Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Stephanie Gamm

Konzeption und Realisierung einer Sprache und Engine für mobile Workflows

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Stephanie Gamm

Konzeption und Realisierung einer Sprache und Engine für mobile Workflows

Masterarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Olaf Zukunft
Zweitgutachter : Prof. Dr. Wolfgang Gerken

Abgegeben am 25. Februar 2008

Stephanie Gamm

Thema der Masterarbeit

Konzeption und Realisierung einer Sprache und Engine für mobile Workflows

Stichworte

Workflows, Geschäftsprozesse, Mobile Computing, Mobilität, Kontextsensitivität, Prozess-Beschreibungssprache, Prozess-Ausführungsumgebung, BPEL

Kurzzusammenfassung

Mobile Workflows unterstützen die Integration mobiler Systeme in Geschäftsprozesse. Als Beschreibungssprache für klassische Geschäftsprozesse im SOA-Umfeld hat sich BPEL in den letzten Jahren etabliert. Im Rahmen dieser Arbeit wird gezeigt, wie sich BPEL und eine geeignete BPEL-Engine um die Unterstützung mobiler Teilnehmer erweitern lassen. Hierfür werden basierend auf den Vorgaben eines Beispiel-Szenarios ein Konzept entwickelt und exemplarisch ausführbare Prozessdefinitionen realisiert. Die entworfenen Erweiterungen ermöglichen zur Laufzeit übertragbare Sub-Prozesse und die Definition von kontextsensitiven Aktivitäten. Durch Letztere kann ein kontextabhängig bestimmter Partner-Service dynamisch in den Prozess eingebunden werden oder die Ausführung einer Aktivität abhängig vom umgebenden Kontext gemacht werden.

Stephanie Gamm

Title of the paper

Concept and implementation of a language and engine for mobile workflows

Keywords

workflows, business processes, Mobile Computing, mobility, Context Awareness, process modelling language, process runtime environment, BPEL

Abstract

Mobile workflows support the integration of mobile systems into business processes. BPEL has found wide acceptance as a language for modelling traditional business processes in a SOA-environment. This thesis demonstrates how to extend BPEL and a BPEL engine so that they support mobile participants. A concept is developed based on the requirements of a representative scenario followed by an implementation of the according executable process definitions. The extensions allow a transfer of sub-processes during runtime and a definition of context aware activities. The latter enables the dynamic integration of partner-services which are selected with respect to the given context into the process. Further it is possible to execute activities depending on the current context.

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

Kapitel 1

Einleitung

Die vorliegende Arbeit ist im Umfeld mobiler Workflows einzuordnen und beschäftigt sich mit der Integration mobiler Geräte in klassische Geschäftsprozesse. Die Untersuchungen konzentrieren sich dabei auf die Konzeption und Realisierung einer Sprache und Engine, die die mit der Mobilität verbundenen Anforderungen unterstützen sollen.

Dieses Kapitel soll einen Einstieg in das Thema geben, wobei Motivation und Zielsetzung der Arbeit vorgestellt werden, und es wird über den Aufbau der Arbeit informieren.

1.1 Motivation

Mobile Geräte wie PDAs und Laptops verbreiten sich zunehmend. Zusätzlich wächst die Leistungsfähigkeit dieser Geräte stetig. Dieser aktuelle Trend fördert den Wunsch nach neuen Anwendungen, die die hinzugewonnene Mobilität unterstützen. Eine Möglichkeit, dieser Entwicklung Rechnung zu tragen, sind mobile Workflows. Dabei handelt es sich um Workflows im Mobile Computing, deren Ziel es ist, einen Geschäftsprozess in Form von einer Abfolge von Aktivitäten kollaborativ und verteilt auf mehreren mobilen (und stationären) Geräten auszuführen.

Neben den zu bewältigenden, eher technischen Eigenschaften mobiler Systeme wie z.B. drahtloser Ad-hoc-Kommunikation gehören speziell die Mobilität und daraus resultierende Anforderungen zu den Herausforderungen im Umfeld mobiler Workflows. Sogenannte kontextsensitive Anwendungen können ihre Umgebung wahrnehmen und auf Veränderungen aufgrund der Mobilität geeignet reagieren.

Die Vielfältigkeit der zu bewältigenden Problemstellungen hat diesem Themengebiet zu aktuellem Forschungsinteresse verholfen. Entgegen klassischen Geschäftsprozessen und Workflow-Management-Systemen hat sich für den Einsatz mobiler Workflows noch kein Standard etabliert.

1.2 Zielsetzung der Arbeit

Den thematischen Hintergrund für diese Arbeit bildet die Integration mobiler Systeme in Geschäftsprozesse. Es wird angestrebt, diese auf Basis einer Service Oriented Architecture umzusetzen und dabei möglichst auf existierende Standards zurückzugreifen. Als Beschreibungssprache in diesem Bereich hat sich die Business Process Execution Language (kurz: BPEL) in den letzten Jahren etabliert. Mit BPEL lassen sich Geschäftsprozesse durch Komposition von Web Services modellieren und die entstandenen Prozessdefinitionen auf einer BPEL-Engine betreiben. BPEL-Prozesse werden von einem zentralen Koordinator in einer feststehenden Infrastruktur ausgeführt, wobei die eingebundenen Services typischerweise stationär sind.

Das Ziel der Masterarbeit besteht darin, BPEL um die Unterstützung mobiler Teilnehmer zu erweitern. Hierfür soll ein Konzept erarbeitet werden, wie der BPEL-Standard um eigene Sprachkonstrukte ergänzt werden kann. Die wichtigsten zu berücksichtigenden Anforderungen sind die Unterstützung von Sub-Prozessen, die zur Laufzeit auf ein anderes System übertragen werden können und dort lokal zur Ausführung kommen, sowie die Möglichkeit der Definition von Laufzeit-Bedingungen, die während der Prozessausführung basierend auf dem aktuellen Kontext berücksichtigt werden sollen. Diese sollen dazu verwendet werden, einen den Kontextvorgaben entsprechenden Service dynamisch in den Ablauf einzubinden oder die Ausführung einer beliebigen Aktivität abhängig vom umgebenden Kontext zu machen.

Der Schwerpunkt der Arbeit liegt auf konzeptioneller Ebene und der Definition der BPEL-Spracherweiterungen. Anhand eines Beispiel-Szenarios sollen die Anforderungen an mobile Workflows aufgezeigt, hierfür die notwendigen Prozesse modelliert sowie realisiert werden. Als Ausführungsumgebung soll unter Beachtung der erweiterten Sprache eine für mobile Workflows geeignete Engine entworfen werden. Abbildung 1.1 veranschaulicht den damit verbundenen Aufbau der Masterarbeit.

1.3 Aufbau der Arbeit

Die Arbeit besteht aus sechs aufeinander aufbauenden Kapiteln. Nach dieser Einleitung werden in Kapitel 2 die Grundlagen vermittelt, die für das weitere Verständnis

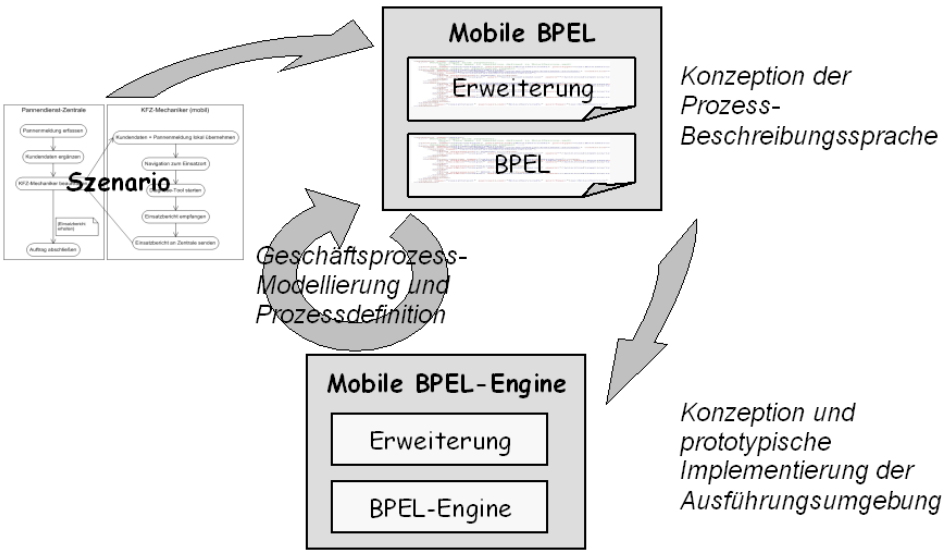

Abbildung 1.1: Aufbau der Masterarbeit

der Arbeit notwendig sind. Neben Begriffsdefinitionen und einer Einführung in die verschiedenen, im Kontext dieser Arbeit relevanten Themengebiete werden die Konzepte der Prozess-Beschreibungssprache BPEL vorgestellt sowie eine thematische Eingrenzung für den weiteren Inhalt vorgenommen.

Kapitel 3 befasst sich mit der Analyse. Zu Beginn wird auf die Erweiterungsmöglichkeiten von BPEL eingegangen und eine BPEL-Engine vorgestellt. Anhand eines Beispiel-Szenarios werden die Anforderungen an die Sprache und Engine aufgestellt, die im weiteren Verlauf zur Unterstützung mobiler Workflows berücksichtigt werden sollen.

Aufbauend auf den Ergebnissen und Vorgaben aus der Analyse wird in Kapitel 4 die Konzeption vorgenommen. Dies beinhaltet die Modellierung der Geschäftsprozesse für das gegebene Szenario, den Entwurf der mobilen BPEL-Spracherweiterungen sowie die Architektur der Engine.

Das Kapitel 5 beschreibt interessante Aspekte der Realisierung, wie sie auf Grundlage der vorhergehenden Konzeption umgesetzt wurde. Hierbei liegt der Schwerpunkt auf der Umsetzung der mobilen Geschäftsprozesse in BPEL-Prozessdefinitionen, wobei die entworfenen mobilen Spracherweiterungen zum Einsatz kommen.

Anschließend wird in Kapitel 6 eine Zusammenfassung der Arbeit vorgenommen. Es folgen eine Bewertung der erzielten Ergebnisse sowie ein Ausblick auf Weiterentwicklungsmöglichkeiten.

Kapitel 2

Grundlagen

Dieses Kapitel beschäftigt sich mit den für das weitere Verständnis der Arbeit notwendigen Grundlagen. Begonnen wird mit einer Definition der Begriffe Geschäftsprozess und Workflow. Daran schließt sich eine Einführung in die Modellierung von ausführbaren Geschäftsprozessen an, wobei u.a. auf Prozess-Beschreibungssprachen – insbesondere BPEL – eingegangen wird. Es folgt ein Überblick über Mobile Computing und damit verbundene Aspekte wie Mobilität und Context Awareness. Im Anschluss an eine Begriffseinführung mobiler Workflows wird für den weiteren Inhalt der Masterarbeit eine thematische Eingrenzung vorgenommen.

Da an dieser Stelle nur auf die wichtigsten Aspekte eingegangen werden kann, wird für darüber hinausgehende Informationen auf die angegebenen Quellen sowie weiterführende Fachliteratur verwiesen.

2.1 Geschäftsprozesse und Workflows

Der Geschäftserfolg eines Unternehmens hängt heute stark von seiner Wirtschaftlichkeit ab. Immer wiederkehrende Abläufe, die in ihrer Gesamtheit die Geschäftsziele verfolgen, werden durch geeignete Formalismen abgebildet, um wirtschaftlich und organisiert arbeiten zu können.

Bei einem *Geschäftsprozess* (engl.: *business process*) handelt es sich nach van der Aalst und van Hee um die Abfolge von Aktivitäten zur Erreichung eines gemeinsamen Ziels [AH-2002]. Ein Geschäftsprozess setzt sich demnach aus einzelnen Aktivitäten und einer Menge an Bedingungen zusammen, die die Reihenfolge der auszuführenden Aktivitäten beschreiben. Dabei wird eine Aktivität als eine logische Einheit verstanden, die von einer Person, einer Maschine oder einer Gruppe von Personen

oder Maschinen ausgeführt wird. Weiterhin kann ein Geschäftsprozess Sub-Prozesse enthalten, die sich ihrerseits wiederum wie Geschäftsprozesse zusammensetzen. Somit lassen sich komplexe Prozesse hierarchisch strukturieren.

Während beim Geschäftsprozess die betriebswirtschaftliche Sichtweise im Vordergrund steht, handelt es sich bei einem *Workflow* um die IT-seitige Repräsentation und Umsetzung eines Geschäftsprozesses [LR-2000]. Systeme der Informationstechnik implementieren die zugrundeliegenden Geschäftsprozesse und führen diese als Workflows aus. Dabei kann der Ablauf in einer verteilten und heterogenen Umgebung sowie unternehmensübergreifend stattfinden.

2.2 Modellierung und Ausführung von Geschäftsprozessen

Heutzutage unterliegen Unternehmen sich rasch verändernden Geschäftsbeziehungen, Strukturen und Abläufen. Entsprechend müssen aktuelle Informationssysteme diese Dynamik unterstützen und sich Veränderungen damit möglichst schnell umsetzen lassen. Aus diesen Gründen hat in den letzten Jahren das Konzept der SOA Einzug in die IT-Landschaft vieler Unternehmen gehalten [NN-2007].

2.2.1 Service Oriented Architecture

Bei der *Service Oriented Architecture* (kurz: *SOA*) handelt es sich um ein abstraktes Konzept einer Software-Architektur, das beinhaltet, wie unterschiedliche Applikationen miteinander agieren können [Mel-2007]. Dazu wird jede einzelne Komponente als ein wiederverwendbarer, offen zugreifbarer *Service* verstanden, der eine plattform- und programmiersprachenunabhängige Nutzung ermöglicht.

Eine SOA zeichnet sich durch ein lose gekoppeltes System von miteinander agierenden, aber voneinander unabhängigen Services aus [NN-2007]. Das Ziel der SOA ist die Integration verschiedener Anwendungen innerhalb eines Unternehmens wie auch unternehmensübergreifend. Dazu lassen sich die einzelnen Services miteinander kombinieren und so zu neuen Funktionalitäten zusammensetzen.

Durch die sogenannte *Komposition von Services* lassen sich beispielsweise Geschäftsprozesse abbilden. Ein entscheidender Vorteil besteht darin, dass durch den Einsatz einer SOA schneller und flexibler auf Veränderungen im Geschäftsumfeld reagiert werden kann. Die einzelnen Services bleiben unverändert; lediglich die übergeordneten Prozesse müssen angepasst werden [Mel-2007].

2.2.2 Web Services

Für die Umsetzung einer Service Oriented Architecture werden sehr häufig *Web Services* verwendet. Darunter versteht man eine Sammlung von Technologien, durch die sich plattform- und herstellerunabhängige, lose gekoppelte, verteilte Anwendungen realisieren lassen [NL-2004].

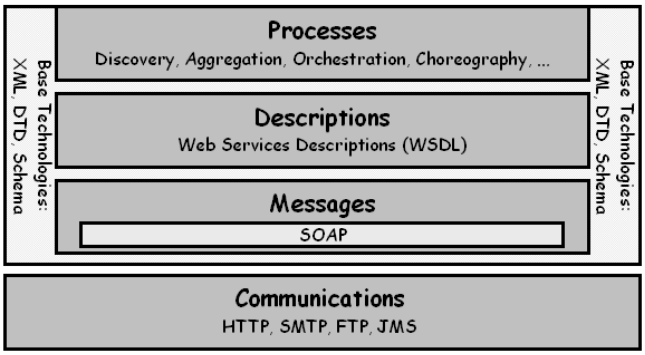

Abbildung 2.1: Web Service Architektur (in Anlehnung an [WS–2004])

Ein wesentliches Ziel der Web Service Architektur ist die Interoperabilität [WS–2004]. Um diese zu erreichen, sind Standards notwendig. Zu den wichtigsten Web Service Standards gehören WSDL, SOAP und XML. Die WSDL (Web Services Definition Language) stellt ein maschinenlesbares Format für die Schnittstellenbeschreibung eines Web Service bereit (mehr zur Spezifikation von WSDL 1.1 unter [WSD-2001]). Die SOAP-Spezifikation beschreibt ein Nachrichtenformat für die Kommunikation mit einem Web Service und dessen Umsetzung auf verschiedene Transportprotokolle. WSDL, SOAP sowie viele weitere Technologien im Web Service-Umfeld basieren auf XML (Extensible Markup Language), so dass diese eine wichtige Grundlage darstellt.

Der Ablauf zur Verwendung eines Web Service lässt sich anhand des in Abbildung 2.2 dargestellten Rollenmodells beschreiben, das sich an den zugrundeliegenden Konzepten einer SOA orientiert [Mel-2007]. Dazu werden die drei Rollen Service Provider, Service Requester und Discovery Service vergeben [RS-2004].

Der Service Provider publiziert seinen Service als WSDL-Dokument bei einem Discovery Service, typischerweise einem UDDI-Verzeichnis (Universal Description, Discovery and Integration). Ein Client, bei dem es sich um einen weiteren Web Service handeln kann, fragt beim Discovery Service die veröffentlichten Services ab und kann anschließend anhand des zur Verfügung gestellten WSDL-Dokuments mit dem ausgewählten Service kommunizieren. Standardmäßig werden hierbei alle Nachrichten über SOAP ausgetauscht. Alternativ kann auf die ersten zwei Schritte verzichtet

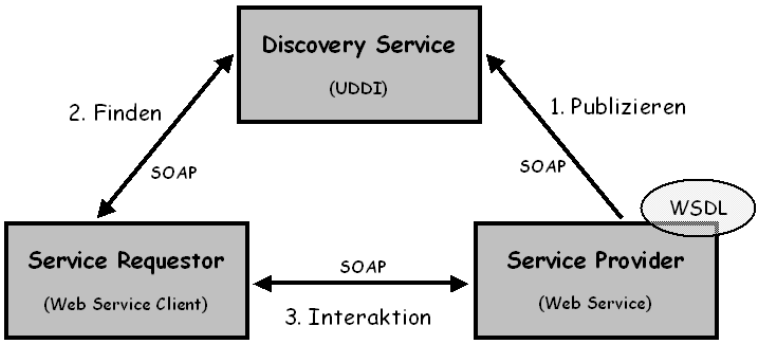

Abbildung 2.2: Web Service Rollenmodell

werden: Wenn sich der Service Requester und der Service Provider beispielsweise aufgrund eines Geschäftsverhältnisses bereits kennen, kann das WSDL-Dokument direkt untereinander ausgetauscht werden, und die Publizierung sowie der Zugriff über den Discovery Service entfallen.

2.2.3 Komposition von Web Services

Wie schon in Abschnitt 2.2.1 erwähnt, lassen sich mehrere Services zu neuen Anwendungen zusammensetzen, beispielsweise zu Geschäftsprozessen. Man spricht hier von der *Komposition von Web Services* [RS-2004]. Abbildung 2.1 zeigt, wie sich die resultierenden Prozesse in die Web Service Architektur eingliedern.

Werden bei der Geschäftsprozess-Modellierung durch Komposition einzelner Web Services traditionelle Programmiersprachen angewendet, kommt es naturgemäß zu einer Vermischung protokollspezifischer Aspekte mit der eigentlichen Geschäftslogik. Aus diesem Grund haben sich in den letzten Jahren Ansätze für die Service-Komposition entwickelt, die sich an den Konzepten herkömmlicher Workflow-Management-Systeme (kurz: WfMS) orientieren [RS-2004]. Mittels Programmiermodellen auf höherer Abstraktionsebene lässt sich die eigentliche Prozesslogik, die die Ausführungsreihenfolge der einzelnen Aktivitäten beschreibt, von der zugrundeliegenden Implementierung der beteiligten Web Services trennen.

Einen möglichen Ansatz für die Komposition stellt die *Web Service Orchestration* dar. Hierbei wird die Logik für einen Geschäftsprozess aus der Sicht eines konkreten Teilnehmers beschrieben [JMS-2004]. Dieser definiert die Geschäftslogik und Reihenfolge der auszuführenden Aktionen und legt das Zusammenspiel der beteiligten Web Services fest, wie Abbildung 2.3 für einen sequentiellen Beispielablauf zeigt. Die darin verwendeten Aktivitäten ”Receive”, ”Invoke” und ”Reply” werden in Abschnitt 2.3.2 vorgestellt. Den involvierten Services ist dabei nicht bewusst, dass sie

Teil eines zusammengesetzten Geschäftsprozesses sind. Das Ergebnis der Orchestration ist ein ausführbarer Prozess mit zentraler Koordination.

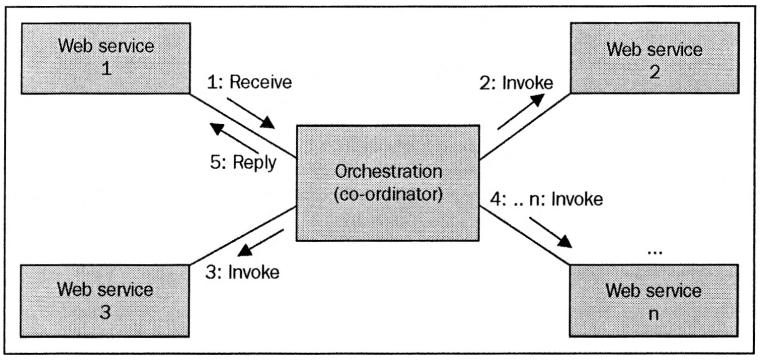

Abbildung 2.3: Web Service Orchestration [JMS-2004]

Für die Modellierung von Geschäftsprozessen im Rahmen der Web Service Orchestration wird eine Sprache zur Prozess-Beschreibung benötigt [JMS-2004]. Damit lassen sich formale Definitionen konkreter Prozesse vornehmen, die den internen Ablauf festlegen. Die Interpretation und Ausführung dieser Prozessdefinitionen geschieht anschließend mit Hilfe einer Prozess-Ausführungsumgebung, wobei es sich um eine spezielle Workflow-Engine, eine sogenannte Orchestration Engine, handelt.

2.2.4 Prozess-Beschreibungssprachen für Web Service Orchestration

In den letzten Jahren sind verschiedene Sprachen im Bereich der Web Service Orchestration entwickelt worden. Zu den ersten Ansätzen gehören die blockbasierte, algebraische Prozessmodellierungssprache XLANG, die im Jahr 2000 von Microsoft veröffentlicht wurde, und die graphbasierte Web Services Flow Language (kurz: WSFL), eine Entwicklung von IBM aus dem Jahr 2001 [Pel-2003]. Beide unterstützen die Interaktion verschiedener Web Services in Form von Geschäftsprozessen.

Aus einer Zusammenführung von XLANG und WSFL ist die *Business Process Execution Language* (kurz: *BPEL*) entstanden [Pel-2003]. BPEL vereint beide Ansätze und bietet als XML-basierte Sprache ein umfangreiches Vokabular zur Beschreibung von Geschäftsprozessen durch Komposition von Web Services an.

In der Version 1.0 ist BPEL im August 2002 als Entwicklung von IBM, Microsoft und BEA unter dem Namen *Business Process Execution Language for Web Services* (kurz: BPEL4WS) veröffentlicht worden. Für die Entwicklung der Version 1.1 stießen bis März 2003 SAP und Siebel hinzu. Im April 2003 wurde die BPEL

Version 1.1 [BPE-2003] an OASIS übergeben und dort das *Web Services Business Process Execution Language Technical Committee* (kurz: WSBPEL TC) gegründet. Dem WSBPEL TC sind seitdem weitere Unternehmen beigetreten, um an der Weiterentwicklung und Standardisierung von BPEL mitzuarbeiten, was die Akzeptanz von BPEL in der Industrie weiter verstärkt hat. Aktuell liegt BPEL in der Version 2.0 vor und ist mit der Spezifikation vom 11.04.2007 zu einem offiziellen OASIS-Standard ernannt worden [BPE-2007]. Mit dieser Version wurde BPEL4WS nach *WS-BPEL* (*Web Services Business Process Execution Language*) umbenannt.

Die Business Process Modeling Language (kurz: BPML) aus dem Jahr 2001 ist eine Meta-Sprache zur Beschreibung von Geschäftsprozessen mit einer BPEL-ähnlichen Syntax. Neben ausführbaren Prozessen werden abstrakte Prozessbeschreibungen unterstützt [JMS-2004]. Die Bedeutung von BPML hat nachgelassen, da es mit BPEL eine neuere Sprache gibt, die die Vorzüge von XLANG und WSFL in sich vereint und die ebenfalls auf den Basis-Technologien der Web Service Architektur aufbaut.

Die Sprache BPEL wird zunehmend im unternehmerischen Umfeld eingesetzt und findet darüber hinaus große Beachtung in diversen Forschungsarbeiten. Aus den genannten Gründen wurde für die vorliegende Arbeit BPEL als Beschreibungssprache ausgewählt, die als Basis für die eigenen Erweiterungen dienen soll. Der folgende Abschnitt 2.3 wird eine kurze Einführung in BPEL geben.

2.3 BPEL – Business Process Execution Language

Für die weiteren Untersuchungen und Ergebnisse im Rahmen dieser Arbeit soll die derzeit aktuelle, standardisierte Version WS-BPEL 2.0 als Beschreibungssprache verwendet werden [BPE-2007]. Im Vergleich zu den 1.x-Versionen wurden bei WS-BPEL 2.0 unter anderem Verbesserungen und Syntax-Umbenennungen vorgenommen, wodurch diese Version nicht abwärtskompatibel ist. Wenn nicht anders angegeben, beziehen sich alle im Folgenden mit ”BPEL” bezeichneten Formulierungen auf diese Version. Die Vorstellung der zugrundeliegenden Konzepte und des Aufbaus von BPEL-Prozessen wird angelehnt an [BPE-2007], [JMS-2004] und [Win-2007] vorgenommen. Für darüber hinaus gehende Informationen wird auf die BPEL-Spezifikation und externe Fachliteratur verwiesen.

Als Prozess-Beschreibungssprache ist BPEL direkter Bestandteil der Web Service Architektur und setzt auf die Schnittstellenbeschreibungssprache WSDL 1.1 auf, wie zuvor in Abbildung 2.1 gezeigt. BPEL setzt voraus, dass alle Web Services, auf die im Prozessverlauf zugegriffen wird, durch WSDL-Dokumente beschrieben sind. Aus WSDL-Nachrichten und XML Schema Typ-Definitionen setzt sich das von BPEL-Prozessen verwendete Datenmodell zusammen. Es beschreibt die Informationen, die

zwischen den Service-Aufrufen ausgetauscht werden. Für die interne Manipulation solcher Daten zur Laufzeit, zum Beispiel zur Weitergabe eines Zwischenergebnisses an den nächsten Service, werden XPath 1.0 und XSLT 1.0 verwendet.

Neben den erwähnten ausführbaren Prozessen lassen sich mit BPEL auch abstrakte Prozesse beschreiben. Letztere sind nicht ausführbar. Abstrakte Prozessbeschreibungen werden zur Definition des öffentlichen, d.h. von außen sichtbaren Nachrichtenaustauschs zwischen Prozessteilnehmern verwendet. Der interne Ablauf bei den konkreten Teilnehmern bleibt verborgen. Für die vorliegende Arbeit sind ausschließlich ausführbare Prozesse relevant.

2.3.1 Ausführbare BPEL-Prozesse

Ein ausführbarer BPEL-Prozess beschreibt aus der Sicht eines Teilnehmers den gesamten internen Ablauf eines Geschäftsprozesses durch Kombination einzelner Web Service-Aufrufe. Die resultierende Prozessdefinition wird durch Deployment auf einer BPEL-Engine zur Ausführung gebracht. Ein solcher Prozess stellt selbst wieder einen Web Service dar, der ebenfalls durch ein WSDL-Dokument beschrieben wird. Dadurch lassen sich auf Basis von Prozessbeschreibungen verschiedene Abstraktionsebenen schaffen, und die verschiedenen, selbst als Service fungierenden Prozesse miteinander kombinieren. (Der BPEL-Standard beinhaltet jedoch keine explizite Unterstützung von Sub-Prozessen; IBM und SAP haben hierzu in einem Whitepaper Erweiterungsmöglichkeiten für BPEL vorgestellt [IBM-2005].)

Ein deployter BPEL-Prozess wird gestartet, indem ein Client die über WSDL ausgeschriebene Start-Operation des den Prozess repräsentierenden Web Service aufruft. Dadurch angestoßen ruft der BPEL-Prozess seinem Ablauf entsprechend alle involvierten Web Services auf. Wenn vorgesehen, wird dabei auch eine Antwortnachricht an den aufrufenden Client zurückgesendet. Grundsätzlich unterstützt BPEL synchrone wie asynchrone Web Service Operationen. Abbildung 2.4 zeigt den schematischen Ablauf eines beispielhaften BPEL-Prozesses.

Konzept der BPEL Partner Links

Der jeweils angegebene Port Type ist Bestandteil von WSDL und stellt die abstrakte Beschreibung einer vom Web Service unterstützten, operationsbasierten Interaktion dar. Der BPEL-Standard setzt auf dieser abstrakten Servicebeschreibung auf und verlangt, dass für jede Verbindung, die der Prozess von und nach außen vorsieht, sogenannte *Partner Links* definiert werden. Dabei wird jedem Partner Link die Rolle zugeteilt, die er in dem jeweiligen Geschäftsprozess einnimmt. Somit ist der BPEL-

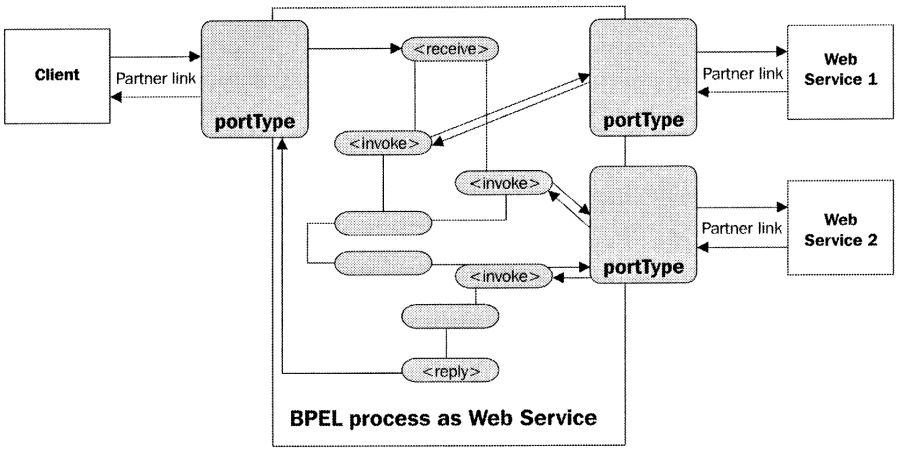

Abbildung 2.4: Beispielhafter BPEL-Prozess als Web Service [JMS-2004]

Prozess in der Lage zwischen den Kommunikationspartnern zu unterscheiden und einen Aufruf nur entgegenzunehmen, wenn der Partner zu dieser Operation berechtigt ist.

Ein weiterer Vorteil dieses Konzepts besteht darin, dass zum Zeitpunkt der Prozessdefinition die abstrakten Servicebeschreibungen als WSDL-Dokumente für jeden Partner-Service ausreichend sind. Die konkrete Servicebeschreibung und das Binding kann erst zum Deployment oder sogar erst zur Laufzeit nachgereicht werden.

2.3.2 Sprachelemente von BPEL

Mit BPEL wird die exakte Reihenfolge festgelegt, in der die am Prozessablauf beteiligten Web Services aufgerufen werden. Die wichtigsten Elemente für einen BPEL-Prozess bilden die *Aktivitäten* (engl.: activities). Dabei unterscheidet BPEL zwischen den sogenannten Basic Activities und den Structured Activities.

Bei den *Basic Activities* handelt es sich um Grundbausteine, die eine einfache Basisfunktionalität realisieren. Hierzu zählen u.a. die Aktivitäten, mittels derer der Prozess mit Partner-Services kommuniziert. Die folgende Auflistung stellt die wichtigsten Basic Activities vor:

- `<receive>` zum Empfangen einer Nachricht, z.B. der Startnachricht zu Beginn eines Prozesses
- `<invoke>` zum Aufrufen einer synchronen Operation eines anderen Web Service beziehungsweise zum Übersenden einer Antwortnachricht bei einer asynchronen Operation

- *<reply>* zum Senden einer Antwortnachricht als Reaktion auf einen synchronen Operationsaufruf
- *<assign>* zum Zuweisen von Daten an Variablen oder an die Parameter eines Service-Aufrufs
- *<throw>* zum Anzeigen eines Fehlers
- *<exit>* zum sofortigen Beenden der gesamten Prozessinstanz

Structured Activities koordinieren dagegen den logischen Prozessablauf. Mittels dieser Aktivitäten lässt sich eine Menge von Basic und Structured Activities miteinander kombinieren, indem man die Reihenfolge und Bedingungen für deren Ausführung definiert. Structured Activities stellen quasi die Programmierlogik in BPEL dar. Eine Auswahl der wichtigsten enthält die folgende Auflistung:

- *<sequence>* zur Definition einer Menge an Aktivitäten, die nacheinander ausgeführt werden
- *<flow>* zur Definition einer Menge an Aktivitäten, die parallel ausgeführt werden
- *<if>* zur Implementierung einer Verzweigung durch bedingtes Verhalten
- *<while>* zum Definieren von Schleifen für wiederholtes Ausführen einer Aktivität
- *<pick>* zum Definieren von Ereignissen, wie Nachrichteneingang oder Timerablauf, und zugehörigen Aktivitäten, die im Fall des eingetroffenen Ereignisses ausgeführt werden

Eine BPEL-Prozessdefinition setzt sich aus den verschiedenen XML-Elementen zusammen, die den Regeln entsprechend ineinander verschachtelt werden können. Das Root-Element jedes BPEL-Dokuments ist das Element *<process>*. Darin enthalten ist normalerweise ein Top-Level-<sequence>-Element, in dem auf die eingehende Anfrage eines Clients gewartet wird, um den Prozessablauf zu starten. Listing 2.1 zeigt den grundsätzlichen Aufbau einer BPEL-Prozessdefinition mit einigen Beispielaktivitäten. Auf eine vollständige Syntax der einzelnen Elemente wird an dieser Stelle verzichtet.

```
<process ...>
   ...
   <partnerLinks>
      <partnerLink .../>
      ...
   </partnerLinks>

   <variables>
```

```
    <variable .../>
    ...
  </variables>

  <sequence>
    <!-- Wait for the incoming request to start the process -->
    <receive name="receiveInitialRequest" .../>

    <!-- Invoke a set of related synchronous web service
       operations, concurrently -->
    <flow>
      <invoke name="invoke-1" .../>
      <invoke name="invoke-2" .../>
    </flow>

    <!-- Invoke an asynchronous operation -->
    <invoke name="invoke-3" .../>
    <!-- Do something else... -->
    <!-- Wait for the asynchronous call-back -->
    <receive name="receive-3" .../>

    <!-- Return a synchronous reply to the caller (client) -->
    <reply name="replyToCaller" .../>
  </sequence>

</process>
```

Listing 2.1: Beispiel für grundlegenden Aufbau eines BPEL-Prozesses

Listing 2.1 enthält neben den bisher vorgestellten BPEL-Elementen einen Block zur Deklaration von Variablen. Diese können später im Prozessverlauf verwendet werden, um beliebige Daten zu speichern, zum Beispiel bei einem Nachrichteneingang. Variablen repräsentieren den internen Zustand eines Prozesses.

Weiterhin unterstützt BPEL das Konzept der Scopes. Jeder Scope bildet einen eigenen Kontext für die darin enthaltenen Aktivitäten und Variablen. Dadurch lässt sich ein Prozess in kleinere, überschaubare Einheiten unterteilen. Der BPEL-Standard bietet auch verschiedene Möglichkeiten zur Ereignis- und Fehlerbehandlung mittels sogenannter Handler an. Die vorgestellten Konzepte und BPEL-Elemente bilden die Basis für das Erstellen eines ausführbaren Prozesses. Der BPEL-Standard enthält weitere Konstrukte, die hier jedoch nicht näher erläutert werden.

2.4 Mobile Computing

Neben der Modellierung von Geschäftsprozessen und deren Ausführung als Workflows durch Web Service Orchestration zählt ein weiterer Themenkomplex zu den Grundlagen der vorliegenden Arbeit. Dieser Abschnitt soll eine Einführung in das *Mobile Computing* und damit verbundene Fragestellungen geben.

Durch zunehmende Verbreitung und wachsende Leistungsfähigkeit verändern mobile, tragbare Geräte wie PDAs oder Laptops die an Softwaresysteme gestellten Anforderungen. Im Vergleich zu stationären Computersystemen verfügen mobile Geräte über geringe Ressourcen und eine vielerorts zugängliche, wenngleich auch keine verlässliche Netzinfrastruktur. Mit den Fragestellungen, die aus dieser Entwicklung resultieren, beschäftigt sich das Mobile Computing [Sat-1996].

Die Untersuchungen beziehen sich beispielsweise auf die Verteilung mobiler Systeme und die Kommunikation untereinander, die typischerweise drahtlos stattfindet. Ein häufig betrachteter Aspekt sind Ad-hoc-Netzwerke und die Unterstützung unterschiedlicher Kommunikationskanäle, wie WLAN oder Bluetooth [Rot-2005]. Bei Projekten im Umfeld des Mobile Computing spielen oftmals die eingeschränkte Leistung mobiler Hardware wie Speicher und Stromversorgung oder die veränderten Anforderungen bezüglich der Ergonomie eine zentrale Rolle [Sat-1996].

Das Mobile Computing dient gern als Oberbegriff für eine Vielzahl an Konzepten und Forschungsinteressen, die sich durch den Einsatz mobiler Systeme ergeben. Auf einige verwandte Begriffe soll an dieser Stelle näher eingegangen werden, da sie für den Hintergrund dieser Arbeit eine wesentliche Rolle spielen.

2.4.1 Mobilität

Neben den bisher erwähnten, zumeist technischen Aspekten des Mobile Computing eröffnet ein weiterer Gesichtspunkt eine Vielzahl neuer Möglichkeiten: Die hinzugewonnene *Mobilität* stellt neue Anforderungen an die Entwicklung von Software-Anwendungen. Aus diesem Grund beschäftigen sich zahlreiche Projekte speziell mit diesem Thema, wobei unterschiedliche Ausprägungen der Mobilität Untersuchungsgegenstand sein können.

Beispielsweise unterscheiden Murphy et al. zwischen den beiden Formen der physikalischen und logischen Mobilität [MPR-2001]. Dabei definiert *physikalische Mobilität* die Bewegung von mobilen Einheiten im physikalischen Raum, wie die Positionsänderung eines PDA. Die *logische Mobilität* bezeichnet die Bewegung von mobilen Einheiten im logischen Raum, wie die Übertragung von Code oder Zuständen auf

ein anderes Gerät. Während besonders die logische Mobilität neue Anforderungen an die Architektur und den Entwurf von Software-Anwendungen stellt, müssen in Verbindung mit der physikalischen Mobilität zumeist technische Herausforderungen überwunden werden.

2.4.2 Context Awareness

Aus der Mobilität resultiert eine sich verändernde Umgebung, denen die Systeme und ihre Anwendungen ausgesetzt sind. Diesen Sachverhalt kann man sich zunutze machen, indem die Systeme ein Bewusstsein für die Mobilität entwickeln und ihre aktuelle, individuelle Umgebung wahrnehmen. Man spricht hier von der Eigenschaft der *Context Awareness* [SAW-1994].

Context Awareness – zu deutsch häufig als *Kontextsensitivität* bezeichnet – ist ein oft verwendeter Begriff in Verbindung mit den verschiedenen Forschungsrichtungen des Mobile Computing. Dem zugrunde liegt der Begriff des *Kontext*s. Als Ergebnis des Vergleichs bekannter Definitionen liefert Dey hierfür folgende Bestimmung:

> *"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."* [Dey-2001]

Demnach beschreibt Kontext jegliche Information bezüglich der Situation einer Entität. Typischerweise werden diese Informationen über Sensoren ermittelt. Diese liefern aktuelle Daten bezüglich Ort, Zeit, Ressourcen oder beliebig anderer Eigenschaften.

Kontextsensitive Anwendungen müssen zwei grundlegenden Anforderungen genügen. Zum einen müssen sie in der Lage sein, Änderungen in der Umgebung zu erkennen und als Kontext zu erfassen. Zweitens muss die Kontextänderung ausgewertet und an die jeweilige Situation angepasst darauf reagiert werden [MPR-2001]. Typischerweise unterliegt die Kontextänderung einer hohen Dynamik, deren Zeitpunkt und Ausprägung unvorhersehbar sind.

Bei kontextsensitiven Anwendungen, die heute verfügbar sind, handelt es sich meist um ortsabhängige Systeme, die ihre Position zum Beispiel über GPS beziehen. Darüber hinaus existieren kaum Anwendungen, die umfassendere Kontextinformationen berücksichtigen. Bei Projekten auf diesem Gebiet handelt es sich noch größtenteils um Forschung.

2.4.3 Ubiquitous Computing

Der Begriff *Ubiquitous Computing* wurde von Mark Weiser, damals leitender Wissenschaftler am Xerox Palo Alto Research Center, geprägt und erstmals im Jahr 1991 in seinem visionären Aufsatz "The Computer for the 21st Century" beschrieben [Wei-1991]. Ubiquitous Computing beschreibt die Allgegenwärtigkeit von Rechnern und der damit verbundenen, allgegenwärtigen IT-Infrastruktur.

Die Vision basiert darauf, dass Computer immer kleiner werden und in beliebige Alltagsgegenstände integriert werden. Das klassische Bild vom Desktop-System wandelt sich zu kaum wahrnehmbaren, überall anzutreffenden, intelligenten Gegenständen. Ausgestattet mit Sensoren nehmen die Gegenstände ihre Umgebung wahr und bilden mittels Vernetzung ein ständig verfügbares, informationsverarbeitendes System [Mat-2003]. Obwohl die Vision des Ubiquitous Computing den Einzug der Informationstechnologie in sämtliche Bereiche unseres Alltags beinhaltet und uns beim alltäglichen Handeln unterstützen soll, tritt die eigentliche Technik für den Anwender in den Hintergrund.

Diese von Weiser geprägte Vorstellung vom Ubiquitous Computing zielt auf eine "unaufdringliche, humanzentrierte Technikvision [...], die sich erst in der weiteren Zukunft realisieren lässt" [Mat-2003]. Davon abgeleitet hat sich mittlerweile der Begriff des Pervasive Computing etabliert, der im folgenden Abschnitt näher definiert wird.

2.4.4 Pervasive Computing

Im Gegensatz zum Ubiquitous Computing bezieht sich der später von der Industrie geprägte Begriff des *Pervasive Computing* vor allem auf kurzfristig realisierbare Lösungen, wobei es auch hier um die überall verfügbare, allgegenwärtige Informationsverarbeitung geht. Pervasive Computing beschreibt wörtlich die Durchdringung von Rechnern. Primär zielt das Interesse beim Pervasive Computing dahin, die heute technisch möglichen Ideen des Ubiquitous Computing in E-Commerce-Szenarien und webbasierte Geschäftsprozesse einfließen zu lassen [Mat-2003].

Das Bundesamt für Sicherheit in der Informationstechnik hat im Jahr 2006 eine Studie zum Pervasive Computing veröffentlicht [BSI-2006]. Abbildung 2.5 gibt die darin aufgezeigten, in den nächsten zehn Jahren zu erwartenden Trends, Entwicklungen und Abhängigkeiten des Pervasive Computing wieder. Es wird erwartet, dass die Entwicklungsziele des Pervasive Computing in verschiedenen Stufen erreicht werden. Die auf etwa die ersten fünf Jahre geschätzte Stufe wird hauptsächlich die Aspekte Mobilität, Ad-hoc-Vernetzung und Kontextsensitivität vorantreiben.

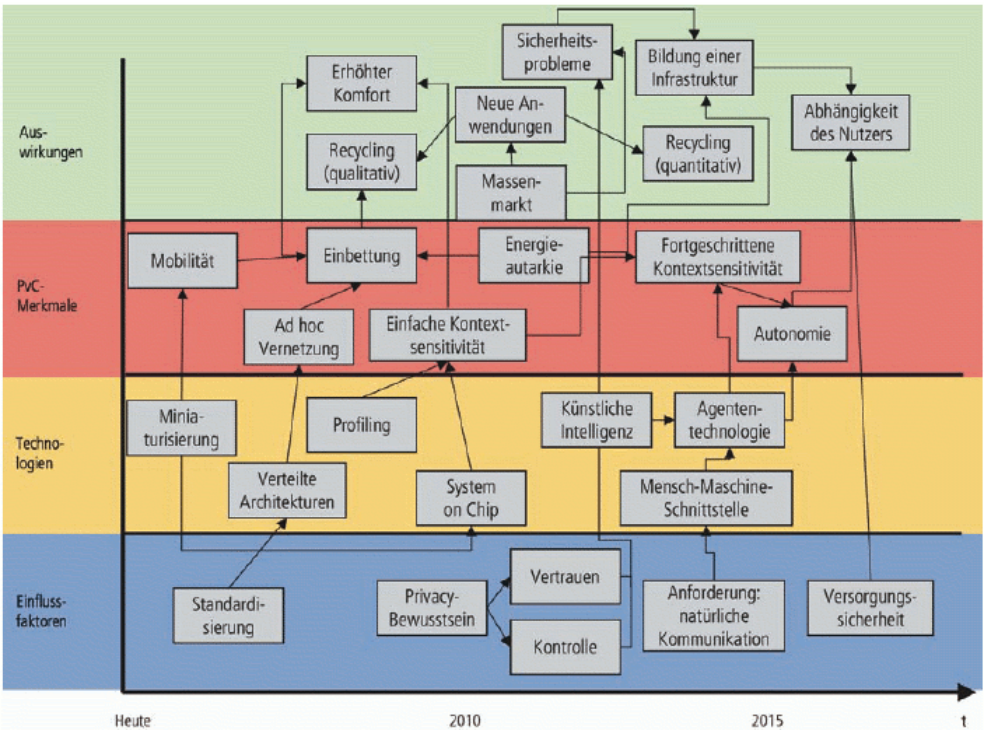

Abbildung 2.5: Zentrale Trends, Entwicklungen und Abhängigkeiten des Pervasive Computing [BSI-2006]

2.5 Mobile Workflows

Mobile Prozesse bzw. *mobile Workflows* sind die Fortführung klassischer Geschäftsprozesse im Mobile Computing. Sie behandeln die verteilte Ausführung eines Prozesses auf mobilen (und stationären) Geräten und versuchen, sich das Bewusstsein für ihre Mobilität dabei zunutze zu machen. Neben der reinen Unterstützung der Mobilität können mobile Workflows davon profitieren, wenn sie kontextsensitiv auf Veränderungen ihrer Umgebung reagieren.

Die bewährten Konzepte klassischer Geschäftsprozesse genügen den neuen Anforderungen nicht, so dass sich mobile Prozesse zu einem Themengebiet mit aktuellem Forschungsinteresse entwickelt haben. Projekte in diesem Bereich erhoffen sich die Ausführung von vorher unbekannten Anwendungen auf mobilen Geräten und in allgegenwärtiger Infrastruktur, wodurch man der Vision des Pervasive Computing einen Schritt näher gekommen wäre [Kun-2005], [Zap-2005].

Typischerweise ist der mobile Mitarbeiter heutzutage noch nicht in die Abbildung betrieblicher Prozesse durch IT-Unterstützung integriert. Diese Tatsache führt zu

einem Bruch in einem eigentlich zusammenhängenden Ablauf. Zur Verbesserung der Geschäftsprozesse sollten mobile Arbeitsplätze daher in die betrieblichen Prozesse einbezogen werden und dieses in den zugrundeliegenden Workflow-Systemen Berücksichtigung finden.

Entgegen klassischen Geschäftsprozessen und Workflow-Management-Systemen hat sich für den Einsatz mobiler Prozesse noch kein Standard etabliert. Auf Eigenschaften mobiler Workflows, die über diese generelle Begriffseinführung hinausgehen, und die damit verbundenen Anforderungen an eine geeignete Prozess-Beschreibungssprache und Engine wird in der sich anschließenden Analyse näher eingegangen (speziell Abschnitte 3.4 und 3.5).

2.6 Einordnung der Arbeit

Die bisher beschriebenen Grundlagen bilden den thematischen Rahmen für die vorliegende Arbeit. Es sollen eine Sprache und Engine entwickelt werden, die mobile Workflows unterstützen. Angelehnt an die Konzepte der Service Oriented Architecture soll die Prozess-Beschreibungssprache BPEL als Basis dienen. Mit BPEL erreicht man bereits eine hohe Flexibilität bei der Modellierung von Geschäftsprozessen, jedoch bringt die Sprache keinerlei Mobilitäts-Unterstützung mit. Standardmäßig wird ein BPEL-Prozess zentral auf einer stationären Engine ausgeführt; die eingebundenen Web Services werden typischerweise ebenfalls von stationären Systemen angeboten.

Bei mobilen Workflows im Kontext dieser Arbeit handelt es sich um eine Verbindung von Web Service Orchestration mit den Konzepten des Mobile Computing. Das bedeutet, mittels BPEL sollen beliebige Web Services zu neuen, ausführbaren mobilen Prozessen kombiniert werden, wobei BPEL-Spracherweiterungen zu entwerfen sind, die speziell Mobilität und Context Awareness unterstützen. Eine BPEL-Prozessdefinition mit mobilen Erweiterungen soll anschließend auf mobilen Geräten von einer geeigneten Engine ausgeführt werden können.

In Abschnitt 2.4.1 wurden die Begriffe physikalische Mobilität und logische Mobilität eingeführt. Beide Ausprägungen werden im Rahmen dieser Arbeit untersucht. Konkret bezieht sich die physikalische Mobilität hierbei auf die Bewegung eines Mitarbeiters mit seinem Laptop, auf dem ein mobiler BPEL-Workflow ausgeführt wird. Die sich dadurch ergebenden Kontextänderungen werden vom mobilen Workflow berücksichtigt. Die logische Mobilität bezeichnet beispielsweise die Übertragung von Programmcode auf ein anderes System. Diese Art der Mobilität ist ebenfalls Inhalt der vorliegenden Arbeit; im weiteren Verlauf wird darauf genauer eingegangen.

Die erwähnten Fragestellungen des Mobile Computing bezüglich der technischen Infrastruktur sind nicht Teil dieser Arbeit. Der Schwerpunkt für die weitergehende Betrachtung liegt auf der konzeptionellen Definition der BPEL-Spracherweiterungen und der Anwendungs-Architektur für die Engine.

Kapitel 3

Analyse

In diesem Kapitel wird vorgestellt, welche Konzepte BPEL für die Entwicklung eigener Sprachelemente unterstützt, sowie eine geeignete BPEL-Engine präsentiert, die im weiteren Verlauf der Arbeit verwendet werden soll. Im folgenden Teil der Analyse werden ein Beispiel-Szenario beschrieben und die Anforderungen herausgearbeitet, die zur Unterstützung mobiler Workflows an eine Prozess-Beschreibungssprache und Engine gestellt werden.

3.1 BPEL als Entwicklungsbasis für eigene Sprache

Für die Definition von Geschäftsprozessen, die mittels Web Service Orchestration realisiert werden sollen, ist der Einsatz einer Beschreibungssprache notwendig (vgl. Abschnitt 2.2.3). Im Rahmen dieser Arbeit wurde sich für die Sprache BPEL entschieden, wie bereits in Abschnitt 2.2.4 dargestellt. Durch die Verwendung von Standards wie BPEL und WSDL wird die Möglichkeit der Wiederverwendung bestehender Services und Prozesse ermöglicht. BPEL erfüllt die Anforderungen zur Modellierung klassischer Geschäftsprozesse (vgl. Abschnitt 3.5.1), so dass der existierende Sprachumfang um Konstrukte erweitert werden soll, die auf die Bedürfnisse mobiler Prozesse ausgelegt sind. Der folgende Abschnitt zeigt auf, inwiefern der BPEL-Standard hierfür eine geeignete Basis darstellt.

3.1.1 Erweiterbarbeit von BPEL

Bei der Standardisierung von WS-BPEL 2.0 wurde auf die individuelle Erweiterbarkeit der Sprache geachtet. Bei den Erweiterungen kann es sich um neue Attribute

oder Elemente handeln. Ebenso können Zuweisungsoperationen und Aktivitäten ergänzt werden, um neue Funktionalitäten zu realisieren. Dabei dürfen die Erweiterungen die Semantik von bestehenden BPEL-Elementen nicht verändern. Erweiterungen müssen in einem eigenen Namensraum definiert werden. Anschließend können sie hierüber qualifiziert und in BPEL-eigene Elemente integriert werden. Listing 3.1 gibt die Syntax der Elemente *<extensions>* und *<extension>* unter Verwendung der im BPEL-Standard festgelegten Konventionen wieder, mittels derer die Einbindung von Erweiterungen in einen BPEL-Prozess vorgenommen wird.

```
<process ...>
   <extensions>?
      <extension namespace="anyURI" mustUnderstand="yes|no" />+
   </extensions>
   ...
</process>
```

Listing 3.1: Syntax des BPEL-Elements `<extensions>` [BPE-2007]

Den Namensraum der Erweiterungen gibt man als `namespace`-Wert eines `<extension>`-Elements an. Zusätzlich muss der Parameter `mustUnderstand` gesetzt werden; hierüber wird gesteuert, ob die entsprechenden Erweiterungen von einer den Prozess ausführenden BPEL-Engine verstanden werden müssen. Für den Fall, dass `mustUnderstand="yes"` gesetzt ist und die Engine mindestens eine der eingebundenen Erweiterungen nicht unterstützt, muss der entsprechende Prozess komplett abgelehnt werden. Erweiterungen, bei denen `mustUnderstand="no"` gesetzt ist, sind optional, d.h. die BPEL-Engine kann deren Ausführung ignorieren.

BPEL unterstützt weiterhin das Element *<extensionActivity>*. Es handelt sich hierbei um eine Aktivität und dient dazu, den Sprachumfang von BPEL um eine neue, nicht im Standard definierte Aktivität erweitern zu können. Diese Möglichkeit soll angewendet werden, um die im Rahmen dieser Arbeit geplanten Erweiterungen für mobile Geschäftsprozesse in die Sprache zu integrieren. Der BPEL-Standard schreibt vor, dass das Element `<extensionActivity>` selbst genau ein XML-Element enthalten muss. Dieses muss wie die meisten BPEL-Elemente Standard-Attribute und -Elemente unterstützen. Listing 3.2 zeigt die Syntax für das BPEL-Element `<extensionActivity>`.

```
<extensionActivity>
   <anyElementQName standard-attributes>
      standard-elements
   </anyElementQName>
</extensionActivity>
```

Listing 3.2: Syntax des BPEL-Elements `<extensionActivity>` [BPE-2007]

Die Syntax für die im Element `<extensionActivity>` referenzierten *`standard-attributes`* stellt Listing 3.3 dar. Für die Attribute `name` und `suppressJoinFailure` sind Default-Werte definiert, falls beim zugehörigen BPEL-Element keine expliziten Angaben gemacht werden: Das Attribut `name` bleibt unbesetzt, während das Attribut `suppressJoinFailure` automatisch denselben Wert von der direkt umschließenden Aktivität oder, falls auch dort nicht angegeben, vom Prozess selbst erhält.

```
name="NCName"? suppressJoinFailure="yes|no"?
```

Listing 3.3: Syntax der BPEL-Standardattribute [BPE-2007]

Die ebenfalls im Element `<extensionActivity>` referenzierten *`standard-elements`* bestehen aus den Elementen `<targets>` und `<sources>`. Beide Elemente sind optional. Ihre Syntax entspricht Listing 3.4.

```
<targets>?
   <joinCondition expressionLanguage="anyURI"?>?
      bool-expr
   </joinCondition>
   <target linkName="NCName" />+
</targets>
<sources>?
   <source linkName="NCName">+
      <transitionCondition expressionLanguage="anyURI"?>?
         bool-expr
      </transitionCondition>
   </source>
</sources>
```

Listing 3.4: Syntax der BPEL-Standardelemente [BPE-2007]

Für das Verständnis der weiteren Arbeit ist lediglich der beschriebene Aufbau des `<extensionActivity>`-Elements von Bedeutung; für weitergehende Informationen bezüglich der einzelnen Konstrukte wird auf die BPEL-Spezifikation verwiesen [BPE-2007].

Der vom BPEL-Standard angebotene Erweiterungsmechanismus bietet, wie hier beschrieben, eine gute Voraussetzung, um eigene Sprachelemente zur Unterstützung mobiler Workflows in den bestehenden Sprachumfang zu integrieren. Die sich in Abschnitt 4.3 anschließende Konzeption der Sprache baut daher auf den hier vorgestellten Eigenschaften auf.

3.1.2 BPEL-Engines

Da mit BPEL ausführbare Prozesse definiert werden sollen, wird eine Engine benötigt, die diese BPEL-Dokumente auswerten kann. Die Engine muss neben den im BPEL-Standard definierten Sprachelementen auch die mobilen Spracherweiterungen, die im Verlauf dieser Arbeit entworfen werden, interpretieren können. Da im zeitlich begrenzten Rahmen dieser Masterarbeit keine komplette, den BPEL-Standard erfüllende Workflow-Engine realisiert werden kann, soll auf eine frei verfügbare Open Source-Engine zurückgegriffen werden. Diese soll so erweitert werden, dass sie die neuen Sprachelemente ebenfalls ausführen kann.

Neben proprietären BPEL-Engines, wie beispielsweise dem Oracle BPEL Process Manager, dem IBM WebSphere Process Server oder dem BizTalk Server von Microsoft, existieren auch einige Open-Source-Implementierungen. Davon sollen auszugsweise drei Engines kurz vorgestellt werden.

- *Apache ODE* (Orchestration Director Engine) [Apa-2008]
  - unterstützt BPEL4WS 1.1 und WS-BPEL 2.0
  - Java-Implementierung
  - Version 1.0 aus dem Jahr 2007
  - keine eigene Prozess-Entwicklungsumgebung verfügbar
- *Active Endpoints ActiveBPEL Engine* [Act-2007c]
  - unterstützt BPEL4WS 1.1 und WS-BPEL 2.0
  - Java-Implementierung
  - Version 4.0 aus Juli 2007
  - zusätzlich *ActiveBPEL Designer* als graphische Entwicklungsumgebung verfügbar
- *JBoss jBPM* [JBo-2008]
  - unterstützt BPEL4WS 1.1 und WS-BPEL 2.0
  - Java-Implementierung
  - aktiv seit Januar 2003
  - unterstützt proprietäre XML-Prozess-Beschreibungssprache jPDL

Um für die Verwendung in Frage zu kommen, mussten die Engines einige Grundvoraussetzungen erfüllen. Dazu gehört die Unterstützung der ausgewählten Spezifikation WS-BPEL 2.0 sowie verfügbare Dokumentation. Da die Dokumentation für die Engine von JBoss relativ undurchsichtig war und es sich bei Apache ODE noch um ein recht junges Projekt handelt, fiel die Wahl auf die ActiveBPEL Engine von Active Endpoints. Neben der meisten Dokumentation waren einige Erfahrungsberichte

anderer Forschungsarbeiten zu finden, die dieser BPEL-Engine gute Wiederverwendbarkeit bescheinigen. Ein erster Durchstich im Rahmen dieser Masterarbeit verlief ebenfalls erfolgreich.

3.2 Active Endpoints ActiveBPEL Engine

Die *ActiveBPEL Engine* ist ein Open Source-Projekt von Active Endpoints, bei dem es sich um eine Ausführungsumgebung für BPEL-Prozesse handelt [Act-2007c]. Sowohl die Spezifikation BPEL4WS 1.1 wie auch den Standard WS-BPEL 2.0 unterstützt die Engine in der Version 4.0 von Juli 2007. Diese wurde im weiteren Verlauf dieser Arbeit eingesetzt. Neben der ActiveBPEL Engine vertreibt Active Endpoints eine kommerzielle Version unter dem Namen *ActiveBPEL Enterprise*. Im Folgenden sollen die grundlegenden Eigenschaften der quelloffenen ActiveBPEL Engine erläutert sowie die für das weitere Verständnis dieser Arbeit relevanten Aspekte ihrer Architektur aufgezeigt werden.

3.2.1 Architektur der ActiveBPEL Engine

Die ActiveBPEL Engine ist in Java geschrieben und setzt einen installierten Servlet Container wie beispielsweise Apache Tomcat [Apa-2007b] voraus. Während der Installation wird die Engine in den Web Server deployed und kann anschließend zusammen mit diesem automatisch gestartet werden. Abbildung 3.1 zeigt die Engine-Architektur und ihre wesentlichen Komponenten.

Für die Kommunikation via SOAP-basierten Web Services wird Apache Axis [Apa-2007a] verwendet. Die Anbindung der BPEL-Engine an den Web Service Container realisieren sogenannte WS-Handler. Sie regeln die Weitergabe von ein- und ausgehenden Nachrichten zwischen Axis und der ActiveBPEL Engine.

Der Zustand der Engine, wozu u.a. sogenannte Deployment Plans und Prozess-Status gehören, kann über austauschbare Speicher-Implementierungen persistent gemacht werden. Somit kann die Engine ihre Arbeit nach einem Systemausfall ohne Datenverlust fortführen. In der Standard-Version werden jedoch In-Memory-Varianten für die verschiedenen Manager ausgeliefert. Diese sind für die Steuerung der Engine sowie für das Deployment und die Ausführung von Prozessen zuständig.

Den Kern der Engine stellt die Komponente BPEL Processor dar. Sie realisiert die BPEL-spezifische Logik. Dazu gehört die Verwaltung der deployten Prozessdefinitionen und der verschiedenen Manager. Die ActiveBPEL Engine unterstützt

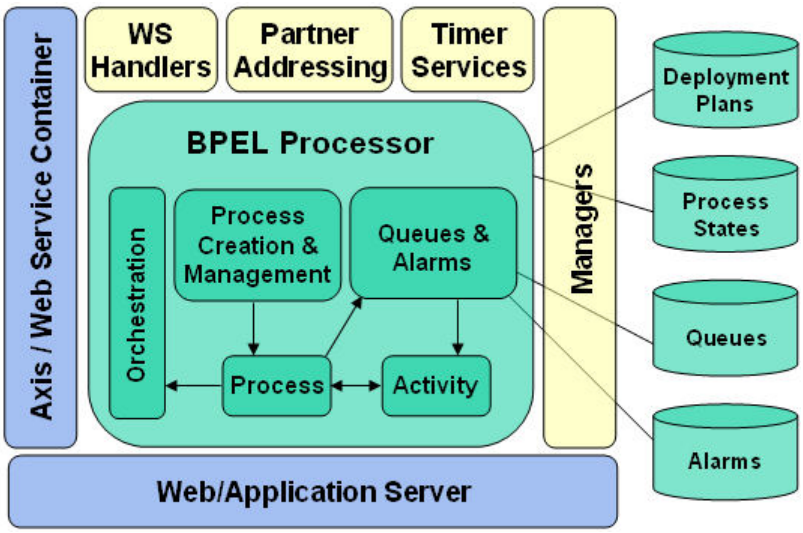

Abbildung 3.1: Architektur der ActiveBPEL Engine [Act-2007c]

ausführbare BPEL-Prozessdefinitionen. Wie deren Interpretation realisiert ist, zeigt der folgende Abschnitt auf.

3.2.2 Prozess-Deployment und -Ausführung

Neben einer standardisierten BPEL-Prozessdefinition und den zugehörigen WSDL-Dokumenten aller beteiligten Web Services benötigt die ActiveBPEL Engine weitere Informationen für das Deployment eines Prozesses. Diese werden in einem Java-Archiv mit der Endung .bpr zusammengestellt. Abbildung 3.2 gibt die Verzeichnisstruktur exemplarisch für ein Deployment-Archiv wieder.

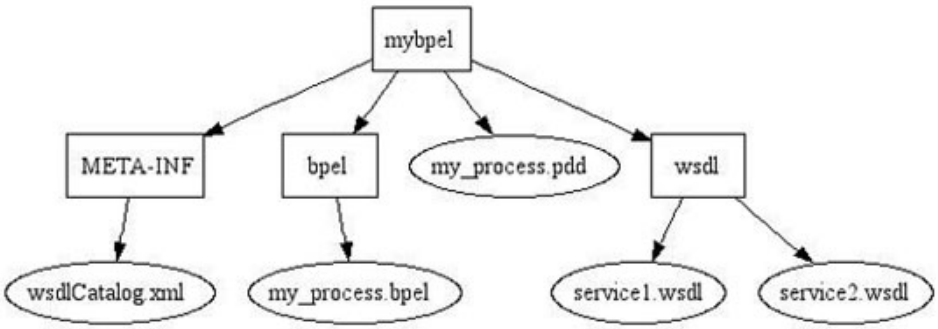

Abbildung 3.2: Verzeichnisstruktur für bpr-Archiv [Act-2007c]

Die Datei wsdlCatalog.xml muss die Pfade zu allen benötigten WSDL- und XSD-Dokumenten innerhalb des Archivs enthalten, sofern diese zur Laufzeit nicht über eine absolute URL zugänglich sind. Weiterhin wird ein engine-spezifischer Process Deployment Descriptor benötigt. Dieser wird in einer Datei mit Endung .pdd hinterlegt und enthält wichtige Deployment-Informationen, beispielsweise Details zu Partner Links und Binding-Strategien.

Um einen Prozess zu deployen, muss das bpr-Archiv in einem entsprechenden Verzeichnis der Engine-Installation abgelegt werden. Beim Starten der Engine wird dieses Verzeichnis eingelesen bzw. bei laufender Engine regelmäßig auf neue oder gelöschte Deployment-Archive untersucht. Neue Prozessbeschreibungen werden somit automatisch übernommen und in eine engine-interne Repräsentation überführt. Dabei wird anhand des Deployment Descriptors auch die Vorgehensweise zur Bestimmung der konkreten Partner Link-Implementierungen festgelegt. Für jede definierte Partner Role aller Partner Links muss der Process Deployment Descriptor entsprechende Angaben enthalten.

Im Rahmen des Deployments liest die ActiveBPEL Engine die XML-basierten BPEL-Dokumente ein und erzeugt dabei sogenannte *Definitions-Objekte*. Für jedes Sprachelement, das im BPEL-Standard definiert ist, existiert demnach in der Engine-Implementierung eine zugehörige Definitions-Klasse, die durch 1:1-Abbildung eine interne Repräsentation des BPEL-Elements generiert und alle notwendigen Informationen für die Prozessausführung enthält. Eine schematische Darstellung der Prozessausführung kann Abbildung 3.3 entnommen werden.

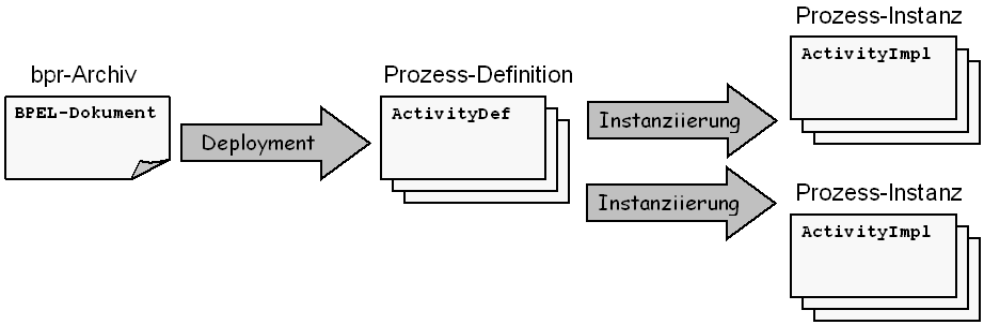

Abbildung 3.3: Vom Prozess-Deployment zur instanz-basierten Ausführung

Laut BPEL-Standard wird eine neue Prozessinstanz durch Auslösen einer definierten Startaktivität erzeugt, beispielsweise durch Eingang einer Nachricht. Die Engine setzt diese Vorgabe um, indem entsprechende Implementierungs-Klassen das Verhalten der BPEL-Definitions-Objekte zur Laufzeit modellieren. Eine neue Prozessinstanz wird also durch Erzeugung der *Implementierungs-Objekte* erstellt, die die Ausführung des Prozesses steuern.

Im Gegensatz zu den Definitions-Klassen muss nicht für jedes BPEL-Sprachelement eine entsprechende Implementierungs-Klasse existieren. Mindestens aber für jede definierte BPEL-Aktivität gibt es eine entsprechende Implementierungs-Klasse, die das jeweilige Verhalten intern umsetzt. Diese Aktivitäts-Implementierungs-Objekte werden durch Einsatz des Visitor-Pattern erzeugt (vgl. [Wik-2007]) und anschließend entsprechend der Prozess-Logik nacheinander abgearbeitet. Dazu muss jede Implementierung eine `execute()`-Methode anbieten, die zum Starten der jeweiligen Aktivität von der Engine aufgerufen wird. Um einen ersten Eindruck von Aufbau der Implementierungs-Objekte zu bekommen, zeigt Abbildung 3.4 die Klassenhierarchie einiger Activity-Implementierungen. Im Rahmen der Engine-Konzeption in Abschnitt 4.4 werden weitere Architektur-Aspekte der ActiveBPEL Engine vorgestellt.

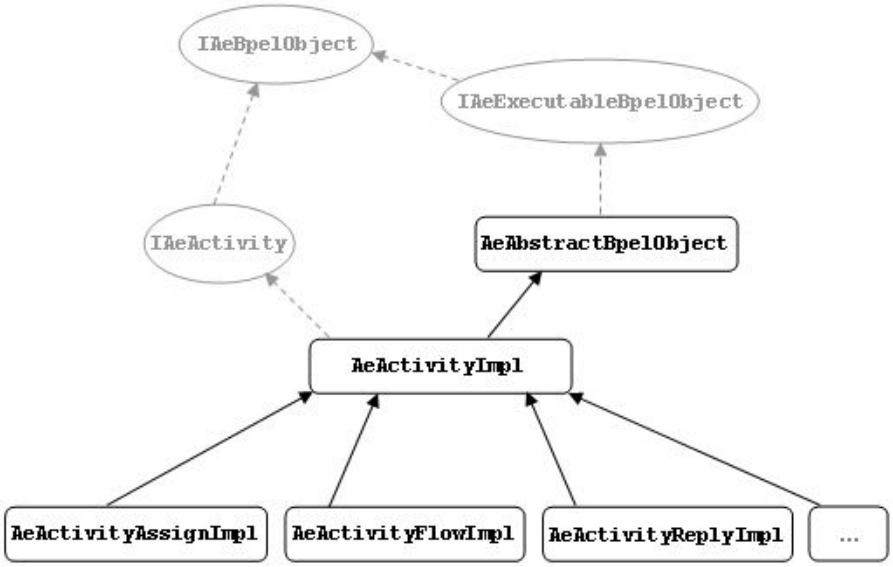

Abbildung 3.4: Klassenhierarchie der Activity-Implementierungen [Act-2007c]

3.2.3 Einsatz der ActiveBPEL Engine

Die ActiveBPEL Engine wird mit einer zugehörigen Web-Applikation ausgeliefert, die sich während der Engine-Installation ebenfalls im zugrundeliegenden Web Server deployed. Hierüber ist die Administration der Engine möglich. Es lassen sich Konfigurationseinstellungen vornehmen und die deployten BPEL-Prozesse einsehen. Beispielsweise kann man den Ausführungszustand einzelner Prozessinstanzen oder den Inhalt empfangener und gesendeter Nachrichten über die Administrationsoberfläche auslesen.

Neben den beiden zuvor erwähnten Engine-Varianten bietet Active Endpoints auch den *ActiveBPEL Designer* an [Act-2007a]. Der ActiveBPEL Designer ist nicht Open Source, jedoch nach Registrierung frei verfügbar. Es handelt sich dabei um eine graphische Entwicklungsumgebung für BPEL-Prozesse auf Eclipse-Basis, die darüber hinaus Hilfsmittel zur Verfügung stellt, mit der sich bpr-Archive inklusive der zum Deployment notwendigen Daten erstellen lassen.

3.3 Szenario "Auto-Pannendienst"

Die Einsatzmöglichkeiten mobiler Workflows sind zahlreich. Allen gemeinsam ist, dass eine prozessorientierte Zusammenarbeit einzelner Teilnehmer in Verbindung mit dem Einsatz mobiler Geräte die Grundlage bildet. Zusätzliche, stationäre Einheiten sind nicht ausgeschlossen. Zu den möglichen Einsatzbereichen zählen u.a. geschäftliche Zusammenarbeit, Unfallszenarien oder Spiele.

Anhand eines Beispiel-Szenarios soll die Anwendung mobiler Workflows exemplarisch aufgezeigt und die daraus resultierenden Anforderungen ermittelt werden. Auf Basis dieser Anforderungen wird der konzeptionelle Entwurf des mobilen Geschäftsprozesses für das Szenario und die notwendigen Erweiterungen der Sprache und Engine sowie die anschließende Realisierung erarbeitet. Da Geschäftsprozesse in der Praxis besonders bei inner- wie zwischenbetrieblichen Abläufen eine Rolle spielen, wurde zur Veranschaulichung die Auftragsabwicklung im B2B-Umfeld gewählt. Das Szenario *"Auto-Pannendienst"* wird in diesem Abschnitt vorgestellt und soll im weiteren Verlauf Berücksichtigung finden.

Der "Auto-Pannendienst" hilft Autofahrern, wenn sie durch einen Defekt an ihrem Fahrzeug liegenbleiben. Im Falle einer Panne wendet sich der betroffene Autofahrer an eine bundesweit zu erreichende Pannendienst-Zentrale. Die Pannenhilfe vor Ort leistet anschließend ein selbstständiger KFZ-Mechaniker, der im Auftrag für die Pannendienst-Zentrale agiert. Zu diesem Zweck arbeiten die Pannendienst-Zentrale und zahlreiche, regional zuständige KFZ-Mechaniker im Rahmen von B2B-Beziehungen zusammen.

Abbildung 3.5 zeigt den Prozessablauf, der jeder Pannenhilfe zugrundeliegt. Gestartet wird der Prozess, wenn bei der Pannendienst-Zentrale eine Pannenmeldung eingeht. Ob dies via Telefon, eine angebundene Software-Anwendung oder über eine beliebige andere Art der Kommunikation geschieht, ist für den weiteren Verlauf des Szenarios unerheblich. Entscheidend ist nur, dass mit der Startaktivität "Pannenmeldung erfassen" alle notwendigen Daten vom Kunden, wie Name, Anschrift, derzeitiger Standort, Fahrzeugdaten und Pannenbeschreibung, aufgenommen werden und mit diesen Informationen der Prozess "Auto-Pannendienst" initiiert wird.

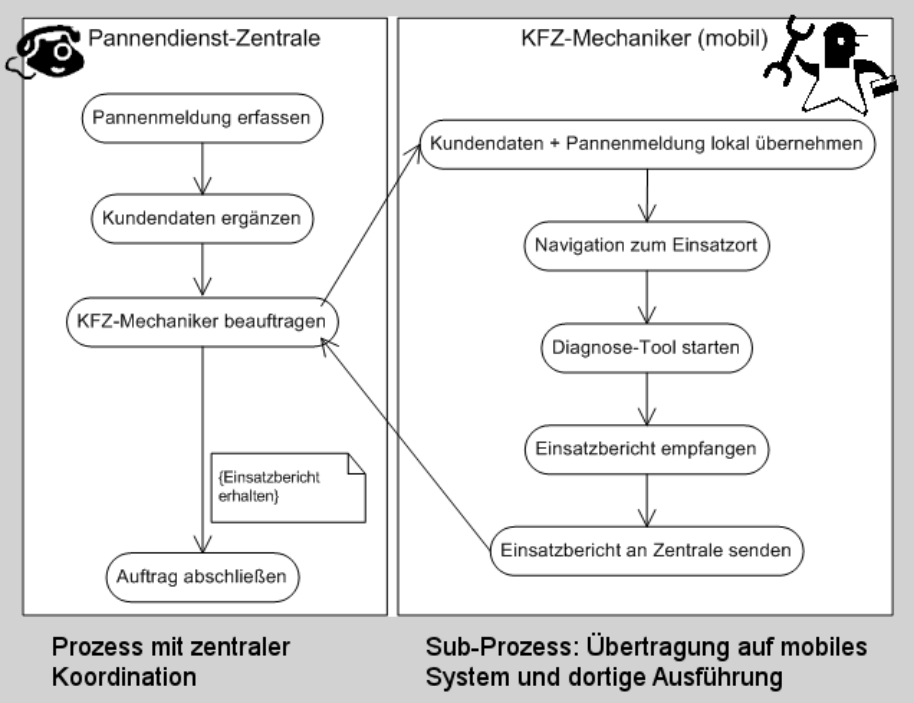

Abbildung 3.5: Prozessablauf für Szenario "Auto-Pannendienst"

Im nächsten Schritt "Kundendaten ergänzen" werden in der Zentrale ggf. schon vorhandene Vertragsdaten des jeweiligen Kunden abgerufen und automatisch ergänzt. Anschließend wird ein KFZ-Mechaniker mit der Pannenhilfe beauftragt; dies geschieht durch die Aktivität "KFZ-Mechaniker beauftragen".

Hierbei werden im Schritt "Kundendaten + Pannenmeldung lokal übernehmen" zunächst die von der Zentrale übermittelten Daten des Liegengebliebenen und die Meldung in eine SW-Anwendung des Mechanikers übertragen, damit dieser den Einsatz auf seinem Laptop festhalten kann. Anschließend findet mittels einem über das System des Mechanikers ansprechbaren Routenplaners die "Navigation zum Einsatzort" statt. Am Zielort angekommen verbindet der Mechaniker sein Laptop mit der entsprechenden Wartungs-Schnittstelle des Fahrzeugs. Der Prozess "Auto-Pannendienst" fährt mit dem Aufruf eines Fahrzeug-Diagnose-Tools fort. Hierfür ist die Aktivität "Diagnose-Tool starten" zuständig. Nun nimmt der Mechaniker die notwendige Reparatur am Fahrzeug vor. Die durchgeführten Maßnahmen hält er in seiner SW-Anwendung fest und beendet seinen Auftrag durch Übertragung eines Einsatzberichts an den laufenden Prozess. Dieser setzt dadurch seinen Ablauf mit dem Schritt "Einsatzbericht empfangen" fort und übermittelt automatisch den Einsatzbericht an die Pannendienst-Zentrale; siehe "Einsatzbericht an Zentrale senden".

Die Zentrale empfängt den Einsatzbericht als Rückmeldung für die Aktivität "KFZ-Mechaniker beauftragen" und übergibt die für den Abschluss der Pannenhilfe notwendigen Daten, z.B. für die Abrechnung, zur weiteren Verarbeitung an einen internen Service "Auftrag abschließen". Mit dieser Aktion ist der komplette Geschäftsprozess abgeschlossen.

3.4 Szenario-spezifische Anforderungen

Der "Auto-Pannendienst" beschreibt eine typische Ausprägung eines Geschäftsprozesses. Für die alltägliche Zusammenarbeit mit den KFZ-Mechanikern, die im Rahmen ihrer Tätigkeit mit dem Auto unterwegs sind, ist die klassische Umsetzung eines Geschäftsprozesses jedoch unzureichend. Bei der Realisierung wird typischerweise eine feststehende Infrastruktur mit einem zentralen Server vorausgesetzt, der jederzeit eine Verbindung zu allen benötigten Services aufbauen kann.

Aus dem in Abschnitt 3.3 beschriebenen Szenario ergibt sich die Forderung nach mehr Flexibilität und Dynamik, um die Mobilität des KFZ-Mechanikers in geeigneter Weise zu unterstützen. Im Folgenden soll geklärt werden, warum es sinnvoller ist, das Szenario in Form eines mobilen Geschäftsprozesses umzusetzen und welche Anforderungen an diesen hinsichtlich der Mobilität zu erfüllen sind.

3.4.1 Übertragbare Sub-Prozesse

Eine wesentliche Anforderung, die mobile Geschäftsprozesse erfüllen sollten, besteht in der Möglichkeit, *Sub-Prozesse* innerhalb des Gesamtprozesses zu definieren und diese *zur Laufzeit auf ein anderes System übertragen* zu können, wo sie lokal zur Ausführung gebracht werden. Hierdurch wird es möglich, innerhalb eines Prozesses logisch zusammengehörige Aktivitäten als Einheit zusammenzufassen und auf Ebene der Prozessbeschreibung eine Abgrenzung gegenüber den umgebenden Aktivitäten zu erreichen.

Im Rahmen des Pannendienst-Szenarios beziehen sich einige Aktivitäten auf die Ausführung durch den Mechaniker. Um diese Tatsache im Rahmen des Geschäftsprozesses zu verdeutlichen, lassen sich diese Aktivitäten als ein Sub-Prozess zusammenfassen, der innerhalb des Gesamtprozesses angesiedelt ist. Die rechte Seite der Abbildung 3.5 stellt somit den Sub-Prozess mit seinen zugehörigen Aktivitäten dar.

Da aus technischen Gründen nicht sichergestellt werden kann, dass zu jeder Zeit zwischen der Pannendienst-Zentrale und dem Mechaniker eine Kommunikations-Verbindung besteht, über die die einzelnen Aktivitäten ausgeführt werden können,

geht die Anforderung über die reine Definition von Sub-Prozessen hinaus. Vielmehr sollen Sub-Prozesse auf ein anderes System, hier das Laptop des Mechanikers, zur Laufzeit übertragbar sein. Für das Szenario bedeutet dies, dass sobald ein Mechaniker den Auftrag erhalten hat, dieser die Pannenhilfe autonom und notfalls ohne Verbindung zur Zentrale durchführen können soll. Der definierte Sub-Prozess wird im Kontext der übergeordneten Aktivität "KFZ-Mechaniker beauftragen" von der Pannendienst-Zentrale auf das System des KFZ-Mechanikers übertragen und dort beginnend mit der Aktivität "Kundendaten + Pannenmeldung lokal übernehmen" zur Ausführung gebracht. Mit Vollendung der "Einsatzbericht an Zentrale senden"-Aktivität wird der Sub-Prozess auf dem mobilen System des Mechanikers beendet, und der Haupt-Prozess in der Zentrale fährt mit seiner Verarbeitung fort. Dabei ist es für die zum Sub-Prozess gehörenden Aktivitäten unerheblich, ob diese als lokale, auf dem mobilen System verfügbaren Services realisiert werden oder externe Services eingebunden werden. Technisch besteht bei den Eigenschaften der in den Sub-Prozess involvierten Services kein Unterschied zu den übrigen Services, z.B. im umgebenden Gesamtprozess, der in der Pannendienst-Zentrale ausgeführt wird.

Eine Alternative gegenüber zur Laufzeit übertragbaren Sub-Prozessen wäre die Definition eines inhaltlich dem Sub-Prozess entsprechenden, eigenständigen Geschäftsprozesses, der seinerseits auf dem mobilen System ausgeführt wird, und dieser als Aktivität in den übergeordneten Prozess der Zentrale eingebunden wird. In diesem Fall wäre es möglich, dass jeder KFZ-Mechaniker einen eigenen Ablauf definiert. Da das Szenario jedoch die Definition der Prozessbeschreibung durch die Pannendienst-Zentrale vorsieht, damit diese die Kontrolle über den gesamten Geschäftsprozess behält und den Ablauf für alle Geschäftspartner vorgeben kann, resultiert hieraus die Forderung nach übertragbaren Prozessen. Weiterhin ist die Zentrale somit in der Lage, flexibler auf Änderungen zu reagieren, die zu einer Anpassung des Geschäftsprozesses führen. Für die Systeme der Mechaniker bedeutet dies, dass sie den empfangenen Sub-Prozess als eigenständige Prozessdefinition verstehen und ohne die Kenntnis des übergeordneten Gesamtprozesses ausführen. Die auf diesem Weg dynamisch in den Prozess integrierten Systeme stellen quasi eine Ausführungsumgebung für die Sub-Prozesse zur Verfügung. Die auszuführenden Prozessdefinitionen sind der Umgebung a priori unbekannt und können als mobiler Code verstanden werden.

Durch die Einführung übertragbarer Sub-Prozesse erreicht man eine stärkere Dynamik, dennoch bleibt die Kontrolle durch die oberste Instanz, hier der Pannendienst-Zentrale, erhalten. Die Ausführung im Vorfeld festgelegter Teilabläufe geschieht per Delegation an ausgewählte Prozess-Teilnehmer. Es besteht die Möglichkeit, mehrere Sub-Prozesse innerhalb eines Prozesses zu definieren. Prinzipiell ist auch die Schachtelung mehrerer Sub-Prozesse denkbar, d.h. dass innerhalb eines Sub-Prozesses die Definition weiterer Sub-Prozesse möglich ist, wodurch verschiedene Ausführungsumgebungen angesprochen werden können.

Das in Abschnitt 3.3 betrachtete Szenario sieht vor, dass der Hauptprozess in der Pannenhilfe-Zentrale solange mit der Ausführung wartet, bis der Sub-Prozess abgeschlossen und der Einsatzbericht vom Mechaniker eingegangen ist. In anderen Szenarien ist es vorstellbar, dass ein Sub-Prozess lediglich durch Übertragung an einen weiteren Teilnehmer initialisiert wird, aber das Ergebnis dessen bzw. eine explizite Rückmeldung nicht notwendig ist. Eine weitere Möglichkeit ist, dass der übergeordnete Prozess nicht solange ruht bis eine Rückmeldung erfolgt, sondern dass in der Zwischenzeit andere Aktivitäten ausgeführt werden und das Ergebnis erst im späteren Verlauf empfangen wird. Vor diesem Hintergrund und verstärkt durch die im mobilen Umfeld nicht dauerhaft verfügbare Netzverbindung ist es sinnvoll, für die Realisierung der übertragbaren Sub-Prozesse ein asynchrones Kommunikationsmodell vorzusehen.

Wenn man das Szenario aus Sicht des Mechanikers betrachtet, kann dieser theoretisch jederzeit beliebig viele Aufträge von der Zentrale erhalten. Bisher hat dieser keine Möglichkeit, einen Auftrag bzw. den Empfang eines Sub-Prozesses abzulehnen; außer er ist technisch nicht erreichbar. Um dieses zu ermöglichen, sollte eine Funktionalität unterstützt werden, mittels derer sich die Übernahme eines Sub-Prozesses zur Laufzeit ablehnen lässt.

3.4.2 Kontextsensitivität

Neben der Einführung übertragbarer Sub-Prozesse spielt die Unterstützung der *Kontextsensitivität* bei mobilen Geschäftsprozessen eine wesentliche Rolle (vgl. Abschnitt 2.4.2). Sie ermöglicht eine sinnvoll an die jeweilige Situation angepasste Reaktion der Anwendung zur Laufzeit. Die wichtigsten Anforderungen in diesem Zusammenhang bestehen in der Wahrnehmung des aktuellen Kontexts, dem Erkennen von Kontextänderungen und der kontextabhängigen Entscheidungsfähigkeit der Prozess-Ausführungsumgebung. Abbildung 3.6 zeigt eine erweiterte Version vom Szenario "Auto-Pannendienst", das die Einbeziehung der Kontextsensitivität veranschaulichen soll.

Im erweiterten Szenario wird der jeweilige Kontext exemplarisch in die Ausführung von insgesamt vier Aktivitäten einbezogen. Drei dieser Aktivitäten werden im Rahmen des Sub-Prozesses auf dem mobilen System des Mechanikers ausgeführt (vgl. Abschnitt 3.4.1); hier kann die Kontextsensitivität naturgemäß in verschiedenster Ausprägung genutzt werden.

Aber auch die Pannendienst-Zentrale kann das Wissen um den aktuellen Kontext sinnvoll einsetzen: Mit der Aktivität "KFZ-Mechaniker beauftragen" wurde bisher ein nicht näher bestimmter Mechaniker zum Panneneinsatz beordert. Es macht Sinn, bei jedem Prozessdurchlauf einen für den aktuellen Fall geeigneten Mechaniker aus-

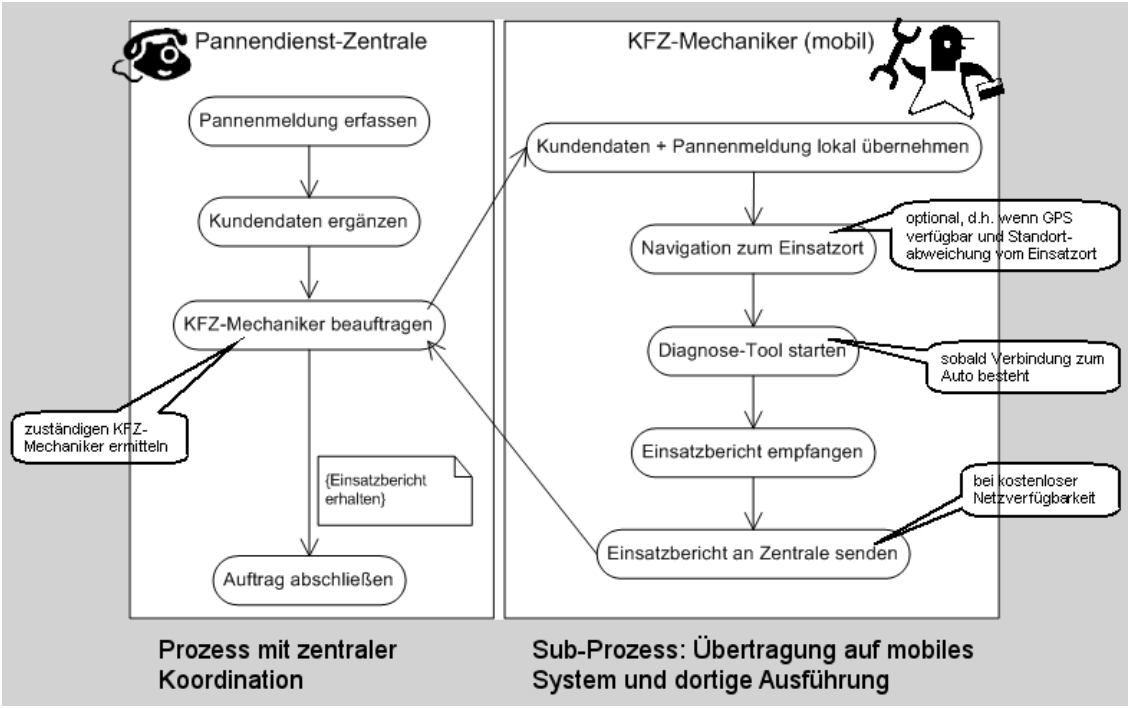

Abbildung 3.6: Kontextsensitiver Prozessablauf für Szenario "Auto-Pannendienst"

zuwählen und diesen mit der Pannenhilfe zu beauftragen. Basierend auf den aus vorhergehenden Aktivitäten vorliegenden Kundendaten und dem aktuellen Standort des Liegengebliebenen kann ein KFZ-Mechaniker ermittelt werden, der sich z.B. in der Nähe des Betroffenen befindet, genug Zeit und die notwendigen Hilfsmittel an Bord hat. Hierfür benötigt die Ausführungsumgebung unter anderem aktuelle Informationen über die zeitliche Auslastung der einzelnen Mechaniker und ihren Standort. Diese Informationen bilden den für diese Aktivität zu berücksichtigenden Kontext. Aus einer Menge an verfügbaren KFZ-Mechanikern muss der am besten geeignete ausgewählt werden und als Service-Partner dynamisch in den Prozess eingebunden werden. Hierfür wird eine Funktionalität benötigt, mittels der zur Laufzeit ausgewertet werden kann, welche Services mit welchem Kontext verfügbar sind.

Ähnlich verhält es sich mit den kontextsensitiven Aktivitäten des Sub-Prozesses. Die "Navigation zum Einsatzort" soll optional ausgeführt werden. Ein lokal oder anderweitig vom mobilen Mechaniker-Laptop nutzbarer Service kann nur sinnvoll in den Ablauf integriert werden, wenn GPS aktuell verfügbar ist. Zudem sollte sich der derzeitige Standort vom Einsatzort unterscheiden. Diese Entscheidungen müssen für jeden Mechaniker individuell und abhängig vom aktuellen Kontext getroffen werden.

Erreicht der Mechaniker den liegengebliebenen Kunden, so verbindet er sein Laptop mit der entsprechenden Wartungsschnittstelle des Kundenfahrzeugs. Die Engine erkennt dieses und führt automatisch die Aktivität "Diagnose-Tool starten" aus.

Wenn der Mechaniker seinen Pannenhilfe-Einsatz beendet, schließt der Sub-Prozess mit der Aktivität "Einsatzbericht an Zentrale senden" ab. Da diese Aktivität eine Verbindung zur Pannendienst-Zentrale voraussetzt, soll die Übertragung erst ausgeführt werden, wenn ein kostenloser Netzzugang gewährleistet ist. Im konkreten Fall kann der Sub-Prozess beispielsweise direkt vor Ort oder zurück im Büro des Mechanikers fortgesetzt werden.

3.5 Anforderungen zur Unterstützung mobiler Workflows

Mobile Workflows zeichnen sich im Allgemeinen durch für sie typische Eigenschaften aus. Dazu gehören:

- inner- sowie zwischenbetriebliche Abläufe
- Einbindung mobiler Mitarbeiter in die Geschäftsprozesse
- mobile (und stationäre) Geräte
- Mobilität
- Context Awareness

Das Szenario "Auto-Pannendienst", das die B2B-Beziehung zwischen der Pannendienst-Zentrale und dem KFZ-Mechaniker abbildet, erfüllt diese Eigenschaften in einem überschaubaren Umfang. Eine Übertragung dieser Eigenschaften auf Szenarien mit ähnlicher Struktur, z.B. im Vertrieb oder in der Logistik, ist leicht möglich. Daher steht der "Auto-Pannendienst" repräsentativ für eine Vielzahl anderer Einsatzmöglichkeiten und dient der vorliegenden Arbeit zur exemplarischen Veranschaulichung.

Die in Abschnitt 3.4 aufgestellten szenario-spezifischen Anforderungen zeigen das Spektrum an Charakteristika auf, für die das zu entwickelnde System geeignet sein soll. Dazu soll dieser Abschnitt die Anforderungen an die Prozess-Beschreibungssprache und an die Engine zusammenfassend herausarbeiten, auf Basis derer die anschließende Konzeption vorgenommen werden kann.

3.5.1 Anforderungen an die Prozess-Beschreibungssprache

Mobile Geschäftsprozesse bauen auf den Grundlagen klassischer Geschäftsprozesse auf. Neben deren hier gewählten Umsetzung durch die Komposition von Web Services gibt es weitere Möglichkeiten, bei denen ebenfalls Prozess-Beschreibungssprachen verwendet werden. Der Vollständigkeit halber soll an dieser Stelle nicht nur auf die Anforderungen zur Unterstützung mobiler Prozesse, sondern auch auf die grundlegenden, sprachlichen Anforderungen eingegangen werden, die zur Beschreibung von Geschäftsprozessen im Allgemeinen gelten.

Allgemeine Anforderungen zur Beschreibung klassischer Geschäftsprozesse

Der Begriffsdefinition entsprechend setzt sich ein Geschäftsprozess aus Aktivitäten und Bedingungen zusammen und legt die Reihenfolge ihrer Ausführung fest (vgl. Abschnitt 2.1). Dementsprechend muss eine Beschreibungssprache eine formale Abbildung dieser Konstrukte ermöglichen und die Eigenschaften des Prozesses unterstützen. In der Literatur lassen sich zahlreiche Arbeiten dazu finden, die einen entsprechenden Anforderungskatalog erarbeitet haben. Hier sollen kurz die wichtigsten, allgemeingültigen Anforderungen an Beschreibungssprachen aufgelistet werden, die in jedem Fall zu berücksichtigen sind (nachzulesen u.a. in [AH-2002], [FL-2003] und [Zap-2005]). Dazu zählen:

- *Kontrollflusskonstrukte*: graphen- oder blockbasierte Beschreibung der Reihenfolge und Bedingungen, denen die Aktivitäten unterliegen
- *Datenflusskonstrukte*: Regelung des Datenflusses, z.B. zwischen Input- und Output-Daten einer Aktivität
- *Fehlerbehandlungsmaßnahmen*: Beschreibung von Fehlersituationen und Reaktionen darauf
- *Beschreibung von Prozessteilnehmern*: konkrete oder abstrakte Definition der am Prozess beteiligten Teilnehmer
- *Plattformunabhängigkeit*: hohe Kompatibilität der verschiedenen Systeme u.a. durch Betriebssystem- und Transportprotokollunabhängigkeit
- *Erweiterbarkeit*: Ergänzung neuer Funktionalität, ohne bestehende Sprachkonstrukte ändern zu müssen

Die obige Auflistung ist sicherlich nicht erschöpfend, dennoch ermöglicht sie einen ersten Überblick. Neben diesen allgemeinen Anforderungen werden im Rahmen dieser Arbeit weitere Bedingungen an die Sprache gestellt, die mindestens erfüllt werden sollten. Dazu gehören – ohne Berücksichtigung der mobilen Erweiterungen – folgende Punkte:

- *Berücksichtigung der SOA-Konzepte*: Beschreibung komponierter Services
- *Kompatibilität* zu bestehenden Ansätzen

Durch die Tatsache, dass für die weitere Konzeption auf die existierende Prozess-Beschreibungssprache BPEL zurückgegriffen wird, müssen die bisher genannten Anforderungen nicht weiter berücksichtigt werden. BPEL erfüllt diese bereits als Sprache zur Modellierung klassischer Geschäftsprozesse durch Komposition von Web Services.

Erweiterte Anforderungen zur Beschreibung mobiler Geschäftsprozesse

Die zu berücksichtigenden Anforderungen ergeben sich durch die geplante Spracherweiterung zur Unterstützung mobiler Workflows. Dazu wurden bereits Forderungen bezüglich übertragbarer Sub-Prozesse und der Integration von Kontextsensitivität aus dem Szenario "Auto-Pannendienst" abgeleitet (vgl. Abschnitte 3.4.1 und 3.4.2). Eine Analyse dieser Anforderungen ergibt, dass für folgende Eigenschaften keine geeigneten Standard-BPEL-Elemente verfügbar sind:

- Übertragung des Pannenhilfe-Auftrags in Form eines Sub-Prozesses
- Ermitteln eines geeigneten KFZ-Mechanikers für den aktuellen Pannenhilfe-Einsatz
- optionale Navigation zum Einsatzort, wenn GPS verfügbar ist und sich der aktuelle Standort vom Einsatzort unterscheidet
- Starten des Diagnose-Tools, sobald eine Verbindung zum Auto besteht
- Senden des Einsatzberichts, sobald ein kostenloser Netzzugang verfügbar ist

Betrachtet man die genannten Eigenschaften losgelöst vom konkreten Szenario, lassen sich die Anforderungen an die sprachliche Modellierbarkeit wie folgt verallgemeinern:

- Definition eines Sub-Prozesses und dessen Übertragung zur Laufzeit an andere Ausführungsumgebung
- Empfang eines Sub-Prozesses und dessen Ausführung
- Auswahl und dynamisches Binding eines Service-Providers abhängig vom aktuellen Kontext
- Ausführung einer (ggf. optionalen) Aktivität abhängig vom aktuellen Kontext

Die vier zuletzt genannten Punkte stellen die Anforderungen zur Unterstützung mobiler Workflows dar, die im Rahmen dieser Arbeit durch Erweiterung der Sprache BPEL erfüllt werden sollen. Weiterführend wird mit der Konzeption der Sprache in Abschnitt 4.3 darauf eingegangen.

3.5.2 Anforderungen an die Engine

Die Anforderungen, die an die ausführende Engine gestellt werden, orientieren sich stark an den Anforderungen der Sprache. Schließlich besteht die Aufgabe der Engine darin, die Prozessbeschreibungen interpretieren zu können. Dazu müssen die Konzepte der konkreten Sprache umgesetzt werden, z.B. indem die verschiedenen Kontrollflusskonstrukte von der Engine korrekt abgearbeitet werden. Darüber hinaus sollten bei der Prozessausführung Fehlersituationen oder Verklemmungen erkannt und geeignet darauf reagiert werden [Zap-2005]. Elementar ist ebenfalls die korrekte Abarbeitung des Lebenszyklus einer Prozessinstanz, der entsprechende Zustände und Zustandsübergänge unterstützen sollte.

In Abschnitt 3.2 wurde bereits die ActiveBPEL Engine vorgestellt. Als Ausführungsumgebung von BPEL-Prozessdefinitionen unterstützt diese die grundlegenden Anforderungen. Für eine Beschreibung der möglichen Zustände der Implementierungs-Objekte, die den zuvor genannten Prozesslebenszyklus umsetzen, sowie weiterführende Informationen wird auf die entsprechende Dokumentation von Active Endpoints verwiesen [Act-2007c]. Die Basisfunktionalitäten werden durch Verwendung der ActiveBPEL Engine daher für die weitere Arbeit als gegeben vorausgesetzt.

Die Konzeption der mobilen Engine-Erweiterung kann sich somit auf die vier zuvor definierten Anforderungen für die Spracherweiterung konzentrieren. Da sich eine konkretere Beschreibung der daraus resultierenden Anforderungen erst aus dem Ergebnis der Sprach-Konzeption ergibt, wird in Abschnitt 4.4 weiterführend darauf eingegangen.

Kapitel 4

Konzeption

Die in der Analyse gewonnenen Ergebnisse und die daraus resultierenden Anforderungen dienen als Basis für die Konzeption, die in diesem Kapitel behandelt wird. Neben der Modellierung der szenario-basierten Geschäftsprozesse liegt der Schwerpunkt auf dem konzeptionellen Entwurf der Sprache und der Engine, mittels derer mobile Workflows unterstützt werden sollen. Begonnen wird mit einer kurzen Vorstellung des gewählten Vorgehens sowie grundlegender Entwurfsaspekte.

4.1 Entwurfsgrundlagen und Vorgehen

Das Vorgehen bei der Konzeption orientiert sich an den Erkenntnissen aus der Analysephase, wobei im weiteren Verlauf auch erste Ergebnisse des Entwurfs wiederum Berücksichtigung finden. Die Konzeption wurde dazu in drei Phasen unterteilt.

Die erste Phase behandelt die Modellierung der Geschäftsprozesse. Dabei werden unter anderem die Abläufe für das in Abschnitt 3.3 vorgestellte Szenario "Auto-Pannendienst" spezifiziert. Dieser Teil der Konzeption bildet die Basis für die Prozessdefinitionen, die im Rahmen der Realisierung mittels BPEL und der mobilen Erweiterungen in Kapitel 5 vorgenommen werden. Weitere Details der Geschäftsprozess-Modellierung enthält Abschnitt 4.2.

Die zweite Phase beschäftigt sich mit der Konzeption der Sprache. Hierbei wird basierend auf den erweiterten Anforderungen aus Abschnitt 3.5.1 konkret auf die Entwicklung mobiler BPEL-Erweiterungen eingegangen. Dabei werden die Anforderungen mit den Erweiterungsmöglichkeiten vom BPEL-Standard abgeglichen und eigene Sprachelemente durch Syntaxbeschreibungen und Beispiele definiert. Abschnitt 4.3 behandelt diese Thematik weitergehend.

Die dritte Phase bildet den letzten Teil der Konzeption und beinhaltet die Architektur der Engine. Die Anforderungen hierfür ergeben sich größtenteils aus den zuvor entworfenen Spracherweiterungen. Der vorgestellte Entwurf umfasst die Integration der mobilen BPEL-Spracherweiterungen in die ActiveBPEL Engine sowie die Umsetzung der jeweiligen Logik. Mit der engine-spezifischen Konzeption beschäftigt sich abschließend Abschnitt 4.4.

Im Rahmen der Konzeption werden verschiedene Definitionen vorgenommen. Neben deutschen Bezeichnungen werden dabei auch Begriffsdefinitionen auf englisch erfolgen. Diese Vereinbarung wurde getroffen, da sowohl die BPEL-Spezifikation als auch die ActiveBPEL Engine auf englisch verfasst sind und die anschließende Realisierung ebenfalls auf englisch erfolgen wird. Von diesem Vorgehen betroffen sind speziell BPEL-Definitionen oder implementierungsnahe Diagramme, wie beispielsweise UML-Klassendiagramme. Um auch im laufenden Text erkennen zu können, wann eine Bezeichnung sich auf die Implementierung bezieht, werden diese Begriffe in einer `anderen Schriftart` wiedergegeben.

4.2 Modellierung der Geschäftsprozesse

In Abschnitt 3.3 wurde das Szenario "Auto-Pannendienst" vorgestellt. Der bisher lediglich grob vorgegebene Ablauf soll nun unter Berücksichtigung der szenario-spezifischen Anforderungen wie Kontextsensitivität und übertragbaren Sub-Prozessen (vgl. Abschnitt 3.4) genauer spezifiziert werden.

4.2.1 Prozess-Komponenten

Aus der Szenario-Beschreibung in Abschnitt 3.3 geht hervor, dass die Ablaufsteuerung auf zwei verschiedenen Systemen stattfindet: in der Pannendienst-Zentrale und beim KFZ-Mechaniker. Auf beiden Umgebungen werden unterschiedliche Aktivitäten ausgeführt. Die Aktivitäten selbst sind als Funktionen verschiedener Services realisiert, die im Verlauf des Prozesses aufgerufen werden. Dabei wird der gesamte Ablauf in den *Prozess "Pannenhilfe beauftragen"* und den *Sub-Prozess "Pannenhilfe ausführen"* unterteilt.

Abbildung 4.1 zeigt alle am Szenario beteiligten Komponenten und ihre Abhängigkeiten untereinander. Die weiß hinterlegten Komponenten stellen die Services dar, die in der IT-Landschaft der Pannendienst-Zentrale angesiedelt sind. Hierzu gehören der *Pannendienst-Service*, der *Stammdaten-Service* und der *Abrechnungs-Service*. Der Stammdaten-Service stellt die Funktionalität "Kundendaten ergänzen" zur Verfügung, während über den Abrechnungs-Service die Aktion "Auftrag abschließen"

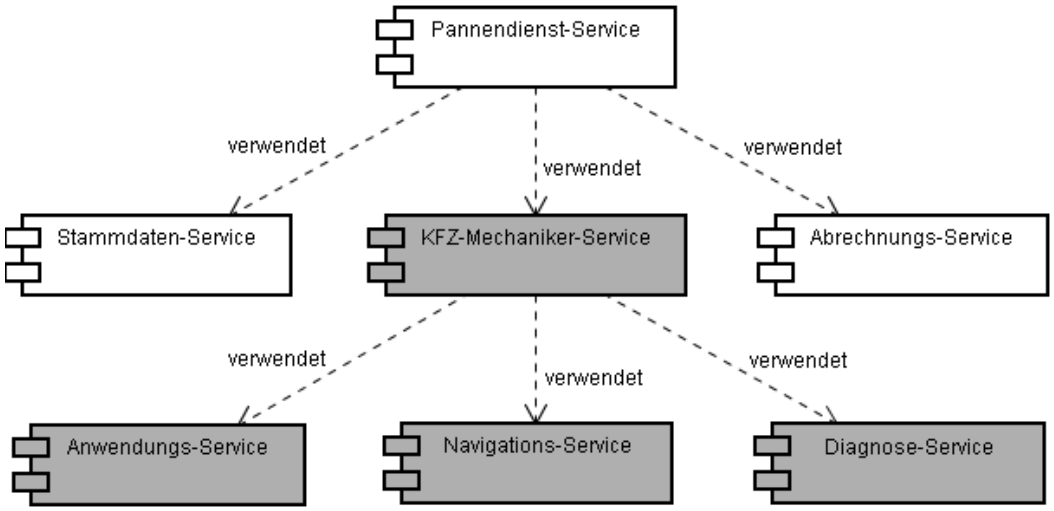

Abbildung 4.1: Komponentendiagramm "Auto-Pannendienst"

ausgeführt werden kann. Beide Services werden vom übergeordneten Pannendienst-Service verwendet, der den Prozess "Pannenhilfe beauftragen" realisiert.

Das Szenario sieht die Ausführung des Sub-Prozesses "Pannenhilfe ausführen" auf dem System des KFZ-Mechanikers vor. Alle hieran beteiligten Komponenten sind in der Abbildung 4.1 grau hinterlegt. Den Ablauf des Sub-Prozesses steuert die Komponente *KFZ-Mechaniker-Service*, an diese wird der Sub-Prozess vom übergeordneten Pannendienst-Service übertragen. In dessen Ausführung sind die Komponenten *Anwendungs-Service*, *Navigations-Service* und *Diagnose-Service* involviert. Die Aktionen "Kundendaten + Pannenmeldung lokal übernehmen" und "Einsatzbericht empfangen" werden vom Anwendungs-Service realisiert. Der Navigations-Service bietet die Funktionalität "Navigation zum Einsatzort" an, während "Diagnose-Tool starten" vom Diagnose-Service ausgeführt wird.

4.2.2 Interne Prozessabläufe

Aufbauend auf dem bisherigen statischen Entwurf beschäftigt sich dieser Abschnitt mit der Modellierung der dynamischen Abläufe innerhalb der einzelnen Komponenten. Für das Szenario ist hierbei insbesondere das Verhalten des Prozesses "Pannenhilfe beauftragen" und des Sub-Prozesses "Pannenhilfe ausführen" von Bedeutung.

Prozessablauf "Pannenhilfe beauftragen"

Die Komponente Pannendienst-Service stellt die zentrale Kontrollinstanz für das Szenario dar. Abbildung 4.2 zeigt das Aktivitätsdiagramm für den Prozess "Pannenhilfe beauftragen". Wenn im Text nicht anders angegeben, handelt es sich bei den Aktivitäten um Service-Aufrufe – entweder in Form einer eingehenden oder ausgehenden Nachricht. Hierauf wird im nachfolgenden Abschnitt 4.2.3 genauer eingegangen.

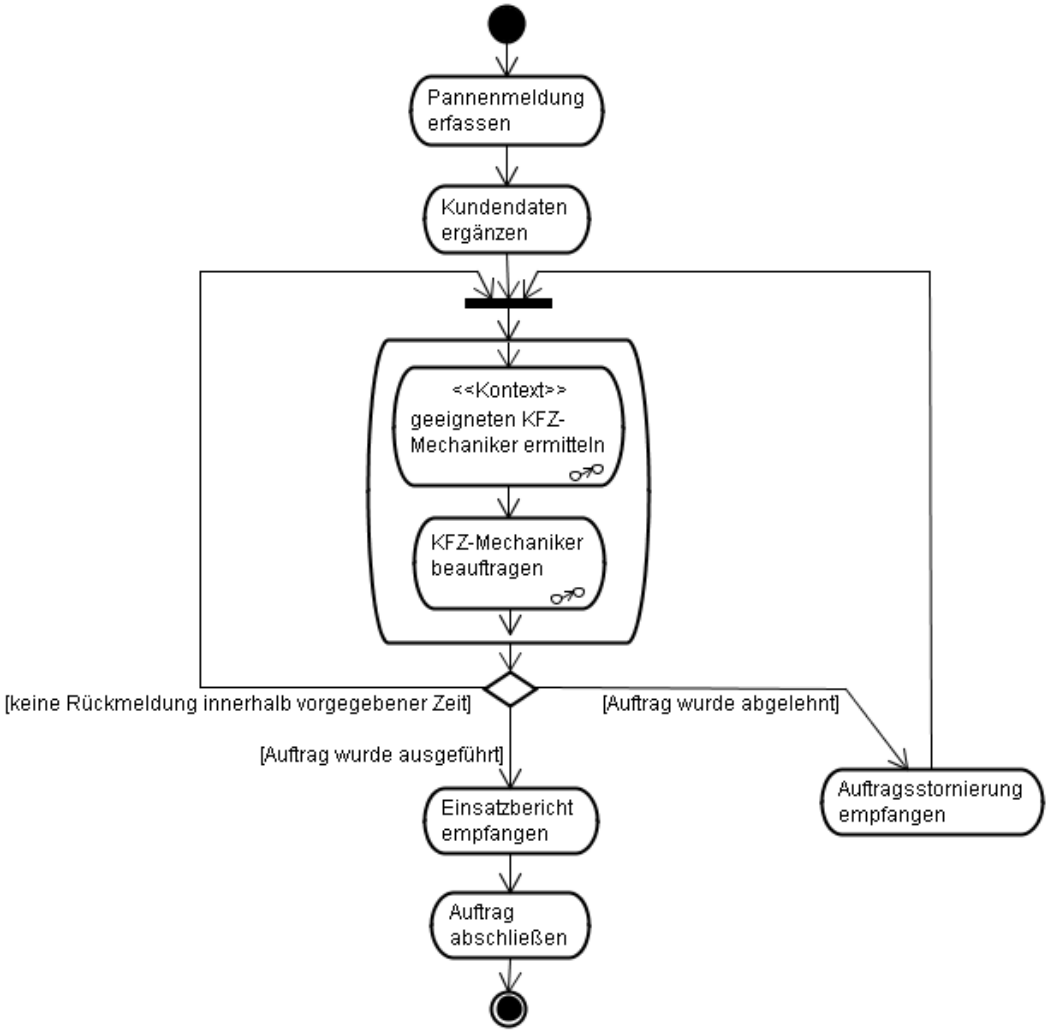

Abbildung 4.2: Aktivitätsdiagramm Prozess "Pannenhilfe beauftragen"

Gestartet wird der Prozess durch Eingang einer Pannenmeldung. In einem zweiten Schritt werden die Kundendaten um ggf. weitere vorhandene Daten ergänzt. Mittels dieser Informationen wird anschließend ein geeigneter KFZ-Mechaniker ermittelt

und dieser mit der Pannenhilfe beauftragt. Bei diesen beiden zuletzt genannten Aktivitäten handelt es sich um erweiterte Anforderungen gemäß Abschnitt 3.5.1. Sie können als Sub-Aktivitäten zu einer Aktivität zusammengefasst werden. Die Sub-Aktivität "geeigneten KFZ-Mechaniker ermitteln" entscheidet kontextsensitiv, welcher Mechaniker eingesetzt wird. Dieses Ergebnis wird in der darauf folgenden Sub-Aktivität "KFZ-Mechaniker beauftragen" berücksichtigt, um den Pannenhilfe-Auftrag in Form einer Sub-Prozessdefinition zu übertragen.

Als nächste Aktion sieht das Szenario den Eingang eines Einsatzberichts vom KFZ-Mechaniker vor, die den erfolgreichen Abschluss der Pannenhilfe symbolisiert. An dieser Stelle wird der Prozessablauf erweitert: zum einen, um der Anforderung, einen erhaltenen Auftrag ablehnen zu können, gemäß Abschnitt 3.4.1 nachzukommen und zum anderen, um Unsicherheiten in Verbindung mit dem Einsatz mobiler Systeme zu berücksichtigen. Da der Mechaniker den Auftrag möglicherweise nicht erhalten hat oder dieser zwar Pannenhilfe geleistet hat, aber sein Einsatzbericht noch nicht eingegangen ist, muss die Zentrale in diesen Fällen davon ausgehen, dass dem betroffenen Autofahrer noch nicht geholfen wurde. Nach einer vordefinierten Zeitspanne, innerhalb der weder eine Auftragsstornierung noch der Einsatzbericht eingeht, wird wieder an die Stelle gesprungen, wo ein KFZ-Mechaniker ermittelt und beauftragt wird. Dasselbe geschieht im Falle einer Auftragsstornierung.

Bei Eingang des Einsatzberichts folgt als letzte Aktivität das Abschließen des Auftrags, und der gesamte Prozess "Pannenhilfe beauftragen" ist beendet. Der beschriebene Ablauf geht immer von einem positiven Ausgang der Pannenhilfe aus. Für die vorliegende Arbeit soll dieses Verhalten genügen. Für den realen Einsatz müsste beispielsweise nach fünfmaligem Fehlversuch eine entsprechende Problemmeldung erfolgen.

Sub-Prozessablauf "Pannenhilfe ausführen"

Die im Rahmen der Pannenhilfe abzuleistenden Aktivitäten übergibt die Zentrale in Form des Sub-Prozesses "Pannenhilfe ausführen" an den KFZ-Mechaniker-Service. Das Aktivitätsdiagramm, das den internen Ablauf des Sub-Prozesses zeigt, ist in Abbildung 4.3 dargestellt. Analog zum Prozessablauf "Pannenhilfe beauftragen", der das Senden der Sub-Prozessdefinition realisiert, enthält der nun betrachtete Ablauf "Pannenhilfe ausführen" eine Aktivität zum Empfangen des Sub-Prozesses. Diese ist die erste Aktion seitens des Mechaniker-Systems.

Im Anschluss an den Erhalt des Pannendienst-Auftrags sind zwei Ausführungspfade möglich – abhängig davon, ob der Auftrag abgelehnt oder angenommen wird. Verschiedene Möglichkeiten, wie diese Entscheidung im Prozessverlauf getroffen werden kann, behandelt Abschnitt 4.3.4 im Kontext der Spracherweiterung für übertragbare

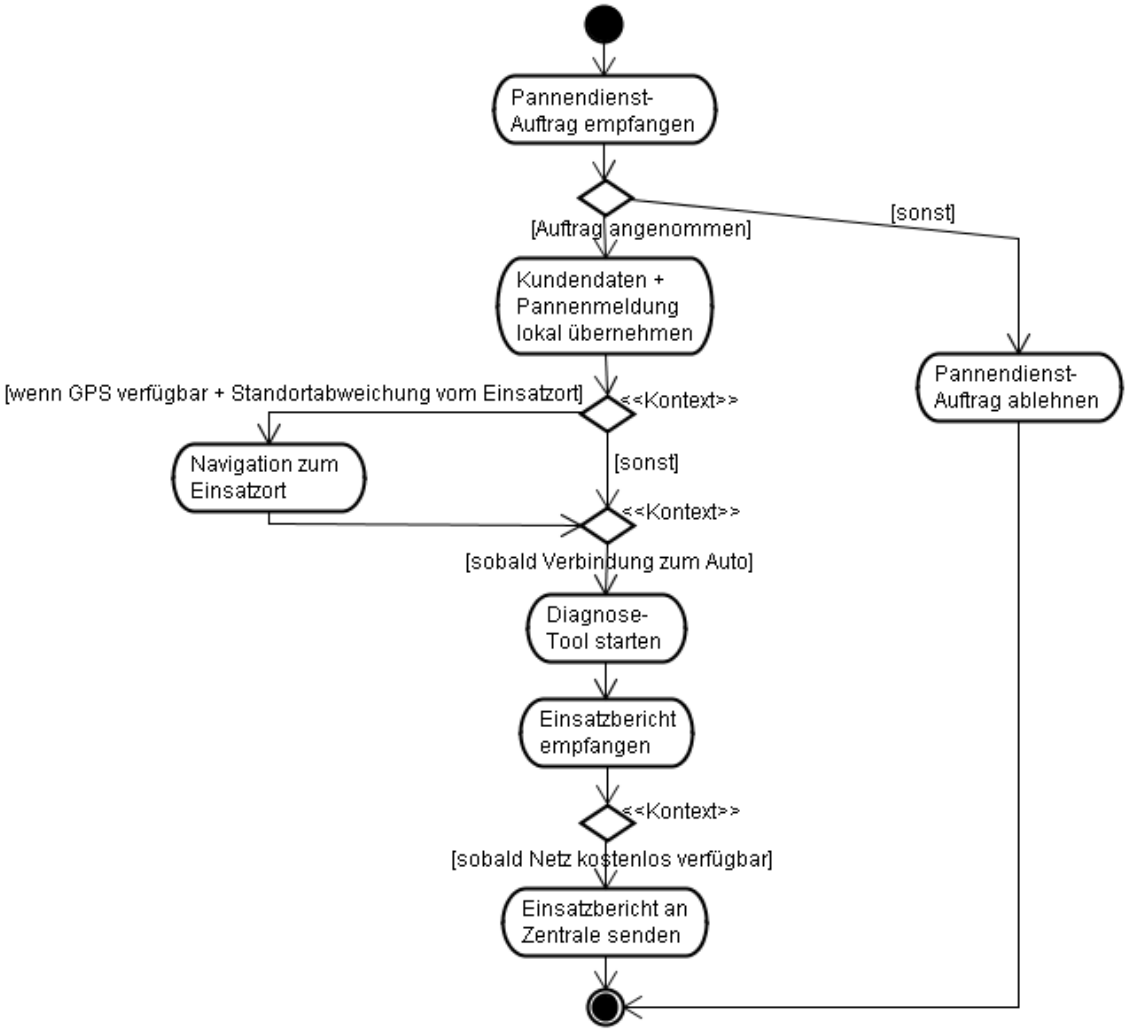

Abbildung 4.3: Aktivitätsdiagramm Sub-Prozess "Pannenhilfe ausführen"

Sub-Prozesse. Sowohl die Übertragung des Sub-Prozesses, an der beide Prozessabläufe kooperierend beteiligt sind, als auch die Möglichkeit, den Auftrag abzulehnen, sind Bestandteile der Anforderungen gemäß der Abschnitte 3.4.1 und 3.5.1. Im Fall der Ablehnung wird eine Auftragsstornierung an die Zentrale gesendet, und der Sub-Prozess ist beendet.

Nimmt der KFZ-Mechaniker dagegen den Auftrag an, werden die erhaltenen Kundendaten und die Pannenmeldung zur Datenübernahme an eine lokale SW-Anwendung weitergeleitet. Darauf folgt die Navigation zum Einsatzort. Laut Szenario handelt es sich hierbei um eine optionale, kontextsensitive Aktivität, die nur ausgeführt wird, wenn das GPS derzeit verfügbar ist und sich der aktuelle Standort vom Einsatzort unterscheidet. Die Ausführung dieser Aktivität ist also vom Ergeb-

nis der vorgelagerten Entscheidung abhängig. Da es sich nicht um eine Entscheidung im eigentlichen Sinne des UML-Aktivitätsdiagramms handelt, sondern um eine kontextsensitive Entscheidung wie in Abschnitt 3.4.2 eingeführt, wird dies im Diagramm mit dem Stereotyp «Kontext» gekennzeichnet. Wenn zum Zeitpunkt der Prozessausführung die Bedingung erfüllt ist, wird die Navigation angestoßen. Andernfalls wird diese Aktivität übersprungen.

In beiden Fällen schließt sich die nächste kontextsensitive Aktivität an. Im Unterschied zur vorangegangenen handelt es sich jedoch nicht um eine optionale Aktivität, deren Ausführung vom derzeit vorherrschenden Kontext abhängig ist, sondern um eine Aktivität, die in jedem Fall ausgeführt werden muss und mit deren Ausführung begonnen wird, sobald ein bestimmter Kontext gegeben ist. Konkret bedeutet dies, dass das Diagnose-Tool auf dem Laptop gestartet wird, sobald das System des Mechanikers eine Datenverbindung zum Kundenfahrzeug aufgebaut hat. Die bestehende Verbindung impliziert für die Prozesslogik, dass der Mechaniker beim Kunden vor Ort eingetroffen ist. Bei dieser Aktivität wird die Prozessausführung solange unterbrochen, bis ein bestimmtes Ereignis eingetroffen ist. Das auslösende Ereignis ist dabei vom umgebenden Kontext abhängig.

Die nächste Aktion ist der Empfang des Einsatzberichtes. Das bedeutet, der Mechaniker hat seine Arbeit vor Ort abgeschlossen und seinen Pannenhilfe-Einsatz in seiner lokalen SW-Anwendung dokumentiert. Diese sendet den Einsatzbericht an den Prozess. Der empfangene Bericht soll an die Zentrale weitergeleitet werden, womit der Sub-Prozess abgeschlossen ist. Auch die Ausführung dieser letzten Aktivität unterliegt dem Szenario entsprechend einer kontextsensitiven Entscheidung. Sie wird erst ausgeführt, sobald ein kostenloser Netzzugang gewährleistet ist.

4.2.3 Interaktion der Komponenten

Um ein Interaktions-Modell entwerfen zu können, muss berücksichtigt werden, wie die im vorhergehenden Abschnitt festgelegten internen Prozessabläufe ”Pannenhilfe beauftragen” und ”Pannenhilfe leisten” ineinander greifen und wie sie mit den übrigen Komponenten in Verbindung stehen, indem sie untereinander entsprechende Nachrichten austauschen. Da beide Abläufe die Möglichkeit vorsehen, dass unterschiedliche Pfade aufgrund laufzeitabhängiger Entscheidungen durchlaufen werden, sind im Resultat ebenfalls unterschiedliche Interaktionsmuster möglich. Im Wesentlichen kann zwischen den beiden Fällen der Auftragsannahme inklusive -durchführung und der Ablehnung mit anschließender Neuvergabe des Auftrags unterschieden werden. Beide Fälle werden im Folgenden untersucht.

Auftragsannahme und -durchführung

Zunächst wird angenommen, dass der im ersten Prozessdurchlauf gewählte Mechaniker den Pannenhilfe-Auftrag annimmt und dieser inklusive der optionalen Navigation zum Einsatzort ausgeführt wird. Über welche Nachrichten der Pannendienst-Service und der KFZ-Mechaniker-Service mit den übrigen Komponenten interagieren, zeigen zwei Sequenzdiagramme: In Abbildung 4.4 liegt der Fokus auf der Kommunikation seitens des Pannendienst-Service; während Abbildung 4.5 den Ausschnitt für den KFZ-Mechaniker-Service darstellt. Die Aufteilung in zwei Diagramme wurde zur besseren Übersichtlichkeit gewählt; logisch greifen beide Sequenzdiagramme ineinander. Die Services der Pannendienst-Zentrale sind in weiß gehalten, die Services auf dem System des Mechanikers grau hervorgehoben. Die für die Übertragung und Ausführung des Sub-Prozesses notwendigen Nachrichten sind ebenfalls grau hinterlegt.

Damit der gesamte Vorgang überhaupt startet, ist der Eingang einer initiierenden Nachricht notwendig; hierzu erhält der Pannendienst-Service vom aufrufenden Client alle notwendigen Daten. Eine Rückmeldung an den Client bei Prozessende ist nicht vorgesehen. Das Ergänzen der Kundendaten über den Stammdaten-Service läuft synchron ab, da beide Services in der Zentrale laufen und mit einer schnellen Antwort gerechnet werden kann.

Der nächste Schritt besteht in der Übertragung des Sub-Prozesses an den KFZ-Mechaniker-Service. Da es sich hierbei um einen langlaufenden Geschäftsprozess handelt, wird der Auftrag asynchron übermittelt. Abbildung 4.5 zeigt die sich anschließende sub-prozessseitige Nachrichtensequenz. Dabei tritt der Anwendungs-Service als Vermittler zwischen dem Sub-Prozess und der SW-Anwendung auf dem Mechaniker-Laptop auf. Erhält dieser die Kundendaten und Pannenmeldung vom KFZ-Mechaniker-Service zur Weitergabe an die lokale SW-Anwendung, wird nach Fertigstellung des Einsatzberichts dieser an den KFZ-Mechaniker-Service übergeben. Da nicht vorhersehbar ist, wie lange der Pannenhilfe-Einsatz dauert, findet die Übertragung asynchron statt. In der Zwischenzeit sendet der KFZ-Mechaniker-Service entsprechende Nachrichten an den Navigations- und den Diagnose-Service. In beiden Fällen ist keine Rückantwort notwendig.

Zum Abschluss des Sub-Prozesses leitet der KFZ-Mechaniker-Service den erhaltenen Einsatzbericht als Resultat an den Pannendienst-Service asynchron weiter. Mit Übermittlung des Auftragabschlusses an den Abrechnungs-Service wird die letzte Nachricht versendet.

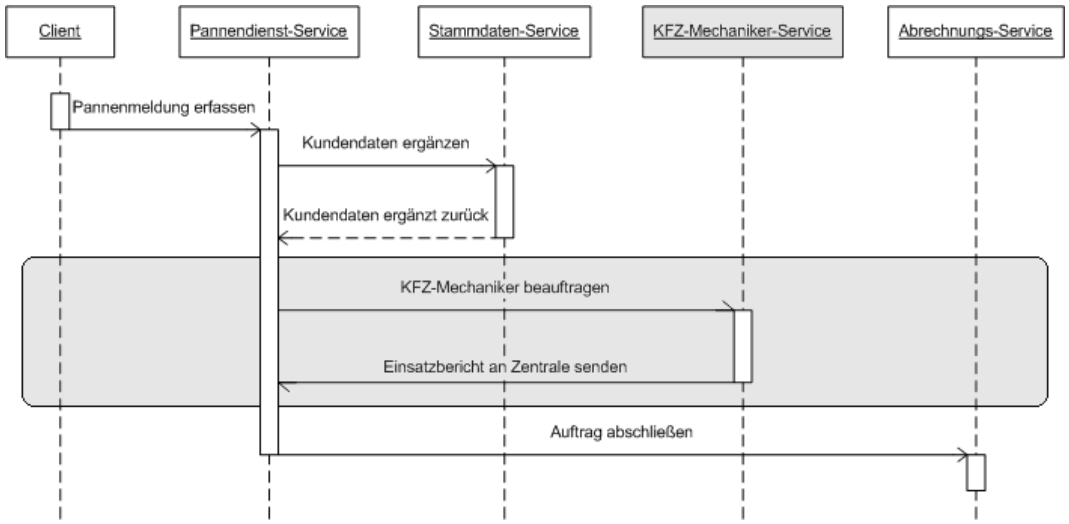

Abbildung 4.4: Sequenzdiagramm Auftragsannahme und -durchführung (Ausschnitt Pannendienst-Zentrale)

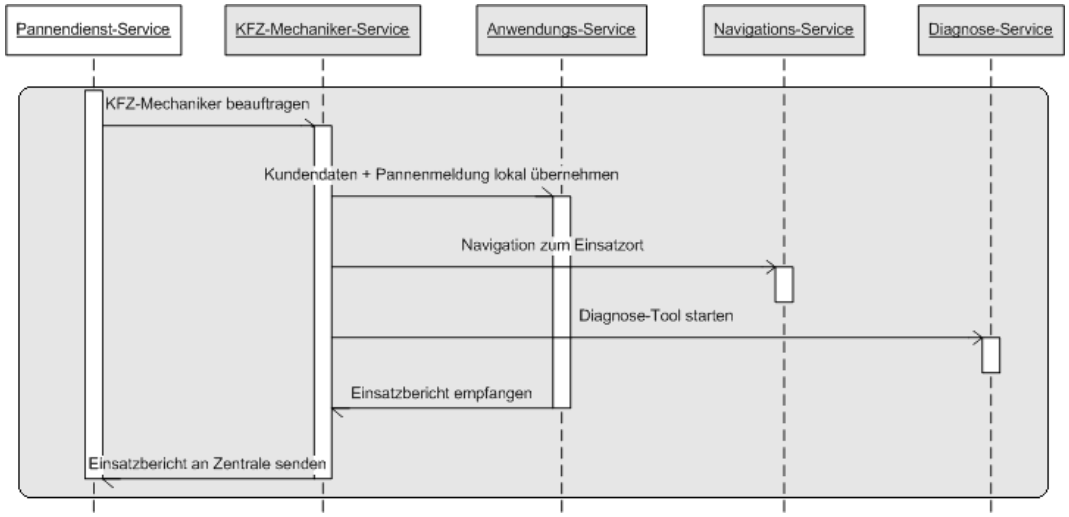

Abbildung 4.5: Sequenzdiagramm Auftragsannahme und -durchführung (Ausschnitt KFZ-Mechaniker)

Auftragsablehnung und -neuvergabe

Eine Abwandlung im Vergleich zum zuvor beschriebenen Nachrichtenaustausch liegt vor, wenn der beauftragte Mechaniker den Panneneinsatz ablehnt und erneut ein Mechaniker bestimmt werden muss. Abbildung 4.6 zeigt für diese Interaktions-Variante das Sequenzdiagramm.

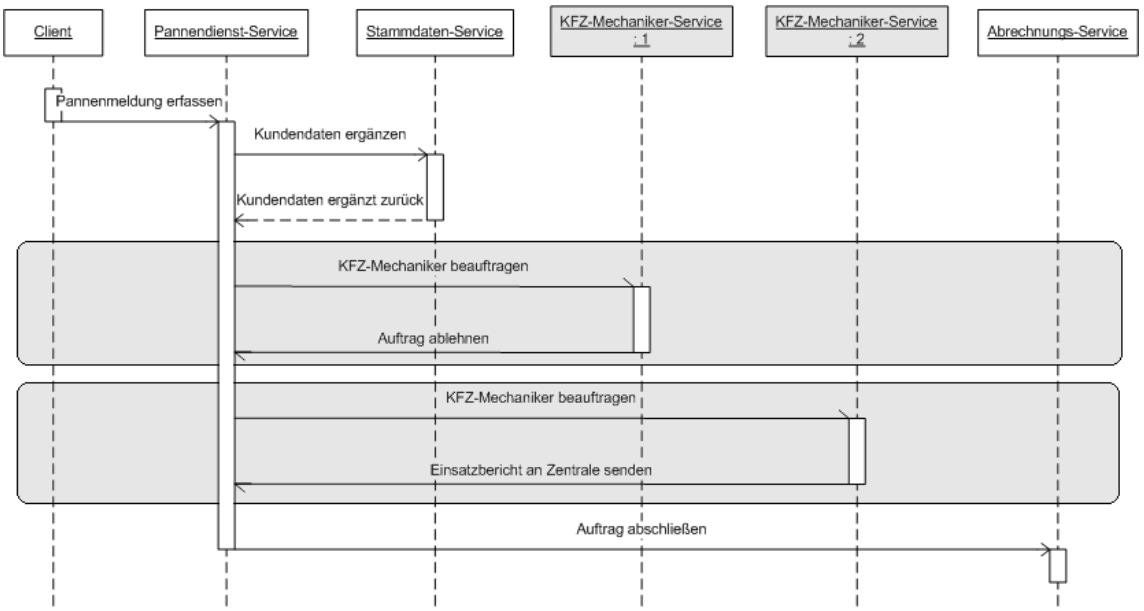

Abbildung 4.6: Sequenzdiagramm Auftragsablehnung und -neuvergabe

Im Unterschied zu vorher sind zwei KFZ-Mechaniker-Services am Vorgang beteiligt. Der zuerst vom Pannendienst-Service aufgerufene lehnt den Auftrag ab, d.h. der übertragene Sub-Prozess wird nicht ausgeführt; lediglich die Auftragsstornierung wird verschickt. Nach Bestimmen eines weiteren KFZ-Mechanikers erhält der zweite Service den Auftrag vom Pannendienst-Service. Dieser führt den Auftrag analog zum zuvor beschriebenen Verfahren aus und endet mit der Rückantwort in Form des Einsatzberichtes an den Pannendienst-Service. Mit Versand der Nachricht zum Abrechnungs-Service ist auch für diese Szenario-Variante der Vorgang abgeschlossen.

4.2.4 Definition der Schnittstellen

Aufbauend auf den Entwurfsvorgaben aus dem vorhergehenden Abschnitt, in dem die laufzeitabhängige Interaktion der Komponenten aufgezeigt wurde, wird im Folgenden die Definition der Schnittstellen für die einzelnen Komponenten vorgenommen. Jede Komponente wird später als Web Service realisiert, wobei die Kompo-

nenten Pannendienst-Service und KFZ-Mechaniker-Service die Prozesslogik entsprechend Abschnitt 4.2.2 umsetzen.

Jede Schnittstellendefinition enthält die ihr zugewiesenen Signaturen und Verweise auf die dafür notwendigen anwendungsspezifischen Datentypen. Zur graphischen Darstellung der Schnittstellen wird das UML-Klassendiagramm verwendet, erweitert um die zwei Stereotypen «web service» und «complex type». Die Schnittstellen dienen als Vorgabe für die Umsetzung als WSDL-Dokumente. Für alle am Szenario beteiligten Services enthält Abbildung 4.7 die Schnittstellendefinitionen.

Der obere Teil der Abbildung – die drei weiß und die vier grau markierten Elemente – zeigt die Web Services. Die vier Elemente im unteren Teil der Abbildung bilden die bei der Nachrichtenübermittlung verwendeten komplexen Datentypen. Dazu gehören *CarBreakdown*, *Customer*, *Address* und *OperationalReport*. `CarBreakdown` enthält die Angaben zur Auto-Panne, `Customer` die Kundendaten und `Address` Adressdaten. `OperationalReport` enthält neben verschiedenen Angaben zum Einsatzbericht eine Referenz auf die Id von `CarBreakdown`. Zum Datentyp `CarBreakdown` gehört je eine Instanz von `Customer` und `Address`. Die Adresse gibt in diesem Fall den aktuellen Standort des Liegengebliebenen an. Weiterhin enthält auch `Customer` eine `Address`-Instanz; hier ist die Anschrift vom Kunden gemeint.

Da die Pannendienst-Zentrale im Rahmen der B2B-Beziehung die Ablaufsteuerung für die Geschäftsprozesse vorgibt, unterliegt es auch ihrer Kontrolle, Vorgaben für die bereitzustellenden Web Services zu machen. Die Zentrale gibt demnach nicht nur vor, welche Services in ihrer IT-Landschaft genutzt werden – in der Abbildung weiß markiert –, sondern auch, welche Services die Mechaniker zur Verfügung stellen müssen – grau hinterlegt dargestellt –, wie beispielsweise den Navigations- oder Anwendungs-Service.

Zunächst werden die Service-Schnittstellen für die Pannendienst-Zentrale betrachtet: Der Stammdaten-Service bzw. `MasterDataService` muss die synchrone Operation `completeCustomer` implementieren, die das Vervollständigen der Kundendaten vornimmt. Der Abrechnungs-Service bzw. `AccountingService` übernimmt die Abrechnung der abgeschlossenen Pannenhilfe mittels der Operation `closeBreakdownOrder`; es wird keine Rückantwort generiert. Als oberste Instanz fungiert der Pannendienst-Service bzw. `CarRepairService`. Dieser nimmt die Pannenmeldung initial über die Operation `enterBreakdownNotification` entgegen. Auch dies geschieht ohne Rückantwort.

Zur Übernahme des Pannenhilfe-Auftrags muss der KFZ-Mechaniker-Service bzw. `MotorMechanicService` die Operation `placeBreakdownOrder` implementieren. Das Übermitteln der asynchronen Antwort geschieht über die Operation `returnOperationalReport`. Für den Fall, dass der Auftrag abgelehnt wird,

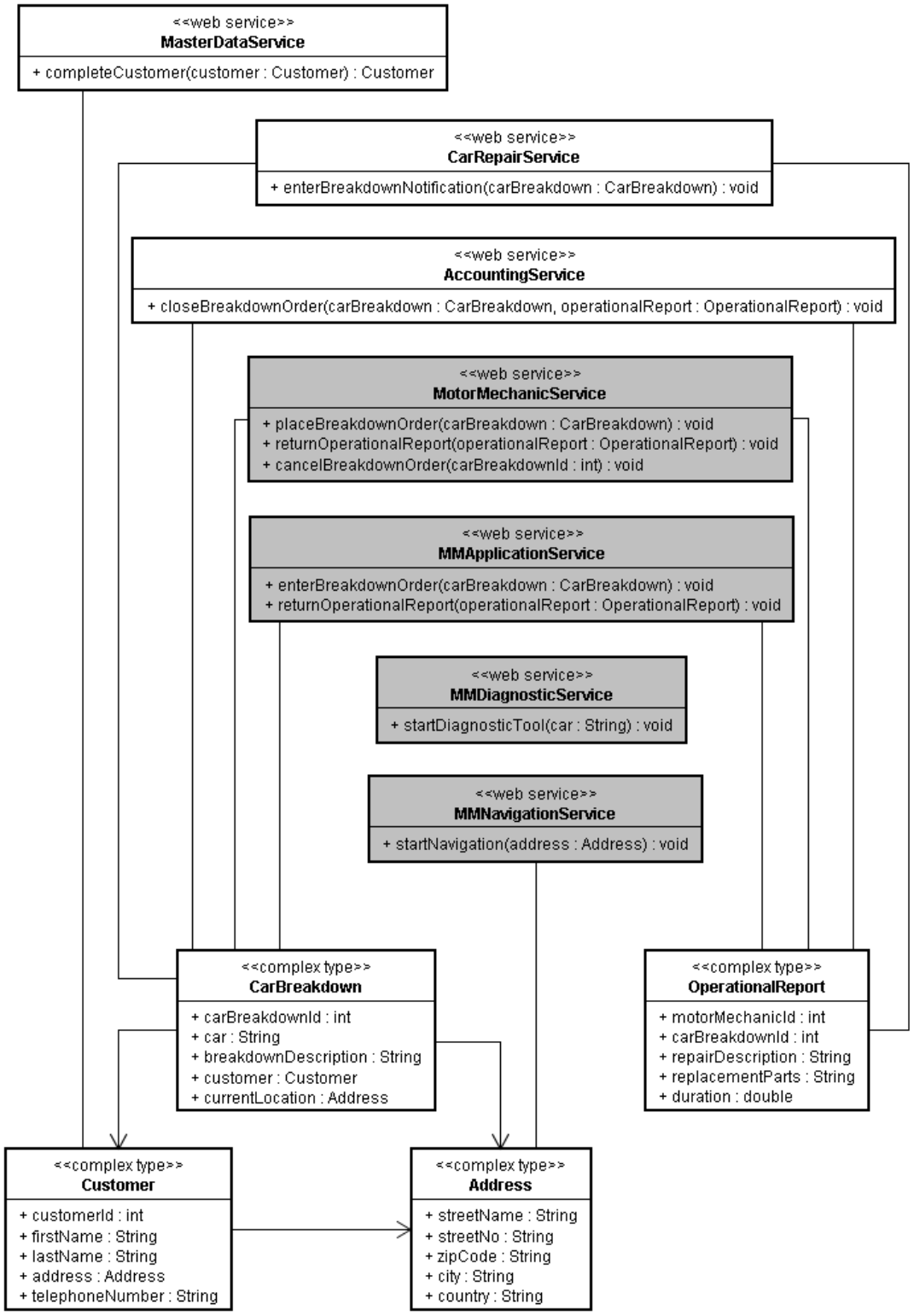

Abbildung 4.7: Schnittstellendefinition der Prozess-Komponenten

wird als Rückmeldung die Operation `cancelBreakdownOrder` verwendet. Neben dem `MotorMechanicService` muss jeder Mechaniker drei weitere Services anbieten, deren englische Bezeichnungen alle mit '`MM`' – für MotorMechanic – beginnen. Der Anwendungs-Service bzw. `MMApplicationService` stellt zwei Operationen zur asynchronen Kommunikation mit der lokalen SW-Anwendung zur Verfügung: `enterBreakdownOrder` zum Übertragen des Pannenhilfe-Auftrags und `returnOperationalReport` für die Rückantwort in Form des Einsatzberichts. Der Navigations-Service bzw. `MMNavigationService` definiert die Operation `startNavigation` zum Starten der Navigation. Über die Operation `startDiagnosticTool` des Diagnose-Service bzw. `MMDiagnosticService` lässt sich das Diagnose-Tool starten.

4.3 Konzeption der Sprache

Die standardisierte Version WS-BPEL 2.0 dient als Basis für die Entwicklung der Sprache zur Unterstützung mobiler Workflows. Im Folgenden wird ein Konzept erarbeitet, womit sich die in Abschnitt 3.5.1 aufgestellten Anforderungen durch Erweiterung des BPEL-Sprachumfangs erfüllen lassen.

4.3.1 Grundlagen mobiler BPEL-Aktivitäten

Ein Großteil der Geschäftsprozesse für das Szenario "Auto-Pannendienst" kann unter Verwendung von Standard-BPEL-Elementen umgesetzt werden, wie beispielsweise der asynchronen Kommunikation mit einem Partner-Service oder dem Versenden einer Nachricht zur Auftragsablehnung. Um den darüber hinaus gehenden Anforderungen gerecht zu werden, sind Erweiterungen der Sprache BPEL notwendig. Im Rahmen der Analyse wurden bereits die konkreten Anforderungen herausgearbeitet, die im Rahmen dieser Arbeit als Beschreibungssprache umgesetzt werden sollen. Folgende Aspekte sollen Abschnitt 3.5.1 entsprechend mittels geeigneter Sprachkonstrukte formal beschreibbar sein:

1. Definition eines Sub-Prozesses und dessen Übertragung zur Laufzeit an andere Ausführungsumgebung
2. Empfang eines Sub-Prozesses und dessen Ausführung
3. Auswahl und dynamisches Binding eines Service-Providers abhängig vom aktuellen Kontext
4. Ausführung einer (ggf. optionalen) Aktivität abhängig vom aktuellen Kontext

Allen vier Anforderungen gemeinsam ist, dass es sich im Sinne von BPEL-Prozessen um Aktivitäten handelt (vgl. Spezifikation [BPE-2007]). Unter Berücksichtigung des BPEL-Standard-Erweiterungsmechanismus – der einleitend in Abschnitt 3.1.1 erklärt wurde – lassen sich eigene Elemente zur Formulierung der genannten Anforderungen entwickeln. Hierbei wird auf das dort beschriebene Element `<extensionActivity>` zurückgegriffen und für alle Erweiterungen ein gemeinsamer Namensraum zur Qualifizierung eingeführt: `"http://informatik.haw-hamburg.de/master/wsbpel/2.0/mobileExtensions"`. Weiterhin soll nach Möglichkeit beim Entwurf der eigenen Elemente das Prinzip des BPEL-Erweiterungsmechanismus fortgeführt werden, d.h. die im Rahmen dieser Arbeit entwickelten Sprachkonstrukte können selbst wieder erweitert werden. Dies wird speziell bei Betrachtung der im Folgenden erstellten XML-Schemata deutlich (, wobei Ideen dazu aus der BPEL-Erweiterung BPEL4People aufgegriffen wurden [4Pe-2007]).

Bevor mit der Definition der neuen BPEL-Aktivitäten begonnen werden kann, muss geklärt werden, wie viele und welche Art von Erweiterungen notwendig sind. Die vier oben aufgelisteten Anforderungen lassen sich zunächst in zwei Kategorien einteilen: Punkte 1 und 2 realisieren in Kombination die übertragbaren Sub-Prozesse, während die Punkte 3 und 4 beide auf Berücksichtigung des Kontexts basieren. In jedem Fall sollte jede neu geschaffene Aktivität für sich alleine stehen können, sinnvoll in einen beliebigen Prozessablauf integriert werden können und für den anwendenden Prozess-Entwickler verständlich und handhabbar sein.

Eine erste Idee zur Wahrung der Minimal-Anforderung bezüglich der Sub-Prozesse – also Punkte 1 und 2 – zeigt, dass hierfür mindestens zwei BPEL-Aktivitäten notwendig sind: eine Aktivität zum Definieren und Senden des Sub-Prozesses und eine weitere Aktivität zum Empfangen und Ausführen des Sub-Prozesses. Erstere wird im Rahmen der Prozessbeschreibung für die Pannendienst-Zentrale benötigt; die zweite muss auf dem System des KFZ-Mechanikers ausgeführt werden, um den Auftrag entgegen zu nehmen. Abschnitt 4.3.4 befasst sich damit weitergehend.

Die Anforderungen der Punkte 3 und 4 weisen die Gemeinsamkeit auf, dass beide den aktuellen Kontext in die Prozessausführung integrieren. In beiden Fällen muss eine geeignete Modellierung des Kontexts ermöglicht werden. Bei der Entscheidung, wie diese beiden Punkte modelliert werden sollten und ob dies als eine kombinierte BPEL-Aktivität oder in Form von zwei Aktivitäten geschieht, sollen die folgenden Überlegungen helfen.

Betrachtet werden zunächst die Unterschiede der Punkte 3 und 4: Punkt 4 macht die Ausführung einer referenzierten Aktivität davon abhängig, ob bestimmte Kontextbedingungen erfüllt sind. Dabei kann es sich prinzipiell um jede beliebige BPEL-Aktivität handeln, deren eigene Semantik aber nicht verändert wird. Punkt 3 bezieht sich dagegen ausschließlich auf den entfernten Operationsaufruf eines Partner-Service; speziell auf die kontextabhängige Auswahl eines Service-Providers, der über die definierte `<invoke>`-Aktivität referenziert wurde. Unter Beachtung der BPEL-Spezifikation [BPE-2007], die besagt, dass die Semantik standardisierter BPEL-Elemente durch Erweiterungen nicht verändert werden darf, sollte in diesem Zusammenhang für Punkt 3 über die Einführung einer spezialisierten `<invoke>`-Erweiterung nachgedacht werden.

Gemäß Szenario soll die Kombination einiger der vier Anforderungen möglich sein, d.h. die XML-basierten BPEL-Erweiterungen müssen ineinander schachtelbar sein. Beispielsweise verwendet Punkt 4 intern die Standard-BPEL-`<invoke>`-Aktivität, um die kontextsensitiven Aktivitäten im mechanikerseitigen Sub-Prozess umzusetzen. Die Punkte 1 und 3 realisieren kombiniert die Auftragsübermittlung als Sub-Prozess an einen zur Laufzeit bestimmten Mechaniker. Über die Szenario-Vorgaben hinaus sind weitere Kombinationen denkbar. Da Punkt 4 eine beliebige BPEL-

Aktivität enthalten kann, sollte es ebenfalls möglich sein, die Erweiterungen Punkte 1, 2 oder 3 in sich aufzunehmen.

Zusammengefasst weisen die Punkte 3 und 4 neben der übereinstimmenden Kontextsensitivität ebenso Unterschiede auf. Diese Tatsache würde einer kombinierten Aktivität Besonderheiten und Ausnahmen in der Verwendung auferlegen. Verbunden mit dem Ziel, alle Aktivitäten miteinander kombinieren zu können, führte dies zu dem Entschluss, für Punkte 3 und 4 zwei getrennte Spracherweiterungen zu entwerfen. Die Abschnitte 4.3.2 und 4.3.3 behandeln diese Thematik im Anschluss.

4.3.2 Dynamische, kontextabhängige Service-Auswahl

Wie schon in den grundlegenden Betrachtungen zur BPEL-Erweiterung erwähnt, soll mittels einer Aktivität die Auswahl eines Service-Providers abhängig vom jeweiligen Kontext getroffen und dieser dynamisch an den Operationsaufruf gebunden werden.

Beginnend sollen kurz die wichtigsten Eigenschaften der im Standard spezifizierten Aktivität `<invoke>` betrachtet werden (weitergehende Informationen siehe Spezifikation [BPE-2007]). `<invoke>` wird dazu verwendet, um eine Operation eines in den Prozess involvierten Web Service aufzurufen. BPEL baut zu diesem Zweck auf der Schnittstellenbeschreibungssprache WSDL 1.1 auf [WSD-2001]. Hier von Interesse ist die Definition von `operation` und `portType` mittels eines WSDL-Dokuments. Eine BPEL-Prozessbeschreibung referenziert diese abstrakte Schnittstellenbeschreibung durch die `<invoke>`-Attribute `operation` und `portType`. Die Interaktion mit einem Partner-Web Service beschreibt BPEL zusätzlich über einen `partnerLink`, der einem konkreten `partnerLinkType` zugewiesen wird und über diese WSDL-Erweiterung auf die entsprechenden `portTypes` verweist.

Zum Zeitpunkt der Prozess-Modellierung reicht demnach die abstrakte Schnittstellenbeschreibung aus. Normalerweise geschieht das Binding an einen konkreten Service-Endpunkt spätestens beim Deployment des BPEL-Prozesses. An dieser Stelle greift die Erweiterung ein, da das Binding dynamisch zur Laufzeit des Prozesses durch kontextabhängige Auswahl eines Service-Providers geschehen soll.

BPEL-Erweiterung `<dynamicInvoke>`

Die zu entwerfende BPEL-Aktivität muss zu diesem Zweck mindestens dieselben Attribute und Elemente enthalten wie das Standard-`<invoke>` – ergänzt um eine geeignete Kontextabbildung. Das neue Element *`<dynamicInvoke>`* wird diese Aufgabe übernehmen. Die Syntax der BPEL-Erweiterung `<dynamicInvoke>` zeigt Listing 4.1.

```
<extensionActivity>
   <dynamicInvoke
    partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    inputVariable="BPELVariableName"?
    outputVariable="BPELVariableName"?
    standard-attributes>
       standard-elements
        <correlations>?
           <correlation set="NCName" initiate="yes|join|no"?
              pattern="request|response|request-response"? />+
        </correlations>
        <catch faultName="QName"?
           faultVariable="BPELVariableName"?
           faultMessageType="QName"?
           faultElement="QName"?>*
           activity
        </catch>
        <catchAll>?
           activity
        </catchAll>
        <compensationHandler>?
           activity
        </compensationHandler>
        <toParts>?
           <toPart part="NCName" fromVariable="BPELVariableName"
              />+
        </toParts>
        <fromParts>?
           <fromPart part="NCName" toVariable="BPELVariableName"
              />+
        </fromParts>
        <select>
           context+
        </select>
   </dynamicInvoke>
</extensionActivity>
```

Listing 4.1: Syntax der BPEL-Erweiterung `<dynamicInvoke>`

Neben den zuvor erwähnten Attributen `partnerLink`, `portType` und `operation` weist `<dynamicInvoke>` dieselbe Syntax wie das Standard-`<invoke>` auf. Neu hinzugekommen ist das Element `<select>`, welches zur Beschreibung des zu berücksichtigenden Kontexts verwendet wird. Bei `context` handelt es sich zunächst um einen Platzhalter, der im Folgenden beschrieben wird.

Kontext-Modellierung

Sowohl die Aktivität `<dynamicInvoke>` als auch die im nachfolgenden Abschnitt 4.3.3 vorgestellte kontextsensitive Aktivität benötigen ein Instrument zur Kontextabbildung. Im Rahmen der sprachlichen Konzeption muss hierfür eine formale Beschreibungsmöglichkeit entworfen werden. Die Interpretation und Berücksichtigung der Kontextvorgaben übernimmt anschließend die ausführende Engine.

Zwar macht das Beispiel-Szenario "Auto-Pannendienst" Vorgaben, was als Kontext zu modellieren ist, jedoch sollte nach Möglichkeit eine Lösung gefunden werden, mittels der sich eine Vielzahl von kontextuellen Eigenschaften formal beschreiben lässt. In Verbindung mit `<dynamicInvoke>` soll gemäß Szenario ein Mechaniker ausgewählt werden, der für den jeweiligen Panneneinsatz geeignet ist. Um eine erste Idee davon zu erlangen, wie diese Kontexterweiterung aussehen könnte, wird zunächst nur die Pannenbeschreibung und der Einsatzort berücksichtigt.

Eine Möglichkeit besteht darin, den Kontext in Form von Schlüssel-Werte-Paaren zu beschreiben. Das `<select>`-Element des zugehörigen `<dynamicInvoke>` sähe beispielsweise wie in Listing 4.2 aus.

```
<select>
  <context name="breakdownDescription" value="engine breakdown"/>
  <context name="destinationStreetName" value="country road"/>
  <context name="destinationStreetNo" value="3a"/>
  <context name="destinationZipCode" value="12345"/>
  <context name="destinationCity" value="capital city"/>
  <context name="destinationCountry" value="D"/>
</select>
```

Listing 4.2: Kontext-Modellierung Möglichkeit 1

Möglichkeit 1 beinhaltet eine simple Aneinanderreihung von `<context>`-Elementen. Über die Attribute `name` und `value` werden die kontextuellen Eigenschaften beschrieben. Ein Vorteil dieser Variante besteht in der sehr einfachen Syntax. Jedoch sind komplexere Kontextabbildungen für den Entwickler schwer zu überblicken und die Interpretation unnötig kompliziert, da wie im Beispiel gezeigt fünf unzusammenhängende `<context>`-Elemente den Einsatzort modellieren.

Eine diese Nachteile auflösende Variante zeigt Möglichkeit 2. Listing 4.3 enthält hierfür ein Beispiel. Der Vorteil gegenüber der ersten Möglichkeit besteht darin, dass logisch zusammengehörige `<property>`-Elemente in einem übergeordneten `<context>`-Element zusammengefasst werden. Das Verständnis für den Entwickler und die Interpretation für die Engine wird vereinfacht.

```
<select>
  <context name="breakdown">
    <property name="description" value="engine breakdown"/>
  </context>
  <context name="destination">
     <property name="streetName" value="country road"/>
     <property name="streetNo" value="3a"/>
     <property name="zipCode" value="12345"/>
     <property name="city" value="capital city"/>
     <property name="country" value="D"/>
  </context>
</select>
```

Listing 4.3: Kontext-Modellierung Möglichkeit 2

Den Möglichkeiten 1 und 2 gemeinsam ist, dass sie auf Schlüssel-Werte-Paaren basieren und somit eine einfache Syntax aufweisen. Möglichkeit 2 vereinfacht die Auswertung für die Engine, indem zu einer kontextuellen Ausprägung gehörende Eigenschaften in einer übergeordneten Entität zusammengefasst werden. Beide Varianten geben eine einheitliche Struktur zur Abbildung jeden beliebigen Kontexts vor.

Erste Überlegungen, wie sich diese Kontext-Modelle später durch die Engine interpretieren lassen, führen jedoch zu dem Entschluss, diese einheitliche Struktur aufzulösen. Schließlich muss jede Kontextangabe von der ausführenden Engine ausgewertet werden können, wofür in jedem Fall eine Erweiterung der Engine notwendig ist. Es genügt nicht, dass der Prozess-Entwickler aufgrund der einfachen Struktur eine neue Kontexteigenschaft hinzufügen kann. Die Engine muss diese neue Angabe auch verstehen können. Da somit jede Ergänzung der Kontextbeschreibung eine Erweiterung der Engine nach sich zieht, ist es sinnvoller, den jeweiligen Kontext direkt abzubilden. Listing 4.4 zeigt hierfür ein Beispiel.

```
<select>
   <motorMechanic>
     <breakdownDescription>"flat tyre"</breakdownDescription>
     <destination>
       <streetName>"country road"</streetName>
       <streetNo>"3a"</streetNo>
       <zipCode>"12345"</zipCode>
       <city>"capital city"</city>
       <country>"D"</country>
     </destination>
   </motorMechanic>
</select>
```

Listing 4.4: Kontext-Modellierung Möglichkeit 3

Angelehnt an das Szenario soll ein Mechaniker ausgewählt werden, der für die jeweilige Panne geeignet ist und sich in der Nähe des Einsatzortes befindet. Dieser Kontext lässt sich mit der dritten aufgezeigten Möglichkeit modellieren, die darüber hinaus eine gute Voraussetzung für die Integration in die Engine bietet. Diese Variante soll für die weitere Konzeption der BPEL-Erweiterungen verwendet werden.

Kontext-Integration in BPEL-Erweiterung `<dynamicInvoke>`

Die Syntax der neuen Aktivität `<dynamicInvoke>` (vgl. Listing 4.1) wurde bisher mit einem Platzhalter für die Kontext-Modellierung beschrieben. Die im vorhergehenden Abschnitt entwickelte Kontextabbildung soll nun darin integriert werden. Listing 4.5 zeigt ein Beispiel für eine komplette `<dynamicInvoke>`-Aktivität angelehnt an das Szenario "Auto-Pannendienst". Eine ausführliche Syntaxbeschreibung kann den XML-Schemata in Listings A.1 und A.2 in Anhang A entnommen werden.

```
<bpel:extensionActivity>
   <mobile:dynamicInvoke name="InvokePlaceOrderAtMotorMechanic"
      partnerLink="MotorMechanicRequestingPL"
      portType="mms:MotorMechanicServicePT"
      operation="placeBreakdownOrder"
      inputVariable="breakdownOrderForMotorMechanic">
     <mobile:select>
        <mobile:motorMechanic>
          <mobile:car>"little van"</mobile:car>
          <mobile:breakdownDescription>"engine breakdown"
             </mobile:breakdownDescription>
          <mobile:duration>2.0</mobile:duration>
          <mobile:destination>
            <mobile:streetName>"country road"
               </mobile:streetName>
            <mobile:streetNo>"3a"</mobile:streetNo>
            <mobile:zipCode>"12345"</mobile:zipCode>
            <mobile:city>"capital city"</mobile:city>
            <mobile:country>"D"</mobile:country>
          </mobile:destination>
        </mobile:motorMechanic>
     </mobile:select>
   </mobile:dynamicInvoke>
</bpel:extensionActivity>
```

Listing 4.5: Beispiel für BPEL-Erweiterung `<dynamicInvoke>`

Bei den bisherigen Betrachtungen zur Kontext-Modellierung wurde lediglich die statische Abbildung berücksichtigt, nicht aber ob sich die Erweiterung problemlos in den Ablauf eines BPEL-Prozesses eingliedert. Die jetzige Lösung setzt voraus, dass die konkreten Werte, die den Kontext beschreiben, schon zur Designzeit des Prozesses feststehen und statisch abgebildet werden können. In einigen Situationen ist dieses Vorgehen ausreichend, wie folgende Vorgabe aus dem Szenario zeigt: Der Einsatzbericht soll vom KFZ-Mechaniker zurück an die Pannendienst-Zentrale gesendet werden, sobald ein kostenloser Netzzugang verfügbar ist. Diese Anforderung lässt sich zur Entwicklungszeit vollständig beschreiben. Anders sieht es bei der Wahl des zuständigen Mechanikers aus; für jede Prozessinstanz können sich die zu berücksichtigenden Vorgaben unterscheiden. Welchen Wert sie einnehmen, entscheidet sich erst zur Laufzeit. Für diese Anforderung ist die bisherige Lösung nicht ausreichend.

Mit dieser Problematik, in einem ähnlichen Zusammenhang, befasst sich bereits die BPEL-Spezifikation selbst, so dass die folgende Alternative daran angelehnt entwickelt wurde [BPE-2007]. Wenn die Kontextvorgaben erst zur Laufzeit feststehen, wird der Kontext in Form einer Variablen an die entsprechende Aktivität übergeben. Die Variable muss dabei der Syntax einer vorgegebenen statischen Kontextstruktur entsprechen und kann im vorhergehenden Prozessverlauf durch Verwendung der BPEL-Aktivität `<assign>` mit beliebigen Werten gefüllt werden. Unter der Annahme, dass eine solche Variable mit dem Namen `"motorMechanicContext"` existiert, zeigt Listing 4.6 ein entsprechendes Beispiel.

```
<bpel:extensionActivity>
   <mobile:dynamicInvoke name="InvokePlaceOrderAtMotorMechanic"
      partnerLink="MotorMechanicRequestingPL"
      portType="mms:MotorMechanicServicePT"
      operation="placeBreakdownOrder"
      inputVariable="breakdownOrderForMotorMechanic">
     <mobile:select>
       <mobile:motorMechanic variable="motorMechanicContext" />
     </mobile:select>
   </mobile:dynamicInvoke>
</bpel:extensionActivity>
```

Listing 4.6: Beispiel für BPEL-Erweiterung `<dynamicInvoke>`

4.3.3 Kontextabhängige Ausführung einer Aktivität

Wie die grundlegenden Betrachtungen zur BPEL-Erweiterung in Abschnitt 4.3.1 gezeigt haben, soll neben der kontextsensitiven Aktivität `<dynamicInvoke>` eine weitere BPEL-Erweiterung den aktuellen Kontext berücksichtigen. Gemeint ist eine Erweiterung, die eine referenzierte Aktivität nur ausführt, sofern der angegebene Kontext erfüllt ist.

In gewisser Weise kann darüber der Ablauf des Prozesses gesteuert werden. Ähnlich wie beispielsweise die `<if>`-Aktivität steuert diese Spracherweiterung den sequentiellen Ablauf. Im Gegensatz zu den im BPEL-Standard definierten Structured Activities werden hierfür nicht im Prozess deklarierte Variablen verwendet, sondern der umgebende Kontext ausgewertet. Diese Anforderung wird durch das neue Element *`<contextActivity>`* umgesetzt.

BPEL-Erweiterung `<contextActivity>`

Wie schon in Abschnitt 3.5.1 aufgelistet, kann die BPEL-Erweiterung `<contextActivity>` zur Prozessbeschreibung gemäß Szenario "Auto-Pannendienst" in drei Fällen zum Einsatz kommen:

1. optionale Navigation zum Einsatzort, wenn GPS verfügbar ist und sich der aktuelle Standort vom Einsatzort unterscheidet
2. Starten des Diagnose-Tools, sobald eine Verbindung zum Auto besteht
3. Senden des Einsatzberichts, sobald ein kostenloser Netzzugang verfügbar ist

In jedem dieser Fälle soll der Operationsaufruf eines Partner-Service per Standard-`<invoke>` nur ausgeführt werden, sofern die jeweilige kontextuelle Bedingung erfüllt ist. Dabei lassen sich folgenden Eigenschaften feststellen:

In Fall 1 wird die Navigation nur ausgeführt, *wenn* die Bedingung erfüllt ist. Hier bezieht sich die Kontextangabe auf eine *Momentaufnahme*. Wenn zu dem Zeitpunkt, an dem der Prozess mit der Ausführung der `<contextActivity>` beginnt, die Bedingung wahr ist, wird das eingeschlossene `<invoke>` ausgeführt. Da die `<contextActivity>` zudem optional ist, wird im Falle der nicht erfüllten Bedingung, das `<invoke>` übersprungen und mit der auf die `<contextActivity>` folgenden Aktivität fortgefahren.

In den Fällen 2 und 3 wird die eingeschlossene Aktion ausgeführt, *sobald* die Bedingung erfüllt ist. Hier wird im Gegensatz zu Fall 1 keine Momentaufnahme herangezogen, sondern die zu erfüllenden Eigenschaften werden laufend mit dem umgebenden,

aktualisierten Kontext verglichen. Über einen *Event-Service*, der anzeigt, dass der Kontext den gewünschten Eigenschaften entspricht, wird die Fortsetzung des Prozesses angestoßen und die eingeschlossene `<invoke>`-Aktivität ausgeführt.

Die `<contextActivity>` muss diese Eigenschaften abbilden können und anschließend von der Engine entsprechend interpretiert werden. Das Szenario sieht in allen drei Fällen das Standard-`<invoke>` als eingeschlossene Aktivität vor. Dennoch ist jede andere BPEL-Aktivität denkbar. Die Syntax der BPEL-Erweiterung `<contextActivity>` zeigt Listing 4.7. Eine Syntaxbeschreibung als XML-Schemata enthalten die Listings A.1 und A.2 in Anhang A.

```
<extensionActivity>
  <contextActivity mode="must|optional"?
  standard-attributes>
    standard-elements
    <requirements mode="now|onEvent"?>?
      context+
    </requirements>
    activity
  </contextActivity>
</extensionActivity>
```

Listing 4.7: Syntax der BPEL-Erweiterung `<contextActivity>`

Neben den `standard-attributes` enthält das Element `<contextActivity>` ein Attribut `mode`. Es kann die Werte `"must"` oder `"optional"` annehmen, wobei `"must"` für den Fall, dass keine Angabe gemacht wurde, der Default-Wert ist. Über die `mode`-Angabe kann gesteuert werden, ob die Ausführung der geschachtelten Aktivität Pflicht oder optional ist.

Dem BPEL-Standard entsprechend weist das Element `<contextActivity>` die `standard-elements` auf. Darüber hinaus enthält es ein Element `<requirements>`, das intern die Kontextbedingungen beschreibt, und ein `activity`-Element. Wobei `activity` an dieser Stelle ein Platzhalter für eine beliebige BPEL-Aktivität darstellt. Die `<requirements>` beinhalten ebenfalls ein Attribut `mode`. Dieses kann die Werte `"now"` oder `"onEvent"` annehmen. Letzteres ist der Default-Wert. `"now"` wird verwendet, um anzuzeigen, dass es sich um eine Momentaufnahme vom Kontext handelt, während `"onEvent"` markiert, dass die Engine den Event-Mechanismus verwenden muss.

Bei `context` handelt es sich ebenfalls um einen Platzhalter, der analog zum in Verbindung mit der `<dynamicInvoke>`-Aktivität in Abschnitt 4.3.2 beschriebenen Verfahren modelliert wird.

Beispiele für BPEL-Erweiterung `<contextActivity>`

Zwei Beispiele aus dem vorgegebenen Szenario sollen veranschaulichen, wie die `<contextActivity>` genutzt wird. Das erste Beispiel bezieht sich auf die weiter oben mit Fall 1 bezeichnete optionale Navigation zum Einsatzort. Listing 4.8 gibt hierfür den BPEL-Code wieder.

```
<bpel:extensionActivity>
   <mobile:contextActivity name="HandleOptionalNavigation"
      mode="optional">
      <mobile:requirements mode="now">
         <mobile:gps>
            <mobile:available>"yes"</mobile:available>
         </mobile:gps>
         <mobile:gpsPositionDeviation
            variable="distanceAndDestinationAddress" />
      </mobile:requirements>
      <bpel:invoke name="InvokeStartNavigation"
         partnerLink="MMNavigationRequestingPL"
         portType="mmns:MMNavigationServicePT"
         operation="startNavigation"
         inputVariable="startNavigationRequest" />
   </mobile:contextActivity>
</bpel:extensionActivity>
```

Listing 4.8: Beispiel für BPEL-Erweiterung `<contextActivity>`

Die `mode`-Angabe der `<requirements>` gibt an, dass es sich um eine Momentaufnahme handelt. Enthalten sind zwei Kontextelemente: `<gps>` und `<gpsPositionDeviation>`. Die Angabe für `<gps>` ist statisch festgelegt, während die Werte für `<gpsPositionDeviation>` – Einsatzort und maximale Standortabweichung – dynamisch zur Laufzeit aus der angegebenen Variable ausgelesen werden.

Das zweite Beispiel kann Listing 4.9 entnommen werden. Hier handelt es sich um den Fall 2: Das Diagnose-Tool soll gestartet werden, sobald eine Verbindung zum Auto besteht. Der `mode` der `<requirements>` ist demnach auf `"onEvent"` gesetzt, und für die statische Kontextbeschreibung steht das Element `<carConnection>` zur Verfügung.

```
<bpel:extensionActivity>
   <mobile:contextActivity name="HandleDiagnosticTool"
      mode="must">
      <mobile:requirements mode="onEvent">
```

```
        <mobile:carConnection>
          <mobile:available>"yes"</mobile:available>
        </mobile:carConnection>
      </mobile:requirements>
    <bpel:invoke name="InvokeStartDiagnosticTool"
       partnerLink="MMDiagnosticRequestingPL"
       portType="mmds:MMDiagnosticServicePT"
       operation="startDiagnosticTool"
       inputVariable="startDiagnosticToolRequest" />
  </mobile:contextActivity>
</bpel:extensionActivity>
```

Listing 4.9: Beispiel für BPEL-Erweiterung `<contextActivity>`

4.3.4 Übertragbare Sub-Prozesse

Nachdem bisher die beiden kontextsensitiven Erweiterungsaktivitäten entworfen wurden, beschäftigt sich dieser Abschnitt mit den noch offenen Anforderungen bezüglich der übertragbaren Sub-Prozesse. Der BPEL-Standard sieht die Kommunikation zweier Prozesse über Web Service-Aufrufe vor. Dadurch kann ein bestehender BPEL-Prozess in die Ausführung eines übergeordneten Prozesses integriert werden. Abbildung 4.8 zeigt den zugrundeliegenden Aufbau für eine derartige Zusammenarbeit.

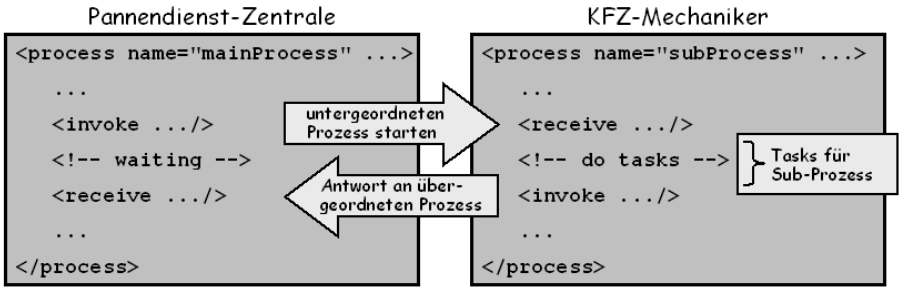

Abbildung 4.8: Aufruf eines untergeordneten BPEL-Prozesses

Angelehnt an das Szenario "Auto-Pannendienst" würde der untergeordnete Prozess den Ablauf des Sub-Prozesses seitens des KFZ-Mechanikers darstellen; der übergeordnete Prozess den Ablauf für die Pannendienst-Zentrale widerspiegeln. Für den dargestellten Aufbau ist es jedoch notwendig, dass der untergeordnete Prozess als eigenständige Definition bereits deployed und somit auf dem System des KFZ-Mechanikers bekannt ist. In Abschnitt 3.4.1 wurde daher für mobile Geschäftsprozesse die Anforderung aufgestellt, dass Sub-Prozesse erst zur Laufzeit auf ein anderes System übertragbar sein sollen.

Die einleitenden, konzeptionellen Überlegungen in Abschnitt 4.3.1 führten zu der Schlussfolgerung, dass hierfür wenigstens zwei zu entwerfende BPEL-Aktivitäten notwendig sind. Eine Aktivität wird benötigt, um den Sub-Prozess im Kontext des Prozesses der Pannendienst-Zentrale zu definieren und diesen an den Mechaniker zu übertragen. Dazu muss dessen System eine festzulegende Schnittstelle in Form eines Web Service zur Verfügung stellen. Die an ihn von der Zentrale übertragene Nachricht enthält dabei die abstrakte BPEL-Prozessdefinition. Das bedeutet, dass für den Empfang des so übermittelten Sub-Prozesses eine zweite Aktivität als Spracherweiterung notwendig ist. Das beschriebene Vorgehen bezieht sich auf den mit 1. bezeichneten Pfeil in der schematischen Darstellung in Abbildung 4.9.

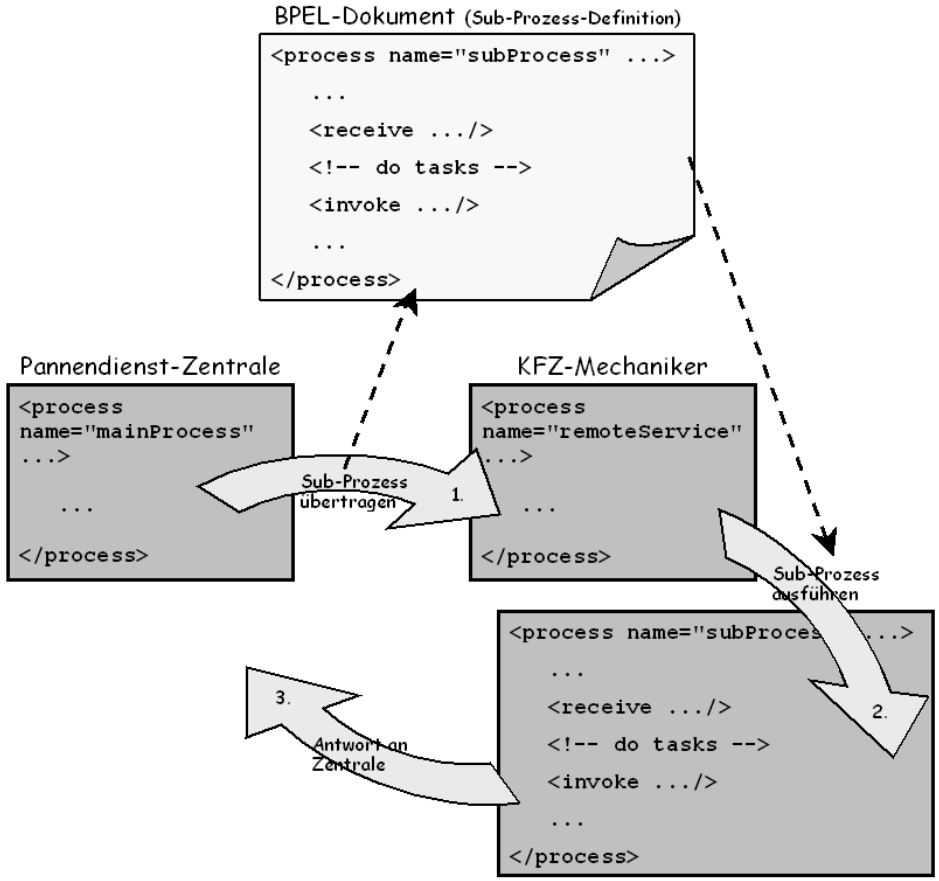

Abbildung 4.9: Übertragung und Ausführung eines mobilen Sub-Prozesses

Eine erste Idee bestand darin, im Rahmen der Empfangsaktivität den erhaltenen Sub-Prozess automatisch zu starten. Weitere Überlegungen sowie die in Abschnitt 3.4.1 beschriebene Anforderung, die Ausführung eines Sub-Prozesses ablehnen zu können, führten zu der Entscheidung, eine dritte Aktivität zu entwerfen. Mittels dieser soll der Start des empfangenen Sub-Prozesses explizit ausgeführt werden (siehe 2. Pfeil in Abbildung 4.9). So ist es möglich, zwischen dem Empfang und dem

Start weitere Aktivitäten auszuführen. Beispielsweise kann der Anwender interaktiv in die Entscheidung, den Sub-Prozess anzunehmen oder abzulehnen, eingebunden werden. Weiterhin ist es möglich, das Starten des Sub-Prozesses abhängig vom umgebenden Kontext zu machen, indem die Startaktivität in die in Abschnitt 4.3.3 vorgestellte BPEL-Erweiterung `<contextActivity>` integriert wird.

Die BPEL-Aktivität, mittels der eine Antwort an die Zentrale zurück gesendet wird, ist Bestandteil des Sub-Prozesses (siehe 3. Pfeil in Abbildung 4.9). Da die Pannendienst-Zentrale die Definition des Sub-Prozesses vornimmt, liegt es in ihrem Ermessen, ob eine Rückantwort gewünscht wird. Benötigt der übergeordnete Prozess diese Rückmeldung nicht, kann in der Prozessdefinition auf die entsprechende Aktivität verzichtet werden.

Die Anforderung, übertragbare Sub-Prozesse zu unterstützen, resultiert aus der Integration mobiler Mitarbeiter und somit mobiler Systeme in die Geschäftsprozesse. Obwohl die im Rahmen dieser Arbeit hierfür entwickelten Konzepte auf dem mobilen "Pannenhilfe"-Szenario basieren, sind übertragbare Sub-Prozesse nicht mobilitätsspezifisch. Die vorgestellte Lösung ist universell einsetzbar und kann auch in nicht mobilen Umgebungen zur Unterteilung eines Prozesses in Aufgabenpakete und deren Weitergabe zur Laufzeit an andere Ausführungseinheiten verwendet werden.

BPEL-Erweiterung `<sendProcess>`

Die erste Spracherweiterung, die für die Definition und das Senden des Sub-Prozesses eingesetzt wird, besteht aus der neuen Aktivität *<sendProcess>*. Ihre Syntax kann Listing 4.10 entnommen werden.

```
<extensionActivity>
   <sendProcess standard-attributes>
      standard-elements
      invoke-activity
      <mobileProcessData>
         <processBpel variable="BPELVariableName"?>
            bpel-process?
         </processBpel>
         <processWsdl variable="BPELVariableName"?>
            wsdl-definitions?
         </processWsdl>
         <processStartParameters
            partnerLink="NCName"
            portType="QName"?
            operation="NCName"
            inputVariable="BPELVariableName">
```

```
      </mobileProcessData>
   </sendProcess>
</extensionActivity>
```

Listing 4.10: Syntax der BPEL-Erweiterung `<sendProcess>`

Die wichtigsten Bestandteile von `<sendProcess>` sind die zwei Elemente `invoke-activity` und `<mobileProcessData>`. Letztgenanntes bildet alle Informationen ab, die in Verbindung mit der Sub-Prozessdefinition übertragen werden müssen. Das eingeschlossene Element `<processBpel>` nimmt hierfür die Prozessbeschreibung im BPEL-Format auf. Der definierte Prozess muss dabei in sich geschlossen und konform zur BPEL-Spezifikation sein. Dies schließt die Definition einer Startaktivität durch Nachrichteneingang ein, mittels der der Sub-Prozess anschließend auf Empfängerseite gestartet werden kann. Weiterhin gehört zu `<mobileProcessData>` das Kind-Element `<processWsdl>`. Dieses nimmt das WSDL-Dokument auf, das den Sub-Prozess beschreibt. Dies ist Voraussetzung dafür, dass der Empfänger den Sub-Prozess auf seinem System deployen kann. Alternativ zur Definition der BPEL- und WSDL-Dokumente innerhalb der Elemente `<processBpel>` und `<processWsdl>` kann in beiden Fällen auf zuvor festgelegte Variablen geeigneten Typs verwiesen werden. Darüber hinaus enthält `<mobileProcessData>` das Element `<processStartParameters>`. Mittels der vier Attribute `partnerLink`, `portType`, `operation` und `inputVariable` werden die Parameter festgelegt, die der Empfänger zum Starten des Sub-Prozesses benötigt. Dabei ist zu beachten, dass `inputVariable` auf eine zuvor definierte Variable verweist. Diese muss vom Typ der zum Sub-Prozess gehörigen WSDL-Message sein und die Start-Informationen für den Sub-Prozess beinhalten. Eine ausführlichere Syntaxbeschreibung der `<sendProcess>`-Aktivität kann dem XML-Schema in Listing A.1 in Anhang A entnommen werden. Daraus geht auch hervor, dass das Element `<mobileProcessData>` erweiterbar ist, um gegebenenfalls weitere Daten aufzunehmen, die zusammen mit der Sub-Prozessdefinition übermittelt werden sollen.

Beim zweiten Hauptbestandteil von `<sendProcess>` handelt es sich um den Platzhalter `invoke-activity`. Dieser bezieht sich auf den Web Service-Aufruf zur Übermittlung der im Element `<mobileProcessData>` definierten Daten an den Empfänger des Sub-Prozesses. Entweder wird der Platzhalter durch die standardisierte BPEL-Aktivität `<invoke>` ersetzt oder es kann die in Abschnitt 4.3.2 entworfene Spracherweiterung `<dynamicInvoke>` eingesetzt werden, die kontextabhängig den involvierten Partner-Service bestimmt. In beiden Fällen ist zu berücksichtigen, dass die WSDL-Servicebeschreibung das für die Sub-Prozess-Übertragung vorgegebene Datenformat in Form einer WSDL-Message implementiert.

Die Entscheidung, ein konkretes Datenformat vorzugeben, beruht auf der Überlegung, dass ein wesentlicher Teil der zu übertragenen Daten in jedem Fall gleich sein

wird, um auf der Gegenseite den Sub-Prozess ausführen zu können. Es wird somit sichergestellt, dass die im Rahmen der <sendProcess>-Aktivität auf Senderseite vorgenommenen Definitionen beim Empfänger von der korrespondierenden Aktivität <receiveProcess> korrekt interpretiert werden können. Zudem ist man mit dieser Lösung in der Lage, dem Prozess-Entwickler Arbeit abzunehmen, da auf das manuelle Konvertieren und Kopieren der zu übertragenen Werte in die zu übermittelnde Nachricht verzichtet werden kann. Die Angabe als <mobileProcessData> genügt, damit im Rahmen der <sendProcess>-Aktivität die Werte automatisch in die inputVariable der invoke-activity übernommen werden.

Die BPEL-Spracherweiterung umfasst daher nicht nur Erweiterungen im Rahmen der <extensionActivity> sondern bezieht sich auch auf Bestandteile der WSDL [WSD-2001]. (Dieses Vorgehen wird schon von der BPEL-Spezifikation bei der WSDL-Erweiterung <partnerLinkType> genutzt [BPE-2007].) Listing 4.11 enthält die Syntax für die zu verwendende WSDL-Message. Es handelt sich hierbei um die Abbildung des BPEL-Elements <mobileProcessData> (vgl. Listing 4.10) in den WSDL-Message-Type <mobileProcessData>. Das zugehörige XML-Schema enthält Listing A.3 in Anhang A. Im Unterschied zur BPEL-Repräsentation werden hier sämtliche Informationen als eingeschlossene Elemente modelliert, wobei <inputVariable> vom Typ tMessage aus dem WSDL 1.1 beschreibenden XML-Schema ist.

```
<mobileProcessData>
  <processBpel>
      bpel-process
   </processBpel>
   <processWsdl>
      wsdl-definitions
   </processWsdl>
   <processStartParameters>
      <partnerLink>"NCName"</partnerLink>
      <portType>"NCName"</portType>?
      <operation>"NCName"</operation>
      <inputVariable name="NCName">
         <part name="NCName" element="QName"? type="QName"?/>*
      </inputVariable>
   </processStartParameters>
</mobileProcessData>
```

Listing 4.11: Syntax der BPEL-Erweiterung <mobileProcessData> für WSDL-Einsatz

BPEL-Erweiterung `<receiveProcess>`

Das Gegenstück zur `<sendProcess>`-Aktivität ist auf der Empfängerseite die neue Aktivität *`<receiveProcess>`*. Listing 4.12 beschreibt die zugrundeliegende Syntax. Das zugehörige XML-Schema kann Listing A.1 in Anhang A entnommen werden.

```
<extensionActivity>
   <receiveProcess
    partnerLink="NCName"
    portType="QName"?
    operation="NCName"
    variable="BPELVariableName"
    createInstance="yes|no"?
    messageExchange="NCName"?
    standard-attributes>
    standard-elements
       <correlations>?
          <correlation set="NCName" initiate="yes|join|no"? />+
       </correlations>
       <fromParts>?
          <fromPart part="NCName" toVariable="BPELVariableName"
             />+
       </fromParts>
   </receiveProcess>
</extensionActivity>
```

Listing 4.12: Syntax der BPEL-Erweiterung `<receiveProcess>`

Die `<receiveProcess>`-Aktivität stellt quasi eine Kopie der zum BPEL-Standard gehörenden `<receive>`-Aktivität dar, die zum Empfangen einer eingehenden Nachricht verwendet wird (weitergehende Informationen siehe Spezifikation [BPE-2007]). Der einzige Unterschied zum Standard-`<receive>` besteht darin, dass das Attribut `variable` für `<receiveProcess>` nicht mehr optional ist, sondern immer belegt werden muss.

Mittels `<receiveProcess>` werden die Sub-Prozessdefinition und sämtliche damit verbundene Daten empfangen. Dabei muss sich das Attribut `variable` auf eine Variable vom Typ der zuvor beschriebenen WSDL-Message beziehen (in Anlehnung an den Vorgang bei `<sendProcess>`). Diese Variable nimmt die als `<mobileProcessData>` übertragenen Daten für den Sub-Prozess entgegen, so dass im weiteren Prozessverlauf darauf zurückgegriffen werden kann.

BPEL-Erweiterung `<startProcess>`

Die dritte Spracherweiterung in Verbindung mit übertragbaren Sub-Prozessen stellt die Aktivität *<startProcess>* dar. Mittels dieser wird der Sub-Prozess auf Empfängerseite gestartet. Die vereinfachte Syntaxbeschreibung kann Listing 4.13 entnommen werden; das zugehörige XML-Schema enthält Listing A.1 in Anhang A.

```
<extensionActivity>
   <startProcess variable="BPELVariableName"
   standard-attributes>
      standard-elements
   </startProcess>
</extensionActivity>
```

Listing 4.13: Syntax der BPEL-Erweiterung `<startProcess>`

Die mittels `<receiveProcess>` empfangenen Daten wurden in einer entsprechenden Variable abgespeichert. Diese wird nun ebenfalls vom Attribut `variable` der `<startProcess>`-Aktivität referenziert. Das Element `<mobileProcessData>` enthält alle notwendigen Daten wie BPEL- und WSDL-Definitionen, damit die ausführende Engine den Prozess deployen kann. Im Anschluss daran und ebenfalls im Kontext der `<startProcess>`-Aktivität wird automatisch eine Prozessinstanz gestartet. Dazu muss der ausführende Prozess eine Nachricht mit den übermittelten Werten aus `<processStartParameters>` an den deployten Sub-Prozess verschicken, wodurch dieser mit der Ausführung beginnt.

4.4 Konzeption und Architektur der Engine

Im vorhergehenden Abschnitt 4.3 wurde die Konzeption der Sprache erarbeitet und die Konstrukte der BPEL-Spracherweiterung zur Unterstützung mobiler Workflows vorgestellt. Die Aktivitäten `<dynamicInvoke>`, `<contextActivity>`, `<sendProcess>`, `<receiveProcess>` und `<startProcess>` stellen dafür die formalen Beschreibungsmöglichkeiten zur Verfügung.

Um diese Aktivitäten eingebettet in eine komplette BPEL-Prozessdefinition in Form von Workflows ausführen zu können, wird eine Engine benötigt, die diese Erweiterungen interpretieren und verarbeiten kann. Für diese Anforderung dient die ActiveBPEL Engine von Active Endpoints im Rahmen dieser Arbeit als Basis (vgl. Abschnitt 3.2). Die als Open Source verfügbare Java-Implementierung soll um die oben genannten Aktivitäten erweitert werden. Dieser Abschnitt beschäftigt sich mit der Konzeption der Engine-Erweiterungen.

4.4.1 Grundlagen zur Erweiterung der ActiveBPEL Engine

Die ActiveBPEL Engine unterstützt in der verwendeten Version 4.0 sowohl die Spezifikation BPEL4WS 1.1 als auch den Standard WS-BPEL 2.0. Da die entworfenen Spracherweiterungen sich auf WS-BPEL 2.0 beziehen, wird bei der Konzeption der Engine-Erweiterungen nur auf die hierfür relevanten Komponenten eingegangen.

Damit eine sinnvolle Integration der neuen Funktionalitäten in die ActiveBPEL Engine erfolgen kann, muss beim Erstellen der Architektur die vorliegende Engine-Architektur berücksichtigt werden. Hierzu diente die unter [Act-2007c] und [Act-2007d] veröffentlichte Dokumentation wie auch die Angaben im Quelltext der Engine, die über Javadoc zur Verfügung stehen. Darüber hinausgehende Zusammenhänge bezüglich des konzeptionellen Aufbaus der ActiveBPEL Engine wurden durch Reverse Engineering erarbeitet, da die verfügbaren Dokumente hinsichtlich ihres Informationsgehalts nicht ausreichend waren. Ein Großteil der im Folgenden vorgestellten Konzepte beruht demnach auf den so erzielten Erkenntnissen.

Im Rahmen der Analyse wurde die engine-spezifische Realisierung vom Deployment der Prozessdefinitionen bis zur Ausführung einzelner Prozessinstanzen einleitend beschrieben (vgl. Abschnitt 3.2.2). Die folgenden Konzepte setzen auf diesem Wissensstand auf, der die Abbildung der BPEL-Sprachelemente in Definitions-Objekte und deren Überführung in instanzbezogene Implementierungs-Objekte beinhaltet.

Da der BPEL-Standard die Erweiterung der Sprache um neue Aktivitäten durch Verwendung des `<extensionActivity>`-Konstrukts vorsieht (vgl. Abschnitt 3.1.1),

wäre eine definierte, in sich geschlossene Schnittstelle seitens der ActiveBPEL Engine wünschenswert. Weder die zuvor genannte Dokumentation noch die eigenen Untersuchungen deuten auf eine solche Unterstützung hin. Dies führt dazu, dass die Integration der neuen Aktivitäten direkt in die bestehenden Komponenten vorgenommen werden muss, beispielsweise durch Unterklassen-Bildung oder ergänzende Anweisungen beim Visitor-Pattern. Für die anschließende Realisierung bedeutet dies, dass die Erweiterungen direkt im Quellcode der ActiveBPEL Engine ansetzen müssen und der originale Engine-Code nicht unverändert als Basis dienen kann.

Eine Besonderheit dieser Arbeit besteht daher darin, dass es sich bei der Konzeption nicht um einen unabhängigen Entwurf handelt, bei dem es nur die selbst aufgestellten Randbedingungen zu berücksichtigen gilt. Vielmehr sollen im weiteren Verlauf die bestehende Architektur und damit verbundene Vorgaben der ActiveBPEL Engine aufgegriffen und der eigene Entwurf darin integriert werden. Es wird versucht, Begrifflichkeiten, Analogien und Vorgehensmuster seitens der ActiveBPEL Engine bei den eigenen Engine-Erweiterungen fortzuführen. Dies betrifft speziell Elemente, die von bestehenden Klassen abgeleitet werden oder entsprechende Interfaces implementieren. Je weiter sich der Entwurf von der vorgegebenen Engine-Architektur entfernt, was besonders die von den Implementierungs-Objekten ausgehende Umsetzung der Aktivitäts-Logik betrifft, werden verstärkt eigene Konzepte realisiert, die sich nach Möglichkeit dennoch in das Gesamtbild eingliedern.

Um deutlich zu machen, welche Komponenten zur Engine-Erweiterung gehören, beginnen deren Namen grundsätzlich mit einem "M" (für Mobilität). Die einzige Ausnahme bilden neu geschaffene Interfaces; sie beginnen mit "IM". Das Kennzeichen der von Active Endpoints entwickelten Komponenten ist das "Ae". Für ein besseres Verständnis wurde bei den verwendeten UML-Klassendiagrammen zum Teil auf Details bei der Abbildung verzichtet. Sie heben primär die gerade untersuchten Sachverhalte hervor und geben somit einen Ausschnitt der Gesamtarchitektur wieder. Aus demselben Grund werden auch getter- und setter-Methoden nicht explizit aufgeführt, sondern lediglich die Attribute und Referenzen der Klassen abgebildet.

Als Einstieg in die Engine-Architektur soll der nachfolgende Teil dieses Grundlagen-Abschnitts dienen. Dabei wird auf die Modellierung der Definitions- und Implementierungs-Objekte eingegangen, und diese am Beispiel der zum BPEL-Sprachumfang gehörenden `<extensionActivity>` vorgestellt sowie Ansatzmöglichkeiten für die eigenen Erweiterungen aufgezeigt. Die folgenden Abschnitte behandeln den konzeptionellen Engine-Entwurf für die in Abschnitt 4.3 definierten Spracherweiterungen. Dabei wird für jede Erweiterung zunächst mit der Modellierung der Definitions-Objekte begonnen. Daran schließt sich der Entwurf der Implementierungs-Objekte und die Umsetzung der Aktivitäts-Logik an.

Grundlagen zur Modellierung der Definitions-Objekte

Abbildung 4.10 zeigt das Klassendiagramm für die Definitions-Objekte, die den BPEL-Erweiterungsmechanismus umsetzen. Hierzu gehören die Sprach-Konstrukte `<extensions>`, `<extension>` und `<extensionActivity>` (vgl. Abschnitt 3.1.1).

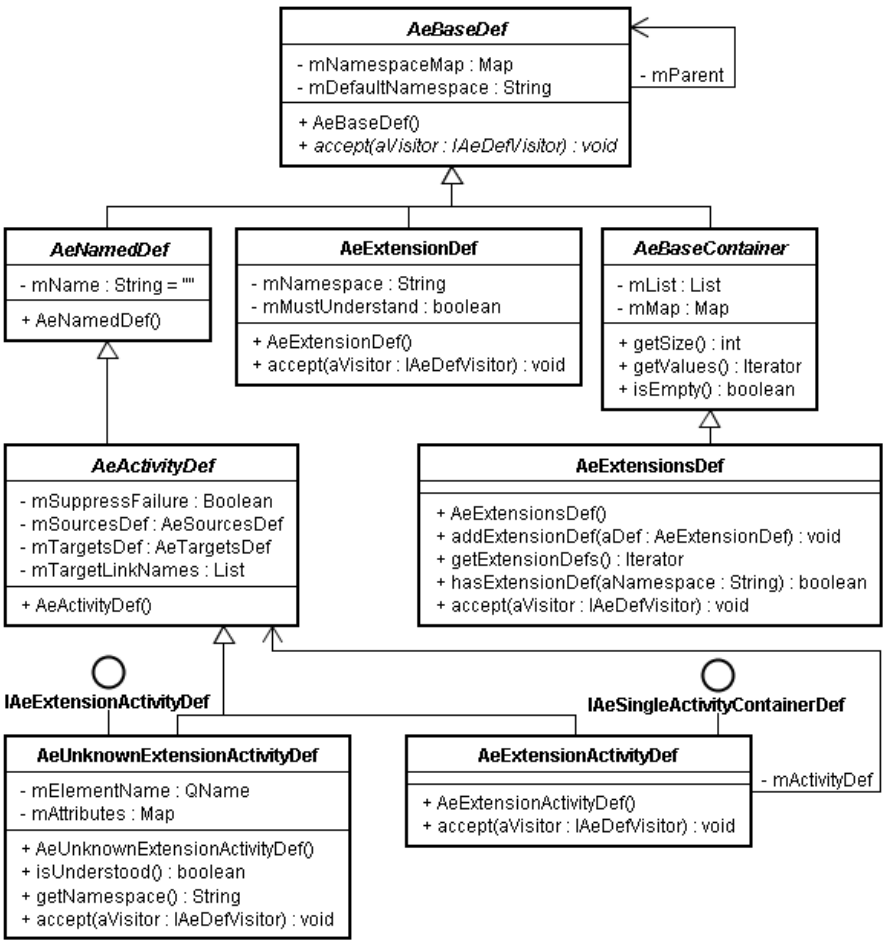

Abbildung 4.10: Klassendiagramm `AeExtensionActivityDef`

Die abstrakte Klasse `AeBaseDef` bildet die Oberklasse für alle Definitions-Objekte, deren Namen zur Kennzeichnung jeweils mit ”Def” enden. Darüber hinaus geben die Bezeichnungen der Attribute und Operationen zumeist Aufschluss darüber, welchen Aspekt der BPEL-Spezifikation sie abbilden (siehe dazu [BPE-2007]). Jedes im BPEL-Standard spezifizierte Sprachelement bildet die ActiveBPEL Engine auf ein eigenes Definitions-Objekt ab. Die Klasse `AeBaseDef` stellt über die `mParent`-Verbindung sicher, dass jede davon abgeleitete Unterklasse ihr übergeord-

netes Definitions-Objekt kennt. Für die Implementierung der Gegenrichtung ist jede Definitions-Klasse – sofern notwendig – selbst verantwortlich. Weiterhin verlangt `AeBaseDef`, dass jede ihrer Unterklassen die Methode `accept()` implementiert. Sie gehört zur Umsetzung des Visitor-Pattern, mit dem in diesem Fall durch Traversierung der Definitions-Objekte die zugehörigen Implementierungs-Objekte erzeugt werden.

Die Klasse `AeExtensionDef` stellt die Abbildung des `<extension>`-Elements dar und enthält die Attribute `mNamespace` und `mMustUnderstand`. Für das Element `<extensions>` ist die Klasse `AeExtensionsDef` als Unterklasse von `AeBaseContainer` zuständig. Da `AeExtensionsDef` mehrere Objekte vom Typ `AeExtensionDef` enthalten kann, dient die Oberklasse `AeBaseContainer` in solchen Fällen als Verwalter für eine Menge an referenzierten Definitions-Objekten.

Der BPEL-Standard sieht vor, dass allen wesentlichen Elementen ein Name zugewiesen werden kann. Dieses Verhalten bildet die abstrakte Klasse `AeNamedDef` über das Attribut `mName` ab. Abbildung 4.10 zeigt exemplarisch deren Unterklasse `AeActivityDef`. Von ihr erben wiederum alle in BPEL definierten Aktivitäten. Als Beispiele wurden hier die Klassen `AeExtensionActivityDef` und `AeUnknownExtensionActivityDef` gewählt. `AeExtensionActivityDef` modelliert das BPEL-Element `<extensionActivity>` und implementiert zu diesem Zweck die Schnittstelle `IAeSingleActivityContainer`. Somit enthält `AeExtensionActivityDef` ein Objekt vom Typ `AeActivityDef`, da `<extensionActivity>` in einer Prozessbeschreibung genau eine BPEL-Aktivität enthalten muss.

Als exemplarische Spracherweiterung modelliert die ActiveBPEL Engine die Klasse `AeUnknownExtensionActivityDef` für alle Erweiterungen, die sie nicht zu interpretieren weiß. Die Schnittstelle `IAeExtensionActivityDef` dient als Markierung und sollte von allen Erweiterungsaktivitäten implementiert werden. Weiterhin sollte jede Erweiterungsaktivität die beiden Methoden `isUnderstood()` und `getNamespace()` realisieren. Erstere wird verwendet, um anzuzeigen, ob die Engine eine Implementierung für die konkrete Erweiterung unterstützt; die zweite Methode gibt den Namensraum der definierten `<extensionActivity>` wieder (siehe Abschnitt 3.1.1).

Grundlagen zur Modellierung der Implementierungs-Objekte

Im Prinzip stellen die Definitions-Objekte ein zur Laufzeit verfügbares Zugriffswerkzeug für die in BPEL verfassten Prozessdefinitionen dar. Die Implementierungs-Objekte setzen als Fortführung die Prozesslogik der einzelnen BPEL-Konstrukte um.

Um einen ersten Eindruck vom Aufbau der Implementierungs-Objekte zu bekommen, zeigt Abbildung 4.11 den Zusammenhang der Klassen `AeAbstractBpelObject` und `AeActivityImpl`. `AeAbstractBpelObject` stellt die Oberklasse aller Implementierungs-Objekte dar und erwartet von allen konkreten Unterklassen die Implementierung der `accept()`-Methode. Das zugrundeliegende Visitor-Pattern benötigt diese zur Verwaltung der Implementierungs-Objekte, beispielsweise für die Zustandsüberwachung der Aktivitäten. Weiterhin weist `AeAbstractBpelObject` eine Vielzahl von Hilfsfunktionen für die Implementierungs-Objekte auf. Zum jetzigen Zeitpunkt sollen diese nicht näher untersucht werden; die für das Verständnis notwendigen Konzepte werden im weiteren Verlauf näher erläutert.

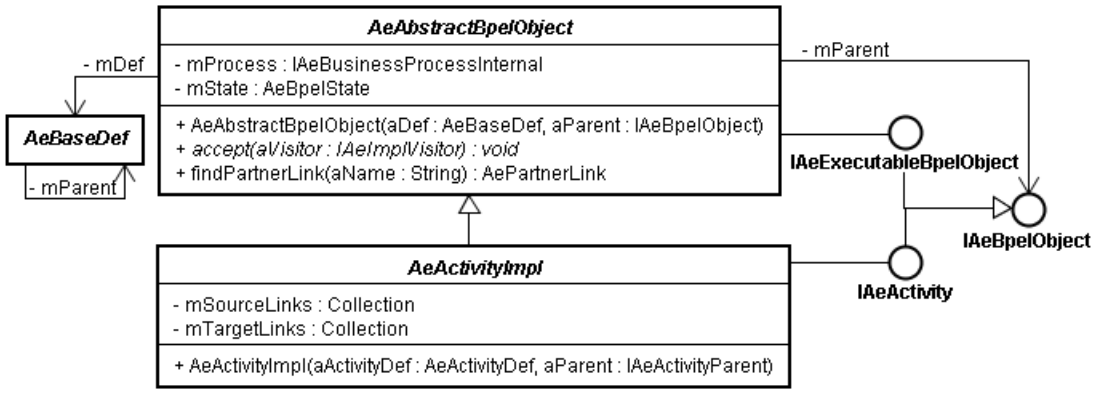

Abbildung 4.11: Klassendiagramm `AeActivityImpl`

Die Unterklasse `AeActivityImpl` ist das Implementierungs-Gegenstück zur `AeActivityDef`-Klasse. Sie realisiert die Basis-Funktionen aller BPEL-Aktivitäten und dient als Oberklasse für alle im Rahmen dieser Arbeit konzipierten Erweiterungsaktivitäten. Die Engine benötigt nicht für jedes Definitions-Objekt ein entsprechendes Implementierungs-Objekt. Lediglich das oberste Element jeder Aktivität muss als Implementierungs-Objekt modelliert werden. Sein Name endet zur Kennzeichnung auf "Impl".

Implementierungs-Objekten wird bei Ihrer Erzeugung eine Referenz auf ihr entsprechendes Definitions-Objekt übergeben, somit stehen bei der prozessinstanz-basierten Ausführung alle notwendigen Informationen zur Verfügung. Die Implementierungs-Objekte müssen daher nicht erneut alle Attribute des zugrundeliegenden BPEL-Elements enthalten, sondern können über ihr Definitions-Objekt auf diese Informationen zurückgreifen.

4.4.2 Dynamische, kontextabhängige Service-Auswahl

Bei der in Abschnitt 4.3.2 eingeführten Aktivität `<dynamicInvoke>` handelt es sich um eine Erweiterung der im BPEL-Standard spezifizierten Aktivität `<invoke>`. Diese ist in der Lage, zur Laufzeit kontextabhängig einen geeigneten Service-Provider zu ermitteln und diesen dynamisch an den Operationsaufruf zu binden, mit dem eine Nachricht an einen Partner-Web Service übertragen wird.

Modellierung der Definitions-Objekte für `<dynamicInvoke>`

Ebenso wie es sich bei der Syntax von `<dynamicInvoke>` um eine Erweiterung von `<invoke>` handelt (vgl. Listing 4.1), kann auch das Definitions-Objekt `MActivityDynamicInvokeDef` als Unterklasse der bestehenden `AeActivityInvokeDef`-Klasse modelliert werden. Abbildung 4.12 gibt das entsprechende Klassendiagramm wieder.

Dabei hält die Klasse `AeActivityPartnerLinkBaseDef` alle Informationen für BPEL-Aktivitäten vor, die sich auf einen Partner Link beziehen. Neben `AeActivityInvokeDef` erben auch `AeActivityReplyDef` und `AeActivityReceiveDef` von dieser Klasse. Um kenntlich zu machen, dass es sich bei `MActivityDynamicInvokeDef` um eine BPEL-Erweiterung handelt, implementiert diese Klasse die Schnittstelle `IAeExtensionActivityDef`. Die Schnittstellen `IMContextParentDef` und `IMSelectParentDef` entstanden in Anlehnung an die Konventionen in der vorliegenden ActiveBPEL Engine-Implementierung. Sie zeigen an, dass `<dynamicInvoke>` intern eine `<select>`-Definition und diese eine `context`-Definition enthält.

Die von `MActivityDynamicInvokeDef` referenzierte Klasse `MSelectDef` bildet das `<select>`-Element ab. Da es den Kontext sowohl als eingeschlossenes Element (vgl. Listing 4.5) oder über eine Variable (vgl. Listing 4.6) beziehen kann, weist auch die Klasse `MSelectDef` für beide Varianten Zugriffsmöglichkeiten auf: entweder direkt über das Attribut `mVariable` oder über die in der Oberklasse `AeBaseContainer` umgesetzte Strategie.

Modellierung der Kontext-Definitions-Objekte

Da es sich bei den Sprachelementen für die Kontextmodellierung nicht um Aktivitäten handelt, werden diese als direkte Unterklasse von `AeBaseDef` modelliert. Zunächst dient die abstrakte Klasse `MContextDef` als Oberklasse für beliebige Kontextdefinitionen, wie Abbildung 4.13 zeigt. Im Rahmen des Szenarios

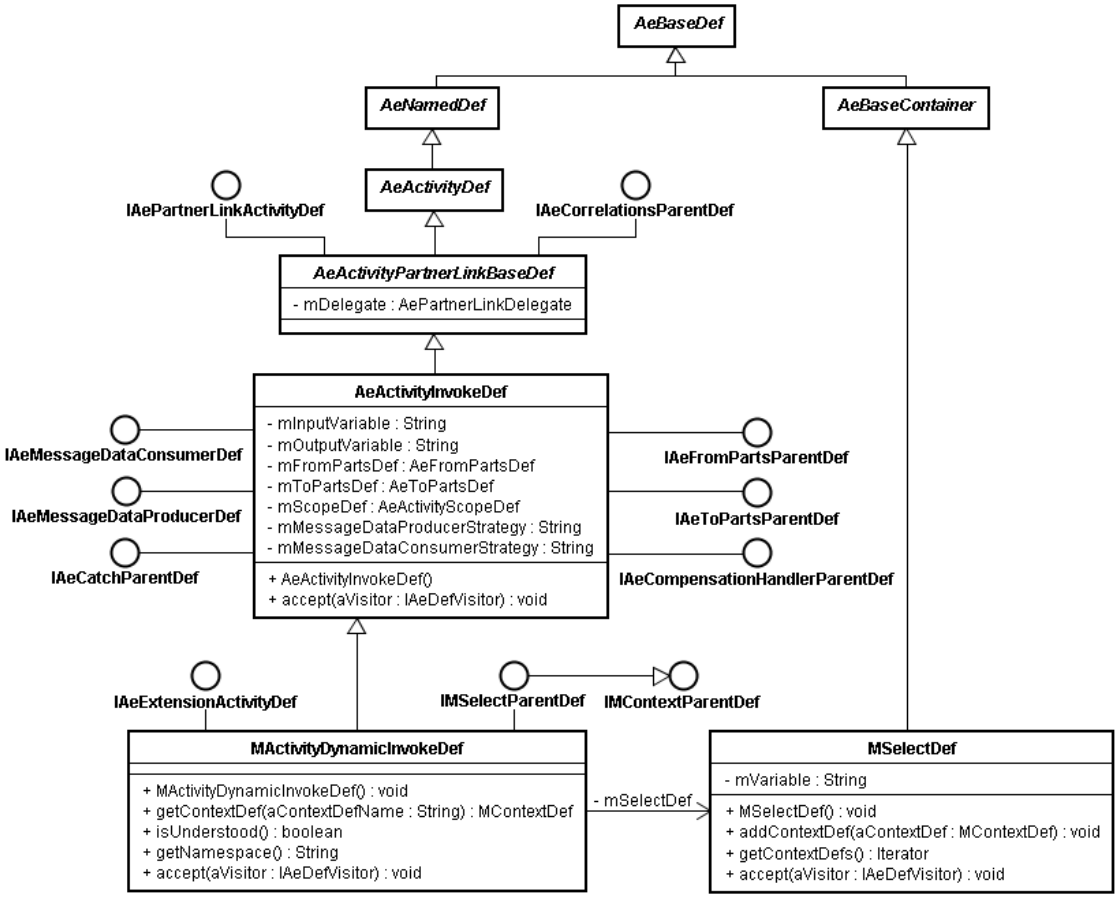

Abbildung 4.12: Klassendiagramm `MActivityDynamicInvokeDef`

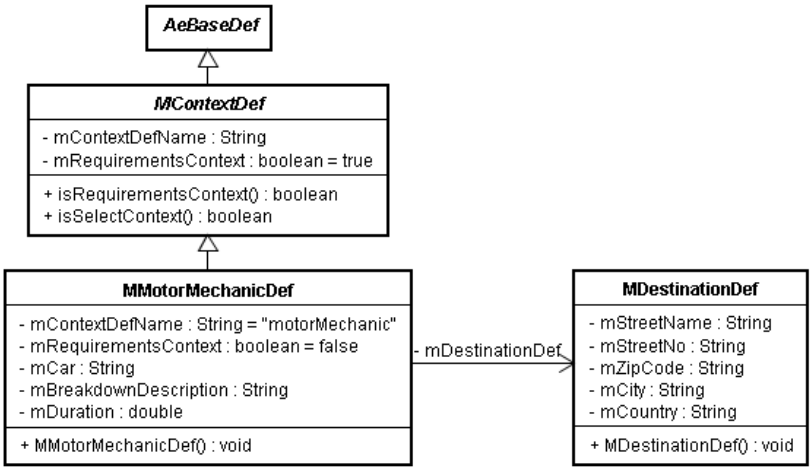

Abbildung 4.13: Klassendiagramm `MContextDef`

”Auto-Pannendienst” existiert für die <dynamicInvoke>-Aktivität lediglich die konkrete Implementierung MMotorMechanicDef. In Abschnitt 4.4.3, der die <contextActivity>-Aktivität behandelt, wird MContextDef als Oberklasse für weitere Kontextdefinitionen verwendet.

Dazu weist MContextDef die beiden Attribute mContextDefName und mRequirementContext auf. Letzteres dient der Unterscheidung, ob die Kontextinformation in einem <requirements>-Element von <contextActivity> oder in einem <select>-Element von <dynamicInvoke> definiert wurde. Das Attribut mContextDefName nimmt in jedem Fall den XML-Element-Namen der BPEL-Erweiterung auf, auf die sich das jeweilige Kontext-Definitions-Objekt bezieht. Im Fall von <motorMechanic> lautet der mContextDefName für die Definitions-Klasse ”motorMechanic”. Dieses Verhalten kann später in den Implementierungs-Objekten dazu genutzt werden, die angegebenen Kontexteigenschaften mit den aktuellen Kontextausprägungen zu vergleichen. Auf diesen Sachverhalt wird an späterer Stelle genauer eingegangen.

Bei MMotorMechanicDef und der ihrerseits referenzierten Klasse MDestinationDef handelt es sich um direkte Abbildungen der Syntax-Vorgaben (vgl. XML-Schema in Listing A.2). Wie das Klassendiagramm in Abbildung 4.13 zeigt, verfügen beide Klassen über entsprechende Attribute und Operationen zum Setzen und Auslesen der kontextuellen Eigenschaften.

Modellierung der Implementierungs-Objekte für <dynamicInvoke>

Wie schon bei der Aufstellung der Syntax für <dynamicInvoke> und der Modellierung des Definitions-Objekts MActivityDynamicInvokeDef geschehen, kann auch das Implementierungs-Objekt vom BPEL-Standard-Invoke abgeleitet werden. Die Klasse MActivityDynamicInvokeImpl wird also als Unterklasse von AeActivityInvokeImpl realisiert, wie das Diagramm in Abbildung 4.14 zeigt. Deren abstrakte Oberklasse AeWSIOActivityImpl dient als Basis für sämtliche Implementierungs-Objekte, die eine ein- oder ausgehende Kommunikation mit Partner Web Services umsetzen.

Neben MActivityDynamicInvokeImpl wurde für die Umsetzung der <dynamicInvoke>-Aktivität die Klasse MSelectImpl modelliert. Sie enthält als Liste alle zu berücksichtigenden MContextDef-Objekte. Das Zusammenspiel dieser beiden Implementierungs-Objekte und wie dadurch die Aktivitäts-Logik umgesetzt wird, zeigen die folgenden Entwurfsergebnisse auf.

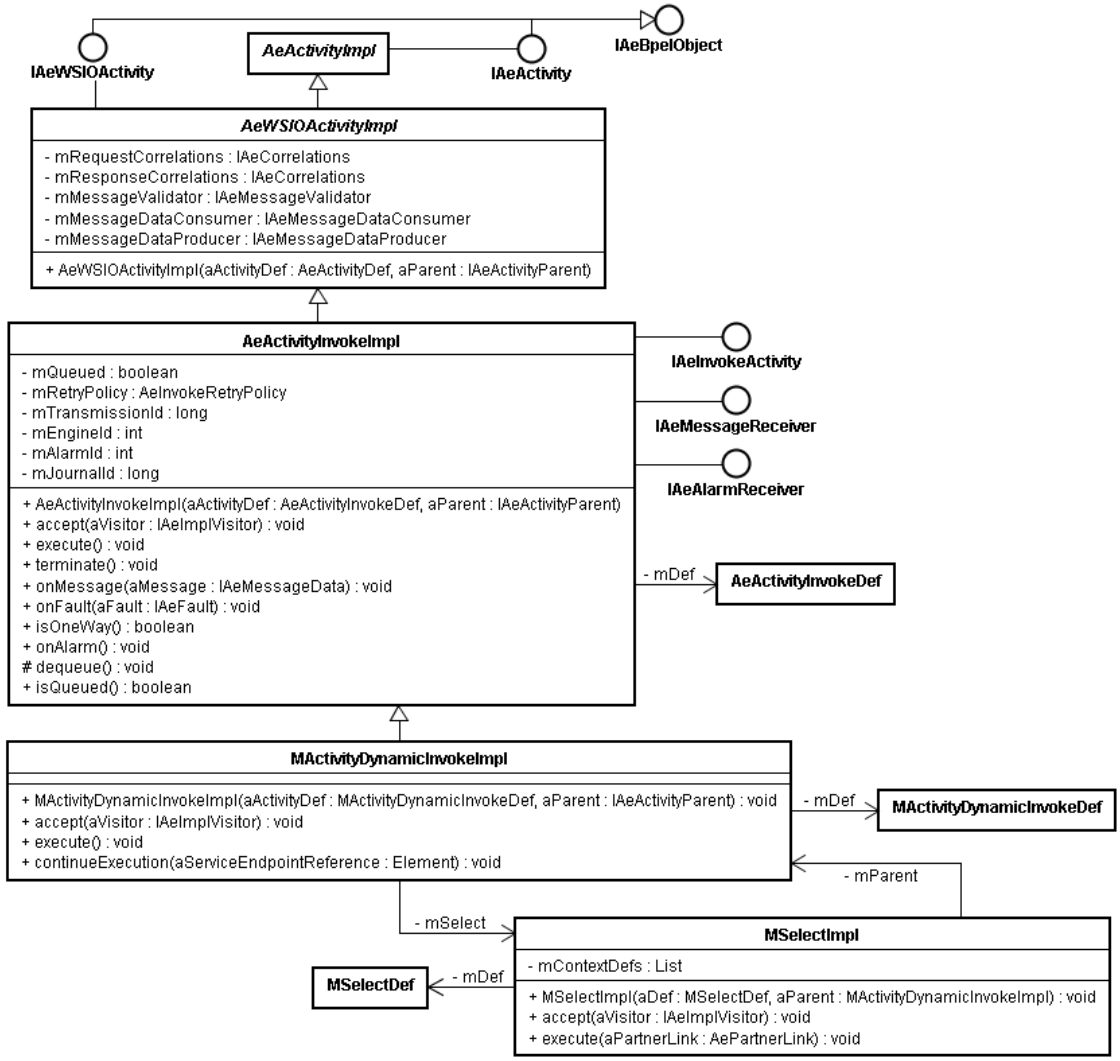

Abbildung 4.14: Klassendiagramm `MActivityDynamicInvokeImpl`

Umsetzung der Aktivitäts-Logik für `<dynamicInvoke>`

Die Umsetzung der Aktivitäts-Logik besteht aus zwei Anforderungen: Zuerst muss anhand der Kontextvorgabe ein geeigneter Partner Web Service ermittelt werden. Im Anschluss muss die entsprechende Binding-Information gesetzt werden, mit der der Web Service-Aufruf letztlich ausgeführt werden soll. Um erklären zu können, welches Vorgehen für das Binding zur Laufzeit gewählt wurde, soll zunächst der Ablauf bei Wahl einer statischen Binding-Strategie vorgestellt werden. Dabei handelt es sich bei den Binding-Methoden in jedem Fall um engine-spezifische Realisierungen. Der BPEL-Standard macht diesbezüglich keine Vorgaben.

Die ActiveBPEL Engine erwartet, dass der im bpr-Archiv enthaltene Process Deployment Descriptor die gewählte Vorgehensweise zur Bestimmung der konkreten Partner Link-Implementierungen beschreibt. Zur Designzeit des BPEL-Prozesses liegen diese nur in abstrakter Beschreibung vor. Bei statischem Binding wird diese Information in Form einer *Web Service Endpoint Reference* angegeben. Standardmäßig verwendet die ActiveBPEL Engine hierfür *WS-Addressing* [WS–2007]. Listing 4.14 enthält exemplarisch eine statische Binding-Definition, wie sie Inhalt einer .pdd-Datei wäre.

```
<partnerLink name="MotorMechanicRequestingPL">
  <partnerRole endpointReference="static">
    <wsa:EndpointReference xmlns:wsa="http://schemas.xmlsoap.org/
        ws/2003/03/addressing"
       xmlns:s="http://informatik.haw-hamburg.de/master/wsdl/
       MotorMechanicService">
      <wsa:Address>http://localhost:8080/active-bpel/services/
          MotorMechanicService</wsa:Address>
      <wsa:ServiceName PortName="MotorMechanicServiceSOAPPort">
          s:MotorMechanicService</wsa:ServiceName>
    </wsa:EndpointReference>
  </partnerRole>
</partnerLink>
```

Listing 4.14: Statische Binding-Angabe mittels Endpoint Reference

Diese Binding-Angaben liest die ActiveBPEL Engine während des Deployment-Vorgangs ein und setzt im Fall der statischen Endpoint Reference die entsprechenden Binding-Informationen. Soll das Service-Binding erst zur Laufzeit ausgeführt werden, wird zunächst auf Dummy-Werte mit identischer Methoden-Signatur, Port Type und Binding-Methode zurückgegriffen. Entsprechend Listing 4.15 muss die Binding-Angabe für eine dynamische Ermittlung der Endpoint Reference wie im Fall der `<dynamicInvoke>`-Aktivität gestaltet sein.

```
<partnerLink name="MotorMechanicRequestingPL">
  <partnerRole endpointReference="dynamic"/>
</partnerLink>
```

Listing 4.15: Dynamische Binding-Angabe mittels Endpoint Reference

Sobald die `<dynamicInvoke>`-Aktivität beginnen kann, startet die Engine das Implementierungs-Objekt `MActivityDynamicInvokeImpl` standardmäßig durch Aufruf der `execute()`-Methode. Die internen Abläufe, die dann ausgeführt werden, veranschaulicht das Sequenzdiagramm in Abbildung 4.15.

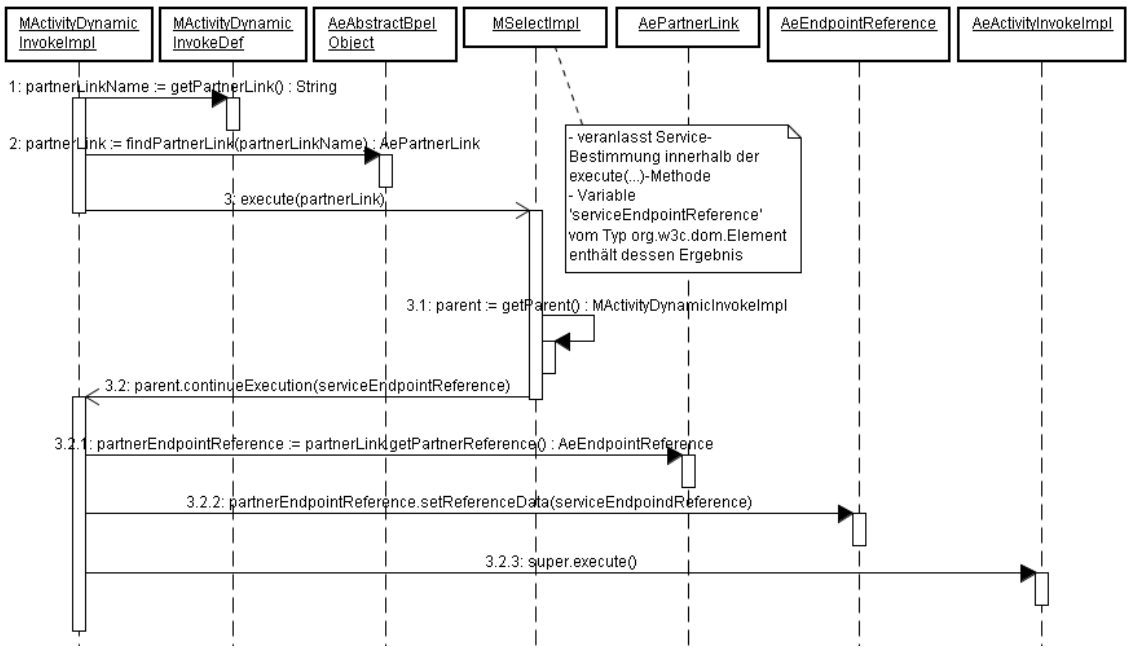

Abbildung 4.15: Sequenzdiagramm "Setzen der Binding-Information"

`MActivityDynamicInvokeImpl` greift auf ihr Definitions-Objekt zu, um sich den Partner Link-Namen für den Web Service-Aufruf zu holen. Dieser wird verwendet, um sich über die von einer Oberklasse implementierten `findPartnerLink()`-Methode das entsprechende `AePartnerLink`-Objekt geben zu lassen. Für den Fall, dass die Binding-Informationen schon während des Deployment-Vorgangs zur Verfügung standen, enthält dieses Objekt alle notwendigen Informationen über die Web Service Endpoint Reference. Im vorliegenden Fall werden diese Informationen jedoch erst zur Laufzeit dem `AePartnerLink` zugewiesen.

Diese Aufgabe leitet `MActivityDynamicInvokeImpl` an `MSelectImpl` durch Aufruf der `execute()`-Methode weiter und übergibt hierzu das `AePartnerLink`-Objekt. Wie die Servicebestimmung durchgeführt wird,

wird im Anschluss erläutert. Zunächst wird davon ausgegangen, dass `MSelectImpl` die ermittelte Web Service Endpoint Reference in Form einer DOM-Element-Struktur an `MActivityDynamicInvokeImpl` durch Übergabe an die `continueExecution()`-Methode zurückliefert (vgl. Element `<EndpointReference>` in Listing 4.14). Dadurch setzt `MActivityDynamicInvokeImpl` mit ihrer Ausführung fort und führt das Setzen der Binding-Information durch.

Der hierfür notwendige Ablauf wurde in Analogie zum in der ActiveBPEL Engine umgesetzten Verfahren entworfen, mit dem sich zur Laufzeit Endpoint References mittels `<assign>` und `<copy>` manipulieren lassen (vgl. [Car-2007]). `MActivityDynamicInvokeImpl` lässt sich vom zuvor ermittelten `AePartnerLink`-Objekt die zugehörige `AeEndpointReference` geben, die in der Engine die Web Service Endpoint Reference modelliert. An dieser müssen die zuvor dynamisch bestimmten Binding-Informationen gesetzt werden; dies geschieht über die Methode `setReferenceData()`, die als Parameter das DOM-Element erwartet. Damit ist das Setzen der Binding-Daten abgeschlossen und die Logik der im BPEL-Standard definierten `<invoke>`-Aktivität kann unverändert ausgeführt werden. Dazu wird an die `execute()`-Methode der Oberklasse `AeActivityInvokeImpl` verwiesen. Erst hiermit findet der Aufruf des konkreten Partner Web Service statt.

Bei der bisher betrachteten Umsetzung der Aktivitäts-Logik wurde die eigentliche Service-Bestimmung ausgelassen. Dies soll nun nachgeholt werden.

Um im Rahmen der `<dynamicInvoke>`-Aktivität abhängig vom definierten Kontext die Binding-Informationen zu ermitteln, wird als Basis eine *Service-Registry* eingeführt, die über eine Menge an verfügbaren Services verfügt. Jeder Service-Eintrag muss dabei seiner WSDL entsprechend Angaben zum Binding und den Port Type enthalten. Über diesen können alle Service-Einträge, die die gewünschte Schnittstelle implementieren, ermittelt werden. Die Service-Registry soll in der Lage sein, Services mit unterschiedlichen Schnittstellen zu verwalten, und dabei dient der Port Type als Zugriffsschlüssel.

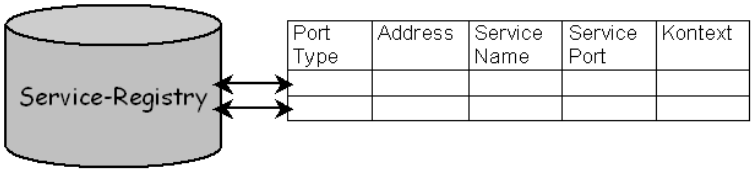

Abbildung 4.16: Service-Registry für `<dynamicInvoke>`

Über den bereits ermittelten Partner Link lässt sich der zugehörige Port Type auslesen. Für diesen muss aus der Service-Registry die Binding-Information für einen konkreten Web Service bezogen werden. Da die Registry in den meisten Fällen mehr als einen Service-Eintrag für den gewünschten Port Type enthalten wird, muss aus dieser Menge an möglichen Services der mit einem passenden Kontext herausgefiltert werden. Das bedeutet, dass jeder Service-Eintrag zusätzlich Angaben zu seinem aktuellen Kontext enthalten muss.

In welcher Form die der Service-Registry zugrundeliegenden Daten abgespeichert werden, ist für den weiteren Entwurf im Rahmen dieser Arbeit unerheblich. Als Datenquelle kann beispielsweise eine XML-Datei, eine Datenbank, aber auch UDDI verwendet werden. Weiterhin wird davon ausgegangen, dass die Service-Registry bereits entsprechende Einträge enthält. Wie die einzelnen Web Services ihre Daten publizieren oder aktualisieren – was besonders für die Kontextangabe regelmäßig zu erwarten ist –, wird ebenfalls in dieser Arbeit nicht weiter untersucht.

Der Entwurf sieht für die Service-Registry die Singleton-Klasse `MServiceRegistry` vor, die den Zugriff auf die Service-Einträge kapselt und konkrete Implementierungen der abstrakten Oberklasse `MService` zurückliefert. Abbildung 4.17 zeigt das zugehörige Klassendiagramm.

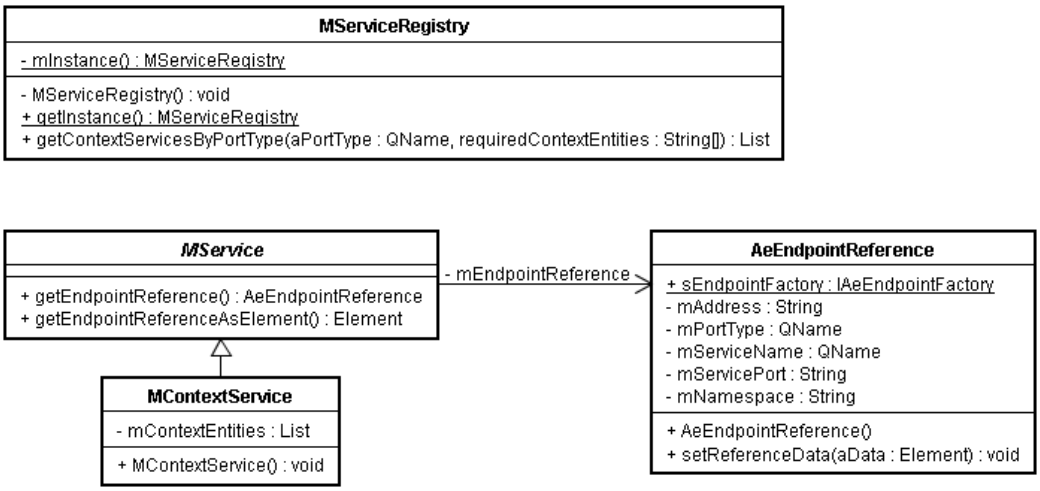

Abbildung 4.17: Klassendiagramm `MServiceRegistry` und `MService`

Dabei wurde der Entwurf so ausgelegt, dass prinzipiell auch andere `MService`-Implementierungen von der `MServiceRegistry` verwaltet werden können. Jeder `MService` enthält eine Referenz auf die schon existierende, engine-interne Endpoint-Repräsentation `AeEndpointReference`, die die jeweiligen Binding-Informationen für den Service abbildet. Für das Szenario "Auto-Pannendienst"

existiert die konkrete Klasse `MContextService`. Sie enthält eine Liste mit Kontextinformationen, die den Service beschreiben. Diese Listen-Einträge sind vom Typ `MSelectContextEntity`, der in Abbildung 4.18 dargestellt ist.

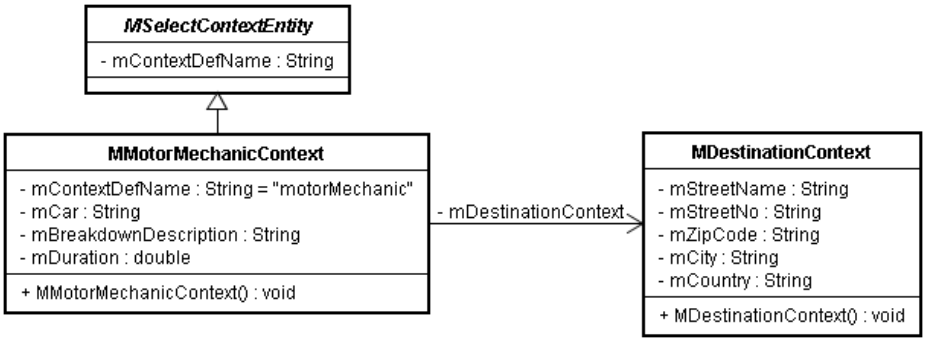

Abbildung 4.18: Klassendiagramm `MSelectContextEntity`

Im Prinzip handelt es sich bei allen drei in Abbildung 4.18 enthaltenen Klassen – also `MSelectContextEntity`, `MMotorMechanicContext` und `MDestinationContext` – um Entsprechungen der Kontext-Definitions-Objekte (vgl. Abbildung 4.13), wobei diese drei nicht den vorgegebenen Kontext aus der Prozessdefinition beschreiben, sondern den jeweiligen Kontext der einzelnen Web Services modellieren. Wie schon die Kontext-Definitions-Objekte enthalten auch die direkten Unterklassen von `MSelectContextEntity` das Attribut `mContextDefName`, das den Namen des entsprechenden BPEL-XML-Elements enthält. Im Fall von `MMotorMechanicContext` lautet dies `"motorMechanic"`. Diese Angabe dient als eindeutiger Schlüssel für den Vergleich von `MContextDef`- mit `MSelectContextEntity`-Objekten. Die `MServiceRegistry` führt diesen Vergleich aus, um nur die `MContextService`-Instanzen zurückzuliefern, die überhaupt eine kontextuelle Beschreibung aufweisen, die der Vorgabe entspricht. Somit werden beispielsweise nur Services gewählt, die eine Angabe in der `<motorMechanic>`-Form enthalten. Alle Service-Einträge, die hierzu keine Angabe enthalten, sind für die weitere Auswertung uninteressant.

Wie zuvor erwähnt, initiiert `MSelectImpl` die kontextabhängige Servicebestimmung. Dazu dient die Singleton-Klasse `MServiceSelectionManager` als Zugriffskomponente, deren Klassendiagramm in Abbildung 4.19 dargestellt ist. Der `MServiceSelectionManager` stellt hierfür die Methode `selectServiceByContext()` zur Verfügung, die als Resultat eine `MContextService`-Instanz zurückliefert.

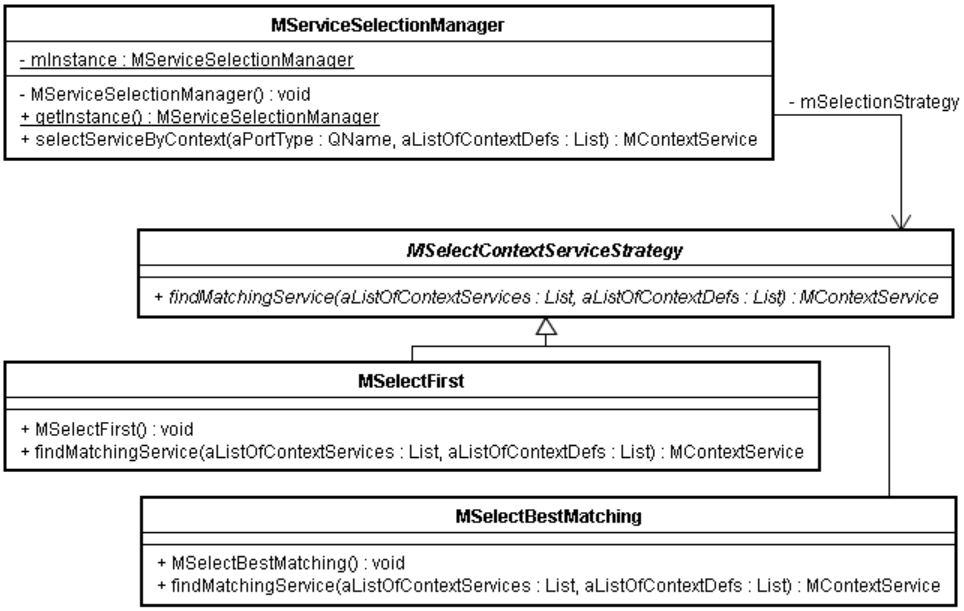

Abbildung 4.19: Klassendiagramm `MServiceSelectionManager`

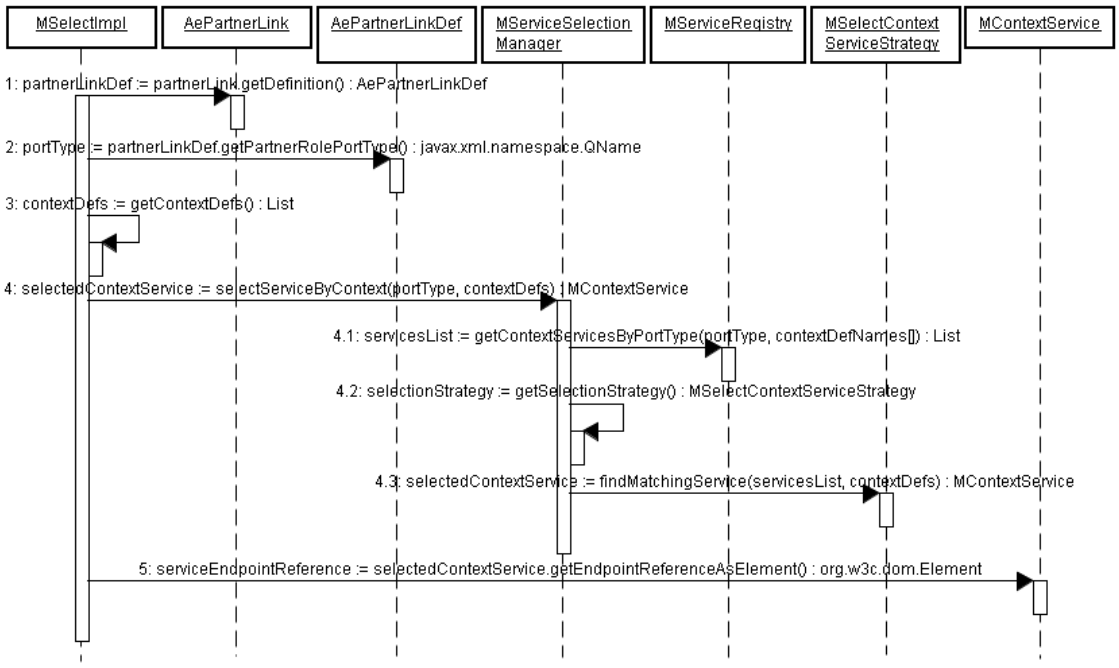

Abbildung 4.20: Sequenzdiagramm "Kontextabhängige Service-Auswahl"

Der Entwurf sieht vor, dass verschiedene Strategien zur kontextabhängigen Auswahl realisiert werden können. Dazu hält der `MServiceSelectionManager` die Referenz auf eine festzulegende `MSelectContextServiceStrategy`. Alle Unterklassen hiervon müssen die Methode `findMatchingService()` implementieren, wobei diese die unterschiedlichen Strategien umsetzt. Das Klassendiagramm in Abbildung 4.19 weist exemplarisch zwei Unterklassen auf: `MSelectFirst` und `MSelectBestMatching`. Erstere ist für eine prototypische Implementierung gedacht, wobei einfach der erste `MContextService` als Ergebnis zurückgeliefert wird. Während die zweite tatsächlich die übergebenen Kontextanforderungen berücksichtigt und den am besten geeigneten Service bestimmt.

Das Zusammenspiel der in die kontextabhängige Service-Auswahl involvierten Komponenten veranschaulicht abschließend das Sequenzdiagramm in Abbildung 4.20.

4.4.3 Kontextabhängige Ausführung einer Aktivität

Neben der `<dynamicInvoke>`-Aktivität wurde in Abschnitt 4.3.3 eine weitere kontextsensitive BPEL-Spracherweiterung definiert. Mittels `<contextActivity>` kann die Ausführung einer BPEL-Aktivität vom Übereinstimmen des vorherrschenden Kontexts mit definierten Vorgaben abhängig gemacht werden.

Modellierung der Definitions-Objekte für `<contextActivity>`

Im Gegensatz zu `<dynamicInvoke>` erweitert die `<contextActivity>` nicht das Verhalten einer im Sprachstandard definierten BPEL-Aktivität. Das Definitions-Objekt für `<contextActivity>` wird entsprechend nicht als Unterklasse einer konkreten Aktivitäts-Definition sondern als eigenständige, konkrete Implementierung `MContextActivityDef` von `AeActivityDef` abgeleitet modelliert.

Wie das Klassendiagramm in Abbildung 4.21 zeigt, implementiert die Klasse `MContextActivityDef` das Marker-Interface `IAeExtensionActivityDef` und analog zum Entwurf von `MActivityDynamicInvokeDef` die Schnittstellen `IMRequirementsParentDef` und `IMContextParentDef`. Dadurch stellt `MContextActivityDef` Methoden zum Zugriff auf die innerhalb `<contextActivity>` geschachtelte Element-Struktur `<requirements>` und `context` zur Verfügung.

`MContextActivityDef` enthält das Attribut `mMode`, das die Werte `"must"` oder `"optional"` entsprechend der Definition in Abschnitt 4.3.3 annimmt. Da das Element `<contextActivity>` genau eine Aktivität enthalten muss, die bei übereinstimmendem Kontext ausgeführt wird, weist das Modell der

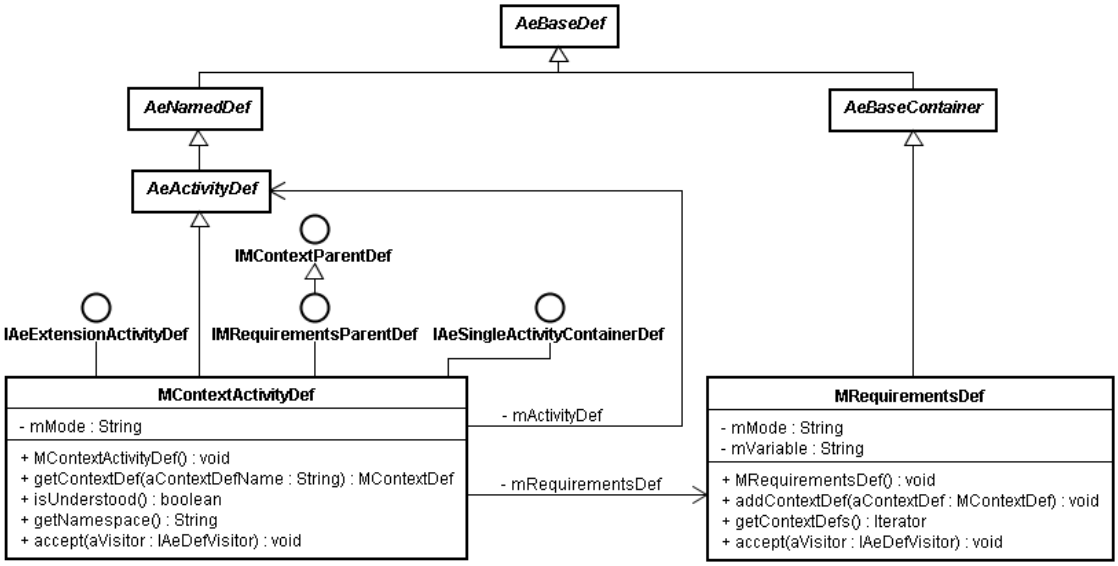

Abbildung 4.21: Klassendiagramm `MContextActivityDef`

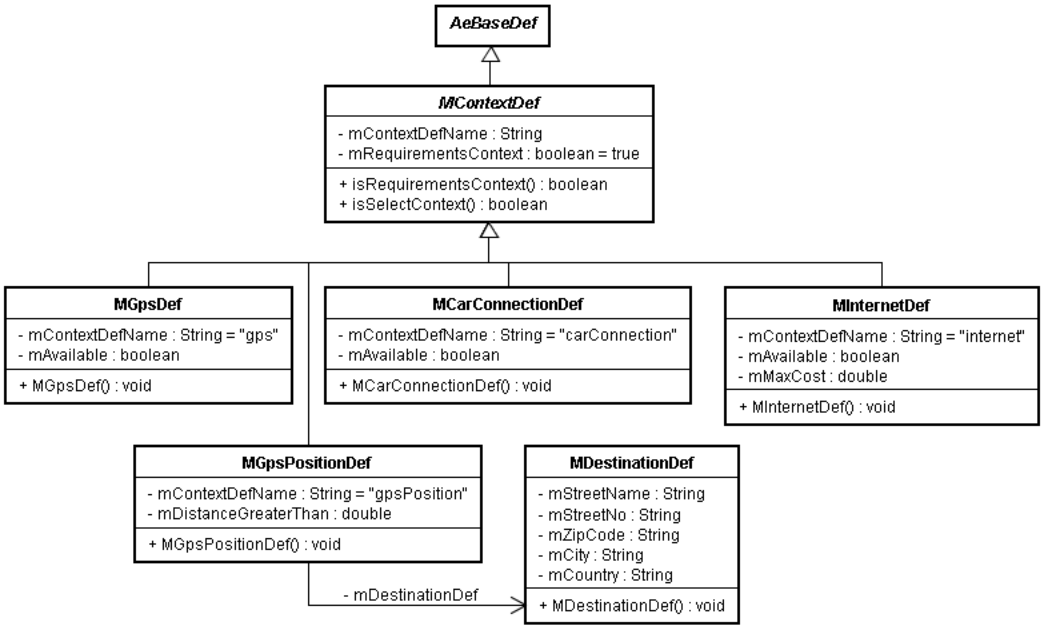

Abbildung 4.22: Klassendiagramm `MContextDef`

`MContextActivityDef` die Schnittstelle `IAeSingleActivityDef` auf und hält eine Referenz auf ein Definitions-Objekt von Typ `AeActivityDef`. Damit kann `MContextActivityDef` nicht nur die BPEL-eigenen Aktivitäten referenzieren, sondern ebenfalls die im Rahmen dieser Arbeit entworfenen Aktivitäten.

Die Konzeption der Klasse `MRequirementsDef` als `<requirements>`-Abbildung wurde analog zu der in Abschnitt 4.4.2 vorgestellten `<select>`-Abbildung in die `MSelectDef`-Klasse vorgenommen. Neben dem dort beschriebenen Attribut `mVariable` verfügt `MRequirementsDef` über das Attribut `mMode`. Dieses zeigt über die möglichen Werte `"now"` oder `"onEvent"` an, um welchen Ausführungstyp es sich handelt.

Wie schon bei der Modellierung der Kontext-Definitions-Objekte in Verbindung mit der `<dynamicInvoke>`-Aktivität beschrieben (siehe Abschnitt 4.4.2), bildet `MContextDef` die abstrakte Oberklasse für beliebige Kontextdefinitionen. Neben der in Abbildung 4.13 enthaltenen `MMotorMechanicDef` beinhaltet der Entwurf die in Abbildung 4.22 dargestellten Klassen. `MGpsDef`, `MGpsPositionDef`, `MCarConnectionDef`, `MInternetDef` und `MDestinationDef` bilden die für das Szenario "Auto-Pannendienst" notwendigen Kontext-Modelle ab.

Modellierung der Implementierungs-Objekte für `<contextActivity>`

Ebenso wie das Definitions-Objekt `MContextActivityDef` von der Klasse `AeActivityDef` abgeleitet wurde, wird das entsprechende Implementierungs-Objekt `MContextActivityImpl` als Unterklasse von `AeActivityImpl` modelliert, wie Abbildung 4.23 zeigt. Die Schnittstelle `IAeActivityParent` und die Referenz auf `AeActivityImpl` realisieren den Zusammenhang mit der kontextabhängig auszuführenden Aktivität.

Vom ebenfalls referenzierten Implementierungs-Objekt `MRequirementsImpl` existieren zwei konkrete Unterklassen. Je nachdem, ob es sich um eine Kontextauswertung als aktuelle Momentaufnahme oder mittels Event-Service handelt (vgl. dazu Abschnitt 4.3.3), wird `MRequirementsNowImpl` oder `MRequirementsOnEventImpl` verwendet. Letztgenannte Klasse ist hierzu als `EventListener` ausgelegt, der ein `MRequirementsContextChangedEvent` verarbeiten kann. Wie die Kontextauswertung abläuft, wird an späterer Stelle erläutert.

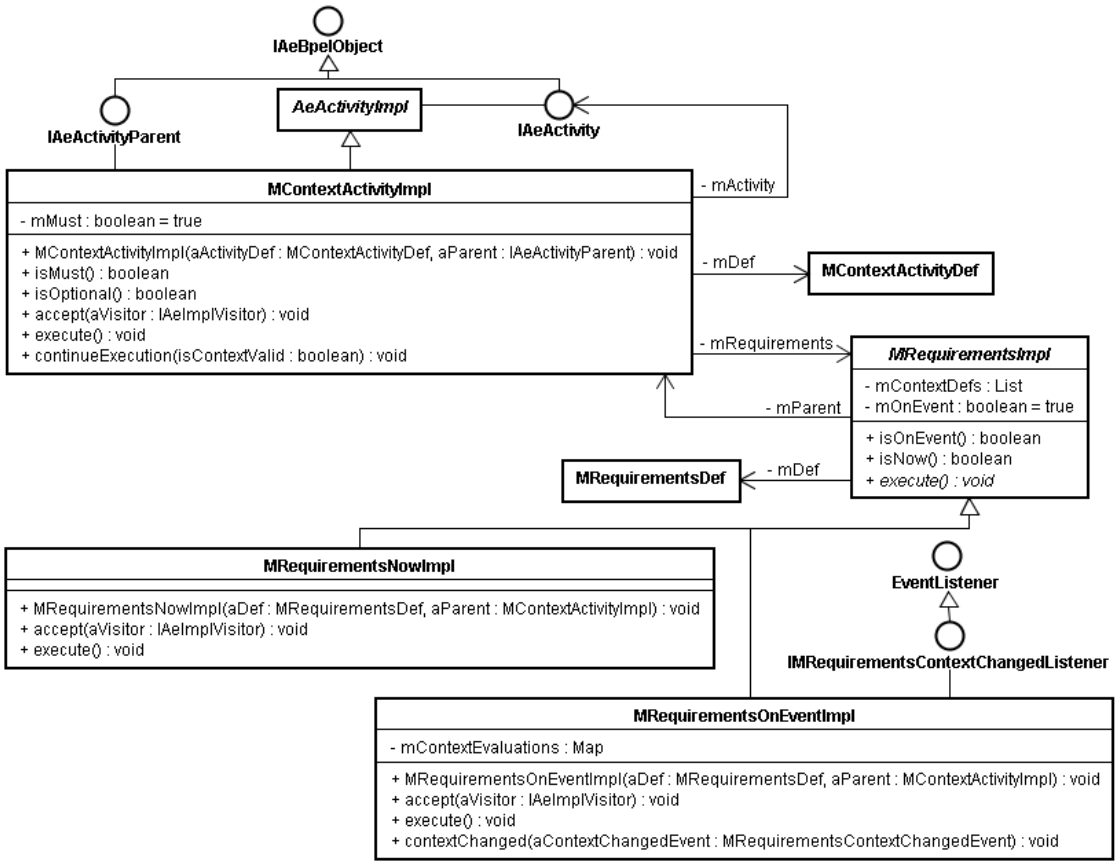

Abbildung 4.23: Klassendiagramm `MContextActivityImpl`

Umsetzung der Aktivitäts-Logik für `<contextActivity>`

Neben den Implementierungs-Objekten zur Steuerung des Prozessablaufs benötigt die Engine Komponenten, die aktuelle Daten für die verschiedenen Kontextausprägungen liefern und diese durch Abgleich mit der Kontextvorgabe in Form der Definitions-Objekte auswerten können. Abbildung 4.24 veranschaulicht den schematischen Aufbau und Zusammenhang der zu diesem Zweck entworfenen Komponenten. Die Implementierungs-Objekte, wozu `MRequirementsNowImpl` und `MRequirementsOnEventImpl` gehören, bilden dabei die oberste Schicht.

Die unterste Ebene besteht aus Sensoren, Adaptern oder ähnlich gearteten Elementen, die die technischen Daten der Engine-Umgebung liefern, wie beispielsweise GPS-Empfangsstärke, Internet-Zugang oder die Datenverbindung zum Auto. Im Rahmen dieser Arbeit werden diese Sensoren nicht näher untersucht; es wird davon ausgegangen, dass die darüber angesiedelte Schicht, der *Context Layer*, diese Daten zur Verfügung stellt. Als Vermittler zwischen dieser eher technischen Schicht und den

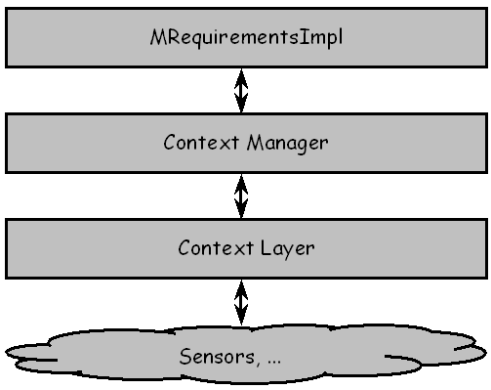

Abbildung 4.24: Komponenten zur Umsetzung der `<contextActivity>`-Logik

Implementierungs-Objekten wurden *Context Manager* entworfen, die darüber hinaus an der Kontextauswertung beteiligt sind. Im Folgenden werden die Bestandteile dieser Komponenten und ihr Zusammenspiel vorgestellt.

Neben den zu berücksichtigenden Kontextvorgaben aus der BPEL-Prozessdefinition, deren Modellierung in Form von Definitions-Objekten bereits in Abbildung 4.22 vorgestellt wurde, wird eine Datenstruktur zur Darstellung der von der Engine erfassten Kontextausprägungen benötigt. Abbildung 4.25 zeigt die abstrakte Klasse `MRequirementsContextEntity` und ihre für das Szenario "Auto-Pannendienst" notwendigen, konkreten Unterklassen `MGpsContext`, `MGpsPositionContext`, `MCarConnectionContext` und `MInternetContext`. Prinzipiell handelt es sich hier um engine-seitige Entsprechungen der prozess-basierten `MContextDef`-Definitions-Objekte. Daher weisen beide Datenstrukturen entsprechende Übereinstimmungen auf.

Die `MRequirementsContextEntity` schreibt für die engine-seitigen Kontextabbildungen die Implementierung der `isContextValidForContextDef()`-Methode vor. Diese erwartet als Parameter das entsprechende Kontext-Definitions-Objekt und führt für die aktuelle Entität den Vergleich des vorherrschenden Kontexts mit der Vorgabe durch. Die Vergleichslogik für die einzelnen Kontextausprägungen ist somit in den Unterklassen von `MRequirementsContextEntity` angesiedelt.

Ebenfalls in Abbildung 4.25 enthalten sind die verschiedenen Ausprägungen des Context Layer. Dabei kann ein Context Layer für mehrere Kontextausprägungen zuständig sein, wie am Beispiel des `MGpsLayer` deutlich wird. Jede `MContextLayer`-Unterklasse stellt Methoden für den Zugriff auf die `MRequirementsContextEntity`-Unterklassen zur Verfügung. Darüber hinaus reichen die Context Layer jede Kontextänderung über die Methode

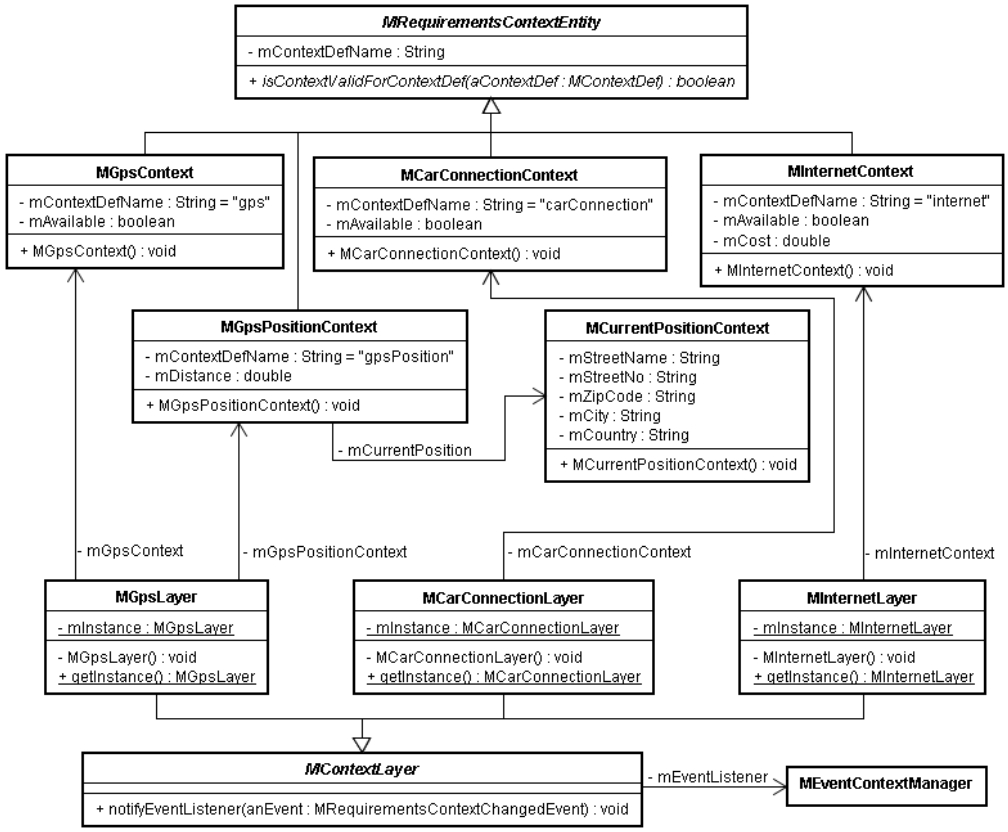

Abbildung 4.25: Klassendiagramm `MRequirementsContextEntity` und `MContextLayer`

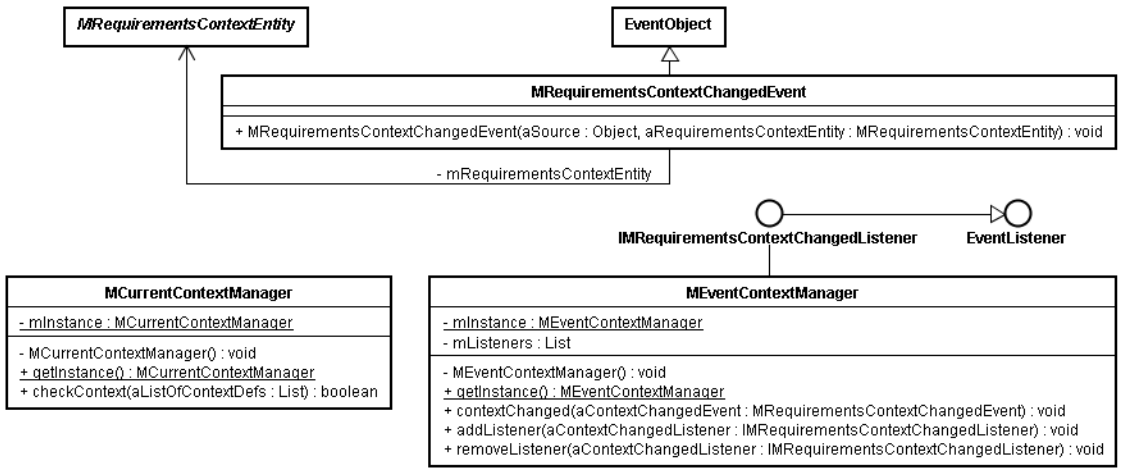

Abbildung 4.26: Klassendiagramm `MCurrentContextManager` und `MEventContextManager`

`notifyEventListener()` an den `MEventContextManager` weiter. Dieser ist für die Verarbeitung der event-gesteuerten Kontextauswertung zuständig.

Neben dem `MEventContextManager` beinhaltet der Engine-Entwurf den `MCurrentContextManager`, der für die Kontextauswertung als aktuelle Momentaufnahme verwendet wird. Beide Context Manager sind in Abbildung 4.26 enthalten, die ebenfalls das vom Event-Mechanismus verwendete `MRequirementsContextChangedEvent` zeigt. Es referenziert eine `MRequirementsContextEntity`, die die aktuellen Kontextwerte enthält.

Das Zusammenspiel aller vorgestellten Komponenten in Verbindung mit der `<contextActivity>` sollen die folgenden zwei Sequenzdiagramme aufzeigen. Dabei wird zuerst die Kontextauswertung als Momentaufnahme betrachtet und im Anschluss die event-getriebene Auswertung.

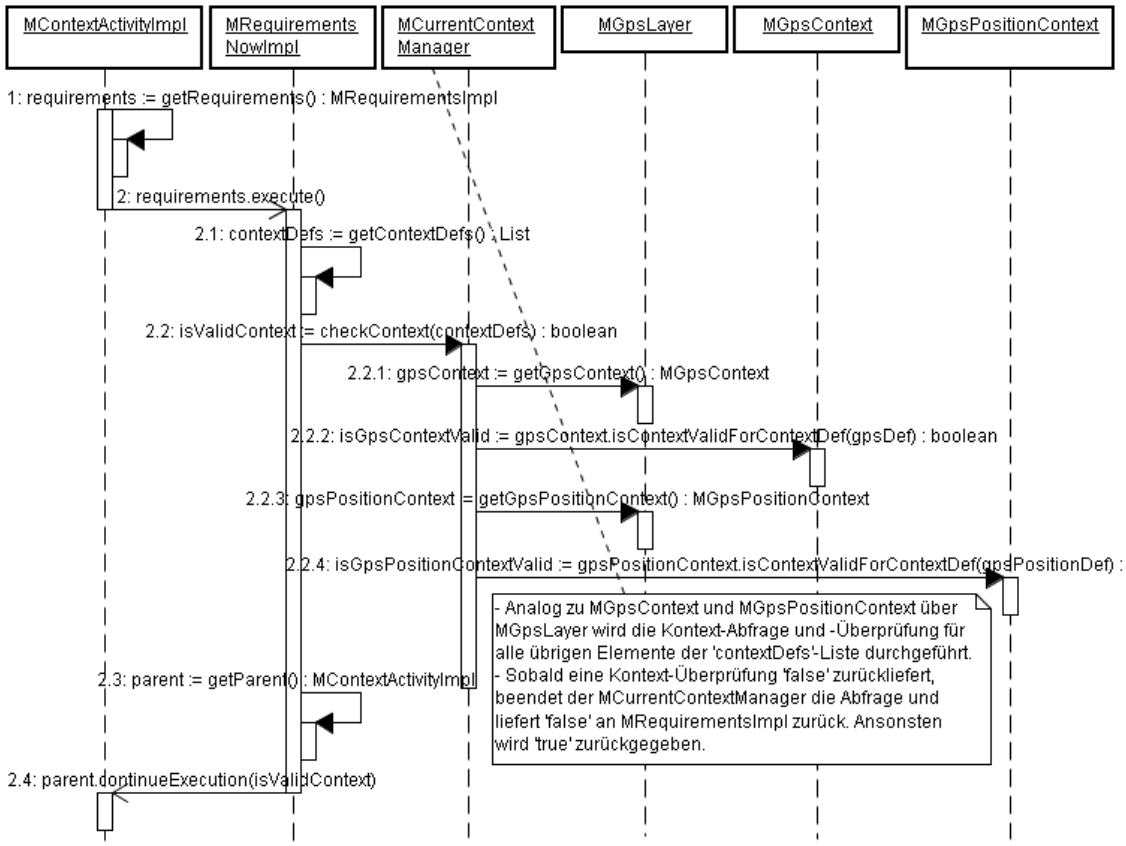

Abbildung 4.27: Sequenzdiagramm "Kontextauswertung als Momentaufnahme"

Wie Abbildung 4.27 zeigt, leitet `MContextActivityImpl` im Rahmen der `execute()`-Methode die Verarbeitung an `MRequirementsNowImpl` durch Auf-

ruf deren `execute()`-Methode weiter. `MRequirementsNowImpl` hat Zugriff auf die im Prozess definierten Kontextvorgaben als Liste von `MContextDef`-Objekten und leitet diese an den `MCurrentContextManager` durch Aufruf der `checkContext()`-Methode weiter.

Der `MCurrentContextManager` führt daraufhin den Abgleich mit den momentanen Kontextausprägungen durch. Hierzu fragt er beim `MGpsLayer` den `MGpsContext` ab und lässt diesen den Vergleich der Kontextvorgabe mit seinen Werten ausführen. Analog dazu verfährt der `MCurrentContextManager` für alle anderen `MContextDef`-Objekte, die er zur Überprüfung erhalten hat. Sobald eine `MRequirementsContextEntity` den Vorgaben nicht genügt, d.h. der Vergleich `"false"` zurückliefert, kann der `MCurrentContextManager` die gesamte Auswertung abbrechen. Das kumulierte Resultat liefert er an `MRequirementsNowImpl` zurück und diese reicht die Antwort an `MContextActivityImpl` per Aufruf der `continueExecution()`-Methode weiter.

Im Fall der event-gesteuerten Kontextauswertung, die das Sequenzdiagramm in Abbildung 4.28 darstellt, übergibt `MContextActivityImpl` die Verarbeitung über die `execute()`-Methode an `MRequirementsOnEventImpl`. Diese registriert sich durch Aufruf der `addListener()`-Methode beim `MEventContextManager` als Interessent für jegliche Kontextänderungen. Der als Singleton ausgelegte `MEventContextManager` ist seinerseits bei der Oberklasse aller Context Layer als Event Listener registriert, so dass im Falle einer Änderung der GPS-Daten der `MGpsLayer` diese Kontextänderung in Form eines `MRequirementsContextChangedEvent` an den `MEventContextManager` durch Aufruf von `contextChanged()` propagiert. Dieser leitet das Event an alle bei ihm registrierten `MRequirementsOnEventImpl`-Instanzen weiter.

Daraufhin holt sich `MRequirementsOnEventImpl` über das übergebene `MRequirementsContextChangedEvent` die zugehörige Implementierung von `MRequirementsContextEntity`. In diesem Fall handelt es sich dabei um `MGpsContext`, der die aktuellen Kontextwerte enthält. `MRequirementsOnEventImpl` liest den Namen der jeweiligen `MRequirementsContextEntity` aus und prüft, ob seine Liste aller zu überprüfenden `MContextDef`-Objekte für diese eine entsprechende Kontextvorgabe enthält. Nur im positiven Fall setzt `MRequirementsOnEventImpl` die Kontextauswertung fort, da nur dann überhaupt Interesse an diesem Kontext-Event besteht. Wenn also eine Kontextvorgabe für `MGpsContext` existiert, wird der Vergleich über die Methode `isContextValidForContextDef()` ausgeführt.

Im Fall der event-gesteuerten Variante von `<contextActivity>` soll die intern geschachtelte Aktivität ausgeführt werden, sobald alle Kontextvorgaben erfüllt sind. Das bedeutet, dass der Zeitpunkt festgestellt werden muss, zu dem al-

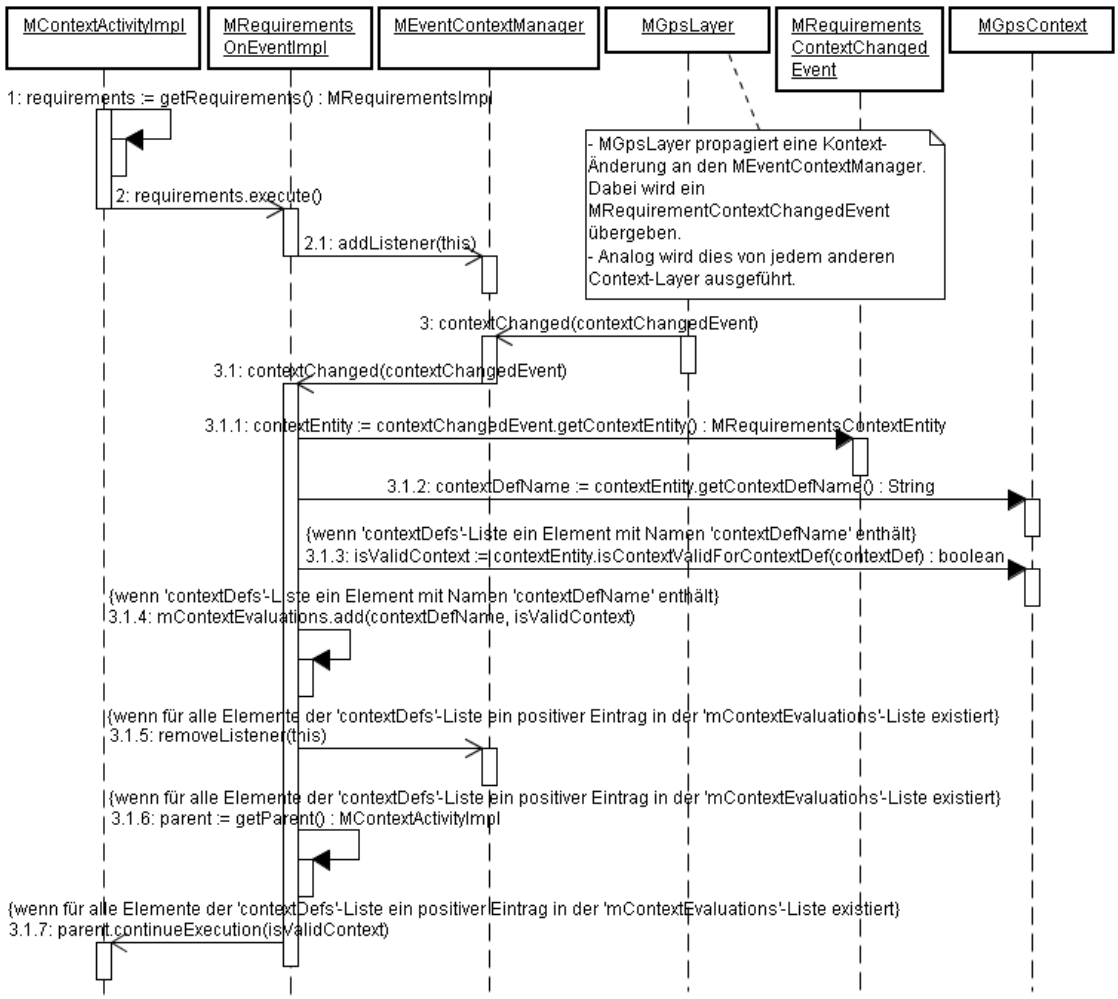

Abbildung 4.28: Sequenzdiagramm "Event-getriebene Kontextauswertung"

le Vorgaben gleichzeitig erfüllt sind. Es genügt nicht, sich zu merken, dass jede Vorgabe irgendwann einmal erreicht wurde. Der tatsächliche Kontext kann zwischenzeitlich wieder von der Vorgabe abgewichen sein. Zu diesem Zweck hält `MRequirementsOnEventImpl` eine Map vor, in der zu jeder zu erfüllenden Kontextdefinition das Vergleichsergebnis aus der letzten Kontextänderung abgelegt wird. Diesen Wert aktualisiert `MRequirementsOnEventImpl` nun und überprüft im Anschluss, ob alle in der Map enthaltenen Ergebnisse gleich `"true"` sind. In diesem Fall wäre der Zeitpunkt erreicht, an dem alle Kontextvorgaben erfüllt sind und das Ergebnis per `continueExecution()` zur Weiterverarbeitung an `MContextActivityImpl` übergeben wird.

4.4.4 Übertragbare Sub-Prozesse

Für die Unterstützung übertragbarer Sub-Prozesse wurden in Abschnitt 4.3.4 insgesamt drei Spracherweiterungen für BPEL entworfen: `<sendProcess>`, `<receiveProcess>` und `<startProcess>`. Dieser Abschnitt betrachtet die Integration der Aktivitäten in die ActiveBPEL Engine und stellt die Umsetzung der jeweiligen Aktivitäts-Logik vor.

Modellierung der Definitions-Objekte für `<sendProcess>`

Das Definitions-Objekt für die Erweiterung `<sendProcess>` wird als Unterklasse `MSendProcessActivityDef` von `AeActivityDef` abgeleitet modelliert, wie das Klassendiagramm in Abbildung 4.29 zeigt. Sie implementiert das Marker-Interface `IAeExtensionActivityDef` sowie das Interface `IAeSingleActivityContainerDef`, um den Zugriff auf die eingeschlossene Invoke-Aktivität zu realisieren. Dabei handelt es sich entweder um `AeActivityInvokeDef` oder `MActivityDynamicInvokeDef`.

Neben der Invoke-Aktivität schließt `<sendProcess>` die mit dem Sub-Prozess zu übermittelnden Daten als `<mobileProcessData>`-Element ein. Dieses wird auf das Definitions-Objekt `MMobileProcessDataDef` abgebildet, das – da es keine spezielle BPEL-Logik umsetzt – als direkte Unterklasse von `AeBaseDef` konzipiert ist. Gleiches gilt für die von `MMobileProcessDataDef` referenzierten Klassen `MProcessBpelDef`, `MProcessWsdlDef` und `MProcessStartParametersDef`, die die von `<mobileProcessData>` eingeschlossenen Elemente `<processBpel>`, `<processWsdl>` und `<processStartParameters>` realisieren. Die drei zuletzt genannten Definitions-Objekte weisen der Spracherweiterung entsprechend alle notwendigen Attribute auf. Beispielsweise enthält `MProcessBpelDef` das Attribut `mVariable`, das den Namen für die Variable aufnimmt, die den zu übertragenden BPEL-Prozess referenziert. Alternativ kann auf die in das BPEL-Element `<process>` eingeschlossene Prozessdefinition direkt über die angebundene, engine-eigene Klasse `AeProcessDef` zugegriffen werden.

Modellierung der Implementierungs-Objekte für `<sendProcess>`

Analog zur Modellierung des Definitions-Objekts `MSendProcessActivityDef` als Unterklasse von `AeActivityDef` wird das Implementierungs-Objekt für `<sendProcess>` von `AeActivityImpl` abgeleitet. Die entsprechende Klasse `MSendProcessActivityImpl` zeigt das Diagramm in Abbildung 4.30.

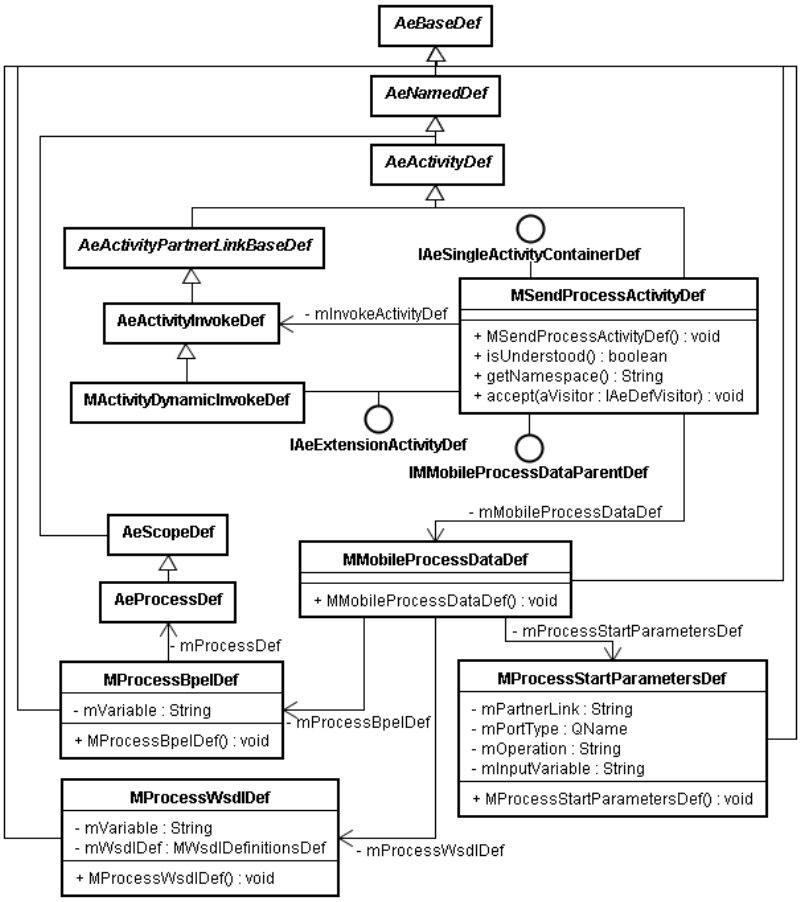

Abbildung 4.29: Klassendiagramm `MSendProcessActivityDef`

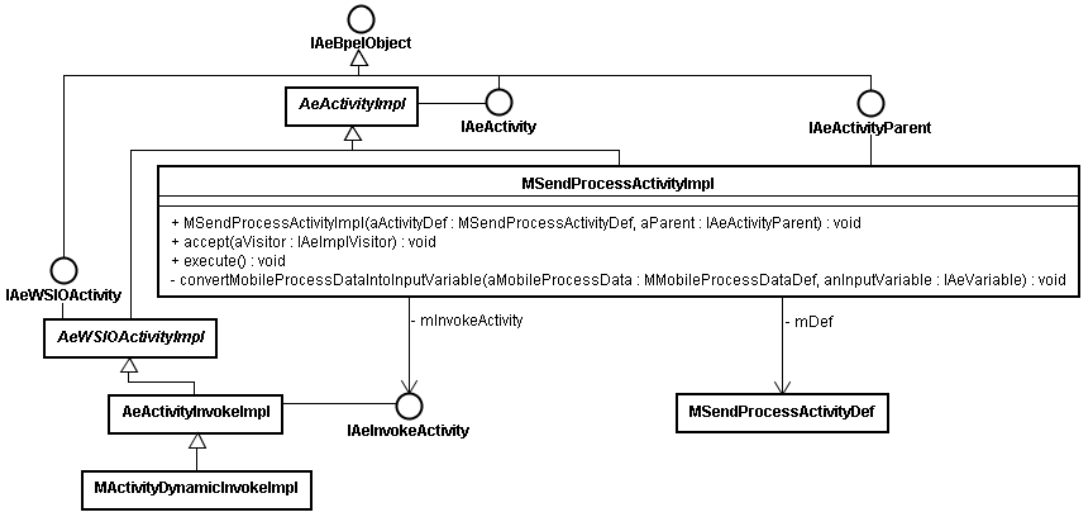

Abbildung 4.30: Klassendiagramm `MSendProcessActivityImpl`

Die Klasse `MSendProcessActivityImpl` implementiert die Schnittstelle `IAeActivityParent`, worüber der Zugriff auf die referenzierte `IAeInvokeActivity` realisiert wird. Bei `IAeInvokeActivity` handelt es sich entweder um die standardisierte `AeActivityInvokeImpl` oder die Erweiterung `MActivityDynamicInvokeImpl`. Daneben hält `MSendProcessActivityImpl` eine Referenz auf ihr Definitions-Objekt. Neben den vorgegebenen Methoden für Erweiterungsaktivitäten weist `MSendProcessActivityImpl` die private Methode `convertMobileProcessDataIntoInputVariable()` auf, die im Folgenden näher erläutert wird.

Umsetzung der Aktivitäts-Logik für `<sendProcess>`

Die Aufgaben von `<sendProcess>` bestehen darin, die Daten aus dem Element `<mobileProcessData>` in eine Nachricht umzuwandeln, deren Typ und interner Aufbau von der eingeschlossenen Invoke-Aktivität vorgegeben wird. Nachdem die Nachricht für den Web Service-Aufruf mit den entsprechenden Daten belegt wurde, muss die Invoke-Aktivität selbst ausgeführt werden.

Zu diesem Zweck wurde im Rahmen der Sprach-Konzeption nicht nur ein `<mobileProcessData>`-Element für den Einsatz in BPEL-Prozessen sondern ebenfalls ein `<mobileProcessData>`-Element für den Einsatz in WSDL-Dokumenten entworfen (vgl. Abschnitt 4.3.4). Wie den XML-Schemata in Anhang A entnommen werden kann, sind die Elemente in ihrem Aufbau nicht identisch, können jedoch leicht ineinander überführt werden. Um diese Konvertierung auszuführen, werden zunächst die in der Prozessdefinition getätigten Angaben für `<mobileProcessData>` benötigt. Das Sequenzdiagramm in Abbildung 4.31 zeigt den notwendigen Ablauf, der mit Aufruf der `execute()`-Methode von `MSendProcessActivityImpl` gestartet wird. Über das Definitions-Objekt `MSendProcessActivityDef` wird auf `MMobileProcessDataDef` zugegriffen.

Im Anschluss ruft `MSendProcessActivityImpl` das Definitions-Objekt der referenzierten `IAeInvokeActivity` ab. Dabei kann es sich konkret um `AeActivityInvokeDef` handeln, falls das Standard-`<invoke>` verwendet wurde. Andernfalls wird `MActivityDynamicInvokeDef` zurückgeliefert, wenn die Erweiterungsaktivität `<dynamicInvoke>` zum Einsatz kam. Über dieses Definitions-Objekt ist die zur Invoke-Aktivität gehörende `inputVariable` als `IAeVariable` zugreifbar. Diese besitzt bereits den geeigneten WSDL-Message-Typ und muss mit den Werten aus `MMobileProcessDataDef` belegt werden. Diese Aufgabe übernimmt die Methode `convertMobileProcessDataIntoInputVariable()` von

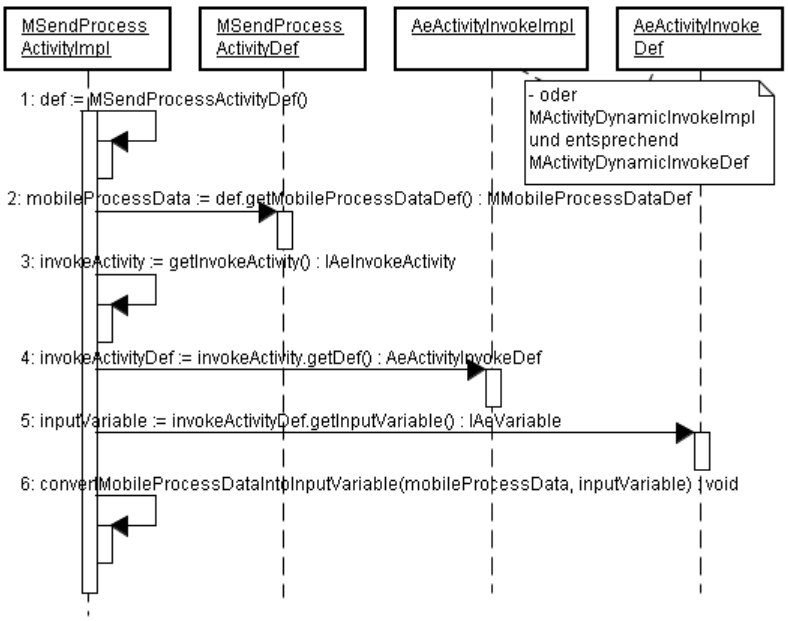

Abbildung 4.31: Sequenzdiagramm "Konvertierung von `<mobileProcessData>`"

`MSendProcessActivityImpl`. Damit ist die eingeschlossene Invoke-Aktivität entsprechend präpariert. Die `MSendProcessActivityImpl` kann ihre Arbeit beenden und die Ausführung über die Engine an die `execute()`-Methode der eingebundenen `AeActivityInvokeImpl` bzw. `MActivityDynamicInvokeImpl` übergeben. Dadurch findet die eigentliche Übertragung der Sub-Prozessdaten an den Empfänger statt.

Modellierung der Definitions-Objekte für `<receiveProcess>` und `<startProcess>`

Wie bei der Spracherweiterung festgelegt, wurde `<receiveProcess>` quasi als Kopie von der standardisierten `<receive>`-Aktivität entworfen. Aus diesem Grund wird das zugehörige Definitions-Objekt als Unterklasse der bestehenden Klasse `AeActivityReceiveDef` modelliert. Das Klassendiagramm in Abbildung 4.32 zeigt die entsprechende Klasse `MReceiveProcessActivityDef`.

Außer den üblichen Eigenschaften von Erweiterungsaktivitäten weist `MReceiveProcessActivityDef` keine Besonderheiten auf. Sie übernimmt unverändert sämtliche Eigenschaften und Methoden der Oberklasse, die die `<receive>`-Aktivität mit allen notwendigen Attributen und Zusammenhängen abbildet. Deren Oberklasse `AeActivityCreateInstanceBaseDef` bie-

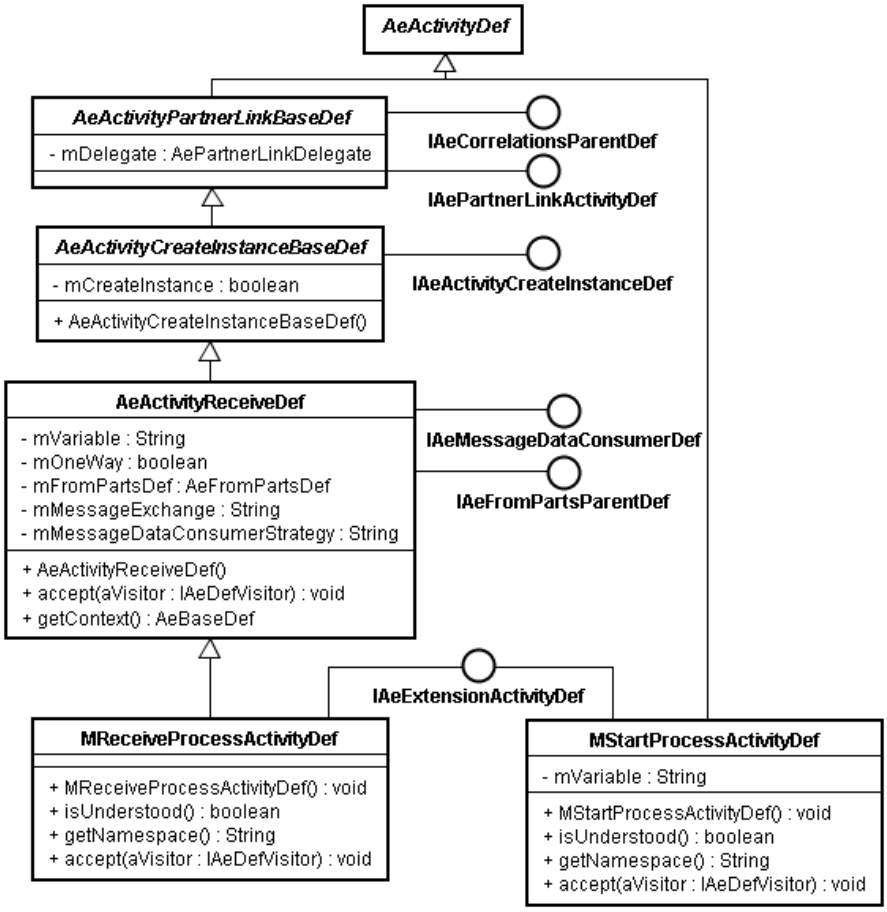

Abbildung 4.32: Klassendiagramm `MReceiveProcessActivityDef` und `MStartProcessActivityDef`

tet die Zugriffsmethoden auf das `mCreateInstance`-Attribut an. Die Klasse `AeActivityPartnerLinkBaseDef` stellt alle Informationen für Aktivitäten zur Verfügung, die sich auf einen Partner Link beziehen.

Die dritte Erweiterungsaktivität für übertragbare Sub-Prozesse ist das Element `<startProcess>`. Für dieses wurde die Klasse `MStartProcessActivityDef` als Definitions-Objekt entworfen, die direkt von der Oberklasse `AeActivityDef` abgeleitet ist. Neben dem Marker-Interface `IAeExtensionActivityDef` weist `MStartProcessActivityDef` das Attribut `mVariable` auf, um auf die Variable mit der über `<receiveProcess>` empfangenen Sub-Prozessdefinition zugreifen zu können.

Modellierung der Implementierungs-Objekte für `<receiveProcess>` und `<startProcess>`

Wie das Definitions-Objekt kann auch das Implementierungs-Objekt für `<receiveProcess>` vom Standard-`<receive>` abgeleitet werden. Hierfür stellt die Engine die Klasse `AeActivityReceiveImpl` bereit. Die Erweiterung wird über die Klasse `MReceiveProcessActivityImpl` realisiert, die dem Klassendiagramm in Abbildung 4.32 zu entnehmen ist. Wie alle Aktivitäten hält sie eine Referenz auf ihr Definitions-Objekt. Da keine über das Standardverhalten hinausgehenden Anforderungen an `<receiveProcess>` formuliert wurden, muss die Engine in diesem Fall keine spezielle Aktivitäts-Logik umsetzen. Die `execute()`-Methode der `MReceiveProcessActivityImpl` kann direkt an die `execute()`-Methode der Oberklasse verweisen.

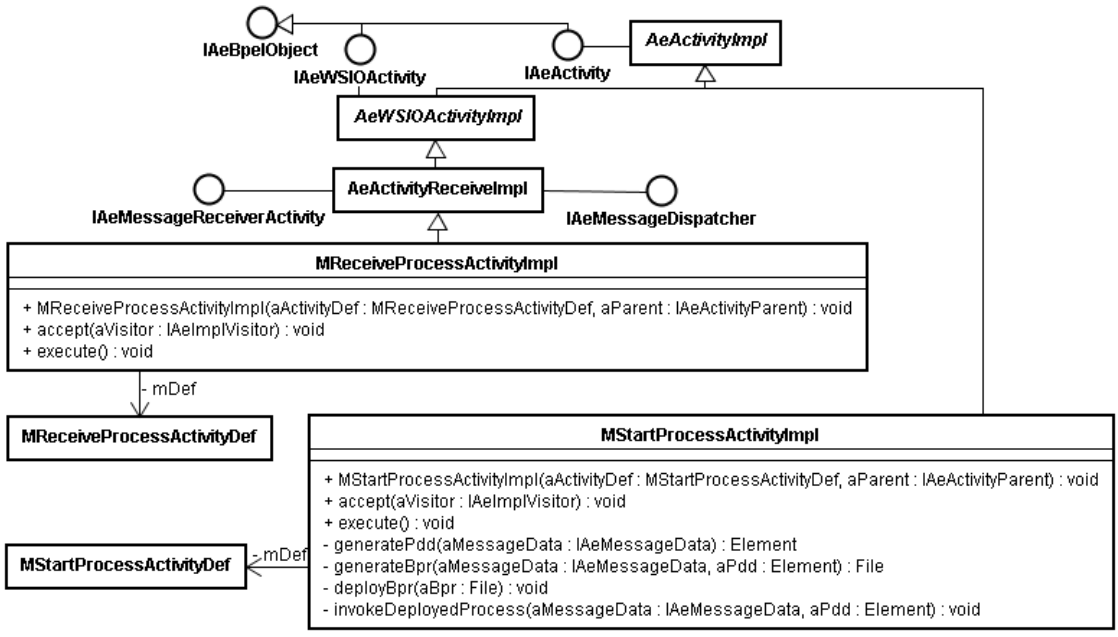

Abbildung 4.33: Klassendiagramm `MReceiveProcessActivityImpl` und `MStartProcessActivityImpl`

Ebenfalls in Abbildung 4.32 enthalten ist das Implementierungs-Objekt `MStartProcessActivityImpl` für die `<startProcess>`-Aktivität. Dieses wird direkt von `AeActivityImpl` abgeleitet und weist neben einer Referenz auf sein Definitions-Objekt und den Standard-Methoden vier private Methoden auf, die für die Umsetzung der Aktivitäts-Logik zuständig sind.

Umsetzung der Aktivitäts-Logik für `<startProcess>`

Die Anforderungen an `MStartProcessActivityImpl` bestehen aus dem Deployment der empfangenen Prozessbeschreibung und dem Starten des deployten Prozesses. Da der BPEL-Standard für das Deployment keine Vorgaben macht, verfolgen die Engine-Implementierungen unterschiedliche Strategien. Der grundsätzliche Deployment-Ablauf der ActiveBPEL Engine wurde einleitend in Abschnitt 3.2.2 beschrieben. Demnach wird das Deployment standardmäßig durch Ablage eines bpr-Archivs in ein entsprechendes Engine-Verzeichnis gestartet. Darüber hinaus bietet die ActiveBPEL Engine eine Administration über Web Services an, die unter anderem ein bpr-Archiv zum Deployment entgegen nehmen. Die dem zugrundeliegende Implementierung kann für das Deployment im Rahmen der `<startProcess>`-Aktivität wiederverwendet werden.

Damit ein Deployment durchgeführt werden kann, müssen die hierfür notwendigen Informationen als bpr-Archiv vorliegen. Neben den übermittelten BPEL- und WSDL-Dokumenten benötigt die ActiveBPEL Engine daher mindestens zwei weitere Dateien. Dabei handelt es sich um engine-spezifische Dokumente, die daher nicht im Rahmen der Sprach-Konzeption Berücksichtigung fanden. Einerseits muss ein Process Deployment Descriptor erstellt werden, der Details zu allen Partner Links und Binding-Informationen für den zu deployenden Prozess enthält. Andererseits ist eine Datei wsdlCatalog.xml zu erstellen, die alle benötigten WSDL- und XSD-Dokumente referenziert. Unter bestimmten Voraussetzungen kann ein Teil dieser Werte durch Analyse der übermittelten Daten generiert werden. Als erste Ausbaustufe sollen im Rahmen dieser Arbeit die notwendigen Informationen als Erweiterung der `<mobileProcessData>` mit übertragen werden.

Mit Beginn der Ausführung der `MStartProcessActivityImpl`-Klasse durch Aufruf der `execute()`-Methode muss zunächst die empfangene Nachricht ausgelesen werden. Dazu ruft `MStartProcessActivityImpl` über ihr Definition-Objekt den Namen der Variablen ab, die die Nachricht enthält. Die eigentliche Variable `IAeVariable` kann mittels `findVariable()` über eine Oberklasse ermittelt werden. Die Methode `getMessageData()` von `IAeVariable` liefert das zugehörige `IAeMessageData`-Objekt, das den übermittelten Sub-Prozess als `<mobileProcessData>`-Element enthält. Abbildung 4.34 zeigt den zugrundeliegenden Ablauf als Sequenzdiagramm.

Die empfangenen Daten werden in mehreren Schritten von `MStartProcessActivityImpl` ausgewertet und die notwendigen Deployment-Informationen zusammengestellt. Über die Methode `generatePdd()` wird zunächst der Deployment Descriptor erstellt. Dieser und die `IAeMessageData` dienen der `generateBpr()`-Methode dazu, die Informationen für wsdlCatalog.xml zu gewinnen und das vollständige bpr-Archiv zusammenzustellen. Dieses wird

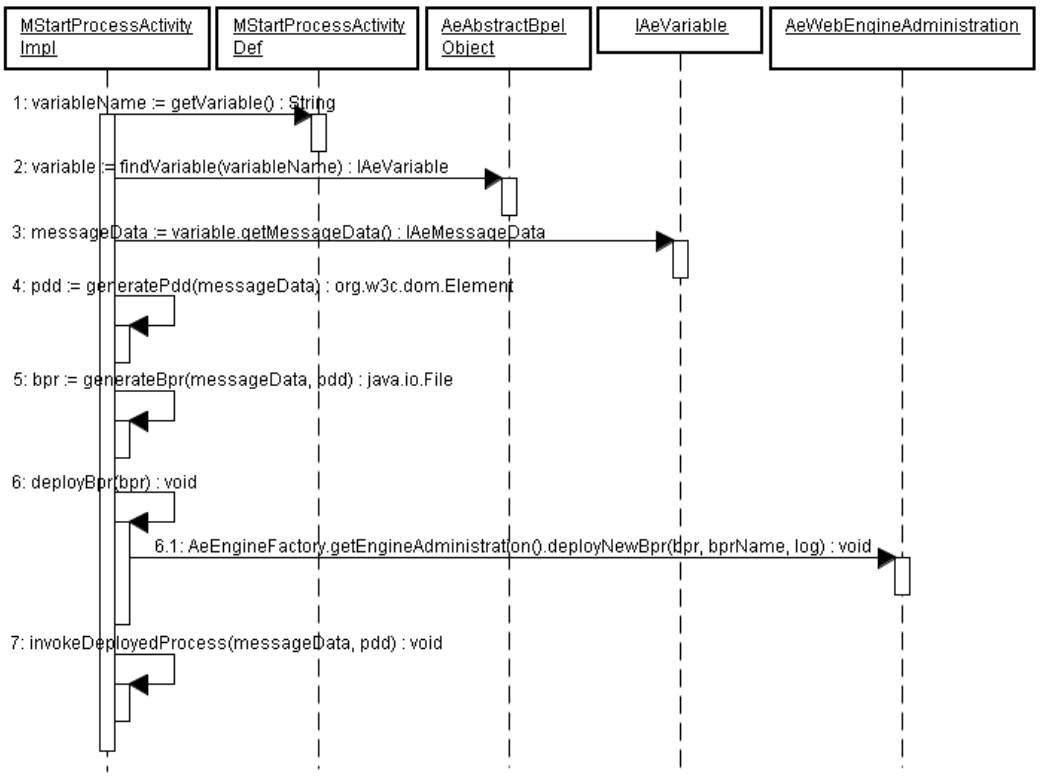

Abbildung 4.34: Sequenzdiagramm "Deployment und Starten des Sub-Prozesses"

daraufhin der Methode `deployBpr()` übergeben, die das Deployment-Archiv intern an die Klasse `AeWebEngineAdministration` weiterreicht, wodurch das Deployment ausgeführt wird.

Um den deployten Prozess zu starten, wird als letzter Schritt die Methode `invokeDeployedProcess()` ausgeführt. Diese erstellt auf Basis der übergebenen `IAeMessageData` eine Nachricht mit den erforderlichen Start-Parametern und führt mittels der im Deployment Descriptor enthaltenen Binding-Informationen den Web Service-Aufruf durch. Damit endet die Ausführung von `MStartProcessActivityImpl` und eine Instanz des deployten Sub-Prozesses wird durch die Engine erzeugt.

Kapitel 5

Realisierung

Dieses Kapitel stellt die Realisierung vor, die auf Grundlage der zuvor präsentierten Konzeption durchgeführt wurde. Es wird auf die Implementierung der Engine eingegangen und gezeigt, wie sich aus den Standard-BPEL-Elementen und den entworfenen, mobilen Spracherweiterungen die für das Szenario "Auto-Pannendienst" notwendigen, ausführbaren Prozessbeschreibungen erstellen lassen.

5.1 Grundlagen und Status der Realisierung

Da die Konzeption – speziell in Bezug auf die Sprache und Engine – sehr detailliert ausgearbeitet wurde, wird sich die Beschreibung der Realisierung auf die besonders interessanten Aspekte und ihre Umsetzung konzentrieren. Für Einzelheiten, die über die wiedergegebenen Ausschnitte der Implementierung hinausgehen, wird auf den Quellcode verwiesen, der der CD-ROM in Anhang C entnommen werden kann.

Die mobilen Spracherweiterungen für BPEL enthalten die Listings in Anhang A in Form von XML-Schemata. Die darin definierten, neuen BPEL-Aktivitäten sollen bei der Umsetzung der in Abschnitt 4.2 modellierten Geschäftsprozesse für das Szenario "Auto-Pannendienst" eingesetzt werden. Die damit verbundenen Aufgaben bilden einen Teil der Realisierung ab. Der andere Teil besteht in der Implementierung der Engine-Erweiterungen.

Aufgrund der zeitlichen Begrenzung der Masterarbeit war es nicht möglich, alle im Rahmen der Konzeption erzielten Ergebnisse zu realisieren. Wie bei der Zielsetzung der Arbeit erwähnt (vgl. Abschnitt 1.2), lag der Schwerpunkt der Realisierung auf der Erstellung der BPEL-Prozesse. Die folgenden zwei Abschnitte sollen den erreichten Realisierungsstatus für die Engine und die BPEL-Prozessdefinitionen aufzeigen.

5.1.1 Status der Engine-Implementierung

Parallel zum Entwurf der in Abschnitt 4.4 beschriebenen Engine-Konzeption, bei der sich nah am vorhandenen Quellcode der ActiveBPEL Engine orientiert wurde, sind erste Erweiterungen der Engine implementiert worden. Bisher wurde hauptsächlich die Integration der Spracherweiterungen `<dynamicInvoke>` und `<contextActivity>` vorangetrieben. Die bisherige Implementierung entspricht dabei den Entwurfsvorgaben aus der Engine-Konzeption. Über die in dieser Arbeit erwähnten Konzepte hinausgehend sind weitere Anpassungen der ActiveBPEL Engine notwendig, um ein lauffähiges System zu erhalten. Beispielsweise müssen für jede Erweiterung sogenannte Validation-Klassen erzeugt werden, die für die formale Validierung der neuen Sprachkonstrukte zuständig sind. Da derartige Aspekte im Hinblick auf mobile Workflows jedoch nicht weiter interessant sind, wurde darauf in dieser Arbeit nicht näher eingegangen.

Im Vorfeld der Konzeption wurde ein Durchstich vorgenommen, bei dem eine zunächst simpel aufgebaute Erweiterungsaktivität in die Engine integriert wurde. Dadurch sollten die Möglichkeiten und Zusammenhänge der Engine-Architektur in Bezug auf ihre Erweiterbarkeit untersucht werden. Es wurde ein entsprechend angepasster BPEL-Prozess auf der modifizierten Engine deployed; dieser war mit den vorgenommenen Erweiterungen ausführbar und verarbeitete dabei die neue Aktivität. Der Nachweis einer erfolgreichen und vollständigen Umsetzung der vorgestellten Engine-Erweiterungen konnte im Rahmen dieser Arbeit nicht erbracht werden, jedoch können die bisher erzielten Ergebnisse als durchaus positiv und erfolgversprechend bewertet werden.

5.1.2 Status der BPEL-Prozessdefinitionen

Der in Abschnitt 4.2 beschriebene Entwurf für die am Szenario "Auto-Pannendienst" beteiligten Geschäftsprozesse wurde vollständig umgesetzt. Das Ergebnis ist ein bpr-Archiv, das unter anderem die WSDL-Dokumente und Definitionen für ausführbare BPEL-Prozesse enthält. Diese wurden zum Teil um die in Abschnitt 4.3 vorgestellten mobilen Spracherweiterungen ergänzt. Da der Entwurf der BPEL-Spracherweiterungen erst im Anschluss an die konzeptionelle Modellierung der Geschäftsprozesse erarbeitet wurde, mussten bei deren Realisierung einige Anpassungen vorgenommen werden. Auf die notwendigen Abweichungen vom Entwurf und die Verwendung der mobilen Erweiterungsaktivitäten wird im folgenden Abschnitt näher eingegangen.

5.2 Umsetzung der mobilen Geschäftsprozesse in BPEL-Prozessdefinitionen

Für die Umsetzung der in Abschnitt 4.2.2 entworfenen Prozessabläufe "Pannenhilfe beauftragen" und "Pannenhilfe ausführen" wurden zunächst alle Web Services implementiert, die als Aktivitäten in die Ausführung dieser beiden Abläufe involviert sind. Dies beinhaltet alle definierten Services mit Ausnahme des `CarRepairService` und des `MotorMechanicService` (vgl. Abbildung 4.7). Da der Fokus auf den beiden übergeordneten Prozessabläufen liegt, wurden die genannten Web Services mit einer prototypischen Logik versehen. Zur Vereinfachung wurde für deren Umsetzung ebenfalls BPEL verwendet. Ihre Schnittstellen in Form von WSDL-Dokumenten entsprechen jeweils der Vorgabe aus der Konzeption.

Weitaus interessanter gestaltet sich die Umsetzung der beiden genannten Abläufe in ausführbare BPEL-Prozesse, da hierbei die mobilen Spracherweiterungen angewendet werden sollen. Ein Abgleich der Entwurfsvorgaben für die modellierten Prozesse, speziell die Abschnitte 4.2.2 und 4.2.4, mit den neuen BPEL-Aktivitäten führte dazu, dass der ursprüngliche Entwurf für den Sub-Prozess "Pannenhilfe ausführen" abgeändert werden musste. Der Ablauf selbst bleibt im Grunde erhalten, jedoch muss das entsprechende Aktivitätsdiagramm aus Abbildung 4.3 in zwei getrennte Abläufe unterteilt werden. Die vom KFZ-Mechaniker auszuführenden Aktivitäten werden erst zur Laufzeit von der Pannendienst-Zentrale übertragen und müssen demnach als eigenständige Prozessdefinition beschrieben werden. Der vom Mechaniker-System ausgeführte Prozess muss lediglich drei Aktivitäten enthalten: zum Empfangen und Starten des Sub-Prozesses sowie zum Ablehnen des Pannendienst-Auftrags. Die Abbildungen 5.1 und 5.2 zeigen die überarbeiteten Aktivitätsdiagramme, die durch Aufteilung des zuvor genannten entstanden sind.

Als Folge dieser Aufteilung musste ebenfalls die Definition der Service-Schnittstellen angepasst werden. Zusätzlich zu den bisher definierten Services wurde der `MMSubProcessService` eingeführt, der den neuen Sub-Prozess analog zu Abbildung 5.2 umsetzt. Die Änderungen im Vergleich zum ursprünglichen Entwurf (vgl. Abbildung 4.7) zeigt Abbildung 5.3.

Dabei weist die Schnittstelle des `MMSubProcessService` die Operation `startBreakdownOrder()` auf – worüber der deployte Sub-Prozess gestartet werden kann – und erwartet als Parameter den Typ `CarBreakdown`. Weiterhin musste die Schnittstelle des `MotorMechanicService` angepasst werden: Die Operation `placeBreakdownOrder()` muss den neu entworfenen Datentyp `MobileProcessData` unterstützen, da hiermit die Daten für den Sub-Prozess übertragen werden. Ebenfalls musste die Operation `returnOperationalReport()` vom `MotorMechanicService` zum

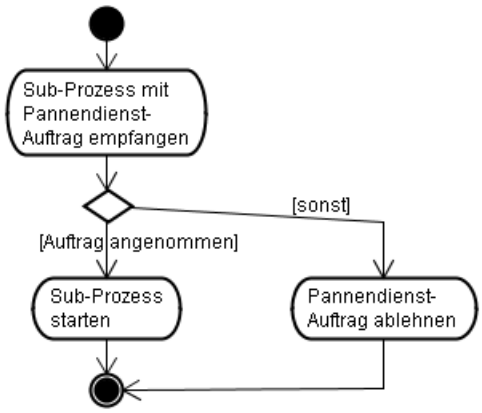

Abbildung 5.1: Aktivitätsdiagramm "Sub-Prozess mit Pannendienst-Auftrag empfangen und starten bzw. ablehnen"

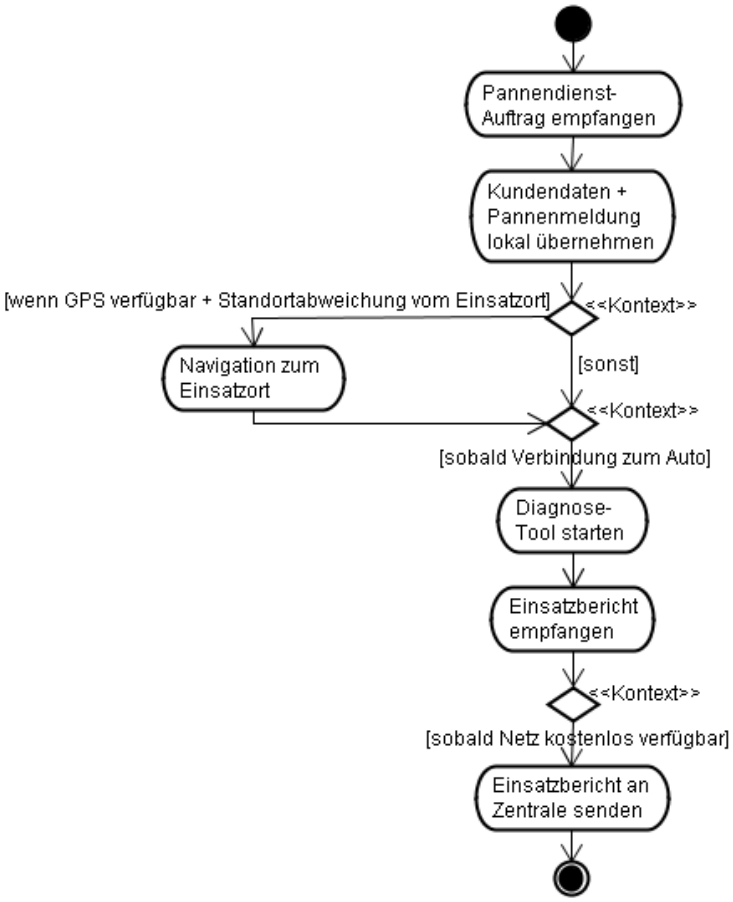

Abbildung 5.2: Aktivitätsdiagramm Sub-Prozess "Pannenhilfe ausführen"

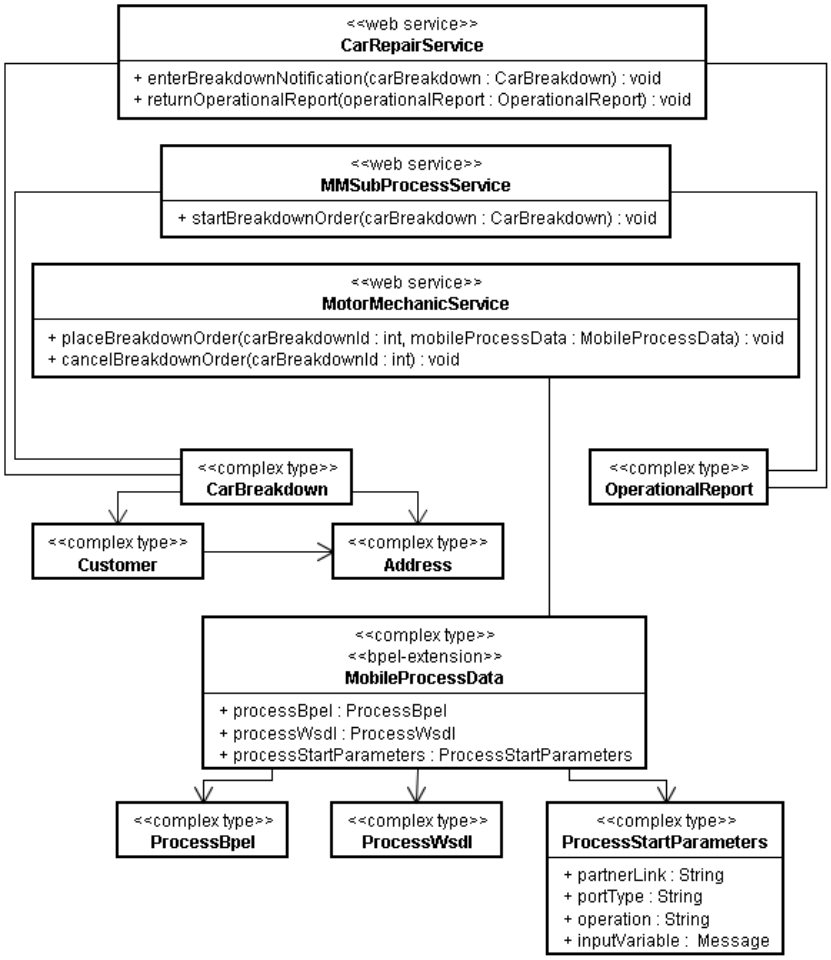

Abbildung 5.3: Schnittstellendefinition: Anpassung für mobile Prozesse

`CarRepairService` verschoben werden. Hierüber liefert der beim Mechaniker zur Ausführung kommende Sub-Prozess seinen Einsatzbericht an die Pannendienst-Zentrale ab.

Die Aufgaben für die weitere Umsetzung bestanden darin, für die beiden neu definierten Abläufe sowie den unveränderten Ablauf aus Abbildung 4.2 entsprechende mobile BPEL-Prozesse zu erstellen. Bei deren Realisierung wurde in zwei Schritten vorgegangen.

Zunächst wurde anhand der angepassten Entwurfsvorgaben für jeden der drei Abläufe eine Prozessdefinition erstellt, bei der ausschließlich Sprachelemente des WS-BPEL 2.0-Standards verwendet wurden. Dieses Vorgehen hatte den Vorteil, dass für die Implementierung zunächst die BPEL-Entwicklungsumgebung ActiveBPEL Designer verwendet werden konnte. Dadurch stand für den BPEL-Quellcode ein

graphischer Editor zur Verfügung, und die erstellten BPEL-Definitionen konnten automatisch auf Korrektheit überprüft werden. Die Entwicklung erfolgte iterativ und solange, bis ein valides Grundgerüst der Prozesse erreicht war.

Der zweite Schritt bestand in der Integration der mobilen BPEL-Aktivitäten und der damit notwendigen Anpassung der bisher fertiggestellten Prozessdefinitionen. Von hier an erfolgte die Implementierung ohne Werkzeugunterstützung, da die neuen Sprachelemente nicht vom ActiveBPEL Designer interpretiert werden und keine Möglichkeit der Definition eigener Konstrukte unterstützt wird. Das Ergebnis sind zwei BPEL-Prozessdefinitionen mit mobilen Sprachelementen, die Anhang B entnommen werden können.

Das darin aufgeführte Listing B.1 enthält den BPEL-Prozess für den `CarRepairService`. Dabei kamen die Spracherweiterungen `<sendProcess>` und `<dynamicInvoke>` zum Einsatz. Die `<sendProcess>`-Aktivität reicht im genannten Listing von den Zeilen 672-701. Darin geschachtelt ist die `<dynamicInvoke>`-Aktivität, da der Sub-Prozess an einen dynamisch bestimmten `MotorMechanicService` übertragen werden soll. Die zu übertragene BPEL-Definition für den `MMSubProcessService` kann den Zeilen 154-486 entnommen werden (und dessen WSDL-Dokument den Zeilen 490-576). Diese interne Sub-Prozessdefinition enthält insgesamt drei mal das Element `<contextActivity>`, in den Zeilen 392-411, 425-440 und 463-483. Darüber wird jeweils die Ausführung der kontextabhängigen Aktivitäten "Navigation zum Einsatzort", "Diagnose-Tool starten" und "Einsatzbericht an Zentrale senden" gesteuert.

Der Empfang des Sub-Prozesses ist in Listing B.2 enthalten, das den BPEL-Prozess für den `MotorMechanicService` wiedergibt. Die Aktivität `<receiveProcess>` kann den Zeilen 46-51 entnommen werden. Das Starten des Sub-Prozesses übernimmt die `<startProcess>`-Aktivität, in den Zeilen 59-61 zu finden.

Alle im Rahmen dieser Arbeit konzipierten Spracherweiterungen konnten in den BPEL-Prozessen wie entworfen umgesetzt werden.

Kapitel 6

Zusammenfassung und Ausblick

Dieses Kapitel fasst die gesamte Arbeit zusammen und führt die wichtigsten Ergebnisse auf. Im Anschluss wird eine Bewertung vorgenommen sowie aufgezeigt, welche Möglichkeiten für eine Weiterentwicklung bestehen.

6.1 Zusammenfassung

Mobile Workflows bzw. mobile Geschäftsprozesse bilden den thematischen Rahmen für die vorliegende Arbeit. Darin wurde gezeigt, wie sich die XML-basierte Prozess-Beschreibungssprache WS-BPEL 2.0 und eine geeignete BPEL-Engine um die Unterstützung mobiler Workflows erweitern lassen.

An eine einleitende Darstellung des Problemfeldes und der Ziele der Arbeit schlossen sich die thematischen Grundlagen an. Begonnen wurden diese mit einer Definition der Begriffe Geschäftsprozess und Workflow sowie einer Einführung in die Modellierung und Ausführung von Geschäftsprozessen. Insbesondere wurde auf den SOA-basierten Lösungsansatz durch Komposition von Web Services eingegangen und BPEL hierfür als standardisierte Beschreibungssprache für ausführbare Prozesse vorgestellt. Weiterhin wurden verschiedene Aspekte des Mobile Computing präsentiert, wozu u.a. Mobilität und Context Awareness gehören. Nach einer allgemeinen Betrachtung mobiler Workflows wurde eine thematische Einordnung der Arbeit vorgenommen.

Im Rahmen der Analyse wurden Erweiterungsmöglichkeiten von BPEL aufgezeigt, mit denen sich eigene Sprachelemente entwickeln und in den bestehenden Sprachumfang integrieren lassen. Dieser Mechanismus wurde für die Entwicklung der mobilen Spracherweiterungen genutzt. Als Ausführungsumgebung für BPEL-Prozesse

wurde eine Engine benötigt. Da im Rahmen der Masterarbeit keine komplette BPEL-Engine entwickelt werden konnte, wurden die mobilen Spracherweiterungen in eine verfügbare Engine integriert. Aus einer Auflistung geeigneter Open-Source-Implementierungen wurde hierfür die ActiveBPEL Engine ausgewählt und ihre Architektur vorgestellt. Darüber hinaus wurde das Szenario "Auto-Pannendienst" präsentiert, anhand dessen die Anforderungen an die BPEL-Erweiterungen zur Unterstützung mobiler Workflows aufgestellt wurden.

Basierend auf den aus der Analyse resultierenden Vorgaben wurde die Konzeption erarbeitet. Neben der Modellierung der szenario-spezifischen Geschäftsprozesse lag der Schwerpunkt auf dem konzeptionellen Entwurf der Sprachkonstrukte und deren Integration in die ActiveBPEL Engine. Insgesamt wurden fünf Aktivitäten als BPEL-Erweiterung entworfen. Durch `<dynamicInvoke>`, einer spezialisierten Form des Standard-Invoke, wurde der aufzurufende Partner-Service erst zur Laufzeit anhand von Kontextvorgaben bestimmt und dynamisch an den Operationsaufruf gebunden. Mittels `<contextActivity>` konnte die Ausführung einer eingebundenen BPEL-Aktivität abhängig vom Übereinstimmen des aktuellen Kontexts mit definierten Vorgaben gemacht werden. Für beide Aktivitäten wurde eine einheitliche Kontextmodellierung vorgesehen. Die Definition eines Sub-Prozesses und dessen Übertragung an eine andere Ausführungsumgebung konnte über die neuen Konstrukte `<sendProcess>`, `<receiveProcess>` und `<startProcess>` vorgenommen werden. Das Starten eines Prozesses wurde vom eigentlichen Empfang entkoppelt, um andere Aktivitäten zwischenschieben zu können, z.B. um die Ausführung des Sub-Prozesses abzulehnen.

Die Integration der neuen Funktionalitäten in die Engine wurde unter Berücksichtigung der bereits existierenden Engine-Architektur vorgenommen und sogenannte Definitions- und Implementierungs-Objekte entworfen, um die Sprachkonstrukte in engine-interne Abbildungen transformieren und interpretieren zu können. Für die Umsetzung der jeweiligen Aktivitäts-Logik wurden Lösungsansätze aufgezeigt.

Die Ergebnisse der Konzeption bildeten die Vorlage für die anschließende Realisierung. Die Erweiterungen der Engine konnten im Rahmen der Arbeit ansatzweise implementiert werden. Für das Szenario "Auto-Pannendienst" wurden die mobilen Geschäftsprozesse den Zielvorgaben entsprechend vollständig in ausführbare BPEL-Prozesse umgesetzt. Hierbei wurden die mobilen Spracherweiterungen wie entworfen mit Standard-BPEL-Elementen kombiniert.

6.2 Bewertung

Das Ziel der Arbeit, BPEL um die Unterstützung mobiler Teilnehmer zu erweitern, kann als erreicht angesehen werden. Der BPEL-Standard wurde um eigene Sprachkonstrukte ergänzt, wodurch sich Sub-Prozesse übertragen und zu berücksichtigende Kontextvorgaben definieren lassen. Diese können wie gefordert für eine dynamische Service-Auswahl oder für die kontextabhängige Ausführung einer Aktivität genutzt werden.

Bei der Entwicklung der Aktivitäten wurde sich an den Vorgaben des BPEL-Standards orientiert und teils auf bestehende Konstrukte zurückgegriffen. So können die Angaben zum Kontext nicht nur statisch sondern auch dynamisch durch Verweis auf eine BPEL-Variable vorgenommen werden. Dies ermöglicht eine für jede Prozessinstanz eigenständige Bestimmung der konkreten Werte sowie eine Änderung der Kontextvorgaben zur Laufzeit. Aufgrund der Tatsache, dass BPEL als Basis verwendet wurde, lassen sich bereits bestehende BPEL-Prozesse und Web Services wiederverwenden.

Neben dem betrachteten Szenario können die entworfenen Spracherweiterungen auch in anderen Zusammenhängen eingesetzt werden. Dabei sind den Kombinationsmöglichkeiten standardisierter BPEL-Elemente mit den mobilen Aktivitäten formal kaum Grenzen gesetzt. Eine Weiterentwicklung bzw. Erweiterung der konzipierten Sprachkonstrukte ist ebenfalls möglich, besonders um weitere Kontextbeschreibungen hinzufügen zu können.

Obwohl der Fokus bei der Konzeption speziell auf mobilen Workflows lag, sind die Erweiterungen für übertragbare Sub-Prozesse auch allgemein nutzbar. Sie können zur Verteilung und Delegation von Aufgaben an andere Systeme verwendet werden.

Die Integration der neuen BPEL-Aktivitäten in die ActiveBPEL Engine hat sich als sehr komplex und zeitaufwendig herausgestellt. Prinzipiell ließen sich eigene Erweiterungen gut in die bestehende Architektur integrieren, jedoch erforderte dies im Vorfeld eine ausführliche Analyse des bestehenden Codes, da aus der verfügbaren Dokumentation die Zusammenhänge und Aufgaben der Engine-Komponenten nicht ausreichend hervorgingen.

6.3 Ausblick

Mit Hilfe der im Rahmen der Masterarbeit entworfenen BPEL-Aktivitäten lassen sich mobile Workflows umsetzen. Die erzielten Ergebnisse zeigen, dass sich sowohl die Sprache BPEL als auch die ActiveBPEL Engine sinnvoll erweitern lassen, um

neue Funktionalitäten zu integrieren. Neben einer Fortführung der Implementierung und vollständigen Umsetzung der vorgestellten Engine-Konzeption bestehen weitere Möglichkeiten für eine Weiterentwicklung der bisherigen Ergebnisse.

Beim Entwurf der `<contextActivity>` wurden bisher zwei Aspekte, die sich aus dem sprachlichen Aufbau der Aktivität ergeben, nicht berücksichtigt: Im Fall der event-getriebenen Verarbeitung mittels `mode="onEvent"` wird solange gewartet, bis die Kontextvorgaben erfüllt sind. Hier sollte über die Einführung eines Timeout nachgedacht werden. Dafür könnte die `<contextActivity>` zusätzliche Parameter für eine Zeitvorgabe und eine alternativ auszuführende Aktivität vorsehen. Der zweite Aspekt betrifft den Kontextabgleich als Momentaufnahme: Wurde der Ausführungs-Mode der eingeschlossenen Aktivität als `"must"` definiert, aber der angegebene Kontext ist aktuell nicht erfüllt, gilt die gesamte Ausführung der `<contextActivity>` als beendet. Hier ist es ebenfalls sinnvoll, eine Alternativ-Aktivität angeben zu können. Dies könnte z.B. durch Ergänzung eines Fault-Handlers gelöst werden.

Sowohl bei `<contextActivity>` als auch `<dynamicInvoke>` wurde bisher davon ausgegangen, dass die Kontextinformationen automatisch aktualisiert werden. Die Konzeption der Context Layer und der Service-Registry muss hierfür detaillierter ausgearbeitet werden. Vor allem wurde bisher nicht das Alter der Kontextangaben berücksichtigt. Für eine zuverlässige Verarbeitung der kontextsensitiven Aktivitäten sind aktuelle Angaben jedoch die Grundvoraussetzung, so dass in diesem Punkt Optimierungsbedarf besteht.

Damit im Rahmen des Szenarios "Auto-Pannendienst" die Ausführung des Sub-Prozesses auf dem System des KFZ-Mechanikers erfolgreich verläuft, schreibt die Pannendienst-Zentrale vor, welche Services seitens der Mechaniker unterstützt werden müssen. Vor dem Hintergrund der B2B-Beziehung ist dieses Vorgehen möglich, jedoch sicherlich nicht immer erwünscht. Über ergänzende Konzepte könnte vor Versand des Sub-Prozesses sichergestellt werden, dass die involvierten Services überhaupt beim Empfänger verfügbar sind. Alternativ könnte der Sub-Prozess vom Empfänger an einen weiteren Teilnehmer weitergereicht werden, der über die notwendigen Services verfügt. Es wäre zu prüfen, ob hierfür auf Konzepte der `<dynamicInvoke>`-Aktivität zurückgegriffen werden kann, indem die angebotenen Services in die Kontextbeschreibung der potentiellen Teilnehmer aufgenommen werden.

Weiterhin ist die Pannendienst-Zentrale nicht in der Lage, den aktuellen Status des Sub-Prozesses zu ermitteln, während dieser auf einer anderen Engine ausgeführt wird. In der Praxis sind viele Fälle denkbar, in denen eine solche Möglichkeit notwendig ist; z.B. wenn ein liegengebliebener Autofahrer erneut in der Pannendienst-Zentrale anruft, um zu erfahren, warum der Mechaniker noch nicht eingetroffen ist. In diesem Fall kann die Zentrale bei Verwendung herkömmlicher Monitoring-Systeme

keine genauere Auskunft geben, als dass bereits ein Mechaniker beauftragt wurde. Hierfür wäre ein Mechanismus zu entwerfen, über den der übergeordnete Prozess den aktuellen Ausführungszustand des Sub-Prozesses abfragen kann.

Die beschriebenen Optimierungs- und Erweiterungsvorschläge stellen einen Ausschnitt der Möglichkeiten dar, wodurch mobile Workflows noch flexibler und zuverlässiger würden, um schließlich unter realen Bedingungen erfolgreich zum Einsatz zu kommen.

Abkürzungsverzeichnis

B2B	**B**usiness-**to**-**B**usiness
BPEL	**B**usiness **P**rocess **E**xecution **L**anguage
BPEL4WS	**B**usiness **P**rocess **E**xecution **L**anguage **for W**eb **S**ervices
DOM	**D**ocument **O**bject **M**odel
GPS	**G**lobal **P**ositioning **S**ystem
IT	**I**nformation **T**echnology
OASIS	**O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards
PDA	**P**ersonal **D**igital **A**ssistent
SW	**S**oft**w**are
UDDI	**U**niversal **D**escription, **D**iscovery and **I**ntegration
UML	**U**nified **M**odeling **L**anguage
URL	**U**niform **R**esource **L**ocator
WLAN	**W**ireless **L**ocal **A**rea **N**etwork
WS-BPEL	**W**eb **S**ervices **B**usiness **P**rocess **E**xecution **L**anguage
WSDL	**W**eb **S**ervices **D**efinition **L**anguage
XML	E**x**tensible **M**arkup **L**anguage
XSL	E**x**tensible **S**tylesheet **L**anguage
XSLT	**XSL T**ransformations

Literaturverzeichnis

[4Pe-2007] Active Endpoints (Hrsg.) ; Adobe (Hrsg.) ; BEA (Hrsg.) ; IBM (Hrsg.) ; Oracle (Hrsg.) ; SAP AG (Hrsg.): *WS-BPEL Extension for People (BPEL4People), Version 1.0*. 25.06.2007. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf, Zugriffsdatum: 18.10.2007

[Act-2007a] Active Endpoints (Hrsg.): *ActiveBPEL Designer*. 2007. http://www.active-endpoints.com/active-bpel-designer.htm, Zugriffsdatum: 10.12.2007

[Act-2007b] Active Endpoints (Hrsg.): *ActiveBPEL Designer and Eclipse Web Tools Project*. 2007. http://www.activebpel.org/samples/samples-3/eclipseWTP_and_BPEL/doc/index.html, Zugriffsdatum: 10.12.2007

[Act-2007c] Active Endpoints (Hrsg.): *ActiveBPEL Engine*. 2007. http://www.active-endpoints.com/active-bpel-engine-overview.htm5, Zugriffsdatum: 07.12.2007

[Act-2007d] Active Endpoints (Hrsg.): *ActiveBPEL InfoCenter v4.0*. 2007. http://www.activebpel.org/infocenter/ActiveBPEL/v40/index.jsp, Zugriffsdatum: 07.12.2007

[AH-2002] Aalst, Wil van d. ; Hee, Kees van: *Workflow Management: Models, Methods, and Systems*. Cambridge : MIT Press, 2002

[Apa-2007a] The Apache Software Foundation (Hrsg.): *Apache Axis*. 2007. http://ws.apache.org/axis, Zugriffsdatum: 08.12.2007

[Apa-2007b] The Apache Software Foundation (Hrsg.): *Apache Tomcat*. 2007. http://tomcat.apache.org, Zugriffsdatum: 07.12.2007

[Apa-2008] The Apache Software Foundation (Hrsg.): *Apache ODE*. 2008. http://ode.apache.org, Zugriffsdatum: 31.01.2008

[BPE-2003] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems (Hrsg.): *Business Process Execution Language for Web Services Version 1.1*. 05.05.2003. http://www.oasis-open.org/committees/download.php/2046/BPEL%20V1-1%20May%205%202003%20Final.pdf, Zugriffsdatum: 10.12.2007

[BPE-2007] OASIS WSBPEL TC (Hrsg.): *Web Services Business Process Execution Language Version 2.0*. 11.04.2007. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html, Zugriffsdatum: 15.10.2007

[BSI-2006] BSI, Bundesamt für Sicherheit in der Informationstechnik: *Pervasive Computing: Entwicklungen und Auswirkungen*. Bonn : SecuMedia Verlags-GmbH, 2006

[Car-2007] Carey, Sean: *Making BPEL Processes Dynamic*. In: *SOA Best Practices: The BPEL Cookbook*. 2007. http://www.oracle.com/technology/pub/articles/bpel_cookbook/carey.html, Zugriffsdatum: 13.12.2007

[Dey-2001] Dey, Anind K.: *Understanding and Using Context*. In: *Personal and Ubiquitous Computing*, Vol. 5, No. 1. London : Springer-Verlag. Februar 2001, S. 4-7. http://www.springerlink.com/content/1d9grxkjvquhpwkw/fulltext.pdf, Zugriffsdatum: 18.01.2008

[DSBW+-2006] Davis, John ; Sow, Daby ; Bourges-Waldegg, Daniela ; Guo, Chang J. ; Hoertnagl, Christian ; Stolze, Markus ; Eagle, Brian W. ; Yin, Ying: *Supporting Mobile Business Workflow with Commune*. In: *Proceedings of the 7th IEEE Workshop on Mobile Computing Systems and Applications*. 2006. http://ieeexplore.ieee.org/iel5/11140/35655/01691698.pdf?tp=&arnumber=1691698&isnumber=35655, Zugriffsdatum: 20.02.2008

[FL-2003] Frank, Ulrich ; Laak, Bodo L.: *Anforderungen an Sprachen zur Modellierung von Geschäftsprozessen*. Arbeitsberichte des Instituts für Wirtschaftsinformatik, Nr. 34, Fachbereich Informatik, Universität Koblenz-Landau. Januar 2003. http://www.uni-koblenz.de/~iwi/publicfiles/Arbeitsberichte/Nr34.pdf, Zugriffsdatum: 1.02.2008

[Gam-2007a] GAMM, Stephanie: *Mobile Prozesse – Kontextsensitive Service-Komposition.* Ausarbeitung im Rahmen der Veranstaltung Seminar, Hochschule für Angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik. 15.02.2007. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07/gamm/report.pdf, Zugriffsdatum: 18.10.2007

[Gam-2007b] GAMM, Stephanie: *Mobile Prozesse.* Seminar-Ausarbeitung im Rahmen der Veranstaltung Anwendungen 2, Hochschule für Angewandte Wissenschaften Hamburg, Fakultät Technik und Informatik. 22.02.2007. http://users.informatik.haw-hamburg.de/~ubicomp/projekte/master06-07-aw/gamm/report.pdf, Zugriffsdatum: 18.10.2007

[HKV-2007] HÉAM, Pierre-Cyrille ; KOUCHNARENKO, Olga ; VOINOT, Jérôme: *How to Handle QoS Aspects in Web Services Substitutivity Verification.* In: *Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises.* Juni 2007. http://ieeexplore.ieee.org/iel5/4407104/4407105/04407183.pdf?tp=&arnumber=4407183&isnumber=4407105, Zugriffsdatum: 20.02.2008

[HTTD-2007] HOECKE, Sofie van ; TAVEIRNE, Kristof ; TURCK, Filip de ; DHOEDT, Bart: *Dynamic Selection of Interactive eHomeCare Services.* In: *Proceedings of the IEEE International Conference on Pervasive Services.* Juli 2007. http://ieeexplore.ieee.org/iel5/4283874/4283875/04283941.pdf?tp=&arnumber=4283941&isnumber=4283875, Zugriffsdatum: 20.02.2008

[IBM-2005] IBM (Hrsg.) ; SAP (Hrsg.): *WS-BPEL Extension for Sub-processes – BPEL-SPE.* September 2005. http://www.ibm.com/developerworks/library/specification/ws-bpelsubproc/, Zugriffsdatum: 31.01.2008

[JBo-2008] JBOSS (Hrsg.): *JBoss jBPM.* 2008. http://www.jboss.com/products/jbpm, Zugriffsdatum: 31.01.2008

[JMS-2004] JURIC, Matjaz B. ; MATHEW, Benny ; SARANG, Poornachandra: *Business Process Execution Language for Web Services.* Birmingham : Packt Publishing, Oktober 2004

[KHC+-2005] KARASTOYANOVA, Dimka ; HOUSPANOSSIAN, Alejandro ; CILIA, Mariano ; LEYMANN, Frank ; BUCHMANN, Alejandro:

Extending BPEL for Run Time Adaptability. In: *Proceedings of the 2005 Ninth IEEE International EDOC Enterprise Computing Conference.* 2005. http://ieeexplore.ieee.org/iel5/10337/32900/01540664.pdf?tp=&arnumber=1540664&isnumber=32900, Zugriffsdatum: 20.02.2008

[Kun-2005] Kunze, Christian P.: *Unterstützung mobiler Prozesse im Mobile Computing.* In: *Technischer Bericht zum 1. GI/ITG KuVS Fachgespräch Energiebewusste Systeme und Methoden.* 2005. http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/261/fgsfpc.pdf, Zugriffsdatum: 26.10.2006

[KZL-2006] Kunze, Christian P. ; Zaplata, Sonja ; Lamersdorf, Winfried: *Mobile Process Description and Execution.* In: *Proceedings of the 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems.* 2006. http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/268/DAIS06CKSZWL.pdf, Zugriffsdatum: 17.11.2006

[LR-2000] Leymann, Frank ; Roller, Dieter: *Production Workflow – Concepts and Techniques.* Upper Saddle River : Prentice Hall International, 2000

[Mat-2003] Mattern, Friedemann: *Vom Verschwinden des Computers – Die Vision des Ubiquitous Computing.* In: *Total Vernetzt.* Berlin : Springer-Verlag, 2003. – S. 1–41. http://www.vs.inf.ethz.ch/publ/papers/VerschwComp.pdf

[Mel-2007] Melzer, Ingo: *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis.* 2. Auflage. München : Elsevier, Spektrum Akademischer Verlag, 2007

[MPR-2001] Murphy, Amy L. ; Picco, Gian P. ; Roman, Gruia-Catalin: *LIME: A Middleware for Physical and Logical Mobility.* In: *Proceedings of the 21st International Conference on Distributed Computer Systems.* 2001. http://www.inf.unisi.ch/faculty/murphy/Papers/icdcs01.pdf, Zugriffsdatum: 18.01.2008

[NL-2004] Newcomer, Eric ; Lomow, Greg: *Understanding SOA with Web Services.* Upper Saddle River : Addison-Wesley, 2004

[NN-2007] Nowicki, Adam ; Niesler, Andrzej: *Towards Dynamic Integration: Deployment of UDDI Registries in a Service-Oriented Archi-*

tecture. In: *Proceedings of 1st International Working Conference on Business Process and Services Computing*. September 2007

[Pel-2003] PELTZ, Chris: *web service orchestration*. Januar 2003. http://www.itee.uq.edu.au/~infs3204/interesting_websites/WSOrchestration.pdf, Zugriffsdatum: 27.01.2008

[Rot-2005] ROTH, Jörg: *Mobile Computing – Grundlagen, Technik, Konzepte*. 2. Auflage. Heidelberg : dpunkt Verlag, 2005

[RS-2004] REICHERT, Manfred ; STOLL, Dietmar: *Komposition, Choreographie und Orchestrierung von Web Services – Ein Überblick*. 2004. http://sunsite.informatik.rwth-aachen.de/Societies/GI-EMISA/forum/content_04_2/21%20ulm%20fachbeitrag.pdf, Zugriffsdatum: 22.01.2008

[Sat-1996] SATYANARAYANAN, Mahadev: *Fundamental Challenges in Mobile Computing*. In: *Proceedings of the fifteenth annual ACM Symposium on Principles of distributed computing*. Mai 1996. http://portal.acm.org/ft_gateway.cfm?id=248053&type=pdf&coll=Portal&dl=ACM&CFID=49841210&CFTOKEN=34291866, Zugriffsdatum: 14.01.2008

[SAW-1994] SCHILIT, Bill ; ADAMS, Norman ; WANT, Roy: *Context-Aware Computing Applications*. In: *Proceedings of the Workshop on Mobile Computing Systems and Applications*. Dezember 1994. http://ieeexplore.ieee.org/iel2/3875/11297/00512740.pdf?tp=&arnumber=512740&isnumber=11297, Zugriffsdatum: 16.02.2007

[Wei-1991] WEISER, Mark: *The Computer for the 21st Century*. In: *Scientific American*. Vol. 265, No. 3. 1991, S. 94–104

[Wik-2007] WIKIPEDIA (Hrsg.): *Visitor pattern*. 09.12.2007. http://en.wikipedia.org/wiki/Visitor_pattern, Zugriffsdatum: 09.12.2007

[Win-2007] WINTERBERG, Torsten: *BPEL wird erwachsen...* In: *Javamagazin*. Ausgabe 7.07. 2007. http://www.opitz-consulting.de/pdf/veroeffentlichungen/javamagazin_07_07.pdf, Zugriffsdatum: 28.01.2008

[WS–2004] W3C (Hrsg.): *Web Services Architecture*. 11.02.2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211, Zugriffsdatum: 22.01.2008

[WS–2007] W3C (Hrsg.): *Web Services Addressing*. 2006-2007. http://www.w3.org/2002/ws/addr, Zugriffsdatum: 08.12.2007

[WSD-2001] W3C (Hrsg.): *Web Services Description Language (WSDL) 1.1*. 15.03.2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315, Zugriffsdatum: 08.11.2007

[Zap-2005] Zaplata, Sonja: *Prozessintegration in Middleware für mobile Systeme*. Diplomarbeit, Universität Hamburg, Arbeitsbereich VSIS. 08.11.2005. http://www.informatik.uni-hamburg.de/SWT/attachments/LVTermine/Prozessintegration%20in%20Middleware.pdf, Zugriffsdatum: 07.11.2006

Anhang A

XML-Schemata der BPEL-Spracherweiterungen

Anhang A enthält die XML-Schemata der WS-BPEL 2.0-Spracherweiterungen, die im Rahmen dieser Arbeit zur Unterstützung mobiler Workflows konzipiert wurden. Die Sprachelemente für den Einsatz in BPEL-Prozessen können den Listings A.1 und A.2 entnommen werden. Listing A.3 beinhaltet die Spracherweiterung für den WSDL-Einsatz.

Das folgende Listing A.1 zeigt das XML-Schema für die BPEL-Spracherweiterungen `<dynamicInvoke>`, `<contextActivity>`, `<sendProcess>`, `<receiveProcess>` und `<startProcess>`.

```
<xsd:schema
   targetNamespace="http://informatik.haw-hamburg.de/master/
      wsbpel/2.0/mobileExtensions"
   xmlns="http://informatik.haw-hamburg.de/master/wsbpel/2.0/
      mobileExtensions"
   xmlns:context="http://informatik.haw-hamburg.de/master/wsbpel/
       2.0/mobileContextData"
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" blockDefault="#all">

   <!-- import other namespaces -->
   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
      schemaLocation="http://www.w3.org/2001/xml.xsd" />
```

```
<xsd:import namespace="http://docs.oasis-open.org/wsbpel/2.0/
   process/executable"
   schemaLocation="ws-bpel_executable.xsd" />
<xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"
   schemaLocation="wsdl-2003-02-11.xsd" />
<xsd:import namespace="http://informatik.haw-hamburg.de/
   master/wsbpel/2.0/mobileContextData"
   schemaLocation="mobileContextData.xsd" />


<!-- base types for extensible elements -->
<!-- see namespace
    http://docs.oasis-open.org/wsbpel/2.0/process/executable
   -->
<xsd:complexType name="tExtensibleElements">
   <xsd:sequence>
      <xsd:element ref="documentation" minOccurs="0"
         maxOccurs="unbounded" />
      <xsd:any namespace="##other" processContents="lax"
         minOccurs="0" maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other"
      processContents="lax" />
</xsd:complexType>

<xsd:complexType name="tExtensibleMixedNamespaceElements">
   <xsd:sequence>
      <xsd:element name="documentation" type="tDocumentation"
         minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="extensions" type="tExtensions"
         minOccurs="0" />
   </xsd:sequence>
   <xsd:anyAttribute namespace="##other"
      processContents="lax" />
</xsd:complexType>

<xsd:element name="documentation" type="tDocumentation" />
<xsd:complexType name="tDocumentation" mixed="true">
   <xsd:sequence>
      <xsd:any processContents="lax" minOccurs="0"
         maxOccurs="unbounded" />
   </xsd:sequence>
   <xsd:attribute name="source" type="xsd:anyURI" />
   <xsd:attribute ref="xml:lang" />
</xsd:complexType>
```

```
<xsd:complexType name="tExtensions">
   <xsd:sequence>
      <xsd:any namespace="##other" processContents="lax"
         minOccurs="0" maxOccurs="unbounded" />
   </xsd:sequence>
</xsd:complexType>


<!-- all mobile extension activities (for WS-BPEL 2.0) -->
<xsd:group name="mobileExtensionActivity">
   <xsd:choice>
      <xsd:element ref="dynamicInvoke"/>
      <xsd:element ref="contextActivity"/>
      <xsd:element ref="sendProcess"/>
      <xsd:element ref="receiveProcess"/>
      <xsd:element ref="startProcess"/>
   </xsd:choice>
</xsd:group>


<!-- element "dynamicInvoke" to be used within
   "bpel:extensionActivity" -->
<!-- for sub elements "bpel:targets", "bpel:sources" and
   attributes "name", "suppressJoinFailure" see type
   "bpel:tActivity" -->
<xsd:element name="dynamicInvoke" type="tDynamicInvoke" />
<xsd:complexType name="tDynamicInvoke">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedNamespaceElements">
         <xsd:sequence>
            <xsd:element ref="bpel:targets" minOccurs="0" />
            <xsd:element ref="bpel:sources" minOccurs="0" />
            <xsd:element name="correlations"
               type="bpel:tCorrelationsWithPattern"
               minOccurs="0"/>
            <xsd:element ref="bpel:catch" minOccurs="0"
               maxOccurs="unbounded"/>
            <xsd:element ref="bpel:catchAll" minOccurs="0"/>
            <xsd:element ref="bpel:compensationHandler"
               minOccurs="0"/>
            <xsd:element ref="bpel:toParts" minOccurs="0"/>
            <xsd:element ref="bpel:fromParts" minOccurs="0"/>
            <xsd:element ref="select" minOccurs="1"
               maxOccurs="1" />
            <xsd:any namespace="##other"
```

```
                processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
          </xsd:sequence>
          <xsd:attribute name="name" type="xsd:NCName" />
          <xsd:attribute name="suppressJoinFailure"
             type="bpel:tBoolean" use="optional" />
          <xsd:attribute name="partnerLink" type="xsd:NCName"
             use="required" />
          <xsd:attribute name="portType" type="xsd:QName"
             use="optional" />
          <xsd:attribute name="operation" type="xsd:NCName"
             use="required" />
          <xsd:attribute name="inputVariable"
             type="bpel:BPELVariableName" use="optional" />
          <xsd:attribute name="outputVariable"
             type="bpel:BPELVariableName" use="optional" />
       </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>

 <xsd:element name="select" type="tSelect" />
 <xsd:complexType name="tSelect">
    <xsd:complexContent>
       <xsd:extension base="tExtensibleElements">
          <xsd:sequence>
             <xsd:group ref="selectContext" minOccurs="1"
                maxOccurs="unbounded" />
          </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
 </xsd:complexType>

 <xsd:group name="selectContext">
    <xsd:choice>
       <xsd:element ref="motorMechanic" />
       <!-- more elements for "selectContext" -->
    </xsd:choice>
 </xsd:group>


 <!-- element "contextActivity" to be used within
    "bpel:extensionActivity" -->
 <!-- for sub elements "bpel:targets", "bpel:sources",
    "bpel:activity" and attributes "name",
    "suppressJoinFailure" see type "bpel:tActivity" -->
 <xsd:element name="contextActivity" type="tContextActivity" />
```

```
<xsd:complexType name="tContextActivity">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedNamespaceElements">
         <xsd:sequence>
            <xsd:element ref="bpel:targets" minOccurs="0" />
            <xsd:element ref="bpel:sources" minOccurs="0" />
            <xsd:element ref="requirements" minOccurs="0"
               maxOccurs="1" />
            <xsd:choice minOccurs="1" maxOccurs="1">
               <xsd:group ref="bpel:activity" minOccurs="1"
                  maxOccurs="1" />
               <xsd:group ref="mobileExtensionActivity"
                  minOccurs="1" maxOccurs="1" />
            </xsd:choice>
            <xsd:any namespace="##other"
               processContents="lax" minOccurs="0"
               maxOccurs="unbounded" />
         </xsd:sequence>
         <xsd:attribute name="name" type="xsd:NCName" />
         <xsd:attribute name="suppressJoinFailure"
            type="bpel:tBoolean" use="optional" />
         <xsd:attribute name="mode"
            type="tContextActivityMode" default="must" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="requirements" type="tRequirements" />
<xsd:complexType name="tRequirements">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
         <xsd:sequence>
            <xsd:group ref="requirementContext" minOccurs="1"
               maxOccurs="unbounded" />
         </xsd:sequence>
         <xsd:attribute name="mode" type="tRequirementsMode"
            default="onEvent" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:group name="requirementContext">
   <xsd:choice>
      <xsd:element ref="gps" />
      <xsd:element ref="gpsPositionDeviation" />
      <xsd:element ref="carConnection" />
```

```
      <xsd:element ref="internet" />
      <!-- more elements for "requirementContext" -->
   </xsd:choice>
</xsd:group>

<xsd:simpleType name="tContextActivityMode">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="must" />
      <xsd:enumeration value="optional" />
   </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="tRequirementsMode">
   <xsd:restriction base="xsd:string">
      <xsd:enumeration value="onEvent" />
      <xsd:enumeration value="now" />
   </xsd:restriction>
</xsd:simpleType>

<!-- context-elements to be used as "selectContext" and/or as
    "requirementContext" for extension-activities
   <dynamicInvoke> and <contextActivity> -->
<xsd:element name="motorMechanic" type="tMotorMechanic" />
<xsd:complexType name="tMotorMechanic">
   <xsd:complexContent>
      <xsd:extension base="context:tMotorMechanic">
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="gps" type="tGps" />
<xsd:complexType name="tGps">
   <xsd:complexContent>
      <xsd:extension base="context:tGps">
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="gpsPositionDeviation"
   type="tGpsPositionDeviation" />
<xsd:complexType name="tGpsPositionDeviation">
   <xsd:complexContent>
```

```
      <xsd:extension base="context:tGpsPositionDeviation">
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="carConnection" type="tCarConnection" />
<xsd:complexType name="tCarConnection">
   <xsd:complexContent>
      <xsd:extension base="context:tCarConnection">
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="internet" type="tInternet" />
<xsd:complexType name="tInternet">
   <xsd:complexContent>
      <xsd:extension base="context:tInternet">
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>


<!-- element "sendProcess" to be used within
   "bpel:extensionActivity" -->
<!-- for sub elements "bpel:targets", "bpel:sources",
   "bpel:invoke" and attributes "name", "suppressJoinFailure"
   see type "bpel:tActivity" -->
<xsd:element name="sendProcess" type="tSendProcess" />
<xsd:complexType name="tSendProcess">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedNamespaceElements">
         <xsd:sequence>
            <xsd:element ref="bpel:targets" minOccurs="0" />
            <xsd:element ref="bpel:sources" minOccurs="0" />
            <xsd:choice minOccurs="1">
               <xsd:element ref="bpel:invoke" />
               <xsd:element ref="dynamicInvoke" />
            </xsd:choice>
            <xsd:element ref="mobileProcessData"
               minOccurs="1" />
```

```
            <xsd:any namespace="##other"
               processContents="lax" minOccurs="0"
               maxOccurs="unbounded" />
         </xsd:sequence>
         <xsd:attribute name="name" type="xsd:NCName" />
         <xsd:attribute name="suppressJoinFailure"
            type="bpel:tBoolean" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="mobileProcessData"
   type="tMobileProcessData" />
<xsd:complexType name="tMobileProcessData">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleMixedNamespaceElements">
         <xsd:sequence>
            <xsd:element ref="processBpel" minOccurs="1" />
            <xsd:element ref="processWsdl" minOccurs="1" />
            <xsd:element ref="processStartParameters"
               minOccurs="1" />
            <xsd:any namespace="##other"
               processContents="lax" minOccurs="0"
               maxOccurs="unbounded" />
         </xsd:sequence>
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="processBpel" type="tProcessBpel" />
<xsd:complexType name="tProcessBpel">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
         <xsd:sequence minOccurs="0">
            <xsd:element ref="bpel:process" minOccurs="1" />
         </xsd:sequence>
         <xsd:attribute name="variable"
            type="bpel:BPELVariableName" use="optional" />
      </xsd:extension>
   </xsd:complexContent>
</xsd:complexType>

<xsd:element name="processWsdl" type="tProcessWsdl" />
<xsd:complexType name="tProcessWsdl">
   <xsd:complexContent>
      <xsd:extension base="tExtensibleElements">
```

```
            <xsd:sequence minOccurs="0">
               <xsd:element ref="wsdl:definitions" minOccurs="1"
                  />
            </xsd:sequence>
            <xsd:attribute name="variable"
               type="bpel:BPELVariableName" use="optional" />
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:element name="processStartParameters"
      type="tProcessStartParameters" />
   <xsd:complexType name="tProcessStartParameters">
      <xsd:complexContent>
         <xsd:extension base="tExtensibleMixedNamespaceElements">
            <xsd:attribute name="partnerLink" type="xsd:NCName"
               use="required" />
            <xsd:attribute name="portType" type="xsd:QName"
               use="optional" />
            <xsd:attribute name="operation" type="xsd:NCName"
               use="required" />
            <xsd:attribute name="inputVariable"
               type="bpel:BPELVariableName" use="required" />
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <!-- element "receiveProcess" to be used within
      "bpel:extensionActivity" -->
   <!-- for sub elements "bpel:targets", "bpel:sources",
      "correlations", "bpel:fromParts" and attributes "name",
      "suppressJoinFailure" see type "bpel:tActivity" -->
   <xsd:element name="receiveProcess" type="tReceiveProcess" />
   <xsd:complexType name="tReceiveProcess">
      <xsd:complexContent>
         <xsd:extension base="tExtensibleMixedNamespaceElements">
            <xsd:sequence>
               <xsd:element ref="bpel:targets" minOccurs="0"/>
               <xsd:element ref="bpel:sources" minOccurs="0"/>
               <xsd:element name="correlations"
                  type="bpel:tCorrelations" minOccurs="0"/>
               <xsd:element ref="bpel:fromParts" minOccurs="0"/>
               <xsd:any namespace="##other"
                  processContents="lax" minOccurs="0"
                  maxOccurs="unbounded" />
            </xsd:sequence>
```

```
            <xsd:attribute name="name" type="xsd:NCName"/>
            <xsd:attribute name="suppressJoinFailure"
               type="bpel:tBoolean" use="optional"/>
            <xsd:attribute name="partnerLink" type="xsd:NCName"
               use="required"/>
            <xsd:attribute name="portType" type="xsd:QName"
               use="optional"/>
            <xsd:attribute name="operation" type="xsd:NCName"
               use="required"/>
            <xsd:attribute name="variable"
               type="bpel:BPELVariableName" use="required"/>
            <xsd:attribute name="createInstance"
               type="bpel:tBoolean" default="no"/>
            <xsd:attribute name="messageExchange"
               type="xsd:NCName" use="optional"/>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <!-- element "startProcess" to be used within
      "bpel:extensionActivity" -->
   <!-- for sub elements "bpel:targets", "bpel:sources" and
      attributes "name", "suppressJoinFailure" see type
      "bpel:tActivity" -->
   <xsd:element name="startProcess" type="tStartProcess" />
   <xsd:complexType name="tStartProcess">
      <xsd:complexContent>
         <xsd:extension base="tExtensibleMixedNamespaceElements">
            <xsd:sequence>
               <xsd:element ref="bpel:targets" minOccurs="0" />
               <xsd:element ref="bpel:sources" minOccurs="0" />
               <xsd:any namespace="##other"
                  processContents="lax" minOccurs="0"
                  maxOccurs="unbounded" />
            </xsd:sequence>
            <xsd:attribute name="name" type="xsd:NCName" />
            <xsd:attribute name="suppressJoinFailure"
               type="bpel:tBoolean" use="optional" />
            <xsd:attribute name="variable"
               type="bpel:BPELVariableName" use="required" />
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

Listing A.1: XML-Schema der BPEL-Spracherweiterungen

Das folgende Listing A.2 zeigt das XML-Schema für die Kontext-Elemente `<motorMechanic>`, `<gps>`, `<gpsPositionDeviation>`, `<carConnection>` und `<internet>`, die im Rahmen der BPEL-Spracherweiterungen `<dynamicInvoke>` und `<contextActivity>` Verwendung finden.

```
<xsd:schema
   targetNamespace="http://informatik.haw-hamburg.de/master/
      wsbpel/2.0/mobileContextData"
   xmlns="http://informatik.haw-hamburg.de/master/wsbpel/2.0/
      mobileContextData"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" blockDefault="#all">

   <!-- import other namespaces -->
   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
      schemaLocation="http://www.w3.org/2001/xml.xsd" />

   <!-- context-elements to be used as "selectContext" and/or as
      "requirementContext" for extension-activities
      <dynamicInvoke> and <contextActivity> -->
   <xsd:element name="motorMechanic" type="tMotorMechanic" />
   <xsd:complexType name="tMotorMechanic">
      <xsd:sequence minOccurs="0">
         <xsd:element name="car" type="xsd:string" />
         <xsd:element name="breakdownDescription"
            type="xsd:string" />
         <xsd:element name="duration" type="xsd:double"
            minOccurs="0" />
         <xsd:element ref="destination" />
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="destination" type="tAddress" />
   <xsd:complexType name="tAddress">
      <xsd:sequence>
         <xsd:element name="streetName" type="xsd:string"/>
         <xsd:element name="streetNo" type="xsd:string"/>
         <xsd:element name="zipCode" type="xsd:string"/>
         <xsd:element name="city" type="xsd:string"/>
         <xsd:element name="country" type="xsd:string"/>
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="gps" type="tGps" />
   <xsd:complexType name="tGps">
```

```
      <xsd:sequence minOccurs="0">
         <xsd:element name="available" type="tBoolean" />
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="gpsPositionDeviation"
      type="tGpsPositionDeviation" />
   <xsd:complexType name="tGpsPositionDeviation">
      <xsd:sequence minOccurs="0">
         <xsd:element name="distanceGreaterThan"
            type="xsd:double" />
         <xsd:element ref="destination" />
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="carConnection" type="tCarConnection" />
   <xsd:complexType name="tCarConnection">
      <xsd:sequence minOccurs="0">
         <xsd:element name="available" type="tBoolean" />
      </xsd:sequence>
   </xsd:complexType>

   <xsd:element name="internet" type="tInternet" />
   <xsd:complexType name="tInternet">
      <xsd:sequence minOccurs="0">
         <xsd:element name="available" type="tBoolean" />
         <xsd:element name="maxCost" type="xsd:double" />
      </xsd:sequence>
   </xsd:complexType>

   <xsd:simpleType name="tBoolean">
      <xsd:restriction base="xsd:string">
         <xsd:enumeration value="yes"/>
         <xsd:enumeration value="no"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:schema>
```

Listing A.2: XML-Schema der Kontext-Elemente für BPEL-Spracherweiterungen

Das folgende Listing A.3 zeigt das XML-Schema für die BPEL-Spracherweiterung `<mobileProcessData>`, die im Rahmen der übertragbaren Sub-Prozesse als Message Type für die Verwendung mit WSDL konzipiert wurde.

```
<xsd:schema
   targetNamespace="http://informatik.haw-hamburg.de/master/
      wsbpel/2.0/mobileProcessData"
   xmlns="http://informatik.haw-hamburg.de/master/wsbpel/2.0/
      mobileProcessData"
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" blockDefault="#all">

   <!-- import other namespaces -->
   <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
      schemaLocation="http://www.w3.org/2001/xml.xsd" />
   <xsd:import namespace="http://docs.oasis-open.org/wsbpel/2.0/
      process/executable"
      schemaLocation="ws-bpel_executable.xsd" />
   <xsd:import namespace="http://schemas.xmlsoap.org/wsdl/"
      schemaLocation="wsdl-2003-02-11.xsd" />

   <!-- base types for extensible elements -->
   <!-- see namespace
      http://docs.oasis-open.org/wsbpel/2.0/process/executable
      -->
   <xsd:complexType name="tExtensibleElements">
      <xsd:sequence>
         <xsd:element ref="documentation" minOccurs="0"
            maxOccurs="unbounded" />
         <xsd:any namespace="##other" processContents="lax"
            minOccurs="0" maxOccurs="unbounded" />
      </xsd:sequence>
      <xsd:anyAttribute namespace="##other"
         processContents="lax" />
   </xsd:complexType>

   <xsd:complexType name="tExtensibleMixedNamespaceElements">
      <xsd:sequence>
         <xsd:element name="documentation" type="tDocumentation"
            minOccurs="0" maxOccurs="unbounded" />
         <xsd:element name="extensions" type="tExtensions"
            minOccurs="0" />
```

```
    </xsd:sequence>
    <xsd:anyAttribute namespace="##other"
       processContents="lax" />
</xsd:complexType>

<xsd:element name="documentation" type="tDocumentation" />
<xsd:complexType name="tDocumentation" mixed="true">
    <xsd:sequence>
       <xsd:any processContents="lax" minOccurs="0"
          maxOccurs="unbounded" />
    </xsd:sequence>
    <xsd:attribute name="source" type="xsd:anyURI" />
    <xsd:attribute ref="xml:lang" />
</xsd:complexType>

<xsd:complexType name="tExtensions">
    <xsd:sequence>
       <xsd:any namespace="##other" processContents="lax"
          minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
</xsd:complexType>


<!-- element "mobileProcessData" to be used as WSDL message
    type (within web service interaction by mobile extension
    activities "sendProcess" and "receiveProcess") -->
<xsd:element name="mobileProcessData"
    type="tMobileProcessData" />
<xsd:complexType name="tMobileProcessData">
    <xsd:complexContent>
       <xsd:extension base="tExtensibleMixedNamespaceElements">
          <xsd:sequence>
             <xsd:element ref="processBpel" minOccurs="1" />
             <xsd:element ref="processWsdl" minOccurs="1" />
             <xsd:element ref="processStartParameters"
                minOccurs="1" />
             <xsd:any namespace="##other"
                processContents="lax" minOccurs="0"
                maxOccurs="unbounded" />
          </xsd:sequence>
       </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>

<xsd:element name="processBpel" type="tProcessBpel" />
<xsd:complexType name="tProcessBpel">
```

```
      <xsd:complexContent>
         <xsd:extension base="tExtensibleElements">
            <xsd:sequence>
               <xsd:element ref="bpel:process" minOccurs="1" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:element name="processWsdl" type="tProcessWsdl" />
   <xsd:complexType name="tProcessWsdl">
      <xsd:complexContent>
         <xsd:extension base="tExtensibleElements">
            <xsd:sequence>
               <xsd:element ref="wsdl:definitions" minOccurs="1"
                  />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>

   <xsd:element name="processStartParameters"
      type="tProcessStartParameters" />
   <xsd:complexType name="tProcessStartParameters">
      <xsd:complexContent>
         <xsd:extension base="tExtensibleMixedNamespaceElements">
            <xsd:sequence>
               <xsd:element name="partnerLink" type="xsd:string"
                  minOccurs="1" />
               <xsd:element name="portType" type="xsd:string"
                  minOccurs="0" maxOccurs="1" />
               <xsd:element name="operation" type="xsd:string"
                  minOccurs="1" />
               <xsd:element name="inputVariable"
                  type="wsdl:tMessage" minOccurs="1" />
               <xsd:any namespace="##other"
                  processContents="lax" minOccurs="0"
                  maxOccurs="unbounded" />
            </xsd:sequence>
         </xsd:extension>
      </xsd:complexContent>
   </xsd:complexType>
</xsd:schema>
```

Listing A.3: XML-Schema der BPEL-Spracherweiterung für WSDL-Einsatz

Anhang B

BPEL-Prozessdefinitionen mit mobilen Spracherweiterungen

Anhang B enthält exemplarisch die zwei wichtigsten BPEL-Prozessdefinitionen, die für das Szenario "Auto-Pannendienst" realisiert wurden. Die Basis bilden jeweils ausführbare WS-BPEL 2.0-Prozesse, ergänzt um die im Rahmen dieser Arbeit entworfenen Spracherweiterungen zur Unterstützung mobiler Geschäftsprozesse. Die beiden Listings B.1 und B.2 geben die Prozessdefinitionen für den `CarRepairService` und den `MotorMechanicService` wieder.

Das folgende Listing B.1 enthält die BPEL-Prozessdefinition für den `CarRepairService`. Darin enthalten ist zudem die BPEL-Definition für den zu übertragenen Sub-Prozess `MMSubProcessService` (Zeile 154-486) und dessen WSDL-Dokument (Zeile 490-576).

```
1  <bpel:process
2     name="CarRepairService"
3     xmlns:as="http://informatik.haw-hamburg.de/master/wsdl/
4        AccountingService"
5     xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
6        executable"
7     xmlns:context="http://informatik.haw-hamburg.de/master/wsbpel/
8        2.0/mobileContextData"
9     xmlns:crs="http://informatik.haw-hamburg.de/master/wsdl/
10       CarRepairService"
11    xmlns:ext="http://www.activebpel.org/2006/09/bpel/extension/
12       query_handling"
13    xmlns:mds="http://informatik.haw-hamburg.de/master/wsdl/
14       MasterDataService"
```

```
15    xmlns:mms="http://informatik.haw-hamburg.de/master/wsdl/
16       MotorMechanicService"
17    xmlns:mmsps="http://informatik.haw-hamburg.de/master/wsdl/
18       MMSubProcessService"
19    xmlns:mobile="http://informatik.haw-hamburg.de/master/wsbpel/
20       2.0/mobileExtensions"
21    xmlns:ns="http://informatik.haw-hamburg.de/master/wsbpel/2.0/
22       mobileProcessData"
23    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
24    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
25    suppressJoinFailure="yes"
26    targetNamespace="http://informatik.haw-hamburg.de/master/wsdl/
27       CarRepairService">
28    <bpel:extensions>
29       <bpel:extension mustUnderstand="yes"
30          namespace="http://informatik.haw-hamburg.de/master/
31             wsbpel/2.0/mobileExtensions" />
32       <bpel:extension mustUnderstand="yes"
33          namespace="http://www.activebpel.org/2006/09/bpel/
34             extension/query_handling" />
35    </bpel:extensions>
36    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
37       location="../wsdl/CarRepairService.wsdl"
38       namespace="http://informatik.haw-hamburg.de/master/wsdl/
39          CarRepairService" />
40    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
41       location="../wsdl/MasterDataService.wsdl"
42       namespace="http://informatik.haw-hamburg.de/master/wsdl/
43          MasterDataService" />
44    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
45       location="../wsdl/MotorMechanicService.wsdl"
46       namespace="http://informatik.haw-hamburg.de/master/wsdl/
47          MotorMechanicService" />
48    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
49       location="../wsdl/AccountingService.wsdl"
50       namespace="http://informatik.haw-hamburg.de/master/wsdl/
51          AccountingService" />
52    <bpel:import importType="http://www.w3.org/2001/XMLSchema"
53       location="../mobile-bpel-extensions/mobileContextData.xsd"
54       namespace="http://informatik.haw-hamburg.de/master/wsbpel/
55          2.0/mobileContextData" />
56    <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
57       location="../wsdl/MMSubProcessService.wsdl"
58       namespace="http://informatik.haw-hamburg.de/master/wsdl/
59          MMSubProcessService" />
60    <bpel:partnerLinks>
```

```
61      <bpel:partnerLink myRole="carRepairService"
62         name="CarRepairServicePL"
63         partnerLinkType="crs:carRepairRequestingPLT" />
64      <bpel:partnerLink name="MasterDataRequestingPL"
65         partnerLinkType="mds:masterDataRequestingPLT"
66         partnerRole="masterDataService" />
67      <bpel:partnerLink myRole="motorMechanicServiceRequester"
68         name="MotorMechanicRequestingPL"
69         partnerLinkType="mms:motorMechanicRequestingPLT"
70         partnerRole="motorMechanicService" />
71      <bpel:partnerLink name="AccountingRequestingPL"
72         partnerLinkType="as:accountingRequestingPLT"
73         partnerRole="accountingService" />
74      <bpel:partnerLink name="MMSubProcessRespondingPL"
75         myRole="mMSubProcessServiceResponseReceiver"
76         partnerLinkType="crs:mMSubProcessRespondingPLT" />
77   </bpel:partnerLinks>
78   <bpel:variables>
79      <bpel:variable
80         messageType="crs:enterBreakdownNotificationRequest"
81         name="breakdownNotificationInput" />
82      <bpel:variable
83         messageType="mds:completeCustomerDataRequest"
84         name="customerDataToComplete" />
85      <bpel:variable
86         messageType="mds:completeCustomerDataResponse"
87         name="customerDataCompleted" />
88      <bpel:variable
89         messageType="mms:placeBreakdownOrderRequest"
90         name=
91         "mobileProcessDataWithBreakdownOrderForMotorMechanic" />
92      <bpel:variable
93         messageType="crs:returnOperationalReportResponse"
94         name="operationalReportFromMotorMechanic" />
95      <bpel:variable
96         messageType="as:closeBreakdownOrderRequest"
97         name="breakdownOrderToClose" />
98      <bpel:variable name="cancelledBreakdownId"
99         type="xsd:integer" />
100     <bpel:variable name="receivedOperationalReport"
101        type="xsd:boolean">
102        <bpel:from>false()</bpel:from>
103     </bpel:variable>
104     <bpel:variable element="context:motorMechanic"
105        name="motorMechanicContext" />
106     <bpel:variable element="bpel:process"
```

```
107         name="subProcessBpel" />
108      <bpel:variable element="wsdl:definitions"
109         name="subProcessWsdl" />
110      <bpel:variable
111         messageType="mmsps:startBreakdownOrderRequest"
112         name="subProcessInputVariable" />
113   </bpel:variables>
114   <bpel:correlationSets>
115      <bpel:correlationSet name="CarRepairAndMotorMechanicCS"
116         properties="crs:carBreakdownId" />
117   </bpel:correlationSets>
118   <bpel:sequence name="HandleCarBreakdownNotificationSequence">
119      <bpel:receive createInstance="yes"
120         name="ReceiveBreakdownNotification"
121         operation="enterBreakdownNotification"
122         partnerLink="CarRepairServicePL"
123         portType="crs:CarRepairServicePT"
124         variable="breakdownNotificationInput" />
125      <bpel:assign name="AssignCopyCustomerData">
126         <bpel:copy ignoreMissingFromData="yes">
127            <bpel:from part="carBreakdown"
128               variable="breakdownNotificationInput">
129               <bpel:query>customer</bpel:query>
130            </bpel:from>
131            <bpel:to part="customerDataInput"
132               variable="customerDataToComplete" />
133         </bpel:copy>
134      </bpel:assign>
135      <bpel:invoke inputVariable="customerDataToComplete"
136         name="InvokeCompleteCustomerData"
137         operation="completeCustomerData"
138         outputVariable="customerDataCompleted"
139         partnerLink="MasterDataRequestingPL"
140         portType="mds:MasterDataServicePT" />
141      <bpel:assign name="AssignCopyCarBreakdownIdForMM">
142         <bpel:copy ignoreMissingFromData="yes">
143            <bpel:from part="carBreakdown"
144               variable="breakdownNotificationInput">
145               <bpel:query>carBreakdownId</bpel:query>
146            </bpel:from>
147            <bpel:to part="carBreakdownId"
148               variable=
149         "mobileProcessDataWithBreakdownOrderForMotorMechanic" />
150         </bpel:copy>
151      </bpel:assign>
152      <bpel:assign name="PrepareSubProcessVariables">
```

```
153         <bpel:copy ignoreMissingFromData="yes">
154               <bpel:from>string('&lt;bpel:process&#13;
155   name="MMSubProcessService" &#13;
156   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
157      executable"&#13;
158   xmlns:context="http://informatik.haw-hamburg.de/master/wsbpel/
159      2.0/mobileContextData"&#13;
160   xmlns:crs="http://informatik.haw-hamburg.de/master/wsdl/
161      CarRepairService"&#13;
162   xmlns:ext="http://www.activebpel.org/2006/09/bpel/extension/
163      query_handling"&#13;
164   xmlns:mmas="http://informatik.haw-hamburg.de/master/wsdl/
165      MMApplicationService"&#13;
166   xmlns:mmds="http://informatik.haw-hamburg.de/master/wsdl/
167      MMDiagnosticService"&#13;
168   xmlns:mmns="http://informatik.haw-hamburg.de/master/wsdl/
169      MMNavigationService"&#13;
170   xmlns:mmsps="http://informatik.haw-hamburg.de/master/wsdl/
171      MMSubProcessService"&#13;
172   xmlns:mobile="http://informatik.haw-hamburg.de/master/wsbpel/
173      2.0/mobileExtensions"&#13;
174   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
175      &#13;
176   ext:createTargetXPath="yes"&#13;
177   suppressJoinFailure="yes"&#13;
178   targetNamespace="http://informatik.haw-hamburg.de/master/bpel/
179      MMSubProcessService"&gt;&#13;
180   &lt;bpel:extensions&gt;&#13;
181      &lt;bpel:extension mustUnderstand="yes"&#13;
182         namespace="http://informatik.haw-hamburg.de/master/
183            wsbpel/2.0/mobileExtensions" /&gt;&#13;
184      &lt;bpel:extension mustUnderstand="yes"&#13;
185         namespace="http://www.activebpel.org/2006/09/bpel/
186            extension/query_handling" /&gt;&#13;
187   &lt;/bpel:extensions&gt;&#13;
188   &lt;bpel:import
189      importType="http://schemas.xmlsoap.org/wsdl/"&#13;
190      location="../wsdl/MMSubProcessService.wsdl"&#13;
191      namespace="http://informatik.haw-hamburg.de/master/wsdl/
192         MMSubProcessService" /&gt;&#13;
193   &lt;bpel:import
194      importType="http://schemas.xmlsoap.org/wsdl/"&#13;
195      location="../wsdl/MMNavigationService.wsdl"&#13;
196      namespace="http://informatik.haw-hamburg.de/master/wsdl/
197         MMNavigationService" /&gt;&#13;
198   &lt;bpel:import
```

```
199       importType="http://schemas.xmlsoap.org/wsdl/"&#13;
200       location="../wsdl/MMDiagnosticService.wsdl"&#13;
201       namespace="http://informatik.haw-hamburg.de/master/wsdl/
202          MMDiagnosticService" /&gt;&#13;
203    &lt;bpel:import
204       importType="http://schemas.xmlsoap.org/wsdl/"&#13;
205       location="../wsdl/MMApplicationService.wsdl"&#13;
206       namespace="http://informatik.haw-hamburg.de/master/wsdl/
207          MMApplicationService" /&gt;&#13;
208    &lt;bpel:import
209       importType="http://schemas.xmlsoap.org/wsdl/"&#13;
210       location="../wsdl/CarRepairService.wsdl"&#13;
211       namespace="http://informatik.haw-hamburg.de/master/wsdl/
212          CarRepairService" /&gt;&#13;
213    &lt;bpel:import
214       importType="http://www.w3.org/2001/XMLSchema"&#13;
215       location="../mobile-bpel-extensions/mobileContextData.xsd"
216          &#13;
217       namespace="http://informatik.haw-hamburg.de/master/wsbpel/
218          2.0/mobileContextData" /&gt;&#13;
219    &lt;bpel:partnerLinks&gt;&#13;
220       &lt;bpel:partnerLink myRole="mMSubProcessService"&#13;
221          name="MMSubProcessRequestingPL"&#13;
222          partnerLinkType="mmsps:mMSubProcessRequestingPLT" /&gt;
223          &#13;
224       &lt;bpel:partnerLink name="MMNavigationRequestingPL"&#13;
225          partnerLinkType="mmns:mmNavigationRequestingPLT"&#13;
226          partnerRole="mmNavigationService" /&gt;&#13;
227       &lt;bpel:partnerLink name="MMDiagnosticRequestingPL"&#13;
228          partnerLinkType="mmds:mmDiagnosticRequestingPLT"&#13;
229          partnerRole="mmDiagnosticService" /&gt;&#13;
230       &lt;bpel:partnerLink name="MMApplicationRequestingPL"&#13;
231          myRole="mmApplicationServiceRequester"&#13;
232          partnerLinkType="mmas:mmApplicationRequestingPLT"&#13;
233          partnerRole="mmApplicationService" /&gt;&#13;
234       &lt;bpel:partnerLink name="MMSubProcessRespondingPL"&#13;
235          partnerLinkType="crs:mMSubProcessRespondingPLT"&#13;
236          partnerRole="mMSubProcessServiceResponseReceiver" /&gt;
237          &#13;
238    &lt;/bpel:partnerLinks&gt;&#13;
239    &lt;bpel:variables&gt;&#13;
240       &lt;bpel:variable
241          messageType="mmsps:startBreakdownOrderRequest"&#13;
242          name="breakdownOrderInput" /&gt;&#13;
243       &lt;bpel:variable
244          messageType="crs:returnOperationalReportResponse"&#13;
```

```
245          name="operationalReportToReturn" /&gt;&#13;
246       &lt;bpel:variable
247          messageType="mmns:startNavigationRequest"&#13;
248          name="startNavigationRequest" /&gt;&#13;
249       &lt;bpel:variable
250          messageType="mmds:startDiagnosticToolRequest"&#13;
251          name="startDiagnosticToolRequest" /&gt;&#13;
252       &lt;bpel:variable
253          messageType="mmas:enterBreakdownOrderRequest"&#13;
254          name="breakdownOrderForMMApplication" /&gt;&#13;
255       &lt;bpel:variable
256          messageType="mmas:returnOperationalReportResponse"&#13;
257          name="operationalReportFromMMApplication" /&gt;&#13;
258       &lt;bpel:variable
259          element="context:gpsPositionDeviation"&#13;
260          name="distanceAndDestinationAddress" /&gt;&#13;
261    &lt;/bpel:variables&gt;&#13;
262    &lt;bpel:correlationSets&gt;&#13;
263       &lt;bpel:correlationSet
264          name="MotorMechanicAndMMApplicationCS"&#13;
265          properties="mmas:carBreakdownId" /&gt;&#13;
266    &lt;/bpel:correlationSets&gt;&#13;
267    &lt;bpel:sequence
268       name="HandleReceiveBreakdownOrderSequence"&gt;&#13;
269       &lt;bpel:receive createInstance="yes"
270          name="ReceiveBreakdownOrder"&#13;
271          operation="startBreakdownOrder"
272          partnerLink="MMSubProcessRequestingPL"&#13;
273          portType="mmsps:MMSubProcessServicePT"
274          variable="breakdownOrderInput" /&gt;&#13;
275       &lt;bpel:assign name="AssignCopyBreakdownOrder"&gt;&#13;
276          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
277             &lt;bpel:from part="carBreakdown"
278                variable="breakdownOrderInput" /&gt;&#13;
279             &lt;bpel:to part="carBreakdown"&#13;
280                variable="breakdownOrderForMMApplication" /&gt;
281                &#13;
282          &lt;/bpel:copy&gt;&#13;
283       &lt;/bpel:assign&gt;&#13;
284       &lt;bpel:invoke
285          inputVariable="breakdownOrderForMMApplication"&#13;
286          name="InvokeSendBreakdownOrderToMMApplication"&#13;
287          operation="enterBreakdownOrder"
288          partnerLink="MMApplicationRequestingPL"&#13;
289          portType="mmas:MMApplicationServicePT"&gt;&#13;
290          &lt;bpel:correlations&gt;&#13;
```

```
291            &lt;bpel:correlation initiate="yes"
292               pattern="request"&#13;
293               set="MotorMechanicAndMMApplicationCS" /&gt;&#13;
294         &lt;/bpel:correlations&gt;&#13;
295      &lt;/bpel:invoke&gt;&#13;
296      &lt;bpel:assign
297         name="AssignContextActivityRequirements"&gt;&#13;
298         &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
299            &lt;bpel:from part="carBreakdown"
300               variable="breakdownOrderInput"&gt;&#13;
301               &lt;bpel:query&gt;
302                  currentLocation/streetName
303               &lt;/bpel:query&gt;&#13;
304            &lt;/bpel:from&gt;&#13;
305            &lt;bpel:to
306               variable="distanceAndDestinationAddress"&gt;&#13;
307               &lt;bpel:query&gt;&#13;
308                  context:destination/context:streetName&#13;
309               &lt;/bpel:query&gt;&#13;
310            &lt;/bpel:to&gt;&#13;
311         &lt;/bpel:copy&gt;&#13;
312         &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
313            &lt;bpel:from part="carBreakdown"
314               variable="breakdownOrderInput"&gt;&#13;
315               &lt;bpel:query&gt;
316                  currentLocation/streetNo
317               &lt;/bpel:query&gt;&#13;
318            &lt;/bpel:from&gt;&#13;
319            &lt;bpel:to
320               variable="distanceAndDestinationAddress"&gt;&#13;
321               &lt;bpel:query&gt;&#13;
322                  context:destination/context:streetNo&#13;
323               &lt;/bpel:query&gt;&#13;
324            &lt;/bpel:to&gt;&#13;
325         &lt;/bpel:copy&gt;&#13;
326         &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
327            &lt;bpel:from part="carBreakdown"
328               variable="breakdownOrderInput"&gt;&#13;
329               &lt;bpel:query&gt;
330                  currentLocation/zipCode
331               &lt;/bpel:query&gt;&#13;
332            &lt;/bpel:from&gt;&#13;
333            &lt;bpel:to
334               variable="distanceAndDestinationAddress"&gt;&#13;
335               &lt;bpel:query&gt;
336                  context:destination/context:zipCode
```

```
337                &lt;/bpel:query&gt;&#13;
338             &lt;/bpel:to&gt;&#13;
339          &lt;/bpel:copy&gt;&#13;
340          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
341             &lt;bpel:from part="carBreakdown"
342                variable="breakdownOrderInput"&gt;&#13;
343                &lt;bpel:query&gt;
344                   currentLocation/city
345                &lt;/bpel:query&gt;&#13;
346             &lt;/bpel:from&gt;&#13;
347             &lt;bpel:to
348                variable="distanceAndDestinationAddress"&gt;&#13;
349                &lt;bpel:query&gt;
350                   context:destination/context:city
351                &lt;/bpel:query&gt;&#13;
352             &lt;/bpel:to&gt;&#13;
353          &lt;/bpel:copy&gt;&#13;
354          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
355             &lt;bpel:from part="carBreakdown"
356                variable="breakdownOrderInput"&gt;&#13;
357                &lt;bpel:query&gt;
358                   currentLocation/country
359                &lt;/bpel:query&gt;&#13;
360             &lt;/bpel:from&gt;&#13;
361             &lt;bpel:to
362                variable="distanceAndDestinationAddress"&gt;&#13;
363                &lt;bpel:query&gt;
364                   context:destination/context:country
365                &lt;/bpel:query&gt;&#13;
366             &lt;/bpel:to&gt;&#13;
367          &lt;/bpel:copy&gt;&#13;
368          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
369             &lt;bpel:from&gt;number(0.2)&lt;/bpel:from&gt;&#13;
370             &lt;bpel:to
371                variable="distanceAndDestinationAddress"&gt;&#13;
372                &lt;bpel:query&gt;
373                   context:distanceGreaterThan
374                &lt;/bpel:query&gt;&#13;
375             &lt;/bpel:to&gt;&#13;
376          &lt;/bpel:copy&gt;&#13;
377       &lt;/bpel:assign&gt;&#13;
378       &lt;bpel:assign
379          name="AssignPrepareNavigationInput"&gt;&#13;
380          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
381             &lt;bpel:from part="carBreakdown"
382                variable="breakdownOrderInput"&gt;&#13;
```

```
383                &lt;bpel:query&gt;
384                   currentLocation
385                &lt;/bpel:query&gt;&#13;
386             &lt;/bpel:from&gt;&#13;
387             &lt;bpel:to part="address"
388                variable="startNavigationRequest" /&gt;&#13;
389          &lt;/bpel:copy&gt;&#13;
390       &lt;/bpel:assign&gt;&#13;
391       &lt;bpel:extensionActivity&gt;&#13;
392          &lt;mobile:contextActivity
393             name="HandleOptionalNavigation"&#13;
394             mode="optional"&gt;&#13;
395             &lt;mobile:requirements mode="now"&gt;&#13;
396                &lt;mobile:gps&gt;&#13;
397                   &lt;mobile:available&gt;
398                      "yes"
399                   &lt;/mobile:available&gt;&#13;
400                &lt;/mobile:gps&gt;&#13;
401                &lt;mobile:gpsPositionDeviation&#13;
402                   variable="distanceAndDestinationAddress" /&gt;
403                      &#13;
404             &lt;/mobile:requirements&gt;&#13;
405             &lt;bpel:invoke
406                inputVariable="startNavigationRequest"&#13;
407                name="InvokeStartNavigation"
408                operation="startNavigation"&#13;
409                partnerLink="MMNavigationRequestingPL"&#13;
410                portType="mmns:MMNavigationServicePT" /&gt;&#13;
411          &lt;/mobile:contextActivity&gt;&#13;
412       &lt;/bpel:extensionActivity&gt;&#13;
413       &lt;bpel:assign
414          name="AssignPrepareDiagnosticInput"&gt;&#13;
415          &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
416             &lt;bpel:from part="carBreakdown"
417                variable="breakdownOrderInput"&gt;&#13;
418                &lt;bpel:query&gt;car&lt;/bpel:query&gt;&#13;
419             &lt;/bpel:from&gt;&#13;
420             &lt;bpel:to part="car"
421                variable="startDiagnosticToolRequest" /&gt;&#13;
422          &lt;/bpel:copy&gt;&#13;
423       &lt;/bpel:assign&gt;&#13;
424       &lt;bpel:extensionActivity&gt;&#13;
425          &lt;mobile:contextActivity name="HandleDiagnosticTool"
426             mode="must"&gt;&#13;
427             &lt;mobile:requirements mode="onEvent"&gt;&#13;
428                &lt;mobile:carConnection&gt;&#13;
```

```
429                  &lt;mobile:available&gt;
430                     "yes"
431                  &lt;/mobile:available&gt;&#13;
432               &lt;/mobile:carConnection&gt;&#13;
433            &lt;/mobile:requirements&gt;&#13;
434            &lt;bpel:invoke
435               inputVariable="startDiagnosticToolRequest"&#13;
436               name="InvokeStartDiagnosticTool"
437               operation="startDiagnosticTool"&#13;
438               partnerLink="MMDiagnosticRequestingPL"&#13;
439               portType="mmds:MMDiagnosticServicePT" /&gt;&#13;
440         &lt;/mobile:contextActivity&gt;&#13;
441      &lt;/bpel:extensionActivity&gt;&#13;
442      &lt;bpel:receive
443         name="ReceiveOperationalReportFromMMApplication"&#13;
444         operation="returnOperationalReport"
445         partnerLink="MMApplicationRequestingPL"&#13;
446         portType="mmas:MMApplicationServiceCallbackPT"&#13;
447         variable="operationalReportFromMMApplication"&gt;&#13;
448         &lt;bpel:correlations&gt;&#13;
449            &lt;bpel:correlation initiate="no"&#13;
450               set="MotorMechanicAndMMApplicationCS" /&gt;&#13;
451         &lt;/bpel:correlations&gt;&#13;
452      &lt;/bpel:receive&gt;&#13;
453      &lt;bpel:assign name="AssignCopyOperationalReport"&gt;&#13;
454         &lt;bpel:copy ignoreMissingFromData="yes"&gt;&#13;
455            &lt;bpel:from part="operationalReport"&#13;
456               variable="operationalReportFromMMApplication"
457                  /&gt;&#13;
458            &lt;bpel:to part="operationalReport"&#13;
459               variable="operationalReportToReturn" /&gt;&#13;
460         &lt;/bpel:copy&gt;&#13;
461      &lt;/bpel:assign&gt;&#13;
462      &lt;bpel:extensionActivity&gt;&#13;
463         &lt;mobile:contextActivity
464            name="HandleReturnOperationalReport"&#13;
465            mode="must"&gt;&#13;
466            &lt;mobile:requirements mode="onEvent"&gt;&#13;
467               &lt;mobile:internet&gt;&#13;
468                  &lt;mobile:available&gt;
469                     "yes"
470                  &lt;/mobile:available&gt;&#13;
471                  &lt;mobile:maxCost&gt;
472                     0.0
473                  &lt;/mobile:maxCost&gt;&#13;
474               &lt;/mobile:internet&gt;&#13;
```

```
475            &lt;/mobile:requirements&gt;&#13;
476            &lt;bpel:invoke
477               inputVariable="operationalReportToReturn"&#13;
478               name="InvokeReturnOperationalReport"
479               operation="returnOperationalReport"&#13;
480               partnerLink="MMSubProcessRespondingPL"&#13;
481               portType="crs:MMSubProcessServiceCallbackPT" /&gt;
482                  &#13;
483         &lt;/mobile:contextActivity&gt;&#13;
484      &lt;/bpel:extensionActivity&gt;&#13;
485   &lt;/bpel:sequence&gt;&#13;
486&lt;/bpel:process&gt;')</bpel:from>
487            <bpel:to variable="subProcessBpel" />
488         </bpel:copy>
489         <bpel:copy ignoreMissingFromData="yes">
490            <bpel:from>string('&lt;wsdl:definitions&#13;
491   name="MMSubProcessService"&#13;
492   targetNamespace="http://informatik.haw-hamburg.de/master/wsdl/
493      MMSubProcessService"&#13;
494   xmlns:crs="http://informatik.haw-hamburg.de/master/wsdl/
495      CarRepairService"&#13;
496   xmlns:tns="http://informatik.haw-hamburg.de/master/wsdl/
497      MMSubProcessService"&#13;
498   xmlns:cbdt="http://informatik.haw-hamburg.de/master/xsd/
499      CarBreakdownDataTypes"&#13;
500   xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
501      &#13;
502   xmlns:xsd="http://www.w3.org/2001/XMLSchema"&#13;
503   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"&#13;
504   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"&gt;&#13;
505   &#13;
506   &lt;wsdl:types&gt;&#13;
507      &lt;xsd:schema elementFormDefault="qualified"&#13;
508         targetNamespace="http://informatik.haw-hamburg.de/
509            master/wsdl/MMSubProcessService"&#13;
510         xmlns:xsd="http://www.w3.org/2001/XMLSchema"&gt;&#13;
511         &lt;xsd:import&#13;
512            namespace="http://informatik.haw-hamburg.de/master/
513               xsd/CarBreakdownDataTypes"&#13;
514            schemaLocation="CarBreakdownDataTypes.xsd" /&gt;&#13;
515      &lt;/xsd:schema&gt;&#13;
516   &lt;/wsdl:types&gt;&#13;
517   &#13;
518   &lt;wsdl:message name="startBreakdownOrderRequest"&gt;&#13;
519      &lt;wsdl:part name="carBreakdown"
520         type="cbdt:CarBreakdown" /&gt;&#13;
```

```
521   &lt;/wsdl:message&gt;&#13;
522   &#13;
523   &lt;wsdl:portType name="MMSubProcessServicePT"&gt;&#13;
524      &lt;wsdl:operation name="startBreakdownOrder"&gt;&#13;
525         &lt;wsdl:input name="startBreakdownOrderInput"&#13;
526            message="tns:startBreakdownOrderRequest" /&gt;&#13;
527      &lt;/wsdl:operation&gt;&#13;
528   &lt;/wsdl:portType&gt;&#13;
529   &#13;
530   &lt;wsdl:binding name="MMSubProcessServiceSOAPBinding"&#13;
531      type="tns:MMSubProcessServicePT"&gt;&#13;
532      &lt;soap:binding style="document"&#13;
533         transport="http://schemas.xmlsoap.org/soap/http"&#13;
534         xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" /&gt;
535            &#13;
536      &lt;wsdl:operation name="startBreakdownOrder"&gt;&#13;
537         &lt;soap:operation soapAction=""&#13;
538            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
539            /&gt;&#13;
540         &lt;wsdl:input name="startBreakdownOrderInput"&gt;&#13;
541            &lt;soap:body&#13;
542               encodingStyle="http://schemas.xmlsoap.org/soap/
543                  encoding/"&#13;
544               namespace="http://informatik.haw-hamburg.de/
545                  master/wsdl/MMSubProcessService"&#13;
546               use="encoded"
547               xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
548               /&gt;&#13;
549         &lt;/wsdl:input&gt;&#13;
550      &lt;/wsdl:operation&gt;&#13;
551   &lt;/wsdl:binding&gt;&#13;
552   &#13;
553   &lt;wsdl:service name="MMSubProcessService"&gt;&#13;
554      &lt;wsdl:documentation
555         xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"&gt;&#13;
556         Motor Mechanic Sub-Process Service&#13;
557      &lt;/wsdl:documentation&gt;&#13;
558      &lt;wsdl:port name="MMSubProcessServiceSOAPPort"&#13;
559         binding="tns:MMSubProcessServiceSOAPBinding"&gt;&#13;
560         &lt;soap:address&#13;
561            location="http://localhost:8080/active-bpel/services/
562               MMSubProcessService"&#13;
563            xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
564            /&gt;&#13;
565      &lt;/wsdl:port&gt;&#13;
566   &lt;/wsdl:service&gt;&#13;
```

```
567    &#13;
568    &lt;plnk:partnerLinkType&#13;
569       xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
570          &#13;
571       name="mMSubProcessRequestingPLT"&gt;&#13;
572       &lt;plnk:role name="mMSubProcessService"&#13;
573          portType="tns:MMSubProcessServicePT" /&gt;&#13;
574    &lt;/plnk:partnerLinkType&gt;&#13;
575    &#13;
576 &lt;/wsdl:definitions&gt;&#13;')</bpel:from>
577             <bpel:to variable="subProcessWsdl" />
578          </bpel:copy>
579          <bpel:copy ignoreMissingFromData="yes">
580             <bpel:from part="carBreakdown"
581                variable="breakdownNotificationInput" />
582             <bpel:to part="carBreakdown"
583                variable="subProcessInputVariable" />
584          </bpel:copy>
585          <bpel:copy ignoreMissingFromData="yes">
586             <bpel:from part="customerDataOutput"
587                variable="customerDataCompleted" />
588             <bpel:to part="carBreakdown"
589                variable="subProcessInputVariable">
590                <bpel:query>customer</bpel:query>
591             </bpel:to>
592          </bpel:copy>
593       </bpel:assign>
594       <bpel:assign name="AssignSubProcessStartParameters">
595          <bpel:copy ignoreMissingFromData="yes">
596             <bpel:from variable="subProcessBpel" />
597             <bpel:to part="mobileProcessData"
598                variable=
599          "mobileProcessDataWithBreakdownOrderForMotorMechanic">
600                <bpel:query>
601                   ns:processBpel/bpel:process
602                </bpel:query>
603             </bpel:to>
604          </bpel:copy>
605          <bpel:copy ignoreMissingFromData="yes">
606             <bpel:from variable="subProcessWsdl" />
607             <bpel:to part="mobileProcessData"
608                variable=
609          "mobileProcessDataWithBreakdownOrderForMotorMechanic">
610                <bpel:query>
611                   ns:processWsdl/wsdl:definitions
612                </bpel:query>
```

```
613             </bpel:to>
614          </bpel:copy>
615          <bpel:copy ignoreMissingFromData="yes">
616             <bpel:from variable="subProcessInputVariable" />
617             <bpel:to part="mobileProcessData"
618                variable=
619          "mobileProcessDataWithBreakdownOrderForMotorMechanic">
620                <bpel:query>
621                   ns:processStartParameters/ns:inputVariable
622                </bpel:query>
623             </bpel:to>
624          </bpel:copy>
625       </bpel:assign>
626       <bpel:repeatUntil name=
627       "RepeatUntilReceivedAnOperationalReportFromMotorMechanic">
628          <bpel:condition>
629             $receivedOperationalReport = true()
630          </bpel:condition>
631          <bpel:sequence name="PlaceOrderAtMotorMechanicSequence">
632             <bpel:assign name="AssignDynamicInvokeSelect">
633                <bpel:copy ignoreMissingFromData="yes">
634                   <bpel:from part="carBreakdown"
635                      variable="breakdownNotificationInput">
636                      <bpel:query>car</bpel:query>
637                   </bpel:from>
638                   <bpel:to variable="motorMechanicContext">
639                      <bpel:query>context:car</bpel:query>
640                   </bpel:to>
641                </bpel:copy>
642                <bpel:copy ignoreMissingFromData="yes">
643                   <bpel:from part="carBreakdown"
644                      variable="breakdownNotificationInput">
645                      <bpel:query>
646                         breakdownDescription
647                      </bpel:query>
648                   </bpel:from>
649                   <bpel:to variable="motorMechanicContext">
650                      <bpel:query>
651                         context:breakdownDescription
652                      </bpel:query>
653                   </bpel:to>
654                </bpel:copy>
655                <bpel:copy ignoreMissingFromData="yes">
656                   <bpel:from>number(1.0)</bpel:from>
657                   <bpel:to variable="motorMechanicContext">
658                      <bpel:query>context:duration</bpel:query>
```

```
659                  </bpel:to>
660               </bpel:copy>
661               <bpel:copy ignoreMissingFromData="yes">
662                  <bpel:from part="customerDataOutput"
663                     variable="customerDataCompleted">
664                     <bpel:query>address</bpel:query>
665                  </bpel:from>
666                  <bpel:to variable="motorMechanicContext">
667                     <bpel:query>context:destination</bpel:query>
668                  </bpel:to>
669               </bpel:copy>
670            </bpel:assign>
671            <bpel:extensionActivity>
672               <mobile:sendProcess
673                  name="SendSubProcessToMotorMechanic">
674                  <mobile:dynamicInvoke
675                     inputVariable="
676            mobileProcessDataWithBreakdownOrderForMotorMechanic"
677                     name="InvokePlaceOrderAtMotorMechanic"
678                     operation="placeBreakdownOrder"
679                     partnerLink="MotorMechanicRequestingPL"
680                     portType="mms:MotorMechanicServicePT">
681                     <bpel:correlations>
682                        <bpel:correlation initiate="join"
683                           pattern="request"
684                           set="CarRepairAndMotorMechanicCS" />
685                     </bpel:correlations>
686                     <mobile:select>
687                        <mobile:motorMechanic
688                           variable="motorMechanicContext" />
689                     </mobile:select>
690                  </mobile:dynamicInvoke>
691                  <mobile:mobileProcessData>
692                     <processBpel variable="subProcessBpel" />
693                     <processWsdl variable="subProcessWsdl" />
694                     <processStartParameters
695                        partnerLink="MMSubProcessRequestingPL"
696                        portType="MMSubProcessServicePT"
697                        operation="startBreakdownOrder"
698                        inputVariable="subProcessInputVariable"
699                        />
700                  </mobile:mobileProcessData>
701               </mobile:sendProcess>
702            </bpel:extensionActivity>
703            <bpel:pick>
704               <bpel:onMessage
```

```
705                  operation="returnOperationalReport"
706                  partnerLink="MMSubProcessRespondingPL"
707                  portType="crs:MMSubProcessServiceCallbackPT"
708                  variable="operationalReportFromMotorMechanic">
709                  <bpel:correlations>
710                     <bpel:correlation initiate="no"
711                        set="CarRepairAndMotorMechanicCS" />
712                  </bpel:correlations>
713                  <bpel:assign name=
714                     "AssignTrueToReceivedOperationalReport">
715                     <bpel:copy>
716                        <bpel:from>true()</bpel:from>
717                        <bpel:to
718                           variable="receivedOperationalReport"
719                           />
720                     </bpel:copy>
721                  </bpel:assign>
722               </bpel:onMessage>
723               <bpel:onMessage operation="cancelBreakdownOrder"
724                  partnerLink="MotorMechanicRequestingPL"
725                  portType="mms:MotorMechanicServiceCallbackPT">
726                  <bpel:correlations>
727                     <bpel:correlation initiate="no"
728                        set="CarRepairAndMotorMechanicCS" />
729                  </bpel:correlations>
730                  <bpel:fromParts>
731                     <bpel:fromPart part="carBreakdownId"
732                        toVariable="cancelledBreakdownId" />
733                  </bpel:fromParts>
734                  <bpel:empty />
735               </bpel:onMessage>
736               <bpel:onAlarm>
737                  <bpel:for>"PT1M0S"</bpel:for>
738                  <bpel:empty />
739               </bpel:onAlarm>
740            </bpel:pick>
741         </bpel:sequence>
742      </bpel:repeatUntil>
743      <bpel:assign name="AssignCopyOperationalReport">
744         <bpel:copy ignoreMissingFromData="yes">
745            <bpel:from part="carBreakdown"
746               variable="subProcessInputVariable" />
747            <bpel:to part="carBreakdown"
748               variable="breakdownOrderToClose" />
749         </bpel:copy>
750         <bpel:copy>
```

```
751             <bpel:from part="operationalReport"
752                 variable="operationalReportFromMotorMechanic" />
753             <bpel:to part="operationalReport"
754                 variable="breakdownOrderToClose" />
755         </bpel:copy>
756       </bpel:assign>
757       <bpel:invoke inputVariable="breakdownOrderToClose"
758         name="InvokeCloseBreakdownOrder"
759         operation="closeBreakdownOrder"
760         partnerLink="AccountingRequestingPL"
761         portType="as:AccountingServicePT" />
762   </bpel:sequence>
763</bpel:process>
```

Listing B.1: BPEL-Prozessdefinition `CarRepairService`

Das folgende Listing B.2 enthält die BPEL-Prozessdefinition für den `MotorMechanicService`.

```
<bpel:process
   name="MotorMechanicService"
   xmlns:bpel="http://docs.oasis-open.org/wsbpel/2.0/process/
      executable"
   xmlns:context="http://informatik.haw-hamburg.de/master/wsbpel/
      2.0/mobileContextData"
   xmlns:ext="http://www.activebpel.org/2006/09/bpel/extension/
      query_handling"
   xmlns:mms="http://informatik.haw-hamburg.de/master/wsdl/
      MotorMechanicService"
   xmlns:mobile="http://informatik.haw-hamburg.de/master/wsbpel/
      2.0/mobileExtensions"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   ext:createTargetXPath="yes"
   suppressJoinFailure="yes"
   targetNamespace="http://informatik.haw-hamburg.de/master/bpel/
      MotorMechanicService">
   <bpel:extensions>
      <bpel:extension mustUnderstand="yes"
         namespace="http://informatik.haw-hamburg.de/master/
            wsbpel/2.0/mobileExtensions" />
      <bpel:extension mustUnderstand="yes"
         namespace="http://www.activebpel.org/2006/09/bpel/
            extension/query_handling" />
   </bpel:extensions>
   <bpel:import importType="http://schemas.xmlsoap.org/wsdl/"
      location="../wsdl/MotorMechanicService.wsdl"
      namespace="http://informatik.haw-hamburg.de/master/wsdl/
         MotorMechanicService" />
   <bpel:import importType="http://www.w3.org/2001/XMLSchema"
      location="../mobile-bpel-extensions/mobileContextData.xsd"
      namespace="http://informatik.haw-hamburg.de/master/wsbpel/
         2.0/mobileContextData" />
   <bpel:partnerLinks>
      <bpel:partnerLink myRole="motorMechanicService"
         name="MotorMechanicServicingPL"
         partnerLinkType="mms:motorMechanicRequestingPLT"
         partnerRole="motorMechanicServiceRequester" />
   </bpel:partnerLinks>
   <bpel:variables>
      <bpel:variable messageType="mms:placeBreakdownOrderRequest"
         name="breakdownOrderInput" />
```

```
43    </bpel:variables>
44    <bpel:sequence name="HandleReceiveBreakdownOrderSequence">
45       <bpel:extensionActivity>
46          <mobile:receiveProcess createInstance="yes"
47             name="ReceiveSubProcess"
48             operation="placeBreakdownOrder"
49             partnerLink="MotorMechanicServicingPL"
50             portType="mms:MotorMechanicServicePT"
51             variable="breakdownOrderInput" />
52       </bpel:extensionActivity>
53       <bpel:if name="IfAcceptOrCancelBreakdownOrder">
54          <bpel:condition>
55             $breakdownOrderInput.carBreakdownId &gt;= number(0)
56          </bpel:condition>
57          <bpel:sequence name="StartSubProcessSequence">
58             <bpel:extensionActivity>
59                <mobile:startProcess
60                   name="DeployAndStartSubProcess"
61                   variable="breakdownOrderInput" />
62             </bpel:extensionActivity>
63          </bpel:sequence>
64          <bpel:else>
65             <bpel:sequence name="CancelBreakdownOrderSequence">
66                <bpel:invoke name="InvokeReturnCancelOrder"
67                   operation="cancelBreakdownOrder"
68                   partnerLink="MotorMechanicServicingPL"
69                   portType="mms:MotorMechanicServiceCallbackPT">
70                   <bpel:toParts>
71                      <bpel:toPart
72                         fromVariable="breakdownOrderInput"
73                         part="carBreakdownId" />
74                   </bpel:toParts>
75                </bpel:invoke>
76             </bpel:sequence>
77          </bpel:else>
78       </bpel:if>
79    </bpel:sequence>
80 </bpel:process>
```

Listing B.2: BPEL-Prozessdefinition `MotorMechanicService`

Anhang C

CD-ROM

Inhalt der CD-ROM:

- Ordner *ActiveBPEL-Engine-4.0* enthält die Original-Distribution der ActiveBPEL Engine 4.0 von Active Endpoints als zip-File, inkl. Dokumentation und Installationshinweisen.
- Ordner *BPEL-Engine-Quellcode* enthält Sourcecode und Binaries für die erweiterte ActiveBPEL Engine als zip-File. Darin enthalten sind:
  - Ordner *dist* enthält Build-Ergebnis der erweiterten ActiveBPEL Engine.
  - Ordner *lib* enthält von der ActiveBPEL Engine benötigte Bibliotheken.
  - Ordner *projects* enthält Eclipse-Projekte mit Sourcecode der mobilen Erweiterungen und dem Original-Sourcecode der ActiveBPEL Engine.
- Ordner *BPEL-Quellcode* enthält Projekte für den ActiveBPEL Designer 4.0:
  - Ordner *BpelMobileExtensions* enthält XML-Schemata für die mobilen BPEL-Erweiterungen, für WS-BPEL 2.0 und WSDL 1.1.
  - Ordner *CarRepairServices* enthält BPEL-Prozesse und weitere Quellen für das Szenario "Auto-Pannendienst".
- Ordner *BPEL-Quellcode-Durchstich* enthält Projekt *TestService* mit BPEL-Prozess und weiteren Quellen aus dem Durchstich.
- Datei *Masterarbeit.pdf* enthält diese Arbeit im pdf-Format.

Die Dateien mit Namen *Info.txt* – zu finden im Root- und in den vier Hauptordnern – enthalten jeweils weitergehende Hinweise zu Inhalt, Installation etc.

CD-ROM:

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den 25. Februar 2008

Stephanie Gamm