als Transportcontainer zur Verfügung stellen bzw. den eigenen Zustand durch einen solchen Container ändern (`DataTransferObjAccess`). Somit ist zum einen jeweils nur ein Methodenaufruf erforderlich und zum anderen einheitlicher Zugriff auf alle `BusinessObject` möglich.

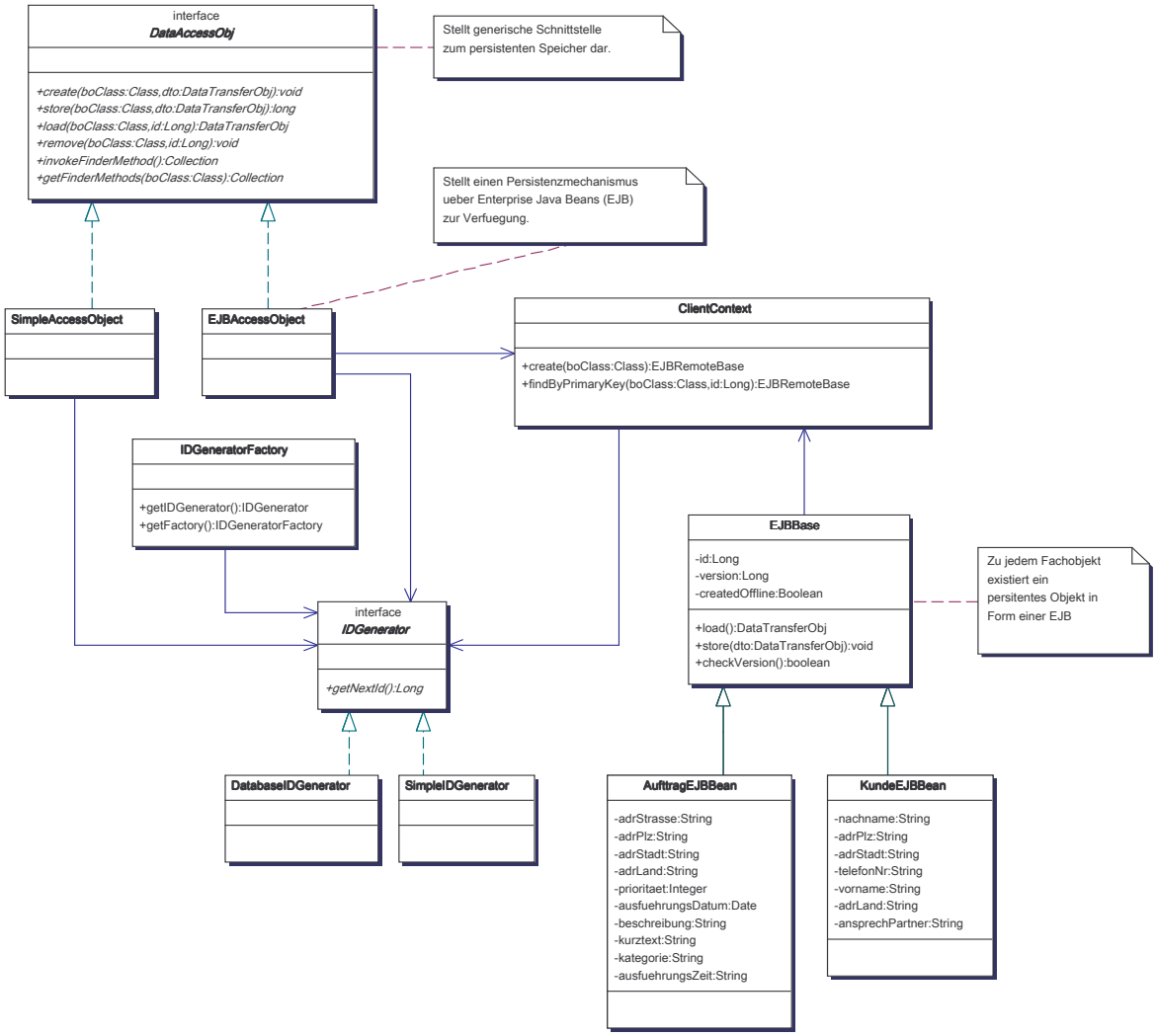

Abbildung 10: Persistenz

3.5.2. McServiceBroker

Die von dem Kooperationspartner „die mobilanten GmbH" in das Vorhaben eingebrachte und in Eigenleistung weiterentwickelte Komponente *McServiceBroker* stellt ein Serviceframework zur Verfügung, das die Definition, die Lokalisierung, die Verwaltung sowie den verteilungstransparenten Aufruf von Diensten auf mobilen Geräten ermöglicht (siehe *Abbildung 11: McServiceBroker – Service*).

Zur einfacheren Nutzung von Diensten werden über einen Generierungsmechanismus spezielle Wrapper-Klassen, sog. `ServiceComponents`, erzeugt. Dazu wird das JavaDoc-basierte OpenSource-Werkzeug *XDoclet* eingesetzt.

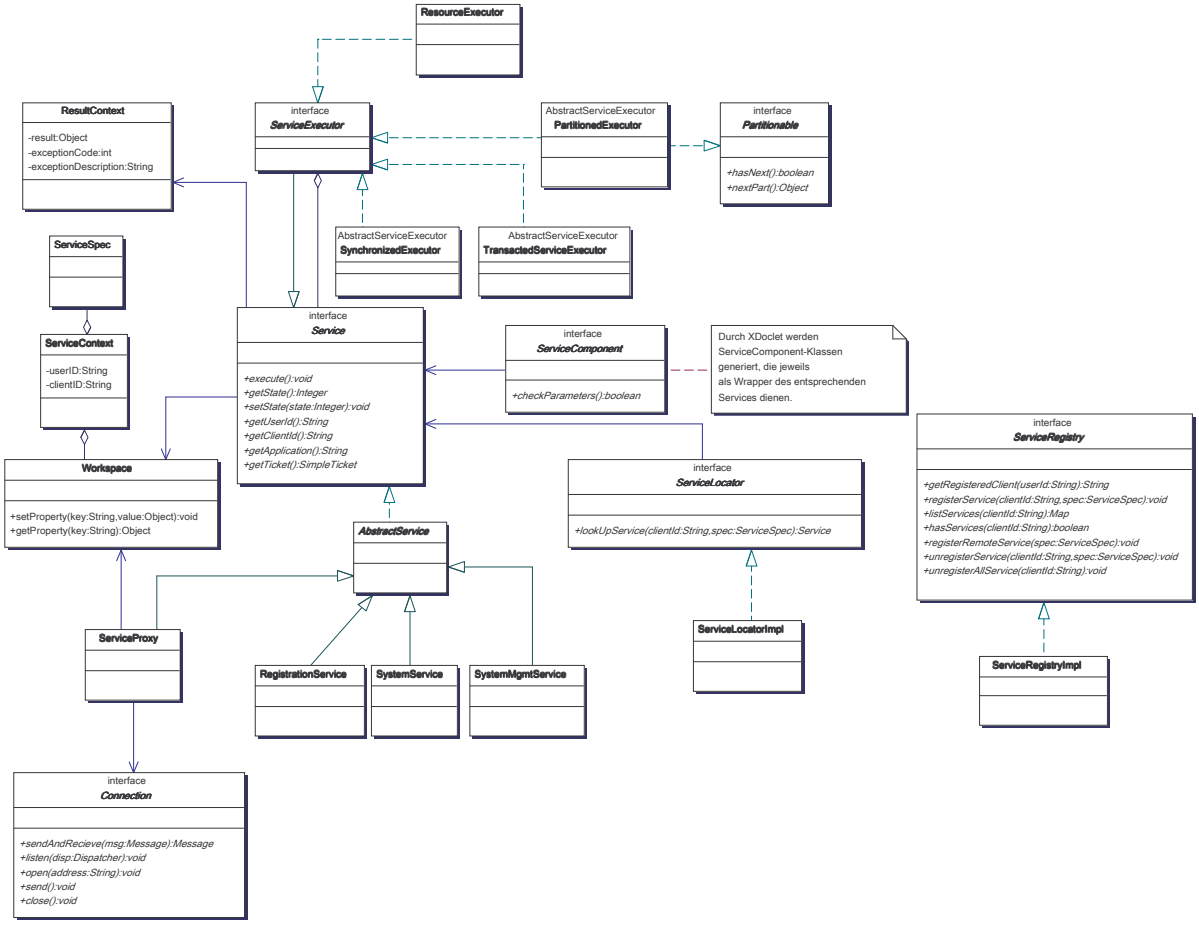

Abbildung 11: McServiceBroker – Service

Zur Datenkommunikation wird XML eingesetzt, d.h. alle Nachrichten werden über einen eigenen, effizienten Serialisierungsalgorithmus in XML-Nachrichten überführt und über einen eigenen Verbindungsmechanismus übertragen. Es wurde auch der Einsatz des XML-basierten SOAP-Protokolls[19] sowie XML-RPC[20] untersucht, jedoch aufgrund des Overheads hinsichtlich des Datenvolumens und der Verarbeitungskomplexität sowie der z.T. fehlenden Verfügbarkeit essentieller Technologien (z.B. geeignete XML-Parser) auf mobilen Plattformen verworfen. Da die MoBiLo-Plattform auch leistungsschwächere mobile Endgeräte wie bspw. Handys unterstützen soll, wurde somit der Entwurf eines effizienten Protokolls erforderlich (siehe *Abbildung 12: McServiceBroker - XML*).

Die zugrundeliegende Idee ist es, ein systemweites Typensystem (`ClassRegistry`) einzusetzen. Dieses beschreibt die Zuordnung von Klassen und numerischen Typkonstanten. Die zu übertragenden XML-Nachrichten enthalten nur die Typkonstanten und nicht eine serialisierte Form oder eine vollständige XML-Repräsentation der entsprechenden Klasse. Dadurch kann zum einen das Datenvolumen minimiert werden, und zum anderen werden mögliche Deserialisierungsprobleme vermieden (z.B. aufgrund unterschiedlicher Java-Versionen auf den beteiligten Geräten).

Zur Transformation nach (`Object2XmlMapper`) bzw. von XML (`Xml2ObjectMapper`) werden spezielle Mapper angeboten, z.B. für Collections oder für Klassen, die den Transportcontainer-Zugriff `DataTransferObjAccess` unterstützen. Der Zustand des zu übertragenden Objektes wird über eine Menge von `FieldDescriptor` beschrieben, wobei ein `FieldDesciptor` ein einzelnes Attribut beschreibt (Typ, Wert,…). Die Instanzierung eines Mapper erfolgt über eine Factory (`ObjectMapperFactory`).

Aus Optimierungsgründen ist neben der zuvor beschriebenen XML-Serialisierung auch eine Bytecode-Serialisierung auf Basis der Java2-Externalisierung eingesetzt worden. Diese hat gegenüber der XML-Serialisierung den Vorteil eines deutlich reduzierten Speicherbedarfs sowie einer wesentlich höheren Ausführungsgeschwindigkeit auf der Clientseite. Demgegenüber steht jedoch der Nachteil einer homogenen Java-Umgebung auf Client und Server. Der zu verwendende Serialisierungsmechanismus kann konfiguriert werden.

---

[19] Simple Object Access Protocol: http://www.w3.org/TR/SOAP/

[20] XML-basierter Remote Procedure Call: http://www.xmlrpc.com

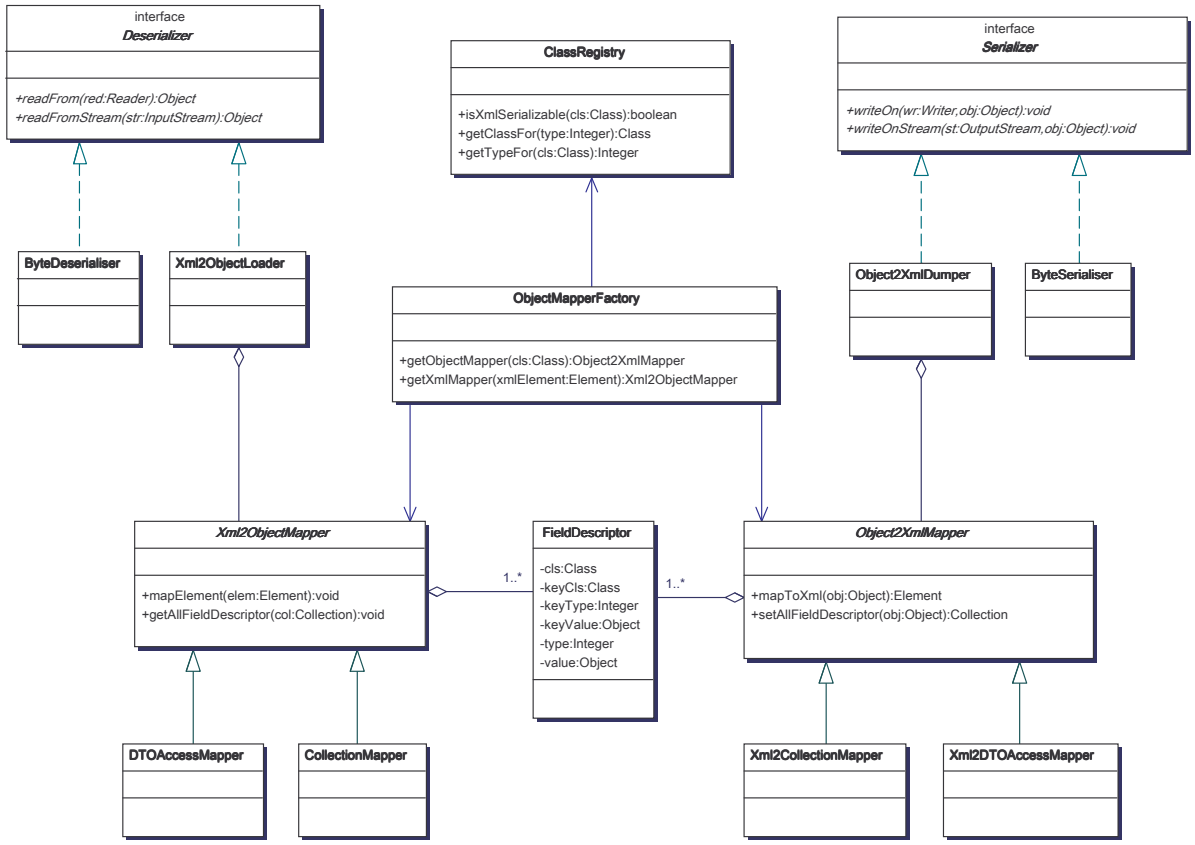

Abbildung 12: McServiceBroker - XML

Zur Reduzierung der Komplexität sowie zur Verbesserung der Adaptionsfähigkeit an neue Übertragungstechnologien (z.B. UMTS), werden diese durch ein generisches Verbindungsmanagement ersetzt. Dies ermöglicht derzeit die transparente Nutzung von GSM, GPRS oder WLAN (siehe *Abbildung 13: McServiceBroker - Connection*).

Generell wird zwischen Client- und Serververbindungen unterschieden, da letztere aktiv auf Verbindungsanfragen warten und daher in eigenen Threads ausgeführt werden müssen. Verwaltet werden die Verbindungen von dem `ConnectionManager`. Die für den Verbindungsaufbau erforderlich Informationen werden in einer `ConnectionSpec` zusammengefasst. Für jede Verbindung kann ein `Dispatcher` angegeben werden, der die Verarbeitung empfangener Nachrichten, d.h. `Messages`, übernimmt. Für die unterschiedlichen Nachrichtentypen (z.B. Service, Ereignis,…) existieren spezialisierte `Dispatcher`.

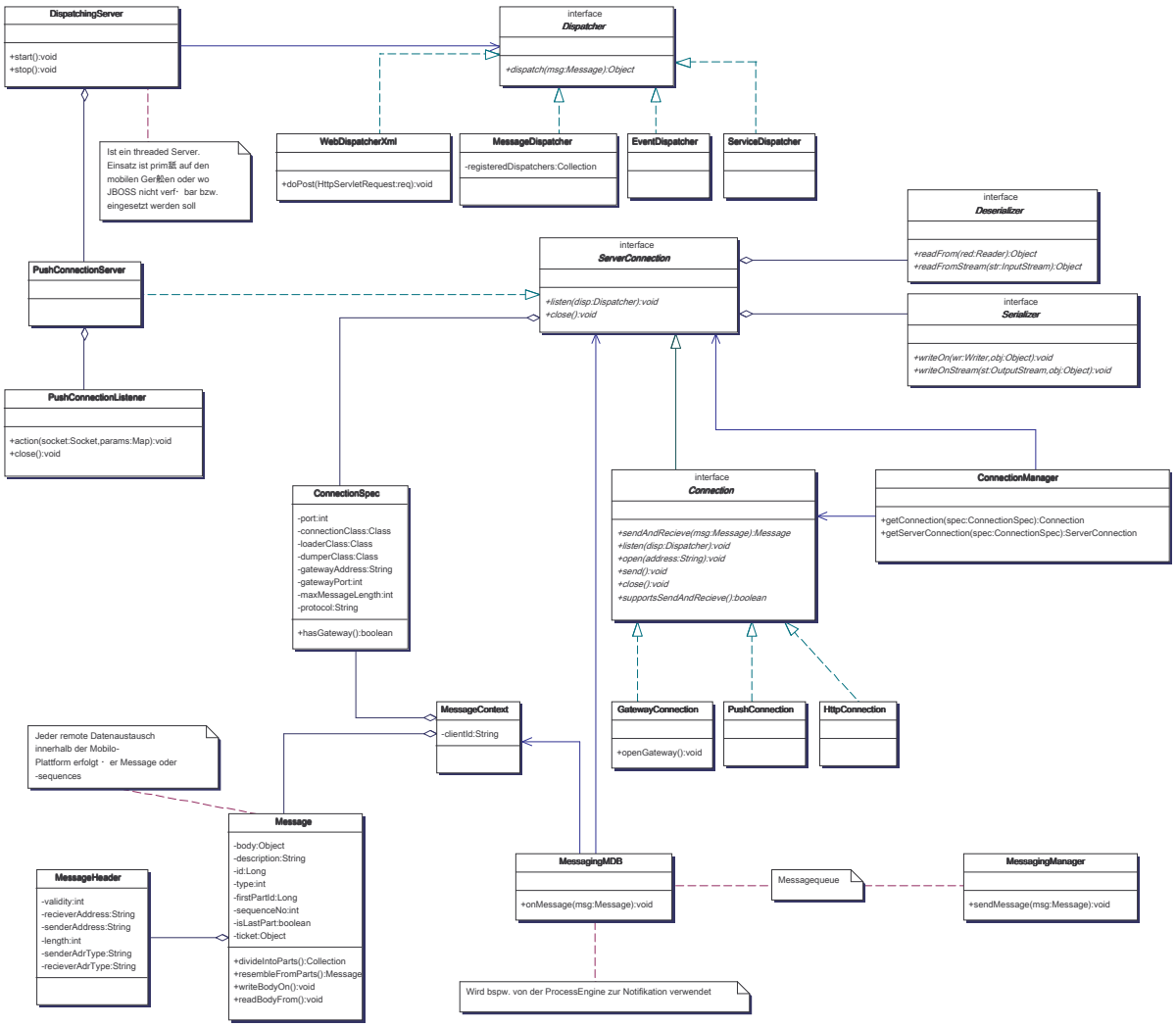

Abbildung 13: McServiceBroker - Connection

Die asynchrone Notifikation von mobilen Geräten erfolgt über die Klasse `MessagingMDB`, die Notifikationsnachrichten aus einer Messagequeue ausliest (siehe *Abbildung 14: Messaging*). In der Queue liegen die Nachrichten als sog. `MessageContexts` vor, die neben der eigentlichen Nachricht auch die erforderlichen Verbindungsinformationen enthalten. Die `MessagingMDB` öffnet dann über den `ConnectionManager` eine Verbindung und verschickt die Nachricht. Kann keine Verbindung aufgebaut werden, weil sich bspw. der mobile Teilnehmer in einem Funkloch befindet, wird der Verbindungsaufbau periodisch weiterverfolgt, bis der Teilnehmer oder ein Zeitlimit erreicht worden ist.

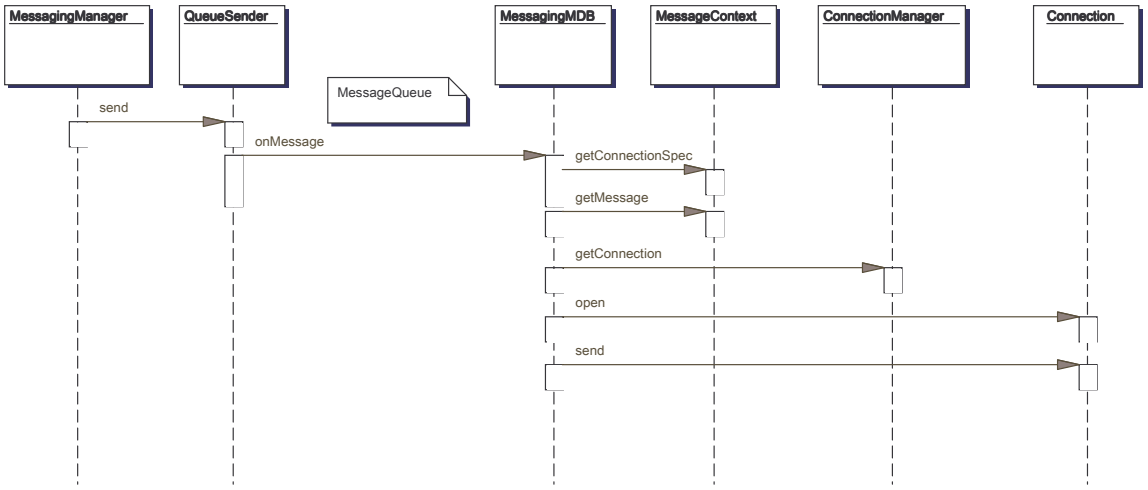

Abbildung 14: Messaging

Mobile Endgeräte registrieren sich mit ihren verfügbaren Diensten (z.B. Lokalisierbarkeit durch GPS-Modul, Barcodescanner etc.) bei dem Middlewareserver, welcher die angemeldeten Geräte mit ihren Diensten verwaltet. Zur Laufzeit ist dann das Aktivieren bzw. Deaktivieren und Konfigurieren von Diensten auf den mobilen Geräten durch den Middlewareserver möglich (siehe *Abbildung 15: McServiceBroker - Registrierung*).

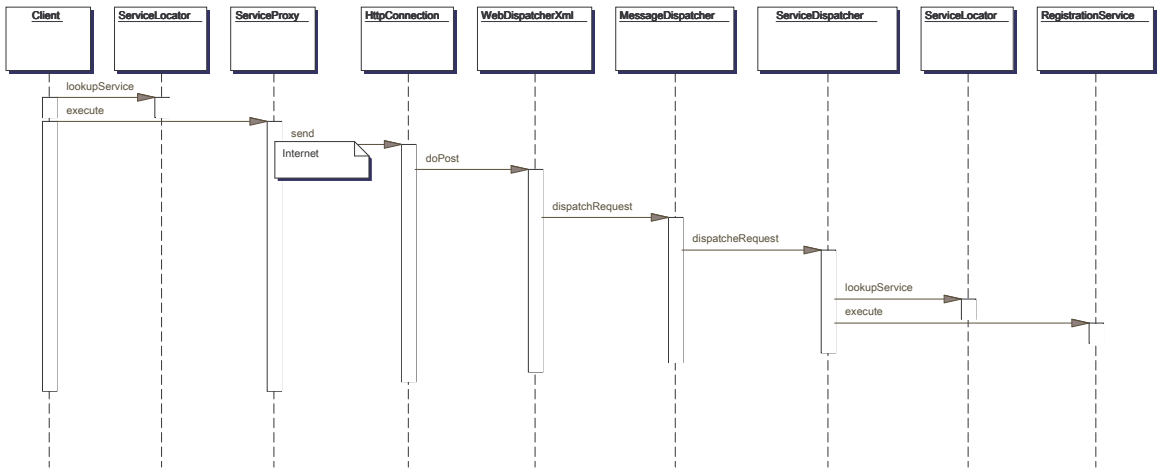

Abbildung 15: McServiceBroker - Registrierung

3.5.3. McProcessEngine

Mobile Geschäftsprozesse[21], d.h. Vorgänge, die generell vom Außendienst bearbeitet werden, müssen in die Abläufe der zentralen Organisation eingebunden werden, um Prozesslaufzeiten und Medienbrüche zu vermindern. Sie sind in der Regel Nachfolgeschritte vorgeschalteter Aktivitäten in der Zentrale und stoßen dort nach ihrer Beendigung weitere Tätigkeiten an.

Die Verfügbarkeit von Informationen bspw. für die verursachergemäße Kostenzuordnung ist eine Voraussetzung für die Planung und Durchführung von Optimierungsmaßnahmen. Aber im Gegensatz zu Innendienstvorgängen, die in der Vergangenheit weitreichend verbessert wurden, sind Laufzeiten und Kosten von mobilen Geschäftsprozessen in dem meisten Unternehmen nicht transparent. Benötigt werden Mechanismen, die es erlauben, mobile Geschäftsprozesse, deren Laufzeiten und die daraus resultierenden Kosten zu verfolgen sowie Mitarbeiter mit Arbeitsaufträgen und benötigten Informationen zu versorgen.

In der zentralen EDV haben sich im Laufe der letzten Jahre Workflow-Management-Systeme etablieren können, die einzelne isolierte Teilschritte zu automatisiert zusammenhängenden Prozessketten integrieren. Die Komponente McProcessEngine repräsentiert in seinem Kern ein einfaches Workflow-Management-System, das eine Integration mobiler und zentraler Workflows erlaubt.

Um die Integration mit anderen Workflow-Management-Systemen zu erleichtern, folgt die Implementierung dem Standard der OMG [Omg00]. In der genannten Spezifikation befindet sich eine detaillierte Beschreibung der Schnittstellen.

Die Umsetzung eines konkreten Workflows erfolgt derzeit per Deklaration von logischen Regeln innerhalb eines Textdokuments, das vom Framework gelesen wird. Zu diesem Zweck wurde mit Jess [Fri02] eine RuleEngine integriert, die vollkommen in Java programmiert und für wissenschaftliche Zwecke frei verfügbar ist.

[21] Unter einem *mobilen Geschäftsprozess* wird ein vollständiger oder auch ein Teil eines Geschäftsprozesses verstanden, der dezentral von mobilen Mitarbeitern durchgeführt wird.

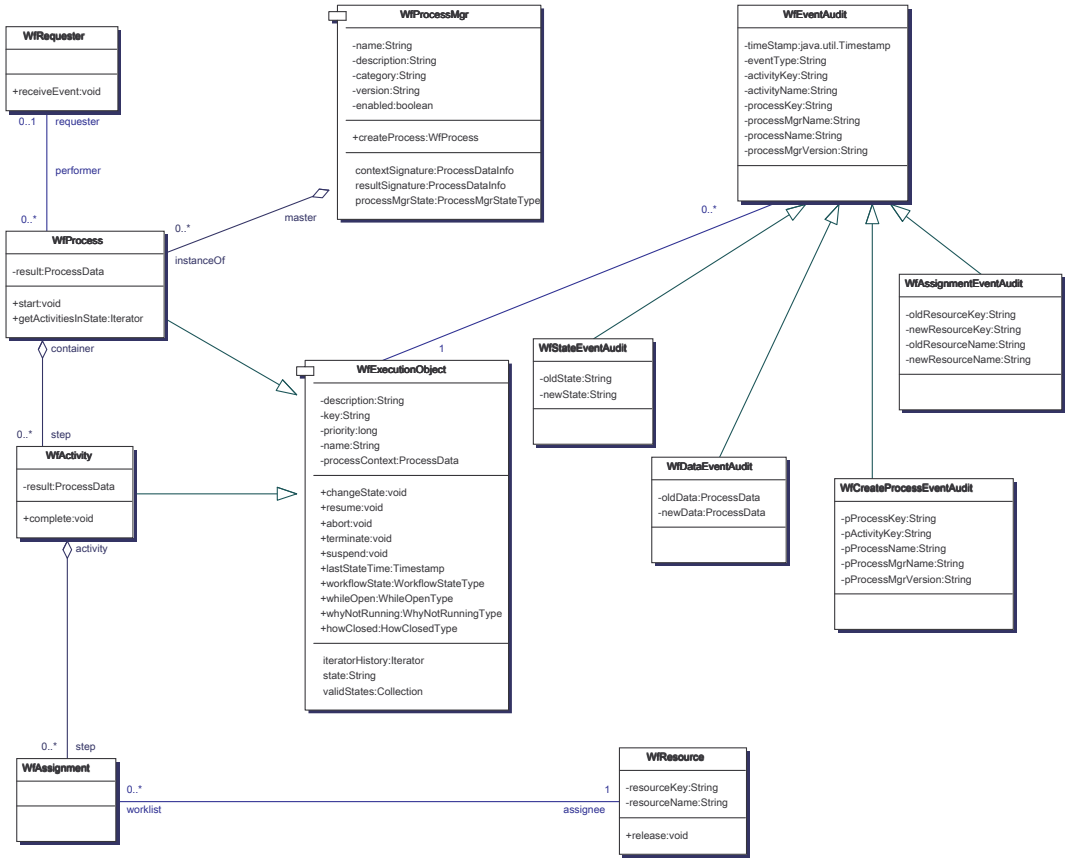

Abbildung 16: McProcessEngine konform zur OMG Workflow Facility (vgl. [Omg00])

Derzeit werden Eignung und Integrierbarkeit des verfügbaren Open-Source Workflow-Management-Systems *Open Business Engine (OBE)* [22] untersucht. Dieses System ist konform zu den Standards der Workflow Management Coalition (WfMC).[23] Mit der *XML Process Definition Language (XPDL)* hat die WfMC eine standardisierte Sprache zur Beschreibung von Prozessen definiert, die von der OBE ausgeführt werden können. Durch die Verwendung von XPDL ist die Prozessbeschreibung von ihrer Ausführung getrennt. Somit können unterschiedliche Werkzeuge eingesetzt werden. Es existieren bereits grafische, XPDL-konforme Editoren zur Prozessbeschreibung.[24]

[22] http://www.openbusinessengine.org

[23] z.B. http://jawe.enhydra.org/

3.5.4. McSyncManager

Mobile Anwendungen sollten neben einem Onlinebetrieb, d.h. die Verfügbarkeit einer permanenten Kommunikationsverbindung zwischen den Systemkomponenten ist gegeben, auch einen Offlinebetrieb erlauben. Bei dieser Betriebsart besteht die Kommunikationsverbindung nur zu bestimmten – meist benutzerdefinierten – Zeitpunkten. Aus unterschiedlichen Gründen kann ein Offlinebetrieb wünschenswert sein. Dazu zählen bspw. Senkung der Kommunikationskosten oder unzureichende Verfügbarkeit von Übertragungskanälen.

Ein Offlinebetrieb setzt voraus, dass die mobilen Anwendungen auf einen lokal vorgehaltenen Teilbestand der Daten zugreifen können. Zu diesem Zweck ist es erforderlich, zu bestimmten Zeitpunkten einen Abgleich zwischen lokalen und zentralen Daten vorzunehmen. Die Überführung dezentraler Datenänderungen wird nachfolgend als *Synchronisation*, die Verteilung von Daten auf mobile Endgeräte als *Replikation* bezeichnet.

Ein Replikationsverfahren muss zwei Probleme adressieren:

- Die Bestimmung der zu übertragenden Datenmenge vom Server auf das mobile Endgerät
- Die Überführung von dezentralen Datenänderungen in den zentralen Datenbestand. Dabei müssen Datenänderungskonflikte erkannt und möglichst automatisch behoben werden.

3.5.4.1. Bestandteile eines Replikations- / Synchronisationverfahrens

Bestimmung der zu übertragenen Datenmenge (Replikation)

Neue mobile Endgeräte definieren neue Anforderungen an die Synchronisation zwischen Client und Server (vgl. [HoPl01]):

- Die Struktur der Kommunikationsnetze, in denen sich die mobilen Endgeräte aufhalten, ist dynamisch. Zu einem bestimmten Zeitpunkt ist nicht vorhersagbar, welche mobilen Endgeräte für eine Kommunikation zur Verfügung stehen.
- Die Bandbreite der Kommunikationskanäle ist dynamisch. So können mobile Endgeräte im freien Gelände oft nur schmalbandige Übertragungskanäle nutzen.
- Die Hardware von mobilen Endgeräten ist nicht so leistungsfähig wie die stationärer Geräte.

- Die Nutzung von Mobilfunknetzen ist im Vergleich zur Festnetzkommunikation wesentlich teurer.

Aufgrund dieser Einschränkungen ist es notwendig, die zu übertragenden und lokal zu verwaltenden Datenmengen zu limitieren.

Die meisten existierenden Ansätze zur Datenreduktion basieren auf einer manuellen Spezifikation der zu replizierenden Daten (z.B. DB2 Everyplace, Oracle 9i Lite, Sybase Anywhere) (vgl. [Hor01b]). In jüngster Zeit werden zunehmend adaptive Verfahren diskutiert, die darauf abzielen, eine optimale Replikationsmenge in Abhängigkeit des Kontextes wie z.B. der aktuellen Position oder der durchzuführenden Aufgabe des Benutzers zu bestimmen (vgl. [HoPl01]).

Überführung von dezentralen Datenänderungen (Synchronisation)

Verfügbare kommerzielle Produkte für die Replikation von Datenbanken basieren i.d.R. auf Primärkopienverfahren (z.B. DB2-Everyplace, SQL-Anywhere, Oracle 9i Lite). Ein zentraler Primärkopienknoten (Server) speichert die Primärkopie eines Datenobjektes und eine beliebige Anzahl temporär verbundener Knoten (Clients, mobile Endgeräte) erhält sekundäre Kopien.

Verbindet sich ein mobiler Knoten zwecks Replikation mit dem zentralen Knoten, werden Aktualisierungen der lokalen Kopien an den zentralen Server als einzelne Schreiboperationen übermittelt. Dabei kann es vorkommen, dass mehrere Knoten parallel dasselbe Datum verändern wollen. Es ist Aufgabe des Replikationsverfahrens automatisch derartige Konflikte zu erkennen.

Erkannte Konflikte sollten nach Möglichkeit von dem DBMS automatisch aufgelöst werden.

So bietet z.B. Oracle vorgefertigte, frei wählbare Regeln für die Konfliktbehebung an. (Manche Datenänderungen sind z.B. kommutativ, d.h. es ist unbedeutend, welches neue Datum sich „durchsetzt". Ist diese Regel aktiviert, wählt das DBMS einen der neuen Datenzustände aus). Abgeschlossene Datenänderungen auf der Primärkopie werden anschließend an die übrigen Client-Knoten propagiert. Nicht auflösbare Konflikte werden an die betroffenen mobilen Knoten zurückgemeldet (vgl. [Hor01a]).

Nachteile

Für Systeme mit einer geringen Anzahl von Knoten, kurzen Offline-Zeiten oder nur sehr seltenen Datenänderungen stellen diese sogenannten *Lazy-Master-Replication-Systeme* eine geeignete Lösung dar. Die Untersuchungen von [GrHeOn96] zeigen aber, dass aufgrund einer exponentiell wachsenden Deadlock- und Konfliktwahrscheinlichkeit sowie mangelnder Skalierbarkeit sich große Systeme mit einer hohen Anzahl von mobilen Knoten nicht ohne weiteres umsetzen lassen.

Weiterhin ist die Abwesenheit von Konflikten nicht hinreichend, um die Integrität des zentralen Datenbestandes zu wahren. Ein einfaches Beispiel: Eine Offline-Überweisung resultiert in einer neuen Zeile in der Datenbank. Weil kein zentrales Vergleichsdatum existiert, wird kein Konflikt erkannt. In der Zentrale wurden zeitparallel Überweisungen vorgenommen, die das Überweisungskonto in den negativen Bereich geführt haben. Weitere Überweisungen lässt der Kontostand nicht zu. Über die konfliktfreie Replikation wird entgegen den Integritätsbedingungen dennoch die neue Überweisung gültig.

Provisorische Transaktionen

Für derartige Konstellationen bieten sogenannte *provisorische Transaktionen* die Möglichkeit, Integritätsverletzungen auf einer semantisch höheren, anwendungsnahen Ebene zu behandeln (vgl. [Hor01b]). Befindet sich ein mobiler Knoten offline, werden lokal alle Transaktionen provisorisch durchgeführt, d.h. alle Datenänderungen einer Transaktion sind provisorisch.

Provisorische Transaktionen unterliegen sogenannten Bereichs- oder auch Akzeptanzregeln, die den Gültigkeitsbereich einer Transaktion bestimmen. Ein Beispiel für den oben genannten Fall könnte sein, dass die Transaktion „Neue Überweisung" nur dann auf dem Server abgeschlossen werden darf, wenn der Kontostand den maximalen Dispositionskredit nicht überschreitet.

Beim Synchronisieren werden die provisorischen Transaktionen auf den zentralen Server übertragen. Dort wird versucht, die übermittelten provisorischen Transaktionen unter Einhaltung der Bereichsregeln global auszuführen.

Wird eine Bereichsregel verletzt, schlägt die betroffene provisorische Transaktion fehl und der synchronisierende Client wird benachrichtigt (vgl. [Gol01]).

Two-Tier-Replication

Ein Verfahren, das mit provisorischen Transaktionen arbeitet, ist das Two-Tier-Replication-Verfahren von [GrHeOn96], das wie folgt arbeitet:

Tätigkeiten des mobilen Knoten

- Alle provisorischen Datenänderungen werden verworfen.
- Von allen aktualisierten Primärkopien auf dem mobilen Knoten werden Repliken an den zentralen Knoten gesendet.
- Alle provisorischen Transaktionen werden inkl. der Eingabeparameter an den Server geschickt, wo sie in der Reihenfolge, wie sie auf dem mobilen Knoten abgeschlossen wurden, ausgeführt werden.
- Empfang von neuen Repliken (herkömmliches Primärkopienverfahren).
- Empfang von Erfolgs- und Fehlermeldungen der übermittelten provisorischen Transaktionen.

Tätigkeiten des Replikationsservers

- Verspätete Aktualisierungstransaktionen werden an den mobilen Knoten geschickt.
- Mit Verzögerung eingetroffene Aktualisierungstransaktionen des mobilen Knoten werden empfangen.
- Die Liste der provisorischen Transaktionen inkl. Eingabeparameter und Akzeptanz-kriterium wird empfangen. Anschließend werden die provisorischen Transaktionen auf dem Server wiederholt und abgeschlossen. Falls das Akzeptanzkriterium verletzt wird, schlägt die Transaktion fehl und es wird eine Rückmeldung an den Client geschickt.
- Erfolgreiche Aktualisierungen werden an weitere Replikationsknoten weiter-gegeben.

3.5.4.2. Aufbau des Replication-Frameworks

Das Replication-Framework realisiert eine Kombination aus zwei Verfahren. Die Versorgung der Clients mit Aktualisierungen erfolgt mit Hilfe eines Master-Copy-Algorithmus. Daten-änderungen, die auf den mobilen Endgeräten erfolgen, werden unter Einsatz einer Two-Tier-Replication in den zentralen Bestand übernommen.

Im Folgenden wird das Zusammenwirken der beteiligten Komponenten auf Client und Server illustriert.

Synchonisation

Clientseitig

In modernen Client/Server-Plattformen wie J2EE, .NET oder CORBA werden applikationsnahe Transaktionen mit den Mitteln des benutzten DBMS (z.B. unter Einsatz von *Stored Procedures*) tendenziell selten implementiert. Stattdessen werden Funktionen in Form von Diensten realisiert, die ein Application Server oder ein Object Request Broker ausführt. Somit repräsentiert auch ein `Service` der Middlewareplattform eine fachliche Transaktion, die repliziert werden kann, wenn sie auf einem mobilen Client ausgeführt wird und die zentrale Datenbank verändert.

Um einen Dienstaufruf als fachliche Transaktion realisieren zu können, ist ein spezieller `ServiceExecutor` (`SynchronisedExecutor`) konzipiert worden, der jedem Dienst per Konfiguration transparent vorgeschaltet werden kann. Dieser Executor fängt den jeweiligen Dienstaufruf ab und erzeugt eine provisorische Transaktion, die aus einer Operation (`ServiceOperation`) und den zugehörigen Aufrufparametern besteht. Dann wird der Dienst lokal ausgeführt und die Transaktion in das `TransactionLog` übernommen. Schlägt der lokale Aufruf fehl, so wird die Transaktion entfernt. *Abbildung 17: Synchronisation - Client* zeigt die clientseitigen Interaktionen.

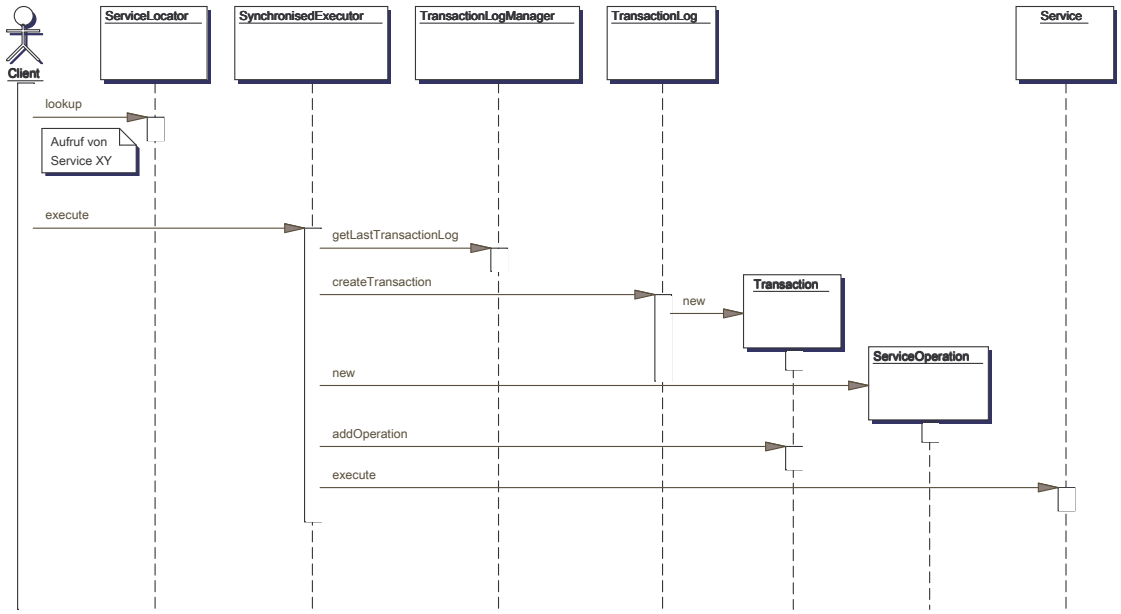

Abbildung 17: Synchronisation - Client

Das ursprüngliche Konzept, das auf Dynamic Proxies (Java2) basierte, ist aus Optimierungsgründen durch den zuvor beschriebenen Ansatz ersetzt worden. Dadurch können die aufwendigen Aufrufe per Java-Reflection vermieden und beliebige Dienste repliziert werden.

Der Vorteil der Two-Tier-Replication besteht vor allem in der Vermeidung zusätzlichen Programmieraufwandes. Provisorische Transaktionen sind vollkommen transparent. Allerdings muss im Allgemeinen darauf geachtet werden, dass persistente Zustandsveränderungen ausschließlich über Dienste zugelassen werden.

Um eine Synchronisation durchzuführen, muss das auf der Clientseite geführte `TransactionLog` über einen Aufruf des `SynchronisationService` zum Server übermittelt werden und dort ausgeführt werden.

Serverseitig

Wird der `SynchronisationService` durch einen Client aufgerufen, so wird zunächst das Transaktionslog registriert (`TransactionLogManager`). Dabei wird überprüft, ob das serverseitig registrierte Transaktionslog bereits erfolgreich synchronisiert wurde. Nur dann kann das neue Transaktionslog registriert werden. Die eigentliche Synchronisation erfolgt über den `SynchronisationManager`, der alle provisorischen Transaktionen des Logs ausführt. *Abbildung 18: Synchronisation - Server* illustriert diesen Ablauf.

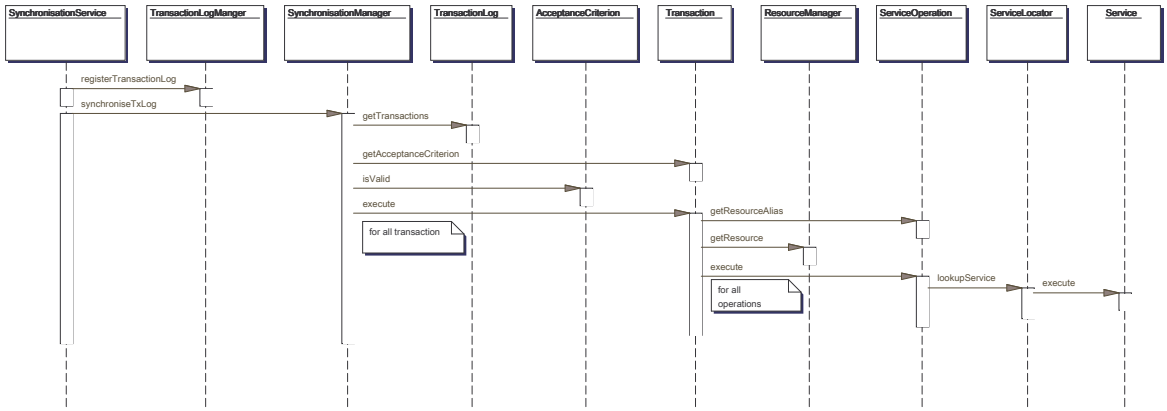

Abbildung 18: Synchronisation - Server

Im Rahmen der Ausführung einer provisorischen Transaktion wird zunächst das Akzeptanzkriterium (`AcceptanceCriterion`) überprüft. Schlägt die Validierung fehl, so wird die Transaktion abgelehnt und verbleibt im Transaktionslog. Andernfalls wird die darin enthaltene `ServiceOperation` ausgeführt. Diese ermittelt über den `ServiceLocator` den entsprechenden Dienst und ruft ihn mit den übermittelten Parametern auf. Rückgabe des Dienstaufrufs ist ein Transaktionslog, das alle zurückgewiesenen Transaktionen enthält.

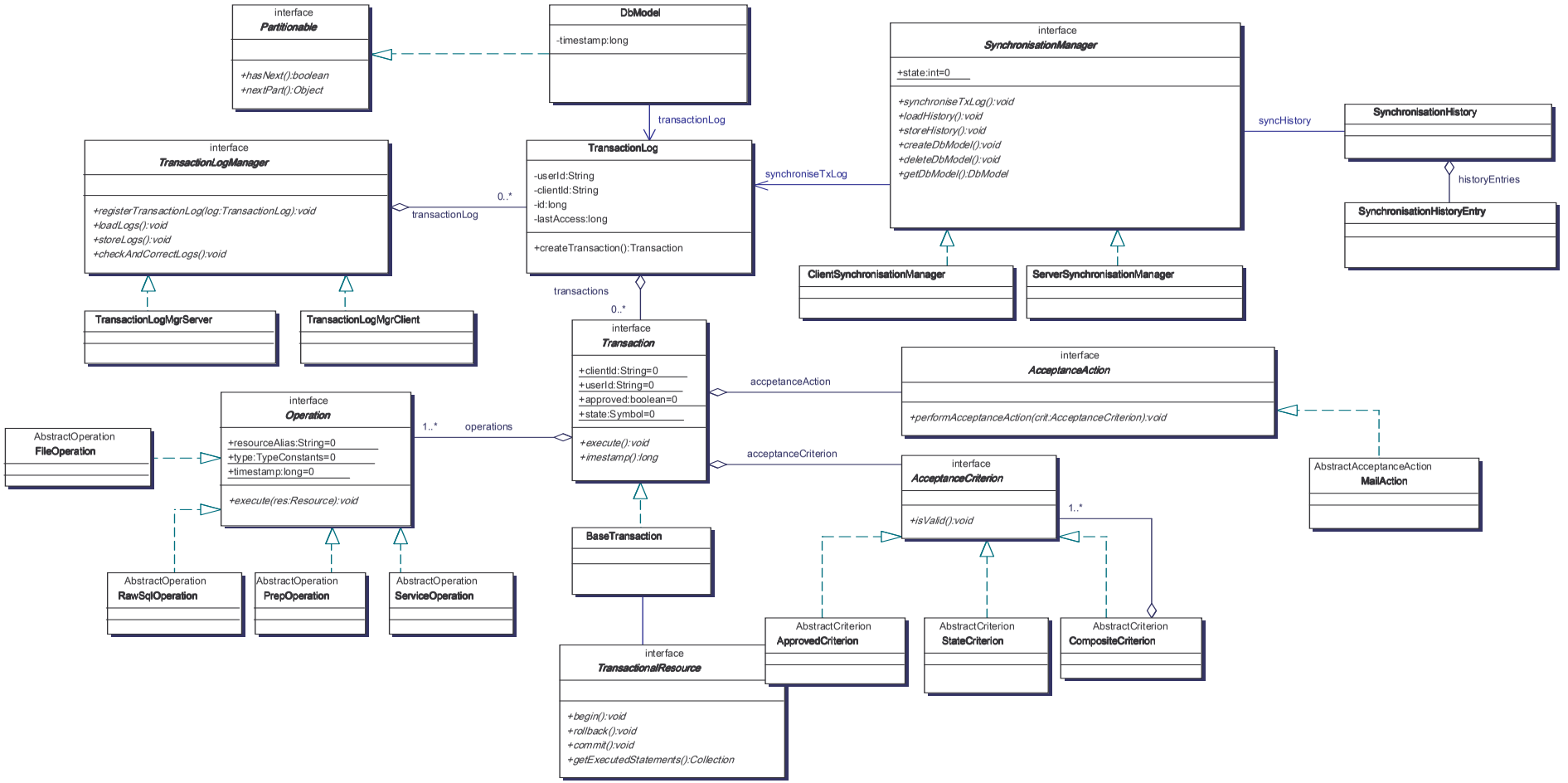

Abbildung 19: Synchronisationsmodell

Replikation

Für jeden Benutzer werden sog. Subskriptionen konfiguriert, welche die zu replizierenden Daten beschreiben. Derzeit können SQL-Datenbanken (`JDBCSubscription`) und Dateien (`FileSubscription`) abonniert werden. Eine einzelne Subskription nutzt eine bestimmte Ressource, die über einen logischen Namen (ResourceAlias) referenziert wird (siehe *Abbildung 20: Subskriptionsmodell*). Die gewünschte Datenselektion wird den Mittel der jeweiligen Ressource beschrieben (z.B. SQL-Select-Statements).

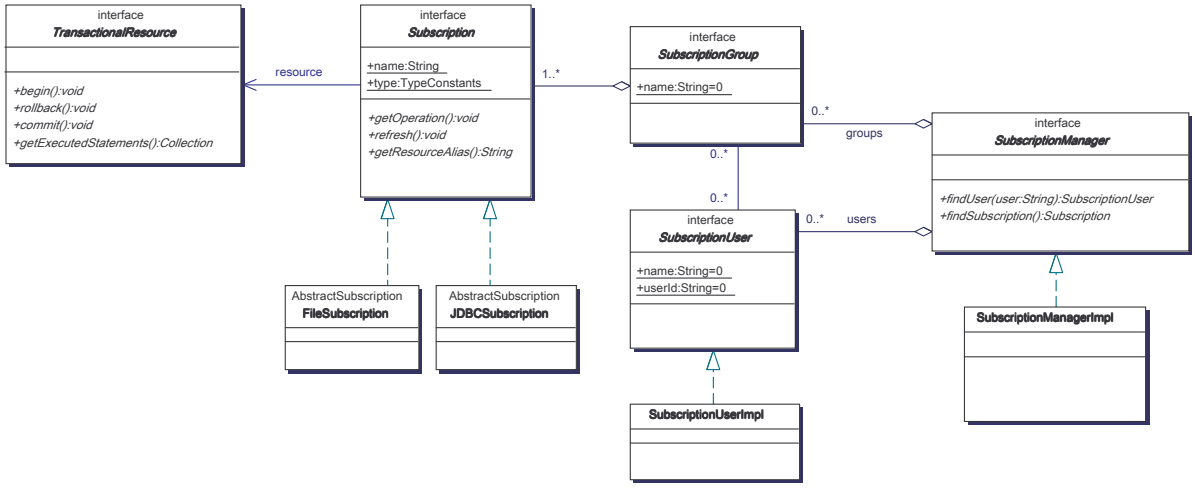

Abbildung 20: Subskriptionsmodell

Aktualisierung der Ressourcen

Führt ein Client eine Synchronisation - wie im vorangehenden Abschnitt erläutert - durch, so werden die im übertragenen Transaktionslog enthaltenen provisorischen Transaktionen mit ihren Operationen ausgeführt. Innerhalb einer Operation werden Zugriffe auf Ressourcen (bspw. JDBC-Verbindungen) über entsprechende Instanzen von `Resource` (z.B. `JDBCResource`) durchgeführt. Die Verwaltung der Ressourcen erfolgt durch den `ResourceManager` (siehe *Abbildung 21: Ressourcenmodell*).

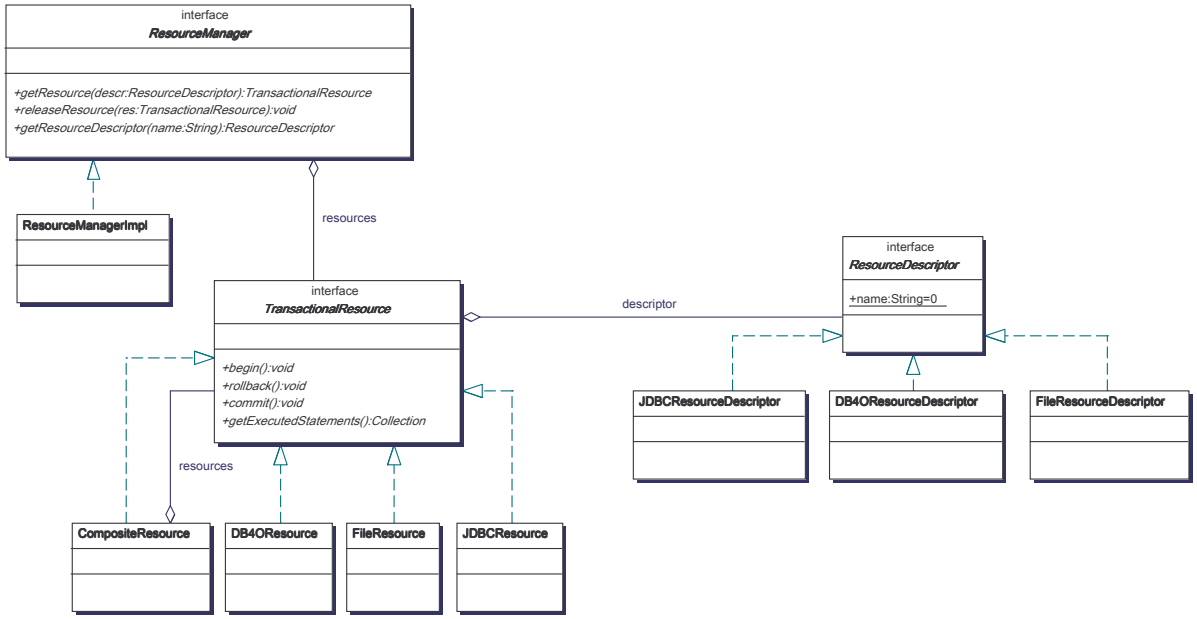

Abbildung 21: Ressourcenmodell

Nach dem Beginn der Synchronisation wird jede einzelne provisorische Transaktion mit ihrer `ServiceOperation` ausgeführt. Diese ruft dann den eigentlichen Dienst auf. Diesem ist jedoch ein spezieller Executor (`ResourceExecutor`) vorgeschaltet, der der Aktualisierung der betroffenen Ressource in einem Transaktionslog protokolliert.

Wird nun ein Dienst aufgerufen, erzeugt der `ResourceExecutor` zunächst eine neue Transaktion, und es wird, falls für die betroffene Ressource noch kein Transaktionslog registriert ist, ebenfalls ein Transaktionslog erzeugt. Die `Resource` fängt alle nativen Kommandos (z.B. SQL-Statements) ab und protokolliert diese, bis die provisorische Transaktion abgeschlossen ist. Der `ResourceExecutor` erfragt dann von der Ressource alle durchgeführten Kommandos und erzeugt für jedes Kommando eine korrespondierende Operation (z.B. `RawSqlOperation`), die der Transaktion hinzugefügt wird. Nach der erfolgreichen Synchronisation des übertragenen Transaktionslogs existiert dann für jede modifizierte Ressource ein Transaktionslog, das alle Aktualisierungen dieser Ressource enthält (siehe *Abbildung 22: Ressourcenaktualisierung*).

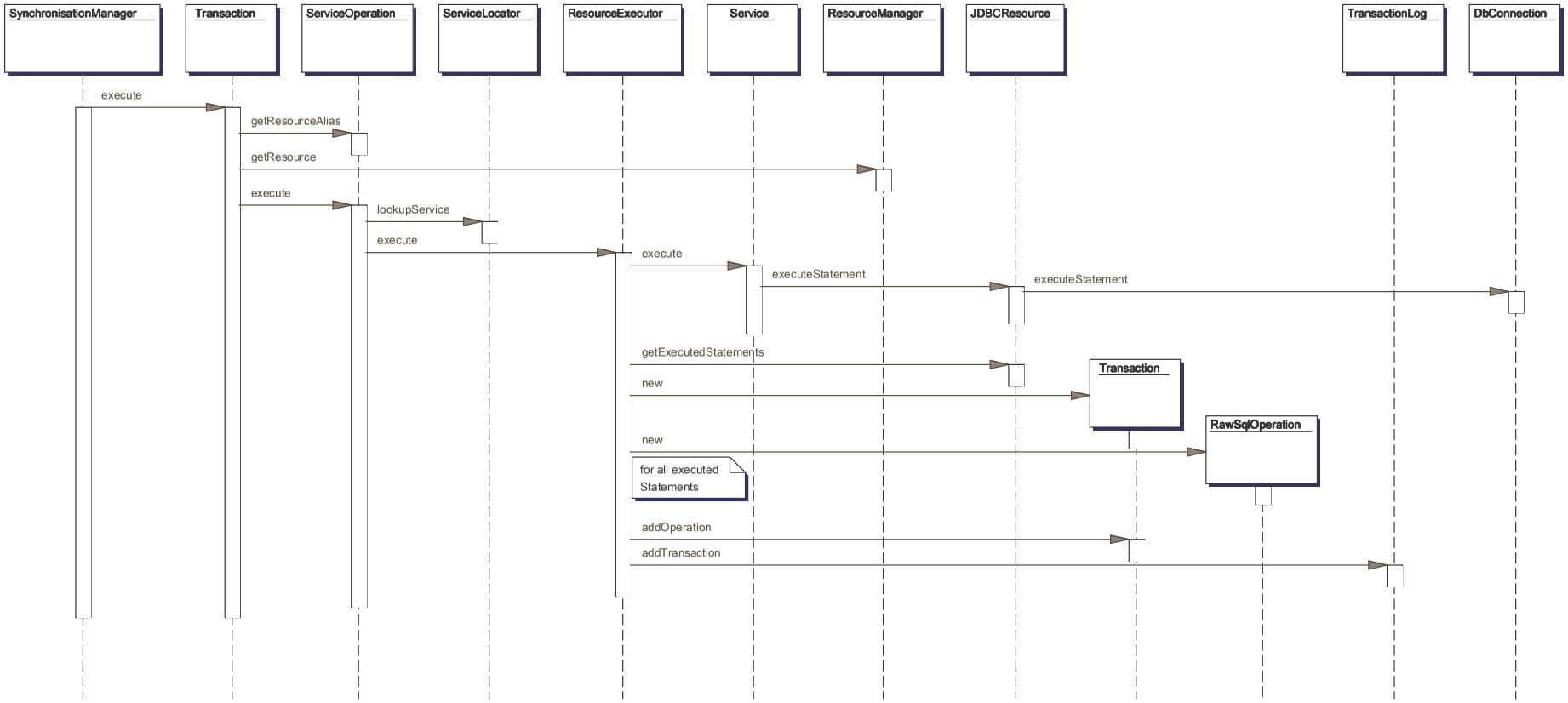

Abbildung 22: Ressourcenaktualisierung

Ermittlung der zu übertragenden Datenmenge durch Subskriptionen

Ruft nun ein Client den `SubscriptionService` auf, so werden über den `SubscriptionManager` die konfigurierten Subskriptionen gelesen. Für jede einer Subskription assoziierten Ressource wird überprüft, ob seit der letzten Replikation Aktualisierungen der jeweiligen Ressource durch andere Clients durchgeführt wurden. Falls ja, dann werden die entsprechenden Transaktionen einem temporären Transaktionslog hinzugefügt. Dieses Vorgehen wird für alle abonnierten Ressourcen durchgeführt. Dieses temporäre Transaktionslog wird als Ergebnis des Dienstaufrufs zum Client überspielt und dort synchronisiert (siehe *Abbildung 23: Subskription – Server*).

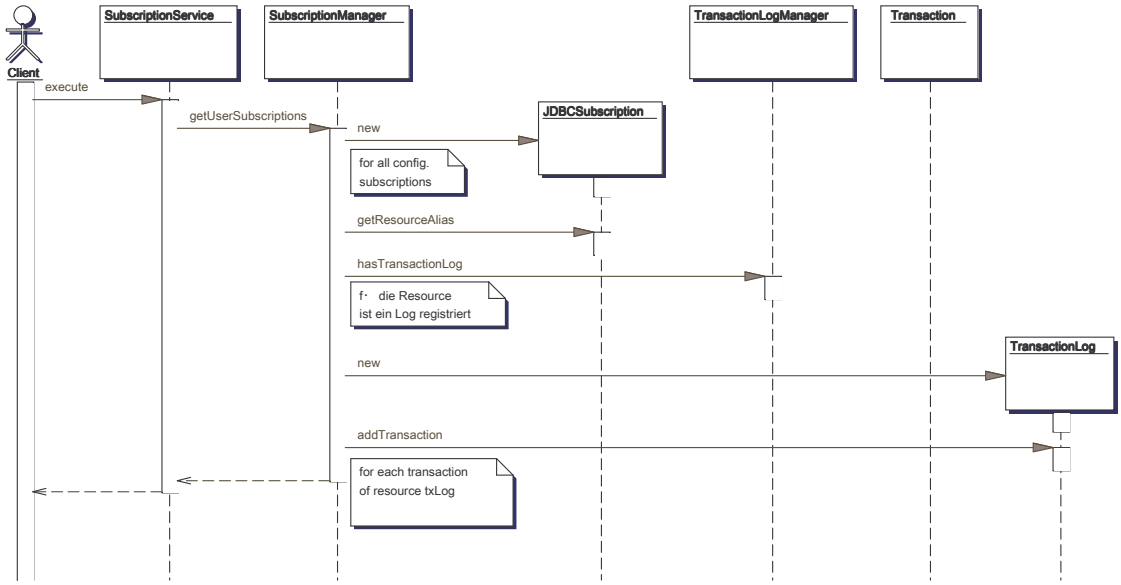

Abbildung 23: Subskription – Server

3.6. Integrierte Technologien und Produkte

3.6.1. WebWork

Mit *WebWork* steht ein Open-Source Web-Application-Framework zur Verfügung, dass die Entwicklung und Wartung von Web-Anwendungen stark vereinfacht. Durch die konsequente Ausrichtung des Frameworks an dem Model-View-Controller-Paradigma (MVC) wird eine modulare Dekomposition von Anwendungen erreicht. Das zentrale Konzept in WebWork stellen *Actions* dar, die für die Steuerung einer Anwendung verantwortlich sind (sog.

Controller). Ihre Abfolge und die zugehörigen Benutzeroberflächen (Views) sind deklarativ per XML konfigurierbar.

Zur Integration von WebWork sind spezielle Adapter konzipiert worden, die eine Nutzung von MoBiLo-Diensten in Actions erlauben (siehe *Abbildung 24: Integration WebWork*). Somit ist es möglich, einzelne Bearbeitungsschritte einer Web-Anwendung transparent auf mobilen Clients ausführen zu lassen oder die Ergebnisse von Dienstaufrufen (z.B. Position eines Clients) zur Anwendungssteuerung zu verwenden.

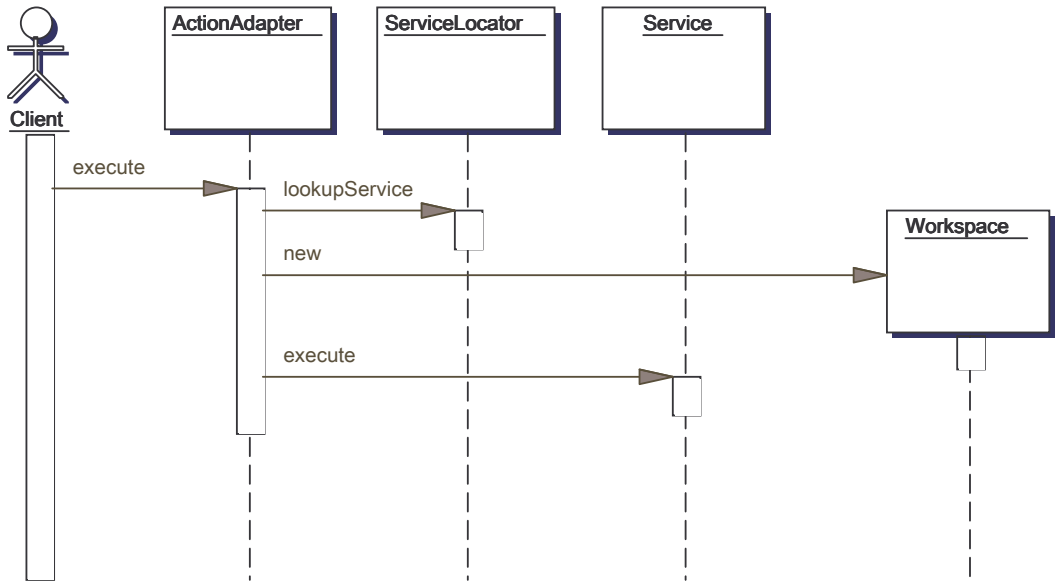

Abbildung 24: Integration WebWork

3.7. Erfahrungen / Ausblick

3.7.1. Mobilgeräteplattformen

Aufgrund der intensiven Nutzung der unterschiedlichen Gerätetypen konnten entwicklungs- und betriebsrelevante Aspekte identifiziert werden, die nachfolgend kurz erläutert werden:

1. Datensicherheit bei mobilen Geräten

   Die eingesetzten Geräte auf Basis von PocketPC und Linux besitzen keinen von der Stromversorgung unabhängigen Speicher. Der Verlust der Stromversorgung geht einher mit der Zurücksetzung in den Ursprungszustands des Gerätes. Die Nutzung

von Sicherung des Datenbestandes erwies sich als sehr umständlich und fehlerträchtig, da die Sicherung oft nicht wiederhergestellt werden konnte. Zudem konnte auf der PocketPC Plattform nur auf dasselbe Gerät zurückgesichert werden, was bei Verlust oder Defekts eines Geräts den vollständigen Datenverlust bedeutet. Diese Problematik kann nicht ausreichend auf Ebene der Anwendungsentwicklung begegnet werden.

2. Unterschiedliche Java-Laufzeitumgebungen (JVM)

   Derzeit existiert eine Vielzahl von JVMs für die verschiedenen Mobilplattformen. Diese unterscheiden sich teilweise gravierend hinsichtlich des Startverhaltens, Ablaufgeschwindigkeit und Ressourcenbedarfs. Dies erschwert prinzipiell die Portierbarkeit und schränkt somit die Plattformunabhängigkeit der Anwendungen ein.

3.7.2. Kommunikationstechnologien

Die Nutzung von GPRS hat sich im Vergleich zu GSM als eher problematisch erwiesen. Ursache ist die Vergabe der IP-Adressen für die mobilen Geräte durch die Netzbetreiber. Aufgrund der Knappheit von IPv4-Adressen werden für GPRS-nutzende Geräte keine öffentlichen IP-Adressen vergeben. Die Folge ist, dass jede Kommunikation von dem mobilen Gerät initiiert werden muss. Eine aktive Aufnahme der Kommunikation durch den MoBiLo-Middlewareserver ist nicht möglich, so dass sich keine Push-Funktionalität (z.B. erforderlich für eine Notifikation) realisieren lässt.

Die WLAN-Kommunikationstechnologie hat sich im Projektverlauf sehr bewährt und ist intensiv im Rahmen der prototypischen Realisierung eingesetzt worden. Sie hat sich im Gegensatz zu seriellen und USB-Verbindung als wesentlich schneller und zuverlässiger erwiesen. Zudem hat auch die Verbreitung von Bluetooth-fähigen mobilen Geräten stetig zugenommen, so dass die Funktionsfähigkeit der MoBiLo-Kommunikationskomponente mit einem Bluetooth-basierten Netzwerk erprobt werden konnte.

3.7.3. Iteratives Vorgehensmodell

Die iterative Vorgehensweise hat sich im bisherigen Projektverlauf bewährt, da sie eine einfache Adaption neuer Technologien ermöglicht hat. Nach jeder Iteration wurde eine Auswertung durchgeführt, die Rückschlüsse auf die Eignung der verwendeten Technologien erlaubte. Wurden diesbezüglich Defizite festgestellt oder eröffneten sich durch neue

Technologien Verbesserungspotenziale, so wurden diese in nachfolgenden Iterationen berücksichtigt.

4. Projektplanung und Ablauf

Das Vorhaben MoBiLo wurde im Januar 2002 begonnen und hatte eine geplante Laufzeit von zwölf Monaten, nach der Bewilligung einer kostenneutralen Laufzeitverlängerung dann von 16 Monaten. MoBiLo konnte planmäßig im April 2003 unter Einhaltung des genehmigten Budgets in Höhe von € 153.388 erfolgreich abgeschlossen werden.

Das Vorhaben wurde in Kooperation mit dem in Koblenz ansässigem Softwareunternehmen *die mobilanten GmbH* durchgeführt, das sich auf die Entwicklung mobiler Anwendungen spezialisiert hat. Das Unternehmen hat insbesondere die Konzeption und Konkretisierung der Architektur der Middlewareplattform unterstützt und erforderliches Expertenwissen in das Projekt eingebracht.

Eine weitere initiierte Kooperation mit dem Telekommunikationsunternehmen Quam konnte nicht umgesetzt werden, das dieses Unternehmen Mitte 2002 den Geschäftsbetrieb eingestellt hat.

Im Bereich der Öffentlichkeitsarbeit wurde ein Artikel über mobile Frameworks am Beispiel der MoBiLo -Middleware bei der renommierten Fachzeitschrift OBJEKT*spectrum* eingereicht und wird dort veröffentlicht. Zudem wird ein Bericht im Rahmen der Beitragsreihe *Arbeitsberichte des Instituts für Wirtschafts- und Verwaltungsinformatik* der Universität Koblenz-Landau erscheinen.

Weiterhin ist das Projekt MoBiLo auf folgenden Messen und Veranstaltungen ausgestellt und einem Fachpublikum präsentiert worden:

- CeBit 2002
- Wirtschaftsinformatik Forum 2003
- CeBIT 2003
- Handwerksmesse Koblenz 2003

5. Erfolgsaussichten und Anschlussfähigkeit

5.1. wissenschaftliche Anschlussfähigkeit

5.1.1. Projekt ECOMOD

Am Institut für Wirtschafts- und Verwaltungsinformatik wird das von der Deutschen Forschungsgesellschaft (DFG) geförderte Projekt *ECOMOD: Unternehmensmodellierung für E-Commerce* [25] durchgeführt, das sich mit der Modellierung unternehmensübergreifender Geschäftsprozesse beschäftigt. Auf Basis der Prozessmodelle soll dann ablauffähiger Programmcode generiert werden.

Es ist beabsichtigt, MoBiLo als Ausführungsplattform für ECOMOD einzusetzen. Derzeit wird untersucht, welche Anpassungen bzw. Erweiterungen der MoBiLo-Plattform erforderlich sind.

5.1.2. Promotionsstelle am Institut für Wirtschafts- und Verwaltungsinformatik

Die am Institut für Wirtschafts- und Verwaltungsinformatik entwickelte Methode zur multi-perspektivischen Unternehmensmodellierung MEMO[26] soll um spezifische Konzepte zur Modellierung und Generierung mobiler Workflows erweitert werden. Dabei ist geplant, auf die Forschungsergebnisse von MoBiLo aufzusetzen und die Middlewareplattform entsprechend zu erweitern. Voraussichtlich ab September 2003 wird eine Promotionsstelle mit einer mehrjährigen Laufzeit für diesen Themenkomplex ausgeschrieben.

5.1.3. Innovationspreis Rheinland-Pfalz 2003

Der Minister für Wirtschaft, Verkehr, Landwirtschaft und Weinbau des Landes Rheinland-Pfalz vergibt seit 1988 jährlich einen Innovationspreis an besonders innovative Unternehmen oder Forschungseinrichtungen zur Anerkennung ihrer Leistungen und ihres Einsatzes für Innovationen in der Wirtschaft des Landes. Die Firma *die mobilanten GmbH* hat sich mit einer Weiterentwicklung der Middleware-Plattform MoBiLo um diesen Preis beworben.

---

[25] http://www.uni-koblenz.de/~ecomod/

[26] http://www.uni-koblenz.de/~iwi/UM/MEMO/index.html

5.1.4. Ideenwettbewerb UMTS Rheinland-Pfalz

Das rheinland-pfälzische Ministerium für Wirtschaft, Verkehr, Landwirtschaft und Weinbau und die Vodafone D2 GmbH veranstaltet einen Ideenwettbewerb für innovative Dienste und Anwendungen auf der Basis verschiedener Mobilfunkgenerationen. *Die mobilanten GmbH* hat ein auf der Weiterentwicklung der MoBiLo-Plattform basierendes Konsumenten-Informations-System (KIS), das sich zurzeit in der Konzeptionsphase befindet, zur Teilnahme an diesem Projekt eingereicht. Dieses KIS soll dazu dienen, Verbrauchern bei Einkäufen schnell und problemlos beispielsweise über Inhaltsstoffe der Produkte zu informieren.

5.2. wirtschaftliche Erfolgsaussichten

Der Projektpartner *die mobilanten GmbH* hat die MoBiLo-Technologie lizenziert und weiter-entwickelt. Die Plattform wird im Rahmen von Beratungs- und Entwicklungsprojekten in mittelständischen Unternehmen eingesetzt. Dadurch werden ein Wissenstransfer sowie ein Multiplikationseffekt erreicht.

Aufgrund der Vorführbarkeit der MoBiLo-Plattform und entsprechenden Anwendungs-prototypen konnten zunehmend Interessenten und potentielle Käufer aus unterschiedlichen Bereichen gewonnen werden:

1. Mittelständische Unternehmen und Organisationen

   Für die Pilotierung der Plattform konnten mittlerweile einige Interessenten gewonnen werden, die sowohl auf der CeBIT als auch im regionalen Umfeld von Koblenz für den Einsatz mobiler Anwendungen überzeugt werden konnten.

   Mittlerweile wird die Plattform auch erfolgreich in mehreren Projekten in kleinen und mittelständischen Versorgungsunternehmen eingesetzt.

   Ein Hersteller für Instandhaltungsplanungs- und Steuerungssysteme hat die MoBiLo-Plattform in seine Produktpalette integriert und Verhandlungen mit mehreren anderen Herstellern stehen vor einem erfolgreichen Abschluss.

2. Netzbetreiber

   Netzbetreiber haben ein generelles Interesse an der Verbreitung von Datendiensten, da ihnen die Infrastruktur für die Kommunikation zur Verfügung steht.

Der Kooperationspartner *die mobilanten GmbH* portiert derzeit MoBiLo auf die Blackberry-Geräteplattform. Das fertige Produkt soll dann mit MoBiLo-Lizenzen über einen deutschen Netzbetreiber vermarktet werden.

3. Application Service Provider

   ASP-Unternehmen betreiben für Kunden IT-Lösungen und können mit einer MoBiLo-Plattform neuartige Lösungen anbieten. Sie haben auch ein besonderes Interesse an flexibler Middlewaretechnologie, da ihre Kunden tendenziell unterschiedliche Anforderungen haben.

4. Softwarehersteller und IT-Beratungsunternehmen

   Softwarehersteller haben unterschiedliche Motive für den Einsatz der MoBiLo-Plattform. Zum einen erwägen sie die Möglichkeit der kostengünstigen Erstellung mobiler Anwendungen und eine Reduzierung des Projektrisikos aufgrund der Verwendung bestehender Komponenten. Zum anderen wollen Anbieter ihre Produkte um Funktionalitäten für die Unterstützung mobiler Mitarbeiter erweitern (Beispiel: Inventursystem-, Logistiksoftwarehersteller).

   Über den Kooperationspartner ist ein namhafter Anbieter von Software für die Energiewirtschaft gewonnen worden, der seinen Kunden auf MoBiLo basierende Lösungen anbieten möchte.

Literaturverzeichnis

[AhAtHa01]	R. Ahri, S. Atkinson, D. Ayers, M. Haglind, B. Ray, R. Machin, N. Nashi, R. Taylor, C. Wiggers. *Professional Java Mobile Programming*. Wrox. 2001.
[CaBlMa02]	L. Capra, G.S. Blair, C. Mascolo, W. Emmerich, P. Grace : *Exploiting Reflection in Mobile Computing Middleware*. In: ACM Mobile Computing and Communications Review. 6(4):34-44. 2002
[CaBlZa01]	L. Capra, C. Mascolo, S. Zachariadis, W. Emmerich : *Towards a Mobile Computing Middleware : a Synergy of Reflection and Mobile Code Techniques*. In Proc. of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems (FTDCS'2001). Bologna, Italy. pp. 148-154. IEEE Computer Society Press. October 2001.
[EdYo00]	E. Yourdon. *Managing Software Requirements: A Unified Approach*. Addison-Wesley. 2000
[FlMa02]	Floyd Marinescu. *EJB Design Patterns: Advanced Pattern, Processes and Idioms*. WILEY. 2002.
[Fri02]	E. Friedman-Hill. *Jess - The Expert System Shell for the Java Platform.* 2002. http://herzberg.ca.sandia.gov/jess
[FrJu01]	Fraunholz, B.; Jung, J.: *Evaluation of Mobile Applications – software-technical and human aspects*. In: Proceedings of the Conference on IS Evaluation, Paris, July 2001
[GaHeJoVl95]	E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley. 1995.
[GoPr99]	C. F. Goldfarb, P. Prescod. *XML Handbuch.* Prentice Hall. 1999.
[Gol01]	C. Gollmick*. Unterstützung mobiler Clients durch erweiterte Client/Server-Datenbanksysteme*. In: Tagungsband des 13. GI-Workshop „Grundlagen von Datenbanken", Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg. Seiten 43-47, Gommern. 2001

[GrHeOn96]	J. Gray, P. Helland P. O'Neill, D. Shasha. *The Dangers of Replication and a Solution.* In: Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Canada. Seiten 173-183. 1996
[Gud99]	T. Gudehus. *Logistik: Grundlagen, Strategie, Anwendungen*. Springer. 1999
[Hoe01a]	H. Höpfner. *Replikationstechniken für mobile Datenbanken*. Diplomarbeit. Fakultät für Informatik. Institut für Technische und Betriebliche Informationssysteme. Universität Magdeburg. 2001
[Hoe01b]	H. Höpfner. *Grundlagen von Replikationstechniken für mobile Datenbanken.* In: Tagungsband des 13. GI-Workshop „Grundlagen von Datenbanken", Preprint Nr. 10, Fakultät für Informatik, Universität Magdeburg. Seiten 68-72, Gommern. 2001
[HoPl01]	H. Höpfner, M. Plack. *Eigenschaftsbasierte Replikation in mobilen Datenbanksystemen*. Arbeitspapier (preprint) der Fakultät für Informatik. Institut für Technische und Betriebliche Informationssysteme. Universität Magdeburg. 2001. http://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen /preprints/2001/HoePla01.html
[JaBoRu99]	I. Jacobson, G. Booch, J. Rumbaugh. *The Unified Software Development Process.* Addison-Wesley. 1999
[JuFr01]	J. Jung, U. Frank. *Konzeption der Architektur eines Flottenmanagement-systems im Kundendienst*. In: Tagungsband Logistikmanagement 01 Aachen. 12. – 14.09. 2001
[JuHa01]	J. Jung, J.F. Hampe. *Konzeption einer Architektur eines Flotten-managementsystems*. Arbeitsberichte des Instituts für Wirtschafts-informatik. Nr. 23. Februar 2001
[JuLa01]	J. Jung, B.L. van Laak. *Flottenmanagementsysteme – Grundlegende Technologien, Funktionen und Marktüberblick*. Arbeitsberichte des Instituts für Wirtschaftsinformatik. Nr. 28. Juli. 2001

[Mc00]	B. McLaughlin. *Java and XML*. O'Reilly. 2000.
[RoRo99]	S. Robertson, J. Robertson. *Mastering the Requirements Process.* Addison-Wesley. 1999
[SiStJo02]	I. Singh, B. Stearns, M. Johnson. *Designing Enterprise Applications with the J2EE Platform.* Addison-Wesley. 2002.
[Omg00]	Object Management Group. *Workflow Management Facility Specification, V1.2*. 2000. http://www.omg.org

<table>
<tr><td>1. ISBN oder ISSN</td><td>2. Berichtsart
Abschlussbericht</td></tr>
<tr><td colspan="2">3a. Titel des Berichts
MoBiLo: Middlewareplattform zur Unterstützung mobiler Logistiksysteme unter Einsatz aktueller Middleware, die auf Internet- und Open-Source-Technologien basiert</td></tr>
<tr><td colspan="2">3d. Titel der Publikation
Die MoBiLo-Plattform: Konzeption einer mobilen Middleware</td></tr>
<tr><td>4a. Autoren des Berichts (Name, Vorname(n))
Hoffmann, Jürgen
Herold, Christian</td><td>5. Abschlussdatum des Vorhabens
April 2003</td></tr>
<tr><td rowspan="2">4b. Autoren der Publikation (Name, Vorname(n))
Hoffmann, Jürgen
Herold, Christian</td><td>6. Veröffentlichungsdatum
Juni 2003</td></tr>
<tr><td>7. Form der Publikation
Arbeitsbericht des Instituts für Wirtschafts- und Verwaltungsinformatik</td></tr>
<tr><td rowspan="4">8. Durchführende Institution(en) (Name, Adresse)
Universität Koblenz-Landau
Institut für Wirtschafts- und Verwaltungsinformatik
Universitätsstr. 1
56075 Koblenz</td><td>9. Ber.Nr. durchführende Institution</td></tr>
<tr><td>10. Förderkennzeichen
01AK049</td></tr>
<tr><td>11a. Seitenanzahl Bericht
51 Seiten</td></tr>
<tr><td>11b. Seitenanzahl Publikation
49 Seiten</td></tr>
<tr><td>13. Fördernde Institution (Name, Adresse)
Bundesministerium für Bildung, Forschung und Technologie (BMF)
53170 Bonn</td><td>12. Literaturangaben</td></tr>
<tr><td></td><td>14. Tabellen</td></tr>
<tr><td></td><td>15. Abbildungen</td></tr>
<tr><td>16. Zusätzliche Angaben</td><td></td></tr>
<tr><td>17. Vorgelegt bei (Titel, Ort, Datum)</td><td></td></tr>
<tr><td colspan="2">18. Kurzfassung
Mit den Java2-Varianten J2ME, und J2EE stehen auf Client- bzw. Serverseite leistungsfähige Plattformen zur Verfügung, die jedoch für eine kostengünstige und schnelle Entwicklung mobiler Anwendungen alleine nicht ausreichend sind. Komplementäre Technologien (wie bspw. XML) zur Unterstützung mobiler Anwendungen stehen bereit, sind aber unzureichend integriert.
Ziel des Vorhabens ist die Konzeption einer Middlewareplattform zur Unterstützung der Entwicklung mobiler Anwendungen. Dazu werden zunächst typische mobile Geschäftsprozesse im Kundendienstbereich analysiert und die daraus resultierenden Anforderungen abgeleitet. Durch Evaluierung werden dann geeignete Technologien identifiziert und zu einer Softwareplattform integriert. Funktionale Lücken werden durch die Konzeption zusätzlicher Komponenten geschlossen. Den kurzen Innovations- und Technologiezyklen wird mit einem iterativen, inkrementellen Vorgehensmodell begegnet.</td></tr>
</table>

Mit der Beendigung des Vorhabens steht eine leistungsfähige Middlewareplattform bereit, die eine schnelle und kostengünstige Entwicklung mobiler Anwendung ermöglicht. Einsatzmöglichkeiten der Plattform ergeben sich insbesondere bei Netzbetreibern, Application Service Providern, Softwareherstellern und IT-Beratungsunternehmen.	
19. Schlagwörter Middleware, Mobile Computing, Framework, Java	
20. Verlag	21. Preis

1. ISBN or ISSN	2. Type of Report Final Report
3a. Report Title MoBiLo: Middleware platform for supporting mobile Logistic Systems utilizing current Middleware, based on Internet- und Open-Source Technologies	
3d. Title of Publication The MoBiLo Platform: Design of a mobile middleware	
4a. Author(s) of the report (Family Name, First Name(s)) Hoffmann, Jürgen Herold Christian	5. End of Project April 2003
4b. Author(s) of the publication (Family Name, First Name(s)) Hoffmann, Jürgen Herold Christian	6. Publication Date June 2003
	7. Form of Publication Report of „Instituts für Wirtschafts- und Verwaltungsinformatik“
8. Performing Institution(s) (Name, Address) Universität Koblenz-Landau Institut für Wirtschafts- und Verwaltungsinformatik Universitätsstr. 1 56075 Koblenz	9. Originator’s Report No.
	10. Reference No. 01AK049
	11a. No. of Pages Report 51 Pages
	11b. No. of Pages Publication 49 Pages
13. Sponsoring Agency (Name, Address) Bundesministerium für Bildung, Forschung und Technologie (BMF) 53170 Bonn	12. No. of References
	14. No. of Tables
	15. No. of Figures
16. Suplementary Notes	
17. Presented at (Title, Place, Date)	

18. Abstract

The Java2 versions J2ME for client-side and J2EE for server-side constitute a powerful platform. But they solely provide an inadequate support for a rapid and competitive development of mobile applications. Complementary technologies (e.g. XML) for supporting mobile applications are available but insufficiently integrated.

The objective of this project is the design of a middleware platform that supports the development of mobile applications. First typical business processes in the area of field service will be analyzed und resulting requirements determined. By the use of evaluation suitable technologies will be identified and integrated to a software platform.

Functional gaps will be filled by the design of additional components. An incremental und iterative approach will cope with the short technical innovation. Reaching the project objectives a powerful middleware platform will be available that allows for the rapid and

competitive development of mobile applications. The platform is especially applicable in the domain of network carriers, application service providers, software vendors and consulting companies.	
19. Keywords Middleware, Mobile Computing, Framework, Java	
20. Publisher	21. Price