Aus dem Institut für Telematik
der Universität zu Lübeck

Direktor:
Prof. Dr. rer. nat. Stefan Fischer

Ein dynamisches, ganzheitliches Geschäftsprozessmanagement in Enterprise-IT-Systemen und drahtlosen Sensornetzen

Inauguraldissertation
zur
Erlangung der Doktorwürde
der Universität zu Lübeck

Aus der Sektion Informatik / Technik

Vorgelegt von
Dipl.-Wirt.-Inf. Nils Glombitza
aus Wolfenbüttel

Lübeck, im Juni 2011

Vorsitzender des Prüfungsausschusses:	Prof. Dr. Jörg-Uwe Meyer
Erster Berichterstatter:	Prof. Dr. Stefan Fischer
Zweiter Berichterstatter:	Prof. Dr. Volker Linnemann
Dritter Berichterstatter:	Prof. Dr. Manfred Hauswirth (National University of Ireland, Galway)
Tag der mündlichen Prüfung:	06. September 2011

Zum Druck genehmigt.
Lübeck, den 12. Januar 2012

Prof. Dr. Jürgen Prestin
Vorsitzender der Sektion für Informatik / Technik und Naturwissenschaften

Meinen Eltern

Vorwort

Die vorliegende Dissertation habe ich am Institut für Telematik an der Universität zu Lübeck unter der Leitung von Herrn Prof. Dr. Stefan Fischer verfasst. An dieser Stelle bedanke ich mich bei all denen, die mir in dieser Zeit zur Seite standen.

In erster Linie bedanke ich mich bei meinem Doktorvater Herrn Prof. Dr. Stefan Fischer, der mich bereits an der Technischen Universität Braunschweig als Student geprägt hat. Schon während meines Studiums entwickelte sich bei mir der Wunsch einer anschließenden Promotion. In einem spontanen persönlichen Gespräch mit Herrn Prof. Dr. Fischer, in dem er mir eine Arbeitsstelle als wissenschaftlicher Mitarbeiter anbot, wurde ich in meinem Vorhaben bestärkt. Ich bedanke mich sehr herzlich bei Herrn Prof. Dr. Fischer, dass er mir die Möglichkeit zur Promotion gegeben hat und meiner Themenwahl Begeisterung und Bestätigung entgegenbrachte, obwohl mein Forschungsschwerpunkt in großen Teilen in Themenbereiche vordringt, die vom Kernforschungsgebiet des Instituts für Telematik abweichen. Bei der Bearbeitung verschiedener Projekte hat er mir jederzeit großes Vertrauen entgegengebracht und sämtliche Freiheiten gelassen, die Forschungsschwerpunkte und Projektinhalte frei zu gestalten. Auch konnte ich meine Forschungsergebnisse stets auf internationalen Fachtagungen präsentieren und diskutieren.

Des Weiteren gilt mein Dank Herrn Prof. Dr. Volker Linnemann sowie Herrn Prof. Dr. Manfred Hauswirth für die Übernahme der Zweit- und Drittgutachten. Außerdem bedanke ich mich bei Herrn Prof. Dr. Jörg-Uwe Meyer dafür, dass er den Prüfungsvorsitz in meinem Promotionsverfahren übernommen hat.

Bei meinen ehemaligen und derzeitigen Kolleginnen und Kollegen bedanke ich mich besonders für das freundliche Miteinander und die angenehme Arbeitsatmosphäre.

Mein Dank gilt nicht zuletzt meinen Eltern, die mir stets die notwendige Ruhe und Sicherheit bei der Erstellung dieser Arbeit gaben. In Dankbarkeit widme ich ihnen diese Arbeit.

Lübeck, im Februar 2012

Kurzfassung

In der Vision des Internets der Dinge sollen in Zukunft verschiedenartige Geräte, von ressourcenschwachen Sensornetzen bis zu leistungsstarken Unternehmensservern, das Internet mit der realen Welt verknüpfen. Aus der Umsetzung dieser Vision würden sich für Unternehmen Möglichkeiten zur Realisierung völlig neuartiger Geschäftsprozesse und zur Generierung von Wettbewerbsvorteilen ergeben. Eine Integration von ressourcenbeschränkten Geräten wie drahtlosen Sensornetzen scheitert zurzeit jedoch an ihrer komplexen und monolithischen Anwendungsentwicklung und der Verwendung von proprietären, zur Enterprise-IT inkompatiblen Technologien sowie an den daraus entstehenden hohen Entwicklungs- und Integrationskosten.

Im Rahmen dieser Arbeit wurden konzeptionelle und technische Lösungen für ein dynamisches und ganzheitliches Geschäftsprozessmanagement in Sensornetzen und Enterprise-IT-Systemen entwickelt, die die o. g. Probleme überwinden. Das Konzept dieser Arbeit sieht zur Ausführung von Geschäftsprozessen serviceorientierte Architekturen als Systemarchitektur vor. Dabei wird jede Funktionalität sowohl im Enterprise-IT-Bereich als auch im Sensornetz als in sich abgeschlossene Dienste bereitgestellt und verwendet. Darauf setzt ein Geschäftsprozessmanagement auf. Es realisiert Geschäftsprozesse informationstechnisch als eine zeitlich logische Folge von Dienstaufrufen und stellt diese ebenfalls als Dienste bereit. Dabei erfolgen die Modellierung, Überwachung und Optimierung von Prozessen aus einer fachlichen Perspektive und können von Fachpersonal durchgeführt werden.

Die zurzeit bedeutendste Umsetzungstechnologie serviceorientierter Architekturen in Enterprise-IT-Umgebungen stellen Webservices dar. Um Sensornetze erfolgreich in die Enterprise-IT zu integrieren, müssen Webservices auch in Sensornetzen eingesetzt werden. Heutige Standards zur Realisierung einer Webservice-Kommunikation sowie zur dynamischen Dienstvermittlung weisen jedoch einen zu hohen Ressourcenverbrauch auf, um in Sensornetzen verwendet werden zu können. Zur Lösung dieses Problems wurden im Rahmen dieser Arbeit ein effizientes Transportprotokoll und ein SOAP-Transport-Binding für eine transparente und ganzheitliche Webservice-Kommunikation in Sensornetzen und Enterprise-IT-Systemen entwickelt.

Jede Webservice-Kommunikation setzt die paarweise Verwendung desselben Transport-Bindings voraus. Zur Erhöhung der Interoperabilität wurde im Rahmen dieser Arbeit eine transparente und konfigurationsfreie automatische Webservice-Transport-Binding-Konvertierung entwickelt. Damit kann auch eine Kommunikation mit Partnern stattfinden, die nicht das im Rahmen dieser Arbeit entwickelte Binding verwenden.

Sensornetze zeichnen sich durch eine hohe Dynamik aus. Eine statische Bindung zwischen Webservice-Anbieter und -Nutzer ist aus diesem Grund nicht sinnvoll. Deshalb wurde im Rahmen dieser Arbeit ein Protokoll zum dynamischen Auffinden und zur Selbstbeschreibung von Webservices entwickelt, das die Vorgaben von Sensornetzen bzgl. eines geringen Ressourcenverbauchs einhält.

Zur technischen Umsetzung eines Geschäftsprozessmanagements wurde im Rahmen dieser Arbeit ein Workflowmanagementsystem entwickelt, das eine transparente und ganzheitliche Definition, Ausführung, Überwachung und Optimierung von Geschäftsprozessen im Kontext von Sensornetzen und Enterprise-IT-Systemen realisiert und direkt von Fachpersonal verwendet werden kann. Zur Prozessdefinition werden unterschiedliche Sprachen eingesetzt. Sie erlauben eine deklarative Prozessbeschreibung durch die Aggregation von Webservices und stellen die Prozessfunktionalität selbst wieder als Webservices bereit. Entsprechende Prozessmodelle sind direkt in dem im Rahmen dieser Arbeit entwickelten Workflowmanagementsystem sowohl auf Enterprise-IT-Systemen als auch auf Sensorknoten ausführbar.

Die technische Umsetzung des Konzepts dieser Arbeit basiert vollständig auf Webservices. Dabei sind alle im Rahmen dieser Arbeit entwickelten Lösungen konform zu existierenden Webservice-Standards oder entsprechen den vom World Wide Web Consortium definierten Erweiterungsmöglichkeiten der Webservice-Architektur.

Mit den im Rahmen dieser Arbeit entwickelten konzeptionellen und technischen Lösungen kann ein Geschäftsprozessmanagement ganzheitlich in drahtlosen Sensornetzen und Enterprise-IT-Systemen realisiert werden. Damit lassen sich Sensornetze nahtlos in die Unternehmens-IT integrieren und ihre Anwendungsentwicklung schnell und flexibel durchführen. So kann das Potenzial dieser Geräteklasse zur Ermöglichung neuartiger Geschäftsprozesse und der Generierung von Wettbewerbsvorteilen ausgeschöpft werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Quelltextverzeichnis

Algorithmenverzeichnis

1 Einleitung

Unternehmen sind heute mehr denn je einem zunehmenden Wettbewerbsdruck ausgesetzt. Um den Fortbestand und die Entwicklung eines Unternehmens zu sichern, bedarf es einer ständigen Anpassung der Geschäftsprozesse. Dabei sind die IT-Nutzer (Fachabteilungen) und IT-Bereitsteller untrennbar miteinander verbunden (s. Abbildung 1.1). So muss zum einen die IT die Optimierung vorhandener Geschäftsprozesse zur Steigerung der Effektivität und Effizienz sowie ihre schnelle Anpassung an sich ständig ändernde Anforderungen des Zielmarktes eines Unternehmens durch eine entsprechende informationstechnische Umsetzung gewährleisten (*anpassen*). Diese Anpassungen der Unternehmens-IT reichen jedoch zur dauerhaften Erzielung von Wettbewerbsvorteilen nicht aus. Deshalb muss die IT zum anderen durch die Entwicklung neuer technischer Optionen, die die Entwicklung völlig neuartiger Geschäftsprozesse ermöglichen, die Unternehmensstrategie aktiv beeinflussen (*ermöglichen*).

Um eine schnelle, flexible und fehlerfreie Anpassung der Unternehmens-IT zu gewährleisten, werden Unternehmensanwendungen gegenwärtig nach dem Architekturkonzept der serviceorientierten Architektur (SOA) entwickelt. In einer SOA wird Anwendungsfunktionalität als in sich abgeschlossene Dienste, die meist über ein Netzwerk verteilt sind, zur Nutzung bereitgestellt. Einzelne Dienste können flexibel zu höherwertigen Diensten und Geschäftsprozessen zusammengesetzt werden. Änderungen der Geschäftsprozesse können in einer SOA ohne großen Aufwand durch eine neue Zusammenstellung der existierenden Dienste umgesetzt werden. Eine grundlegende Neuimplementierung von Anwendungsfunktionalität ist meist nicht notwendig.

Die durch eine SOA erzielte Flexibilität der Anwendungssystemgestaltung wird durch den Einsatz eines Geschäftsprozessmanagements ergänzt. Es erlaubt die informationstechnische Realisierung sowie die Überwachung und Optimierung von Prozessen aus einer fachlichen

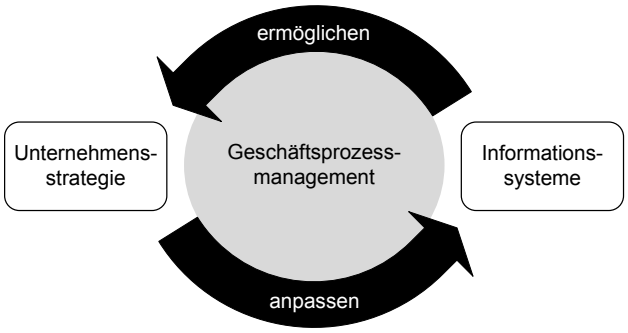


Abbildung 1.1: Geschäftsprozessmanagement zwischen „ermöglichen“ und „anpassen“ (in Anlehnung an [104])

Perspektive. Dabei werden meistens grafische, domänenspezifische Sprachen für die Prozessmodellierung verwendet, die eine schnelle und flexible Definition von Geschäftsprozessen ermöglichen, indem bereits existierende Dienste deklarativ zu neuen Diensten zusammengesetzt werden. Entsprechende Prozessmodelle können anschließend direkt in speziellen Laufzeitumgebungen ausgeführt werden.

Parallel zu den Forschungen zu serviceorientierten Architekturen und deren Umsetzungstechnologien sowie der darauf aufbauenden Geschäftsprozessmodellierung wird zurzeit an Konzepten und Technologien zur Realisierung des zukünftigen Internets (engl.: *Future Internet*) geforscht. So soll das Internet der Dinge (engl.: *Internet of Things*, IoT) das Internet in Zukunft auf die reale Welt erweitern, indem jegliche Gerätearten standardisiert über das Internet miteinander kommunizieren. Ressourcenschwache Sensorknoten sollen in dieser Vision genauso in Unternehmensanwendungen integriert werden, wie leistungsstarke Server.

Für Unternehmen ergeben sich die Möglichkeiten, auf Basis des Internets der Dinge völlig neuartige Anwendungen und Geschäftsprozesse zu entwickeln und so ihre strategischen Wettbewerbsvorteile zu sichern (*ermöglichen*). So ist z. B. der Einsatz von drahtlosen Sensornetzen (WSNs) in logistischen Wertschöpfungsketten, in denen viele Unternehmen an der Produktion und Lagerung sowie dem Versand von Waren beteiligt sind, denkbar. Drahtlose Sensornetze bestehen aus sogenannten Sensorknoten, die Prozessor, Speicher, drahtlose Kommunikationstechnik und Sensorik in einem Kleinstcomputer vereinen. Sie können Informationen über den Standort und Zustand von Gütern automatisch und in Echtzeit erfassen und den Unternehmensanwendungen oder Geschäftsprozessen zur Verfügung stellen. So kann die Prozessqualität gesteigert und die Abläufe effizienter gestaltet werden.

1.1 Motivation

Trotz ihres großen Potenzials werden Sensornetze und andere IoT-Geräte gegenwärtig nicht in Unternehmensanwendungen oder Geschäftsprozesse integriert. Als Gründe dafür sind vor allem ihre komplexe und monolithische Anwendungsentwicklung und die daraus resultierenden hohen Entwicklungs- und Integrationskosten zu nennen.

Zwar ist es bereits heute möglich, Sensornetze in Unternehmensanwendungen zu integrieren. Dabei stellt jedoch das Sensornetz einen monolithischen Anwendungsblock dar, der intern proprietäre Kommunikationsprotokolle verwendet und über ein applikationsspezifisches Gateway in die Unternehmensanwendungen integriert wird. Durch dieses Anwendungsdesign ist eine schnelle und flexible Entwicklung und Anpassung von Sensornetzapplikationen und dadurch eine schnelle Reaktion auf sich ständig ändernde Marktanforderungen nicht möglich. Dieses gilt bereits bei einer unternehmensinternen Geschäftsprozessintegration. Eine unternehmensübergreifende Integration ist unter diesen Bedingungen völlig undenkbar.

Zur schnellen und flexiblen Anwendungsentwicklung und -anpassung wird in Unternehmensanwendungen das SOA-Konzept als Systemarchitektur zur informationstechnischen Umsetzung von Geschäftsprozessen verwendet. Um Sensornetze und andere IoT-Netze erfolgreich in Unternehmensanwendungen und Geschäftsprozesse zu integrieren, muss eine serviceorientierte Anwendungsentwicklung auch in diesen Netzen umgesetzt werden. Dabei reicht jedoch

die konzeptionelle Umsetzung alleine nicht aus. Vielmehr müssen sich auch die verwendeten Umsetzungstechnologien nahtlos in die bestehenden Infrastrukturen der Enterprise-IT integrieren.

Während Sensorknoten bzgl. ihres Stromverbrauchs, ihrer Beschaffungskosten sowie Größe auf ein Minimum optimiert und dementsprechend ressourcenschwach sind, verfügen Enterprise-IT-Systeme über sehr große Ressourcen. Entsprechend sind SOA-Technologien in erster Linie auf Kompatibilität und weniger auf Ressourcenschonung ausgelegt. Insbesondere Webservices als die zurzeit bedeutendste Technologie zur Umsetzung einer SOA überfordern die Kapazitäten von Sensornetzplattformen bei Weitem. So existiert bislang keine Lösung, die eine effiziente und transparente Webservice-Kommunikation zwischen Enterprise-IT-Diensten und Diensten im Sensornetz ermöglicht.

Darüber hinaus sind Sensornetze durch eine hohe Dynamik und Ad-hoc-Kommunikationsbeziehungen geprägt. Das setzt die Fähigkeit der Selbstbeschreibung und des automatischen, dezentralen Auffindens von Diensten voraus. Existierende Webservice-Lösungen zum Auffinden von Diensten sowie zum Austausch von Schnittstellenbeschreibungen lassen sich aufgrund ihrer Ressourcenanforderungen jedoch nicht im Sensornetz verwenden. Darüber hinaus ist es nicht einmal möglich, die Schnittstellenbeschreibungen entsprechender Webservices, die zurzeit ausschließlich in der Sprache WSDL (*Web Services Description Language*, [232]) beschrieben werden, auf Sensorknoten zu speichern.

In einer Webservice-basierten Enterprise-IT werden nicht elementare Dienste und Geschäftsprozesse im Rahmen eines Geschäftsprozessmanagements nicht unter Verwendung von Programmiersprachen wie C, C++ oder Java implementiert. Stattdessen werden spezielle Geschäftsprozessbeschreibungssprachen wie BPEL (*Business Process Execution Language*, [159]) zur domänenspezifischen Prozess- und Anwendungsentwicklung eingesetzt. Mangels einer zu entsprechenden Standards kompatiblen Realisierung einer Webservice-Kommunikation im Sensornetz, ist die Integration von Sensornetzen in entsprechende Enterprise-IT-Geschäftsprozesse unmöglich.

Heutige Laufzeitumgebungen zur Ausführung von Geschäftsprozessen verursachen einen hohen Ressourcenverbrauch. Deshalb ist ihre Ausführung auf Sensorknoten aufgrund der geringen Ressourcenausstattung dieser Geräteklasse zurzeit gänzlich ausgeschlossen. Stattdessen ist die Entwicklung von Sensornetzapplikationen durch eine extrem systemnahe Programmierung gekennzeichnet, die ausschließlich von Experten durchgeführt werden kann. Entsprechend steigern die hohe Komplexität und Fehleranfälligkeit sowie der erhöhte Testaufwand den Zeitbedarf und die Kosten bei der Entwicklung, Integration sowie Anpassung von WSN-Applikationen und stellen damit eine hohe Markteintrittsbarriere für diese Technologie dar.

Das Ziel dieser Arbeit ist es, die eben beschriebenen Probleme zu lösen und ein dynamisches, ganzheitliches Geschäftsprozessmanagement in Enterprise-IT-Systemen und drahtlosen Sensornetzen zu realisieren. Damit sollen nicht nur Sensornetze in Geschäftsprozesse der Enterprise-IT integriert werden. Vielmehr sollen sie als gleichwertige Plattform zur informationstechnischen Prozessumsetzung dienen. So sollen die technologischen Voraussetzungen geschaffen werden, um mit dieser innovativen Technologie analog zu Abbildung 1.1 neue Unternehmensstrategien und damit die Generierung von Wettbewerbsvorteilen zu ermöglichen.

1.2 Wissenschaftliche Beiträge und Struktur der Arbeit

Die vorliegende Arbeit wird durch die genannten Probleme und Ziele motiviert. Als wissenschaftliche Beiträge liefert sie konzeptionelle und technologische Lösungen zur Realisierung eines dynamischen, ganzheitlichen Geschäftsprozessmanagements in Enterprise-IT-Systemen und Sensornetzen. Diese sind im Einzelnen:

1. ein entsprechendes Geschäftsprozessmanagementkonzept,
2. ein effizientes Transportprotokoll und ein dem SOAP-Standard [256] entsprechendes Transport-Binding für eine transparente, ganzheitliche Webservice-Kommunikation in Enterprise-IT-Systemen und Sensornetzen,
3. ein Workflowmanagementsystem zur transparenten, ganzheitlichen Modellierung, Ausführung, Simulation, Überwachung und Optimierung von Geschäftsprozessen in Enterprise-IT-Systemen und Sensornetzen,
4. eine domänenspezifische Sprache zur grafischen Aggregation von Webservices,
5. ein Protokoll zur ganzheitlichen dynamischen, dezentralen Webservice-Vermittlung bestehend aus je einem Protokoll zum Auffinden von Webservices und zum Austausch ihrer Schnittstellenbeschreibungen sowie einem Verfahren zur optimierten Kompression von WSDL-Dokumenten,
6. ein Verfahren zur transparenten und konfigurationsfreien automatischen Konvertierung zwischen verschiedenen Webservice-Transport-Bindings.

Bevor die eben aufgezählten Forschungsbeiträge dieser Arbeit beschrieben werden, führt Kapitel 2 die relevanten Grundlagen ein. Dazu werden als Erstes die konzeptionellen Grundlagen serviceorientierter Architekturen sowie die eines Geschäftsprozessmanagements erläutert. Im Anschluss werden mit Webservices und drahtlosen Sensornetzen die technologischen Grundlagen dieser Arbeit beschrieben.

Kapitel 3 diskutiert die Anforderungen eines Geschäftsprozessmanagements im Kontext von Sensornetzen und bewertet aktuelle Ansätze zur Realisierung von Sensornetzapplikationen auf ihre Verwendbarkeit im Rahmen eines Prozessmanagements. Darüber hinaus wird in Kapitel 3 das im Rahmen dieser Arbeit entwickelte Konzept zum dynamischen, ganzheitlichen Geschäftsprozessmanagement in Enterprise-IT-Systemen und Sensornetzen beschrieben.

In den Kapiteln 4 bis 6 werden die technologischen Beiträge dieser Arbeit zur Umsetzung des o. g. Konzepts präsentiert. Dazu wird als Erstes in Kapitel 4 die Umsetzung einer Webservice-Kommunikation in Sensornetzen und Enterprise-IT-Systemen beschrieben. Nach einer Definition von Anforderungen und einer Diskussion verwandter Arbeiten wird das vom Autor dieser Arbeit entwickelte SOAP-Transport-Binding LTP+SMC beschrieben. Es verwendet eine effiziente Kompression von SOAP-Nachrichten namens SMC (*SOAP Message Compression*) und das im Rahmen dieser Arbeit entwickelte *Lean Transport Protocol* (LTP). Das Kapitel schließt mit einer Analyse der Leistungsmerkmale von LTP+SMC.

Nach einer Diskussion der Anforderungen und verwandter Arbeiten beschreibt Kapitel 5 das im Rahmen dieser Arbeit entwickelte *Integral Workflow Management System* (IWFMS) zur

ganzheitlichen Geschäftsprozessumsetzung in Sensornetzen und Enterprise-IT-Systemen. Neben dem existierenden Standard BPEL setzt IWFMS die im Rahmen dieser Arbeit entwickelte Prozessrealisierungssprache SM4RCD (*Web Services State Machine for Resource Constrained Devices*) zur Prozessdefinition ein. SM4RCD wird ebenfalls in Kapitel 5 beschrieben, das mit einer quantitativen Analyse von IWFMS am Beispiel eines Geschäftsprozesses aus der Logistikdomäne schließt.

Kapitel 6 beschreibt als Erstes das im Rahmen dieser Arbeit entwickelte Protokoll LDDE (*Lean Description, Discovery and Exchange*) zur dynamischen Dienstvermittlung, bevor im zweiten Teil die transparente und konfigurationsfreie automatische Webservice-Transport-Binding-Konvertierung beschrieben wird. In beiden Teilen werden zu den im Rahmen dieser Arbeit entwickelten Lösungen verwandte Arbeiten präsentiert sowie eine Analyse der jeweiligen Leistungsmerkmale durchgeführt.

Kapitel 7 schließt diese Arbeit mit einer Zusammenfassung und einem Ausblick auf zukünftige Forschungsaktivitäten.

2 Grundlagen

In diesem Kapitel werden die Konzepte und Technologien beschrieben, die die Grundlage dieser Arbeit bilden. In Abschnitt 2.1 wird mit der serviceorientierten Architektur das in dieser Arbeit verwendete informationstechnische Konzept zur Gestaltung von Unternehmensanwendungen und zur Umsetzung von Geschäftsprozessen beschrieben. Dazu bilden die in Abschnitt 2.2 erläuterten Grundlagen des Geschäftsprozessmanagements mit der fachlichen Gestaltung von Geschäftsprozessen das betriebswirtschaftliche Gegenstück. Während die ersten beiden Abschnitte die Realisierung von Geschäftsprozessen vor allem auf einer konzeptionellen Ebene beschreiben, wird in Abschnitt 2.3 mit Webservices eine konkrete Umsetzungstechnologie beschrieben, mit der sich geschäftsprozessrealisierende serviceorientierte Architekturen konkret implementieren lassen. Mit diesen konzeptionellen und technologischen Grundlagen lässt sich die Anpassung der IT an die Unternehmensstrategie verwirklichen. Mit den drahtlosen Sensornetzen werden in Abschnitt 2.4 die Grundlagen einer neuen Technologie beschrieben. Mit den im Rahmen dieser Arbeit entwickelten Konzepten und Technologien ermöglichen Sensornetze analog zu Abbildung 1.1 auf Seite 1 die Umsetzung völlig neuartiger Geschäftsprozesse.

2.1 Grundlagen serviceorientierter Architekturen

In Unternehmensanwendungen wird das Paradigma der serviceorientierten Architektur (SOA) für die Implementierung und Anpassung von Geschäftsprozessen, deren Prozessschritte häufig über ein Netzwerk verteilt ausgeführt werden, verwendet. Dabei muss eine SOA wie jedes andere IT-Konzept die Erfüllung fachlicher Aufgaben ermöglichen. Zusätzlich sind für heutige Geschäftsanwendungen eine geringe Dauer und geringe Kosten zur Erlangung der Marktreife als Schlüsselanforderungen zu nennen. Sie setzen ein hohes Maß an Flexibilität und Wiederverwendbarkeit voraus. Gleichzeitig werden Anwendungssysteme und Geschäftsprozesse immer stärker unternehmensintern sowie unternehmensübergreifend integriert.

Zum Thema SOA entstanden in der Vergangenheit sowohl aus universitären Forschungsprojekten sowie aus Erfahrungsberichten der Unternehmenspraxis eine Vielzahl an Publikationen. Trotzdem – oder gerade deshalb – existiert zurzeit keine einheitliche Definition einer SOA. So definiert OASIS (*Organization for the Advancement of Structured Information Standards*[1]) in ihrem SOA-Referenzmodell [156] eine SOA sehr allgemein als ein Paradigma zur Verwaltung und Nutzung von verteilten Kompetenzen, die sich unter der Kontrolle von verschiedenen Eigentümern befinden können. Erl [53] hingegen sieht eine SOA als ein Architekturmodell, das Effizienz, Agilität und Produktivität eines Unternehmens durch den Einsatz einer serviceorientierten Lösungslogik steigern will.

[1] OASIS ist ein aus über 600 Organisationen bestehendes Konsortium zur Standardisierung von Basistechnologien für Informationssysteme.

Bei den hier aufgeführten sowie bei weiteren gebräuchlichen Definitionen entstehen Überlappungen und Unterschiede durch die als jeweils wichtig angesehenen konstituierenden Merkmale einer SOA sowie durch unterschiedliche Generalitätsgrade der Definitionen. Deshalb wird im Rahmen dieser Arbeit auch auf eine Auflistung und Diskussion unterschiedlicher SOA-Definitionen verzichtet und im Folgenden auf die Entwurfsmerkmale und -prinzipien einer SOA eingegangen.

Basis einer serviceorientierten Lösungslogik sind nach allen SOA-Definitionen *Services*, die in sich abgeschlossene Softwarekomponenten darstellen. Services kapseln Anwendungsfunktionalität, die die Erreichung eines strategischen Ziels innerhalb einer SOA unterstützt. Neben der Basiseinheit Service ist die *Servicekomposition* das zweite konstituierende Merkmal einer serviceorientierten Lösungslogik. Verschiedene in Diensten gekapselte Anwendungslogik kann über standardisierte Schnittstellen von Nutzern konsumiert werden. Die so entstandene Anwendungsfunktionalität kann selbst wieder in zusammengesetzten Diensten, sogenannten *Servicekompositionen* oder *-aggregaten*, bereitgestellt werden. Ziel einer SOA ist es, die Anwendungsfunktionalität und Prozesse eines Unternehmens so in unabhängigen Diensten zu kapseln, dass die Erstellung und Anpassung der entsprechenden Geschäftsprozesse allein durch eine (Re-)Komposition einzelner Dienste aus einer Sammlung von Diensten, dem sogenannten *Serviceinventar*, ermöglicht werden.

In den folgenden Abschnitten werden die Ziele und Vorteile (s. Abschnitt 2.1.1) von Serviceorientierung sowie die zu ihrer Realisierung notwendigen Merkmale (s. Abschnitt 2.1.2) einer SOA erläutert. Dieser Grundlagenabschnitt schließt mit einer Kategorisierung von Diensten (s. Abschnitt 2.1.3) und der Beschreibung der in einer SOA existierenden Rollen sowie den Interaktionen zwischen den verschiedenen Rollen (s. Abschnitt 2.1.4).

2.1.1 Ziele und Vorteile von Serviceorientierung

Durch das Design der Unternehmens-IT entsprechend dem serviceorientierten Entwurfsparadigma können sowohl betriebswirtschaftliche als auch technische Ziele erreicht und Vorteile für das Gesamtunternehmen generiert werden. Erl definiert sieben miteinander in Zusammenhang stehende strategische Ziele einer Serviceorientierung [53]. Dabei lassen sich die drei betriebswirtschaftlichen Ziele der Verbesserung der *Investitionsrendite*, der *Agilität des Unternehmens* und die Verringerung der *IT-Belastung* als Vorteile für das Gesamtunternehmen aus den technischen Zielen der Verbesserung der *inhärenten Interoperabilität*, der *Föderation*, der *Herstellerunabhängigkeit* sowie der *Abstimmung von Geschäft und Technologie* generieren.

Mit Interoperabilität ist der Einfachheitsgrad des Informationsaustauschs zwischen Softwarekomponenten gemeint. In einer SOA wird Interoperabilität nicht durch Integration, also den Entwicklungsprozess der Herstellung von Interoperabilität, erreicht. Stattdessen sollen die konsistente Anwendung von einheitlichen Entwurfsprinzipien und der Einsatz von Standards eine inhärente Interoperabilität gewährleisten. Beliebige Dienste des Serviceinventars können so unabhängig von ihrem Erstellungszeitpunkt und -projekt in beliebigen Servicekompositionen verwendet werden.

Obwohl die Interaktion zwischen verschiedenen Diensten essenzielles Merkmal einer SOA ist, sollen einzelne Dienste autonom und selbstständig verwaltet sowie gesteuert und damit die Föderation verbessert werden. Um dieses Ziel zu erreichen, müssen durchgängig standardisierte

und kompositionsfähige Dienste, die jeweils ein Unternehmenssegment kapseln, eingesetzt werden.

Auch wenn sich durch die Bindung an einen oder wenige Hersteller von IT-Produkten Synergie- und Kostendegressionseffekte erzielen lassen, so birgt eine Herstellerabhängigkeit erhebliche betriebswirtschaftliche und technologische Risiken. Ein Unternehmen ist in einem solchen Szenario der Produkt- und Preispolitik eines Herstellers vollkommen ausgeliefert [12]. Durch den Einsatz von Standards soll in einer serviceorientierten Architektur eine Herstellerabhängigkeit vermieden oder zumindest reduziert werden, um bei Beschaffungsentscheidungen für neue Produkte und Technologien nicht bereits durch Kaufentscheidungen der Vergangenheit eingeschränkt zu werden.

Um die Geschäfts- bzw. Domänenanforderungen optimal zu erfüllen, muss die Geschäftslogik entsprechend durch die Lösungslogik ausgedrückt werden. Auch wenn monolithische oder vertikal organisierte Anwendungssysteme gemäß der Geschäftsanforderungen entworfen werden, so ist nach diesem Entwurfsparadigma eine nachträgliche Anpassung an sich ändernde Anforderungen nicht bzw. nur schwer möglich. Durch das serviceorientierte Entwurfsparadigma soll Geschäftslogik präzise in Services gekapselt und neue bzw. geänderte Geschäftsprozesse durch die (Re-)Komposition vorhandener Dienste ermöglicht werden. Diese durch eine Serviceorientierung erlangte Flexibilität ermöglicht die Ausrichtung der Unternehmens-IT an den Geschäftsprozessen, d. h. eine horizontale IT-Organisation.

Die Anpassung monolithischer Anwendungen ist langwierig, fehleranfällig sowie komplex und somit kostenintensiv. Durch das Wiederverwendungspotenzial unabhängiger Dienste des Serviceinventars stellen Investitionen in SOAs bleibende Vermögenswerte dar, die wiederholbare, langfristige Gewinne erzielen und damit die Investitionsrendite verbessern [53].

Die Wettbewerbsfähigkeit und das Überleben eines Unternehmens hängen von seiner Agilität, also der Fähigkeit einer schnellen Anpassung, ab. Gerade in monolithischen IT-Umgebungen wird die Agilität des gesamten Unternehmens durch eine zeit- und ressourcenintensive Anpassung der IT an sich ändernde Geschäftsanforderungen stark eingeschränkt. Durch eine Wiederverwendung und Komposition agnostischer Dienste kann die Agilität bzw. Flexibilität der Unternehmens-IT und damit die Agilität des gesamten Unternehmens gesteigert werden. Eine kontinuierliche gegenseitige Beeinflussung von Unternehmensstrategie und IT wird erst durch eine Serviceorientierung möglich. Des Weiteren wird trotz schnellerer Markteinführungszeiten eine höhere Softwarequalität erreicht.

Durch Serviceorientierung werden nicht nur bei der Entwicklung bzw. Anpassung von IT-Komponenten, sondern auch bei deren Betrieb, Effektivitäts- und Kosteneffizienzvorteile erzielt. Diese sind insbesondere auf eine Reduktion von Redundanzen und damit auf eine Verminderung des Wartungsaufwands zurückzuführen.

2.1.2 Merkmale serviceorientierter Architekturen

Zur Realisierung der im vorigen Abschnitt beschriebenen Ziele und Vorteile einer Serviceorientierung müssen in einer SOA verschiedene technische Konzepte und Merkmale erfüllt werden. Trotz der Heterogenität serviceorientierter Architekturen sind sie durch eine hohe *Interoperabilität* gekennzeichnet. Interoperabilität steht mit weiteren SOA-Merkmalen in

wechselseitigen Beziehungen. So ist sie zum einen Voraussetzung zur Umsetzung der in den folgenden Absätzen beschriebenen SOA-Merkmale. Zum anderen tragen die einzelnen Eigenschaften einer SOA ihrerseits dazu bei, Interoperabilität zu erreichen [53].

Wie bereits beschrieben, ist die Basiseinheit einer serviceorientierten Lösungslogik und damit auch einer SOA der Service. Funktionalität wird so innerhalb in sich abgeschlossener Dienste gekapselt, dass die internen Implementierungsdetails vor dem Nutzer verborgen bleiben. Diese Eigenschaft von Diensten wird in der Literatur als *Servicekapselung* [43] oder *Serviceabstraktion* [53] bezeichnet. Die Interaktion zwischen Dienstnutzer und -anbieter erfolgt einzig über *standardisierte Serviceverträge*, die den Zweck und die Fähigkeiten eines Dienstes ausdrücken. Serviceverträge sind in diesem Zusammenhang zum einen als technische bzw. funktionale Schnittstellen zu sehen, die z. B. die ausgetauschten Daten beschreiben. Zum anderen regeln sie nichtfunktionale Eigenschaften von Diensten, wie z. B. Qualitätsrichtlinien und Informationen über Abrechnungsmodelle.

Da der Begriff der *losen Kopplung* sehr subjektiv ist und sich auf verschiedene Aspekte beziehen kann, wird er bewusst nicht im OASIS-SOA-Referenzmodell [156] verwendet. Trotzdem wird er in vielen Publikationen, so auch in dieser, als zentrale Eigenschaft einer SOA angesehen. Mit Kopplung wird ein Maß definiert, das die Abhängigkeit zweier in Beziehung stehender „Dinge“ (hier also Dienste) beschreibt. Bei einer losen Kopplung werden diese Abhängigkeiten reduziert, um die Auswirkungen von Veränderungen und Fehlverhalten in einem im höchsten Maße dynamischen Umfeld wie einer SOA zu minimieren und darüber hinaus die Skalierbarkeit der Systeme zu erhöhen.

Um die Fähigkeiten eines Dienstes zuverlässig und konsistent realisieren zu können, muss die verantwortliche Organisationseinheit eigenständig und unabhängig die Umgebung und Ressourcen eines Dienstes kontrollieren sowie Entscheidungen bzgl. seines Lebenszyklus treffen können. Diese *Serviceautonomie* setzt dabei auf der technologischen Ebene das Prinzip der losen Kopplung zur Minimierung der Auswirkungen aufseiten des Dienstnutzers durch autonome Entscheidungen des Anbieters voraus. Demgegenüber regeln auf der organisatorischen Ebene Serviceverträge zwischen Anbieter und Nutzer die Ausprägung bestimmter Dienstmerkmale und schränken somit unter Umständen die Serviceautonomie wieder ein.

Die *Wiederverwendbarkeit* ist eines der wichtigsten Merkmale einer SOA und somit Kernbestandteil aller Serviceanalyse- und -entwurfsprozesse. Sie wird durch die Positionierung von Diensten als Unternehmensressource mit agnostischen funktionalen Kontexten ermöglicht und vermeidet die Implementierung redundanter Lösungslogik.

Serviceorientierung ist eine Lösungslogik für große, komplexe Systeme. Zur Beherrschung dieser Systeme wird mit der Trennung von Anforderungen (engl.: *Separation of Concerns*) eine Theorie des Software-Engineering angewendet, die besagt, dass sich große Probleme leichter lösen lassen, wenn sie in eine Vielzahl kleinerer Probleme zerlegt werden. Analog wird durch eine serviceorientierte Lösungslogik erreicht, dass komplexe Dienste (zur Lösung komplexer Aufgaben) durch die *Komposition existierender Dienste* des Serviceinventars erstellt werden können. Diese Servicekompositionen können ihrerseits wieder in übergeordneten Kompositionen verwendet werden und so eine rekursive Dienstaggregation realisieren.

Damit Dienste langfristig und wiederholt eine Investitionsrendite erwirtschaften können, ist ihre potenzielle Wiederverwendbarkeit und Kompositionsfähigkeit eine notwendige, aber keine hinreichende Voraussetzung. Zusätzlich ist das *Auffinden* geeigneter Dienste aus dem

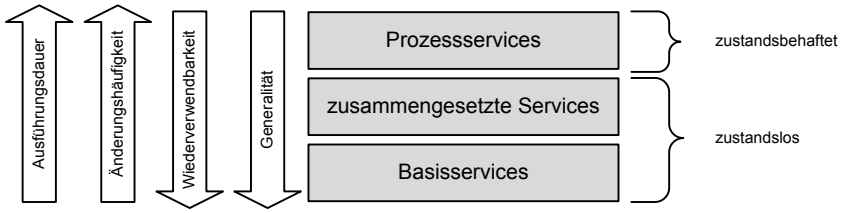


Abbildung 2.1: Servicekategorien und ihre charakteristischen Merkmale

Serviceinventar essenzielle Voraussetzung für eine Wiederverwendung bzw. für die Vermeidung einer Implementierung von Diensten mit redundanter Lösungslogik. Ein dynamisches Auffinden von Diensten zur Laufzeit und das Merkmal der losen Kopplung ermöglichen ein dynamisches Binden von Dienstnutzern und -anbietern zur Laufzeit.

Die Verwaltung einer Vielzahl an Zustandsinformationen kann negative Einflüsse auf die Verfügbarkeit und Skalierbarkeit von Diensten ausüben. Aus diesem Grund wird die *Zustandslosigkeit* von vielen Autoren als weiteres Merkmal einer SOA angesehen. Nach Meinung des Autors dieser Arbeit ist Zustandslosigkeit zwar eine erstrebenswerte Eigenschaft. Sie lässt sich allerdings nicht bei allen Diensten realisieren. Insbesondere bei lang laufenden Geschäftsprozessen ist eine Zustandslosigkeit nicht realisierbar.

2.1.3 Dienstkategorien in serviceorientierten Architekturen

Wie in den vorangegangenen Abschnitten beschrieben, repräsentieren in einer SOA Services fachliche Funktionalitäten. Dabei kann z. B. in einer logistischen Wertschöpfungskette der Prozess der Verschiffung eines Trailers (inkl. Buchung, Durchführung und Überwachung sowie Abrechnung der Passage) informationstechnisch durch einen Dienst repräsentiert werden. Genauso werden aber auch die Optimierung der Ladungsverteilung auf Schiffen oder so elementare Funktionen wie das Auslesen eines Temperatursensors in Diensten gekapselt. Aus diesen Besonderheiten in der fachlichen Funktionalität ergeben sich auch unterschiedliche Anforderungen an Dienste, die folglich auch andersgeartete Eigenschaften aufweisen. Eine Gleichbehandlung aller Dienste ist damit weder aus technischer noch aus fachlicher Sicht sinnvoll.

Zur differenzierten Betrachtung werden Dienste deshalb klassifiziert. Für diese Arbeit, die technische Lösungen zur Umsetzung einer SOA in Umgebungen ressourcenbeschränkter Geräte präsentiert, ist eine Klassifizierung von Diensten anhand technischer Merkmale am geeignetsten. Die Einteilung von Diensten in die drei Kategorien *Basisservice*, *Composed-Service* (dt.: zusammengesetzter Dienst) sowie *Prozessservice* besitzt in der Literatur große Akzeptanz [20, 21, 94, 167]. Auch wenn die Benennung zwischen unterschiedlichen Autoren variiert und einige Autoren die drei Basisklassen differenzierter in weitere Klassen unterteilen, so stimmen doch die klassenbildenden Merkmale größtenteils überein. Im Rahmen dieser Arbeit hilft diese Kategorisierung bei der Auswahl der richtigen Realisierungstechnologien für konkrete Service- bzw. Geschäftsprozessimplementierungen. Abbildung 2.1 zeigt ein Schichtenmodell dieser Kategorien ergänzt um die jeweiligen charakteristischen Merkmale. Dabei drückt die Pfeilrichtung die Zunahme der Ausprägung eines jeweiligen Merkmals aus.

Basisdienste stellen jeweils eine fachliche Basisfunktionalität bereit. Sie haben typischerweise die Aufgabe, Entitäten oder logische Grundfunktionalität bzw. Geschäftsregeln zu kapseln und Nutzern sowie höherwertigen Diensten bereitzustellen. Aus diesem Grund werden sie häufig in Daten- und Logikdienste unterteilt. Basisdienste sind durch einen hohen Generalitäts- und somit Wiederverwendbarkeitsgrad gekennzeichnet und unterliegen selten Änderungen. Darüber hinaus sind sie kurz laufend und zumindest konzeptionell zustandslos.

Eine Ebene über den Basisdiensten befinden sich zusammengesetzte Dienste (engl.: *Composed-Services*). Sie sind Serviceaggregate, die aus Basisdiensten sowie anderen zusammengesetzten Diensten aggregiert werden. Im Gegensatz zu Prozessservices, die ebenfalls Serviceaggregate sind, sind Composed-Services eher durch einen Mikrokontrollfluss gekennzeichnet. Das bedeutet, dass sie aus einer Folge relativ schneller und kurz laufender Dienstaufrufe bestehen. Composed-Services sind genau wie Basisdienste selbst kurz laufend und konzeptionell zustandslos. Darüber hinaus ist der Generalitäts- und Wiederverwendbarkeitsgrad von Composed-Services zwar etwas geringer als bei Basisdiensten, aber immer noch wenig prozessspezifisch. Die Änderungsfrequenz ist ebenfalls nur leicht höher als die von Basisdiensten und weitaus geringer als bei Prozessdiensten.

Aufbauend auf Basis- und Composed-Services repräsentieren Prozessservices ganze Geschäftsprozesse. Sie setzen sich aus Composed-, aber auch aus Basisservices zusammen und sind damit ebenfalls Serviceaggregate. Bei Prozessdiensten handelt es sich um eine eher lang laufende Folge von Aktivitäten bzw. Dienstaufrufen. Im Gegensatz zu Basis- und Composed-Services besitzen Prozessdienste üblicherweise einen mehrere Serviceaufrufe überdauernden Zustand. So vergehen oft zwischen der Instanziierung einer Prozessinstanz (z. B. der Buchung einer Schiffspassage) bis zu deren Abschluss (der Trailer verlässt den Zielhafen) mehrere Wochen oder gar Monate. Prozessdienste sind des Weiteren sehr spezifisch auf einzelne betriebswirtschaftliche Geschäftsprozesse ausgerichtet und weisen somit einen geringen Wiederverwendungsgrad auf. Aufgrund der direkten Einflüsse von betriebswirtschaftlichen Marktveränderungen sind Prozessservices durch eine entsprechend hohe Änderungshäufigkeit charakterisiert.

2.1.4 Rollen in serviceorientierten Architekturen

In einer SOA sollen Dienste durch das Endsystem bzw. Servicekompositionen verwendet werden. Um dieses zu realisieren, ist Interaktion zwischen den Akteuren einer SOA über ein gemeinsames Nachrichten-Backbone notwendig. Dabei werden den Akteuren drei Rollen zugeordnet: *Serviceanbieter*, *Servicenutzer* sowie *Servicevermittler*. Bekannte Autoren aus dem SOA-Umfeld, wie z. B. Papazoglou [167] oder Melzer et al. [130] verwenden anstelle des Begriffs „Servicevermittler“ den Begriff „Dienstverzeichnis“. In dieser Arbeit wird die entsprechende Rolle jedoch bewusst als Servicevermittler bezeichnet, da das SOA-Rollenmodell eine konzeptionelle Ebene beschreibt, wohingegen der Begriff „Dienstverzeichnis“ bereits eine Umsetzungslösung festlegt. Diese Ungenauigkeit ist insbesondere problematisch, da in heutigen SOA-Umgebungen diese Rolle nicht mehr ausschließlich durch Verzeichnisse, sondern verstärkt durch dezentrale Vermittlungslösungen realisiert wird.

Das in Abbildung 2.2 dargestellte SOA-Rollenmodell beschreibt die Interaktion zwischen den einzelnen Rollen. Im ersten Schritt stellen Serviceanbieter entsprechende Dienste bereit und

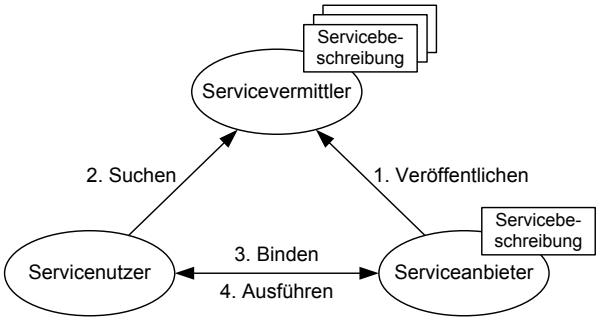


Abbildung 2.2: Interaktionen im SOA-Rollenmodell

registrieren diese beim Servicevermittler, um das Auffinden dieser Dienste zu ermöglichen. Bei dezentralen Vermittlungslösungen übernehmen die Dienstanbieter häufig gleichzeitig die Rolle des Vermittlers. Anschließend können Servicenutzer Dienste mithilfe der Servicevermittler suchen. Dabei erhalten die Nutzer die für die Nutzung eines Dienstes notwendigen Informationen. Somit ist der Nutzer in der Lage, sich dynamisch an den Serviceanbieter zu binden und den Dienst zu konsumieren.

2.2 Grundlagen des Geschäftsprozessmanagements

Das Geschäftsprozessmanagement ist heute weder aus der strategischen und taktischen noch aus der operativen Unternehmensführung wegzudenken. So ergaben verschiedene Umfragen unter IT-Entscheidungsträgern, die zur Mitte der letzten Dekade durchgeführt wurden, dass trotz rückläufiger Budgets und allgemeiner Kostenreduktionstrends die entsprechenden Unternehmen planten, ihre Ausgaben für das Geschäftsprozessmanagement zu steigern [59, 60, 84]. Das Ziel dieser Organisationen ist es, dem durch den derzeitigen Wettbewerbsdruck verursachten Kosten- und Anpassungsdruck durch flexible und optimierte wertschöpfende Geschäftsprozesse erfolgreich entgegenzuwirken.

Im Rahmen dieses Abschnitts wird der Themenbereich des Geschäftsprozessmanagements nach einer Einführung grundlegender Begriffe zuerst auf einer allgemeinen, sowohl betriebswirtschaftlichen als auch technisch-konzeptionellen Ebene betrachtet. Zum Abschluss dieses Grundlagenabschnitts werden die Zusammenhänge und Synergien von Geschäftsprozessmanagement und SOA diskutiert.

2.2.1 Einführung in das Geschäftsprozessmanagement

Mit dem Begriff des Prozess- bzw. Geschäftsprozessmanagements werden alle Maßnahmen beschrieben, die der Realisierung und Verwaltung sowie der Optimierung der Geschäftsprozesse und Workflows von Organisationen dienen. Bevor in diesem Abschnitt ein integriertes Konzept zum Geschäftsprozess- und Workflowmanagement vorgestellt wird, werden mit „Geschäftsprozess“ und „Workflow“ zuerst die beiden zentralen Begriffe dieses Forschungsbereichs eingeführt und voneinander abgegrenzt.

Geschäftsprozesse und Workflows

Der Begriff „Geschäftsprozess“ wird in der Literatur unterschiedlich definiert. Neben der Abgrenzung zum Begriff „Workflow“ ist in diesem Zusammenhang vor allem die Abgrenzung technischer Prozesse, z. B. Produktionsplanungs- und -durchführungsprozesse, von betriebswirtschaftlichen bzw. kaufmännischen Prozessen ein häufig diskutiertes Thema [16, 193]. Unstrittig ist jedoch, dass Geschäftsprozesse der Erstellung von Leistungen entsprechend der aus den Unternehmenszielen abgeleiteten Prozesszielen dienen. Die meisten Autoren vertreten jedoch die Meinung, dass letztlich technische genauso wie betriebswirtschaftliche Prozesse der Erreichung der Unternehmensziele dienen. Aus diesem Grund wird auch dieser Arbeit in Anlehnung an Gadatsch [61] und Österle [204] folgende nicht differenzierende Geschäftsprozessdefinition zugrunde gelegt.

Ein *Geschäftsprozess* ist eine zielgerichtete, zeitlich logische Abfolge von Aufgaben, die von mehreren Organisationen oder Organisationseinheiten arbeitsteilig, verteilt ausgeführt werden kann. Über diese konstituierenden Merkmale hinaus besitzen Geschäftsprozesse einen definierten Start- und Endpunkt sowie Eingaben (Ressourcen, Material und Informationen) und Ausgaben [78, 167]. Während der meist lang laufenden Prozessausführung können sie auf Ereignisse reagieren sowie selbst Ereignisse auslösen. Geschäftsprozesse sind hierarchisch aufgebaut. D. h., jede Aufgabe (auch Aktivität genannt) kann in weitere Teilaufgaben oder Subprozesse zerlegt werden. Können Aufgaben nicht weiter zerlegt werden, so wird von elementaren Aufgaben oder Aktivitäten gesprochen. Die Prozessausführung kann durch den Einsatz von Informations- und Kommunikationstechnologien unterstützt werden, sodass Geschäftsprozesse bzw. deren Aktivitäten automatisiert, teilautomatisiert oder manuell ausgeführt werden können.

Bei strikter Einhaltung der Terminologie des Geschäftsprozessmanagements müssen Workflows klar von Geschäftsprozessen unterschieden werden. Während ein Geschäftsprozess einen Prozess auf der Makroebene beschreibt, werden *Workflows* durch die sukzessive Zerlegung von Geschäftsprozessen in einer Mikroebene erreicht [204]. So wird in der Literatur ein Prozess dann als Workflow angesehen, wenn das Abstraktionsniveau seiner Beschreibung (das Workflowmodell) so detailliert ist, dass es technischen Handlungsvorschriften entspricht, die direkt ganz oder teilweise automatisiert ausgeführt werden können [227]. Becker et al. [15] und Galler et al. [62, 188, 191] ordnen Workflows klar der IT-Ebene zu. So sehen sie einen Workflow als einen Geschäftsprozess abbildendes Eingabe- und Regelwerk, das formal beschrieben ist und direkt in einem Workflowmanagementsystem bzw. in der Kontrollsphäre eines Anwendungssystems ausgeführt werden kann. Um dieses Merkmal realisieren zu können, muss ein Workflow zeitliche, fachliche und ressourcenbezogene Spezifikationen für die automatische Steuerung des Arbeitsablaufs eines Prozesses sowie formale Beschreibungen der Interaktionen mit anderen Workflows enthalten [61, 167]. Vereinfacht zusammengefasst kann demnach ein Workflow als informationstechnische Umsetzung von Geschäftsprozessen angesehen werden, dessen konkrete Ausführung als Workflowinstanz bezeichnet wird.

Da sowohl Geschäftsprozesse als auch Workflows Arbeitsabläufe beschreiben sowie eine prozessorientierte Sichtweise auf Systeme und Organisationen anstreben und auch bei strikter definitorischer Abgrenzung eine klare Differenzierung im realen Umfeld nicht immer möglich ist, werden diese Begriffe größtenteils synonym verwendet. Auch in dieser Arbeit wird nicht immer eine strikte Abgrenzung verfolgt.

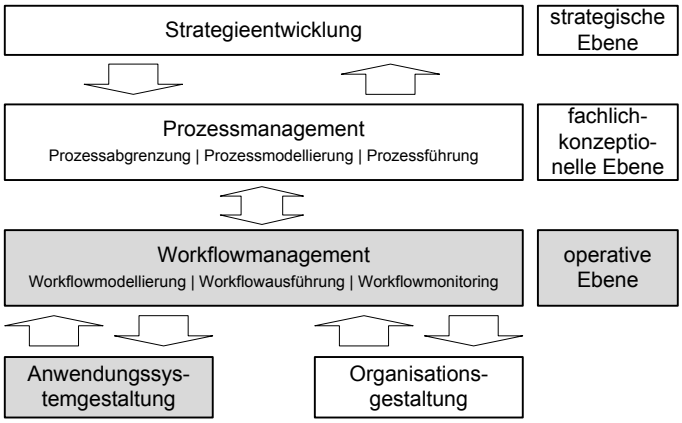


Abbildung 2.3: Ebenen eines integrierten Geschäftsprozess- und Workflowmanagements (Quelle: [61])

Integriertes Geschäftsprozess- und Workflowmanagement

In Unternehmen werden zum Abgleich mit den Unternehmensstrategien, der organisatorischen Gestaltung von Geschäftsprozessen sowie deren technischer Umsetzung mit geeigneten Informations- und Kommunikationssystemen bereits seit Anfang der 1990er Jahre integrierte Konzepte für das Geschäftsprozess- und Workflowmanagement eingesetzt. Abbildung 2.3 zeigt den Gestaltungsrahmen eines in der Literatur weitverbreiteten Konzepts zum integrierten Geschäftsprozess- und Workflowmanagement [61, 64], das auch dieser Arbeit zugrunde liegt.

Gadatsch [61] unterteilt das Konzept in mehrere Ebenen, die sich in wechselseitigen Beziehungen beeinflussen. Auf der strategischen Ebene werden die Geschäftsfelder eines Unternehmens betrachtet. Im Rahmen des Prozessmanagements auf der direkt unter der strategischen Ebene angeordneten fachlich-konzeptionellen Ebene erfolgt die Ableitung der Prozesse aus der Unternehmensstrategie. Diese Ebene umfasst die drei Phasen Prozessabgrenzung, -modellierung und -führung. Im Rahmen der Prozessabgrenzung werden in Abhängigkeit der Unternehmensstrategie und -ziele einzelne Aktivitäten bestimmten Prozessen zugeordnet. Die so ermittelten Prozesse werden in der zweiten Phase in Modelle überführt, die im Rahmen des Workflowmanagements in informationstechnisch ausführbare Workflows überführt werden. Durch die Erfassung und Bewertung von Kennzahlen eines Prozesses im Rahmen der Prozessführung wird die Erreichung der Prozessziele überprüft. Entsprechend der daraus resultierenden Ergebnisse kann die Notwendigkeit einer Anpassung des Prozesses oder gar der Unternehmensstrategie aufgezeigt werden.

Das Workflowmanagement auf der operativen Ebene besteht mit der Workflowmodellierung und -ausführung sowie dem Prozessmonitoring ebenfalls aus drei Phasen. Im Rahmen der Workflowmodellierung wird das während der Prozessmodellierung erstellte Geschäftsprozessmodell so in ein Workflowmodell überführt, dass es automatisch informationstechnisch in einem Workflowmanagementsystem ausgeführt werden kann (Workflowausführung). Analog zur Prozessführung werden auf der operativen Ebene die Kennzahlen des Workflows im Prozessmonitoring überwacht, deren Ergebnisse evtl. Anpassungen insbesondere der Workflows (also der operativen Ebene), aber auch der über- und untergeordneten Ebenen zur Folge haben können.

Entsprechend den Anforderungen des Workflowmanagements müssen die Anwendungssysteme auf der technischen Seite sowie die Organisationsstrukturen eines Unternehmens auf der betriebswirtschaftlichen Seite angepasst werden. Beide Bereiche beeinflussen im Gegenzug ihrerseits das Workflowmanagement. Insbesondere die Unternehmens-IT wird nicht nur als Mittel zur Umsetzung von Workflows gesehen. Vielmehr ist eine völlige Ausschöpfung der innovativen Informationsverarbeitung zum optimalen Einsatz von Geschäftsprozess- und Workflowmanagementmethoden unverzichtbar [76, 104]. Innovationen in der IT müssen also ganz neue Wettbewerbsvorteile generierende Geschäftsprozesse ermöglichen.

Im Rahmen dieser Arbeit wurden Lösungen entwickelt, die insbesondere in den in Abbildung 2.3 grau hervorgehobenen Elementen einzuordnen sind. So wurden ein Konzept und technische Umsetzungslösungen für ein ganzheitliches Workflowmanagement im Kontext ressourcenbeschränkter Geräte entwickelt, das auf der Anwendungssystemebene eine Webservice-basierte SOA verwendet (s. Abschnitt 3.4).

2.2.2 Geschäftsprozess- und Workflowmodellierung

Eine der zentralen Aufgaben eines Geschäftsprozessmanagements ist die Modellierung sowohl auf der fachlich-konzeptionellen als auch auf der operativen Ebene. Auf der fachlich-konzeptionellen Ebene dient die Geschäftsprozessmodellierung der Analyse der Ist- und Sollgeschäftsprozesse sowie deren Gestaltung, Optimierung und Dokumentation. Dazu werden in einem ersten Schritt Geschäftsprozessmodelle erstellt, die die Prozesse meist (semi-)formal beschreiben. Workflowmodelle dienen dagegen der detaillierten formalen Beschreibung von Workflows mit dem Ziel, diese in einem Workflowmanagementsystem (WFMS) ausführen oder automatisiert in entsprechende Anwendungssoftware überführen zu können. Sie werden typischerweise durch Verfeinerung aus Geschäftsprozessmodellen abgeleitet. Je nach Grad der Detaillierung und der technischen Informationen ist eine manuelle Verfeinerung der Geschäftsprozessmodelle während der Transformation zum Workflowmodell notwendig. Ziel ist es jedoch, diese Modell-zu-Modelltransformation weitestgehend automatisiert durchzuführen.

Damit ist die Prozessmodellierung sehr eng mit der *modellgetriebenen Softwareentwicklung* (MDSD) verwandt. MDSD wird nach Stahl et al. als Oberbegriff für Techniken definiert, die aus formalen Modellen automatisiert lauffähige Software erzeugen [201]. Grundlage einer jeden Prozessmodellierung sind formale Modelle, die im Kontext einer Domäne, d. h. in einem begrenzten Interessen- und Wissensgebiet stehen. In einem Metamodell wird diese Anwendungsdomäne formal beschrieben. Es umfasst die abstrakte Syntax und die statische Semantik einer Modellierungssprache oder einer domänenspezifischen Sprache (engl.: *Domain Specific Language*, DSL). In diesem Zusammenhang beschreibt die abstrakte Syntax die Elemente eines Metamodells sowie ihre Beziehungen untereinander. Die statische Semantik beschreibt starre Regeln, die ein Modell erfüllen muss, damit es wohlgeformt ist. Neben der abstrakten Syntax und der statischen Semantik bestehen DSLs und Prozessmodellierungssprachen aus einer Spezifikation der Bedeutung von Sprachelementen, der dynamischen Semantik sowie aus einer konkreten Syntax [201].

Formale Geschäftsprozess- und Workflowmodellierungsprachen sind letztlich domänenspezifische Sprachen. Mit einer textuellen oder einer grafischen Syntax (Diagrammsprachen) können DSLs durch zwei verschiedene Arten konkreter Syntax repräsentiert werden. Textuelle

Sprachen zeichnen sich durch eine an Programmiersprachen angelehnte Notation aus, die eine sehr hohe Präzision der Modellspezifikation ermöglicht. Sie sind jedoch in der Regel nicht zur Verwendung durch Fachpersonal geeignet, da sie wesentlich komplexer als Diagrammsprachen sind. Diagrammsprachen sind im Gegensatz dazu durch eine hohe Anschaulichkeit und eine einfache sowie intuitive Modellierung charakterisiert [61, 201]. Entsprechend wurden in der Vergangenheit eine Reihe von Sprachen entwickelt, die eine grafische Prozessmodellierung erlauben. Sie stellen in der Regel die Prozesse in datenfluss-, kontrollfluss- oder objektorientierten Graphen dar [61].

Die wohl federführende Methode zur fachlich-konzeptionellen Modellierung von Geschäftsprozessen ist die von Keller et al. [95, 96] auf Basis von Petri-Netzen [170] entwickelte *ereignisgesteuerte Prozesskette* (EPK). Aufgrund ihrer Einfachheit eignet sie sich insbesondere für die Diskussion zwischen Fach- und IT-Personal. EPKs ermöglichen eine kontrollflussorientierte Modellierung und sind zentraler Bestandteil der von Scheer vorgeschlagenen *Architektur Integrierter Informationssysteme* (ARIS, [189]) sowie die zentrale Prozessmodellierungssprache im Enterprise Resource Planning System SAP R/3 [184, 187, 188, 190].

Eine weitere bedeutende Prozessmodellierungssprache ist die *Business Process Modeling Notation* (BPMN, [148, 165, 223]). Sie ist ebenfalls eine kontrollflussorientierte Sprache und wird von der Object Management Group (OMG)[2] standardisiert und weiterentwickelt. Das Designziel von BPMN war eine leicht verständliche grafische Notation für die Modellierung von Geschäftsprozessen, die einerseits auf der fachlich-konzeptionellen Modellierungsebene die einfache Definition und Analyse von Geschäftsprozessen ermöglicht. Gleichzeitig sollen BPMN-Modelle durch IT-Personal um informationstechnische Aspekte ergänzt werden, sodass eine möglichst automatische Transformation in einen ausführbaren Workflow möglich ist. Im Vergleich zu EPKs bietet BPMN deshalb differenzierte Objekte sowohl zur Beschreibung fachlicher Aspekte als auch für eine spätere informationstechnische Ausführung. Dadurch können BPMN-Modelle komplexer, aber auch ausdrucksstärker als EPKs sein. Damit zielt BPMN darauf ab, die Kluft zwischen fachlich-konzeptioneller und operativer Geschäftsprozessmodellierung zu überwinden.

Neben speziell für die Prozessmodellierung entwickelten Sprachen wie EPK und BPMN werden im Geschäftsprozessmanagement auch Sprachen aus der Softwaretechnik eingesetzt. So wird z. B. die von der OMG für die Modellierung von Software und anderen Systemen entwickelte und standardisierte *Unified Modeling Language* (UML, [146]) auch zur Modellierung von Geschäftsprozessen und Workflows verwendet. UML definiert zur Modellierung ein Metamodell und eine abstrakte Syntax sowie eine statische und dynamische Semantik. Die konkrete Beschreibung erfolgt über unterschiedliche Diagramme als konkrete Syntax von UML. Zur Geschäftsprozess- und Workflowmodellierung werden neben Anwendungsfall- und Aktivitätsdiagrammen insbesondere Zustandsdiagramme verwendet [61].

Neben den hier aufgeführten allgemeinen Standards zur Prozessmodellierung hat sich bedingt durch unterschiedliche Anwendungsdomänen über die Jahre eine Vielzahl an weiteren Prozessbeschreibungssprachen entwickelt. Nach Meinung des Autors dieser Arbeit wird in Zukunft neben der Verwendung von Standardsprachen durch den Einsatz von Konzepten

[2] Die Object Management Group ist ein über 800 Mitglieder umfassendes Konsortium, das sich mit der Entwicklung von Standards im Bereich der objektorientierten Programmierung und des Geschäftsprozessmanagements beschäftigt.

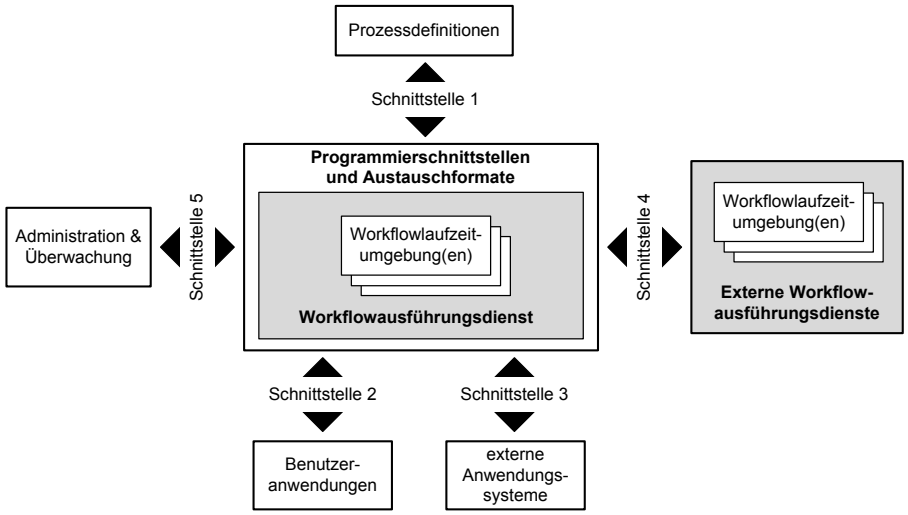


Abbildung 2.4: WfMC-Referenzmodell eines Workflowmanagementsystems (übersetzt aus: [226])

und Technologien der modellgetriebenen Softwareentwicklung eine Vielzahl an weiteren Spezialsprachen entstehen. Diese bilden keine Standards, die von Organisationen wie der OMG entwickelt und standardisiert werden. Vielmehr werden sie von einzelnen Unternehmen für den unternehmensinternen Einsatz entwickelt. Sie sind absolut auf die bestimmten Aufgaben und Umgebungen spezialisiert und nicht allgemein anwendbar.

2.2.3 Workflowmanagementsysteme

Zur Unterstützung der Modellierung, Ausführung und Überwachung sowie Administration von Geschäftsprozessen auf der operativen Workflowebene werden *Workflowmanagementsysteme* (WFMS) eingesetzt. Sie sind anwendungsunabhängige, dem Middleware-Bereich zuzuordnende Softwaresysteme. Die zentrale Aufgabe eines WFMS ist die Steuerung der Ausführung der in einem formalen Workflowmodell spezifizierten Prozesslogik. Dazu muss ein WFMS zum einen die Durchführung der einzelnen Prozessaktivitäten durch menschliche oder maschinelle Aufgabenträger koordinieren sowie ggf. zusätzliche Informationen und Anwendungsprogramme in die Prozessausführung integrieren [62, 145, 192].

Zur Systematisierung und Abgrenzung einzelner Komponenten eines WFMS sowie zur Etablierung eines einheitlichen Begriffssystems hat die *Workflow Management Coalition* (WfMC)[3] ein Referenzmodell für Workflowmanagementsysteme definiert [226]. Abbildung 2.4 zeigt die entsprechenden Modellkomponenten sowie ihre Beziehungen zueinander. Die zentrale Systemkomponente ist der *Workflowausführungsdienst* (engl.: Workflow Enactment Service), der aus mehreren Workflowlaufzeitumgebungen besteht. Die Laufzeitumgebungen sind Softwarekomponenten, die die Prozessausführung steuern. Sie generieren Prozessinstanzen und führen diese unter der Einbeziehung weiterer Systemkomponenten aus. Die Interaktion

[3] Die Workflow Management Coalition ist ein Zusammenschluss von Anwendern, Herstellern und Forschungseinrichtungen aus dem Gebiet des Geschäftsprozess- und Workflowmanagements.

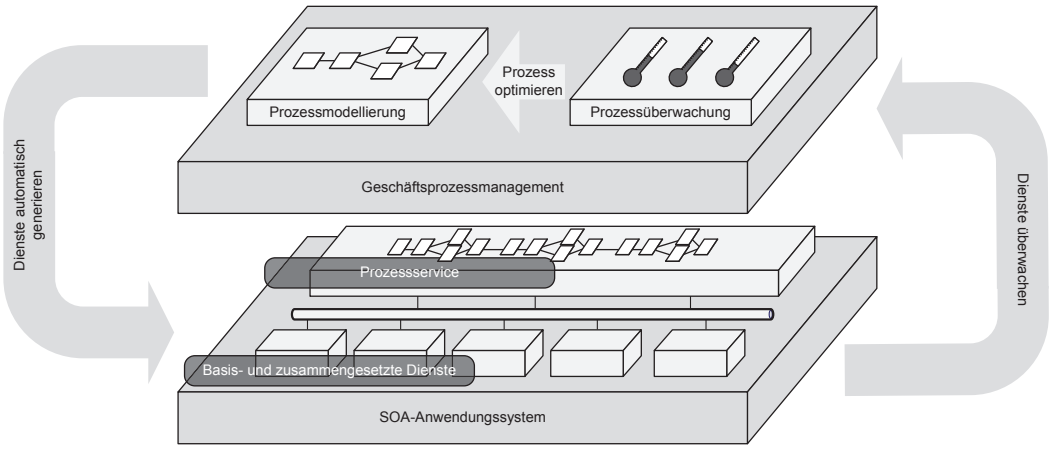


Abbildung 2.5: Geschäftsprozessmanagement auf Basis serviceorientierter Architekturen

mit diesen Komponenten erfolgt über fünf Programmierschnittstellen sowie verschiedene Austauschformate.

Über die erste Schnittstelle werden Prozessmodellierungswerkzeuge und -sprachen an das WFMS gebunden und somit die Ausführung entsprechender Prozessmodelle bzw. *Prozessdefinitionen* ermöglicht. Mit standardisierten *Benutzeranwendungen* wird über die zweite Schnittstelle die Interaktion mit menschlichen Aufgabenträgern spezifiziert. Mit den Schnittstellen 3 und 4, den *externen Anwendungssystemen* und den *externen Workflowausführungsdiensten* ist es möglich, Prozessaktivitäten von maschinellen Aufgabenträgern durchführen zu lassen, die nicht vom internen Ausführungsdienst verwaltet bzw. bereitgestellt werden. Mit der *Administration* und *Überwachung* als fünfte Komponente wird die Anbindung von Werkzeugen zur Verwaltung der Workflowmanagementsysteme und einzelner Prozesse sowie deren Überwachung ermöglicht.

2.2.4 Geschäftsprozessmanagement auf Basis serviceorientierter Architekturen

Bei der informationstechnischen Umsetzung von Geschäftsprozessen stehen Unternehmen sowohl vor fachlichen als auch vor technischen Herausforderungen. Entsprechend muss zur Prozessmodellierung und -ausführung ein umfassender, ganzheitlicher Ansatz verfolgt werden [202, 214]. Abbildung 2.5 veranschaulicht einen solchen Ansatz, der sich dem operativen Geschäftsprozessmanagement zur fachlichen Definition, Überwachung und Optimierung von Geschäftsprozessen bedient und ihre informationstechnische Umsetzung durch Workflowmanagement- und Anwendungssysteme realisiert, die dem serviceorientierten Paradigma folgen.

Aus dem IT-Blickwinkel betrachtet stellen nach dem SOA-Konzept realisierte Anwendungssysteme eine optimale Systemarchitektur zur informationstechnischen Umsetzung von Geschäftsprozessen dar. Im Kontext des Geschäftsprozessmanagements liegt die Motivation von Serviceorientierung in der informationstechnischen Umsetzung einzelner Geschäftsprozessaktivitäten und Subprozesse durch verteilte Dienste und Dienstaggregate. So können elementare Prozessaktivitäten oder der Zugriff auf externe Anwendungssysteme genauso über Dienste

(meist Basis- oder zusammengesetzte Dienste) realisiert werden wie die Integration ganzer Subprozesse (meist als Prozessservices bereitgestellt). Im SOA-Umfeld können Geschäftsprozesse als eine Menge von Serviceaufrufen realisiert werden, die einer logischen und zeitlichen Reihenfolge unterliegen und selbst wieder als Dienste bereitgestellt werden.

Indem Anwendungssysteme entsprechend dem SOA-Konzept gestaltet werden, wird zwar aufgrund der Kapselung von Funktionalität sowie der Wiederverwendbarkeit und Kompositionsfähigkeit von lose gekoppelten Diensten die für die Umsetzung von Geschäftsprozessen notwendige Flexibilität erreicht. Prozesse mit ihren komplexen Kontrollflüssen und Interaktionen müssen aber trotzdem durch IT-Personal durch die Verwendung von Programmiersprachen der 3. Generation realisiert werden. Genau diese Schwäche wird vom Geschäftsprozessmanagement kompensiert. Es realisiert den entscheidenden Erfolgsfaktor bei der Dienstkomposition [124,125]. Durch grafische Prozessmodellierungssprachen wie EPK oder BPMN wird eine einfache und domänenspezifische Definition von Prozessmodellen durch die Komposition von Diensten ermöglicht. Diese Vorgehensweise hat eine Vielzahl von Vorteilen. So erlaubt eine Abstraktion die formale Modellierung sowie die Prozessüberwachung und -optimierung aus einer fachlichen Perspektive. Entsprechende Sprachen und Werkzeuge sind optimal auf das Domänenproblem ausgerichtet und können direkt von Fachpersonal ohne Paradigmenbruch verwendet werden.

Aus formalen und auf Korrektheit überprüften Prozessmodellen lassen sich durch Interpretation oder Codeerzeugung automatisch Dienste generieren. Dadurch wird eine sehr hohe Entwicklungsgeschwindigkeit und Softwarequalität erreicht. So müssen vom Fachpersonal erstellte Prozessmodelle nicht manuell und damit zeit- und kostenaufwendig durch IT-Personal in Software realisiert werden. Auch werden Fehler im Realisierungsprozess vermieden. So lassen sich bei einer manuellen Realisierung von Geschäftsprozessen durch IT-Personal Fehler durch auf den Business-IT-Gap zurückzuführende Fehlinterpretationen von Prozessmodellen genauso wenig vermeiden wie menschliche Fehler jeglicher Art. Nicht weniger fehleranfällig ist die bei einer manuellen Implementierung häufig charakteristische Verwendung von unterschiedlichen Paradigmen für die Prozessdefinition (geschäftsprozessorientiert) und -implementierung (häufig objektorientiert).

Der Entwicklungs- bzw. Lebenszyklus von Geschäftsprozessen schließt mit der fachlichen Prozessüberwachung und der daraus resultierenden Prozessoptimierung. Dazu fließen Daten, die bei der Ausführung von Diensten erhoben wurden, in die fachliche Prozessüberwachung ein.

2.3 Grundlagen von Webservices

In den Abschnitten 2.1 und 2.2 wurden serviceorientierte Architekturen und Aspekte des Geschäftsprozessmanagements auf einer konzeptionellen Ebene betrachtet. Es wurde gezeigt, wie mithilfe von Prozessmodellierungsmethoden und -werkzeugen Dienste eines Serviceinventars zu Geschäftsprozessen zusammengesetzt werden und wie aus Geschäftsprozessmodellen automatisch Dienste generiert werden können, die diese Prozesse informationstechnisch in einer SOA realisieren.

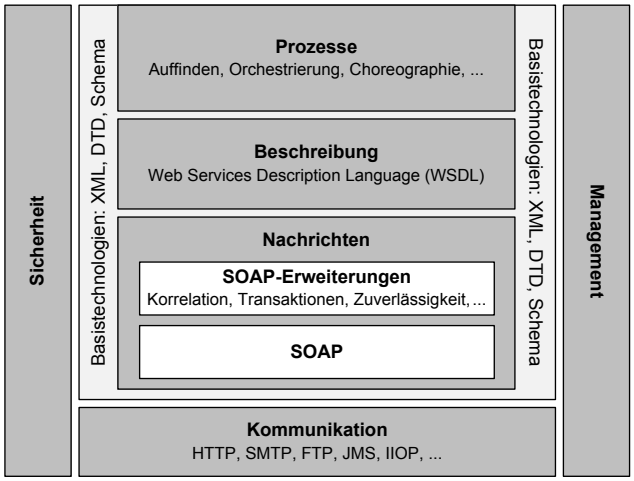


Abbildung 2.6: Abstrakte Schichten und konkrete Spezifikationen der Webservice-Architektur (übersetzt aus [238])

SOA beschreibt jedoch lediglich ein Architekturkonzept zum Design verteilter Anwendungssysteme. Zur konkreten Umsetzung können unterschiedliche Middleware-Technologien eingesetzt werden. Eine Middleware bezeichnet eine anwendungsneutrale Softwarekomponente zwischen dem Betriebssystem und einer Anwendung. Sie dient der Realisierung einer Verteilungstransparenz, also dem Verbergen der Verteiltheit des Systems vor dem Entwickler [213]. Serviceorientierte Architekturen lassen sich mit verschiedenen Middleware-Technologien, wie z. B. DCOM (*Distributed Component Object Model*, [80]), RMI (*Remote Method Invocation*, [206]) oder CORBA (*Common Object Request Broker Architecture*, [147]) implementieren. Während der Einsatz von DCOM sowie RMI durch ihre Plattformabhängigkeit nur in homogenen Systemen geeignet ist, verhindert die Komplexität von CORBA dessen verbreitete Verwendung [130]. Die zurzeit erfolgversprechendste Umsetzungstechnologie stellen Webservices dar, da sie auf offenen Standards basieren und eine flexible und plattformunabhängige Kommunikation zwischen Anwendungen und Diensten ermöglichen [29, 52, 94].

In diesem Abschnitt werden als Erstes das Zusammenspiel des Webservice-Architekturkonzepts und seiner Umsetzungen beschrieben, bevor die Basisstandards im Einzelnen vorgestellt werden. Der Abschnitt schließt mit der Einführung von Standards zur Geschäftsprozessrealisierung auf Basis von Webservices.

2.3.1 Webservice-Architektur

Für den Begriff *Webservice* gibt es eine Vielzahl an Definitionen. Sie orientieren sich meistens an den unterschiedlichen Einsatzszenarien und verwendeten Technologien. In [238] definiert das (W3C) ein Architekturkonzept (WS-*-Architektur), das die Charakteristika von Webservices unabhängig von konkreten Technologien oder Standards beschreibt. Abbildung 2.6 zeigt in einem Schichtenmodell (dem sog. *Web Services Technology Stack*) die bei der Implementierung und Nutzung von Webservices relevanten abstrakten Komponenten. Zusätzlich zur abstrakten Sichtweise zeigt das Modell einige Optionen zur konkreten technischen Ausgestaltung der jeweiligen Schichten.

Die Differenzierung von abstrakter und konkreter Architektur ermöglicht auf den einzelnen Schichten den Einsatz unterschiedlicher Realisierungsstandards, wodurch ein sehr hohes Modularitätsniveau und daraus resultierend eine hohe Flexibilität erreicht wird. Dadurch kann die Webservice-Technologie optimal an unterschiedlichste Anwendungsszenarien angepasst werden. Insbesondere das W3C und OASIS (*Organization for the Advancement of Structured Information Standards*[4]) haben mittlerweile über 50 offene Standards zur konkreten Ausgestaltung der Schichten der Webservice-Architektur spezifiziert [94].

Auch wenn die Verwendung von offenen Standards nicht zwingend notwendig ist, so werden diese jedoch zur Gewährleistung von Interoperabilität empfohlen. Offen bedeutet in diesem Zusammenhang, dass jegliche Informationen, die zur Implementierung der Standards notwendig sind, frei verfügbar sein müssen und keinen Nutzungseinschränkungen unterliegen dürfen. Aber auch bei der Verwendung von offenen Standards kann die große Flexibilität von Webservices zu Interoperabilitätsproblemen führen. Dieser Problematik nimmt sich die WS-I[5] an. Sie spezifiziert sog. Profile, die beschreiben wie bestimmte Standards verwendet werden müssen, um Interoperabilität zu gewährleisten.

Im Folgenden werden die einzelnen Schichten der Webservice-Architektur und entsprechende Realisierungsstandards sowie das Zusammenspiel der einzelnen Schichten betrachtet. Die unterste Schicht realisiert den Austausch von Webservice-Nachrichten zwischen Dienstnutzer und -anbieter. Üblicherweise werden für den Transport weitverbreitete Internetprotokolle, allen voran HTTP (*Hypertext Transfer Protocol*, [17, 54, 251]), aber auch SMTP (*Simple Mail Transfer Protocol*, [100, 233]) oder FTP (*File Transfer Protocol*, [177]), verwendet. Somit kann auf eine oftmals bereits bestehende Internetinfrastruktur aufgebaut werden. Darüber hinaus ist aber auch der Einsatz von beliebigen weiteren verfügbaren Anwendungsprotokollen wie IIOP (CORBAs *Internet Inter-ORB Protocol*, [147]) oder verschiedene Message-Queing-Technologien wie JMS (*Java Message Service*[6], [207,258]) genauso möglich und sinnvoll für den Webservice-Transport, wie die direkte Verwendung von Transportschichtprotokollen wie TCP (*Transmission Control Protocol*, [176, 208]) oder UDP (*User Datagram Protocol*, [161, 174]). Auch die Integration völlig neuer Transportprotokolle in das Schichtenmodell ist vorgesehen und gewünscht. Die Wahl des richtigen Protokolls hängt somit vor allem von seiner Eignung für die Erfüllung der Anforderungen des Anwendungsszenarios sowie von einer evtl. bereits bestehenden Infrastruktur ab.

Für alle oberhalb der Kommunikationsschicht gelegenen Schichten des Webservice-Technologiestapels bilden XML (*Extensible Markup Language*, [244]) und verwandte Technologien, wie *XML-Schema* [239] (selten auch *Document Type Definition* [244]) sowie XML-Namensräume [259], die Grundlage. Die Protokolle zum Austausch von Nachrichten und die Nachrichten selbst basieren ebenso auf XML wie die Beschreibung von Schnittstellen und Prozessen.

Direkt auf der Kommunikationsschicht baut die Nachrichtenschicht auf. Die Webservice-Architektur des W3C definiert zwar ein abstraktes Nachrichtenmodell, das prinzipiell be-

[4] OASIS ist eine nicht gewinnorientierte Organisation, die sich mit der Entwicklung von E-Business und Webservice-Standards beschäftigt [152].

[5] Die WS-I (*Web Services Interoperability Organization*) war ein Industriekonsortium, das sich mit Interoperabilität von Webservice-Standards beschäftigte [218]. Sie wurde mittlerweile Teil von OASIS.

[6] JMS ist streng genommen eine Programmierschnittstelle für ein Kommunikationsprotokoll. Es wird jedoch häufig mit einem Kommunikationsprotokoll gleichgesetzt.

liebige Protokolle zulässt. Allerdings wird vom W3C auf dieser Ebene der Einsatz des in Abschnitt 2.3.2 näher vorgestellten Nachrichtenprotokolls SOAP [256] empfohlen. Auch in der Praxis hat sich SOAP mit seinen Erweiterungen zum Austausch von Webservice-Nachrichten durchgesetzt.

SOAP-Erweiterungen sind vor allem aufgrund der Transportprotokollunabhängigkeit von Webservices und SOAP notwendig. Sie hat zur Folge, dass Funktionen wie Sicherheit (z. B. durch *Web Services Security* [158]), Korrelation und Adressierung (z. B. durch *Web Services Addressing* [245]) sowie Transaktionen und Zuverlässigkeit (z. B. durch *Web Services Atomic Transaction* [162], *Web Services Reliable Messaging* [164]) nicht immer vom verwendeten Transportprotokoll bereitgestellt werden und deshalb auf Nachrichtenebene realisiert werden müssen.

Oberhalb der Nachrichtenschicht definiert die Beschreibungsschicht die Schnittstelle eines Webservices. Als Beschreibungssprache hat sich die in Abschnitt 2.3.2 näher beschriebene *Web Services Description Language* (WSDL, [232]) etabliert. Sie beschreibt maschinenlesbar die Struktur sowie Folge von Webservice-Nachrichten und wie diese über ein Nachrichten-Backbone auszutauschen sind. Mit Richtlinien, die z. B. über WS-Policy [255] und ergänzende Standards definiert werden, können Webservice-Beschreibungen um nicht funktionale Eigenschaften erweitert werden.

In der Prozessschicht als oberste Ebene des Schichtenmodells werden sämtliche Prozesse zusammengefasst, die sich mit dem Zusammenspiel von Diensten befassen. Dazu gehören z. B. die in Abschnitt 2.3.2 genauer erläuterten Standards zum Auffinden von Diensten sowie die in Abschnitt 2.3.3 beschriebenen Standards zur Komposition einzelner Webservices zu Dienstaggregaten und Geschäftsprozessen.

Die Funktionen *Sicherheit* und *Management* erstrecken sich über alle Schichten des Modells, da sie auf allen Ebenen relevant sind oder in unterschiedlichen Schichten realisiert werden können. So kann eine Verschlüsselung z. B. auf der Transportebene (*Transport Layer Security*, TLS [39]) genauso wie auf der Nachrichtenebene (z. B. durch *Web Services Security*, [158]) realisiert werden. Darüber hinaus können mit *WS-SecurityPolicy* [160] Webservice-Beschreibungen um Sicherheitsrichtlinien erweitert werden.

Das Management befasst sich mit der Verwaltung und Überwachung von Diensten. In diesem Zusammenhang sind die Standards *Web Services for Management* [40], *Web Services Distributed Management* [155] sowie *Management of Web Services* [157] zu nennen.

Aus der abstrakten Architektur und den konkreten technischen Umsetzungen hat das W3C folgende in der Literatur und Unternehmenspraxis weitverbreitete Definition eines Webservices abgeleitet, die auch dieser Arbeit zugrunde liegt [238]:

> „A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.“

Alternativ zur WS-*-Architektur gewinnen zurzeit auch Webservices an Bedeutung, die dem REST-Paradigma folgen. REST (*Representational State Transfer*, [55]) bezeichnet einen Architekturstil, dessen Basiseinheit die Ressource darstellt. Jede Ressource kann eindeutig identifiziert werden und erlaubt den Zugriff über eine uniforme Schnittstelle auf Basis einer zustandslosen Kommunikation.

RESTful-Webservices werden in der Regel auf Basis von Technologien des *World Wide Web* umgesetzt. So erfolgt die Adressierung der Ressourcen über URIs (*Uniform Resource Identifier*, [18]), während HTTP als Transportprotokoll verwendet wird. Die HTTP-Operationen stellen dabei die uniforme Zugriffsschnittstelle auf die Ressourcen dar.

Die bedeutenden Geschäftsprozessrealisierungsstandards (s. Abschnitt 2.3.3) bauen auf der WS-*-Architektur auf und unterstützen keine RESTful-Webservices. Deshalb spielt diese Webservice-Art für die vorliegende Arbeit keine Rolle und wird auch nicht näher betrachtet.

2.3.2 Basistechnologien

Wie im SOA-Rollenmodell in Abschnitt 2.1.4 beschrieben, sind die grundlegenden Funktionen einer SOA die Kommunikation zwischen Dienstnutzer und -anbieter sowie die Beschreibung und das Vermitteln von Diensten. In diesem Abschnitt werden Standards einführend erläutert, die diese Funktionen in einer auf Webservices basierenden SOA ausfüllen.

SOAP

SOAP [256] beschreibt ein Format zum Austausch von Nachrichten zwischen Webservices sowie allgemeine Regeln, wie diese Nachrichten über nahezu beliebige Transportprotokolle ausgetauscht werden können. Damit nimmt SOAP im SOA-Rollenmodell die Funktion der Kommunikation zwischen Dienstanbieter und -nutzer ein.

Während SOAP bis einschließlich Version 1.1 [231] als Abkürzung für *Simple Object Access Protocol* stand, wurde diese Abkürzung ab Version 1.2 [249] verworfen, da SOAP weder einfach ist, noch den Zugriff auf Objekte ermöglicht. Wegen seines hohen Bekanntheitsgrads wurde SOAP aber als Protokollname beibehalten.

Da SOAP 1.1 und SOAP 1.2 sich grundsätzlich nur geringfügig unterscheiden und einen ähnlichen Funktionsumfang bieten, finden zurzeit beide Versionen Verwendung in der Unternehmenspraxis. Aber gerade für die Verwendung in extrem ressourcenbeschränkten Umgebungen weist SOAP 1.2 aufgrund seiner vollständigen Transportprotokollunabhängigkeit und der Flexibilität in der Nachrichtenserialisierung Vorteile gegenüber der Version 1.1 auf, sodass dieser Arbeit SOAP 1.2 zugrunde liegt. Für Details zu den Unterschieden zwischen beiden SOAP-Versionen sei auf eine Übersicht des W3C [234] sowie die Spezifikation von SOAP 1.2 [249] verwiesen.

Die SOAP-1.2-Dokumentation und -Spezifikation besteht aus drei Dokumenten. Im Gegensatz zu Teil 1 (*Messaging Framework*, [250]) und Teil 2 (*Adjuncts*, [251]), die normativen Charakter haben, ist Teil 0 [249] als deskriptives Informationsdokument gedacht.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <soap12:Envelope xmlns:soap12="http://www.w3.org/2003/05/soap-envelope">
3    <soap12:Header xmlns:wsa="http://www.w3.org/2005/08/addressing">
4      [...]
5      <wsa:FaultTo soap12:mustUnderstand="1">
6        <wsa:Address>http://host.itm.uni-luebeck.de/FaultHandler</wsa:Address>
7      </wsa:FaultTo>
8    </soap12:Header>
9    <soap12:Body>
10     <sendPosition xmlns="http://www.itm.uni-luebeck.de/glombitza/PositionWS">
11       <timestamp>2010-09-07T22:05:26.609375+02:00</timestamp>
12       <latitude>53.836533</latitude>
13       <longitude>10.70338</longitude>
14     </sendPosition>
15   </soap12:Body>
16 </soap12:Envelope>
```

Quelltext 2.1: Beispiel einer SOAP-Nachricht

Struktur von SOAP-Nachrichten Der Aufbau einer SOAP-Nachricht wird im ersten Teil der Spezifikation beschrieben und ist exemplarisch im Quelltext 2.1 als XML-Serialisierung dargestellt. Sie besteht aus einem Rahmendokument, dem sog. *Envelope* (Zeile 2 bis 16). Der Envelope enthält einen optionalen *Header-* (Zeile 3 bis 8) und obligatorischen *Body-Block* (Zeile 9 bis 15). Der Body enthält applikationsspezifische Daten. In Teil 2 der SOAP-Spezifikation wird dazu allgemein festgelegt, wie z. B. entfernte Methodenaufrufe (engl.: *Remote Procedure Calls*, RCP) im Body (*rpc/encoded*) oder aber XML-Dokumente direkt (*document/literal*) codiert werden. Welche Codierungsart in einer konkreten Webservice-Interaktion verwendet wird, muss in der Schnittstellenbeschreibung des Dienstes definiert werden. In dem hier aufgeführten Beispiel wird in den Zeilen 10 bis 14 eine Positionsmeldung bestehend aus einem Zeitstempel und der eigentlichen Positionsangabe an den Empfänger übermittelt. Als Codierung wird dabei *document/literal* verwendet.

Der optionale Header enthält Informationen, die nicht anwendungsspezifisch sind und von sog. Handlern (dt.: Steuerungsprogrammen) verarbeitet werden. Handler übernehmen in diesem Zusammenhang Aufgaben wie z. B. Routing, Logging, Verschlüsselung, Transaktionsverwaltung oder Abrechnung. Der Inhalt des Headers wird in einzelnen *Header-Blöcken* gruppiert und ist nicht in SOAP spezifiziert. Dabei können einzelne Blöcke mit Attributen aus dem SOAP-Namensraum versehen werden, die beschreiben, ob ein Handler einen entsprechenden Block verstehen und verarbeiten muss. Darüber hinaus kann über Rollen beschrieben werden, an welchen Handler bestimmte Header-Blöcke gerichtet sind. In Quelltext 2.1 werden im *Header* Elemente aus *Web Services Addressing* (WS-Addressing, [245]) zur Adressierung bzw. zum Routen von SOAP-Nachrichten im Fehlerfall verwendet (Zeile 5 bis 7).

SOAP-Nachrichten werden üblicherweise mit XML serialisiert. Während in der Version 1.1 eine XML-Serialisierung zwingend vorgeschrieben ist, definiert SOAP 1.2 Nachrichten als sog. *XML-Infosets* [235]. Damit wird ein abstraktes Modell einer SOAP-Nachricht spezifiziert, das neben XML auch weitere Serialisierungsarten zulässt. Gerade im Umfeld ressourcenbeschränkter Geräte ist die Verwendung von Verfahren, die im Vergleich zu XML eine wesentlich geringere Nachrichtengröße erreichen, sinnvoll oder gar unverzichtbar.

Transport von SOAP-Nachrichten Im ersten Teil der SOAP-Spezifikation werden mit dem *SOAP Binding Framework* allgemeine Regeln definiert, wie SOAP-Nachrichten an ein Transportprotokoll zu koppeln sind. Eine Kopplung wird *(Transport-)Binding* genannt und beschreibt die Codierung von SOAP-Nachrichten und Regeln, wie diese über ein bestimmtes Transportprotokoll ausgetauscht werden. Damit ist die Grundlage dafür geschaffen, dass der Webservice-Technologiestapel, wie in Abschnitt 2.3.1 beschrieben, auf der Transport- und Nachrichtenebene beliebig erweitert werden kann, indem Transport-Bindings für beliebige Transportprotokolle mit unterschiedlichen Codierungs- und Serialisierungsarten von jedem spezifiziert werden können. Mit HTTP+SOAP wurde bereits vom W3C im zweiten Teil der SOAP-Spezifikation ein solches Binding für HTTP definiert. Mit LTP+SMC wurde im Rahmen dieser Arbeit ein SOAP-Binding entwickelt, das eine Webservice-Kommunikation in ressourcenbeschränkten Umgebungen realisiert (s. Kapitel 4).

Web Services Addressing Obwohl SOAP ein Protokoll für die Kommunikation in heterogenen Systemen ist, werden Informationen bzgl. der Identität und Adressierung von Nachrichten nicht in SOAP-Nachrichten codiert. Stattdessen werden Mechanismen des jeweilig verwendeten Transportprotokolls verwendet. Als Folge entsteht eine Verletzung der SOA-Eigenschaft der losen Kopplung, da somit Webservices enger an ein Transportprotokoll gebunden werden. Darüber hinaus bieten die meisten Transportprotokolle Adressierungsmechanismen, die komplexen Anwendungsszenarien oder teilweise sogar elementarer Webservice-Kommunikation nicht genügen. So lassen sich z. B. mit HTTP Fehlernachrichten nur an den aufrufenden Konsumenten und nicht an einen beliebigen Endpunkt schicken. Protokolle wie TCP oder UDP erlauben nicht einmal im Standardszenario die Adressierung eines Endpunkts.

Aus diesen Gründen wurde mit WS-Addressing (*Web Services Addressing*, [245]) ein mächtiger und transportprotokollunabhängiger Adressierungsmechanismus spezifiziert. WS-Addressing definiert eine Menge von XML-Infoset-Elementen, die eine umfangreiche Adressierung von SOAP-Nachrichten und eine Beschreibung ihrer Identität ermöglichen. Dabei verwendet WS-Addressing die oben beschriebene Möglichkeit der Erweiterung von SOAP über die Definition eigener *Header*-Elemente.

Die wichtigste Aufgabe von WS-Addressing ist die Adressierung von Webservice-Endpunkten. Dabei beschränkt sich WS-Addressing bei der Beschreibung von Endpunkten nicht wie die meisten Transportprotokolle auf einfache URLs (*Uniform Resource Locator*, [19]). Stattdessen werden sog. Endpunktreferenzen verwendet. Endpunktreferenzen können neben einer URI noch zusätzliche Informationen, z. B. zur Beschreibung der vorausgesetzten Dienstgüte, enthalten.

WS-Addressing definiert vier verschiedene Elemente zur Beschreibung von Endpunktreferenzen. *To* enthält die Empfängerreferenz eines Webservices. *From* referenziert den Sender einer Nachricht. Mit *ReplyTo* und *FaultTo* kann festgelegt werden, dass Antwort- bzw. Fehlernachrichten nicht an den ursprünglichen Sender, sondern an einen anderen Endpunkt gesendet werden sollen. Während *To* verpflichtend ist, sind *From*, *ReplyTo* und *FaultTo* optional.

Neben dem Endpunkt muss jedoch auch die Webservice-Operation identifiziert werden. So kann über das verpflichtende *Action*-Element eine in WSDL definierte Eingabe-, Ausgabe- oder Fehlernachricht referenziert werden.

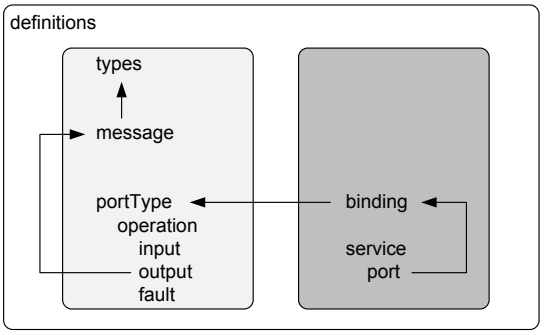


Abbildung 2.7: Allgemeine Struktur von WSDL 1.1 (in Anlehnung an [94])

Neben Informationen über Endpunkte und Operationen beschreibt WS-Addressing auch die Identität von Nachrichten. So kann jede Nachricht optional über das Element *MessageID* mit einem Identifikator (ID) versehen werden. Über diese ID kann die Relation zwischen verschiedenen Nachrichten beschrieben werden. Dazu können optional *RelatesTo*-Elemente verwendet werden, die vorangegangene Nachrichten referenzieren. So ist es z. B. möglich, eine Request-Response-Kommunikation über asynchrone Transportprotokolle zu realisieren.

Im einfachsten Fall betreffen die mit WS-Addressing beschriebenen Informationen den Anbieter sowie Nutzer eines Webservices. WS-Addressing-Elemente können aber auch an SOAP-Zwischenknoten gerichtet sein und so ein sehr komplexes Routing realisieren.

Web Services Description Language

Um in einer SOA Dienstaggregate flexibel aus Diensten eines Dienstinventars komponieren zu können, ist die maschinenlesbare Beschreibung der Schnittstellen unerlässlich. Eine solche Beschreibung sollte alle für eine Dienstnutzung wesentlichen Informationen sprach- und plattformunabhängig bereitstellen. Jeder Nutzer und Anbieter muss sich bei der Nutzung bzw. Realisierung eines entsprechenden Dienstes an diese Beschreibung halten.

Die *Web Services Description Language* (WSDL, [232]) ist eine solche Sprache zur Beschreibung von Webservice-Schnittstellen. Sie ist eine XML-Grammatik und definiert Datentypen, Nachrichten, Operationen und den Nachrichtenaustausch in einer auf Webservices basierenden Kommunikation. Ähnlich wie bei SOAP existieren zurzeit auch bei WSDL mit der Version 1.1 [232] und WSDL 2.0 [252–254] zwei verschiedene Versionen. Im Gegensatz zu SOAP sind die Unterschiede jedoch wesentlich. Und obwohl die Version 2.0 den Status einer „W3C Recommendation“ hat, findet sie kaum Verwendung. Die meisten auf WSDL aufbauenden Standards und Werkzeuge verwenden ausschließlich WSDL 1.1. Aus diesem Grund liegt auch dieser Arbeit WSDL in der Version 1.1 zugrunde und wird im Folgenden näher erläutert.

Abbildung 2.7 stellt die Struktur eines WSDL-Dokuments schematisch dar. WSDL lässt sich in einen abstrakten (hellgrau hinterlegt) und einen konkreten Teil (dunkelgrau hinterlegt) gliedern. Der abstrakte Teil beschreibt die Schnittstelle eines Webservices, ohne auf die Verwendung konkreter Nachrichtenformate und -codierungen sowie Transportprotokolle einzugehen. Diese Eigenschaften können optional im konkreten Teil eines WSDL-Dokuments definiert werden.

Zur abstrakten Beschreibung eines Webservices gehört die Definition von Datentypen (*types*), die in Webservice-Nachrichten übertragen werden. In der WSDL-Spezifikation wird *XML-Schema* [239–242] zur Beschreibung der Datentypen empfohlen. Prinzipiell ist aber auch die Verwendung anderer Datenbeschreibungssprachen wie z. B. RELAX NG (*REgular LAnguage for XML Next Generation*, [153]) oder DTD möglich, was jedoch selten praktiziert wird [130].

Nachrichten (*message*) beschreiben den abstrakten Aufbau einer Webservice-Nachricht. Dazu referenzieren sie eine Menge von Datentypen, die innerhalb dieser Nachricht übertragen werden. Sie repräsentieren die Ein- (*input*) und Ausgaben (*output*) der Operationen (*operation*) eines Webservices. Je nach *Message Exchange Pattern* (MEP, dt.: Nachrichtenaustauschmuster) referenzieren Operationen entsprechende Nachrichten. WSDL definiert die folgenden vier MEPs:

One-way Der Webservice-Nutzer sendet eine Nachricht an den Anbieter.

Request-response Der Nutzer sendet eine Anfrage an den Anbieter und erhält eine Antwort zurück.

Solicit-response Der Webservice-Anbieter sendet eine Nachricht an den Nutzer und erhält eine Antwortnachricht zurück.

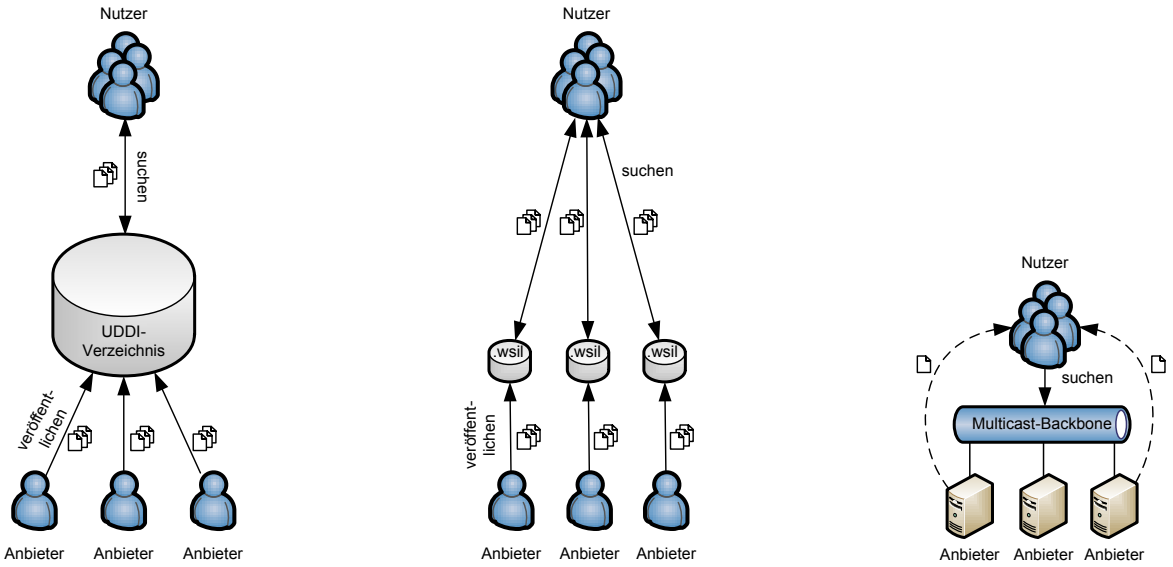
Notification Der Anbieter sendet eine Nachricht an den Webservice-Nutzer.

Zusätzlich zu den regulären Ein- und Ausgaben können Operationen Nachrichten referenzieren, die im Fehlerfall (*fault*) ausgetauscht werden.

Der sog. *portType* fasst einzelne Operationen zu einer abstrakten Webservice-Schnittstelle zusammen und schließt damit die abstrakte Schnittstellendefinition ab. Im konkreten Teil eines WSDL-Dokuments wird die abstrakte Webservice-Beschreibung um das *binding* ergänzt. Dazu wird das zu verwendende Transportprotokoll und Nachrichtenformat sowie Codierungsdetails definiert. Über eine Referenz zum *portType* wird festgelegt, für welche abstrakte Schnittstelle eine Binding-Definition festgelegt wird. Dabei können für einen *portType* auch mehrere Bindungen definiert werden. Obwohl prinzipiell beliebige Nachrichtenformate für den Austausch von Webservice-Nachrichten verwendet und in der WSDL-Beschreibung definiert werden können, hat sich für diese Aufgabe SOAP als einziger Standard durchgesetzt.

Im Abschnitt *service* werden alle konkreten Endpunkte eines Webservices definiert. Ein Service enthält dazu ein oder mehrere *port*-Elemente, für die jeweils eine Endpunktadresse als URI definiert wird. Zusätzlich wird eine Binding-Definition referenziert, die verwendet werden muss, um einen über die URI identifizierten Webservice zu verwenden.

Die dargestellten Elemente von WSDL dienen der formalen, maschinenlesbaren Beschreibung von Webservice-Schnittstellen. WSDL ermöglicht entsprechenden Webservice-Umgebungen die automatische Bereitstellung jeglicher für die Kommunikation nötige Middleware-Funktionalität. Darüber hinaus kann die formale Beschreibung um eine nicht formale, an den Menschen gerichtete Dokumentation erweitert werden.



(a) UDDI (b) WS-Inspection (c) WS-Discovery

Abbildung 2.8: Funktionsweise ausgewählter Webservice-Standards zur Dienstvermittlung

Webservice-Standards zur Dienstvermittlung

Mit der Verwendung von SOAP und WSDL können Dienste in einer auf Webservices aufbauenden SOA beschrieben und verwendet werden. So reichen im einfachsten Fall die Kenntnis der WSDL-Beschreibung zur Designzeit und ihre statische Einbindung in den Programmcode aus, um einen Webservice per SOAP zu nutzen. Alle Informationen der Webservice Schnittstelle wären dann für die Laufzeit unveränderlich festgelegt und würden damit die wichtige SOA-Eigenschaft der losen Kopplung einschränken. Zur Lösung dieses Problems definiert das SOA-Rollenmodell die Rolle der Dienstvermittlung. Sie realisiert das automatische Veröffentlichen und Finden von Diensten und ist wesentlich für die Erreichung einer losen Kopplung.

In einer auf Webservices basierenden SOA existieren verschiedene Standards, die die Rolle der Dienstvermittlung implementieren. Mit UDDI (*Universal Description, Discovery and Integration*, [154]), WS-Inspection (*Web Services Inspection Language*, [13]) und WS-Discovery (*Web Services Dynamic Discovery*, [163]) werden die drei wichtigsten Standards im Folgenden vorgestellt. Abbildung 2.8 zeigt die Funktionsweise der drei Standards. Sie unterscheiden sich vor allem in ihrem Zentralisierungsgrad.

Universal Description, Discovery and Integration Das Ziel der Entwicklung von UDDI war der Aufbau eines globalen Dienstverzeichnisses, über das Webservice-Nutzer automatisch zur Laufzeit Dienste finden und in ihre Dienstkompositionen einbinden können. Neben technischen Schnittstelleninformationen (z. B. WSDL-Dokumenten) sollen über UDDI-Verzeichnisse auch allgemeine Geschäftsdaten (z. B. Kontakt- und Brancheninformationen) abgerufen werden können.

Wie in Abbildung 2.8a dargestellt, ist ein UDDI-Verzeichnis ein zentrales Verzeichnis, in dem Informationen über Webservices gespeichert werden. Webservice-Anbieter veröffentlichen, d. h. registrieren die von ihnen betriebenen Webservices beim UDDI-Verzeichnis und ergänzen die Registrierungen um technische und nicht technische Informationen bzw. um Verweise auf entsprechende Dokumente. Webservice-Nutzer können nun über verschiedene Suchkriterien beim zentralen Verzeichnis nach passenden Diensten suchen und erhalten über das Verzeichnis Zugriff auf entsprechende Dienstinformationen.

Technisch betrachtet ist ein UDDI-Verzeichnis selbst ein Webservice. Die UDDI-Spezifikation definiert eine Reihe von Datentypen zur Beschreibung der in einem Verzeichnis abgespeicherten und in SOAP-Nachrichten übertragenen Daten sowie mehrere WSDL-Dokumente zur Beschreibung des Zugriffs auf UDDI-Verzeichnisse. Neben dem maschinellen Zugriff ermöglichen die meisten UDDI-Implementierungen auch einen Zugriff für Menschen über eine Webschnittstelle per Browser.

Web Services Inspection Language WS-Inspection verwirklicht genau wie UDDI eine Vermittlung auf Basis von Verzeichnissen. Anders als bei UDDI realisiert WS-Inspection jedoch nicht ein zentrales, sondern mehrere dezentrale Verzeichnisse auf eine vollständig dokumentenbasierte Weise (s. Abbildung 2.8b). Ein Anbieter von Webservices schreibt Informationen über alle von ihm angebotenen Webservices in ein entsprechendes Dokument, das auf seinem Webserver zum Abruf bereitgelegt wird. Damit Nutzer die Dokumente finden und interpretieren können, muss eine solche WS-Inspection-Datei den Namen *inspection.wsil* haben und einer in der WS-Inspection-Spezifikation definierten XML-Grammatik entsprechen. Informationen über Webservices können in einer WS-Inspection-Datei über Verweise auf WSDL-Dokumente oder UDDI-Einträge referentiell abgespeichert werden.

Darüber hinaus können sie Verweise auf weitere WS-Inspection-Dateien haben und erlauben dem Nutzer seine Suche nach dem Hyperlink-Prinzip auszudehnen. Um Dienste zu suchen, muss also ein Webservice-Nutzer alle potenziellen Anbieter kennen und dort die Verzeichniseinträge abfragen oder ausgehend von einem bekannten Anbieter rekursiv den in den WS-Inspection-Dateien gefundenen Hyperlinks folgen.

Web Services Dynamic Discovery WS-Discovery ist ein Protokoll zum Auffinden von Webservices, das im Gegensatz zu UDDI und WS-Inspection auf ein Verzeichnis verzichtet (s. Abbildung 2.8c). Im Ad-hoc-Modus benötigt WS-Discovery keine zentrale Komponente. Es wird jedoch vorausgesetzt, dass sich alle Webservices in einer gemeinsamen Multicast-Gruppe befinden.

Tritt ein Dienst in ein Netzwerk bzw. in die Multicast-Gruppe ein, so schickt er eine Multicast-Nachricht (*Hello*) mit verschiedenen Dienstinformationen an die Multicast-Gruppe. In dieser Nachricht sind ein Bezeichner sowie optional Informationen über assoziierte Nachrichtengruppen (z. B. ein referenzierter *portType*), logische Suchbereiche und eine Transportadresse enthalten. Potenzielle Nutzer können diese Informationen verwenden oder für die spätere Verwendung zwischenspeichern. Wenn ein Dienst das Netz wieder verlässt, soll er sich per Multicast-Nachricht (*Bye*) wieder abmelden.

Neben dieser passiven Art des Auffindens von Diensten ist es auch möglich, dass Dienstnutzer aktiv nach Anbietern suchen. Dazu wird eine Anfrage mit Suchkriterien zu Nachrichtengruppen und/oder zum logischen Suchbereich per Multicast-Nachricht (*Probe*) verschickt. Alle Webservices, die den Kriterien entsprechen, schicken per Unicast eine Antwortnachricht (*ProbeMatches*) mit denselben Informationen zurück, die auch beim *Hello* übertragen werden.

Bezeichner sind in der Regel logische, transportunabhängige Namen. Sie werden meist als UUID (*Universally Unique Identifier*, [108]) codiert. Zur Auflösung von konkreten Transportadressen für Bezeichner sieht WS-Discovery *Resolve*-Anfragen vor.

Zur Reduktion des Multicast-Aufkommens können im verwalteten Modus Proxies Dienstinformationen aus *Hello*-Nachrichten zwischenspeichern. Aktiv anfragende Nutzer können dann beim Proxy nach Diensten suchen und müssen keine Multicast-Suchanfrage an die gesamte Gruppe senden.

Wie UDDI verwendet auch WS-Discovery die Webservice-Technologie zur Kommunikation. Dazu werden SOAP-Nachrichten verwendet, die vorwiegend über UDP, z. T. aber auch über HTTP ausgetauscht werden.

Über WS-Discovery können im Gegensatz zu UDDI und WS-Inspection Dienste nur gefunden werden. Für den Abruf von Metainformationen wie die WSDL-Beschreibungen muss WS-Discovery mit weiteren Standards wie *WS-MetadataExchange* [260] und *WS-Transfer* [243] kombiniert werden.

2.3.3 Informationstechnische Umsetzung von Geschäftsprozessen mit Webservices

Die Fähigkeit, Geschäftsprozesse automatisch in Workflowmanagementsystemen auszuführen, ist heute ein entscheidender Erfolgsfaktor des operativen Geschäftsprozessmanagements. Wie in Abschnitt 2.2.4 beschrieben, können Geschäftsprozesse in einer SOA als eine zeitlich logische Folge von Dienstaufrufen realisiert werden. Mit den in Abschnitt 2.3.2 erläuterten Basistechnologien stehen umfangreiche Standards zur Verfügung, mit denen sich dieses Konzept auf Basis von Webservices plattformunabhängig und über ein Netzwerk verteilt umsetzen lässt.

Heutzutage existieren für nahezu jede im Unternehmensumfeld eingesetzte Programmiersprache Umgebungen, die die Realisierung und Nutzung von Webservices erlauben und somit den Aufbau eines Serviceinventars und die Nutzung entsprechender Dienste in Dienstkompositionen ermöglichen. Zur Realisierung von Basisdiensten und einfachen zusammengesetzten Diensten stellt diese Art von Programmiersprachen in der Regel ein geeignetes Implementierungsmittel dar. Für die Realisierung von komplexen zusammengesetzten Diensten und Prozessservices mit ihren komplizierten Kontrollflüssen und Interaktionsbeziehungen sind sie jedoch nicht zweckmäßig. Wie in Abschnitt 2.2.4 erläutert, stellt im Geschäftsprozessmanagement die modellgetriebene Prozessmodellierung mit domänenspezifischen Sprachen für die Realisierung dieser Dienstkategorien einen entscheidenden Erfolgsfaktor dar.

Aufbauend auf den in Abbildung 2.6 auf Seite 21 dargestellten Webservice-Technologien existieren eine Reihe von Sprachen, die dem prozessorientierten Paradigma folgen und den zeitlich logischen Prozessablauf deklarativ als eine Folge von Webservice-Aufrufen beschreiben.

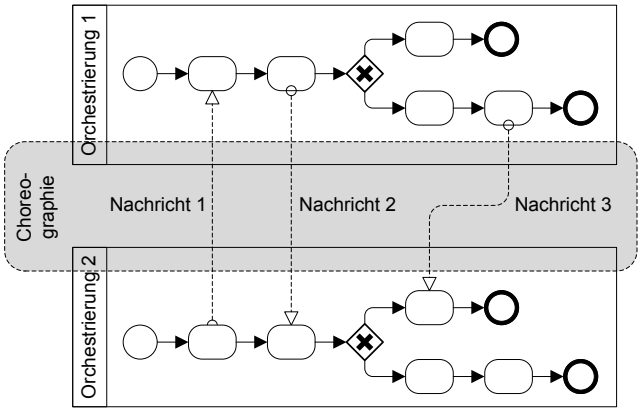


Abbildung 2.9: Dienstorchestrierung und -choreographie

Die so erstellten Webservice-Kompositionen können meist direkt in Workflowmanagementsystemen installiert und ausgeführt werden. Sie bilden das informationstechnische Gegenstück zur fachlich-konzeptionellen Modellierung.

Im Folgenden werden ausgewählte Standards zur Webservice-Komposition vorgestellt. Dabei lassen sich die Sprachen mit Orchestrierung und Choreographie in zwei Arten von Kompositionsstandards unterteilen und auf der Prozessebene in den Webservice-Technologiestapel einordnen. Abbildung 2.9 zeigt ihre konzeptionellen Unterschiede. Orchestrierungen beschreiben die ausführbaren Aspekte eines Geschäftsprozesses aus der prozessinternen Sicht. Das bedeutet, dass das Aggregat durch die zeitlich logische Einbindung von externen Diensten in das Prozessmodell definiert wird. Bei der Choreographie hingegen wird eine Dienstkomposition durch die Festlegung des Zusammenspiels zwischen unterschiedlichen Diensten unter dem Aspekt der Kollaboration definiert [34, 94, 130].

Webservice-Orchestrierung mit der Business Process Execution Language

Die von OASIS standardisierte Sprache BPEL (*Web Services Business Process Execution Language*, [159]) ermöglicht die Modellierung und Ausführung von Geschäftsprozessen als Aggregation von Webservices. Sie gilt als De-facto-Standard [50, 94] zur Webservice-Orchestrierung und ist aus der Vereinigung von WSFL (*Web Services Flow Language*, [112]) und XLang (*Extensible Language*, [135]) entstanden. BPEL wurde bis zur Version 1.1 unter dem Namen *Business Process Execution Language for Web Services* (BPEL4WS, [7]) standardisiert. Mit der Version 2.0 wurde dieser Name gemäß der gängigen Praxis, Webservice-Standards mit WS-* (Web Services *) zu benennen, zu WS-BPEL geändert. Auch wenn zwischen den unterschiedlichen BPEL-Versionen keine konzeptionellen Unterschiede existieren, führen die technischen Unterschiede zu Kompatibilitätsproblemen. Heutzutage wird BPEL vorwiegend in der Version 2.0 eingesetzt, die auch dieser Arbeit zugrunde liegt.

BPEL ist eine XML-basierte Sprache, die sich durch ihr prozessorientiertes Paradigma auszeichnet. Sie stellt eine Vielzahl an Sprachkonstrukten zur Verfügung, die eine domänenspezifische Prozessdefinition durch die Komposition von Webservices erlauben. BPEL

ermöglicht die deklarative Beschreibung der Webservice-Interaktionen und deren Koordination, also den die Prozesslogik abbildenden Kontrollfluss. Ergänzend dazu bietet BPEL Sprachmittel, die eine imperative, Turing-vollständige Programmierung ermöglichen. Aufgrund ihrer standardisierten Beschreibung des Kontroll- und Nachrichtenflusses sowie der Verwendung von Webservice-Standards können BPEL-Prozesse direkt in allen standardkonformen Laufzeitumgebungen ausgeführt werden und mit beliebigen Webservices interagieren. Obwohl BPEL im Gegensatz zu BPMN oder EPK selbst keine grafische Repräsentation standardisiert, werden BPEL-Modelle üblicherweise nicht durch manuelles Schreiben von XML-Quelltext realisiert. Heutzutage werden vor allem grafische Entwicklungswerkzeuge eingesetzt, die die Darstellung und Modellierung von BPEL als Prozessgraphen ermöglichen. Das prozessorientierte Paradigma von BPEL kombiniert mit der intuitiven, domänenspezifischen Bedienbarkeit der Entwicklungswerkzeuge ermöglicht, dass Geschäftsprozesse auch informationstechnisch von Domänenexperten implementiert werden können.

Die Orchestrierung von Webservices mit BPEL kann weitestgehend deklarativ durchgeführt werden. Da BPEL aber direkt ausgeführt werden kann, können die grafischen Entwicklungswerkzeuge nicht alle technischen Details dem Designer verbergen. Es muss z. B. deklariert werden, dass eine bestimmte Prozessaktivität durch den Aufruf eines Webservices realisiert wird. Darüber hinaus bildet BPEL auch nicht immer alle relevanten Aspekte einer fachlich-konzeptionellen Modellierung ab. Es ist z. B. nicht möglich, organisationale Verantwortungsbereiche zu beschreiben. Aus diesem Grund gibt es in der Geschäftsprozessmanagementforschung auch Ansätze, die BPEL nicht direkt zur Prozessmodellierung nutzen. Stattdessen werden fachlich-konzeptionelle Sprachen zur Modellierung verwendet und entsprechende Modelle vollständig oder teilweise automatisch nach BPEL transformiert, um BPEL zur Workflowausführung zu nutzen [1, 2, 103, 203, 224]. So wird bereits im BPMN-Standard [148] eine Abbildung von BPMN-Elementen nach BPEL spezifiziert. Auch für eine Transformation von EPK nach BPEL existieren zahlreiche Arbeiten [203, 269].

BPEL-Prozesse sind üblicherweise lang laufend und zustandsbehaftet. Sie beschreiben eine Aggregation von Diensten durch die Interaktionen zwischen dem Prozess und seinen Partnern sowie durch die Definition ihrer zeitlich logischen Abfolge, dem Kontrollfluss. Partner können in diesem Zusammenhang sowohl Dienste sein, die vom Prozess konsumiert werden, wie auch Konsumenten, die vom Prozess zur Verfügung gestellte Funktionalität nutzen. Die Interaktionen mit Partnern basieren vollständig auf einer Webservice-Kommunikation. Das bedeutet, dass sowohl Prozessaktivitäten durch Webservices in den Prozess eingebunden, als auch, dass die vom Prozess selbst realisierte Geschäftslogik wieder als ein oder mehrere Webservices angeboten werden. Dadurch realisiert BPEL ein rekursives Aggregationsmodell.

BPEL beschreibt einen Prozess grundsätzlich aus der prozessinternen Sicht und ist damit den Orchestrierungssprachen zuzuordnen. Neben ausführbaren Modellen können mit BPEL aber auch abstrakte Prozessmodelle definiert bzw. aus ausführbaren Modellen generiert werden. Abstrakte BPEL-Prozesse sind nicht ausführbar, da sie keine internen Realisierungsdetails definieren müssen. Sie bilden lediglich eine externe Prozesssicht, die sich auf die Schnittstellen zu Partnern und die Nachrichtenfolge beschränkt. Abstrakte Prozesse können so zur Abstimmung mit Geschäftspartnern verwendet werden, ohne Implementierungsdetails preiszugeben. Entsprechend werden abstrakte BPEL-Prozesse von vielen Autoren auch als Choreographie klassifiziert [168]. Da sie aber prinzipiell alle Elemente eines konkreten Prozesses haben

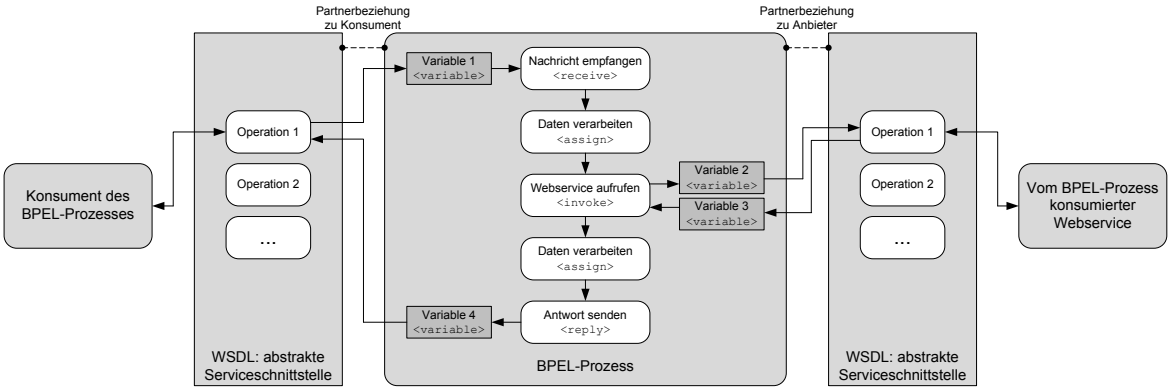


Abbildung 2.10: Schematische Darstellung eines exemplarischen BPEL-Prozesses und seiner Interaktionen mit Kommunikationspartnern (in Anlehnung an [167])

können, hängt es letztlich vom Grad der beschriebenen Implementierungsdetails ab, ob ein abstrakter Prozess eine Choreographie oder Orchestrierung ist.

Struktur Der BPEL-Standard fügt sich nahtlos in den modularen Aufbau des Webservice-Technologiestapels (s. Abbildung 2.6 auf Seite 21) ein. Er setzt auf einer Reihe von Webservice-Spezifikationen und verwandter Standards auf. WSDL 1.1 wird zur Beschreibung der Webservice-Schnittstellen, XML-Schema 1.0 zur Beschreibung von Datentypen sowie XPath 1.0 (*XML Path Language*, [229]) und XSLT 1.0 (*XSL Transformations*, [230]) zum Zugriff bzw. zur Manipulation der Prozessdaten eingesetzt. Entsprechend kann ein mit BPEL definiertes Geschäftsprozessmodell aus einer Vielzahl an Dokumenten bestehen. Das zentrale Dokument eines Prozessmodells ist die eigentliche Prozessbeschreibung als Abbildung der Geschäftslogik. Es beschreibt den Daten- und Kontrollfluss sowie die Webservice-Kommunikation und Fehler- bzw. Transaktionsbehandlung. Abbildung 2.10 zeigt schematisch den Aufbau eines Beispiel-BPEL-Prozesses und die Interaktionen mit seinen Kommunikationspartnern.

Wie bereits erwähnt, verwendet BPEL zum Austausch von Nachrichten mit seinen Partnern Webservices. Üblicher- aber nicht notwendigerweise wird SOAP als Nachrichtenprotokoll verwendet. Zur formalen Beschreibung von Kommunikationsbeziehungen zu Partnern lassen sich mit BPEL Partnerbeziehungen definieren. Mit einer abstrakten WSDL-Beschreibung werden die Schnittstellen der Dienste und die entsprechenden Operationen beschrieben. Über das Konstrukt der Partnerbeziehungen, das im BPEL- und WSDL-Dokument definiert werden muss, werden die Partner aus dem BPEL-Prozess referenziert. In diesen Beziehungen kann der BPEL-Prozess die Rolle des Webservice-Anbieters und/oder des -Konsumenten einnehmen. Jeder BPEL-Prozess kann in beiden Rollen mit beliebig vielen Partner-Webservices in Beziehung stehen. Mindestens muss jedoch eine Partnerbeziehung existieren, in der der Prozess die Rolle des Anbieters einnimmt.

BPEL unterstützt zum Austausch von Nachrichten alle in WSDL spezifizierten Austauschmuster. Zur Beschreibung des Nachrichtenflusses spezifiziert der Standard mit *invoke*, *receive* und *reply* grundsätzlich drei Aktivitäten, die je nach Kommunikationsrichtung und Austauschmuster verwendet werden können. *invoke* ermöglicht den synchronen und asynchronen Webservice-Aufruf und blockiert im Fall der synchronen Kommunikation bis zum Eintreffen

der Antwort. Mit *receive* wird der Empfang einer Nachricht von einem aufrufenden Partner realisiert. Dabei blockiert die Prozessinstanz ebenfalls bis zum Eintreffen der Nachricht. Neben *pick* (s. u.) ist *receive* eine von zwei möglichen Aktivitäten, die einen Prozess instanziieren können und somit dessen erste Aktivität sein müssen. Mit der *reply*-Aktivität können Antwortnachrichten in einer synchronen Kommunikation an den aufrufenden Partner zurückgeschickt werden. Entsprechend muss jedem *reply* ein korrespondierendes *receive*-Element zum Empfang der Anfragenachricht vorausgehen.

Die Identifizierung der Webservice-Operationen, auf die sich die *invoke*-, *receive*- oder *reply*-Aktivitäten beziehen, wird durch deren Verknüpfung mit einer Partnerbeziehung und der entsprechenden im WSDL-Dokument definierten abstrakten Webservice-Schnittstelle und -Operation realisiert. Je nach Aktivität und zu realisierenden Kommunikationsmuster werden Ein- und/oder Ausgabevariablen deklariert und mit der entsprechenden Aktivität verknüpft. Über diese Variablen kann der Prozess auf die Daten der Nachrichten zugreifen bzw. diese verändern.

BPEL-Prozesse können einen mehrere Webservice-Aufrufe überdauernden Prozesszustand haben. Da es sich bei Webservices aber um eine zustandslose Kommunikationstechnologie handelt, müssen Korrelationen zwischen eingehenden Webservice-Nachrichten und Prozessinstanzen auf Prozessebene ermittelt werden. BPEL verwendet zum Identifizieren der Instanzen einzelne Felder der Nutzlast einer Webservice-Nachricht. Welche Felder verwendet werden, muss im entsprechenden WSDL-Dokument definiert und vom BPEL-Prozess referenziert werden.

Neben Aktivitäten zur Kommunikation zwischen dem Prozess und seinen Partnern stellen Anweisungen zur Beschreibung des Prozess- bzw. Kontrollflusses die zweite Gruppe an Sprachelementen dar. BPEL beschreibt Kontrollflüsse grundsätzlich als hierarchisch ineinander verschachtelte Blöcke und ermöglicht so die Definition von Gültigkeitsbereichen. Über das Konstrukt der Linksemantik können auch Prozessflussabhängigkeiten zwischen Aktionen unterschiedlicher, parallel ausgeführter Blöcke definiert und damit prinzipiell die Blockstruktur aufgebrochen und eine Graphenstruktur erreicht werden.

Als elementare Aktivitäten zur Beschreibung des Kontrollflusses existieren mit *sequence* und *flow* zwei Elemente zur sequenziellen bzw. parallelen Aktivitätsausführung. Darüber hinaus stehen auch in BPEL die aus Programmiersprachen (wie z. B. Java) bekannten bedingten Anweisung *if*, *ifelse* und *else* sowie die Schleifenkonstrukte *while*, *repeatUntil* und *forEach* zur Verfügung. Lediglich die Funktionsweise von *forEach* weicht von der in Java gewohnten ab. In BPEL können entsprechende Schleifendurchläufe wahlweise sequenziell oder parallel ausgeführt werden.

Mit der *pick*-Aktivität wird eine Menge von Ereignissen zusammengefasst. Erreicht der Prozessfluss eine solche Aktivität, dann blockiert dieser. Tritt eines der definierten Ereignisse ein, dann wird genau die damit verknüpfte Aktion ausgeführt. Alle weiteren mit *pick* verknüpften Ereignisse werden daraufhin verworfen. Mit *onMessage* und *onAlarm* gibt es zwei verschiedene Ereignistypen. *onMessage* repräsentiert ein Ereignis, das durch eine eingehende Nachricht ausgelöst wird, und ist damit mit *receive* vergleichbar. Enthält *pick* nur *onMessage*-Ereignisse, dann kann es zur Prozessinstanziierung verwendet werden. *onAlarm* beschreibt ein zeitgesteuertes Ereignis als Ablauf einer Wartezeit oder als Erreichen eines Zeitpunkts.

Als dritte Gruppe von Sprachkonstrukten sind Aktivitäten zur Beschreibung des Datenflusses zu nennen. In BPEL können Daten in Variablen gespeichert werden. Ihre aktuelle Belegung bestimmt zusammen mit dem aktuellen Fortschritt des Prozessablaufs den Zustand einer BPEL-Instanz. BPEL ist typensicher. Seine Variablen können Daten der Typen WSDL-Nachrichtentyp, XML-Schema-Datentyp und XML-Schema-Element sowie einfache und komplexe XML-Datentypen enthalten. Variablen können global deklariert oder verschiedenen beliebig verschachtelten Gültigkeitsbereichen zugeordnet und somit als lokale Variablen deklariert werden. Variablen unterschiedlicher Gültigkeitsbereiche können sich überdecken.

Der Zugriff auf Variablen oder Teile des in ihnen gespeicherten XML-Ausdrucks wird mithilfe der XML-Abfragesprache XPath realisiert. Zur Manipulation stellt BPEL die *assign*-Aktivität zur Verfügung. Neben dem Kopieren von Daten zwischen verschiedenen Variablen bzw. ihrer Kindelemente lassen sich auch eine Reihe elementarer, in BPEL spezifizierter Operationen verwenden. Diese werden innerhalb des Prozesses und nicht als Webservice-Aufruf ausgeführt und können um selbst definierte Operationen erweitert werden. Zusätzlich lassen sich Datenzugriffe und -manipulationen in *assign* auch durch XSLT realisieren.

Während der Prozessausführung können im internen Prozessablauf, bei der Kommunikation oder bei der Ausführung von externen Aktivitäten durch Webservices Fehler auftreten. BPEL ermöglicht eine Fehlerbehandlung über sog. *faultHandler*-Elemente. Sie können dem Gesamtprozess oder einzelnen Gültigkeitsbereichen zugeordnet sein. Für einzelne oder alle Fehlerfälle können beliebige Kontrollflüsse definiert werden, die die entsprechenden Fehler behandeln.

Eng verwandt mit der Fehlerbehandlung ist die Transaktionsverwaltung. BPEL-Prozesse können sowohl selbst Teil einer Transaktion sein, aber auch verschiedene Aktionen zu einer Transaktion zusammenfassen. Im Datenbankenbereich werden hauptsächlich ACID-Transaktionen eingesetzt. Sie erfüllen die harten Transaktionsanforderungen Atomarität, Konsistenz, Isoliertheit und Dauerhaftigkeit. Da Geschäftsprozesse durch eine lange Laufzeit gekennzeichnet sind, handelt es sich auch bei den entsprechenden Transaktionen vorwiegend um lang laufende Transaktionen (engl.: *Long-Running Transactions*, LRT). Für dieses Szenario sind ACID-Transaktionen nicht geeignet, da z. B. das exklusive Sperren einzelner Ressourcen über lange Zeiträume nicht möglich oder sinnvoll ist. BPEL verwendet in diesem Zusammenhang das im Webservice-Umfeld präferierte Konzept der Kompensation. Hier werden einzelne Aktivitäten nicht zu einer atomaren zusammengefasst, die mit Abschluss der Transaktion entweder komplett oder gar nicht ausgeführt wurde. Stattdessen werden Aktivitäten voneinander unabhängig ausgeführt. Für den Fall des Scheiterns werden für jede Aktivität Kompensationsaktivitäten definiert, die die bereits ausgeführten Aktivitäten wieder rückgängig machen. Als Protokoll zur Koordination von verteilten Transaktionen über Prozessgrenzen hinweg empfiehlt BPEL den Einsatz von WS-Transactions (*Web Services Transactions Specifications*, [83]).

Erweiterungen Im Mittelpunkt des BPEL-Standards steht die grundlegende automatische Ausführung von Geschäftsprozessen. Ähnlich wie andere Webservice-Standards kann auch BPEL flexibel erweitert werden. Im Folgenden werden mit WS-HumanTask (*Web Services Human Task*, [4]), BPEL4People (*WS-BPEL Extension for People*, [5]), BPEL-SPE (*WS-*

BPEL 2.0 Extensions for Sub-Processes, [102]) und BPELJ (*BPEL for Java*, [22]) die wichtigsten BPEL-Erweiterungen vorgestellt.

Geschäftsprozesse werden selten vollständig automatisch ausgeführt [101]. In der Regel existieren Aktivitäten, die von menschlichen Aufgabenträgern ausgeführt werden. Durch die Verwendung der Webservice-Technologie können mit BPEL problemlos die Schnittstellen 3 und 4 im WfMC-Referenzmodell eines Workflowmanagementsystems (s. Abbildung 2.4 auf Seite 18) realisiert werden. Die Anbindung menschlicher Interaktion (Schnittstelle 2) bedarf jedoch der aufwendigen manuellen Implementierung entsprechender Benutzerschnittstellen, die per Webservices mit dem BPEL-Prozess interagieren.

WS-HumanTask ermöglicht eine deklarative Definition menschlicher Interaktionen als Webservices. Neben der eigentlichen Benutzerinteraktion (z. B. Dateneingabe) werden insbesondere die Zuordnung einer Aktivität zu Personen oder Gruppen sowie die Festlegung von Regeln zur Verwaltung des Lebenszyklus einer Aufgabe (z. B. Abarbeitungsfristen) deklariert. WS-HumanTask setzt direkt auf WSDL und XML-Schema auf und kann in beliebiger Webservice-Kommunikation verwendet werden.

WS-HumanTask ist vollständig losgelöst von BPEL. Die Integration von WS-HumanTask in BPEL wird durch BPEL4People spezifiziert. BPEL4People baut auf WS-HumanTask auf und erweitert BPEL um die Fähigkeit eine mit WS-HumanTask definierte menschliche Interaktion direkt als Aktivität in den BPEL-Prozess zu integrieren.

Ein weiteres Problem, das von BPEL nicht adressiert wird, ist die Strukturierung von Prozessen. Anders als in Programmiersprachen wie Java, in denen Methoden definiert und beliebig oft an anderen Stellen im Programmcode wiederverwendet werden können, müssen in BPEL mehrfach verwendete Subprozesse in eigene BPEL-Prozesse ausgelagert und dann als Webservices eingebunden werden. BPEL-SPE ermöglicht die Definition von Prozessfragmenten bzw. Subprozessen innerhalb eines übergeordneten Elternprozesses. Die Subprozesse können sowohl über eine zu definierende Schnittstelle oder über gemeinsame Variablen mit dem Elternprozess Daten austauschen. So definierte interne Subprozesse sind direkt an den Lebenszyklus des Elternprozesses gekoppelt.

Darüber hinaus ermöglicht BPEL-SPE die Definition von Subprozessen außerhalb des Elternprozesses. Diese Variante ähnelt sehr stark der Auslagerung von Subprozessen als eigenständige BPEL-Prozesse. Externe BPEL-SPE-Subprozesse sind jedoch im Gegensatz zu eigenständigen BPEL-Prozessen durch eine Kopplung des Lebenszyklus des Subprozesses an den des Elternprozesses gekennzeichnet.

Auch wenn BPEL Turing-vollständig ist, lassen sich bestimmte Anwendungslogiken einfacher mit Programmiersprachen der 3. Generation wie Java lösen. Mit BPELJ existiert ein Ansatz, der die Integration von Java-Quelltext als Aktivitäten im BPEL-Prozess erlaubt. Solche Java-Aktivitäten haben vollen Zugriff auf den Kontext (z. B. auf Variablen) einer Prozessinstanz. Neben BPELJ existieren ähnliche Spezifikationen für weitere Programmiersprachen. BPELJ oder vergleichbare Spezifikationen müssen jedoch mit Vorsicht eingesetzt werden, da die Vermischung von BPEL und Programmiersprachen der 3. Generation schnell zu unübersichtlichen Prozessmodellen führen kann sowie deren Portabilität einschränkt.

Alternative Sprachen zur Webservice-Orchestrierung

Neben BPEL als wichtigste Sprache zur Beschreibung ausführbarer Geschäftsprozesse existiert eine Vielzahl weiterer Sprachen, von denen die bedeutendsten im Folgenden vorgestellt werden. Die *Business Process Modeling Language* (BPML, [28]) ist eine XML-basierte Sprache zur abstrakten Beschreibung und konkreten Ausführung von Geschäftsprozessen als Webservice-Orchestrierungen. Genau wie in BPEL werden Subprozesse und Aktivitäten über Webservice-Interaktionen in das Prozessmodell eingebunden. Dazu stellt BPML ähnliche Sprachkonstrukte zum Senden und Empfangen von Nachrichten, zur Nachrichtenkorrelation sowie zur Daten-, Fehler- und Transaktionsverwaltung zur Verfügung. Auch zur Beschreibung des koordinierenden Kontrollflusses stellt BPML ähnliche Sprachelemente wie BPEL zur Verfügung, die ebenfalls in rekursiv verschachtelbaren Blockstrukturen beschrieben werden. BPML selbst definiert keine grafische Repräsentation. Stattdessen wird diese Aufgabe von BPMN übernommen [130]. BPML wurde insbesondere von BPEL aber auch von XPDL verdrängt und wird nicht mehr weiterentwickelt [167].

XPDL (*XML Process Definition Language*, [228]) ist eine von der WfMC standardisierte Sprache zur Beschreibung ausführbarer Geschäftsprozesse. Sie spezifiziert ein Format zum Austausch von üblicherweise in BPMN definierten Prozessbeschreibungen zwischen unterschiedlichen Workflowmanagementsystemen. XPDL wurde also nicht speziell für die Beschreibung von Geschäftsprozessen als Webservice-Orchestrierungen entworfen. Der Standard beschreibt vielmehr den Ablauf von Prozessen und die Verteilung von Prozessaktivitäten und Subprozessen entsprechend des WfMC-Referenzmodells für Workflowmanagementsysteme unabhängig von den tatsächlichen Realisierungstechnologien eines solchen Systems [194].

In XPDL ist eine Aktivität oder ein Subprozess ein Arbeitsvorgang, der von einer Ressource (z. B. einem Menschen) unter der Verwendung von Daten und einer Applikation ausgeführt wird. Als Applikation beschreibt der Standard die Einbindung von Anwendungen über verschiedene Mechanismen. Sie können z. B. durch EJBs (*Enterprise JavaBeans*, [151]), normale Java-Objekte (*Plain Old Java Objects*, POJO), XSLT-Skripte oder auch durch Webservices realisiert und eingebunden werden. Die Unterstützung einzelner Mechanismen durch ein WFMS ist jedoch optional.

Der Kontrollfluss wird in XPDL anders als in BPEL und BPML nicht als Blockstrukturen, sondern als Graph modelliert. Da die fachlich-konzeptionellen Modellierungssprachen meist ebenfalls graphenorientiert sind, wird so eine einfachere Transformation entsprechender Modelle nach XPDL ermöglicht [194]. Resultierend aus den fehlenden Blockstrukturen verfügt XPDL jedoch über keine Transaktionsbehandlung.

Microsofts *Windows Workflow Foundation* (WF, [138]) ist eine proprietäre Umgebung zur workflowgesteuerten Anwendungsentwicklung. Sie ist Teil der *Microsoft.NET*-Umgebung (.NET, [137]) und ermöglicht die Prozessausführung in eigenständigen Applikationen genauso wie in Anwendungsservern. Damit unterscheidet sich die WF von BPEL, BPML und XPDL, die typischerweise in schwergewichtigen Laufzeitumgebungen in Anwendungsservern ausgeführt werden.

Die WF ermöglicht die deklarative Modellierung von Prozessflüssen als zeitlich logische Folge von vordefinierten Aktivitäten und Ereignissen. Der Kontrollfluss wird mit einem grafischen Editor modelliert. Es stehen mit sequenziellen und graphenorientierten Workflows sowie

Statuscomputern drei verschiedene Diagrammarten zur Modellierung zur Verfügung. Sequenzielle Workflows ähneln BPEL-Prozessen und ermöglichen die Kontrollflussmodellierung in rekursiven Blockstrukturen. Dabei unterstützt die WF weitestgehend vergleichbare Sprachelemente wie BPEL (inklusive Transaktions- und Fehlerbehandlung). Graphenorientierte Workflows ermöglichen analog zu XPDL die Kontrollflussmodellierung als beliebige gerichtete, zusammenhängende Graphen. Ergänzend zu sequenziellen und graphenorientierten Workflows können Prozesse auch als Statuscomputer modelliert werden. In diesem Fall wird der Kontrollfluss als Zustandsautomat modelliert, der Aktivitäten ausführen und durch Ereignisse angesteuert werden kann.

Für alle Modellierungsarten steht eine Vielzahl an vordefinierten Ereignissen und Aktivitäten bereits im Sprachumfang zur Verfügung. Darüber hinaus können eigene Aktivitäten und Ereignisse definiert werden. Dieses kann deklarativ über die Definition beliebiger WF-Prozessmodelle oder imperativ mithilfe von .NET-Programmiersprachen realisiert werden. Zur Kommunikation mit externen Diensten wird die WCF (*Windows Communication Foundation*, [137]), die ebenfalls Bestandteil der .NET-Umgebung ist, verwendet. WCF abstrahiert die Kommunikation verteilter Anwendungen über verschiedene Transportmechanismen wie DCOM (*Distributed Component Object Model*, [80]), MSMQ (*Microsoft Message Queuing*, [136]) oder Webservices. In der WF stehen Aktivitäten und Ereignisse zur Verfügung, die den Empfang und das Versenden von Nachrichten über die WCF und somit auch eine Webservice-Orchestrierung erlauben.

Sprachen zur Webservice-Choreographie

Choreographien beschreiben die Interaktionen zwischen verschiedenen Geschäftsprozessen oder Diensten unter dem Aspekt der Zusammenarbeit, also aus einem globalen Blickwinkel. Im Gegensatz zu den oben beschriebenen Orchestrierungssprachen sind Choreographiesprachen nicht ausführbar [34, 167]. Sie dienen der formalen Beschreibung und Verifikation der Interaktionen zwischen verschiedenen Kommunikationspartnern [14]. Aus ihnen können für jeden Kommunikationspartner abstrakte Prozessgerüste einer Orchestrierung (z. B. abstrakte BPEL-Prozesse) abgeleitet werden [97, 237]. In einem zweiten Schritt können die abstrakten Orchestrierungen um die für ihre automatische Ausführung notwendigen (internen) Implementierungsdetails ergänzt werden.

Für die Definition von Choreographien existieren eine Reihe implementierungsunabhängiger und implementierungsspezifischer, an Webservices gebundener Choreographiesprachen. Als implementierungsunabhängige Sprachen sind MSC (*Message Sequence Chart*, [90]), *UML-Sequenzdiagramme* [146], ebXML/BPSS (*Electronic Business using XML/Business Process Specification Schema*, [216]) und *Let's Dance* [266] zu nennen. Auch BPMN kann zur Choreographie eingesetzt werden, da die Sprache Elemente für beide Kompositionsarten besitzt. Im Kontext der Webservice-Technologie können WSFL oder BPEL4Chor [35] eingesetzt werden. Die wohl momentan bedeutendste Webservice-Choreographiesprache ist jedoch die vom W3C spezifizierte *Web Services Choreography Description Language* (WS-CDL, [236]). WS-CDL ist eine XML-Spezifikation, die auf bestehenden Webservice- und XML-Technologien wie WSDL, XML-Schema sowie XPath aufbaut und sich auf der Prozessebene in den Webservice-Technologiestapel einfügt. Sie kann zur Beschreibung lang laufender Kompositionen von

gleichberechtigten Webservices eingesetzt werden und beschreibt den Austausch von Nachrichten zwischen den Diensten unabhängig von der konkreten Dienstimplementierung.

2.4 Grundlagen drahtloser Sensornetze

Durch die in den vorhergehenden Kapiteln beschriebenen Konzepte und Technologien lassen sich die fachlich-konzeptionellen Geschäftsprozesse eines Unternehmens modellieren und informationstechnisch umsetzen. Analog zu Abbildung 1.1 auf Seite 1 lassen sich die Informationssysteme an die Geschäftsprozesse *anpassen*, die die Unternehmensstrategie realisieren. Wie bereits erwähnt, reicht zur Generierung von Wettbewerbsvorteilen durch ein erfolgreiches Geschäftsprozessmanagement die optimale Anpassung der Informationssysteme an die Unternehmensstrategie bzw. die schnelle und flexible informationstechnische Umsetzung fachlich-konzeptioneller Geschäftsprozesse nicht aus. Es müssen zusätzlich immer neue technische Möglichkeiten erschlossen werden, die völlig neuartige Geschäftsprozesse und Unternehmensstrategien *ermöglichen*.

Mit den *drahtlosen Sensornetzen* (kurz: Sensornetze; engl.: Wireless Sensor Networks, WSN) werden in diesem Kapitel die Grundlagen einer innovativen Technologie beschrieben. Nach Meinung des Autors dieser Arbeit hat sie das Potenzial, die Rolle des *Ermöglichens* einzunehmen, wenn es gelingen würde, diese Technologie in die existierenden Geschäftsprozesse und Enterprise-IT-Systeme zu integrieren.

2.4.1 Sensorknoten und drahtlose Sensornetze

Drahtlose Sensornetze bezeichnen ein Rechnernetz, das aus miteinander in Kommunikationsbeziehung stehenden *Sensorknoten* (kurz: Knoten) gebildet wird. Sie gestatten in verschiedenen Anwendungsszenarien die Wahrnehmung und die Überwachung von physikalischen Phänomenen. Diese Informationen können automatisch in Echtzeit ermittelt und in angeschlossenen Anwendungssystemen in einer hohen Datenqualität zur Verfügung gestellt werden.

Abbildung 2.11 stellt den grundlegenden Aufbau eines Sensornetzes sowie dessen typische Integration in angeschlossene Anwendungssysteme dar. Sensornetze bestehen aus fest installierten oder mobilen Sensorknoten zur Erfassung und Vorverarbeitung von Sensordaten. Sie werden in ihre Messumgebung eingebettet und sind meist auf der Plattform von ressourcenbeschränkten Mikrocomputern realisiert. Sensorknoten verfügen über einen schnellen, flüchtigen (*Random Access Memory*, RAM) und nichtflüchtigen, aber langsamen Speicher sowie über einen Prozessor und eine Energieversorgung. Zur Erfassung von Sensordaten besitzen sie des Weiteren einen oder mehrere Sensoren, also Messwandler, die ein physikalisches Phänomen, wie z. B. Wärme, Licht oder Bewegung, in elektrische oder sonstige Signale umwandeln. Zur Kommunikation mit anderen Knoten oder weiteren Systemkomponenten verfügen Sensorknoten zusätzlich über eine Kommunikationsschnittstelle, die meist als eine kombinierte Sende- und Empfangseinheit (engl.: *Transceiver*) realisiert ist. Je nach Einsatzszenario sind das vor allem Transceiver für eine drahtlose Kommunikation im Kurzstreckenbereich (10-100 m).

Die grundlegende Idee eines Sensornetzes ist nicht, dass jeder einzelne Sensorknoten eine isolierte Überwachungsaufgabe ausführt. In einem Sensornetz wird stattdessen ein bestimmtes

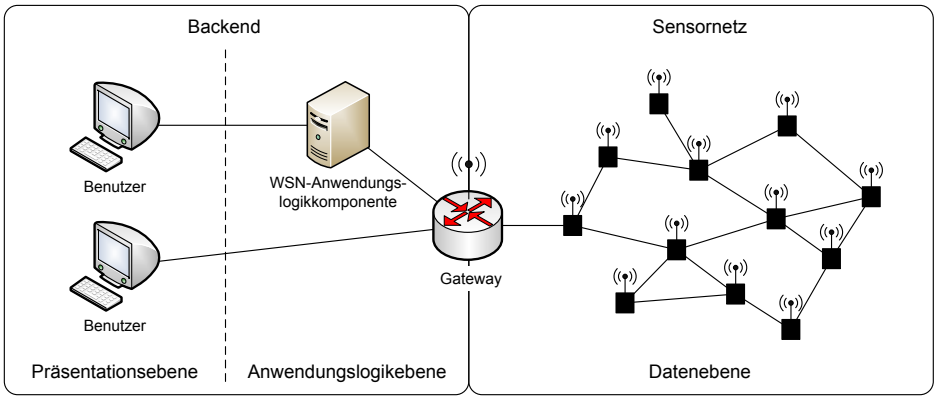


Abbildung 2.11: Grundlegender Aufbau eines Anwendungssystems mit integriertem Sensornetz

Phänomen kollaborativ überwacht. Das bedeutet, dass verschiedene Knoten eine große Menge an Rohsensordaten mit einem geringen Informationsniveau erfassen. Diese Daten werden zu einer kleinen Datenmenge mit einem hohen Informationsniveau verknüpft, indem die Rohdaten korreliert, aggregiert und in einen zeitlichen und räumlichen Kontext gesetzt werden. Indem z. B. die Bewegungssensoren mehrerer Sensorknoten nacheinander eine Bewegung registrieren, lässt sich der Bewegungspfad eines Objekts durch ein per Sensornetz überwachtes Areal ermitteln.

Heutige Sensornetze dienen lediglich der Erfassung und Vorverarbeitung von Sensordaten. Zwar werden einfache Verknüpfungsprozesse von Rohdaten, z. B. die Ermittlung eines Bewegungsvektors oder die Überwachung von Grenzwerten, innerhalb des Sensornetzes durchgeführt. Komplexe Verknüpfungen von Sensordaten oder weitergehende Anwendungslogik werden heute jedoch nicht auf Sensorknoten, sondern im sog. *Backend* ausgeführt. D. h., die Aufgaben des WSN sind weitestgehend auf das Erfassen und Weiterleiten von Sensordaten an das Backend beschränkt. Dieses Paradigma wird in der Sensornetzforschung als *sense and send* (dt.: erfasse und sende) bezeichnet. Entsprechend sind heutige Sensornetze im n-Tier-Architekturmodell [43] eines verteilten Anwendungssystems in der Datenebene einzuordnen.

Mithilfe von *Gateways* können WSNs direkt oder über Weitverkehrsnetze in das sog. *Backend* integriert werden. Das Backend eines Sensornetzes bezeichnet eine einzelne Anwendung oder ein Anwendungssystem, das (relativ zum WSN) auf ressourcenstarken Computern (z. B. auf Enterprise-IT-Systemen) ausgeführt wird und mit dem angeschlossenen WSN Daten austauscht. Entsprechende Anwendungskomponenten enthalten komplexere Logik zur Verarbeitung von Sensordaten, weitergehende Anwendungslogik sowie evtl. Komponenten zur Einbindung menschlicher Nutzer. Innerhalb von komplexeren Anwendungssystemen wie Unternehmensanwendungen empfiehlt sich die Verwendung einer Drei- oder Mehrebenenarchitektur. In diesem Fall werden die vom WSN erfassten und vorverarbeiteten Daten über das Gateway an eine Anwendungskomponente im Backend zur weiteren Verarbeitung gesendet. Diese Komponente ist in der Anwendungslogikebene eines n-Tier-Architekturmodells anzuordnen und ermöglicht Benutzern und weiteren Anwendungskomponenten den Zugriff auf die Sensordaten.

Häufig besteht das Backend eines Sensornetzes jedoch nicht aus einem komplexen Anwen-

dungssystem, sondern lediglich aus einer einzelnen Anwendung, die Sensordaten verarbeitet und einem menschlichen Nutzer präsentiert. In diesem Fall kann die Backend-Anwendung auch als sog. *Fat-Client* realisiert und damit der Präsentationsebene in einem 2-Schichtenmodell zugeordnet werden.

2.4.2 Einsatzszenarien drahtloser Sensornetze

Drahtlose Sensornetze sind ursprünglich aus militärischen Anwendungsszenarien entstanden. Es sollte z. B. eine Vielzahl an Sensorknoten zur feindlichen Aufklärung über einem Kampfgebiet aus dem Flugzeug abgeworfen werden. Diese sollten dann am Boden selbstorganisierend ein Sensornetz zur Überwachung von Truppenbewegungen bilden. Neben den militärischen Szenarien ist der Einsatz von WSNs auch in vielfältigen zivilen Anwendungsfällen möglich und sinnvoll. Sie eignen sich insbesondere zur Datenerfassung in Szenarien, in denen personen- oder kabelgebundene Messsysteme nicht oder nur mit relativ hohem technischen oder finanziellen Aufwand installiert werden können [264].

Einen der wichtigsten zivilen Anwendungsbereiche stellt die Logistik dar. Verschiedene Unternehmen einer logistischen Wertschöpfungskette benötigen zur Abwicklung ihrer Geschäftsprozesse Informationen über die geografische Position und den Zustand von transportierten Gütern. Durch die Einbettung von Sensorknoten in diese Wertschöpfungsprozesse (z. B. durch die Installation von Sensorkoten in Containern oder im Hafengebiet) lassen sich entsprechende Informationen automatisch in Echtzeit und in einer hohen Datenqualität erfassen und in die Prozesse integrieren.

Da die im Rahmen dieser Arbeit entstandenen Konzepte und Technologien im Logistikprojekt L2D2 (*Lübecker Logistikdatendrehscheibe*, [65, 217]) auf ihre Anwendbarkeit in der Unternehmenspraxis überprüft wurden, ist dieses auch der Anwendungsbereich, der im Fokus dieser Arbeit steht. Darüber hinaus existieren aber auch weitere wichtige Anwendungsbereiche, von denen einige im Folgenden exemplarisch aufgeführt werden.

Im Rahmen einer Industrieanlagenüberwachung können z. B. physikalische Daten von Produktionsprozessen erfasst werden. Es existieren bereits Projekte zur Überwachung des Energieverbrauchs von Gebäuden und Anlagen [140] sowie zur Überwachung von Öl- und Gasförderungen [93]. Im Kontext einer Liegenschaftsüberwachung existieren Projekte, die Gelände und Anlagen auf Eindringlinge sowie den baulichen Zustand von Gebäuden überwachen [11, 98, 99, 181, 182, 225]. Forschungsprojekte im Umfeld der Domäne Biologie befassen sich mit der Beobachtung von Tieren mit WSNs in ihrem natürlichem Lebensraum [122, 209, 210, 267]. Und im Rahmen einer Umweltüberwachung wurden bereits Projekte zur Überwachung von Gletschern und zur Detektion von Waldbränden durchgeführt [51, 265].

Ein weiteres Anwendungsfeld von WSNs, das bedingt durch den zurzeit stattfindenden demografischen Wandel an Bedeutung gewinnt, ist die Überwachung medizinischer Kennzahlen von Patienten in ihrer Wohnung oder im Krankenhaus [117, 123, 183, 196]. In diesem Kontext lassen sich die Kosten der Patientenbetreuung reduzieren und die Versorgungsqualität verbessern. Darüber hinaus müssen Patienten in vielen Krankheitsfällen nicht ihre gewohnte häusliche Umgebung verlassen.

2.4.3 Eigenschaften drahtloser Sensornetze

Aus dem in Abschnitt 2.4.1 beschriebenen grundlegenden Aufbau eines Sensornetzes und den in Abschnitt 2.4.2 beschriebenen Anwendungsfällen lassen sich verschiedene Anforderungen an und Eigenschaften von Sensorknoten sowie des Sensornetzes als Ganzes ableiten. Im Rahmen dieser Arbeit sind die extreme Ressourcenbeschränkung sowie der verteilte und eingebettete Charakter von WSNs von entscheidender Bedeutung und werden im Folgenden vorgestellt.

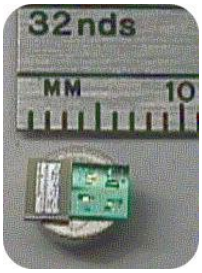
Ressourcenbeschränkung von Sensornetzen

Sensornetze sollen schnell und kostengünstig installiert werden und autark über einen längeren Zeitraum hinweg operieren. Der Aufbau einer Infrastruktur ist jedoch sehr aufwendig und kostenintensiv oder in bestimmten Anwendungsszenarien gar unmöglich. Aus diesem Grund wird im Kontext von WSNs typischerweise auf den Aufbau und die Verwendung einer Infrastruktur verzichtet. Neben der Netzinfrastruktur ist davon vor allem die Energieversorgung betroffen. Sensorknoten können nicht an ein Stromnetz angeschlossen werden, sondern müssen über eine autarke Energiequelle verfügen. Das hat zur Folge, dass Knoten entweder in ihrer Umgebung selbstständig Energie gewinnen (z. B. durch Solarzellen) oder a priori mit einer Energiequelle ausgestattet werden müssen. Aufgrund ihrer Kompaktheit werden Sensorknoten vorwiegend mit Batterien betrieben [26, 32, 179]. Da der Energievorrat von Batterien jedoch begrenzt ist, ist zur Erreichung langer autarker Betriebszeiten eine Minimierung des Energieverbrauchs unerlässlich.

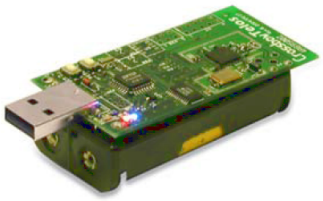
Im WSN kann der Energieverbrauch sowohl durch hardware- als auch durch softwareseitige Optimierungen reduziert werden. Bei der Energieeinsparung durch Hardwareanpassungen können zwei Strategien verfolgt werden. Zum einen wird versucht, Komponenten mit besonders hohem Energieverbrauch vollständig zu substituieren. Bei nicht substituierbaren Hardwarekomponenten werden vor allem einfache Komponenten eingesetzt. Diese zeichnen sich durch einen geringen Energieverbrauch aus, der jedoch nur durch eine Reduzierung der Leistungsmerkmale erreicht werden kann.

Softwareseitig stellt das sog. *Duty Cycling* (dt.: Betriebszyklus) eine weitere bedeutende Methode zum Energiesparen dar. Es bezeichnet einen kontinuierlichen Kreislauf bestehend aus alternierenden Phasen, in denen einzelne oder alle Hardwarekomponenten (exklusive des Zeitgebers) ausgeschaltet werden (Schlafphase), sowie Phasen, in denen diese Komponenten eingeschaltet und betriebsbereit sind (Wachphase). Mit zunehmender Länge der einzelnen Schlafphasen kann der Energieverbrauch reduziert werden. Gleichermaßen werden jedoch die Ressourcen eines Sensorknotens eingeschränkt, da in den Schlafphasen weder Programmlogik ausgeführt noch Nachrichten ausgetauscht werden können.

Eine weitere wichtige Anforderung an Sensornetze sind niedrige Kosten. Durch ihre einfache Installation und ihren autarken Betrieb verursachen WSNs im Vergleich zu kabelgebundenen Messsystemen geringere Installations- und Betriebskosten. Da Sensornetze jedoch aus einer Vielzahl an Knoten bestehen, stellen ihre Stückkosten einen weiteren entscheidenden Kostenfaktor dar. Visionäre sehen Sensorknoten für die Zukunft als kostengünstige Wegwerfartikel [26]. Um dieses Ziel zu erreichen, bestehen Knoten aus einfachen und somit kostengünstigen, aber ressourcenarmen Hardwarekomponenten.



(a) Smart Dust [173]



(b) TelosB [33]

(c) iSense [27]

(d) Pacemate [67]

Abbildung 2.12: Verschiedene Sensornetzplattformen

Eine weitere wichtige Anforderung an Sensorknoten ist ihre Baugröße. Damit sie unauffällig in die entsprechenden Messumgebungen eingebettet werden können und die physikalischen Prozesse nicht beeinflussen, müssen Knoten möglichst klein sein. Die Vision ist es, die Baugröße auf die Größe von Staubkörnern zu reduzieren. Bereits heute existieren Sensorknoten, die einen Durchmesser von unter einem Zentimeter aufweisen. Abbildung 2.12a zeigt einen als *Smart Dust* (dt.: intelligenter Staub) bezeichneten Sensorknoten, der einen Durchmesser von wenigen Millimetern aufweist [173]. Um diese Größenbeschränkungen zu erfüllen, ist es genau wie zur Erfüllung der Energie- und Kostenrestriktionen notwendig, einfache und kompakte, aber ressourcenbeschränkte Hardware einzusetzen.

Wie eben erläutert, sollen einzelne Sensorknoten möglichst klein sein, und die Beschaffung, Installation sowie der Betrieb eines Sensornetzes sollen kostengünstig, langlebig und autark sein [26, 128]. Die Erfüllung dieser Anforderungen führt automatisch zu einer Einschränkung der Ressourcen von Sensornetzen. Entsprechend sind Sensorknoten meist durch eine geringe Rechenleistung, einen kleinen permanenten und flüchtigen Speicher sowie durch eine niedrige Datenrate charakterisiert. Im Rahmen dieser Arbeit kommen mit den *TelosB*- [33], *iSense*- [27] und *Pacemate*-Knoten [116] drei verschiedene Sensornetzplattformen zum Einsatz. Die TelosB-Sensorknoten (s. Abbildung 2.12b) verfügen über einen 16-Bit-RISC-Prozessor, der mit 8 MHz operiert. Des Weiteren verfügt die Plattform über einen IEEE-802.15.4- [85] und ZigBee-kompatiblen [270] Transceiver mit einer Übertragungsrate von 250 kbit/s, 48 KB Programm-Flash, 1 MB externen Flash sowie 10 KB RAM. Bei iSense (s. Abbildung 2.12c) handelt es sich um eine modulare Soft- und Hardwareplattform, dessen Basismodul über einen IEEE-802.15.4-kompatiblen Transceiver mit einer Übertragungsrate von 250 kbit/s, einen 32-Bit-RISC-Prozessor mit 16 MHz und über einen 96 KB RAM- sowie über einen 128 KB Flash-Speicher verfügt. Die Pacemate-Plattform (s. Abbildung 2.12d) verfügt über einen Phillips 32-Bit-LPC-2136-Prozessor mit 60 MHz, 32 KB RAM- und 256 KB Flash-Speicher. Der Transceiver erreicht eine Übertragungsrate von 152 kbit/s.

Kommunikation in Sensornetzen

Wie in Abschnitt 2.4.1 beschrieben, ist die grundlegende Idee von Sensornetzen die kollaborative Überwachung physikalischer Phänomene. Um dieses Ziel zu erreichen, bildet das

Sensornetz zusammen mit dem Backend ein verteiltes System, in dem drei verschiedene Kommunikationsarten unterschieden werden können:

1. Kommunikation zwischen verschiedenen Backend-Komponenten.
2. Kommunikation zwischen verschiedenen Sensorknoten.
3. Kommunikation zwischen Backend-Komponenten und Sensorknoten.

Heutige verteilte Systeme und Protokolle des Backends basieren auf dem OSI- (*Open Systems Interconnection Reference Model*) oder *TCP/IP-Referenzmodell* [211,213]. Beide Modelle teilen verschiedene Kommunikationsaufgaben auf einzelne Ebenen mit klar definierten Teilaufgaben auf. Dadurch lässt sich ein hohes Maß an Flexibilität und eine Reduktion der Komplexität der einzelnen Teilaufgaben erreichen. Zur Kommunikation zwischen verschiedenen Backend-Komponenten werden heutzutage vor allem Protokolle verwendet, die mit dem TCP/IP-Referenzmodell verknüpft sind. Das sind weitverbreitete Internetprotokolle, die auf der Transportebene auf TCP (*Transmission Control Protocol*, [176]) oder UDP (*User Datagram Protocol*, [174]) aufbauen. TCP und UDP verwenden IP (*Internet Protocol*, [175]) auf der Vermittlungsebene.

Obwohl es mit 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*, [106]) oder μIP [44] auch die Möglichkeit gibt, IP [36, 175] in WSNs zu realisieren, wird aufgrund des hohen Ressourcenverbrauchs in den meisten Sensornetzapplikationen auf den Einsatz von IP verzichtet [46, 82]. Ebenfalls resultierend aus den hohen Ressourcenanforderungen ist der Einsatz von Transport- oder gar Anwendungsprotokollen des TCP/IP-Referenzmodells absolut unmöglich.

Neben der Verwendung konkreter Protokolle unterscheidet sich die Kommunikation in Sensornetzen auch konzeptionell von der im Backend bzw. von klassischen, internetbasierten verteilten Systemen. Auch bei der Verwendung alternativer Protokolle ist die klare Trennung der Schichten analog zum OSI- oder TCP/IP-Referenzmodell im WSN nicht möglich oder sinnvoll. Die Reduktion des Verbrauchs unterschiedlicher Ressourcen bedarf eines schichtenübergreifenden oder gar applikationsspezifischen Wissens. So steuert z. B. die Applikation das Duty Cycling eines Knotens. Um trotz der daraus resultierenden temporären Diskonnektivität ihre Aufgaben korrekt und effizient ausführen zu können, ist dieses applikationsspezifische Wissen für die einzelnen Schichten essenziell. Darüber hinaus werden in einem WSN gegenwärtig nicht wie in klassischen Netzen eine Vielzahl verschiedener, konkurrierender, sondern eine oder wenige zusammengehörige Anwendungen ausgeführt. Dementsprechend werden Kommunikationsprotokolle auf das Gesamtziel der Anwendung hin optimiert. Folglich müssen Protokolle aller Schichten optimal an die Applikation angepasst werden. Um diese Anforderungen zu erfüllen, bauen WSN-Applikationen häufig direkt auf unteren Ebenen (z. B. der Medienzugriffsebene) des Referenzmodells auf oder verwenden Kommunikationsmodelle mit unterschiedlichsten, an die jeweilige Applikation angepassten, ebenenübergreifenden Protokollen.

Orientiert an diesen Besonderheiten statten viele Hersteller ihre Sensorknoten mit IEEE-802.15.4-kompatiblen [85] Transceivern aus. IEEE 802.15.4 spezifiziert die Bitübertragung und Medienzugriffssteuerung. Dabei erlaubt der Standard maximale Paketgrößen von 127 Byte bei einer Datenrate von 250 kbit/s. Für die Standardisierung höherer Schichten ist der

ZigBee-Standard [270] zu nennen, der aufgrund der o.g. Gründe jedoch meist nicht in WSN-Anwendungen Verwendung findet [171].

Die Verwendung unterschiedlicher Kommunikationsprotokolle im WSN und Backend verhindert eine transparente Ende-zu-Ende-Kommunikation zwischen Komponenten beider Teilnetze. Aus diesem Grund ist der Einsatz von Gateways zur Integration notwendig. Aufgrund der engen Verknüpfung der Sensornetzanwendung mit den Kommunikationsaufgaben muss auch das Gateway über entsprechendes Anwendungswissen verfügen. Aus diesem Grund werden zur Integration schwergewichtige, applikationsspezifische Gateways verwendet.

Einbettung von Sensornetzen in ihre Messumgebung

Sensornetze werden zur Überwachung physikalischer Phänomene in ihre Messumgebung eingebettet. Sie sind für den Benutzer weitestgehend unsichtbar und agieren in der Regel vollständig ohne oder zumindest ohne direkte Benutzerinteraktion. Sensorknoten verfügen über keine oder nur über sehr eingeschränkte Möglichkeiten zur Ein- und Ausgabe von Daten. Damit stellen WSNs ein *eingebettetes System* dar.

Die Entwicklung von Software für eingebettete Systeme unterscheidet sich wesentlich von der für Arbeitsplatz- oder Serverrechner. Die Software kann nicht auf der Zielplattform entwickelt werden. Stattdessen wird sie auf einem Entwicklungssystem implementiert. Um die Software auf der Zielplattform ausführen zu können, muss sie mit einem Cross-Compiler kompiliert sowie in einem zeitraubenden Vorgang auf die Sensorknoten übertragen und auf ihnen installiert werden. Zusätzlich erschweren die o. g. fehlenden Ein- und Ausgabemöglichkeiten die Durchführung von Tests. Diese Besonderheiten führen dazu, dass der Entwicklungs- und Testaufwand bei eingebetteten Systemen wesentlich höher ist.

3 Geschäftsprozessmanagement im Kontext von drahtlosen Sensornetzen

Drahtlose Sensornetze sollen eine wesentliche Komponente des zukünftigen Internets darstellen. Zusammen mit einer Vielzahl an weiteren eingebetteten Geräten sollen sie das Internet der Dinge (engl.: *Internet of Things*, IoT) bilden. In der Vision des IoT werden verschiedenartige Geräte, von ressourcenbeschränkten Sensorknoten bis zu leistungsstarken Unternehmensservern, das Internet mit der realen Welt verbinden [127]. Nach Meinung des Autors dieser Arbeit kann so die nahtlose und flexible Integration von Sensornetzen die Basis für völlig neuartige Anwendungen und Geschäftsprozesse schaffen und so analog zu Abbildung 1.1 auf Seite 1 die Funktion des *Ermöglichens* neuartiger Geschäftsprozesse und Unternehmensstrategien einnehmen.

Trotz ihres Potenzials sind WSNs kein Bestandteil heutiger Unternehmensanwendungen. Das reine Potenzial, dass sich mithilfe von Sensornetzen neuartige fachlich-konzeptionelle Geschäftsprozesse generieren lassen, ist lediglich eine notwendige, jedoch keine hinreichende Voraussetzung, um tatsächlich Wettbewerbsvorteile durch ihre Integration in Geschäftsprozesse zu erzielen. Die Gestaltung von WSN-Applikationen sowie ihre Integration in die existierenden Geschäftsprozesse eines Unternehmens müssen zusätzlich, genau wie jegliche Workflows in klassischen Unternehmensanwendungen, eine schnelle und flexible Anpassung an sich ändernde Unternehmensstrategien gewährleisten. Das bedeutet, dass Sensornetze die Funktion des *Ermöglichens* nur in Kombination mit der Fähigkeit zur *Anpassung* einnehmen können.

Mit den in Kapitel 2 beschriebenen Grundlagen des Geschäftsprozessmanagements, der serviceorientierten Architekturen sowie verschiedener Webservice-Technologien existieren bereits heute für den Enterprise-IT-Bereich hinreichende konzeptionelle und technologische Lösungen zur flexiblen informationstechnischen Realisierung von Geschäftsprozessen bzw. zur Anpassung von Workflows und Informationssystemen an die Unternehmensstrategie. Ein Geschäftsprozessmanagement im Kontext von Sensornetzen ist heute jedoch noch unmöglich. Das Ziel dieser Arbeit ist es, diesen Mangel durch die Entwicklung konzeptioneller und technischer Lösungen zu beheben und ein ganzheitliches Geschäftsprozessmanagement für Sensornetze sowie Enterprise-IT-Systeme zu realisieren.

Zu Beginn dieses Kapitels wird in Abschnitt 3.1 mit einem Logistikprozess zur Überwachung von Gefahrguttransporten ein realer Anwendungsfall aus dem Projekt *Lübecker Logistikdatendrehscheibe* (L2D2) vorgestellt. Er dient der Ableitung von Anforderungen, die bei der Integration von WSNs in heutige komplexe Prozesse erfüllt werden müssen und wird dazu verwendet, die im Rahmen dieser Arbeit entwickelten Konzepte und Technologien in einem realen Szenario zu evaluieren. Anschließend werden in Abschnitt 3.2 die Anforderungen an ein Geschäftsprozessmanagement im Kontext von Sensornetzen definiert, bevor in Abschnitt 3.3

der aktuelle Stand der Wissenschaft bei der WSN-Anwendungsentwicklung beschrieben und gezeigt wird, warum sich weder die verwendeten Konzepte noch die entsprechenden Technologien für eine erfolgreiche Integration von Sensornetzen in Geschäftsprozesse eignen. Als Konsequenz daraus wird in Abschnitt 3.4 das vom Autor dieser Arbeit entwickelte Konzept für ein dynamisches, ganzheitliches Geschäftsprozessmanagement in Sensornetzen und im Enterprise-IT-Umfeld präsentiert. Es *ermöglicht* neuartige Geschäftsprozesse bei gleichzeitiger Gewährleistung der Fähigkeit zur *Anpassung*.

3.1 Anwendungsfall Logistik

In diesem Abschnitt wird ein konkreter Anwendungsfall präsentiert, der an einem Beispielprozess aus der Domäne Logistik zeigt, wie sich durch die Integration von Sensornetzen in Geschäftsprozesse völlig neuartige Prozesse und Wettbewerbsvorteile generieren lassen. Dieser Anwendungsfall wurde aus zwei Gründen bewusst gewählt. Zum einen sind die Chancen, die sich aus der Integration von WSNs in Logistikprozesse ergeben, besonders hoch, da eine Vielzahl an Einsatzmöglichkeiten besteht. Zum anderen sind auch die Anforderungen der Logistik sehr hoch, sodass Lösungen, die in dieser Domäne anwendbar sind, einen hohen Generalitätsgrad aufweisen.

In der Logistik ist eine Vielzahl an Unternehmen mit der Produktion, Lagerung und dem Versand von Gütern beschäftigt. Die (oft manuelle) Erfassung von Informationen über den Zustand und den Standort der transportierten Güter für die Weiterverarbeitung in Unternehmensanwendungen ist ein sehr aufwendiger Prozess. Durch die Integration von Sensorknoten in die Güter bzw. die entsprechenden Transporteinheiten könnten diese Informationen automatisch und in Echtzeit erfasst und in Unternehmensanwendungen zur Verfügung gestellt werden. Dadurch könnte der Erfassungsaufwand reduziert, gleichzeitig die Datenqualität erhöht und völlig neuartige Prozesse generiert werden.

Im Rahmen dieser Arbeit wurde als konkreter Logistikprozess die Überwachung von Gefahrguttransporten im kombinierten Verkehr über den Hafen Lübeck gewählt und im Rahmen des L2D2-Projekts vom Autor dieser Arbeit realisiert. Das Ziel dieses Prozesses ist die Überwachung des Zustands und der Position von transportierten Gefahrgütern, um Gefahrgutunfälle proaktiv vermeiden oder zumindest schnell darauf reagieren zu können. Zusätzlich soll mit dieser Anwendung die Überwachung der Einhaltung gesetzlicher Gefahrguttransportvorschriften realisiert werden. Alle am Prozess beteiligten und berechtigten Partner in dieser logistischen Transportkette haben jederzeit über eine Webschnittstelle sowie über entsprechende Dienste einen Zugriff auf alle Prozess- und Sensordaten.

Abbildung 3.1 zeigt im Überblick diesen Geschäftsprozess. Jedes Gefahrgut, das über den Hafen Lübeck verschifft wird, muss bei den Reedereien und beim Hafenamt deklariert werden und darf nur nach entsprechender Genehmigung in den Hafen eingeführt werden. Jeder Prozess beginnt mit einer entsprechenden Gefahrgutanmeldung und der Beladung des LKW/ Trailers sowie der Ausstattung mit einem oder mehreren Sensorknoten. Anschließend wird das Gefahrgut per LKW zum Hafen transportiert. In dieser Phase werden für jeden Transport isoliert die entsprechenden Gefahrgüter mit Sensorknoten überwacht und die Informationen im Backend gespeichert. Beim Erreichen des Hafen-Gates melden die Sensorknoten den Transport automatisch am Gate zur Abfertigung an. Nach der Abfertigung durch menschliche

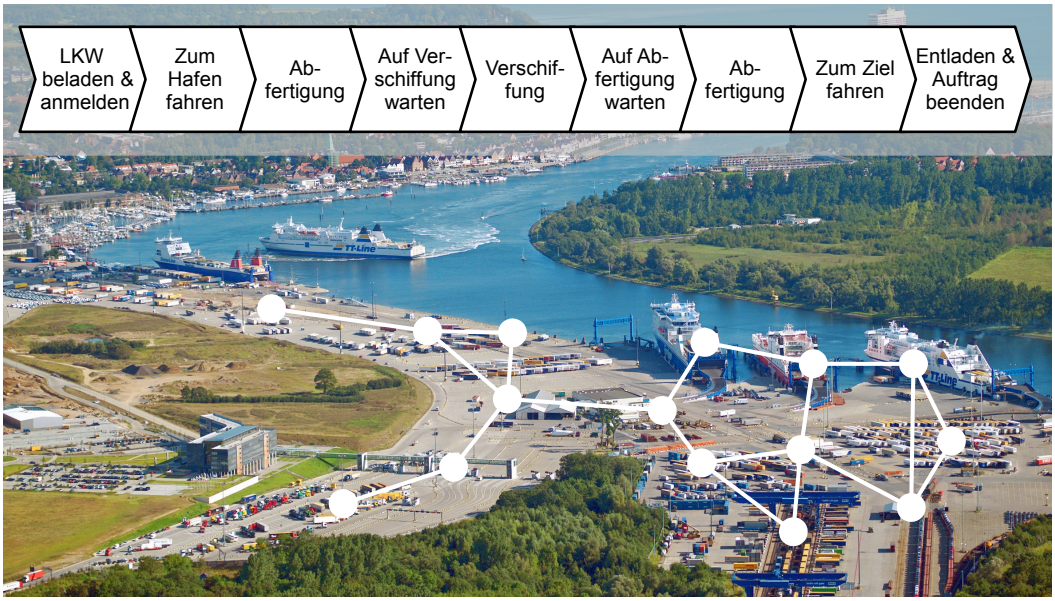


Abbildung 3.1: Gefahrgutüberwachung mit Sensornetzen im kombinierten Verkehr (Quelle Hintergrundbild: TraDaV GmbH)

Aufgabenträger wird einem Transport ein bestimmter Parkplatz im Hafen zugeordnet, auf dem er auf die Verladung auf das entsprechende Schiff wartet. Dabei gelten im Hafengebiet (anders als auf der Straße) sehr strenge Vorschriften. Z. B. müssen einige Gefahrguttransporte bestimmte Mindestabstände zu anderen einhalten. Um diese Regeln kollaborativ zu überprüfen, bilden die Sensorknoten verschiedener Gefahrguttransporte wie in Abbildung 3.1 gezeigt ad hoc ein Sensornetz. Die Überprüfung der Regeln erfolgt durch die einzelnen Knoten. Lediglich im Fall der Verletzung einer Vorschrift wird ein Alarm an das Backend gesendet. Nach der Verschiffung erfolgt der bisherige Teilprozess wieder in umgekehrter Reihenfolge. Der Transport wartet im Hafengebiet evtl. auf Abholung durch einen LKW; nach der Abholung und Abfertigung wird der Gefahrguttransport per LKW an seinen Zielort transportiert. Mit der Entladung endet der Prozess der Gefahrgutüberwachung.

Neben den besonders hohen Chancen, die sich bei der Integration von WSNs in Logistikprozesse ergeben, wurde die Logistikdomäne sowie im Speziellen dieser Anwendungsfall auch aufgrund der besonders hohen Anforderungen ausgewählt. Logistische Wertschöpfungsketten zeichnen sich durch eine extrem hohe Dynamik und durch eine Vielzahl an Marktteilnehmern aus. In den langen Wertschöpfungsketten werden verschiedene Teilaufgaben bzw. -prozesse von unterschiedlichen, organisational unabhängigen Aufgabenträgern durchgeführt. So sind in dem o. g. Beispiel Speditionen, Hafenbetreiber, Reedereien und Hafenbehörden sowie die eigentlichen Versender der Güter involviert. Das bedeutet, dass hier Geschäftsprozesse und IT-Systeme in einer sehr heterogenen Umgebung organisationsübergreifend integriert werden müssen. Aber nicht nur horizontal entlang der Wertschöpfungskette sind verschiedene Akteure beteiligt. Auch vertikal werden unterschiedliche Teilprozesse durch unterschiedliche Unternehmen ausgeführt. Diese können mit anderen Unternehmen kooperieren oder als deren Konkurrenten auftreten. Darüber hinaus sind die Gesetze und Regelungen (insbesondere bei internationalen Wertschöpfungsketten) ständigen Änderungen unterworfen. Auch än-

dern sich die Teilnehmer bestimmter Lieferketten durch die Änderung der Marktsituation oder Optimierung der eigenen Geschäftsprozesse sehr dynamisch. Das bedeutet, dass die informationstechnischen Prozessrealisierungen bzw. die IT-Systeme flexibel anpassbar sein müssen.

Als zusätzliche erschwerende Anforderung kommt hinzu, dass sich in einzelnen Subprozessen die Verantwortungsbereiche vermischen. So bieten z. B. Speditionen eine Plattform zum Transport von Waren an. Das beinhaltet den LKW mit Trailer genauso wie die darauf installierten Sensorknoten. Die Verantwortung für die Software zur Überwachung des Transportes könnte beim Spediteur liegen. Genauso ist es aber auch vorstellbar, dass der Kunde seine eigene Prozesslogik auf den angemieteten Knoten ausführen möchte. In diesem Fall würde der Kunde die Plattform zur Ausführung von Workflows als Dienstleistung (engl.: *Platform as a Service*) dazu mieten.

Der hier präsentierte Anwendungsfall dient einerseits dazu, zu zeigen, dass die konzeptionellen und technischen Anforderungen an die Entwicklung von Sensornetzapplikationen von den heutigen Lösungen nicht erfüllt werden. Andererseits wurden anhand dieses Beispiels die im Rahmen dieser Arbeit entstandenen Konzepte und Technologien zur Integration von Sensornetzen in Geschäftsprozesse auf ihre Anwendbarkeit in der Unternehmenspraxis überprüft.

3.2 Anforderungen an drahtlose Sensornetze und Enterprise-IT-Systeme

Wie bereits beschrieben, können Sensornetze nur erfolgreich in ein Geschäftsprozessmanagement integriert werden, wenn sie genau wie Enterprise-IT-Systeme ihre schnelle und flexible Anpassung an sich ändernde fachlich-konzeptionelle Geschäftsprozesse erlauben. Wie in Abschnitt 2.2 allgemein beschrieben und in Abschnitt 3.1 an einem konkreten Anwendungsfall belegt, ist die Flexibilität der Systemarchitektur für die informationstechnische Umsetzung von Geschäftsprozessen die zentrale Anforderung an die realisierenden Anwendungssysteme. Um Flexibilität zu erreichen, muss eine monolithische Architektur vermieden und die Gesamtanwendungsfunktionalität modular als in sich abgeschlossene und lose gekoppelte Funktionseinheiten (z. B. Dienste) strukturiert werden. Sie müssen die Möglichkeit bieten, in beliebigen Softwarekomponenten (wieder-)verwendet zu werden. Dieses gilt nicht nur für das Backend, sondern auch für die von Sensorknoten bereitgestellte und konsumierte Funktionalität.

In Abschnitt 2.2.4 wurde bereits erläutert, dass die Funktionskapselung, Wiederverwendbarkeit sowie Kompositionsfähigkeit eine notwendige, aber keine hinreichende Voraussetzung für eine flexible Realisierung von Workflows ist. Genauso wichtig für die Erreichung von Flexibilität ist die Art der Einbindung bzw. Komposition von vorhandener Anwendungslogik sowie die Erstellung neuer Logik. Durch die Verwendung deklarativer Sprachen zur Umsetzung und Aggregation von Prozesslogik können der Komplexitätsgrad des Realisierungsprozesses sowie die Fehlerrate und die Entwicklungszeiten stark reduziert werden. Die Erfüllung dieser Anforderung ist im WSN sogar noch wichtiger als in Enterprise-IT-Systemen, da diese Kennzahlen aufgrund der besonderen charakteristischen Merkmale von Sensornetzen (Verteiltheit,

aber insbes. Ressourcenbeschränktheit und Einbettung in die Messumgebung) in diesen Umgebungen relativ ungünstig ausfallen. Neben der Verwendbarkeit durch Fachpersonal wird von einer entsprechenden Sprache zusätzlich erwartet, dass es für den Entwickler vollständig transparent ist, ob er Logik zur Ausführung auf Sensorknoten oder im Backend realisiert. Im Idealfall soll es sogar möglich sein, nach der Prozessrealisierung die Ausführungsplattform beliebig auszutauschen.

Als dritte und letzte technische Anforderung ist das Kommunikationsverhalten von entscheidender Bedeutung. In diesem Zusammenhang soll eine vollständige Verteilungstransparenz erreicht werden. Das betrifft Kommunikationsanforderungen auf zwei aufeinander aufbauenden Ebenen. Zuerst muss ein transparenter Ende-zu-Ende-Nachrichtenaustausch zwischen allen Systemkomponenten ermöglicht werden, der die folgenden vier Kommunikationsmuster unterstützt:

1. Kommunikation zwischen Sensorknoten innerhalb desselben Sensornetzes.

2. Kommunikation zwischen Sensorknoten und Komponenten in Enterprise-IT-Systemen, also im Internet.

3. Kommunikation zwischen verschiedenen Enterprise-IT-Komponenten.

4. Kommunikation zwischen Sensorknoten in unterschiedlichen, entfernten Sensornetzen.

Aufbauend auf einem Nachrichtenaustausch, der o. g. Voraussetzungen erfüllt, muss als zweite Kommunikationsanforderung ein Zugriff auf verteilte Softwarekomponenten ermöglicht werden. Auch in diesem Zusammenhang muss eine vollständige Transparenz erreicht werden. Das bedeutet, dass es zum einen keinen Unterschied machen darf, auf welcher Geräteart bzw. in welchem Teilnetz sich der Nutzer oder Erbringer von verteilter Funktionalität befindet. Zum anderen muss beiden (weitestgehend) verborgen bleiben, dass überhaupt verteilte Softwarekomponenten aufgerufen werden.

Zusätzlich zu diesen technischen Aspekten müssen jedoch auch nicht technische sowie betriebswirtschaftliche Anforderungen betrachtet werden. Lösungen zur Realisierung eines Geschäftsprozessmanagements im Kontext von WSNs können nur erfolgreich sein, wenn sie kompatibel zu gängigen, bereits im Rahmen eines Geschäftsprozessmanagements in Enterprise-IT-Systemen eingesetzten offenen Standards sind. Neben dem betriebswirtschaftlichen Interesse des Schutzes bereits geleisteter Investitionen verlangt auch eine organisationsübergreifende Prozessintegration sowie die Realisierung der Ausführung von Prozessen auf angemieteten Plattformen (z. B. auf den Sensorknoten von angemieteten LKWs) den Einsatz bereits verbreiteter Lösungen und Standards.

Es ergeben sich also mit der Flexibilität der Systemarchitektur, dem geringen Komplexitätsgrad der Logikrealisierung, verschiedenen Kommunikationsanforderungen sowie der Kompatibilität zu etablierten Standards zur Realisierung von Geschäftsprozessen vier Anforderungsbereiche, die von Sensornetzen und Enterprise-IT-Systemen gleichermaßen erfüllt werden müssen. Nur wenn ausschließlich alle Anforderungen erfüllt sind, kann ein Gesamtsystem aus Sensornetzen und Enterprise-IT-Systemen die für ein Geschäftsprozessmanagement notwendige Anpassbarkeit gewährleisten.

Abstraktion	Systemarchitektur	Logikrealisierung	Kommunikation	Verwendung von Standards
Betriebssystem	−	−	−	−
Systemprogrammierung	−	◯	−	−
Datenzentrische Middleware	−	◯	−	−
Serviceorientierte Middleware	−	−	−	◯

(+: erfüllt; ◯: teilweise erfüllt; −: nicht erfüllt)

Tabelle 3.1: Bewertung der Eignung von WSN-Programmierabstraktionen für die Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen

3.3 Bewertung aktueller Abstraktionen zur Anwendungsrealisierung in Sensornetzen

Die Entwicklung von klassischen verteilten und eingebetteten Systemen ist bereits eine große Herausforderung für Softwareentwickler [109]. Durch die extremen Ressourcenbeschränkungen, denen WSNs unterliegen, wird die Komplexität ihrer Anwendungsentwicklung noch einmal erhöht. In klassischen verteilten und eingebetteten Systemen wird durch den Einsatz von Programmierabstraktionen versucht, diese Komplexität vor den Programmierern zu verbergen. Der Einsatz von Programmierabstraktionen geht meist jedoch mit einem zusätzlichen Ressourcenverbrauch einher. Gerade die in klassischen eingebetteten und verteilten Systemen verwendeten Programmierabstraktionen verursachen einen zu großen Overhead, um in WSNs eingesetzt zu werden. Aus diesem Grund werden speziell für WSNs entwickelte Middleware-Lösungen eingesetzt oder z. T. eine Anwendungsentwicklung sogar direkt auf dem Betriebssystem aufgesetzt [128].

In diesem Abschnitt werden verschiedene Programmierabstraktionen für Sensornetze vorgestellt und anhand der im vorigen Abschnitt beschriebenen Anforderungen bewertet. Tabelle 3.1 fasst die Ergebnisse zusammen und zeigt, ob die Systemarchitektur, die Logikrealisierungs- und Kommunikationscharakteristika sowie der Einsatz von etablierten Standards aus dem Workflowmanagement eine Integration von Sensornetzen in ein Geschäftsprozessmanagement auf Basis der jeweiligen Abstraktion erlauben.

3.3.1 Betriebssystem

Die Programmierabstraktion mit dem geringsten Overhead ist das Betriebssystem. Es abstrahiert den Zugriff auf die verwendete Hardware und kontrolliert sowie koordiniert die Zuteilung der auf dem Gerät zur Verfügung stehenden Ressourcen [212]. Das am weitesten verbreitete Betriebssystem für Sensornetze ist *TinyOS* [110]. Es zeichnet sich durch einen modularen Aufbau aus, der die flexible Anpassung des Betriebssystems an die Anwendungsanforderungen erlaubt. Jegliche Anwendungsfunktionalität wird ebenfalls als Modul realisiert und kann in anderen Modulen verwendet werden. Das schafft auf den ersten Blick ein gewisses Flexibilitätsniveau. Allerdings ist die Aggregation von verschiedenen Modulen auf einen einzelnen Sensorknoten beschränkt und ermöglicht keinen direkten Aufruf von entfernten Modulen. Auch weisen Modulaggregationen einen sehr engen Kopplungsgrad auf und reduzieren so die

Flexibilität. Diese wird noch durch die Art der Programmierung weiter eingeschränkt. Alle TinyOS Module werden in der extra für dieses Betriebssystem entwickelten Sprache nesC [63] programmiert. nesC stellt eine Obermenge zur Programmiersprache C dar und ist durch einen sehr hohen Komplexitätsgrad gekennzeichnet. Dadurch wird die Flexibilität des Systems weiter reduziert und eine Realisierung von Prozesslogik durch Fachpersonal verhindert.

TinyOS stellt eine Vielzahl an Kommunikationsprotokollen zur Verfügung. Neben zahlreichen WSN-Protokollen ermöglicht 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*, [106]) sogar eine Kommunikation nach dem Internetstandard IPv6 [36]. Damit gestattet TinyOS eine Ende-zu-Ende-Kommunikation zwischen Sensorknoten und Enterprise-IT-Komponenten. Allerdings ist 6LoWPAN auf den reinen Nachrichtenaustausch beschränkt. Zum Aufruf von über das Netz verteilter Anwendungsfunktionalität stellt TinyOS keine Lösung zur Verfügung, sodass der Anwendungsprogrammierer dieses manuell realisieren muss. Das erhöht den Entwicklungsaufwand und schränkt die Flexibilität sowie Interoperabilität ein.

Auch neuere Betriebssysteme für Sensorknoten wie das von Lipphardt et al. vorgestellte *SuferOS* [114,115], die explizit auf eine flexible Anpassbarkeit und die Anwendungsentwicklung durch Nichtsensornetzexperten hin entworfen wurden, genügen nicht den Anforderungen eines Geschäftsprozessmanagements. Zwar wird jede Anwendungsfunktionalität als in sich abgeschlossenes Modul gekapselt. Die Implementierung und Komposition dieser Module muss jedoch in der Programmiersprache C realisiert werden. Dabei wird der ohnehin schon zu hohe Komplexitätsgrad von C noch weiter dadurch erhöht, dass der Programmierer verstärkt Funktionsreferenzen (engl.: *Function Pointer*) verwenden muss. Dadurch ist die Programmierung sowie die Anpassung/Entwicklung von Anwendungsfunktionalität nicht für ein Geschäftsprozessmanagement geeignet. Auch SurferOS stellt ähnlich wie TinyOS eine Reihe von Kommunikationsprotokollen zur Verfügung. Keines ermöglicht jedoch eine Ende-zu-Ende-Kommunikation mit Enterprise-IT-Komponenten. Darüber hinaus fehlt auch eine Abstraktion für den Aufruf von entfernter Anwendungsfunktionalität.

Mit *Contiki* [45], *Mantis* [3], *iSense* [27] oder *SOS* [77] existiert neben TinyOS und SurferOS eine Vielzahl weiterer Betriebssysteme für Sensorknoten. Sie unterscheiden sich jedoch in den für ein Geschäftsprozessmanagement relevanten Eigenschaften nicht wesentlich von TinyOS und werden aus diesem Grund in dieser Arbeit nicht näher betrachtet. Auch virtuelle Maschinen wie *Maté* [111] unterscheiden sich in diesem Kontext nicht wesentlich von Betriebssystemen und werden deshalb ebenfalls nicht näher betrachtet. Eine virtuelle Maschine stellt eine zusätzliche Abstraktion oberhalb des Betriebssystems dar, die nur eine bestimmte Instruktionsmenge erlaubt. Ihr einziger für ein Geschäftsprozessmanagement relevanter Vorteil gegenüber Betriebssystemen ist die Möglichkeit, Code auf heterogenen Plattformen auszuführen und eine Codemigration zu realisieren. Damit könnte die organisationsübergreifende Bereitstellung von Sensornetzhardware leichter realisiert werden.

Zusammenfassend können WSN-Betriebssysteme und virtuelle Maschinen als nicht ausreichende Abstraktion für ein Geschäftsprozessmanagement bewertet werden. Zur fehlenden Flexibilität, komplexen Anwendungsentwicklung und fehlenden Verteilungstransparenz kommt als zusätzliches Defizit hinzu, dass mit dieser Abstraktion keine Kompatibilität zu Workflowrealisierungsstandards der Unternehmens-IT erreicht wird.

3.3.2 Systemprogrammierung

Eine weitere in WSNs verwendete Programmierabstraktion ist die Systemprogrammierung. Die Idee hinter dieser Abstraktion ist, dass Anwendungsfunktionalität nicht mehr auf der Ebene einzelner Knoten, sondern für das Sensornetz als Ganzes realisiert wird [171]. Anders als bei Betriebssystemen wird durch die Ableitung des Verhaltens einzelner Knoten aus der Beschreibung des Gesamtsystems sowie der damit einhergehenden transparenten Kommunikation die Komplexität der Programmierung reduziert und eine Verteilungstransparenz erreicht. Die verwendeten Programmiersprachen sind für die Verwendung von Fachpersonal immer noch zu komplex. Der bedeutendste Nachteil dieser Programmierabstraktion ist jedoch, dass die WSN-Applikation als ein monolithischer Block realisiert wird und damit die Flexibilitätsanforderungen an die Systemarchitektur nicht erfüllt. Da sich das WSN lediglich als Ganzes in die Enterprise-IT integrieren lässt, ist die Verwendung von proprietären Kommunikationsprotokollen innerhalb des Sensornetzes bei dieser monolithischen Architektur irrelevant. Jedoch werden auch hierfür keine Standards aus dem Workflowmanagement verwendet. Eine Integration in Enterprise-IT-Systeme ist nur über schwergewichtige, anwendungsspezifische und damit unflexible Gateways möglich.

3.3.3 Datenzentrische Middleware

Wie in Abschnitt 2.4.1 beschrieben, stellt die Erfassung und Weiterleitung von Sensordaten an das Backend die zentrale Aufgabe heutiger Sensornetze dar. Um dieser Eigenschaft gerecht zu werden, wird häufig eine datenzentrische Middleware als Programmierabstraktion zwischen Betriebssystem und Anwendung eingesetzt [31, 121, 180, 262]. Lösungen wie z. B. *TinyDB* [121] oder *Cougar* [262] ermöglichen dem Entwickler von Backend-Applikationen über spezielle Anfragesprachen den einfachen Zugriff auf und die Aggregation von Sensordaten. Die verwendeten Anfragesprachen erlauben einen einfachen, deklarativen Datenzugriff. Sie sind z. T. an die zum Zugriff auf relationale Datenbanken verwendete Sprache SQL (*Structured Query Language*, [89]) angelehnt und damit leicht für IT-Personal erlernbar.

Um nun tatsächlich Daten aus dem Sensornetz abzufragen, werden Anfragen vom Backend an ein Gateway gesendet. Dieses agiert als Datenbankmanagementsystem, interpretiert die Anfragen und führt diese für den Entwickler transparent aus. Dabei bleibt der Abruf der Sensordaten von den Knoten sowie die komplette dazugehörige Kommunikation dem Backend-Entwickler verborgen.

Der große Vorteil einer datenzentrischen Abstraktion ist die Einfachheit der Programmierung. Zwar lässt sich dadurch auch ein gewisses Maß an Flexibilität erzielen. Dieses ist jedoch auf das Backend beschränkt. Das WSN selbst stellt ein monolithisches, geschlossenes System dar, das nicht von Anwendungsentwicklern verändert werden kann. Bei der Verwendung einer datenzentrischen Middleware kann ein WSN lediglich in der Datenebene einer n-Ebenen-Architektur eingeordnet werden, da es nur eine Abfrage- und keine Programmierschnittstelle bietet. Die für ein Geschäftsprozessmanagement nötige Ausführung von Prozesslogik auf Sensorknoten ist mit dieser Abstraktion nicht möglich.

Da Anwendungsentwickler nur mit dem Gateway kommunizieren, ist die Verwendung von proprietären Kommunikationsprotokollen innerhalb des Sensornetzes ohnehin nicht mehr relevant.

Aber auch die Integration in das Backend ist nicht zu den Workflowrealisierungsstandards von Enterprise-IT-Systemen kompatibel.

3.3.4 Serviceorientierte Middleware

Alle bisher beschriebenen Programmierabstraktionen verfügen nicht über eine für ein Geschäftsprozessmanagement ausreichende Flexibilität. Wie in Abschnitt 2.1 beschrieben, stellen serviceorientierte Architekturen die ideale Systemarchitektur zur informationstechnischen Realisierung von Geschäftsprozessen dar. Mit serviceorientierten Middleware-Lösungen für Sensornetze wurde versucht, durch die Verwendung dieses Konzepts in WSNs, auch hier die Anwendungsentwicklung flexibler zu gestalten. Entsprechende Ansätze werden von Blumenthal et al. [23], Marin-Prianu et al. [126], Delicato et al. [37] und Kushwaha et al. [107] bzw. Amundson et al. [6] vorgeschlagen. In allen Arbeiten wird von einzelnen Knoten bereitgestellte Anwendungsfunktionalität als Dienst gekapselt und damit eine Flexibilität der Systemarchitektur erreicht.

Zum Aufruf der einzelnen Dienste werden im WSN jeweils proprietäre Kommunikationsprotokolle verwendet. Zur Integration des WSN in das Backend werden schwergewichtige, applikationsspezifische Gateways benötigt, die zwischen den proprietären WSN-Protokollen und den im Backend verwendeten Standards konvertieren. Dadurch geht die auf konzeptioneller Ebene durch die Verwendung des serviceorientierten Paradigmas gewonnene Flexibilität technologisch bei allen Ansätzen wieder verloren. Bei Delicato et al. und Blumenthal et al. wird die Flexibilität sogar noch weiter eingeschränkt, da diese Ansätze lediglich den Aufruf eines von einem Sensorknoten angebotenen Dienstes durch einen Konsumenten im Backend erlauben.

Zwar verwenden Delicato et al. sowohl im Backend als auch in komprimierter Form im WSN SOAP-Webservices, dabei wird SOAP jedoch stark eingeschränkt und dient nur als Container eines proprietären Nachrichtenformats, das sowohl im WSN als auch im Backend verwendet wird. Das bedeutet, dass auch hier trotz der Verwendung von Webservices nicht von einer Standardkonformität gesprochen werden kann. Das Gleiche gilt für alle weiteren hier beschriebenen Ansätze, die eine vollständig proprietäre Kommunikation verwenden.

Die Realisierung sowie die Komposition von Diensten erfolgen bei allen Ansätzen mit hardwarenahen Programmiersprachen und sind entsprechend komplex. Lediglich Kushwaha et al. bzw. Amundson et al. bieten eine Möglichkeit zur grafischen Realisierung einer Dienstnutzung. Dabei können einfache Regeln für die Verarbeitung von Daten aus Dienstaufrufen definiert werden. Diese Sprache eignet sich jedoch nicht für ein Geschäftsprozessmanagement, da sie keine rekursive Aggregation ermöglicht und nur über eine sehr eingeschränkte Ausdrucksstärke verfügt. Die Realisierung der eigentlichen Logik der Dienste erfolgt auch in diesem Ansatz durch eine hardwarenahe Programmierung.

Obwohl die Bezeichnung serviceorientierte Middleware für Sensornetze eine Eignung für den Einsatz zur Realisierung von Geschäftsprozessen suggeriert, erfüllen heutige Ansätze weder technologisch noch konzeptionell die notwendigen Voraussetzungen. Diese Ansätze stellen bei näherer Betrachtung gar keine serviceorientierte Architektur dar. Eine SOA setzt voraus, dass eine inhärente Interoperabilität existiert [53]. Das bedeutet, dass beliebige Dienste des Serviceinventars ohne Weiteres in beliebigen Servicekompositionen verwendet werden können.

Dieses wird durch die Notwendigkeit von anwendungsspezifischen Gateways sowie durch die Einschränkung der Nachrichtenaustauschmuster verhindert. So lassen z. B. Delicato et al. und Blumenthal et al. nur Dienstaufrufe vom Backend in Richtung WSN zu.

Auch erste Lösungen zur Realisierung von Webservices, die dem REST-Paradigma sowie der WS-*-Architektur folgen, sind nicht zur Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen geeignet. Entsprechende Ansätze werden in Abschnitt 4.2.1 diskutiert.

3.3.5 Bewertung der Eignung aktueller Sensornetzprogrammierabstraktionen für ein Geschäftsprozessmanagement

Die Bewertung der WSN-Programmierabstraktionen hat gezeigt, dass die heute verwendeten Ansätze zur Realisierung von Sensornetzapplikationen weder konzeptionell noch technologisch die Anforderungen eines Geschäftsprozessmanagements erfüllen. Die Entwicklung von Sensornetzanwendungen kann nur von Sensornetzexperten realisiert werden. Sie ist durch die Verwendung komplexer Programmiersprachen, einer monolithischen Anwendungsarchitektur sowie durch eine fehlende Verteilungstransparenz gekennzeichnet. Und obwohl lediglich datenzentrische Middleware-Ansätze eine Prozessausführung innerhalb des Sensornetzes von vornherein direkt verhindern, beschränken auch die anderen aufgeführten Abstraktionen WSNs auf ein einfaches *sense & send*. Die monolithische Anwendungsarchitektur, komplexe Programmierung und fehlende Verteilungstransparenz führen dazu, dass nur die nötigste Logik tatsächlich im WSN ausgeführt wird. Die Folge ist eine indirekte Beschränkung des WSNs auf *sense & send*. Darüber hinaus werden keinerlei Standards aus dem Geschäftsprozessmanagement im Umfeld von Enterprise-IT-Umgebungen eingesetzt.

Eine schnelle und flexible Entwicklung und Anpassung von WSN-Anwendungen an sich ändernde fachlich-konzeptionelle Geschäftsprozesse und Unternehmensstrategien ist also nicht möglich. Und selbst wenn davon auszugehen wäre, dass WSNs nicht ständig angepasst werden müssten, könnte mithilfe heutiger Abstraktionen keine Ermöglichung neuartiger Prozesse erreicht werden. Wie die Evaluation in Abschnitt 5.6.6 zeigt, stellen die Entwicklungs- und Integrationszeit sowie -kosten bei der aktuellen Anwendungsentwicklung selbst in einem statischen Szenario eine zu hohe Eintrittbarriere dar.

3.4 Ganzheitliches Konzept zum Geschäftsprozessmanagement im Kontext von Sensornetzen

Bei der informationstechnischen Umsetzung von Geschäftsprozessen stehen Unternehmen sowohl vor technischen als auch vor fachlichen Herausforderungen. Entsprechend muss ein erfolgreiches Geschäftsprozessmanagement beide Aspekte berücksichtigen. Im Rahmen dieser Arbeit wurde ein ganzheitliches Konzept zur informationstechnischen Modellierung, Ausführung, Überwachung und Optimierung von Geschäftsprozessen im Kontext von Sensornetzen entwickelt, das alle in Abschnitt 3.2 definierten Anforderungen erfüllt. Es ermöglicht die transparente informationstechnische Realisierung von Geschäftsprozessen im Sensornetz und im Backend sowie die transparente Interaktion aller Teilprozesse über Plattformgrenzen

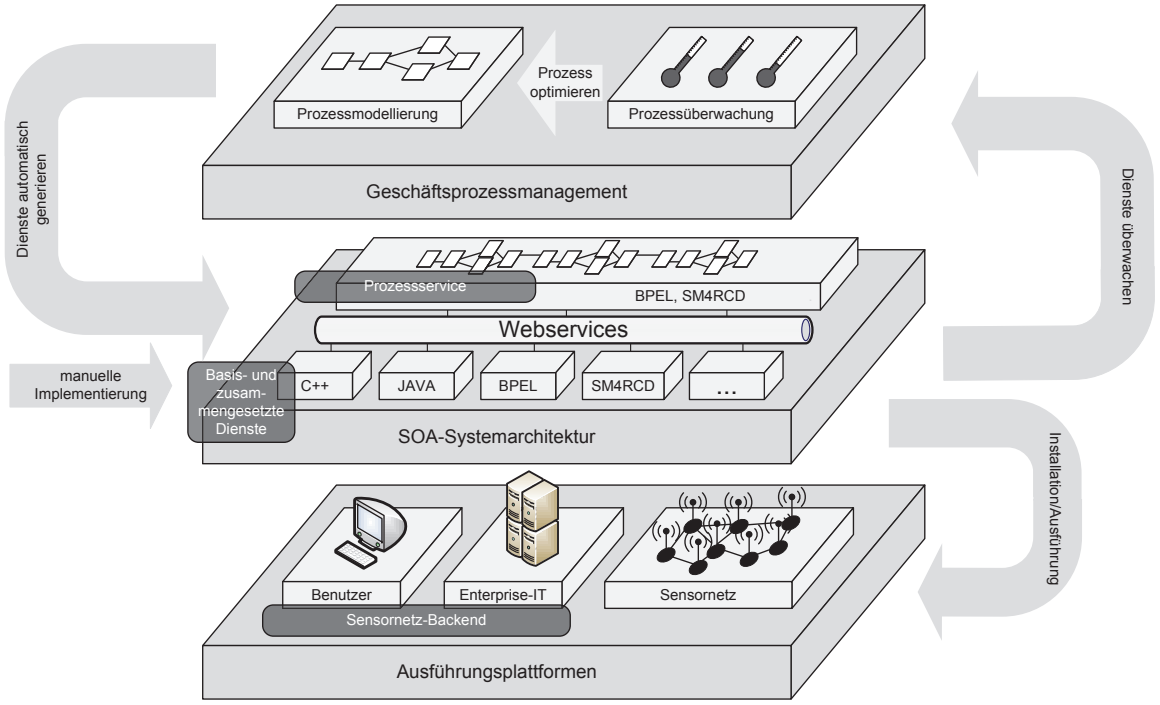


Abbildung 3.2: Ganzheitliches Konzept zur Realisierung eines dynamischen Webservice-basierten Geschäftsprozessmanagements in Sensornetzen und Enterprise-IT-Systemen

hinweg. Abbildung 3.2 stellt das Konzept in drei Ebenen grafisch dar. Die unterste Ebene beschreibt die Systemkomponenten, die der Ausführung von Prozess- bzw. Anwendungslogik durch maschinelle und menschliche Aufgabenträger dienen. Die mittlere Ebene beschreibt die Systemarchitektur, die dem Anwendungssystem zur Ausführung der Prozesse zugrunde liegt. Darauf aufbauend beschreibt die oberste Ebene die fachliche und technische Prozessrealisierung bzw. -modellierung, -überwachung und -optimierung.

3.4.1 Ausführungsplattformen

Ein Anwendungssystem zur informationstechnischen Umsetzung von Geschäftsprozessen im Kontext von Sensornetzen wird auf verschiedenen Plattformen ausgeführt. Da sind zum einen, wie in bisherigen Systemen zur maschinellen Ausführung von Workflows, die Enterprise-IT-Systeme. Sie dienen der Ausführung von Geschäftslogik und können der Ebene der Anwendungslogik in einem n-Ebenen-Modell zugeordnet werden. Typischerweise werden sie in Applikationsservern auf leistungsstarker Serverhardware ausgeführt. Darüber hinaus existieren in einem Anwendungssystem noch Komponenten, die der Interaktion mit menschlichen Aufgabenträgern dienen. Dieses können Standalone-Clients sein, die vollständig der Präsentationsebene zugeordnet werden. Gegenwärtig werden für diese Aufgabe jedoch verstärkt Webschnittstellen verwendet, die auf der Präsentationsebene einen Browser als schlanken Client und in der Applikationsebene eine Webapplikation zur dynamischen Generierung von Webseiten verwenden.

Enterprise-IT-Komponenten und Komponenten zur Benutzerinteraktion sind auch bei der Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen wichtige Bestandteile des Gesamtsystems. Sie bilden in dem hier vorgeschlagenen Konzept gemeinsam das Backend eines oder mehrerer Sensornetze. Sensornetze werden nun als neue Ausführungsplattform in dieses Gesamtsystem integriert. Anders als bei den in Abschnitt 3.3 betrachteten WSN-Programmierabstraktionen wird das Sensornetz jedoch nicht der Datenebene zugeordnet. Zur Realisierung eines Geschäftsprozessmanagements in Sensornetzen muss auf den einzelnen Knoten beliebige Prozesslogik ausgeführt werden und eine transparente Interaktion mit beliebigen Systemkomponenten möglich sein. Sensornetze werden dazu in diesem Konzept wie jede Logikkomponente der Enterprise-IT der Applikationsebene zugeordnet und agieren ebenfalls als maschineller Aufgabenträger in einem Prozess.

3.4.2 Verwendung serviceorientierter Architekturen als Systemarchitektur

Wie in Abschnitt 2.1 beschrieben, stellen SOAs die momentan geeignetste Systemarchitektur zur Umsetzung von Geschäftsprozessen dar. Neben vielen weiteren Vorteilen bieten sie vor allem die notwendige Flexibilität. Im Rahmen dieses Konzepts wird jegliche Anwendungs- bzw. Prozesslogik sowie die Präsentationskomponenten analog zum SOA-Konzept als in sich abgeschlossene Dienste über klar definierte Schnittstellen zur Verfügung gestellt und ein Serviceinventar aufgebaut. Durch eine Aggregation von Diensten aus dem Serviceinventar lassen sich dann schnell und flexibel neue zusammengesetzte Dienste oder Prozessservices realisieren.

Der entscheidende Erfolgsfaktor ist, dass nicht wie bei o. g. WSN-Programmierabstraktionen das Sensornetz als ein monolithischer Block oder als Pseudo-SOA in das Backend integriert wird. D. h., es existiert keine SOA im Backend, in die die Sensornetze (evtl. als zweite, parallele in sich abgeschlossene Pseudo-SOA) über applikationsspezifische Gateways integriert werden. Im Rahmen dieser Arbeit wird das SOA-Konzept ganzheitlich im Gesamtsystem angewendet. Das bedeutet, dass nicht nur jegliche Funktionalität als Dienste gekapselt wird. Es ist darüber hinaus vollständig transparent, ob diese Dienste im Backend oder auf Sensorknoten ausgeführt werden. Dabei werden keinerlei Unterschiede zwischen den drei Dienstkategorien Basisservice, zusammengesetzter Service und Prozessservice gemacht. Alle drei Kategorien können sowohl auf Sensorknoten als auch auf Enterprise-IT-Servern ausgeführt werden. Damit sind Sensornetzdienste absolut gleichwertig und gleichberechtigt zu Backend-Diensten. Die Ausführungsplattform eines Dienstes ist sowohl für den Dienstnutzer als auch für den -erbringer vollständig transparent.

Durch die ganzheitliche Anwendung des SOA-Konzepts im Gesamtsystem können neben weiteren in Abschnitt 2.1 beschriebenen Vorteilen vor allem die für ein Geschäftsprozessmanagement erforderliche Flexibilität der Anwendungsarchitektur für das Gesamtsystem und eine Wiederverwendbarkeit von Diensten sowie eine bessere organisationsinterne und -übergreifende Integrationsfähigkeit erreicht werden.

3.4.3 Verwendung von Webservices als Middleware-Technologie

Serviceorientierte Architekturen stellen lediglich ein Architekturkonzept dar. Ihre Umsetzung kann über verschiedene Middleware-Technologien erfolgen. Heutzutage wird in Enterprise-IT-Systemen zu diesem Zweck vor allem, wie in Abschnitt 2.3 beschrieben, die Webservice-Technologie eingesetzt. Der hohe Verbreitungsgrad der Webservice-Technologie sowie der darauf aufbauenden Geschäftsprozessrealisierungsstandards in der Unternehmens-IT und die große Anzahl an Werkzeugen für die Unterstützung des Entwicklungsprozesses legen bereits die Verwendung von Webservices fest. Als plattform- und programmiersprachenunabhängige Middleware ermöglichen sie eine organisationsinterne und -übergreifende Bereitstellung und Nutzung von über ein Netzwerk verteilten Diensten. Dadurch kann die auf konzeptioneller Ebene durch das SOA-Konzept erreichte Flexibilität und inhärente Interoperabilität auch technologisch erreicht werden. Darüber hinaus eignen sich Webservices besonders gut für heterogene Umgebungen und lassen sich durch ihren modularen Aufbau optimal an das Einsatzszenario anpassen.

Um die durch die ganzheitliche Anwendung des SOA-Konzepts erreichte vollständige Transparenz bzgl. der Ausführung sowie Nutzung von Diensten auch technisch umzusetzen, sieht dieses Konzept die Verwendung einer ganzheitlichen und transparenten Webservice-Kommunikation im Backend sowie im WSN vor. Damit wird die in Abschnitt 3.2 geforderte vollständige Verteilungstransparenz erreicht.

Wie in den Kapiteln 4 und 6 im Detail dargelegt wird, verhindern die extremen Ressourcenbeschränkungen von WSNs den Einsatz der gegenwärtigen Webservice-Lösungen im Kontext von WSNs. Um Webservices in Sensornetzen einsetzen zu können, muss der Webservice-Technologiestapel an verschiedenen Stellen erweitert werden. Abbildung 3.3 fasst die im Rahmen dieser Arbeit entwickelten Lösungen zusammen. Sie sind konform zu den vom W3C definierten Erweiterungsmöglichkeiten der Webservice-Architektur (s. Abschnitt 2.3.1 und [238]). In Abbildung 3.3 beschreiben die schwarz hinterlegten Elemente eine Erweiterung des Technologiestapels mit neuen, im Rahmen dieser Arbeit entwickelten Lösungen. Grau schraffiert sind aktuelle Webservice-Standards, für deren Anwendung im Umfeld von Sensornetzen im Rahmen dieser Arbeit spezielle Lösungen erarbeitet werden mussten, die jedoch keiner Änderung an den jeweiligen Standards bedurften.

Der wichtigste Schritt zur Realisierung einer ganzheitlichen Webservice-Kommunikation ist die Entwicklung eines leichtgewichtigen Webservice-Transport-Bindings, das die in Abschnitt 3.2 definierten Anforderungen erfüllt. Im Rahmen dieser Arbeit wurde ein solches Binding entwickelt (s. Kapitel 4). Es besteht zum einen aus einer kompakten Serialisierung/Komprimierung von SOAP-Nachrichten, die eine effiziente Verarbeitung auf Sensorknoten erlaubt. Diese wird im Folgenden als *SOAP Message Compression* (SMC) bezeichnet. Zum anderen wurde zum Nachrichtenaustausch im WSN und Enterprise-IT-Bereich das leichtgewichtige Transportprotokoll LTP (*Lean Transport Protocol*) entwickelt. Beim Einsatz von LTP+SMC ist es für den Anwendungsentwickler vollständig transparent, ob sich die Webservice-Nutzer oder -Anbieter im WSN oder Backend befinden. Mit LTP+SMC werden mit einem transparenten, organisationsinternen und -übergreifenden Nachrichtenaustausch im Sensornetz und im Backend sowie zwischen beiden Teilnetzen und mit der Transparenz beim Aufruf verteilter Funktionalität beide Kommunikationsanforderungen ganzheitlich erfüllt. Da LTP+SMC ein

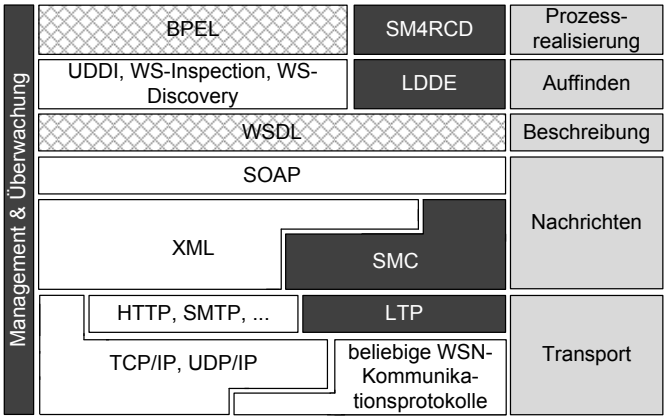


Abbildung 3.3: Erweiterter Webservice-Technologiestapel für ein Geschäftsprozessmanagement im Kontext von Sensornetzen

SOAP-Transport-Binding ist, können entsprechende Dienste mit Standard-WSDL beschrieben werden.

LTP+SMC ist das im Konzept dieser Arbeit vorgeschlagene Transport-Binding für jegliche Kommunikation zwischen Endpunkten im Gesamtsystem. Bezogen auf die Kommunikationsebene lassen sich mit LTP+SMC eine Interaktion zwischen den Rollen des Servicenutzers und -anbieters analog zum SOA-Rollenmodell (s. Abschnitt 2.1.4) umsetzen. Zwar können für dieses Binding die Webservice-Schnittstellen beschrieben werden. Allerdings erfolgt die Bindung zwischen Webservice-Anbieter und -Nutzer statisch zur Designzeit. Sensornetze im Allgemeinen, aber auch im Speziellen der Anwendungsfall einer Gefahrgutüberwachung sind durch eine hohe Dynamik gekennzeichnet. Verschiedene Sensorknoten bauen ad hoc Kommunikationsbeziehungen oder gar ganze Netze auf bzw. verlassen diese wieder. So wechseln z. B. bei der Gefahrgutüberwachung die Sensorknoten häufig ihre Umgebung und müssen in die lokalen WSNs und Backend-Systeme dynamisch integriert werden. In diesen Szenarien, insbesondere wenn sowohl die Sensorknoten als auch die Backend-Dienste unterschiedliche organisationale Zugehörigkeiten aufweisen, reicht ein statisches Binden nicht mehr aus, um die Anforderungen der Systemdynamik zu erfüllen. Gerade im Umfeld von Sensornetzen ist deshalb eine Dienstvermittlung wichtig, die dynamisch zur Laufzeit sowohl Dienste auffindet als auch deren Schnittstellenbeschreibungen zur Verfügung stellt. Mit LDDE (*Lean Description, Discovery and Exchange*) wurde im Rahmen dieser Arbeit aufbauend auf LTP+SMC und WSDL ein Protokoll für die Realisierung genau dieser Aufgabe des SOA-Rollenmodells entwickelt (s. Abschnitt 6.1). Es realisiert ein dynamisches Auffinden von Webservices im Gesamtsystem sowie selbstbeschreibende Dienste. Zwar wird standardkonformes WSDL zur Webservice-Beschreibung eingesetzt. Da diese Dokumente jedoch aufgrund der Ressourcenbeschränkungen weder auf Sensorknoten gespeichert noch im WSN übertragen werden können, realisiert die Selbstbeschreibung neben einem effizienten Austauschprotokoll auch eine optimierte WSDL-Kompression.

Das Konzept dieser Arbeit zum ganzheitlichen Geschäftsprozessmanagement sieht LTP+SMC zwar als primäres Webservice-Transport-Binding vor. LTP+SMC ist zur WS-*-Architektur des W3C und insbesondere zu SOAP konform. Aber wie bei jedem anderen Transport-Binding setzt die Kommunikation zwischen Webservice-Anbieter und -Nutzer voraus, dass beide

Kommunikationspartner dieses Binding unterstützen. Für die Einführung von LTP+SMC in die Unternehmenspraxis ergeben sich daraus zumindest temporär folgende Probleme. So existieren in Unternehmen bereits umfangreiche Serviceinventare, die weiterhin genutzt und auch Sensornetzen zur Verfügung gestellt werden sollen. Verwenden diese Webservices ein Framework, das auch LTP+SMC unterstützt, dann beschränkt sich der Migrationsaufwand auf die Administration des Frameworks. Andernfalls würde die Migration einen zusätzlichen Investitionsaufwand für die Unternehmen darstellen, der eine Eintrittbarriere bedeuten könnte. Auch ist es eher unwahrscheinlich, dass Serviceanbieter, die eine große Marktmacht aufweisen, ihre Dienste zusätzlich über LTP+SMC anbieten, wenn die Nutzer im Sensornetz nicht deren primäre Zielgruppe darstellen oder der Verbreitungsgrad von LTP+SMC noch keine kritische Grenze überschritten hat. Um auch in diesen Fällen eine direkte Webservice-Kommunikation zwischen beliebigen Endpunkten im Gesamtsystem zu ermöglichen, wurde im Rahmen dieser Arbeit eine automatische, konfigurationsfreie und für die Kommunikationspartner transparente Konvertierung zwischen LTP+SMC und anderen Webservice-Transport-Bindings realisiert (s. Abschnitt 6.2).

Mit den hier präsentierten Webservice-Lösungen zum Transport zur Verarbeitung und Codierung von Nachrichten sowie zur Dienstbeschreibung und -suche sowie zur Konvertierung von Transport-Bindings stehen optimale Lösungen zur Verfügung, um das o. g. SOA-Konzept ganzheitlich im WSN und Backend umzusetzen und als Kommunikationsbasis für ein Webservice-basiertes Geschäftsprozessmanagement zu dienen. Diese werden noch um verschiedene Werkzeuge zur Durchführung verschiedener Management- und Überwachungsaufgaben ergänzt.

3.4.4 Verwendung eines ganzheitlichen Workflowmanagementsystems

Wie in den Abschnitten 2.2.4 und 3.2 beschrieben, ist die Flexibilität der Anwendungs- bzw. Systemarchitektur eine notwendige, aber keine hinreichende Bedingung für ein erfolgreiches Geschäftsprozessmanagement. Zwar wird, wie in Abschnitt 3.4.2 beschrieben, durch die ganzheitliche Verwendung serviceorientierter Architekturen durch die Wiederverwendbarkeit und Kompositionsfähigkeit lose gekoppelter Dienste genau diese Flexibilität für das Gesamtsystem erreicht. Darüber hinaus wird auch durch die in Abschnitt 3.4.3 beschriebenen Webservice-Lösungen diese konzeptionelle Flexibilität auf technischer Ebene bewahrt und eine ganzheitliche Verteilungstransparenz erreicht. Ohne ein Workflowmanagementsystem müssen jedoch Programmiersprachen der 3. Generation wie C++ oder Java durch IT- und Sensornetzexperten zur Prozessrealisierung verwendet werden.

Zwar stellt die Verwendung von Programmiersprachen der 3. Generation zur Realisierung von Basis- und einfachen zusammengesetzten Diensten häufig eine gute Wahl dar und wird auch in diesem Konzept für diese Dienstkategorien empfohlen. Aus den in Abschnitt 2.2.4 diskutierten Gründen wird jedoch für die Realisierung von komplexeren zusammengesetzten Diensten und Prozessservices in diesem Konzept ausschließlich die Verwendung eines Workflowmanagementsystems zur domänenspezifischen Prozessrealisierung empfohlen. Vom Autor dieser Arbeit wurde mit IWFMS (*Integral Workflow Management System*) ein Workflowmanagementsystem für ein ganzheitliches Geschäftsprozessmanagement entwickelt. Es realisiert alle Schnittstellen

des WfMC-Referenzmodells zur Umsetzung eines Workflowmanagementsystems und ermöglicht die ganzheitliche und transparente Modellierung, Ausführung (inkl. Simulation) und Überwachung sowie Optimierung von Geschäftsprozessen durch Fachpersonal.

Zur Prozessmodellierung bzw. -definition werden in IWFMS mit BPEL und SM4RCD (*State Machine for Resource Constrained Devices*) zwei Realisierungssprachen vorgeschlagen, die eine domänenspezifische Definition von Geschäftsprozessen durch eine rekursive Aggregation von Webservices ermöglichen. Während BPEL den heute bedeutendsten Standard zur Realisierung von Geschäftsprozessen in einer auf Webservices basierenden SOA darstellt, ist SM4RCD eine domänenspezifische Sprache, die im Rahmen dieser Arbeit entwickelt wurde. Prozessmodelle können in beiden Sprachen direkt von Fachpersonal durch die Verwendung entsprechender grafischer Editoren erstellt oder aus Modellen fachlich-konzeptioneller Geschäftsprozessmodellierungssprachen (z. B. EPK oder BPMN) automatisch generiert werden. Bei der Modellierung der Prozessmodelle ist die spätere Ausführungsplattform vollständig transparent. In diesem Konzept können entsprechende Prozessmodelle sowohl auf Sensorknoten als auch auf Enterprise-IT-Servern ausgeführt werden.

Bei der Ausführung von Geschäftsprozessen müssen zwei Herausforderungen bewältigt werden. Zum einen ist das die Kommunikation zwischen dem Prozess sowie weiteren Prozessen bzw. Diensten und zum anderen die Ausführung der eigentlichen Prozesslogik. Wie in Abbildung 3.3 zu sehen, bauen alle Prozessrealisierungssprachen auf dem Webservice-Standard WSDL auf. Da die in diesem Konzept verwendeten Webservice-Lösungen zu WSDL konform sind, können diese auch als ganzheitliche Kommunikationsbasis für eine Prozessrealisierung mit BPEL und SM4RCD dienen.

Die Ausführung der Prozessmodelle erfolgt für den Entwickler in IWFMS plattformtransparent. Intern muss jedoch bei der Ausführung der eigentlichen Prozesslogik zwischen der Ausführung im Backend und auf Sensorknoten unterschieden werden. Während die Ressourcenstärke von Enterprise-IT-Systemen die Ausführung von ressourcenverbrauchenden Laufzeitumgebungen für die Interpretation der jeweiligen Prozessmodelle problemlos ermöglicht, sind diese Lösungen nicht auf Sensorknoten übertragbar. Aus diesem Grund wurden im Rahmen dieser Arbeit spezielle Ausführungskonzepte und Laufzeitumgebungen entwickelt, um BPEL- und SM4RCD-Prozesse auch auf Sensorknoten ausführen zu können. Diese Unterscheidung betrifft jedoch ausschließlich interne Aspekte der Prozessausführung von IWFMS, die dem Entwickler bzw. Modellierer verborgen bleiben.

Um das Geschäftsprozessmanagement im Kontext von Sensornetzen sowohl auf technischer als auch auf fachlicher Ebene abzurunden, realisiert IWFMS neben der Prozessdefinition und -ausführung auch eine Prozessüberwachung und -optimierung. IWFMS stellt eine Reihe an Schnittstellen zur Verfügung, um diese Aufgaben durch verschiedene Werkzeuge zu unterstützen. Dazu wurden im Rahmen dieser Arbeit eine Reihe von Werkzeugen entwickelt, die eine grafische Überwachung während der Entwicklung sowie während des Produktionsbetriebs erlauben. Zusätzlich ermöglichen die Schnittstellen von IWFMS auch die Anbindung existierender Überwachungs- und Optimierungswerkzeuge aus dem fachlich-konzeptionellen Prozessmanagement.

Die ganzheitliche Realisierung von Geschäftsprozessen wird in Kapitel 5 im Detail beschrieben.

3.5 Zusammenfassung

In diesem Kapitel wurde ein Konzept zur Realisierung eines ganzheitlichen Geschäftsprozessmanagements in Sensornetzen und Enterprise-IT-Systemen präsentiert. Es schlägt die Verwendung von Integral Workflow Management System vor, das durch Fachpersonal verwendet werden kann und die Modellierung, Überwachung und Optimierung sowie die direkte Ausführung von Prozessmodellen realisiert. Zur informationstechnischen Umsetzung der Prozesse baut das Workflowmanagementsystem auf einer SOA als Systemarchitektur mit Webservices als Realisierungstechnologie auf. Durch den Einsatz des SOAP-Bindings LTP+SMC realisiert die Webservice-Kommunikation eine vollständige Verteilungstransparenz im Gesamtsystem bestehend aus Sensornetzen, Enterprise-IT-Systemen und Diensten zur Interaktion mit menschlichen Aufgabenträgern.

Durch die Umsetzung dieses Konzepts mit den im Rahmen dieser Arbeit entwickelten Umsetzungslösungen wird analog zu Abbildung 1.1 auf Seite 1 eine optimale Anpassung informationstechnischer Prozessrealisierungen an fachlich-konzeptionelle Geschäftsprozesse und Unternehmensstrategien sowohl im Backend als auch im WSN realisiert. Erst dadurch kann das Potenzial von WSNs zur Ermöglichung neuartiger Geschäftsprozesse ausgeschöpft werden, da ansonsten die fachlichen Vorteile des Einsatzes von WSNs in der Regel durch den erhöhten Entwicklungsaufwand wieder aufgehoben werden und keine Prozessagilität erreicht werden kann.

Der Einsatz von offenen, im Umfeld eines Geschäftsprozessmanagements in Enterprise-IT-Systemen etablierten Standards gewährleistet eine nahtlose Integration in bestehende Anwendungssysteme und den Schutz der dortigen Investitionen. Gleichzeitig wird eine organisationsübergreifende Prozessintegration sowie die Ausführung von Prozessen auf angemieteten Plattformen ermöglicht.

4 Ganzheitliche Webservice-Kommunikation im Sensornetz und Backend

Für die Umsetzung des in Abschnitt 3.4 beschriebenen Konzepts zur Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen wird ein Webservice-Transport-Binding benötigt, das eine ganzheitliche und transparente Kommunikation zwischen beliebigen Endpunkten in Sensornetzen und Backend-Systemen ermöglicht. Gleichzeitig muss es effizient genug sein, um die Ressourcenbeschränkungen von WSNs einzuhalten. In diesem Kapitel werden in den Abschnitten 4.1 und 4.2 die Herausforderungen einer Webservice-Kommunikation in Sensornetzen sowie verwandte Arbeiten zu diesem Thema diskutiert, bevor in den darauf folgenden Abschnitten das vom Autor dieser Arbeit entwickelte Webservice-Transport-Binding LTP+SMC im Detail beschrieben und evaluiert wird. LTP+SMC verwendet das vom Autor dieser Arbeit entwickelte *Lean Transport Protocol* (LTP) zum ganzheitlichen Austausch von speziell komprimierten SOAP-Nachrichten (*SOAP Message Compression*, SMC). Es erfüllt die in Abschnitt 3.2 definierten Anforderungen einer vollständigen Verteilungstransparenz sowie einer Konformität zur WS-*-Architektur und weist gleichzeitig einen extrem geringen Ressourcenverbrauch auf.

Die in diesem Kapitel präsentierten Forschungsergebnisse wurden bereits vorab in [65, 69, 70] und insbesondere in [72] veröffentlicht[1].

4.1 Herausforderungen bei der Realisierung von Webservices in drahtlosen Sensornetzen

Zur Realisierung einer Webservice-Kommunikation werden Komponenten bzw. Standards von drei Ebenen des Webservice-Technologiestapels benötigt. Das sind auf der Nachrichtenebene ein Format zur Codierung von Webservice-Nachrichten und auf der Transportebene ein Protokoll zum Austausch dieser Nachrichten zwischen Webservice-Anbieter und -Nutzer. Zusätzlich wird eine Sprache benötigt, die die Schnittstelle eines Webservices formal beschreibt.

4.1.1 Nachrichten

SOAP stellt auf der Nachrichtenebene des Webservice-Technologiestapels gegenwärtig den wichtigsten Standard dar. Wie in Abschnitt 2.3.2 beschrieben, werden SOAP-Nachrichten derzeit vor allem als XML-Dokumente serialisiert. Wie der Quelltext 2.1 auf Seite 25 zeigt, entsteht durch eine XML-Serialisierung ein erheblicher Overhead. Für diese einfache Anfrage,

[1] In älteren Publikationen wurde sowohl für das Transport-Binding LTP+SMC als auch für das Transportprotokoll LTP die Bezeichnung WSNTB (*Wireless Sensor Network Transport Binding*) verwendet.

die lediglich zwei Fließkommazahlen und einen Zeitstempel enthält, beträgt die Größe des XML-Dokuments bereits 638 Byte bzw. 409 Byte (ohne Verwendung von WS-Addressing). Mit zunehmender Datenmenge und Komplexität der Datenstrukturen erreichen die Nachrichten sehr schnell Größen von einigen Kilobyte.

Die direkte Verwendung von XML erzeugt einen zu großen Overhead und ist keine Option für Sensornetze. Bei den geringen zur Verfügung stehenden Datenraten, die in der Regel durch ein Duty Cycling weiter reduziert werden, werden schon bei wenigen Kommunikationspartnern (je nach Frequenz des Nachrichtenaustauschs) sehr schnell die vorhandenen Ressourcen ausgeschöpft. Dabei ist der zusätzlich entstehende Overhead der verwendeten Transportprotokolle noch nicht mit berücksichtigt. Darüber hinaus verhindern auch die geringen Speicherkapazitäten von Sensorknoten den Einsatz von XML. Die Speicherung der XML-Dokumente auf Sensorknoten, die selbst oft nur über wenige Kilobyte Arbeitsspeicher verfügen, stellt bereits einen zu hohen Ressourcenverbrauch dar, sodass eine Verarbeitung unmöglich ist. Zusätzlich überschreitet auch die Codegröße entsprechender Parser die Kapazitäten von Knoten.

Um trotzdem einen SOAP-Nachrichtenaustausch zu realisieren, müssen spezielle Kompressionsverfahren oder alternative, effiziente Codierungen verwendet werden. Beide Varianten sind konform zum SOAP-Standard, der XML lediglich als eine mögliche Serialisierung vorsieht.

4.1.2 Transport von Nachrichten

Neben dem eigentlichen Nachrichtenformat legt der SOAP-Standard, wie in Abschnitt 2.3.2 beschrieben, fest, wie SOAP-Nachrichten ausgetauscht werden können. Gegenwärtig stellt HTTP das meistverwendete Webservice-Transportprotokoll dar. Der SOAP-Standard (*SOAP Protocol Binding Framework*) spezifiziert die Anforderungen an ein Transportprotokoll für den Austausch von SOAP-Nachrichten. Dadurch ist es möglich auch weitere Transportprotokolle wie TCP, UDP, FTP, SMTP, u. a. zum Nachrichtenaustausch zu verwenden und somit die Kommunikation optimal an das Anwendungsszenario anzupassen.

Das Problem ist, dass keines dieser Protokolle für den Einsatz im Sensornetz geeignet ist. Werner et al. [222] haben den Kommunikations-Overhead dieser Protokolle für einen einfachen Webservice-Aufruf ohne Aufruf- und Rückgabeparameter analysiert. Dabei erreichen HTTP mit 560 Byte, FTP mit 576 Byte und SMTP mit 2.535 Byte einen Overhead, der die Kapazitäten von Sensornetzen bei Weitem überschreitet. Dieser bezieht sich jedoch nur auf die Anwendungsschicht. Auf den unterliegenden Schichten wird weiterer Overhead generiert, sodass der Gesamt-Overhead bedeutend höher ausfällt (HTTP: 1.096 Byte, SMTP: 4.487 Byte, FTP: 3.177 Byte). Auch wenn eine Webservice-Kommunikation direkt auf Transportschichtprotokollen wie TCP oder UDP aufsetzt, ist der Overhead zu groß. Bei TCP wird insgesamt ein Overhead von 858 Byte erreicht. Bei der Verwendung von UDP scheint der Overhead mit insgesamt 56 Byte auf den ersten Blick auch für Sensornetze akzeptabel zu sein. Allerdings setzt der Einsatz von UDP, wie auch der von TCP, die Verwendung von WS-Addressing voraus, da UDP und TCP nur Rechner und darauf laufende Applikationen, aber keine Webservice-Endpunkte adressieren können. WS-Addressing als XML-Grammatik hat jedoch einen erheblichen weiteren Overhead zur Folge. Damit scheidet auch UDP als Transportprotokoll für Webservices im WSN aus.

Beim Entwurf eines Transportprotokolls, das einen effizienten und transparenten Nachrichtenaustausch im WSN und Backend ermöglicht, stellt sich die Frage, auf welchen Kommunikationsprotokollen ein solches Protokoll aufsetzen soll. Alle eben bewerteten Protokolle setzen das Internetprotokoll IP voraus. Aus dem Blickwinkel der Enterprise-IT betrachtet, stellt IP die ideale Basis dar, da es Sensorknoten direkt in das Internet integrieren und einen transparenten Nachrichtenaustausch zwischen allen Geräten erlauben würde. Die direkte Verwendung von IP zum Austausch von Webservice-Nachrichten ist jedoch nicht möglich, da mit IP nur Geräte aber keine Webservice-Endpunkte adressiert werden können.

Bei der Anwendbarkeit von IP im Sensornetz existieren gegenwärtig zwei gegensätzliche Meinungen. Ein Teil der Forscher ist davon überzeugt, dass weder IPv4 noch IPv6 in WSNs sinnvoll einsetzbar sind [46]. Ein anderer Teil arbeitet jedoch an optimierten IP-Implementierungen, die die Restriktionen von Sensornetzen bzgl. des beschränkten Speichers und der geringen CPU-Leistung erfüllen [82]. Mit μIP präsentieren Dunkels et al. [44] eine RFC-kompatible IPv4-Implementierung für 8 Bit Mikroprozessoren. Analog zu μIP präsentieren Durvy et al. [46] μIPv6 als IPv6-kompatiblen Protokollstapel für ressourcenbeschränkte Geräte. Der wohl gegenwärtig erfolgversprechendste Ansatz IPv6 in WSNs zu etablieren ist 6LoWPAN (*IPv6 over Low-Power Wireless Personal Area Networks*). 6LoWPAN ist ein IETF-Standard [106], der beschreibt, wie IPv6-Datagramme über IEEE-802.15.4-kompatible Transceiver ausgetauscht werden können. Zur Reduktion des Kommunikations-Overheads spezifiziert der Standard zusätzlich Möglichkeiten zur Kompression von Header-Informationen.

Häufig wird trotzdem auf den Einsatz von 6LoWPAN verzichtet. Das geschieht insbesondere aus drei Gründen. Als Erstes setzt 6LoWPAN einen IEEE-802.15.4-kompatiblen Transceiver voraus. Dieser wird zwar von den meisten aktuellen, aber eben nicht von allen Sensornetzplattformen zur Verfügung gestellt. Als zweiter Grund ist die flexible Anpassung der Kommunikation in WSNs an die Applikationsbesonderheiten zu nennen, die durch den Einsatz von 6LoWPAN eingeschränkt wird. Und als letzten Grund ist die enorme Codegröße von 6LoWPAN zu nennen. Für die iSense-Plattform beträgt diese 42,1 Kilobyte.

Im Rahmen des in Abschnitt 3.4 beschriebenen Konzepts für das ganzheitliche Geschäftsprozessmanagement werden nicht nur einzelne Sensornetze und einzelne Backends integriert. Es entsteht ein Gesamtsystem aus vielen WSNs und vielen Backends. In einem solchen Szenario muss davon ausgegangen werden, dass sowohl Sensornetze in die Anwendungsarchitektur integriert werden müssen, die eine der verschiedenen IP-basierten Lösungen verwenden. Gleichzeitig müssen aber auch WSNs integriert werden, die darauf verzichten und intern viele verschiedene applikationsspezifische Kommunikationsprotokolle verwenden. Ein Webservice-Transportprotokoll muss mit beiden Möglichkeiten umgehen können.

4.1.3 Schnittstellenbeschreibung

Neben dem Nachrichtenformat und dem Transport dieser Nachrichten wird zur Realisierung einer Webservice-Kommunikation noch die Möglichkeit der formalen Beschreibung von Schnittstellen vorausgesetzt. Gegenwärtig wird zu diesem Zweck ausschließlich die in Abschnitt 2.3.2 eingeführte Sprache WSDL verwendet. Webservice-Anbieter und -Nutzer müssen sowohl den abstrakten als auch den konkreten Teil einer WSDL-Beschreibung kennen, um kommunizieren zu können. Im einfachsten und zugleich typischen Fall reicht die

Kenntnis der Schnittstellenbeschreibung zur Implementierungszeit aus. In diesem statischen Szenario werden die WSDL-Beschreibungen lediglich im Implementierungsprozess auf den Entwicklungscomputern verwendet. Aus den Beschreibungen wird dann auf diesen Rechnern Programmcode automatisch erzeugt, der die eigentliche Webservice-Kommunikation zur Laufzeit auf der jeweiligen Ausführungsplattform realisiert. Es muss nur dieser Middleware-Programmcode, aber nicht das WSDL-Dokument selbst auf den Knoten gespeichert werden.

Da der abstrakte Teil ohnehin keinen Bezug auf ein konkretes Transport-Binding nimmt, bedarf dieser Teil keiner Anpassung bei der Verwendung eines neuen Transportprotokolls bzw. Transport-Bindings. Auch der konkrete Teil von WSDL kann unverändert bleiben. WSDL ist so flexibel gestaltet, dass es ohne Veränderung des Standards mit beliebigen neuen Bindings umgehen kann. Damit kann WSDL ohne Anpassungen im o. g. statischen Szenario verwendet werden.

4.2 Verwandte Arbeiten

Für das im Rahmen dieser Arbeit entwickelte Transport-Binding zur ganzheitlichen Webservice-Kommunikation in Sensornetzen und Backends sind zwei Kategorien von verwandten Arbeiten relevant. In Abschnitt 4.2.1 werden Arbeiten präsentiert, die sich ebenfalls mit der Realisierung von Webservices in Sensornetzen beschäftigen. Es wird gezeigt, warum sich diese Arbeiten nicht zur Umsetzung des in Abschnitt 3.4 präsentierten Konzepts zur Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen eignen. In Abschnitt 4.2.2 werden verschiedene Arbeiten zur effizienten Serialisierung und Komprimierung von SOAP-Nachrichten vorgestellt, auf denen diese Arbeit aufbaut.

4.2.1 Webservice-Kommunikation

In Unternehmensanwendungen existieren Sensornetze nicht ohne Backend. Und obwohl Webservices die gegenwärtig vielversprechendste Middleware-Technologie darstellen, existieren nur sehr wenige Forschungsarbeiten, die sich mit der Realisierung von Webservices in WSNs oder mit einer Webservice-basierten Integration von Sensornetzen in die Enterprise-IT beschäftigen. Die Forschungsansätze zum Thema Webservices in Sensornetzen lassen sich anhand zweier Kriterien klassifizieren. Das erste klassenbildende Kriterium ist die Eigenschaft, ob die Ansätze sich eher an der WS-*-Webservice-Architektur oder am REST-Prinzip (s. Abschnitt 2.3.1) orientieren. Als zweites Kriterium wird die Notwendigkeit einer Nachrichten- oder Protokollkonvertierung mittels Gateway betrachtet. Abbildung 4.1 stellt die Zuordnung der im Folgenden vorgestellten Ansätze zur jeweiligen Kategorie dar.

Gateway-basierte Ansätze verwenden im Backend Kommunikationslösungen, die auf Standards basieren. Im Gegensatz dazu werden innerhalb der Sensornetze proprietäre Kommunikationstechnologien eingesetzt. Zur Integration von Sensornetzen und Backend-Systemen werden in diesen Ansätzen Gateways eingesetzt, die zwischen den verschiedenen Kommunikationslösungen konvertieren.

Amundson et al. [6] bzw. Kushwaha et al. [107] sowie Souza et al. [197] präsentieren Gateway-basierte Ansätze, die im Backend WS-*-Technologien oder zumindest konzeptionell

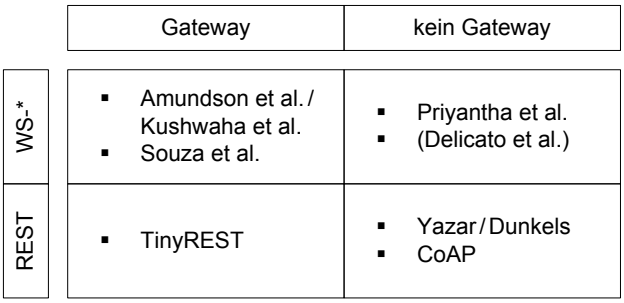


Abbildung 4.1: Kategorisierung verwandter Arbeiten zum Forschungsthema Webservices im Sensornetz

eng verwandte Lösungen verwenden. Die Arbeiten von Amundson et al. bzw. Kushwaha et al. wurden bereits in Abschnitt 3.3.4 z. T. erläutert. Sie ermöglichen den Aufruf von im Backend ausgeführten Webservices durch Sensorknoten sowie umgekehrt den Aufruf von Diensten, die auf Sensorknoten ausgeführt werden, durch Backend-Komponenten mit Webservices. Dabei wird die Webservice-Kommunikation jedoch nur im Backend verwendet. Innerhalb des Sensornetzes kommen proprietäre Lösungen zum Einsatz, sodass ein Gateway benötigt wird, um zwischen beiden Kommunikationslösungen zu konvertieren.

Einen ähnlichen Ansatz haben Souza et al. publiziert. Sie präsentieren eine auf WS-*-Webservices basierende Middleware zur Einbindung von Sensornetzen in Enterprise-IT-Systeme. Die zentrale Komponente dieses Ansatzes bildet ein DPWS-kompatibler (*Devices Profile for Web Services*, [42]) Nachrichtenbus, an den WSNs über ein Gateway angeschlossen werden. Auch in diesem Ansatz wird innerhalb des Sensornetzes keine Webservice-Kommunikation realisiert.

Anders als Amundson et al. bzw. Kushwaha et al. sowie Souza et al., die sich an der WS-*-Architektur orientieren und entfernte Methodenaufrufe über ein Gateway ermöglichen, verwendet *TinyREST* [119] das REST-Prinzip. Aber auch in diesem Ansatz wird innerhalb des WSNs ein proprietäres Kommunikationsprotokoll verwendet und der Zugriff auf Ressourcen nur über ein Gateway realisiert.

Anders als bei TinyREST präsentieren Yazar/Dunkels [263] einen REST-Ansatz, der kein Konvertierungs-Gateway benötigt. Die Autoren verwenden sowohl im Backend als auch im WSN HTTP zum Zugriff auf Ressourcen. Der Vorteil dieses Ansatzes ist die durchgehende Verwendung von HTTP im Sensornetz und im Backend. Mit einer Nachrichtengröße von 55 Byte (Anfrage) und 91 Byte (Antwort) für eine einfache Temperaturabfrage ist der Ressourcenverbrauch jedoch zu hoch. Dabei ist der TCP- und IP-Overhead noch nicht mit eingerechnet.

Während das REST-Prinzip derzeit in der Enterprise-IT ausschließlich auf Basis von HTTP realisiert wird, schlagen WSN-Forscher CoAP (*Constrained Application Protocol*, [195]) als Realisierungsprotokoll vor. CoAP ist stark an HTTP angelehnt und ermöglicht genau wie dieses Protokoll eine Request-/Response-Kommunikation mit den Methoden *PUT*, *GET*, *POST* und *DELETE*. Anders als HTTP basiert CoAP auf UDP anstatt auf TCP und reduziert somit den Kommunikations-Overhead. Durch eine effizientere Codierung des Nachrichten-Headers sowie die Beschränkung auf wesentliche Header-Informationen kann im Vergleich

zu HTTP der Overhead weiter reduziert werden. Grundsätzlich ist CoAP jedoch eng an HTTP angelehnt, sodass eine Konvertierung zwischen beiden Protokollen problemlos möglich ist. Wie die Evaluation in Abschnitt 4.7 zeigt, ist aber auch mit CoAP ein bedeutender Overhead verbunden. Darüber hinaus setzt CoAP die Existenz von UDP und damit auch IP im Sensornetz voraus.

Priyatha et al. [178] präsentieren einen Webservice-Ansatz, der der WS-*-Architektur folgt und kein Konvertierungs-Gateway benötigt. Er verwendet WSDL zur Beschreibung von Webservices und das *HTTP-Binding* [232] zum Nachrichtenaustausch im Backend und im WSN. Dabei werden Nachrichten nach Möglichkeit in URLs codiert. Nur bei komplexen Nachrichten werden XML-Codierungen verwendet. Dieser Ansatz eignet sich grundsätzlich für den Einsatz im Rahmen eines Geschäftsprozessmanagements, da dieses Transport-Binding problemlos mit BPEL und vergleichbaren Sprachen verwendet werden kann. Allerdings ist der Overhead in diesem Ansatz zu hoch. Für das einfache Setzen eines Temperaturwertes wird eine Nachrichtengröße von 202 Byte erzeugt. Bei komplexeren Nachrichten, die nicht mehr als URLs codiert werden können, ist der Overhead gleich um ein Vielfaches höher. Dabei ist der Overhead von TCP und IP noch gar nicht mit berücksichtigt.

Delicato et al. [37] präsentieren einen Ansatz, der im Backend und auch im WSN (in komprimierter Form) SOAP-Nachrichten verwendet. Allerdings werden diese auf ein bestimmtes Format beschränkt und dienen nur als Container eines proprietären Kommunikationsformats. Darüber hinaus ist dieser Ansatz im Szenario dieser Arbeit nicht zu verwenden, da nur Anfragen vom Backend an das WSN gesendet werden können.

Als abschließende Bewertung können alle Gateway-basierten Ansätze als ungeeignet zur Umsetzung des in Abschnitt 3.4 beschriebenen Konzepts eines Geschäftsprozessmanagements im Kontext von Sensornetzen angesehen werden, da sie noch nicht einmal eine ganzheitliche Webservice-Kommunikation im Gesamtsystem ermöglichen. Ansätze, die kein Konvertierungs-Gateway benötigen, erlauben die Verwendung der jeweiligen Middleware im Gesamtsystem. Die meisten Autoren sehen RESTful-Webservices als geeigneter für den Einsatz in Sensornetzen als WS-*-Webservices, da sie als ressourcenschonender angesehen werden. Beim Vergleich der hier präsentierten Ansätze trifft diese Aussage auch zu. Die Evaluation in Abschnitt 4.7 zeigt jedoch, dass das im Rahmen dieser Arbeit entwickelte SOAP-Transport-Binding ressourcenschonender als die hier präsentierten REST-Umsetzungen ist, die außerdem als zu ressourcenverbrauchend anzusehen sind. Darüber hinaus eignen sich RESTful-Webservices ohnehin nicht für ein Geschäftsprozessmanagement, da BPEL und vergleichbare Prozessrealisierungssprachen die WS-*-Architektur voraussetzen.

Lediglich der Ansatz von Priyantha et al. erfüllt sowohl die Anforderungen an eine ganzheitliche Realisierung einer Webservice-Kommunikation im Gesamtsystem, die gleichzeitig auch mit BPEL und anderen Sprachen verwendet werden kann. Allerdings ist der Ressourcenverbrauch dieses Ansatzes für den Einsatz im Sensornetz zu hoch (s. o.).

4.2.2 Serialisierung von Webservice-Nachrichten

Wie bereits erwähnt, stellen SOAP-Nachrichten XML-Infosets dar, die gegenwärtig vor allem als XML-Dokumente serialisiert werden. Der damit verbundene Overhead verhindert bisher für Sensornetze den Einsatz von SOAP. Allerdings existieren bereits heute eine Reihe

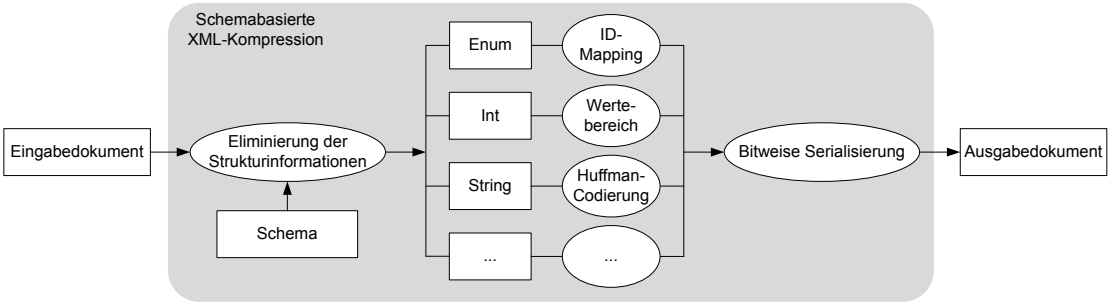


Abbildung 4.2: Funktionsweise einer schemabasierten XML-Kompression

von Lösungen, die eine effiziente Kompression von XML-Dokumenten oder eine kompakte, binäre Serialisierung von XML-Infosets erlauben. Dabei wird zwischen schemabasierter und schemaloser Kompression bzw. Serialisierung unterschieden.

Bei einer schemalosen Kompression werden keinerlei Informationen über die Grammatik eines XML-Dokuments bei der Kompression berücksichtigt. Für diese Art der Kompression können generische Kompressionsverfahren wie z. B. *GZIP* [38] eingesetzt werden. Zusätzlich existieren aber auch spezielle XML-Kompressoren wie *XMill* [113] oder *Fast Infoset* [91]. Insbesondere Fast Infoset hat sich als Standard bereits etabliert und wird auch außerhalb der universitären Forschung zur Kompression von SOAP-Nachrichten eingesetzt [149].

Bei einer schemabasierten Kompression bzw. Serialisierung wird das Wissen über die Grammatik eines XML-Dokuments bzw. -Infosets ausgenutzt, um diese optimiert komprimieren zu können. Abbildung 4.2 stellt die Funktionsweise dieses Ansatzes dar. Da die Schemainformationen und damit die Struktur eines Dokuments bei der (De-)Komprimierung bekannt sind, können diese Informationen weitestgehend eliminiert werden, sodass nur die eigentlichen Nutzdaten gespeichert werden müssen. Durch diese Vorgehensweise kann bereits der größte Kompressionseffekt erreicht werden. Zusätzlich beschreibt eine XML-Grammatik neben der grundlegenden Dokumentenstruktur meist auch sehr feingranular die Eigenschaften der Nutzdaten. So werden z. B. die Datentypen und ihre Wertebereiche oder Aufzählungen (engl.: *Enumeration*) sehr detailliert beschrieben. Diese Informationen können zur weiteren Größenreduktion ausgenutzt werden. So können z. B. Strings in einer effizienten Huffman-Codierung gespeichert werden, während Zahlen in Abhängigkeit ihres Wertebereichs genauso bitweise serialisiert werden können wie Aufzählungen, deren Strings durch IDs ersetzt werden.

Auch zur Realisierung einer schemabasierten XML-Kompression bzw. Serialisierung existieren bereits eine Reihe von Arbeiten. Neben *XGrind* [215] sind das vor allem *microFibre* [172] und *Xenia* [220, 221] aus der universitären Forschung. Mit Xenia und microFibre können im Vergleich zum Ursprungsdokument Kompressionsraten von bis zu 98 % erreicht werden.

Darüber hinaus existiert mit EXI (*Efficient XML Interchange*, [261]) bereits ein Standard, der den Status einer W3C-Empfehlung hat. EXI ermöglicht sowohl eine schemalose als auch eine schemabehaftete Kompression und kann durch diese Flexibilität in unterschiedlichsten Szenarien eingesetzt werden. Höller et al. [79] präsentieren einen weiteren Ansatz, der sowohl eine schemabasierte als eine schemalose Kompression erlaubt. Dieser zeichnet sich insbesondere dadurch aus, dass auf dem komprimierten XML-Dokument umfangreiche XML-Abfragen ausgeführt werden können.

4.3 LTP+SMC als ganzheitliches Webservice-Transport-Binding im Sensornetz und Backend

Zur Umsetzung des in Abschnitt 3.4 beschriebenen Konzepts zum Geschäftsprozessmanagement im Kontext von Sensornetzen ist eine ganzheitliche Webservice-Kommunikation im Gesamtsystem notwendig. Damit diese als Basis für etablierte Geschäftsprozessrealisierungsstandards dienen kann, muss sie vollständig konform zur Webservice-Architektur des W3C (WS-*-Architektur) sein. Wie in den vorigen beiden Abschnitten beschrieben, existiert gegenwärtig kein Webservice-Transport-Binding, das diese Anforderungen erfüllt.

In den folgenden Abschnitten wird das im Rahmen dieser Arbeit entwickelte SOAP-Webservice-Transport-Binding LTP+SMC beschrieben. Abbildung 3.3 auf Seite 60 zeigt seine Einordnung in den Webservice-Technologiestapel. Wie in Abschnitt 2.3.2 beschrieben, definiert der SOAP-Standard ein Format zum Austausch von Webservice-Nachrichten sowie allgemeine Regeln für den Nachrichtentransport. LTP+SMC verwendet zum ganzheitlichen und transparenten Nachrichtenaustausch im Gesamtsystem das im Rahmen dieser Arbeit entwickelte *Lean Transport Protocol* (LTP, s. Abschnitt 4.5). Die SOAP-Nachrichten werden nicht als XML-Dokumente, sondern komprimiert in einer effizienten, binären Codierung serialisiert. Diese wird im Folgenden als SMC (*SOAP Message Compression*) bezeichnet und im nächsten Abschnitt beschrieben.

LTP+SMC entspricht dem SOAP-Standard, der explizit die Verwendung nahezu beliebiger Transportprotokolle spezifiziert. Auch die binäre Serialisierung der Nachrichten ist standardkonform, da SOAP XML nur als eine mögliche Serialisierung vorschlägt. Als SOAP-Transport-Binding können die Schnittstellen von LTP+SMC-Webservices ohne Änderung des WSDL-Standards problemlos mit dieser Sprache beschrieben werden (s. Abschnitt 4.6).

4.4 Kompression von SOAP-Nachrichten

In der W3C-Webservice-Architektur stellt SOAP den bedeutendsten Standard der Nachrichtenschicht dar. Wie bereits in Abschnitt 4.1 festgestellt, halten XML-Dokumente und damit auch in XML-serialisierte SOAP-Nachrichten sowohl während des Transports als auch bei der Verarbeitung auf den Sensorknoten die Vorgaben bzgl. einer niedrigen Datenrate sowie eines geringen Verbrauchs an Speicher- und CPU-Ressourcen nicht ein. Im Gegensatz zur Version 1.1 erlaubt SOAP 1.2 neben XML noch beliebige weitere Serialisierungen. Dieses ermöglicht den Einsatz von Kompressions- oder effizienteren direkten Serialisierungsverfahren, um die Größe von SOAP-Nachrichten zu reduzieren und trotzdem die Standardkonformität zu wahren.

Grundsätzlich erreichen schemabasierte XML-Kompressionsverfahren, die Kenntnis der XML-Grammatik vorausgesetzt, größere Kompressionsraten als schemalose Kompressoren [219]. Im Rahmen dieser Arbeit durchgeführte Untersuchungen zeigen, dass dieser Vorteil proportional zur Restriktivität der Grammatik ist (s. Abschnitt 6.1.3). SOAP-Nachrichten stellen in der Regel sehr restriktive XML-Grammatiken dar. Sowohl das SOAP-Format selbst als auch der Inhalt des Bodys sind durch den SOAP-Standard sowie durch die WSDL-Beschreibung der Webservice-Schnittstelle inkl. Datentypen (meist in XML-Schema) genau definiert und

enthalten meist nur geringe Freiheitsgrade. Lediglich die Daten im Header sind in der Regel nicht genau oder zumindest nicht formal beschrieben. Aus diesem Grund wird in dieser Arbeit generell eine schemabasierte Kompression für den Body und eine schemalose Kompression für den Header von SOAP-Nachrichten empfohlen.

Im Rahmen dieser Arbeit wird SMC (*SOAP Message Compression*) zur Komprimierung bzw. Serialisierung von SOAP verwendet. SMC stellt kein eigenständiges Kompressionsverfahren dar. Es ist eine Spezifikation, die beschreibt, wie SOAP-Nachrichten im Transport-Binding LTP+SMC komprimiert werden müssen. Mit microFibre, Xenia, EXI und XGrind existieren bereits verschiedene Verfahren, die alle in SMC eingesetzt werden können. Besonders EXI eignet sich für eine SOAP-Kompression, da es sowohl eine schemalose als auch eine schemabasierte Kompression ermöglicht. Damit lassen sich sowohl für den SOAP-Body als auch für den -Header sehr gute Kompressionsergebnisse erzielen. Zusätzlich stellt EXI eine W3C-Spezifikation dar, sodass zukünftig wohl von einem hohen Verbreitungsgrad ausgegangen werden kann. Aus diesen Gründen wird auch im Rahmen dieser Arbeit für die Zukunft der Einsatz von EXI in SMC empfohlen. Da EXI erst im März 2011 den Status einer W3C-Empfehlung erhielt, wurde es bisher noch nicht in SMC verwendet.

Zurzeit wird stattdessen microFibre zur schemabasierten Kompression von SOAP-Nachrichten eingesetzt. Anders als z. B. bei Xenia werden bei microFibre nicht nur die XML-Strukturen (also Elemente und Attribute), sondern auch die Daten selbst in Abhängigkeit des Datentyps und definierter Restriktionen komprimiert. Dadurch kann die Kompressionsrate von microFibre weiter verbessert werden [171]. Zusätzlich existiert für microFibre ein umfangreiches Codegenerierungs-Framework, das aus einer in XML-Schema beschriebenen Grammatik sowie aus in WSDL definierten Nachrichtentypen Quellcode für die Programmiersprachen C und Java generiert. Dieser Code realisiert die Abbildung zwischen der Repräsentation der Daten- und Nachrichtenstrukturen im Arbeitsspeicher und der in XML sowie der in einem kompakten/komprimierten binären Format. Dabei wird auch eine direkte (De-)Serialisierung eines XML-Infosets aus der bzw. in die komprimierte Repräsentation ermöglicht. Nur so ist auch die Verarbeitung von SOAP auf Sensorknoten möglich, da keine unkomprimierten XML-Dokumente auf den Knoten gespeichert und verarbeitet werden müssen.

Der von microFibre erzeugte Quellcode enthält jedoch lediglich die Nachrichten- und Datenstrukturen sowie die entsprechende Serialisierungsfunktionalität für den Body von regulären SOAP-Nachrichten. Weder der Header noch Fehlernachrichten sind dabei berücksichtigt. Zum Einsatz in SMC musste das microFibre-Codegenerat im Rahmen dieser Arbeit um die entsprechenden Strukturen und die Serialisierungsfunktionalität erweitert werden. Als Ausgabe erhält der Entwickler nun SMC-Quellcode, zur kompletten (De-)Serialisierung der SOAP-Nachrichten eines Webservices und zur Abbildung der entsprechenden Daten- und Nachrichtenstrukturen im Arbeitsspeicher.

4.5 Lean Transport Protocol

In diesem Abschnitt wird das vom Autor dieser Arbeit entwickelte *Lean Transport Protocol* beschrieben. Dazu wird als Erstes die Architektur von LTP beschrieben, bevor im Anschluss die technischen Details des Protokolls erläutert werden. Die Forschungsergebnisse zu diesem Thema wurden bereits vorab in [69, 70, 72] veröffentlicht.

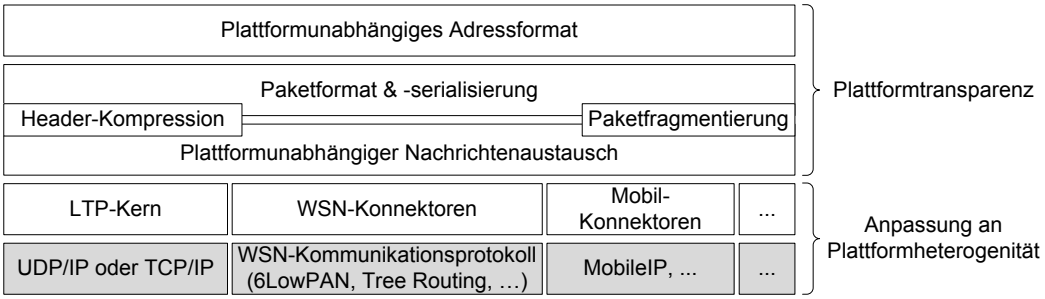


Abbildung 4.3: Aufbau von LTP

4.5.1 Architektur

Beim Design von LTP müssen zwei wesentliche, interferierende Anforderungen erfüllt werden. Zum einen muss wie in Abschnitt 3.4 beschrieben, eine transparente, ganzheitliche Kommunikation im Gesamtsystem erreicht werden. Gleichzeitig muss LTP jedoch in heterogenen Teilnetzen verwendet werden können. Dabei zeichnen sich diese Teilnetze, wie z. B. Sensornetze, insbesondere durch ihre Ressourcenbeschränkungen oder sonstige besondere Merkmale, die eine sehr spezifische Anpassung der Kommunikationsprotokolle an die Plattform oder gar an das Anwendungsszenario verlangen, aus (s. Abschnitt 2.4.3). LTP erfüllt durch seinen modularen Aufbau beide Anforderungen.

Abbildung 4.3 stellt den Aufbau von LTP schematisch dar. LTP besteht aus plattformunabhängigen Komponenten, die dem Anwendungsentwickler eine Plattformtransparenz bieten. Darüber hinaus existieren plattformspezifische Module, die die Anpassung der LTP-Kommunikation an die Besonderheiten der verschiedenen Sensornetze oder sonstigen Subnetzarten ermöglichen. LTP selbst besteht aus einem plattformunabhängigen Adressformat, einem Paketformat mit einer effizienten Nachrichtenserialisierung sowie einem plattformunabhängigen und plattformspezifischen Nachrichtenaustausch. Zum plattformspezifischen Nachrichtenaustausch definiert LTP verschiedene Konnektoren, die beliebige Protokolle unterschiedlicher Ebenen zum eigentlichen Nachrichtentransport verwenden. Dieses können z. B. UDP, TCP, 6LoWPAN oder *MobileIP* [92, 169], aber auch einfache Sensornetzprotokolle wie *Tree Routing* sein.

Im Rahmen dieser Arbeit liegt der Fokus auf der Einbindung von drahtlosen Sensornetzen in die Gesamtarchitektur. LTP erlaubt jedoch nicht nur die Anbindung von Sensornetzen über die Konnektoren. Andere Netze mit beschränkten Ressourcen (engl.: *Resource Constrained Network*, RCN) können genauso in das Internet integriert werden wie weitere Netze mit besonderen Eigenschaften wie z. B. mobile Geräte. So wurde z. B. im L2D2-Projekt ein LTP-Austausch über das vom Autor dieser Arbeit entwickelte LMMS-Protokoll (*Lean Mobile Messaging Service*) realisiert, um die Anforderungen einer Mobilkommunikation (z. B. temporäre Diskonnektivität) zu erfüllen [86].

Essentiell für einen transparenten Nachrichtenaustausch ist die Adressierung von Webservice- bzw. Kommunikationsendpunkten. LTP stellt zu diesem Zweck ein umfangreiches Adressierungsschema zur Verfügung. Es adressiert Endpunkte im Backend auf dieselbe Weise wie Endpunkte in RCNs. Dazu verwendet LTP, wie in Abbildung 4.4 gezeigt, ganzheitlich die im Webservice-Umfeld weitverbreitete URL-Notation. Eine LTP-URL hat das Protokollpräfix

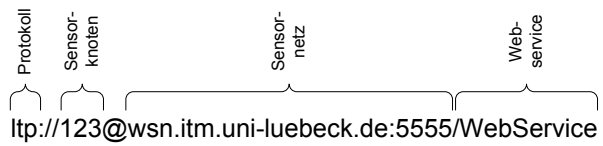


Abbildung 4.4: Beispiel einer LTP-URL zur Adressierung eines Sensorknotens

„ltp“ und kann Unicast-, Multicast- sowie Broadcast-Kommunikationsendpunkte adressieren. In diesem Beispiel wird der Webservice „WebService“ auf dem Sensorknoten „123“ adressiert. Dieser befindet sich im Sensornetz „wsn.itm.uni-luebeck.de:5555“.

Das Paketformat von LTP definiert den Aufbau der Nachrichten, die von LTP ausgetauscht werden. Der Header enthält umfangreiche Informationen zur Adressierung von Webservice-Endpunkten, -Operationen und -Nachrichten sowie zur Korrelation verschiedener LTP-Pakete. Mit diesen Informationen ermöglicht LTP die Realisierung aller in WSDL 1.1 definierten Nachrichtenaustauschmuster. Darüber hinaus erreicht LTP auch bzgl. der Adressierung von Endpunkten und der Flexibilität beim Routen von Nachrichten eine Mächtigkeit, die auch in Webservice-Transportprotokollen der Enterprise-IT nur durch die zusätzliche Verwendung von WS-Addressing möglich ist (z. B. die Definition von speziellen Endpunkten für Antwort- oder Fehlernachrichten). Obwohl LTP vor allem für den Transport von Webservice-Nachrichten entworfen wurde, können beliebige Daten als Nutzlast eines LTP-Pakets übertragen werden.

LTP verwendet eine sehr effiziente, bitweise Paketserialisierung. Trotzdem können die Paketgrößen sehr schnell relativ groß werden. Neben den als URLs codierten Kommunikationsendpunkten werden viele weitere Informationen im LTP-Header als Strings repräsentiert. Obwohl diese Codierung mit einem erhöhten Ressourcenverbrauch einhergeht, wurde dieses Design gewählt, um soweit wie möglich kompatibel zu WS-Addressing zu sein. Aus diesem Grund definiert LTP verschiedene Header-Kompressionstechniken, um die Paketgröße zu reduzieren und die Ressourcen zu schonen. Insgesamt erreicht LTP damit einen sehr geringen Protokoll-Overhead.

Trotz des geringen Protokoll-Overheads hängt die Paketgröße letztendlich von der zu übertragenden Nachricht ab. Die Transceiver von Sensorknoten begrenzen die maximale Größe von Nachrichten sehr stark. IEEE-802.15.4-kompatible Plattformen erlauben z. B. nur eine maximale Paketgröße von 127 Byte. Um nicht die maximale Größe eines LTP-Pakets entsprechend zu limitieren, bietet LTP die Möglichkeit der Paketfragmentierung.

Der Transport wird in LTP mit dem plattformunabhängigen und plattformspezifischen Nachrichtenaustausch auf zwei Ebenen realisiert. Der plattformunabhängige Austausch definiert Regeln, wie LTP-Pakete unabhängig von einem konkreten Transportmechanismus geroutet werden müssen. Zusammen mit der plattformunabhängigen Adressierung kann so, wie in Abbildung 4.5 gezeigt, der transparente Nachrichtenaustausch zwischen Sensorknoten innerhalb desselben Sensornetzes (1), Sensorknoten und Komponenten im Backend (2), also im Internet, zwischen verschiedenen Enterprise-IT-Komponenten (3) sowie zwischen Sensorknoten in unterschiedlichen, entfernten Sensornetzen (4) realisiert werden. Dabei ist die Kommunikation nicht auf eine einzelne Organisation begrenzt.

Unterhalb des plattformunabhängigen Nachrichtenaustauschs verwendet LTP verschiedene Protokolle zum Austausch von Nachrichten innerhalb der einzelnen Subnetze. In LTP stellt

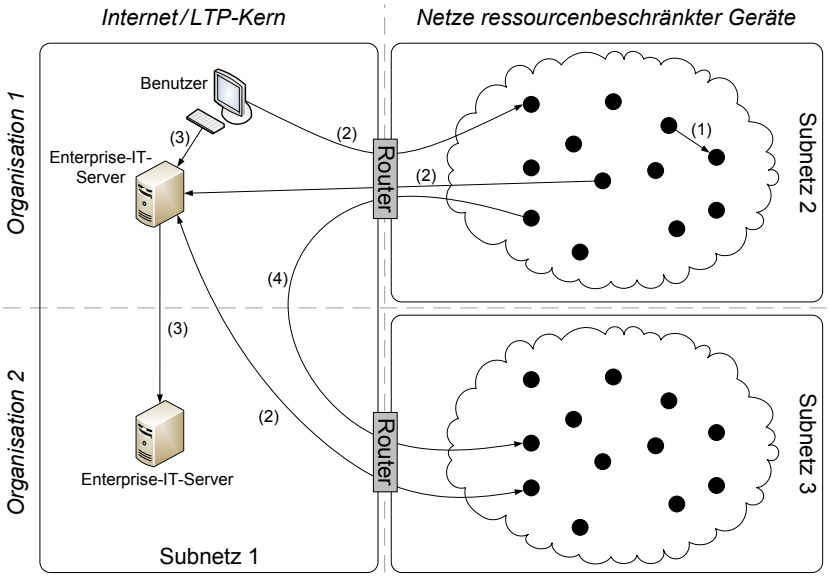


Abbildung 4.5: Nachrichtenaustausch mit LTP

das Internet das zentrale Subnetz, den LTP-Kern, dar. Zum Nachrichtentransport im Internet verwendet LTP je nach Konfiguration wahlweise UDP oder TCP. Über sog. Konnektoren können weitere Subnetze mit dem LTP-Kern vernetzt werden. Für jedes Subnetz werden individuell das in diesem Subnetz verwendete Transportprotokoll sowie Konfigurationen für die Kompression und Fragmentierung von Paketen definiert. Zum Transport können dabei verschiedene Protokolle unterschiedlicher Schichten eingesetzt werden, solange sie eine Ende-zu-Ende-Kommunikation zwischen den Geräten im entsprechenden Subnetz ermöglichen. Zwischen dem LTP-Kern und den angebundenen Subnetzen realisieren anwendungsunabhängige, zustandslose Router die Weiterleitung der Pakete über Subnetzgrenzen hinweg. Dazu wenden die Router die Fragmentierungs- und Kompressionsstrategien des Zielnetzes an und leiten das Paket entsprechend der plattformunabhängigen Routing-Regeln mit dem plattformspezifischen Transportprotokoll des Zielteilnetzes weiter.

4.5.2 Paketformat und -serialisierung

Abbildung 4.6 beschreibt den Aufbau eines LTP-Pakets. Es besteht aus einem Header und einem Body. Während der Body die eigentlich zu übertragenen Daten enthält, besteht der Header insbesondere aus Adressierungsinformationen sowie Auskünften zur Semantik eines Pakets. Um eine maximale Kompatibilität zu WS-Addressing zu gewährleisten, wurden die Felder so gestaltet, dass die wichtigsten WS-Addressing-Felder direkt auf LTP-Felder abgebildet werden können. Gleichzeitig wird jedoch eine kompakte Codierung der Header-Information in LTP realisiert, sodass der Ressourcenverbrauch minimiert werden kann.

LTP definiert vier Felder zur Adressierung von Kommunikationsendpunkten. Endpunkte können prinzipiell beliebige Kommunikationspartner beschreiben. Da LTP jedoch zur Webservice-Kommunikation entworfen wurde, sind dieses typischerweise Webservice-Endpunkte. *to*, *from*, *replyTo* und *faultTo* beschreiben die Empfänger- und Absenderendpunkte eines LTP-Pakets

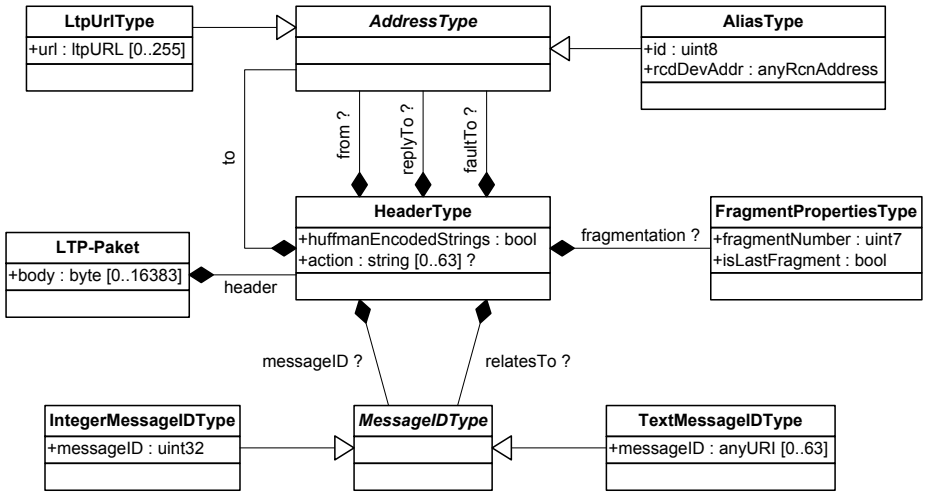


Abbildung 4.6: LTP-Paketformat

sowie die Endpunkte, an die die Antwort- oder Fehlernachricht gesendet werden soll. Während *to* in jedem Paket verpflichtend ist, sind *from*, *replyTo* und *faultTo* optional. Alle Adressierungsfelder von LTP sind vom Datentyp *AddressType*. *AddressType* ermöglicht die Codierung eines Endpunkts entweder in der URL-Notation (mit einer maximalen Länge von 255 Zeichen) oder als Identifikator vom Typ *AliasType*. Aliase ersetzen bei der Verwendung von Header-Kompressionstechniken URL-codierte Endpunkte durch eine kompakte Repräsentation. Zu Details zur Header-Kompression sei auf den Abschnitt 4.5.3 verwiesen.

Zur Beschreibung der Semantik eines LTP-Pakets werden optional die Felder *action*, *messageID* und *relatesTo* definiert. *action* enthält eine beliebige, maximal 63 Zeichen lange URI, die prinzipiell eine beliebige Aktion beim Kommunikationsendpunkt auslösen kann. Im Kontext einer Webservice-Kommunikation identifiziert das *action*-Feld die Eingangs-, Ausgangs- oder Fehlernachricht einer Operation eines Webservices. Dazu kann zum einen das in der korrespondierenden WSDL-Beschreibung definierte *SOAP-Action*-Feld verwendet werden. Zum anderen kann die Kennzeichnung der Ein- und Ausgabe- sowie Fehlernachrichten einer Operation in WSDL mit den entsprechenden Elementen aus der WSDL-Erweiterung von WS-Addressing [246, 257] zur Identifikation von Nachrichten verwendet werden (s. Abschnitt 4.6).

Die Felder *messageID* und *relatesTo* haben dieselbe Semantik wie die gleichnamigen Felder in WS-Addressing. *messageID* identifiziert ein LTP-Paket eindeutig, während *relatesTo* die *messageID* einer korrelierenden Nachricht enthält. So kann z. B. ausgedrückt werden, dass ein LTP-Paket die Antwort auf ein vorausgegangenes Paket darstellt. In LTP sind beide Felder vom Datentyp *MessageIDType*. In diesem Typ kann die ID analog zu WS-Addressing als URI (mit einer maximalen Länge von 63 Zeichen) beschrieben werden. Alternativ kann zur Kompression die ID auch als positive, ganzzahlige 32-Bit-Zahl codiert werden.

LTP-Pakete können optional fragmentiert werden. Um diese Eigenschaft im Paketformat abzubilden, definiert LTP ein optionales Feld vom Typ *FragmentationPropertiesType*. Es enthält die Fragmentnummer des aktuellen Teilpakets als positive, ganzzahlige 7-Bit-Zahl codiert und einen booleschen Wert, der das letzte Teilpaket eines fragmentierten LTP-Pakets kennzeichnet. Zu Details zur Fragmentierung sei auf den Abschnitt 4.5.5 verwiesen.

Als letztes Feld des LTP-Headers wird eine boolesche Kennzeichnung gesetzt. Sie beschreibt die Codierung aller als Strings repräsentierten Header-Informationen. Dabei können Strings entweder in ASCII (*American Standard Code for Information Interchange*, [30]) oder in einer effizienten Huffman-Codierung [81] serialisiert werden.

Der LTP-Body enthält die eigentlichen Nutzdaten. LTP kann prinzipiell beliebige Daten übertragen. Da das Protokoll jedoch speziell für den Transport von Webservice-Nachrichten entworfen wurde, werden im Body vor allem (komprimierte) SOAP-Nachrichten übertragen. Der Inhalt, das Format sowie die Codierung des Bodys werden deshalb typischerweise in einer WSDL-Beschreibung festgelegt (s. Abschnitt 4.6).

LTP-Pakete müssen möglichst klein sein, um die Kapazitäten der ressourcenbeschränkten Netze nicht zu übersteigen. Aus diesem Grund verwendet LTP eine kompakte, bitweise Serialisierung (s. Anhang A).

4.5.3 Header-Kompression

Wie gerade beschrieben, enthalten einige Felder des LTP-Paket-Headers Informationen, die als Strings codiert werden können. Daraus resultiert ein Overhead, der zwar für Backend-Systeme irrelevant ist. Für Geräte bzw. Netze mit beschränkten Ressourcen resultiert daraus jedoch ein wesentlicher Ressourcenverbrauch. Aus diesem Grund realisiert LTP verschiedene Header-Kompressionsmechanismen. Dadurch kann nicht nur das Nachrichtenaufkommen, sondern auch der Verbrauch an Rechenkapazität und Arbeitsspeicher reduziert werden.

Bei LTP werden zwei verschiedene Kompressionsstrategien verfolgt. Zum einen werden Informationen, die als Strings codiert sind, durch eine kompaktere Repräsentation ersetzt. Zum anderen kann auch die String-Repräsentation beibehalten werden. Für diesen Fall ist eine im Vergleich zur ASCII-Codierung kompaktere Huffman-Codierung vorgesehen. Dazu wird ein statischer Huffman-Code für LTP vorgegeben. Dieser entstand aus der statistischen Analyse von 25.000 zufällig aus dem Internet ermittelten URIs.

Obwohl sich eine Huffman-Codierung positiv auf die Nachrichtengröße auswirkt, geht mit ihr ein erheblicher zusätzlicher Ressourcenverbrauch bei der CPU und Codegröße einher. Aus diesem Grund ist die Huffman-Codierung aller Strings in einem Paket nur optional und kann über das entsprechende Feld im Paket-Header gekennzeichnet werden. Unterstützt ein Empfänger keine Huffman-Codierung, muss er eine entsprechende Fehlernachricht an den Absender zurückschicken, der das Paket mit einer ASCII-Codierung erneut verschicken muss.

Obwohl die Länge von Strings mit dieser Strategie für Längen zwischen einem und 255 Zeichen im Mittel auf 74,6 % im Vergleich zu ASCII reduziert werden kann und im Fall von Endpunkt-URLs durch das Entfernen des in den serialisierten Paketfeldern überflüssigen Protokollpräfixes („ltp://“) der Overhead weiter vermindert werden kann, verursachen Strings weiterhin den größten Teil der Header-Größe. Aus diesem Grund werden zur weiteren Größenreduzierung Strings durch kompaktere Repräsentanten ersetzt. Zur Kompression von Adressierungsinformationen können URLs in RCNs durch Aliase ersetzt werden. Wie in Abbildung 4.6 gezeigt, bestehen diese aus den Feldern *id* und *rcdDevAddr*. *id* ist ein lokal eindeutiger Identifikator, der eine URL als ganzzahlige 8-Bit-Zahl repräsentiert. *rcdDevAddr* enthält die Geräteadresse des Kommunikationsendpunkts, der von der entsprechenden URL adressiert wird. Falls die

zu ersetzende URL einen Endpunkt außerhalb des lokalen Subnetzes adressiert, enthält *rcdDevAddr* die Geräteadresse, über die der Router mit dem lokalen Subnetz verbunden ist. Je nach Art und Größe der Geräteadressen in den entsprechenden Teilnetzen kann *rcdDevAddr* in unterschiedlichen Subnetzen eine abweichende Größe und Codierung aufweisen. In WSNs werden Knoten in der Regel über eine 16-Bit-Zahl adressiert. In diesem Fall trägt auch *rcdDevAddr* für diese Teilnetze eine Zahl dieser Größe. Das Tupel aus *id* und *rcdDevAddr* identifiziert eine URL immer eindeutig. Jede URL kann jedoch auf mehrere Aliase abgebildet werden.

Jeder Sender eines LTP-Pakets sowie jeder Router können Endpunkt-URLs durch Aliase ersetzen. Aliase sind jedoch nur im lokalen Subnetz gültig. Aus diesem Grund müssen die Router bei eingehenden Paketen die Adressinformationen je nach Routing-Richtung komprimieren oder dekomprimieren.

LTP sieht die Ersetzung von URLs durch Aliase ausschließlich in RCNs und nicht im LTP-Kern vor. Diese Designentscheidung resultiert aus der Notwendigkeit einer Alias-URL-Auflösung. Eine Auflösungsstrategie muss innerhalb eines Teilnetzes einheitlich sein. Aber keine der im Folgenden vorgestellten Strategien lässt sich sinnvoll im gesamten Kern anwenden. Da jedoch davon ausgegangen wird, dass der Kern ohnehin aus nicht ressourcenbeschränkten Geräten besteht, ist eine Kompression in diesem Teilnetz auch nicht zwingend notwendig.

Die URL-Kompression basiert auf der Ersetzung von URLs durch Aliase. Bei der Abbildung (engl: *Mapping*) zwischen beiden Endpunktrepräsentationen kann LTP verschiedene Strategien verwenden. So kann prinzipiell in einem statischen Szenario mit homogenen Diensten, die auf vielen Knoten innerhalb eines Subnetzes ausgeführt werden, eine regelbasierte Abbildung erfolgen. Spätestens jedoch, wenn ein Subnetz heterogene Dienste ausführt und gar verschiedenen Besitzern gehört, funktioniert diese Strategie nicht mehr. Aus diesem Grund wird ein Ansatz vorgeschlagen, der die Verwendung dezentraler Infrastrukturdienste vorsieht, die die Abbildung zwischen Alias und URL für das jeweilige Subnetz verwalten.

Infrastrukturdienste stellen ganz normale Webservices dar, die über LTP+SMC verwendet werden können. Damit können sie sowohl im WSN als auch im Backend ausgeführt werden. Da dieser Dienst jedoch bezogen auf die Ressourcen von WSNs relativ große Datenmengen verwalten muss, wird die Ausführung des Infrastrukturdienstes im Backend empfohlen. Abbildung 4.7 beschreibt die Schnittstelle des Infrastruktur-Webservices (IWS) und seine Interaktion mit einem LTP-Knoten sowie die entsprechenden Anwendungslogiken. Jeder LTP-Endpunkt kann Aliase registrieren sowie Abbildungen abrufen. Wird ein Alias für eine noch nicht registrierte URL abgerufen, dann erzeugt und speichert der IWS automatisch einen neuen Alias. Um den Kommunikations-Overhead zu minimieren, wird empfohlen, dass die Knoten eine gewisse Anzahl an Abbildungen zwischenspeichern und nur Auflösungen beim IWS vornehmen, wenn der lokale Zwischenspeicher (engl.: *Cache*) eine benötigte Abbildung nicht enthält. Alternativ zu dieser aktiven Aliasauflösung (engl.: *Pull*), kann auch eine passive Strategie (engl.: *Push*) verfolgt werden. Der IWS sendet in diesem Szenario periodisch oder ereignisorientiert die Abbildungen per LTP-Broadcast in das Subnetz. Die Knoten speichern dann die für sie relevanten Abbildungen. Periodisch entfernen sowohl der IWS als auch die Knoten Abbildungen, die über einen längeren Zeitraum ungenutzt waren, wieder aus der Abbildungstabelle bzw. dem Zwischenspeicher.

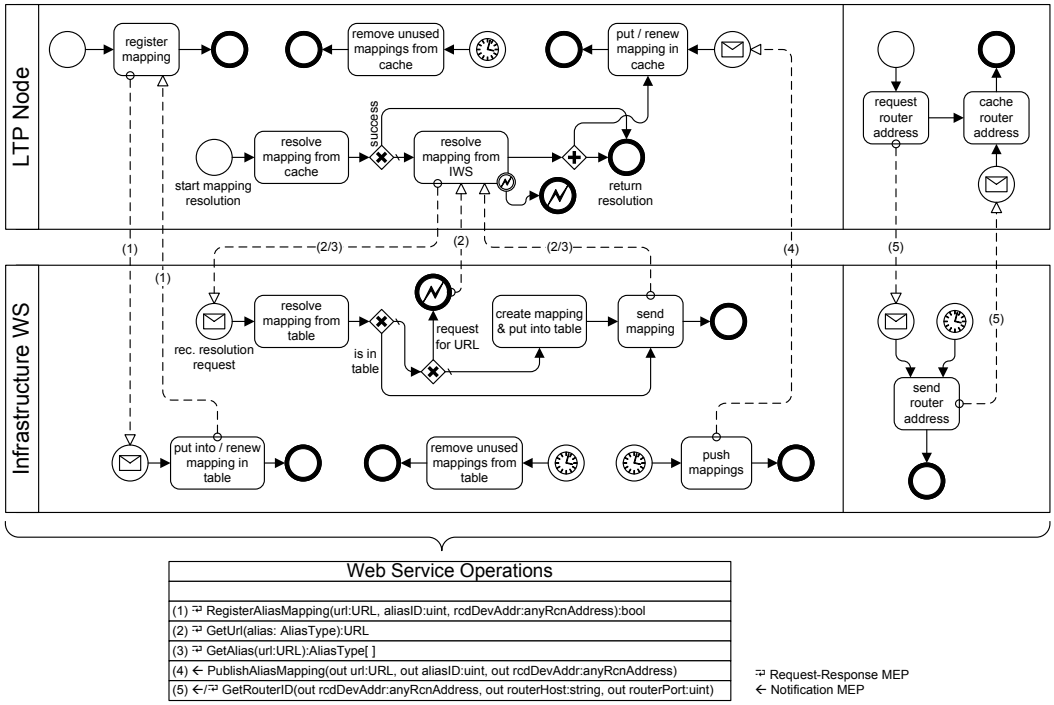


Abbildung 4.7: Infrastrukturdienst und die Interaktion mit einem LTP-Kommunikationsendpunkt

Der Alias enthält mit der Geräteadresse bereits Informationen, die in den meisten Fällen auch ohne Aliasauflösung Routingentscheidungen erlauben. Lediglich in drei Fällen muss ein Alias zwingend aufgelöst werden. i) Bei der Weiterleitung von LTP-Paketen im Router. Dieser Fall ist unproblematisch, da sowohl der Router und in der Regel auch der Infrastrukturdienst auf ressourcenstarker Hardware ausgeführt werden. ii) Beim Versand einer Nachricht, die keine Antwort zu einer vorher empfangenen Nachricht darstellt. In diesem Fall kann der Alias nicht aus der Anfragenachricht entnommen werden und der Knoten muss für die Endpunkt-URL den Alias auflösen. iii) Darüber hinaus muss jeder Knoten die Abbildungen für die Dienste kennen, die von ihm selbst angeboten werden. Nur so kann der Knoten Anfragen mit Aliasen konkreten Webservice-Endpunkten zuordnen. Für den reinen Nachrichtenaustausch würde auch in diesem Fall die Kenntnis des Alias ausreichen.

Die Verwendung der URL-Kompression mit Aliasen wird innerhalb von RCNs empfohlen. Die direkte Verwendung von URLs ist jedoch jederzeit möglich. Somit kann auch beim Ausfall eines IWS eine Kommunikation in dem entsprechenden Subnetz weiter gewährleistet werden.

Neben der Möglichkeit der Alias-Auflösung stellt der IWS noch eine weitere Operation bereit. Über *GetRouterID* kann per Push- und Pull-Prinzip die *rcdDevAddr* eines Routers, über den das Subnetz mit dem LTP-Kern verbunden ist, dynamisch ermittelt werden. Zusätzlich werden der Rechnername bzw. die IP-Adresse und der Port des Routers übermittelt, über die das WSN aus dem LTP-Kern erreichbar ist. Dazu werden das *Notification*-MEP mit einer Multicast- oder Broadcast- sowie das *Request-Response*-MEP mit einer Unicast-Kommunikation eingesetzt.

Neben den Adressierungsfeldern enthalten auch *messageID* und *relatesTo* im Standardfall

URIs. Damit wird eine direkte Abbildung der entsprechenden WS-Addressing-Felder nach LTP ermöglicht. Wird diese LTP-Eigenschaft nicht benötigt, ist die sehr viel effizientere Repräsentation der IDs als 32-Bit-Zahl empfohlen.

Die explizite Adressierung einer Webservice-Operation ist in vielen Fällen überflüssig, da diese Information je nach Serialisierung einer SOAP-Nachricht auch aus dieser abgeleitet werden kann. Auch ist es möglich, bei Antwortnachrichten diese Informationen aus der entsprechenden Anfragenachricht herzuleiten. In diesen Fällen muss das *action*-Feld von LTP nicht gesetzt werden.

4.5.4 Nachrichtenaustausch

LTP ermöglicht einen transparenten Ende-zu-Ende-Nachrichtenaustausch zwischen beliebigen Kommunikationsendpunkten im Gesamtsystem. Dieser wird mit dem plattformunabhängigen und dem plattformspezifischen Nachrichtenaustausch auf zwei verschiedenen Ebenen realisiert. Zur Adressierung der Endpunkte verwendet LTP das plattformunabhängige Adressierungsschema.

Wie in Abschnitt 4.5.1 beschrieben, wird ein LTP-Netz aus verschiedenen Teilnetzen gebildet. In ihnen verwendet LTP jeweils unterschiedliche, existierende Kommunikationsprotokolle, die eine auf das jeweilige Subnetz begrenzte Ende-zu-Ende-Kommunikation als plattformspezifischen Nachrichtenaustausch gewährleisten. Über Konnektoren werden diese Protokolle bzw. die entsprechenden Subnetze an LTP angebunden. Auf dieser Menge an Subnetzen bzw. Konnektoren baut LTP mit dem plattformunabhängigen Nachrichtenaustausch als zweite Transportebene auf und realisiert einen Ende-zu-Ende-Nachrichtenaustausch zwischen beliebigen Kommunikationspartnern im Gesamtsystem. Dazu definiert LTP eine Menge allgemeiner Routingregeln, die unabhängig von einem konkreten Netzprotokoll den Nachrichtenaustausch innerhalb eines Subnetzes sowie über Subnetzgrenzen hinweg beschreiben. Innerhalb der einzelnen Subnetze wird dann zum eigentlichen Nachrichtentransport das im jeweiligen Konnektor definierte plattformspezifische Kommunikationsprotokoll verwendet.

Die Paketweiterleitung zwischen Subnetzen wird durch Router realisiert. Wie in Abschnitt 4.5.1 beschrieben, muss jedes Subnetz von Geräten mit beschränkten Ressourcen oder sonstigen besonderen Eigenschaften an den LTP-Kern angebunden werden. In dieser Topologie ergeben sich drei Kommunikationsmuster. i) Ein Nachrichtenaustausch zwischen Kommunikationsendpunkten im selben Subnetz, ii) zwischen Endpunkten und Routern sowie iii) zwischen verschiedenen Routern.

Zur Adressierung beliebiger Endpunkte verwendet LTP das plattformunabhängige Adressierungsschema. Es definiert eine spezielle zum URI-Standard konforme URL-Notation. Abbildung 4.4 auf Seite 75 zeigt beispielhaft eine LTP-URL, dessen allgemeiner Aufbau in Algorithmus 4.1 in der EBNF-Darstellung (*Erweiterte Backus-Naur-Form*, [87]) formal beschrieben ist. Als Protokollpräfix wird „ltp“ verwendet. Der für den Nachrichtenaustausch relevante Teil der URL ist *authority*. *authority* beschreibt den Nachrichtenaustausch im LTP-Kern und im RCN. Durch die Angabe einer IP-Adresse oder eines Rechnernamens in der DNS-Notation (*Domain Name System*, [139]) und der optionalen Angabe eines TCP- oder UDP-Ports (Standardport: 6666) wird der Empfänger eines Pakets im LTP-Kern definiert. Enthält *authority* ein „@“, dann wird ein Endpunkt innerhalb eines RCN adressiert. In diesem Fall

Algorithmus 4.1 Formale Beschreibung einer LTP-URL in EBNF

```
 1: ltpUrl      =  "ltp://" authority [ path [ "?" query ] [ "#" fragment ] ] .
 2: authority   =  ( rcdDevAddr "@" routerHost [ ":" routerPort ] ) | ( eprHost [ ":" eprPort ] ) .
 3: rcdDevAddr  =  { unreserved | pctEncoded | subDelims | ":" } .
 4: path        =  { "/" { pchar } } .
 5: query       =  { pchar | "/" | "?" } .
 6: fragment    =  { pchar | "/" | "?" } .
 7: pchar       =  unreserved | pctEncoded | subDelims | ":" | "@" .
 8: routerHost  =  host .
 9: eprHost     =  host .
10: eprPort     =  port .
11: routerPort  =  port .
12: host        =  IPLITERAL | DNSNAME .
13: port        =  UDPPORT | TCPPORT .
14: pctEncoded  =  "%"HEXDIG HEXDIG .
15: unreserved  =  ALPHA | DIGIT | "-" | "." | "_" | "~" .
16: subDelims   =  "!" | "$" | "&" | "'" | "(" | ")" | "*" | "+" | "," | ";" | "=" .
```

legen die Rechneradresse und der -port (*routerHost* und *routerPort*) den Router fest, über den dieses Subnetz zu erreichen ist. *rcdDevAddr* beschreibt dann das tatsächliche Endgerät innerhalb des entsprechenden Subnetzes. Von einigen reservierten Zeichen abgesehen, kann *rcdDevAddr* prinzipiell einen beliebigen String enthalten. Der genaue Aufbau hängt von den im RCN verwendeten Geräteadressen ab und muss sich auf das gleichnamige Feld im Alias abbilden lassen. Verwendet z. B. ein WSN eine 16-Bit-Zahl zur Adressierung, dann darf *rcdDevAddr* für dieses Subnetz auch nur diese Zahlen enthalten.

LTP-URLs können je nach Möglichkeiten der verwendeten plattformspezifischen Protokolle neben Unicast- auch Multicast- oder Broadcast-Endpunkte adressieren. Dazu müssen in *rcdDevAddr* oder *host* jeweils die Multicast- oder Broadcast-Adressen des jeweiligen Subnetzes verwendet werden.

Neben den Informationen zum Routing kann eine LTP-URL zusätzlich optional über einen Pfad (engl.: *path*) verfügen, der optional um einen Abfrage- (engl.: *query*) sowie einen Fragmentteil (engl.: *fragment*[2]) ergänzt werden kann. Für eine Webservice-Kommunikation ist insbesondere der Pfad von Bedeutung, da nur so auf einem Knoten im RCN sowie in einer Applikation auf einem Gerät im Kern mehrere Webservice-Endpunkte adressiert werden können. *path*, *query* und *fragment* entsprechen in ihrem Aufbau dem URI-Standard. Die beiden Letztgenannten sind für eine WS-*-Webservice-Kommunikation weniger von Bedeutung, da zum Austausch von SOAP-Nachrichten in der Regel das *SOAP-Request-Response-MEP* verwendet wird. Dieses legt fest, dass sowohl in einer SOAP-Anfrage als auch in einer eventuellen -Antwort SOAP-Nachrichten verwendet und diese in der Nutzlast des Transportprotokolls übertragen werden. Mit *path*, *query* und *fragment* lässt sich zusätzlich auch mit *SOAP-Response* das zweite SOAP-MEP umsetzen. In diesem Fall wird nur eine Antwortnachricht als SOAP-Nachricht codiert. Die Anfrage wird im Transportprotokoll direkt, z. B. in *path*, *query* und *fragment*, codiert. Auf dieselbe Art wird der Einsatz von LTP auch in weiteren Szenarien realisiert. So kann z. B. auch das REST-Paradigma mit LTP effizient realisiert werden, was jedoch in dieser Arbeit nicht weiter beschrieben wird.

Für das Versenden und Weiterleiten von LTP-Paketen definiert das Protokoll verschiedene Regeln. Den einfachsten Fall stellt das Versenden einer Nachricht von einem Endpunkt im

[2] *fragment* ist für die Nutzung durch die Applikation bestimmt. Für die Fragmentierung von LTP-Paketen hat es keine Bedeutung.

Algorithmus 4.2 Versand eines LTP-Pakets durch ein Gerät in einem ressourcenbeschränkten Subnetz

Input: Let t be the *to* field of the LTP packet to route. Let t_{Alias} and t_{URL} be the respective Alias and URL values of t. Let h_{R} be the host, p_{R} the UDP or TCP port and n_{R} the device address of the router's link to the local RCN.
Output: d as next hop destination of p in the local RCN.

1: $\langle isCompr(t) \rangle \longrightarrow \langle d := rcdDevAddr(t_{\text{Alias}}) \rangle$
2: $\langle \neg isCompr(t) \wedge host(t_{\text{URL}}) = h_{\text{R}} \wedge port(t_{\text{URL}}) = p_{\text{R}} \wedge rcdDevAddr(t_{\text{URL}}) \neq N/A \rangle$
$\longrightarrow \langle d := rcdDevAddr(t_{\text{URL}}) \rangle$
3: $\langle \neg isCompr(t) \wedge (host(t_{\text{URL}}) \neq h_{\text{R}} \vee port(t_{\text{URL}}) \neq p_{\text{R}}) \rangle \longrightarrow \langle d := n_{\text{R}} \rangle$

Algorithmus 4.3 Weiterleitung eines LTP-Pakets an Routern

Input: Let m be the LTP packet to route and t_{Alias} respectively t_{URL} the Alias and URL value of the *to* field of m.

1: **if** m was received from the Internet **then**
2: let d be a device address used in the connected RCN n
3: $d := \begin{cases} rcdDevAddr(t_{\text{Alias}}) & isCompressed(t) \\ rcdDevAddr(t_{\text{URL}}) & \neg isCompressed(t) \end{cases}$
4: send m to d using the networking protocol of n
5: **else if** m was received from the RCN **then**
6: **if** $isCompressed(t)$ **then**
7: $t_{\text{URL}} := \begin{cases} getUrlFromCache(t_{\text{Alias}}) & isUrlInCache(t_{\text{Alias}}) \\ getUrlFromIWS(t_{\text{Alias}}) & else \end{cases}$
8: **end if**
9: send m to $(host(t_{\text{URL}}), port(t_{\text{URL}}))$ using TCP or UDP
10: **end if**

LTP-Kern dar, da im Kern die Verwendung von Aliasen nicht erlaubt ist. Unabhängig davon, ob sich der Empfänger einer Nachricht im Kern oder in einem RCN befindet, schickt der Absender das Paket mit TCP oder UDP an den Rechner und den Port, die in der Ziel-URL angegeben sind.

Algorithmus 4.2 beschreibt das Versenden eines LTP-Pakets auf einem Gerät innerhalb eines RCN. Pakete, deren Ziel als Alias codiert sind, können direkt an die darin enthaltene Geräteadresse gesendet werden. Dabei ist es irrelevant, ob der Alias einen Endpunkt innerhalb oder außerhalb des lokalen Subnetzes adressiert. Ist bei einem zu versendenden Paket nur die URL und nicht der Alias des Empfängers bekannt, muss unterschieden werden, ob der Endpunkt innerhalb oder außerhalb des lokalen Subnetzes liegt. Subnetzinterne Pakete können direkt an den Empfänger (*rcdDevAddr*) gesendet werden, während Pakete für Empfänger außerhalb des Subnetzes an den Router gesendet werden müssen. Zur Entscheidung, ob eine URL einen subnetzinternen oder -externen Endpunkt adressiert, muss der Rechner (h_{R}) und Port (p_{R}), mit dem der Router mit dem Kern verbunden ist, bekannt sein. Diese Informationen können statisch konfiguriert oder dynamisch vom Infrastrukturdienst bezogen werden (s. Abbildung 4.7 auf Seite 80). Gleiches gilt für die *rcdDevAddr* (n_{R}) des Routers.

Algorithmus 4.3 beschreibt die Paketweiterleitung an Routern. Aus dem Kern empfangene und an ein RCN gerichtete Pakete werden von Routern mit den entsprechenden Kommunikationsprotokollen der RCNs weitergeleitet. Unabhängig von der Routingrichtung wird dieser Algorithmus jeweils nach der (De-)Komprimierung und (De-)Fragmentierung von Paketen ausgeführt. Pakete mit Ziel RCN enthalten je nach Konfiguration der Konnektoren der RCNs die Zieladresse in der URL oder im Alias. Beide Repräsentationen enthalten mit *rcdDevAddr* die für den plattformspezifischen Transport notwendige Geräteadresse, an die das Paket mit dem lokalen Subnetzprotokoll an den endgültigen Empfänger weitergeleitet wird. Pakete aus dem RCN, die an den Kern oder einen anderen Router weitergeleitet werden sollen, werden

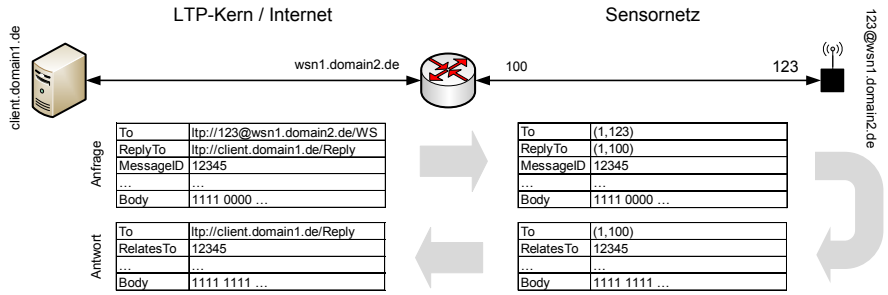


Abbildung 4.8: Beispiel einer LTP-Kommunikation

nach einer Defragmentierung und Dekompression der URL direkt per TCP oder UDP an den in der URL codierten Rechner und Port gesendet. Aliase sind auf ein Subnetz beschränkt und dürfen im Kern nicht verwendet werden. Die Alias-nach-URL-Auflösung in Zeile 7 ist also aufgrund der bereits zwingend durchgeführten Dekompression der URL-Komprimierung an dieser Stelle nicht mehr notwendig und eigentlich Teil der Header-Kompression und nicht des Routings. Sie wurde in diesem Algorithmus jedoch bewusst aufgeführt, um zu zeigen, dass lediglich an dieser Stelle keine direkte Routingentscheidung ohne Auflösung des Alias möglich wäre. In allen anderen Routingentscheidungen in den Algorithmen 4.2 und 4.3 reicht auch die Kenntnis von Aliasen aus.

Mithilfe der eben beschriebenen Routingregeln können LTP-Pakete transparent zwischen beliebigen Kommunikationsendpunkten im Gesamtsystem ausgetauscht werden. Durch den paketorientierten Nachrichtenaustausch realisiert LTP grundsätzlich erstmal eine asynchrone Kommunikation. Durch das Setzen der Felder *from*, *replyTo* und *faultTo* können der Absender eines Pakets sowie die Empfänger von regulären Antwortpaketen und Fehlernachrichten festgelegt werden. Mit den Feldern *messageID* und *relatesTo* können Pakete eindeutig identifiziert und in Beziehung gestellt werden. Damit realisiert LTP neben einer asynchronen auch eine synchrone Kommunikation.

Die in diesem und vorigen Abschnitt beschriebenen formalen Aspekte der Header-Kompression und des Nachrichtenaustauschs in LTP werden an einem kurzen Beispiel abschließend betrachtet (s. Abbildung 4.8). Eine Nachricht soll von einem PC an einen Webservice auf dem Sensorknoten „123“ im Sensornetz „wsn1.domain2.de:6666“ geschickt werden. Dazu wird in der Zieladresse das Sensornetz, also der Rechner und Port (Standardport) des Routers, sowie die Adresse des Sensorknotens im WSN codiert. Zusätzlich wird mit „WS“ der Webservice im Pfad der URL codiert. Neben der Nachrichten-ID und der Nutzlast (SOAP-Nachricht) wird noch in *replyTo* der Antwortendpunkt codiert, der die Rechneradresse des Absenders enthält und somit festlegt, dass die Antwort an diesen zurückgesendet werden soll. Der Pfad „/Reply“ dient lediglich der Verarbeitung der Nachricht in der Applikation. Auf Protokollebene wird diese Angabe nicht benötigt.

Die Nachricht wird nun entsprechend der Routingregeln an den Router gesendet. Dieser ersetzt die URLs durch die entsprechenden Aliase, die die *rcdDevAddr* vom Knoten (*to*) sowie die *rcdDevAddr* des Routers (*replyTo*) und je eine ID zur Repräsentation der jeweiligen URL enthalten. Nach der Kompression wird die Nachricht mit dem WSN-Kommunikationsprotokoll an den Knoten weitergeleitet. Auf dem Knoten verarbeitet der entsprechende Webservice die SOAP-Nachricht, generiert die SOAP-Antwort und sendet diese mit dem Alias aus *replyTo* als

neuem *to* und der Nachrichten-ID der Anfragenachricht als *relatesTo* an den Router. Dieser löst den Alias wieder auf und schickt per TCP oder UDP die Nachricht an den in *to* codierten Rechner, also den Absender der Anfragenachricht, zurück. Über den Wert in *messageID* der Anfrage und *relatesTo* der Antwort werden die asynchronen Nachrichten zu einer synchronen Kommunikation vereint.

4.5.5 Paketfragmentierung

Die maximale Größe einer Übertragungseinheit in Netzen mit beschränkten Ressourcen ist sehr gering. So legt der IEEE-802.15.4-Standard eine maximale Paketgröße von 127 Byte fest. Dieser Limitierung begegnet LTP durch die Möglichkeit zur Paketfragmentierung. Dazu wird für jedes Subnetz eine individuelle maximale Paketgröße s festgelegt. Pakete, die s überschreiten, werden fragmentiert. Dazu wird der Body eines LTP-Pakets p in verschiedene einzelne LTP-Pakete p_i aufgeteilt und einzeln ausgetauscht. Durch das Setzen des Paketfeldes *fragmentation* wird gekennzeichnet, dass p_i Fragmente von p sind. *fragmentNumber* gibt zusätzlich den Index eines Fragments an, während *isLastFragment* das Fragment mit dem höchsten Index kennzeichnet. Über das Feld *messageID* werden alle p_i p zugeordnet. Die Reihenfolge der Versendung und des Empfangs von Fragmenten ist irrelevant. Allerdings werden Fragmentverluste nicht von LTP behoben. Fehlen nach Ablauf einer Zeitspanne noch einzelne Fragmente, dann wird p insgesamt verworfen.

Die Fragmentierung wird in LTP in jedem Subnetz individuell realisiert. Für jedes RCN wird eine gesonderte maximale Übertragungsgröße s_{local} festgelegt. Bei der Weiterleitung von Fragmenten am Router wird p wieder defragmentiert und vor der Weiterleitung entsprechend der maximalen Paketgröße des nächsten Subnetzes s_{next} neu fragmentiert. Im Vergleich zu Fragmentierungsstrategien, die z. B. bei der Weiterleitung nur neu fragmentieren, wenn $s_{\text{local}} > s_{\text{next}}$ ist, wird ein erhöhter Fragmentierungsaufwand verursacht. Dieser Sachverhalt ist jedoch nicht problematisch, da die Router im Vergleich zu den Geräten im RCN relativ ressourcenstark sind. Der Vorteil, der sich aus dieser Strategie ergibt, ist, dass die maximale Paketgröße in jedem Teilnetz optimal ausgenutzt und so die notwendige Datenrate minimiert wird.

4.5.6 Implementierungen

Zum Nachweis der Realisierbarkeit des Konzepts wurden vom Autor dieser Arbeit drei Frameworks entwickelt, die eine LTP+SMC-Webservice-Kommunikation sowie eine allgemeine LTP-Kommunikation erlauben. Als erste Variante wurde ein eigenständiges Framework (LTP-Standalone) implementiert. Es ist in Java programmiert, kann aber mithilfe von IKVM.NET [57] auch für Micrsoft.NET-Umgebungen kompiliert werden und steht somit auf den beiden momentan wichtigsten Programmierplattformen der Enterprise-IT zur Verfügung. LTP-Standalone zielt auf den Einsatz auf PC- oder Serverhardware ab und realisiert sowohl eine LTP+SMC-Webservice-Kommunikation als auch eine allgemeine LTP-Kommunikation. Die Standalone-Version zeichnet sich vor allem durch einen schlanken Aufbau sowie ein sehr gutes Laufzeitverhalten aus.

```
1  package ws;
2
3  import javax.jws.*;
4
5  @WebService()
6  public class TemperatureWebService
7  {
8      @WebMethod(operationName = "addTemperatureValue")
9      public boolean addTemperatureValue(
10         @WebParam(name = "temperature") final int temperature,
11         @WebParam(name = "cargoID")     final int cargoID)
12     {
13         // Add application code here
14         return true;
15     }
16 }
```

Quelltext 4.1: Beispiel der Implementierung eines Webservices mit Axis2

Als zweite, ebenfalls für den Einsatz im LTP-Kern bestimmte Variante, wurde LTP+SMC als zusätzliches Transport-Binding in das Webservice-Framework Axis2 [8] integriert. Diese Implementierung weist zwar gegenüber der Standalone-Variante ein schlechteres Laufzeitverhalten auf. Allerdings bietet sie den großen Vorteil, dass die Bereitstellung und Nutzung eines Webservices unabhängig von einem konkreten Transport-Binding realisiert wird. Quelltext 4.1 zeigt beispielhaft die Implementierung eines Webservices in Axis2. Die Kennzeichnung einer einfachen Java-Klasse und ihres Inhalts mit verschiedenen Annotationen aus dem Namensraum „javax.jws.*“ reichen aus, um entsprechende Funktionalität als Webservice bereitzustellen. Auf diese Weise kann dieselbe Implementierung lediglich durch eine entsprechende Konfiguration des Frameworks gleichzeitig über alle oder ausgewählte, vom Framework unterstützte Transport-Bindings bereitgestellt werden. Ein LTP+SMC-Webservice kann so gleichzeitig auch über HTTP oder SMTP mit einer XML-basierten SOAP-Serialisierung bereitgestellt und damit die Interoperabilität erhöht werden. Gleiches gilt auch für die Dienstnutzung.

Neben diesen beiden für den LTP-Kern bzw. das Backend bestimmten Implementierungen wurde LTP auch für ressourcenbeschränkte Geräte implementiert. Diese Variante wurde mit den Programmiersprachen C und C++ realisiert und kann für alle Plattformen kompiliert werden, für die ein aktueller GCC-Compiler (*GNU Compiler Collection*, [56]) existiert. Getestet wurde sie auf den iSense-, Pacemate- und TelosB-Sensornetzplattformen sowie auf einem PDA (*Personal Digital Assistant*) mit *Windows Mobile* [134] als Betriebssystem. Da jede zusätzliche Funktionalität mit einem zusätzlichen Overhead verbunden ist, wurde das Framework sehr modular aufgebaut. Anwendungsentwickler können zur Kompilierzeit Funktionen wie die Ausgabe von Debug-Nachrichten, den Zugriff auf den IWS oder die Verwendung von Huffmancodierten Strings aktivieren und deaktivieren sowie eine der folgenden Konfigurationen auswählen, die jeweils mit einer statischen oder dynamischen Speicherverwaltung ausgeführt werden können:

- *Minimal*: stellt lediglich die minimal nötige Funktionalität bereit.
- *Callback Handler*: Anfrage- und Antwortnachrichten werden vom Framework automatisch in Beziehung gesetzt. Zusätzlich werden automatisch die entsprechend registrierten Methoden zur Verarbeitung der in Relation gesetzten Nachrichten auf Applikationsebene aufgerufen.

- *Fragmentation*: realisiert die automatische (De-)Fragmentierung von LTP-Paketen.
- *Callback Handler & Framentation*: aktiviert beide entsprechenden Konfigurationen.

Genau wie die Standalone-Version ermöglicht auch dieses Framework sowohl eine eigenständige LTP-Kommunikation als auch eine Webservice-Kommunikation über LTP+SMC als Transport-Bindung. Außerdem ermöglicht es die Integration von ergänzenden Komponenten z. B. zur Ausführung von Geschäftsprozessen oder zur Realisierung von Sicherheitsmechanismen.

Zusätzlich zu den Frameworks zur Bereitstellung und Nutzung von Webservices bzw. zur allgemeinen LTP-Kommunikation für alle Arten von Endgeräten wurde ein Router implementiert, der die RCNs mit dem LTP-Kern verbindet. Darüber hinaus wurden Werkzeuge entwickelt, mit denen die Router überwacht und einem Administrator der Zugriff auf Infrastrukturdienste ermöglicht werden.

4.6 Schnittstellenbeschreibung von LTP+SMC-Webservices

LTP erfüllt die Anforderungen, die SOAP an einen Nachrichtentransport stellt. Die SOAP-Nachrichten selbst können mit SMC effizient und gleichzeitig standardkonform zu SOAP 1.2 codiert werden. LTP und SMC lassen sich folglich zu einem SOAP-Transport-Binding kombinieren. Damit lassen sich alle Webservices, die dieses Transport-Binding verwenden, ohne jegliche Anpassung bestehender Standards vollständig formal beschreiben. LTP+SMC-Webservices werden mit WSDL 1.1 und ergänzenden Standards beschrieben. Quelltext 4.2 zeigt die für die Bindung eines Webservices an LTP+SMC wichtigen Ausschnitte aus einem WSDL-Dokument. Die Bindung einer abstrakten Webservice-Schnittstelle an ein Transport-Binding erfolgt im konkreten WSDL-Teil. Zur Definition werden verschiedene Elemente aus dem WSDL- sowie aus dem SOAP-Namensraum zur Erweiterung von WSDL um SOAP-spezifische Angaben (*WSDL 1.1 Binding Extension for SOAP 1.2*, [248]) verwendet. Das *binding*-Element aus WSDL umschließt diese Binding-Definition (Zeilen 10 bis 21).

Da LTP+SMC konform zu SOAP ist, unterscheidet es sich bzgl. der Binding-Definitionen auch nicht von anderen SOAP-Bindings. LTP+SMC tauscht SOAP-Nachrichten direkt als XML-Infosets analog zu den referenzierten XML-Datenstrukturen („document/literal“, Zeilen 11, 15 und 18) aus. Die Definition, wie der Transport realisiert wird, erfolgt bei allen SOAP-Bindings in WSDL über eine URI im *transport*-Attribut des *binding*-Elements aus dem SOAP-Namensraum (Zeile 11). Die URI `http://www.itm.uni-luebeck.de/users/glombitza/LTP+SMC` definiert, dass SOAP-Nachrichten mit SMC komprimiert bzw. serialisiert und mit LTP ausgetauscht werden.

Um Webservice-Operationen während der Kommunikation auflösen zu können, muss in LTP das *action*-Feld korrekt gesetzt werden. Die Zuordnung zwischen *action*-Feld und einer Operation bzw. einer zu einer Operation gehörenden Nachricht wird ebenfalls im WSDL-Dokument festgelegt und kann auf zwei verschiedene Arten erfolgen. Zum einen kann in der Binding-Definition das Attribut *soapAction* angegeben werden (Zeile 13). In diesem Fall muss ein LTP-Anfragepaket genau diesen Wert im *action*-Feld enthalten. Bei der LTP-Antwort ist dieses Feld leer. Eine Zuordnung erfolgt dann über die Korrelation von LTP-Anfrage und -Antwort. Die zweite Möglichkeit zur Auflösung von Operationen ist die

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <definitions name="DcNodeWS" targetNamespace="http://www.itm.uni-luebeck.de/
       users/glombitza/wsdl/l2d2/DcNodeWS" xmlns="http://schemas.xmlsoap.org/wsdl
       /" xmlns:ns="http://www.itm.uni-luebeck.de/users/glombitza/wsdl/l2d2/
       DcNodeWS" xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/" xmlns:
       wsam="http://www.w3.org/2007/02/addressing/metadata">
3    [...]
4    <portType name="DcNodeWSType">
5      <operation name="registerCargo">
6        <input name="input1" message="ns:registerCargo" wsam:Action="register"
             />
7        <output name="output1" message="ns:registerCargoResponse" wsam:Action="
             registerResponse"/>
8      </operation>
9    </portType>
10   <binding name="DcNodeWSBinding" type="ns:DcNodeWSType">
11     <soap12:binding style="document" transport="http://www.itm.uni-luebeck.de/
         users/glombitza/LTP+SMC"/>
12       <operation name="registerCargo">
13         <soap12:operation soapAction="register"/>
14         <input name="input1">
15           <soap12:body use="literal" />
16         </input>
17         <output name="output1">
18           <soap12:body use="literal" />
19         </output>
20       </operation>
21   </binding>
22   <service name="DcNodeWSService">
23     <port name="DcNodeWSPort" binding="ns:DcNodeWSBinding">
24       <soap12:address location="ltp://123@wsn1.itm.uni-luebeck.de/DcNWS"/>
25     </port>
26   </service>
27 </definitions>
```

Quelltext 4.2: Beispiel eines WSDL-Dokuments zur Beschreibung eines LTP+SMC-Webservices

Verwendung des WSAM-Standards (*Web Services Addressing 1.0 Metadata*, [257]) oder der Vorgängerspezifikation WSAW (*WS-Addressing 1.0 WSDL Binding*, [247]). Diese erweitern WSDL um WS-Addressing-spezifische Ausdrücke. Werden im abstrakten Teil der WSDL die Eingabe-, Ausgabe- und Fehlernachrichten der Operationen eindeutig mit dem entsprechenden *Action*-Parameter benannt (Zeilen 6 und 7), dann können auch diese Werte in *action* von LTP stehen und zur Operationsauflösung dienen. Auch in diesem Fall kann bei optional nicht gesetzten *action*-Feld von LTP-Antwort- oder -Fehlernachrichten eine Operationsauflösung ebenfalls über die Paketkorrelation erfolgen.

Obwohl auch WSAM bzw. WSAW bei der Beschreibung von LTP+SMC-Webservices eingesetzt werden kann, wird WS-Addressing nicht bei einer LTP-Kommunikation verwendet. LTP unterstützt selbst sowohl eine synchrone als auch eine asynchrone Kommunikation und realisiert direkt mit *one-way*, *request-response*, *solicit-response* und *notification* alle vier in WSDL definierten MEPs.

Soll im konkreten WSDL-Teil neben dem Transport-Binding auch ein konkreter Webservice-Endpunkt festgelegt werden, dann kann dieses über das *address*-Attribut aus dem SOAP-Namensraum (Zeile 24) erfolgen. Über das Attribut *location* wird der Endpunkt über eine LTP-URL eindeutig festgelegt. Dabei kann eine URL Unicast-, Multicast- sowie Broadcast-

Plattform	Protokolllogik				Paketserialisierung				IWS
	min.	CBH	Frag.	CBH & Frag.	ASCII stat.	ASCII dyn.	Huffman stat.	Huffman dyn.	
Jennic	2.476	4.300	6.864	8.688	2.689	2.829	3.583	3.723	5.309
Pacemate	2.888	5.100	7.992	10.204	2.196	2.312	3.016	3.132	3.924
Win Mobile	2.664	5.364	8.352	11.052	2.212	2.324	3.044	3.144	4.908
TelosB	1.946	2.868	4.412	5.334	1.870	1.950	2.436	2.516	3.498

Tabelle 4.1: Übersicht der Codegrößen der LTP-Implementierung für ressourcenbeschränkte Geräte (in Byte)

Endpunkte beschreiben.

4.7 Analyse der Leistungsmerkmale von LTP+SMC

In diesem Abschnitt werden die Ergebnisse der Evaluation von LTP sowie LTP+SMC präsentiert und bewertet. Viele der Vorteile der Realisierung einer ganzheitlichen, transparenten Webservice-Kommunikation in RCN und Backend wie die Erhöhung der Entwicklungsgeschwindigkeit, die schnelle, einfache und flexible Anpassung der Anwendungen und Prozesse sowie die nahtlose Integration von Enterprise-IT-Systemen und RCNs sind schwer quantifizierbar. Sie werden in Abschnitt 5.6.6 als Teil der Evaluation des ganzheitlichen Geschäftsprozessmanagements separat betrachtet. Die Evaluation in diesem Abschnitt beschränkt sich mit der Untersuchung der Code- und Nachrichtengröße, der Berechnungskomplexität sowie des Arbeitsspeicherverbrauchs auf die direkt quantifizierbaren Eigenschaften von LTP und LTP+SMC.

4.7.1 Evaluation der Codegröße

Tabelle 4.1 zeigt die Codegröße der LTP-Implementierung für ressourcenbeschränkte Geräte. Der C/C++-Code wurde für die Pacemate-, Jennic- und TelosB-Sensornetzplattform (s. Abschnitt 2.4.3) sowie für das Windows-Mobile-Betriebssystem (32-Bit-ARM-CPU) kompiliert. Die LTP-Implementierung besteht mit der Protokolllogik und der Paketserialisierung aus mindestens zwei Komponenten, die jeweils in unterschiedlichen Konfigurationen verwendet werden können. Die Protokolllogikkomponente kann mit der minimal notwendigen Konfiguration (min.), dem Callback Handler (CBH), der Fragmentierung (Frag.) sowie der gleichzeitigen Verwendung von CBH und Fragmentierung (CBH & Frag.) in den vier in Abschnitt 4.5.6 beschriebenen Konfigurationsausprägungen verwendet werden. Zusätzlich zur Protokolllogik müssen LTP-Pakete (de-)serialisiert werden. Mit Serialisierungscode für ASCII- sowie Huffman-codierte Strings werden zwei Konfigurationen unterschieden, die beide sowohl mit einer statischen (stat.) als auch mit einer dynamischen (dyn.) Speicherverwaltung verwendet werden können. Soll optional mit einem IWS interagiert werden, wird zusätzlich entsprechender Code benötigt.

Mit einer minimalen Größe zwischen 1.946 Byte und 2.888 Byte sowie einer maximalen Größe zwischen 5.334 Byte und 11.052 Byte ist die Codegröße der Protokolllogik für alle Plattformen

und Konfigurationen sehr klein. Auch der Code für die Paketserialisierung hat eine sehr geringe Größe. Je nach Konfiguration und Plattform entstehen zusätzlich zwischen 1.870 Byte und 3.723 Byte an Codegröße. Der Code (Logik und SMC-Serialisierung) zur Interaktion mit dem IWS benötigt zwischen 3.498 Byte und 5.309 Byte zusätzlichen Code.

Die Evaluationsergebnisse zeigen, dass mit einer minimalen Codegröße von 3.816 Byte und einer maximalen Codegröße von 19.104 Byte LTP für die Beispielplattformen sehr wenig Code benötigt. Damit erfüllt LTP in Bezug auf die Codegröße alle Anforderungen, um auf ressourcenbeschränkten Geräten verwendet werden zu können. Die Codegröße ist deutlich geringer als die von CoAP, das (exklusive DNS-Auflösung und inklusive der sonstigen unterliegenden Protokolle) für die Jennic-Plattform eine Größe von 62 KB aufweist.

Auch der Overhead von SMC ist sehr gering. Zur Evaluation von SMC wurde zusätzlich zum IWS ein einfacher Additions-Webservice implementiert, der zwei ganze Zahlen als Eingabe entgegennimmt und eine ganze Zahl als Rückgabe übermittelt. Mit einer Codegröße von 1.624 Byte (Windows Mobile), 1.440 Byte (Pacemate), 1.728 Byte (Jennic) und 1.174 Byte (TelosB) ist der Code zur (De-)Serialisierung der komprimierten SOAP-Nachrichten und zur Repräsentation der Daten- und Nachrichtentypen sehr klein. Für eine ausführlichere Evaluation der SMC-Codegrößen im Kontext eines Logistikanwendungsfalls sei auf Abschnitt 5.6.2 verwiesen.

4.7.2 Evaluation des Arbeitsspeicherverbrauchs

Neben der Codegröße wurde auch der Arbeitsspeicherverbrauch von LTP gemessen. Als Ausführungsplattform für die Testreihen diente das im Rahmen dieser Arbeit auf allen WSN-Plattformen eingesetzte iSense-Betriebssystem in einer 32-Bit-Version. Gemessen wurde der Verbrauch in den drei Konfigurationen „minimal“, „Callback Handler“ und „Fragmentierung“. Im minimalen Szenario beinhaltet die Messung die Verarbeitung einer Sequenz aus einem eingehenden und einem ausgehenden LTP-Paket. Im Szenario „Fragmentierung“ wurden beide Pakete auf jeweils drei Fragmente aufgeteilt, während im Szenario „Callback Handler“ ein LTP-Paket verschickt und eine dazugehörige Antwort empfangen wurde. Der *body* aller LTP-Pakete bzw. -Fragmente hat eine Länge von 10 Byte, während die URLs jeweils 30 Zeichen lang sind. Alle drei Szenarien wurden mit einer vollständigen Header-Kompression sowie mit URL-codierten Endpunkten gemessen. Bei der Verwendung von URLs wird zwischen ASCII- und Huffman-codierten URLs unterschieden. Zusätzlich wird unterschieden, ob die Repräsentation eines LTP-Pakets im RAM einer statischen oder dynamischen Speicherverwaltungsstrategie folgt.

Abbildung 4.9 zeigt den maximalen Speicherverbrauch für die genannten Testreihen. Darin ist der minimale Speicherverbrauch, also der Verbrauch nach der Initialisierung von LTP, enthalten. Er beträgt je nach Konfiguration 40 Byte (min.), 52 Byte (Frag.) oder 56 Byte (CBH). Diese Werte sind weder von der Verwendung der Header-Kompression noch von der String-Codierung oder der Speicherverwaltungsstrategie abhängig.

Die von der LTP-Implementierung ermöglichte dynamische und statische Speicherverwaltung betrifft die Repräsentation eines LTP-Pakets im Arbeitsspeicher. Im statischen Szenario werden statisch Strukturen zur Speicherung aller möglichen LTP-Paketinhalte unabhängig

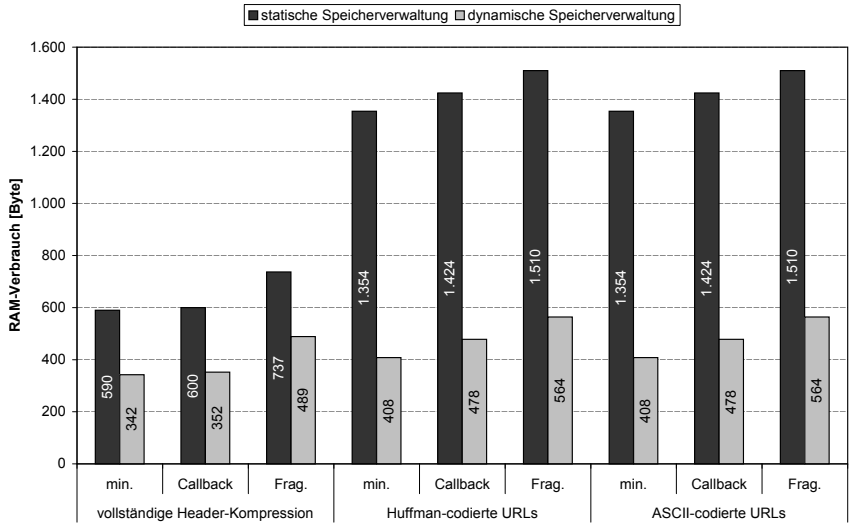


Abbildung 4.9: Maximaler Arbeitsspeicherverbrauch von LTP

von der tatsächlichen Existenz und Größe im RAM angelegt. Bei der dynamischen Speicherverwaltung werden diese Felder und Größen an die tatsächliche Existenz und Größe der entsprechenden Felder im LTP-Paket angepasst. Dadurch lässt sich der Speicherverbrauch in dynamischen Szenarien erheblich gegenüber den entsprechenden statischen Messreihen reduzieren. Bei der Verwendung einer vollständigen Header-Kompression betragen die Einsparungen zwischen 34 % und 42 %, während beim Verzicht auf Aliase Einsparungen zwischen 63 % und 70 % erreicht werden können. Das größere Einsparpotenzial beim Verzicht auf diese Kompressionstechnik lässt sich auf den Bedarf der Speicherung von potenziell möglichen Strings zurückführen. Während *to*, *from*, *replyTo*, *faultTo*, *action*, *messageID* und *relatesTo* gesetzt und alle als Strings repräsentiert sein können, werden in der Regel nur wenige dieser Felder gleichzeitig verwendet. Da im statischen Szenario auch beim Verzicht auf Strings, also beim Einsatz einer vollständigen Header-Kompression, weitere Paketinhalte in statischen Puffern abgespeichert werden, ist der Verbrauch in den entsprechenden statischen Szenarien immer noch höher als bei der Verwendung von URLs in dynamischen Szenarien. So konnte für den CBH zwar durch die Alias-Kompression eine RAM-Verbrauchsreduktion von 1.424 Byte auf 600 Byte erreicht werden. Diese liegt aber immer noch über dem Verbrauch des CBH bei der Verwendung von URLs im dynamischen Szenario (478 Byte).

Weiterhin ist bei der Auswertung der Messreihen auffällig, dass die Ergebnisse in den Szenarien ohne URL-Kompression zwischen ASCII- und Huffman-Codierung identisch sind. Das ist darauf zurückzuführen, dass der maximale Speicherverbrauch aufgrund der Charakteristik des Verbrauchs über die Zeit vor allem von der Repräsentation eines LTP-Pakets im RAM abhängt. Diese unterscheidet sich zwischen dynamischer und statischer Speicherverwaltung sowie zwischen ein- und ausgeschalteter URL-Kompression. Die String-Codierung hat jedoch keinen Einfluss auf den RAM-Verbrauch, da die Strings im RAM immer als 8-Bit-Zeichen codiert werden.

Abschließend kann LTP bzgl. des Arbeitsspeicherverbrauchs als geeignet für ressourcenbeschränkte Geräte bewertet werden. Mit einem Verbrauch zwischen 342 Byte und 564 Byte bei dynamischer und zwischen 590 Byte und 1.510 Byte bei statischer Speicherverwaltung kann

es problemlos auf typischen WSN-Plattformen verwendet werden. Durch die Anwendung eines dynamischen Speichermanagements auch bei der Protokolllogik und nicht nur bei der Repräsentation des LTP-Pakets im RAM könnte der Speicherverbrauch noch weiter reduziert werden. Allerdings ist in diesem Zusammenhang eine Bewertung der Zielplattform bzgl. der Gefahr der Fragmentierung des dortigen Arbeitsspeichers, die durch ein dynamisches Speichermanagement verursacht werden kann, vorzunehmen.

Auch der Arbeitsspeicherverbrauch von SMC ist sehr gering. Bei der serverseitigen Realisierung des Additions-Webservices beträgt der maximale SMC-RAM-Verbrauch 52 Byte. Die Verarbeitung findet zu einem Zeitpunkt statt, an dem die Speicherallokation von LTP (unabhängig von der Konfiguration) 163 Byte unter dem maximalen Verbrauch liegt. Aus diesem Grund hat der Additions-Webservice keinen Einfluss auf den Gesamtspeicherverbrauch von LTP+SMC. Allgemein wird der Einfluss von SMC auf den Gesamtverbrauch in Byte in Formel 4.1 beschrieben. Sei s der RAM-Verbrauch von SMC, l der von LTP, dann ist r der Gesamtverbrauch.

$$r = l + \max(s, 163) - 163 \tag{4.1}$$

4.7.3 Evaluation der Nachrichtengröße

Eine weitere wichtige Anforderung bei einer Webservice-Kommunikation in RCNs ist die Nachrichtengröße des Transportprotokolls sowie die Größe der SOAP-Nachrichten. Der Protokoll-Overhead von LTP lässt sich mit den Formeln 4.2 bis 4.7 berechnen. Sei b die Body-Größe, h die Header-Größe (beides in Bit) und n die Anzahl der Fragmente, in die das LTP-Paket bei einer maximalen Paketgröße von m (in Bit) aufgeteilt werden muss (s. Gleichung 4.3), dann ist c das Gesamtnachrichtenaufkommen in Byte für ein (evtl. fragmentiertes) LTP-Paket (s. Gleichung 4.2).

$$c = \left\lceil \frac{b}{8} \right\rceil + \left\lceil \frac{h}{8} \right\rceil \cdot n + \begin{cases} n & n > 1 \\ 0 & \text{else} \end{cases} \tag{4.2}$$

$$n = \begin{cases} 1 & h + b \leq m \\ \left\lceil -\frac{b}{8+h-m} \right\rceil & \text{else} \end{cases} \tag{4.3}$$

Die Größe des Headers[3] lässt sich mit der Gleichung 4.4 ermitteln. Dabei gibt t die Länge der URL des *to*-Felds und a die Länge des Strings des *action*-Felds an. A und M sind Multimengen. A enthält die Längen der URLs von *from*, *replyTo* und *faultTo* während M die Längen der Strings von *messageID* und *relatesTo* enthält. Werden diese Felder nicht gesetzt, ist ihre Länge nicht in A bzw. M enthalten. Werden *action*, *from*, *replyTo*, *faultTo*, *relatesTo* oder *messageID* komprimiert, dann existiert für das entsprechende Feld in A bzw. M ein Eintrag mit dem Wert 0. Die Länge aller Strings wird in Zeichen angegeben. Allerdings wird bei der

[3] In Gleichung 4.4 ist zwar die Größe des Feldes zur Kennzeichnung einer Fragmentierung, nicht aber die von *FragmentPropertiesType* enthalten, da diese bereits in Gleichung 4.2 berücksichtigt wird.

Serialisierung von LTP-URLs „ltp://“ aus Optimierungsgründen nicht mit berücksichtigt. Zur korrekten Ermittlung der URL-Länge dürfen diese Zeichen nicht mitgezählt werden.

$$\begin{aligned} h \;=\; & \text{addressTypeSize}(t) + \textstyle\sum_{i \in A} \text{addressTypeSize}(i) \\ & + \textstyle\sum_{i \in M} \text{messageIdTypeSize}(i) + 21 + \begin{cases} 1 & a = 0 \\ 7 + \text{stringSize}(a) & \text{else} \end{cases} \end{aligned} \tag{4.4}$$

Mit den Funktionen 4.5 und 4.6 lassen sich die Größen der Felder vom Typ *AddressType* und *MessageIDType* berechnen. s gibt dabei jeweils die Länge des Strings in Zeichen an. $s = 0$ wird als Verwendung der jeweiligen Header-Kompression interpretiert. Die Konstante r gibt die Größe von *rcdDevAddr* in Bit an.

$$\text{addressTypeSize}(s) := 1 + \begin{cases} 8 + r & s = 0 \\ 8 + \text{stringSize}(s) & \text{else} \end{cases} \tag{4.5}$$

$$\text{messageIdTypeSize}(s) := 1 + \begin{cases} 32 & s = 0 \\ 6 + \text{stringSize}(s) & \text{else} \end{cases} \tag{4.6}$$

Die Funktion 4.7 beschreibt die Größe eines serialisierten Strings in Abhängigkeit der String-Länge s. Für eine ASCII-Codierung werden 7 Bit pro Zeichen benötigt. Bei Huffman-codierten Strings ist die Länge der Codierung von den tatsächlich codierten Zeichen abhängig. Um die Länge eines Huffman-codierten Strings lediglich in Abhängigkeit der String-Länge abschätzen zu können, wurde die in Funktion 4.7 aufgeführte Schätzfunktion ermittelt. Der in LTP verwendete Huffman-Baum ist für URIs optimiert. Er entstand aus der Analyse von 25.000 zufällig aus dem Internet ermittelten URIs. Weitere 25.000 zufällig aus dem Internet geladene URIs dienten zur Ermittlung einer Regressionsgeraden, die in Abhängigkeit der Länge eines Strings in Zeichen die Länge dessen Huffman-Codierung in Bit abschätzt, ohne jedes Zeichen einzeln betrachten zu müssen.

$$\text{stringSize}(s) := \begin{cases} 5,04 \cdot s + 7,52 & \text{HUFFMAN} \\ 7 \cdot s & \text{ASCII} \end{cases} \tag{4.7}$$

Der von LTP erreichte Funktionsumfang bzgl. der Möglichkeiten bei der Adressierung kann auch mit den im Enterprise-IT-Umfeld verwendeten Transportprotokollen nur durch die Verwendung von WS-Addressing erreicht werden. Aus diesem Grund wird LTP zur Analyse des Protokoll-Overheads mit WS-Addressing verglichen. Der Overhead der unterhalb von LTP und WS-Addressing liegenden Protokolle wird in dieser Analyse nicht berücksichtigt. Allerdings ist der Overhead eines einfachen WSN-Protokolls unterhalb von LTP (z. B. *Tree Routing*) wesentlich geringer als z. B. der von HTTP unterhalb von WS-Addressing. Zur Evaluation wurde eine Anfragenachricht mit einer 38 Zeichen langen *to*- und einer 43 Zeichen langen *replyTo*-URL, einer 10 Ziffern langen Zahl als *messageID* und einem 13 Zeichen langen *action*-Feld erzeugt. Zusätzlich wurde eine entsprechende Antwortnachricht generiert, deren *action*-Feld 21 Zeichen lang ist. Alle Pakete enthalten keine Nutzlast.

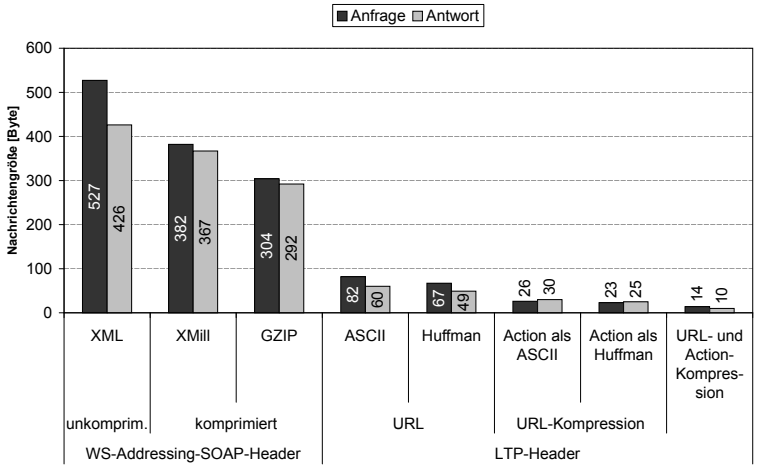


Abbildung 4.10: Vergleich der LTP-Paketgrößen

Abbildung 4.10 zeigt die entsprechenden Nachrichtengrößen. Unkomprimiertes XML repräsentiert die Größe von WS-Addressing im SOAP-Header bei der Verwendung in herkömmlichen Webservices. XMill und GZIP dienen als Bezugspunkt zum Potenzial der Anwendung verschiedener Kompressionstechniken auf WS-Addressing. Mit 426 Byte (unkomprimiert), 367 Byte (XMill) und 292 Byte (GZIP) für das Antwortpaket ist WS-Addressing zu ressourcenverbrauchend für den Einsatz in WSNs. Zusätzlich zur geringen Kompressionsrate von nur 14 % bzw. 31 % spricht auch die Nachrichtenverarbeitung gegen den Einsatz von WS-Addressing. Entsprechende Nachrichten müssten auf den Sensorknoten dekomprimiert und Roh-XML verarbeitet werden. Mit 60 Byte (ASCII) bzw. 49 Byte (Huffman) erreicht LTP ohne Verwendung der Alias-Kompressionstechnik eine Kompressionsrate von 86 % bzw. 88 %. Mit der Anwendung aller Kompressionstechniken kann die Kompressionsrate weiter auf 98 % im Vergleich zu WS-Addressing und 83 % gegenüber unkomprimiertem LTP gesteigert werden. Mit einem absoluten Overhead zwischen 82 Byte und 10 Byte für die Testpakete ist der Einsatz von LTP in RCNs problemlos möglich.

Zusätzlich zur Nachrichtengröße von LTP wurden die Größen der SOAP-Nachrichten des IWS und Additions-Webservices evaluiert. Zum Inhalt der IWS-Nachrichten sei auf Abbildung 4.7 auf Seite 80 verwiesen. Die Länge der URL, für die ein Mapping erstellt und abgefragt wird, ist 38 Zeichen. Abbildung 4.11 vergleicht die Größen von unkomprimierten SOAP-Nachrichten sowie die Nachrichtengrößen bei der Verwendung der Kompressionstechniken XMill, GZIP und SMC. SMC erreicht im Vergleich zu den anderen Serialisierungen eine Kompressionsrate zwischen 94 % und 99 %. Mit absoluten Größen zwischen 3 Byte und 27 Byte können mit SMC komprimierte SOAP-Nachrichten problemlos mit LTP in RCNs übertragen werden. Weitere Evaluationsergebnisse zu Nachrichtengrößen von SMC werden in Abschnitt 5.6.2 präsentiert.

Abschließend kann LTP+SMC als sehr effizient bzgl. des Nachrichten-Overheads angesehen werden. Sowohl der durch das Transportprotokoll LTP verursachte Overhead als auch die Nachrichtengrößen der komprimierten SOAP-Nachrichten sind extrem gering. LTP+SMC kann damit problemlos in RCNs verwendet werden. Zusätzlich stellt es aber auch in Umgebungen mit ausschließlich ressourcenstarker Hardware eine Alternative dar. Gerade in Ländern mit einer schlechten Kommunikationsinfrastruktur, in mobilen Szenarien oder bei einem

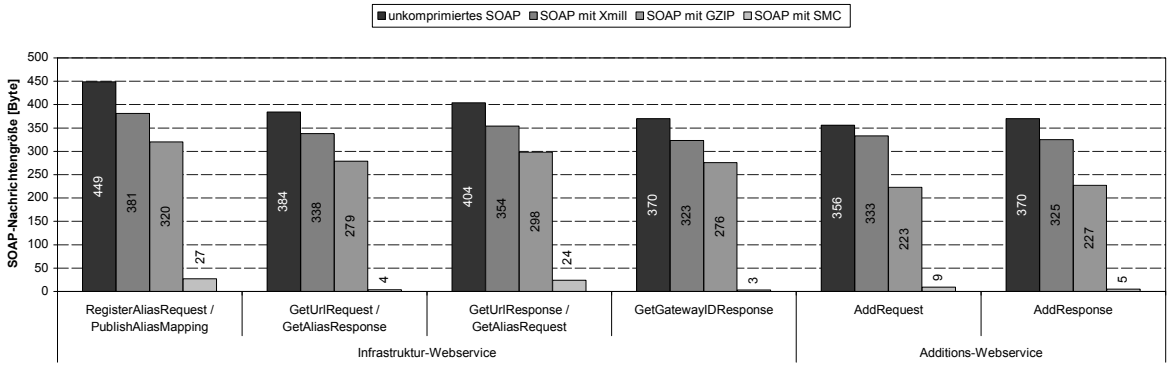


Abbildung 4.11: Größe der SOAP-Nachrichten des Infrastruktur- und Additions-Webservices

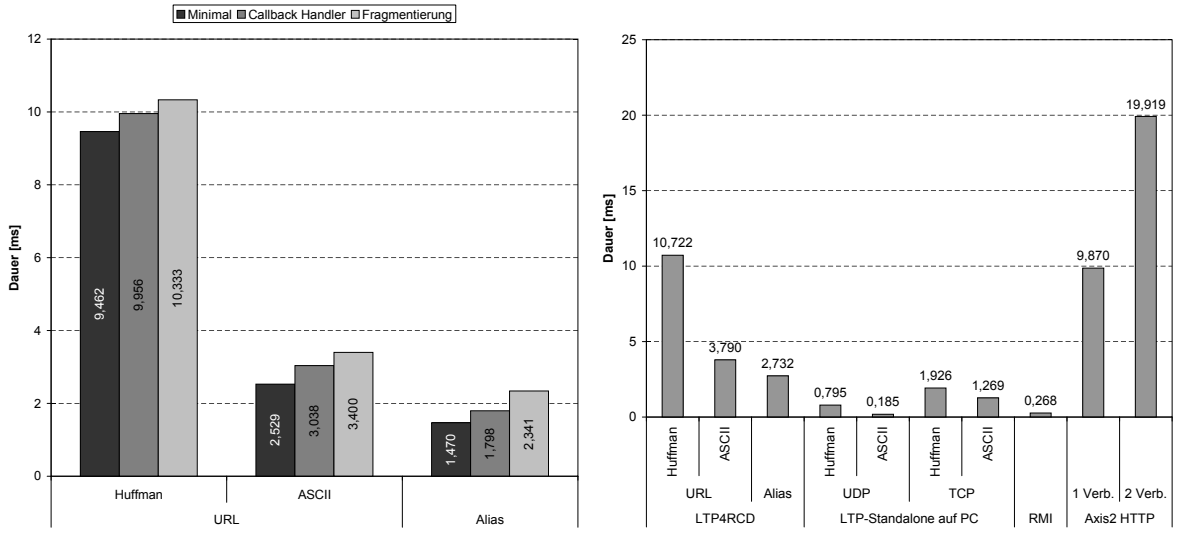
volumenabhängigen Nutzungsentgelt für Netzzugänge weist LTP+SMC Vorteile gegenüber etablierten Transport-Bindings auf.

Im Vergleich zu LTP+SMC ist der Overhead von CoAP wesentlich höher. Um z. B. analog zu *RegisterAlias* eine URL an einen CoAP-Dienst (`coap://wsn01.itm.uni-luebeck.de/Infrastructure/WebService`) zu übertragen, entsteht ein gesamter Overhead von 108 Byte. Die zu übertragende Nutzlast wird in diesem Fall manuell codiert. Bei einer zu übertragenen URL mit einer Länge von 38 Zeichen entsteht dabei ein Overhead von 38 Byte. Das CoAP Paket selbst verursacht weitere 32 Byte. Anders als LTP, das unabhängig von den unterliegenden Protokollen ist, ist CoAP eng mit UDP und IP verzahnt. Der Absender, Empfänger sowie die Größe der Nutzlast werden ausschließlich in UDP und IP codiert, sodass ein weiterer Overhead von 38 Byte berücksichtigt werden muss. LTP+SMC erzeugt für dieselbe Nachricht insgesamt lediglich einen Overhead von 33 Byte (Nachricht und Transport bei vollständiger Header-Kompression).

4.7.4 Evaluation des Laufzeitverhaltens

Als letzte Eigenschaft von LTP und LTP+SMC wurde das Laufzeitverhalten evaluiert. Die LTP-Laufzeitmessungen wurden auf der Pacemate-Sensornetzplattform durchgeführt. Dazu wurden dieselben Pakete wie bei der Evaluation des Arbeitsspeichers versendet. Gemessen wurde die Umlaufzeit von der Versendung des Anfragepakets durch den Absender bis zum Empfang und der vollständigen Verarbeitung der Antwortnachricht, gemittelt über 10.000 Iterationen. Um unkontrollierbare Störeinflüsse einer drahtlosen Kommunikation zu vermeiden, befinden sich beide LTP-Endpunkte auf demselben Sensorknoten.

Abbildung 4.12a fasst die Ergebnisse zusammen. Signifikant ist der große Overhead der Huffman-Codierung von Strings. Allgemein geht mit einer Huffman-Codierung ein Geschwindigkeitsverlust einher. Im Rahmen dieser Arbeit wurde bewusst eine um ca. 32 % langsamere Implementierung der Huffman-Codierung eingesetzt, um im Gegenzug die Codegröße um 28 % reduzieren zu können. Bei der Verwendung von ASCII-codierten Strings wird eine Reduktion der Umlaufdauer um 67 % bis 73 % je nach Messszenario erreicht. Bei einer vollständigen Header-Kompression kann die Laufzeit weiter reduziert werden. Gegenüber unkomprimierten Paketen mit ASCII-codierten Strings kann eine weitere Zeitersparnis von 31 % bis 42 % erreicht werden. Die Verwendung einer Header-Kompression wirkt sich nicht nur positiv auf



(a) Laufzeitverhalten von LTP

(b) Laufzeitverhalten von LTP+SMC

Abbildung 4.12: Analyse des Laufzeitverhaltens von LTP+SMC

die Nachrichtengröße, sondern auch auf das Laufzeitverhalten aus. Mit absoluten Umlaufzeiten zwischen 1,470 ms und 2,341 ms für vollständig komprimierte LTP-Pakete, 2,529 ms und 3,400 ms für ASCII-codierte URLs sowie zwischen 9,462 ms und 10,333 ms für Huffman-codierte URLs weist LTP ein sehr gutes Laufzeitverhalten auf und kann problemlos auf Sensorknoten und anderen ressourcenbeschränkten Geräten eingesetzt werden.

Zusätzlich zur Verarbeitungsgeschwindigkeit von LTP wurde die Dauer eines Webservice-Aufrufs mit LTP+SMC evaluiert. Dazu wurde die Umlaufzeit des Additions-Webservices gemessen. Als Vergleich dient eine Implementierung desselben Webservice mit HTTP+SOAP (XML-serialisiert) als Transport-Binding sowie eine Implementierung derselben Funktionalität mit Java RMI. LTP+SMC wurde sowohl auf den Pacemate-Sensorknoten als auch auf Basis der Standalone-Implementierung auf einem PC[4] ausgeführt. Die HTTP+SOAP-Variante wurde mit Axis2 implementiert und genau wie die RMI-Variante auf dem PC ausgeführt. Wie bei der Evaluation von LTP ohne SMC wurden auch bei dieser Evaluation Dienstanbieter und -konsument jeweils auf demselben PC bzw. Sensorknoten ausgeführt.

Abbildung 4.12b fasst die Evaluationsergebnisse zusammen. LTP+SMC weist auf einem PC ein Laufzeitverhalten auf, das um ein Vielfaches besser ist als das von HTTP+SOAP. Mit UDP als unter LTP liegenden Transportschicht benötigt LTP+SMC gerade 0,185 ms (ASCII-codierte URLs) bzw. 0,795 ms (Huffman). Damit kann LTP+SMC sogar schneller als RMI (0,268 ms) sein. Aber auch mit TCP als unterliegendem Transportprotokoll ist LTP+SMC mit beiden String-Codierungen noch um ein Vielfaches schneller als HTTP+SOAP. Im besten Fall (9,870 ms) wird bei HTTP+SOAP für eine SOAP-Anfrage und -Antwort dieselbe Verbindung zum Nachrichtenaustausch verwendet. Da LTP aber auch die Fähigkeit besitzt, Antworten

[4] Athlon 2,17 GHz CPU mit 1024 MB RAM, auf dem ein 32-Bit Debian Linux (Kernel 2.6.26-2-686) im Einzelbenutzermodus ohne weitere gestartete Prozesse lief. Als Java-Laufzeitumgebung wurde OpenJDK (1.6.0 0-b11) [205] verwendet. Als Webcontainer diente Apache Tomcat (5.5.27) [9].

an vom Absender abweichende Endpunkte zu versenden, wurde dieser Fall auch für HTTP+SOAP gemessen, indem mithilfe von WS-Addressing je eine Verbindung für die Anfrage und Antwort verwendet wurde. Mit 19,919 ms weist HTTP+SOAP in diesem Fall ein noch mal deutlich schlechteres Laufzeitverhalten auf.

Mit 10,722 ms mit Huffman-codierten URLs ist LTP+SMC auf Sensorknoten noch ähnlich schnell wie HTTP+SOAP im besten Fall auf einem PC. Bei der Verwendung von ASCII-codierten URLs (3,790 ms) oder einer vollständigen Header-Kompression (2,732 ms) lässt sich das Laufzeitverhalten von LTP+SMC weiter verbessern. Es ist dann auf einem Knoten sogar deutlich besser als das von HTTP+SOAP auf einem leistungsstarken PC. Dieses ist hauptsächlich auf eine effizientere und schnellere SOAP-Serialisierung sowie auf ein besseres Laufzeitverhalten von LTP gegenüber HTTP, zum Teil aber auch auf die Tatsache zurückzuführen, dass bei LTP+SMC auf den Sensorknoten keine TCP-Verbindung aufgebaut werden muss.

Zusammenfassend kann LTP+SMC bzgl. der Geschwindigkeit als geeignet für den Einsatz auf ressourcenbeschränkten Geräten angesehen werden. Des Weiteren ist es deutlich schneller als HTTP+SOAP. Damit kann es auch für die Verwendung von Webservices in Umgebungen, in denen Geschwindigkeit das entscheidende Kriterium ist, eingesetzt werden.

4.8 Zusammenfassung

In diesem Kapitel wurde mit LTP+SMC das vom Autor dieser Arbeit entwickelte Webservice-Transport-Binding beschrieben und evaluiert. Mit der durch LTP realisierten ganzheitlichen, transparenten Ende-zu-Ende-Kommunikation in Enterprise-IT-Systemen und Sensornetzen sowie in anderen Netzen mit beschränkten Ressourcen wird eine Transparenz auf der Nachrichtenebene erreicht. Darauf aufbauend wird mit der Kombination von LTP und SMC ein transparenter Webservice-Zugriff im Gesamtsystem und somit eine vollständige Verteilungstransparenz erreicht.

Zusätzlich zu diesen funktionalen Eigenschaften berücksichtigt LTP+SMC die Besonderheiten von ressourcenbeschränkten Netzen. Trotz der ganzheitlichen und damit plattformunabhängigen Kommunikation mit LTP lässt sich das Protokoll sehr feingranular an die plattformspezifischen Besonderheiten der verschiedenen Subnetze anpassen. LTP+SMC weist des Weiteren eine ausgezeichnete Effizienz auf. So sind die Evaluationsergebnisse dieses Bindings bei Weitem besser als die von etablierten Webservice-Standards. Dabei stellt der absolute Ressourcenverbrauch von LTP+SMC kein Problem für dessen Einsatz in ressourcenbeschränkten Umgebungen wie drahtlosen Sensornetzen dar.

Als SOAP-Binding ist LTP+SMC vollständig konform zur WS-*-Architektur und insbesondere zu WSDL. Damit kann es direkt als Kommunikationsbasis für beliebige auf WSDL und SOAP aufbauende Webservice-Technologien dienen. Das ermöglicht insbesondere die Verwendung von LTP+SMC für eine Geschäftsprozessrealisierung mit BPEL oder vergleichbaren Realisierungssprachen.

5 Ein Workflowmanagementsystem für ein ganzheitliches Geschäftsprozessmanagement

Das Ziel dieser Arbeit ist die Realisierung eines ganzheitlichen Geschäftsprozessmanagements in Sensornetzen und Backend-Systemen. In Abschnitt 3.4 wird das vom Autor dieser Arbeit entwickelte Konzept beschrieben, mit dem dieses Ziel erreicht wird. Es sieht als Systemarchitektur eine SOA vor, die mit Webservices als Umsetzungstechnologie transparent im Gesamtsystem realisiert wird. Darauf aufbauend wird ein Geschäftsprozessmanagement verwendet, das eine Modellierung, Ausführung (inkl. Simulation) und Überwachung sowie Optimierung von Geschäftsprozessen aus einer fachlichen Perspektive durch Fachpersonal ermöglicht. Entsprechende durch die Aggregation von Diensten modellierte Prozessdefinitionen können ohne weitere Umsetzung durch IT- oder Sensornetzexperten direkt sowohl auf Enterprise-IT-Servern als auch auf Sensorknoten ausgeführt werden.

Mit dem in Kapitel 4 präsentierten Transport-Binding kann eine transparente, ganzheitliche Webservice-Kommunikation realisiert und die für ein Geschäftsprozessmanagement notwendige Systemflexibilität durch die Realisierung einer SOA erreicht werden. Allerdings müssen durch die direkte Verwendung der in Kapitel 4 präsentierten Webservice-Lösung Dienste und Dienstaggregationen unter der Verwendung der Programmiersprachen C, C++ sowie Java implementiert werden. Wie in Abschnitt 3.4.4 diskutiert, kann diese Art der Prozessumsetzung für Basisdienste und einfache zusammengesetzte Dienste geeignet sein. Für komplexe zusammengesetzte Dienste und Prozessservices sieht das vom Autor dieser Arbeit entwickelte Konzept jedoch ausschließlich die Verwendung von domänenspezifischen Sprachen für eine grafische Geschäftsprozessmodellierung durch eine deklarative Dienstaggregation vor.

Die Prozessrealisierung stellt die entscheidende Herausforderung bei der Umsetzung eines Geschäftsprozessmanagements im Kontext von Sensornetzen dar und stand damit auch im Zentrum der Forschung dieses Kapitels. Dazu werden mit BPEL (*Business Process Execution Language*) und SM4RCD (*State Machine for Resource Constrained Devices*) zwei Sprachen zur domänenspezifischen Geschäftsprozessmodellierung verwendet. Während BPEL als zurzeit bedeutendster Standard zur Webservice-basierten Prozessrealisierung eine etablierte Sprache zur flussorientierten Dienstaggregation darstellt, wurde SM4RCD im Rahmen dieser Arbeit entwickelt und erlaubt eine zustandsbasierte Aggregation. Beide Sprachen erlauben die Modellierung durch Fachpersonal und die transparente Ausführung der Prozessmodelle auf Sensorknoten sowie im Backend.

Wie in Abschnitt 3.4.4 beschrieben, sind neben der Prozessdefinition und -ausführung die Aspekte der Überwachung und Optimierung jedoch nicht minder bedeutend für ein Geschäftsprozessmanagement. Aus diesem Grund wurde im Rahmen dieser Arbeit mit

IWFMS (*Integral Workflow Management System*) ein umfassendes Workflowmanagementsystem (WFMS) zur Realisierung eines ganzheitlichen Geschäftsprozessmanagements entwickelt. Es folgt dem WfMC-Referenzmodell zur Umsetzung eines Workflowmanagementsystems und realisiert alle darin geforderten Schnittstellen. So können nicht nur die Prozessmodellierung und -ausführung, sondern alle Aufgaben im Rahmen eines Prozessmanagements mit entsprechenden Werkzeugen von Fachpersonal durchgeführt werden.

In den Abschnitten 5.1 und 5.2 werden als Erstes die Herausforderungen eines Prozessmanagements im Kontext von Sensornetzen sowie verwandte Arbeiten diskutiert. In der Folge wird das vom Autor dieser Arbeit entwickelte IWFMS beschrieben. Um seine Umsetzbarkeit und Leistungsfähigkeit nachzuweisen, wird in Abschnitt 5.6 eine Evaluation präsentiert, bevor das Kapitel in Abschnitt 5.7 mit einer Zusammenfassung schließt.

Die vom Autor dieser Arbeit entwickelten und in diesem Kapitel präsentierten Lösungen zum ganzheitlichen Prozessmanagement wurden z. T. in folgenden Publikationen bereits vorab veröffentlicht [66, 69, 71, 73]

5.1 Herausforderungen

Zur Unterstützung der Modellierung, Ausführung, Überwachung und Optimierung von Geschäftsprozessen auf der operativen Workflowebene werden Workflowmanagementsysteme eingesetzt. In Abschnitt 2.2.3 wurde das Referenzmodell eines WFMS der WfMC vorgestellt. Es stellt die Ausführung von Geschäftsprozessen ins Zentrum des Modells. Der für die Ausführung verantwortliche Workflowausführungsdienst interagiert über verschiedene Schnittstellen mit den weiteren Funktionen eines WFMS. Der Anschluss von Werkzeugen und Sprachen zur Prozessmodellierung und Überwachung sowie Administration an den Ausführungsdienst erfolgt genauso über diese Schnittstellen wie die Interaktion mit prozessexternen Aufgabenträgern (Benutzern, externen Anwendungssystemen und externen Ausführungsdiensten).

Um ein Geschäftsprozessmanagement im Kontext von Sensornetzen auf Basis des in dieser Arbeit entwickelten und in Abschnitt 3.4 präsentierten Konzepts umzusetzen, muss ein vollständiges WFMS zur Verfügung gestellt werden. Dabei wird vom Konzept dieser Arbeit ein ganzheitlicher Ansatz gefordert. Ganzheitlich bedeutet in diesem Zusammenhang, dass zum einen dieselben Sprachen und Werkzeuge zur Modellierung, Überwachung und Optimierung von Prozessen unabhängig von der jeweiligen Ausführungsplattform der Prozesse eingesetzt werden sollen. Zum anderen muss die Interaktion eines Prozesses mit anderen Diensten transparent erfolgen. Zusätzlich muss die komplette Verwendung des WFMS domänenorientiert und ohne besondere IT- oder WSN-Expertise möglich sein, damit das Fachpersonal in der Lage ist, das vollständige Geschäftsprozess- bzw. Workflowmanagement durchzuführen.

Die entscheidende Forschungsherausforderung und der Schlüssel zum Erfolg bei der Realisierung eines ganzheitlichen Geschäftsprozessmanagements liegen in der Ausführung von Geschäftsprozessen, da die Laufzeitumgebungen zur Ausführung der Prozessmodelle sowohl im Backend als auch im WSN ausgeführt werden müssen und mit beliebigen prozessexternen Aufgabenträgern interagieren müssen. Die Ausführung von Prozessen in einem Workflowausführungsdienst betrifft mit der Kommunikation sowie der eigentlichen Prozessausführung zwei Kernaufgaben.

Wie im WfMC-Referenzmodell (s. Abbildung 2.4 auf Seite 18) zu sehen, interagiert der Ausführungsdienst über die Schnittstellen 2 bis 4 mit externen Aufgabenträgern. Das im Rahmen dieser Arbeit entwickelte Konzept sieht vor, dass jegliche prozessexterne Funktionalität als über ein Netz verteilte Dienste zur Verfügung gestellt wird, die über Webservices eingebunden werden. Es wird folglich ein Webservice-Transport-Binding benötigt, das eine ganzheitliche Kommunikation ermöglicht und gleichzeitig im Zusammenspiel mit Geschäftsprozessrealisierungssprachen verwendet werden kann. BPEL und SM4RCD als die beiden im Rahmen dieser Arbeit verwendeten Prozessrealisierungssprachen verlangen die Verwendung von mit WSDL 1.1 beschreibbaren Bindings. Das in Kapitel 4 vorgestellte LTP+SMC erfüllt beide Anforderungen und löst damit bereits die Kommunikationsherausforderungen eines ganzheitlichen Prozessmanagements.

Die zweite Herausforderung bei der Ausführung von Geschäftsprozessen ist die eigentliche Prozessausführung. Sie umfasst die Verwaltung von Prozessinstanzen, die zeitlich logische Abarbeitung der Prozessschritte sowie die Steuerung der Kommunikation eines Prozesses. Dabei muss zwischen der Ausführung im Backend und auf ressourcenbeschränkten Sensorknoten unterschieden werden. Die Unterscheidung darf jedoch nur interne Aspekte der Ausführung innerhalb der Laufzeitumgebungen betreffen. Nach außen müssen jeweils alle Schnittstellen des WFMS realisiert werden. Aufgrund der Ressourcenstärke von Backend-Systemen müssen für diese Ausführungsplattform keine Besonderheiten bei der Ausführung beachtet werden. Entsprechende Laufzeitumgebungen unterscheiden sich lediglich in der Kommunikation mit externen Aufgabenträgern von existierenden Laufzeitumgebungen der Enterprise-IT.

Anders als im Backend sind zur Realisierung der Prozessausführung im WSN aufgrund der Ressourcenbeschränkungen große Forschungsherausforderungen zu bewältigen. Prozessmodelle werden heute in der Regel in schwergewichtigen Laufzeitumgebungen interpretiert. Aber sowohl diese Laufzeitumgebungen als auch die Prozessmodelle selbst sind zu groß und können nicht einmal auf Knoten gespeichert werden. Für den passiven Überwachungsprozess (s. Abschnitt 5.6.1) erzeugt das komplette Prozessmodell eine Größe von 21 KB. Genau wie BPEL ist SM4RCD ebenfalls eine XML-Sprache. Beide Sprachen benötigen zusätzlich zur eigentlichen Prozessbeschreibung noch weitere ergänzende XML-Dokumente (z. B. WSDL und XML-Schema). Darüber hinaus werden noch weitere XML-Standards (z. B. XPath) in die Prozessmodelle eingebunden.

Um entsprechende Prozessmodelle auf Sensorknoten ausführen zu können, müssen besondere, ressourceneffiziente Laufzeitumgebungen entwickelt werden. Sie müssen alle notwendigen Funktionalitäten zur Ausführung der Prozesslogik, zur Verwaltung der Prozessinstanzen und zur Steuerung der Kommunikationen realisieren. Dabei ist die größte Herausforderung die Realisierung der umfangreichen und komplexen Sprachkonstrukte und die Abbildung von massiv parallelisierten Kontrollstrukturen in einer Laufzeitumgebung bei gleichzeitiger Einhaltung der Vorgaben bzgl. eines geringen Ressourcenverbrauchs. Gleiches gilt für die Prozessmodelle selbst. Sie müssen zum einen in einer im Vergleich zu den XML-Quelldokumenten kompakteren Repräsentation auf den Knoten gespeichert werden. Gleichzeitig muss eine direkte Ausführung der entsprechenden Modellrepräsentationen ohne zusätzlichen Overhead erfolgen können. Selbst wenn z. B. BPEL-XML-Dokumente direkt auf Knoten gespeichert werden könnten, würde ihre Interpretation bei Weitem die Kapazitäten der Knoten übersteigen.

Aus fachlicher Sicht betrachtet ist die Prozessmodellierung sowie die -überwachung und -optimierung nicht minder wichtig. Aus technischer Sicht betrachtet, bedarf es hier jedoch eines

Ansatz	Plattform	Kommunikation	Logik
Pandey et al.	Backend	◯	◯
GWELS	Backend	◯	◯
Sadilek	Sensorknoten	–	–
Naumowicz et al.	Sensorknoten	–	–
Losilla et al.	Sensorknoten	–	–
Fuchs et al.	Sensorknoten	–	◯
Hackman et al.	PDA	◯	◯
Spieß et al. (I)	Sensorknoten & Backend	–	◯
Spieß et al. (II)	PDA	◯	◯
Windows Workflow Foundation	PC	◯	◯

(+: erfüllt; ◯: teilweise erfüllt; –: nicht erfüllt)

Tabelle 5.1: Bewertung der Eignung verwandter Arbeiten zur Umsetzung des Konzepts dieser Arbeit zum ganzheitlichen Geschäftsprozessmanagement

weniger großen Forschungseinsatzes, da diese Vorgänge bzw. die entsprechenden Werkzeuge auf Backend-Systemen ausgeführt werden. Bei der Realisierung der Ausführungsdienste muss lediglich darauf geachtet werden, dass die entsprechenden Schnittstellen zu diesen Programmen bzw. Sprachen eingehalten werden. Der Einsatz von bereits etablierter Software in diesem Bereich bietet den Vorteil der einfachen Integration eines Geschäftsprozessmanagements im Kontext von Sensornetzen in ein bereits bestehendes Prozessmanagement und somit den Schutz von Investitionen.

5.2 Verwandte Arbeiten

In diesem Abschnitt werden verwandte Arbeiten präsentiert und deren Eignung zur Umsetzung des in Abschnitt 3.4 präsentierten Konzepts zum ganzheitlichen Geschäftsprozessmanagement bewertet. Tabelle 5.1 fasst die Ergebnisse zusammen.

Erste Ansätze zur Realisierung eines Geschäftsprozessmanagements im Kontext von Sensornetzen bzw. zur generellen modellgetriebenen Integration von Sensornetzen und Enterprise-IT-Systemen wurden von Pandey et al. [166] sowie in früheren Publikationen [67] des Autors dieser Arbeit präsentiert. Beide Ansätze erlauben eine grafische Aggregation von Diensten, die von Sensorknoten angeboten werden sowie die Nutzung dieser Aggregationen durch Knoten. Während Pandey et al. BPEL verwenden, wird vom Autor dieser Arbeit mit GWELS (*Graphical Workflow Execution Language*) eine selbst entwickelte DSL genutzt. In beiden Ansätzen werden die Prozessmodelle ausschließlich im Backend ausgeführt. Eine Integration mit anderen Backend-Systemen erfolgt über Webservices. Allerdings ist diese Kommunikation auf das Backend beschränkt. Zwischen den Prozessen und Sensorknoten werden in beiden Ansätzen proprietäre Kommunikationsprotokolle eingesetzt. Sowohl aufgrund der fehlenden ganzheitlichen Webservice-Kommunikation als auch wegen der Limitierungen bei der Prozessausführung sind beide Ansätze nicht geeignet, um das Konzept dieser Arbeit zum ganzheitlichen Prozessmanagement umzusetzen.

Losgelöst von der Geschäftsprozessdomäne existieren einige Forschungsansätze, deren Ziel ebenfalls die Modellierung von Anwendungslogik zur Ausführung auf Sensorknoten ist. Sie

verwenden eine modellgetriebene Softwareentwicklung auf der Basis unterschiedlicher, selbst entwickelter domänenspezifischer Sprachen. Entsprechende Ansätze werden von Sadilek, Naumowicz et al. und Losilla et al. beschrieben.

Sadilek präsentiert mit *EPROMISE* [185, 186] ein Framework zur Definition von domänenspezifischen Sprachen. Mit diesen DSLs definierte Modelle lassen sich in Maschinencode sowie in Bytecode überführen und direkt oder in einer virtuellen Maschine auf Sensorknoten ausführen. EPROMISE ist lediglich ein Framework zur Definition von DSLs. Es stellt selbst keine DSLs und insbesondere keine DSLs für ein Prozessmanagement zur Verfügung. Auch sind weder das SOA-Paradigma noch Webservices in diesem Ansatz berücksichtigt. Damit eignet er sich ebenfalls nicht zur Umsetzung des Konzepts dieser Arbeit.

Mit *Flow* [143, 144] präsentieren Naumowicz et al. einen grafischen Editor und verschiedene DSLs zur Erzeugung von Software für drahtlose Sensornetze. Aus Modellen zur Beschreibung der Hardware, der Datenstrukturen sowie des Datenflusses wird mit einem Codegenerator C-Code zur Ausführung auf Sensorknoten generiert. Die Sprachen sind jedoch weder zur Beschreibung von Geschäftsprozessen noch zur direkten Verwendung durch Fachpersonal geeignet. Zusätzlich verhindert auch die sowohl im WSN als auch im Backend fehlende Webservice-Kommunikation sowie die nicht vorhandene Serviceorientierung die Verwendung dieses Ansatzes zur Umsetzung des Konzepts dieser Arbeit.

Losilla et al. [118] verwenden den Ansatz der modellgetriebenen Softwareentwicklung zur WSN-Anwendungsentwicklung in der Präzisionslandwirtschaft. Entsprechende Modelle werden mit speziellen DSLs erstellt und in nesC-Code übersetzt und können so direkt auf Sensorknoten ausgeführt werden. Und obwohl der Sprachumfang sehr gering ist, kann nur ein Teil der gesamten Modellierung durch Domänenexperten durchgeführt werden. Auch ist die DSL nicht auf ein Prozessmanagement übertragbar. Als zusätzliches Ausschlusskriterium kommt hinzu, dass dieser Ansatz weder eine Webservice-Kommunikation noch das SOA-Paradigma verwendet. Eine weitere Verwendung im Rahmen dieser Arbeit ist also nicht möglich.

Anders als die drei letztgenannten Ansätze verwenden Fuchs et al. [58] keine eigene DSL zur WSN-Anwendungsentwicklung. Stattdessen setzt dieser Ansatz UML-Aktivitätsdiagramme [146] zur grafischen Modellierung ein. Zwar eignen sich Aktivitätsdiagramme sowohl aufgrund ihrer Abbildung der Geschäftsprozessdomäne als auch wegen ihrer Nutzbarkeit durch Fachpersonal prinzipiell für eine Prozessmodellierung. Allerdings erlaubt der Ansatz keine ausreichende deklarative Beschreibung der Kommunikationsaspekte. Zusätzlich verhindert die fehlende Unterstützung von Webservices und einer Serviceorientierung eine Integration von auf Knoten ausgeführten Anwendungen in sonstige Prozesse oder Dienste im Gesamtsystem. Damit kann auch dieser Ansatz nicht zur Umsetzung des Konzepts dieser Arbeit zum ganzheitlichen Prozessmanagement verwendet werden.

Hackman et al. [75] präsentieren mit *Sliver* eine leichtgewichtige BPEL-Laufzeitumgebung für mobile Endgeräte. Die Verwendung von BPEL zur Prozessrealisierung, der Einsatz von WSDL zur Dienstbeschreibung und die Verwendung von SOAP zum Nachrichtenaustausch würden eine sehr gute Eignung zur Umsetzung des Konzepts dieser Arbeit begründen. Allerdings ist dieser Ansatz für Ausführungsplattformen mit einer Ressourcenausstattung der PDA-Klasse entwickelt worden. Weder das verwendete Transport-Binding noch die BPEL-Laufzeitumgebung können im WSN bzw. auf Sensorknoten verwendet werden.

Einen ersten Ansatz zur partiellen Ausführung von BPEL im WSN präsentieren Spiess et al. [198–200]. Dabei werden BPEL-Prozesse in einzelne Subprozesse aufgeteilt, die verteilt z. T. auf Sensorknoten und z. T. auf Enterprise-IT-Servern ausgeführt werden. In einer ersten Entwicklungsstufe verwendet dieser Ansatz jedoch keine Webservice-Kommunikation. Zwar kann eine ganzheitliche und auch weitestgehend transparente Prozessrealisierung vorgenommen werden. Die WSN-Applikation selbst folgt jedoch nicht dem SOA-Paradigma, sondern besteht aus monolithischen Blöcken. Weder konzeptionell noch technisch ist so eine flexible organisationsübergreifende Prozessintegration realisierbar. In einer zweiten Entwicklungsstufe wird durch die Verwendung von Sliver als BPEL-Laufzeitumgebung eine Webservice-Kommunikation ermöglicht. Allerdings wird dadurch, wie oben diskutiert, der Einsatz im Sensornetz verhindert. Damit ist auch der Ansatz von Spiess et al. nicht für ein Geschäftsprozessmanagement im Rahmen dieser Arbeit geeignet.

Auch die in Abschnitt 2.3.3 vorgestellte *Windows Workflow Foundation* stellt einen MDSD-Ansatz dar, der nicht ausschließlich typische Enterprise-IT-Umgebungen als Ausführungsplattform vorsieht. Alle drei Sprachtypen eignen sich sehr gut für die Umsetzung von Geschäftsprozessen. Allerdings existiert zurzeit keine Laufzeitumgebung für Sensornetze oder vergleichbare Geräteklassen. Auch die von der Windows Workflow Foundation unterstützte Webservice-Kommunikation eignet sich nicht für den Einsatz in Sensornetzen.

Keiner der hier aufgeführten Ansätze erfüllt die Voraussetzungen, um zur Umsetzung des Konzepts eines ganzheitlichen Geschäftsprozessmanagements eingesetzt werden zu können. Damit ist die Entwicklung eines eigenen Ansatzes notwendig.

5.3 Sprachen zur ganzheitlichen Realisierung von Geschäftsprozessen

Wie im Konzept dieser Arbeit beschrieben, wird ein Geschäftsprozessmanagement technisch durch die ganzheitliche Umsetzung von Anwendungs- und Prozessfunktionalität als Dienste und Dienstaggregationen auf Basis der Webservice-Technologie realisiert. Mit BPEL (s. Abschnitt 2.3.3) existiert bereits eine geeignete und weitverbreitete Sprache, die die Modellierung von ausführbaren Geschäftsprozessen durch eine Aggregation von Webservices realisiert. Diese Kompositionen stellen selbst wieder Webservices dar, sodass ein rekursives Aggregationsmodell umgesetzt wird. BPEL realisiert eine hierarchische, flussorientierte Prozessmodellierung und stellt einen großen Sprachumfang zur Verfügung.

Neben einer flussorientierten stellt eine zustandsorientierte Prozessmodellierung eine weitere wichtige Modellierungsart dar. Mit UML-Zustandsdiagrammen (s. Abschnitt 2.2.2) und WF-Statuscomputern (s. Abschnitt 2.3.3) existieren bereits verschiedene Sprachen und Technologien, die eine Modellierung auf Basis von Zustandsdiagrammen erlauben. UML-Zustandsdiagramme realisieren eine allgemeine Modellierung von Softwaresystemen und sind nicht speziell auf eine Webservice-Aggregation ausgerichtet. Wie im vorigen Abschnitt beschrieben, existiert für die WF zurzeit weder eine Laufzeitumgebung noch eine Webservice-Lösung für Sensornetze. Eine Ausführung entsprechender Modelle im WSN wird insbesondere dadurch verhindert, dass die WF neben einer plattformunabhängigen Webservice-Aggregation auch die Einbindung und Realisierung von Prozessfunktionalität über eine Vielzahl weiterer

Technologien ermöglicht. Diese sind größtenteils jedoch nicht für den Einsatz im WSN geeignet und weisen oft eine sehr enge Abhängigkeit zur .NET-Plattform auf. Aus diesen Gründen wurde mit WS-SM4RCD (*Web Services State Machine for Resource Constrained Devices*, kurz: SM4RCD) im Rahmen dieser Arbeit in Anlehnung an UML-Zustandsdiagramme und WF-Statuscomputer eine eigene domänenspezifische Sprache zur zustandsbasierten Prozessdefinition entwickelt. SM4RCD wurde bereits vorab in [73] vorgestellt. Im Gegensatz zu UML-Zustandsdiagrammen und WF-Statuscomputern ist sie gezielt auf den Einsatz von Technologien der WS-*-Architektur und eine rekursive Webservice-Aggregation ausgerichtet und beschränkt sich weitestgehend auf die minimal notwendigen Sprachkonstrukte dieser Kompositionsart. SM4RCD lässt im Gegensatz zu WF-Statuscomputern durch seinen minimalistischen Sprachumfang eine Prozessausführung auf ressourcenbeschränkten Geräten zu.

Im Rahmen dieser Arbeit werden sowohl BPEL als auch SM4RCD zur Geschäftsprozessrealisierung eingesetzt. Bevor in Abschnitt 5.3.2 das Zusammenspiel von SM4RCD und BPEL sowie in Abschnitt 5.4 ein WFMS zur informationstechnischen Umsetzung entsprechender Prozessmodelle beschrieben wird, führt der folgende Abschnitt die Sprache SM4RCD ein.

5.3.1 Web Services State Machine for Resource Constrained Devices

SM4RCD ist eine domänenspezifische Sprache zur Orchestrierung von Webservices auf Basis von Zustandsautomaten. Wie in Abbildung 3.3 auf Seite 60 dargestellt, erweitert sie den Webservice-Technologiestapel auf der Prozessrealisierungsebene. Sie baut genau wie BPEL auf dem abstrakten Teil einer WSDL (Version 1.1) sowie auf XML-Schema auf. Abbildung 5.1 zeigt ein UML-Diagramm mit der abstrakten Syntax von SM4RCD.

Wie jeder endliche Zustandsautomat besteht ein SM4RCD-Modell im Kern aus einer endlichen Menge an Zuständen und Transitionen, die die Zustandsübergänge beschreiben. Die Zustandsübergänge werden durch verschiedene Ereignisse ausgelöst. Sowohl während der Transitionen als auch beim Eintritt in und Verlassen von Zuständen können Aktionen ausgeführt werden. Darin unterscheidet sich SM4RCD grundlegend von bekannten Ansätzen der Automatentheorie wie Mealy- [129] und Moore-Automaten [141], bei denen entweder Transitionen oder Eintritte in Zustände mit Ausgaben verknüpft sind. Auch erlaubt keiner der beiden Ansätze Ausgaben beim Verlassen eines Zustands. Der Verzicht auf diese Restriktionen hat zwar keine Auswirkungen auf die Ausdrucksstärke von SM4RCD. Er erlaubt jedoch eine wesentlich flexiblere und kompaktere Modellierung mit weniger Redundanzen.

Zur Speicherung der Daten und zum Austausch von Daten zwischen verschiedenen Ereignissen und Aktionen werden Variablen (*Variable*) verwendet. Variablen sind in SM4RCD prozessweit gültig. Sie haben einen Namen und einen Datentyp. Als Datentypen können ausschließlich in einem WSDL- oder XML-Schema-Dokument, das über *DefinitionDocument* referenziert wird, definierte Datentypen verwendet werden. Datentypen weisen entsprechend dem XML-Modell eine hierarchische Struktur auf.

SM4RCD stellt zwei Arten von Aktionen zur Verfügung. Über *InvokeWebService* kann ein Webservice aufgerufen werden. Dazu muss eine Referenz auf die WSDL-Beschreibung des aufzurufenden Webservices angegeben werden. *PortType* sowie *Operation* identifizieren in

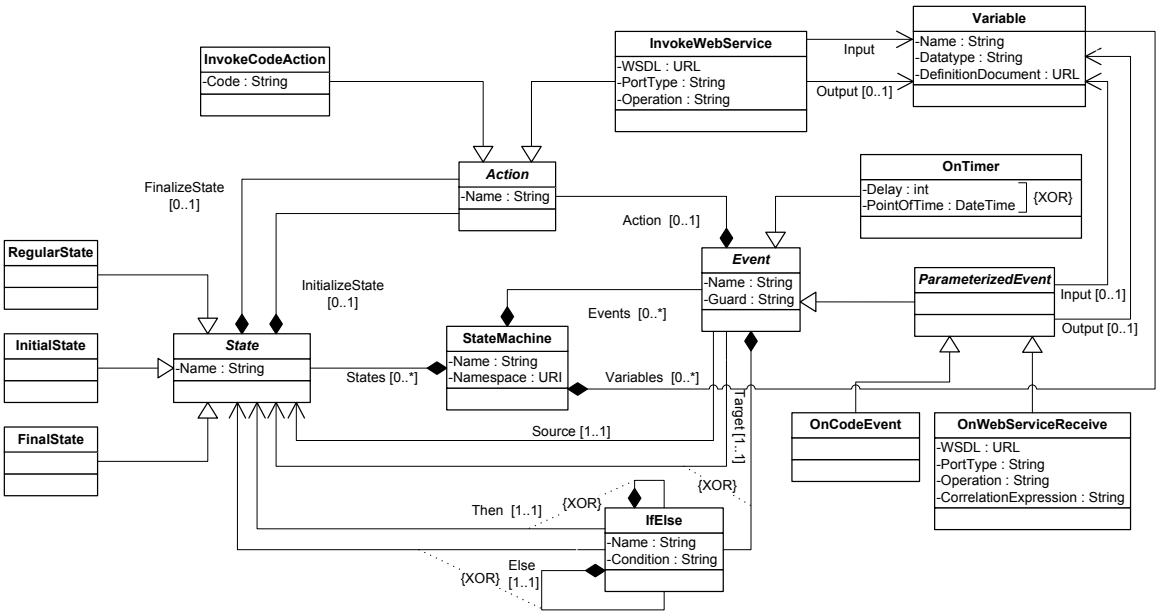


Abbildung 5.1: Abstrakte Syntax von SM4RCD

dieser WSDL den abstrakten Dienst sowie die aufzurufende Operation. Das konkrete Binding sowie der Webservice-Endpunkt können aus der WSDL übernommen oder während des Installationsprozesses separat festgelegt werden. Zusätzlich müssen die Ein- und Ausgabevariablen (nur bei Request-Response-Kommunikation) referenziert werden, aus denen der Inhalt für eine Webservice-Anfrage gelesen bzw. der einer -Antwort für die weitere Verarbeitung im Prozess gespeichert wird. Dazu kann einfach der Name einer deklarierten Variablen angegeben werden. Alternativ können auch über eine einfache Punktnotation (z. B. `Variable.Kindelement`) Kindelemente einer deklarierten Variablen verwendet werden. Die Übereinstimmung des Datentyps der referenzierten Variablen bzw. des Kindelements mit dem der Ein- oder Ausgabenachricht ist obligatorisch.

Da im Rahmen dieser Arbeit eine Geschäftsprozessrealisierung durch eine Webservice-Aggregation die konzeptionelle Grundlage bildet und SM4RCD auch für genau diesen Einsatz entworfen wurde, werden Aktionen in der Regel als Webservice-Aufrufe realisiert. In einigen Fällen muss aber auch Funktionalität realisiert werden, in denen Webservice-Aufrufe nicht zweckmäßig sind. Z. B. wäre ein Webservice-Aufruf für das einfache Einschalten einer Leuchtdiode zu überdimensioniert. Aus diesem Grund bietet SM4RCD mit *InvokeCodeAction* noch einen weiteren Aktionstypen. Er erlaubt die Definition der Aktionslogik in einer Programmiersprache der 3. Generation (zurzeit C++). Innerhalb dieses Codes kann auf alle Prozessvariablen zugegriffen werden. Lokal in diesem Codesegment deklarierte Variablen sind bzgl. ihrer Sichtbarkeit auf das Segment oder auf bestimmte Codeblöcke innerhalb dieses Codesegments beschränkt.

Transitionen werden in SM4RCD durch Ereignisse (*Event*) ausgelöst und repräsentiert. Bei der Durchführung des Zustandsübergangs kann optional eine Aktion ausgeführt werden. Jedes Ereignis hat genau einen Quell- und mindestens einen Zielzustand. Es kann entweder direkt einen Zielzustand oder durch verschachtelte *IfElse*-Elemente mehrere Zielzustände referenzie-

ren und so bedingte Transitionen erlauben. Neben den bedingten Transitionszielen kann auch der Start einer Transition über den booleschen Ausdruck *Guard* an Bedingungen geknüpft sein. *Guard* wird in der C++-Notation beschrieben und kann auf alle Prozessvariablen zugreifen.

Obwohl zu einem Quellzustand mehrere bedingte Transitionsziele definiert werden können, stellt SM4RCD einen deterministischen Automaten dar. Deterministisch bedeutet in diesem Zusammenhang, dass nach jedem Ereignis entweder keine Transition oder eine Transition zu genau einem Zielzustand durchgeführt wurde. Der Automat kann nicht mehrere Zustände gleichzeitig annehmen.

SM4RCD unterscheidet zurzeit drei Arten von Ereignissen. *OnTimer* löst zeitgesteuert ein Ereignis aus. Dieses kann der Ablauf einer Zeitspanne oder das Erreichen eines Zeitpunkts sein.

OnWebServiceReceive löst eine Transition aufgrund einer eingehenden Webservice-Nachricht aus. Der Prozess nimmt dabei die Rolle des Webservice-Anbieters ein. Dazu muss die abstrakte Webservice-Schnittstelle in einem WSDL-Dokument beschrieben und aus dem Modell referenziert werden. Zusätzlich muss der abstrakte Dienst sowie die Operation, die der Prozess zur Ereignisauslösung anbietet, definiert werden. Analog zu *InvokeWebServiceAction* müssen auch hier die Prozessvariablen zur Speicherung der Webservice-Ein- und -Ausgaben (nur bei Request-Response-Kommunikation) referenziert werden.

SM4RCD-Modelle stellen Prozesse dar, die einen mehrere Webservice-Aufrufe überdauernden Zustand aufweisen können. Damit sind sie auch im Sinne einer Webservice-Kommunikation zustandsbehaftet. Da eine Webservice-Kommunikation jedoch grundsätzlich zustandslos ist, müssen einzelne Nachrichten Prozessinstanzen zugeordnet werden. Über *CorrelationExpression* wird ein Datenfeld innerhalb der Eingabevariablen festgelegt, das einen Wert zur eindeutigen Identifizierung der Prozessinstanz enthält. Dieses muss ein primitiver Datentyp sein. Die Adressierung dieses Datenfeldes erfolgt ebenfalls mit der o. g. Punktnotation.

Als letzten Ereignistypen spezifiziert SM4RCD *OnCodeEvent*. Es ist das Gegenstück zu *InvokeCodeAction* und ermöglicht die Auslösung einer Transition durch direkte Methodenaufrufe. So kann z. B. auf einem Sensorknoten ein Prozessereignis direkt durch ein Hardwareereignis ausgelöst werden. Auch hier können Ein- und Ausgabevariablen definiert werden. Sie dienen dem direkten Datenaustausch zwischen Prozessinstanz und Aufrufer/Ereignisauslöser.

Neben Ereignissen sind die Zustände das zweite Kernelement von SM4RCD. *State* repräsentiert einen Zustand in einem SM4RCD-Modell. Mit initialen (*InitialState*), regulären (*RegularState*) sowie finalen Zuständen (*FinalState*) werden drei Zustandsarten unterschieden. Jedes Modell muss genau einen initialen Zustand haben und kann über eine beliebig große, endliche Menge an regulären und finalen Zuständen verfügen. Beim Erreichen sowie Verlassen eines Zustands kann eine Aktion ausgeführt werden. Finale und initiale Zustände weichen bzgl. der Definition von Aktionen und Transitionen von den regulären Zuständen ab. Initiale Zustände dürfen zwar Aktionen für die Zustandsinitialisierung definieren. Allerdings werden diese lediglich beim erneuten Erreichen des Zustands und nicht bei der Prozessinstanziierung ausgeführt, da sich der Prozess bei der Instanziierung bereits in diesem Zustand befindet und nicht in diesen überführt wird. Auch dürfen zwar beliebige ausgehende Transitionen definiert werden. Eine Instanziierung ist jedoch nur durch Ereignisse vom Typ *ParameterizedEvent* möglich. Finale Zustände beenden die Prozessinstanz. Sie dürfen keine ausgehenden Transitionen und keine Aktion für den Zustandsaustritt haben.

StateMachine repräsentiert das Wurzelelement des gesamten Prozessmodells. Genau wie alle anderen Elemente verfügt es über einen eindeutigen Namen. Zusätzlich kann es einem durch eine URI beschriebenen Namensraum zugeordnet werden.

5.3.2 Zusammenspiel von BPEL und SM4RCD

Sowohl BPEL als auch SM4RCD eignen sich sehr gut als Realisierungssprachen zur Umsetzung des in Abschnitt 3.4 beschriebenen Konzepts zum ganzheitlichen Geschäftsprozessmanagement. Beide realisieren ein rekursives Aggregationsmodell, indem sie sowohl als Konsument als auch als Anbieter eines oder mehrerer Webservices agieren. Die genauen Unterschiede liegen in der Art der Prozessmodellierung sowie im Sprachumfang. BPEL ist vom Sprachumfang wesentlich umfassender als SM4RCD und bietet Elemente, die die Definition komplexer Kontrolllogik erlauben. So bietet BPEL z. B. verschiedene Elemente, um parallele Prozessflüsse zu formulieren, Gültigkeitsbereiche zu definieren sowie Elemente zur Fehlerbehandlung und zur Kompensation. Darüber hinaus ist BPEL ein Standard, der bereits einen hohen Verbreitungsgrad in der universitären Forschung sowie in der Unternehmenspraxis hat.

Demgegenüber ist SM4RCD wesentlich schlanker und verursacht einen viel geringeren Overhead als BPEL (s. Abschnitt 5.6). SM4RCD erlaubt durch seine auf Zustandsautomaten basierende, graphenorientierte Prozessmodellierung eine wesentlich größere Flexibilität bei der Formulierung der Ablauflogik im Vergleich zu BPEL. BPEL ermöglicht zwar eine flussorientierte Prozessmodellierung. Diese folgt jedoch einem hierarchischen Aufbau und ist damit weniger flexibel.

BPEL und SM4RCD stehen in dieser Arbeit jedoch nicht in einer Konkurrenzsituation. Beide Sprachen erfüllen die Anforderungen des Konzepts dieser Arbeit und weisen starke Synergieeffekte auf. Je nach Prozessart kann aus fachlicher Sicht eine fluss- oder eine zustandsorientierte Prozessmodellierung geeigneter sein. In vielen Fällen, insbesondere im Umfeld von Sensornetzen, stellt die Kombination beider Sprachen die beste Lösung dar.

Wird z. B. ein zustandsorientierter Prozess modelliert, eignet sich SM4RCD sehr gut zur Definition der Zustände und Zustandsübergänge. BPEL ist für diesen Zweck weniger gut geeignet. Bei der Beschreibung der Logik der Aktionen, deren Ausführung mit den Zuständen und Zustandsübergängen verknüpft ist, ist genau das Gegenteil der Fall. Sie können in der Regel durch eine flussorientierte Modellierung besser abgebildet werden und verlangen komplexere Sprachelemente, die SM4RCD nicht bieten kann. Für diese Teilprozesse stellt BPEL die geeignetere Realisierungssprache dar. Durch das rekursive Webservice-basierte Aggregationsmodell beider Sprachen lassen sich verschiedene Teilprozesse flexibel zu einem Gesamtmodell kombinieren.

5.4 Integral Workflow Management System

In diesem Abschnitt wird das vom Autor dieser Arbeit entwickelte Workflowmanagementsystem IWFMS (*Integral Workflow Management System*) beschrieben. Es realisiert das in Abschnitt 3.4 beschriebene Konzept zum ganzheitlichen Geschäftsprozessmanagement und

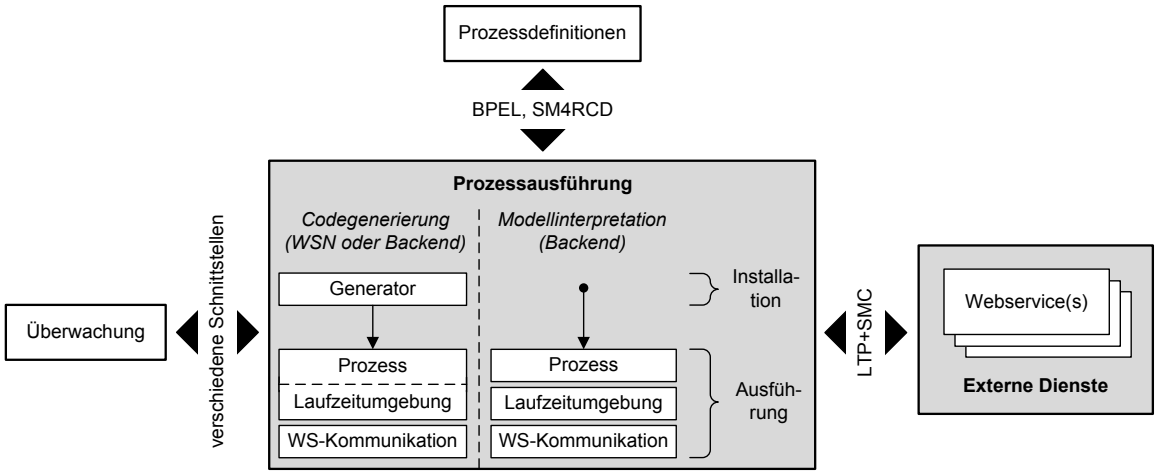


Abbildung 5.2: Aufbau des ganzheitlichen Workflowmanagementsystems IWFMS

erfüllt die Vorgaben des WfMC-Referenzmodells zur Gestaltung von Workflowmanagementsystemen. Abbildung 5.2 stellt die Architektur von IWFMS grafisch dar.

Im Zentrum von IWFMS steht die Ausführung von Geschäftsprozessen. IWFMS realisiert eine Prozessausführung sowohl auf Sensorknoten oder anderen ressourcenbeschränkten Geräten als auch auf Backend-Computern. Dabei ist die Ausführungsplattform bei der Prozessdefinition und -überwachung vollständig transparent. Diese Prozessmanagementaufgaben werden über klar definierte Schnittstellen an die Prozessausführung angeschlossen. Während als Schnittstelle für die Prozessdefinition mit SM4RCD und BPEL zwei Sprachen zur Prozessmodellierung durch Webservice-Aggregation unterstützt werden, erfolgt die Überwachung über verschiedene Schnittstellen.

Die Interaktion mit externen Diensten erfolgt über eine Webservice-Kommunikation. Durch den Einsatz von LTP+SMC als Transport-Binding kann ein Prozess mit beliebigen Endpunkten im Gesamtsystem kommunizieren. LTP+SMC stellt damit die Schnittstelle zwischen der Prozessausführung und externen Diensten dar.

5.4.1 Prozessausführung

Im Zentrum von IWFMS steht die Ausführung von Geschäftsprozessmodellen. Wie in Abbildung 5.2 dargestellt, ist bei der Ausführung das Zusammenspiel von drei Komponenten notwendig. Zum einen bedarf es einer formalen Definition des Prozesses (also der Logik) und der Interaktion mit Kommunikationspartnern. Zusätzlich ist eine Laufzeitumgebung erforderlich, die die Ausführung der Logik, die Steuerung der Kommunikation sowie die Verwaltung von Prozessinstanzen realisiert. Als dritte Komponente wird ein Framework benötigt, das die Webservice-Kommunikation zwischen den Kommunikationspartnern durchführt.

Im Rahmen des Konzepts dieser Arbeit wird jegliche Funktionalität als Dienst bereitgestellt. Das betrifft sowohl die Funktionalität, die von einem auszuführenden Prozess realisiert und von der Prozessausführungskomponente bereitgestellt wird, als auch jegliche externe Funktionalität, die von diesen Diensten/Prozessen konsumiert wird. In dieser Kommunikationsbeziehung

ist die Realisierungsform (manuelle Programmierung oder prozessorientierte Modellierung) genauso irrelevant wie die Unterscheidung zwischen maschinellen und menschlichen Aufgabenträgern, wie sie im WfMC-Referenzmodell getroffen wird. Auch die entsprechenden Ausführungsplattformen sind vollständig transparent. Jegliche externe Interaktion erfolgt mit einem Dienst über eine Webservice-Kommunikation.

IWFMS unterstützt mit SM4RCD und BPEL (Version 2.0) zwei Sprachen, die eine Prozessrealisierung durch Webservice-Aggregation auf Basis beliebiger mit WSDL beschreibbarer Webservice-Transport-Bindings erlauben. Da zurzeit jedoch ausschließlich LTP+SMC eine transparente Webservice-Kommunikation im Gesamtsystem realisiert, kann momentan auch nur dieses Binding zur Kommunikation mit externen Diensten eingesetzt werden.

Geschäftsprozesse sollen sowohl auf Enterprise-IT-Rechnern als auch auf ressourcenbeschränkten Geräten wie Sensorknoten ausgeführt werden. Aufgrund der unterschiedlichen Ressourcenausstattungen beider Gerätearten müssen unterschiedliche Ausführungsstrategien verwendet werden. Diese Unterscheidung bzgl. der Ausführungsplattformen betrifft allerdings ausschließlich interne Aspekte. Die Schnittstellen der Prozessausführung zu den anderen Komponenten des Workflowmanagementsystems bleiben davon unberührt, sodass die Ganzheitlichkeit und Transparenz des Konzepts dieser Arbeit gewahrt bleiben.

Die generelle Ausführung von Geschäftsprozessen auf Computern der Enterprise-IT ist unproblematisch. Laufzeitumgebungen für unterschiedliche Realisierungssprachen existieren schon seit Jahren. In Backend-Systemen werden Prozessmodelle in der Regel in einem Interpreter ausgeführt. In diesem Fall werden die Modelle bei der Installation direkt an die Laufzeitumgebung übergeben und können dort ohne weitere Installationsschritte zur Laufzeit eingelesen und ausgeführt werden. Um das Konzept dieser Arbeit umsetzen zu können, kann zur Realisierung der Prozessausführung im Backend zumindest für BPEL auf bereits existierende Laufzeitumgebungen aufgebaut werden. Diese müssen lediglich dahingehend erweitert werden, dass sie die ganzheitliche Webservice-Kommunikation mit LTP+SMC unterstützen. Bzgl. der eigentlichen Prozessausführung sind keinerlei Anpassungen notwendig.

Im Gegensatz zum Backend stellt die Ausführung von Geschäftsprozessen auf Sensorknoten eine sehr große Herausforderung dar. Da die Modelle von IWFMS mit BPEL und SM4RCD (inkl. ergänzender Standards wie XML-Schema, WSDL oder XPath) vollständig aus XML bestehen, erreicht die Beschreibung des Gesamtmodells schnell eine Größe von mehreren Kilobyte. Dieser Ressourcenverbrauch stellt bereits bzgl. der einfachen Speicherung der Prozessmodelle auf den Knoten einen zu großen Overhead dar. Eine Ausführung der Modelle in dieser Repräsentation wäre vollkommen unmöglich. Der benötigte Code zur Verarbeitung der XML-Dokumente und insbesondere der Interpreter der Modelle würden sowohl bzgl. der Codegröße als auch der Berechnungskomplexität (RAM und CPU) die Kapazitäten von Sensorknoten überschreiten. Die direkte Speicherung und Interpretation der Prozessmodelle auf den Sensorknoten ist folglich nicht realisierbar.

Weder BPEL noch SM4RCD legen fest, wie entsprechende Prozessmodelle ausgeführt werden müssen. Sie spezifizieren lediglich ein Format zur formalen Beschreibung von ausführbaren Modellen. Damit bilden beide Sprachen die Schnittstelle zwischen der Prozessdefinition und -ausführung. Die Repräsentation während der Ausführung betrifft nur interne Aspekte der Ausführung und bleibt dem Modellierer verborgen.

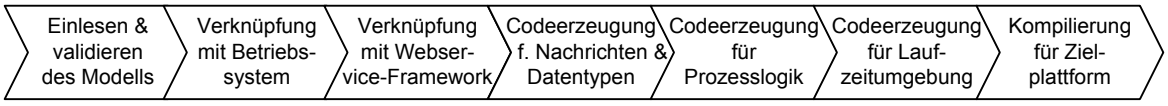


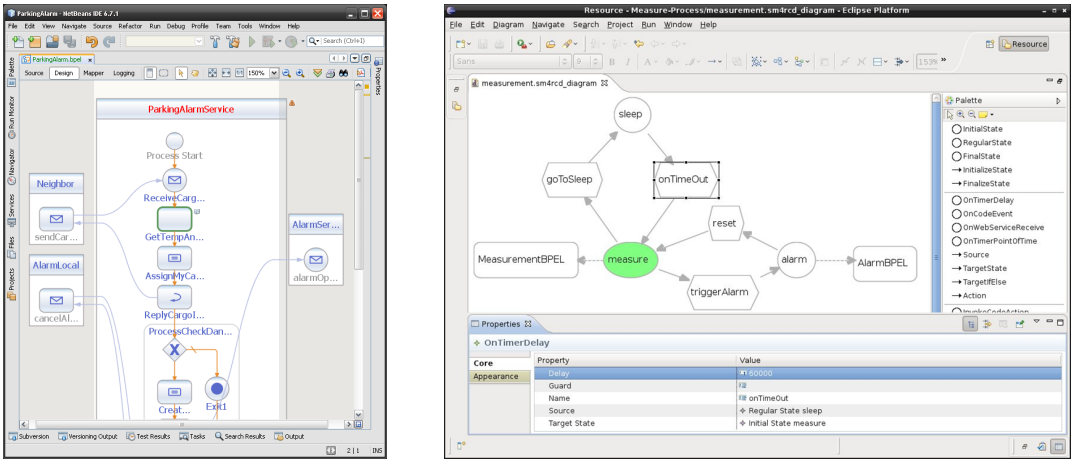
Abbildung 5.3: Schritte der Codegenerierung zur Ausführung von Geschäftsprozessmodellen

Anstatt einer Modellinterpretation stellt die Codegenerierung für ressourcenbeschränkte und eingebettete Geräte wie Sensorknoten die geeignetere Ausführungsstrategie dar. Im Rahmen dieser Arbeit werden BPEL- und SM4RCD-Modelle (inkl. ergänzender Dokumente) als Eingabe für einen Codegenerierungsvorgang verwendet und in optimierten C- und C++-Code übersetzt. Dieser bildet das komplette Prozessmodell sowie die notwendige Funktionalität einer BPEL- bzw. SM4RCD-Laufzeitumgebung ab. Der generierte Code kann in den Maschinencode der Zielplattform übersetzt und direkt auf den Knoten ausgeführt werden. Diese Ausführungsstrategie hat wesentliche Vorteile gegenüber einer Interpretation:

- Sowohl das Modell als auch die Laufzeitumgebung liegen in Maschinencode vor. Das vermeidet eine ineffiziente Speicherung der Modelle in einer XML-Repräsentation auf den Knoten. Zusätzlich verbraucht ein optimierter und in Maschinencode kompilierter Programmcode weniger Arbeitsspeicher und kann schneller ausgeführt werden als interpretierter Code. Damit kann der Speicher- und CPU-Verbrauch minimiert werden.
- Ein Interpreter muss immer den gesamten Sprachumfang enthalten, da das Modell erst zur Laufzeit bekannt ist. Bei der Codegenerierung ist das Modell bereits vor der Laufzeit während des Generierungsvorgangs bekannt. Damit ist es möglich, nur die vom konkreten Modell benötigte Laufzeitfunktionalität mit in das Generat einzufügen und die Codegröße zu minimieren. Das gilt neben den eigentlichen Laufzeitfunktionen auch für zusätzlichen Code, der z. B. zum Testen benötigt wird.
- Eine Vielzahl von Fehlern kann bei einer Codegenerierung bereits bei einer Modellvalidierung sowie beim Kompilieren des Generats erkannt werden. Bei einer Interpretation würden diese Fehler erst zur Laufzeit auftreten. Gerade aufgrund des eingebetteten Charakters von Sensornetzen und des damit verbundenen erhöhten Aufwands bei der Fehlersuche ist die Fehlerfrüherkennung ein weiteres wichtiges Merkmal.

Abbildung 5.3 stellt den Codegenerierungsvorgang dar. Er wird nach Abschluss der Modellierung während des Installationsvorgangs durchgeführt. Im ersten Schritt wird das gesamte Prozessmodell eingelesen und validiert. Anschließend wird das Betriebssystem der Zielplattform bestimmt, für das der in den folgenden Schritten generierte Code ausgeführt werden soll. Konzeptionell ist die Verwendung beliebiger WSN- und Backend-Betriebssysteme sowie verschiedener Simulationsplattformen möglich. In einem weiteren Generierungsschritt wird der Prozess an ein Webservice-Framework gebunden, das den Nachrichtenaustausch zwischen Kommunikationspartnern durchführt. Grundsätzlich können beliebige Frameworks verwendet werden, solange sie eine transparente Kommunikation im Gesamtsystem erlauben und konform zur WS-*-Architektur sind.

In den nächsten drei Generierungsschritten wird der modellspezifische Code generiert. Dazu wird als Erstes für jeden Kommunikationspartner der Code zur Repräsentation sowie zur (De-)Serialisierung der mit diesem Partner ausgetauschten SOAP-Nachrichten entsprechend der referenzierten WSDL-Beschreibungen erstellt. Zusätzlich werden C-Repräsentationen



(a) BPEL-Editor von NetBeans

(b) SM4RCD-Editor in Eclipse

Abbildung 5.4: Direkte Modellierung von Geschäftsprozessen in IWFMS mit BPEL und SM4RCD

der in den referenzierten XML-Schema-Beschreibungen definierten Datentypen generiert. Im nächsten Schritt wird die Prozesslogik in C++-Code überführt. Im letzten Generierungsschritt wird die Laufzeitumgebung für die entsprechende Modellausführung generiert, bevor das Generat für die ausgewählte Zielplattform kompiliert wird. Mit der Installation und Bereitstellung auf der entsprechenden Plattform wird der Generierungs- und Installationsvorgang beendet.

5.4.2 Prozessdefinition

Die IWFMS-Prozessausführung bietet mit BPEL und SM4RCD zwei Prozessrealisierungssprachen zur Beschreibung ausführbarer Prozessmodelle an. Sie bilden die Schnittstelle zwischen Prozessdefinition bzw. -modellierung und -ausführung. Der Prozessmodellierer kann unabhängig von der späteren Ausführungsstrategie und -plattform Prozesse direkt oder indirekt mit diesen Sprachen definieren. Wie im vorigen Abschnitt beschrieben, bleiben sowohl die unterschiedlichen Ausführungsstrategien einer Modellinterpretation und einer Codegenerierung als auch die Ausführungsplattformen vor dem Modellierer verborgen.

Für BPEL als etablierten Standard existiert bereits eine Vielzahl an Werkzeugen zur grafischen Modellierung. Abbildung 5.4a zeigt, wie der passive Überwachungsprozess aus der Evaluation (s. Abschnitt 5.6.1) mit dem BPEL-Editor der Entwicklungsumgebung *NetBeans* [150] modelliert wird. BPEL wird jedoch auch indirekt zur Prozessdefinition verwendet. Wie in Abschnitt 2.3.3 beschrieben, existieren für viele fachlich-konzeptionelle Modellierungssprachen wie BPMN oder EPK Werkzeuge, mit denen diese Modelle in einem Modell-zu-Modellkonvertierungsvorgang nach BPEL überführt und so in BPEL-Laufzeitumgebungen ausgeführt werden können. Das bietet den Vorteil, dass bei der Modellierung Sprachen verwendet werden können, die enger an der Domäne orientiert sind und technische Aspekte stärker verbergen. Auch werden Aspekte der Dienstchoreographie stärker berücksichtigt. Mit BPMN ist es z. B.

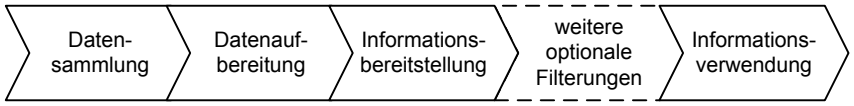


Abbildung 5.5: Schritte der Überwachung von Geschäftsprozessen in IWFMS

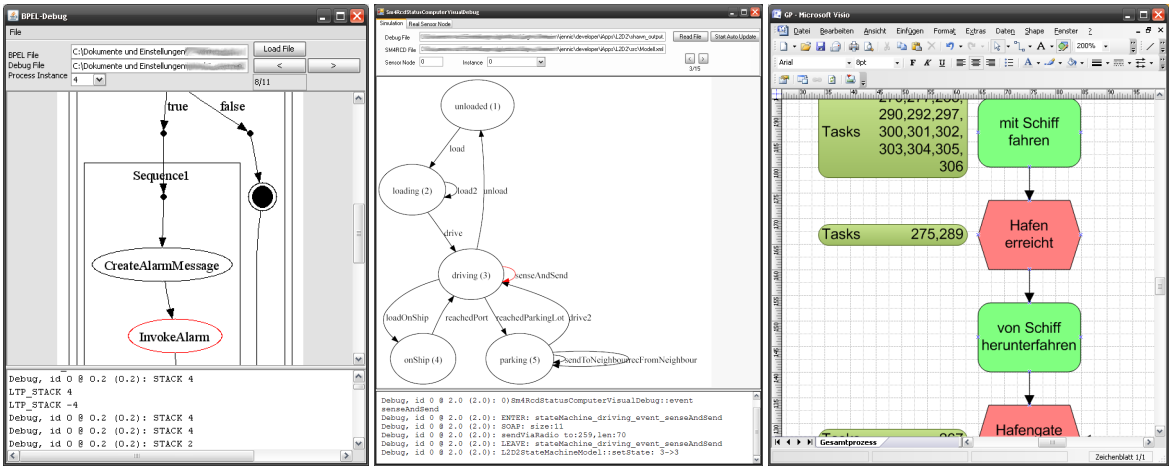
nicht nur möglich, einen Prozess, sondern auch das Zusammenspiel zwischen verschiedenen Prozessen zu definieren. So kann das Gesamtsystem in einem Modell beschrieben werden, aus dem dann automatisch verschiedene BPEL-Prozesse generiert werden, die auf unterschiedlichen Systemkomponenten ausgeführt werden. Gleichzeitig können diese Modelle als Teil einer auf Webservices basierenden SOA informationstechnisch ideal umgesetzt werden.

Da SM4RCD erst im Rahmen dieser Arbeit entstand, musste im Gegensatz zu BPEL auch ein Editor zur grafischen Prozessmodellierung implementiert werden. Während in einer ersten Version der Zustandsautomateneditor aus der Entwicklungsumgebung *Visual Studio* [133] verwendet und um SM4RCD-Elemente erweitert wurde, wird die aktuelle Version (s. Abbildung 5.4b) auf Basis von EMF (*Eclipse Modeling Framework*, [48]) und GMF (*Graphical Modeling Framework*, [49]) entwickelt. Sie ermöglicht eine Modellierung in der Entwicklungsumgebung *Eclipse* [47]. Obwohl SM4RCD mit diesen Editoren eine grafische Modellierung erlaubt, stellen die Modelle letztlich XML-Dokumente dar. Damit kann SM4RCD genau wie BPEL als Zielsprache eines Modell-zu-Modellkonvertierungsvorgangs dienen.

5.4.3 Prozessüberwachung

Neben der Ausführung und Modellierung ist die Überwachung eine weitere wichtige Aufgabe eines Geschäftsprozessmanagements. Die Überwachung beinhaltet sowohl die fachliche als auch die technische Prozessüberwachung während der Entwicklungs- und Testphase sowie während des Produktionsbetriebs.

Aufgrund der unterschiedlichen Ausführungsplattformen, Realisierungssprachen sowie Programmen zur Anzeige und Auswertung von Überwachungsinformationen wird im Rahmen dieser Arbeit ein Ansatz empfohlen, der eine schrittweise und modulare Verarbeitung der Überwachungsdaten vorsieht. Abbildung 5.5 zeigt die verschiedenen Verarbeitungsschritte. Jeder Arbeitsschritt kann über verschiedene Werkzeuge durchgeführt werden. Im ersten Schritt müssen die Überwachungsdaten gesammelt werden. Im Fall der Prozessausführung im Simulator oder in einem Interpreter im Backend werden diese Daten in der Regel einfach in einer Textdatei gespeichert. Bei der Ausführung auf einem Knoten können diese Daten entweder über eine serielle Verbindung oder über das Netz mit LTP ausgetauscht werden. Dabei ist jedoch zu berücksichtigen, dass durch die Erzeugung und das Versenden von Überwachungsinformationen durch einen Sensorknoten für diesen und zumindest für Teile des Sensornetzes ein wesentlicher Overhead entstehen kann. Als Überwachungsinformationen können optional je nach Ausführungsplattform und Realisierungssprache Informationen über den allgemeinen Prozesszustand bzw. -verlauf automatisch von der Laufzeitumgebung erzeugt werden. Zusätzlich ist es möglich, dass der Modellierer eigene Überwachungsinformationen für die Ausgabe während der Prozessausführung definiert. Diese können eine beliebige textuelle Struktur aufweisen.



(a) BPEL4WSN-Debug (b) SM4RCD-Debug (c) Visio

Abbildung 5.6: Überwachung von Geschäftsprozessen in IWFMS

Im zweiten Schritt werden die Rohdaten aufbereitet. Es werden die laufzeitspezifischen Informationen herausgefiltert und mit den vom Modellierer definierten Informationen in eine zeitlich logische Beziehung gesetzt. So wird z. B. für einen SM4RCD-Prozess ermittelt, wann der Prozess über welche Transitionen in welche Zustände überführt wurde, welche Ereignisse die Transitionen ausgelöst haben und welche Aktionen ausgeführt wurden. Zusätzlich werden die Ausgaben der modelliererspezifschen Informationen zeitlich und logisch entsprechenden Zuständen und Transitionen zugeordnet. Bei BPEL erfolgt diese Analyse analog in Bezug auf die Abfolge der Ausführung der BPEL-Aktivitäten.

Über einen Bereitsteller werden die aufbereiteten Informationen den Programmen zur fachlichen und technischen Überwachung bereitgestellt. Dieses können sowohl externe Werkzeuge wie z. B. *Visio* [132] oder *Excel* [131] als auch mit *BPEL4WSN-Debug* und *SM4RCD-Debug* spezielle im Rahmen dieser Arbeit entwickelte Programme sein. Während BPEL4WSN-Debug und SM4RCD-Debug alle Verarbeitungsschritte in einer zusammenhängenden Applikation vereinen, erfolgt die Anbindung externer Werkzeuge über eine Webservice-, REST- und Dateischnittstelle. Optional können auch weitere, benutzerspezifische Filter zwischen die Datenbereitstellung und -verwendung geschaltet werden.

Die Abbildungen 5.6a und 5.6b zeigen die Überwachung eines BPEL- und SM4RCD-Prozesses in BPEL4WSN-Debug und SM4RCD-Debug. Über die grafische Anzeige kann der Prozesszustand bzw. die aktuelle Aktivität angezeigt werden. Das jeweilige Textfenster zeigt die textuellen Ausgaben des Prozesses in diesem Zustand bzw. der aktuellen Aktion. Über Steuerknöpfe kann sich der Benutzer durch den Prozessablauf „klicken“. Abbildung 5.6c zeigt die Überwachung des Gesamtprozesses des Transports von Gefahrgütern (s. Abschnitt 5.6.1) in Visio. Der Prozess wird als EPK dargestellt und zeigt, welche Prozessinstanzen (über die ID des Transportauftrags identifiziert) sich an welcher Stelle im Prozessablauf befinden.

5.5 Realisierung der IWFMS-Prozessausführung

Wie bereits in Abschnitt 5.4 beschrieben, stellt die Ausführung von Geschäftsprozessen die zentrale Forschungsherausforderung bei der Umsetzung von IWFMS dar. Zur ganzheitlichen Ausführung von Prozessmodellen wurden im Rahmen dieser Arbeit ein Interpreter zur Ausführung von BPEL-Prozessen sowie je ein Codegenerator für die Sprachen BPEL und SM4RCD entwickelt. Während der BPEL-Interpreter auf die Ausführung von Prozessen im Backend beschränkt ist, lässt sich der Ausgabecode der Generierungsvorgänge sowohl auf Sensorknoten als auch auf Backend-Systemen ausführen.

5.5.1 Interpretation von Prozessmodellen

Wie bereits beschrieben, stellt zur Realisierung des Konzepts dieser Arbeit bei der Ausführung von Prozessmodellen im Interpreter auf Backend-Systemen lediglich die Umsetzung einer ganzheitlichen Webservice-Kommunikation die zu erfüllende Herausforderung dar. Mit Apache ODE (*Apache Orchestration Director Engine*, [10]) existiert eine quelloffene Laufzeitumgebung zur Ausführung von BPEL-Prozessmodellen auf PC-Hardware. ODE verwendet zur Realisierung der Webservice-Kommunikation zwischen den in ODE ausgeführten Prozessen und ihren Kommunikationspartnern das Framework Axis2.

Wie in Abschnitt 4.5.6 beschrieben, wurde LTP+SMC vom Autor dieser Arbeit in Axis2 integriert. Wird ODE zusammen mit dieser Implementierung verwendet, kann ein BPEL-Prozess neben den zahlreichen standardmäßig von Axis2 bereitgestellten Bindings gleichzeitig auch LTP+SMC als Kommunikationsbasis verwenden und so transparent mit beliebigen Endpunkten im Gesamtsystem interagieren.

Im Rahmen dieser Arbeit wurde LTP+SMC zusätzlich in die Windows Workflow Foundation integriert. Damit können Backend-Prozesse auch mit allen Sprachen dieses Frameworks beschrieben werden. Da diese Prozesse jedoch ausschließlich im Backend ausgeführt werden können und so keine ganzheitlich transparente Modellierung realisiert werden kann, wird die Windows Workflow Foundation in dieser Arbeit nicht näher betrachtet. Für Details sei auf eine frühere Publikation vom Autor dieser Arbeit verwiesen [69].

Bei der Integration beliebiger Dienste des Gesamtsystems in Geschäftsprozesse, die im Backend ausgeführt werden, genügt das Hinzufügen von LTP+SMC als Binding in das von der Laufzeitumgebung verwendete Webservice-Framework. Dasselbe Prinzip ist auch bei weiteren auf einer Webservice-Kommunikation aufbauenden Standards, wie z. B. WS-HumanTask, anwendbar. Mit dem Hinzufügen von LTP+SMC als zusätzliches Binding in schwergewichtige Enterprise-Service-Busse ist auch die Integration aller LTP+SMC-Dienste in weitere Laufzeitumgebungen von Standards, die nicht direkt auf Webservices aufbauen, möglich. Im Umfeld von Sensornetzen sind insbesondere Sprachen im Rahmen des *Complex Event Processing* [120] interessant, um eine einfache Verarbeitung kontinuierlicher, komplexer Datenströme zu realisieren.

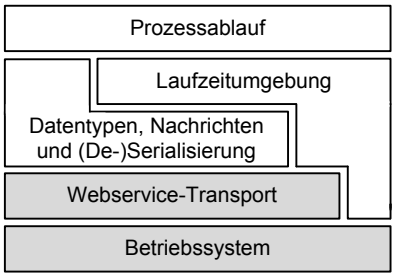


Abbildung 5.7: Aufbau eines Codegenerats zur Ausführung von Geschäftsprozessmodellen

5.5.2 Codegenerierung aus Prozessmodellen

Die große Herausforderung bei der Umsetzung von IWFMS stellt die Realisierung der Codegeneratoren dar. Bei der Codegenerierung muss Code zur Abbildung der Datentypen und Nachrichten sowie zu deren (De-)Serialisierung, Code zur Abbildung des Prozessablaufs und der jeweiligen Laufzeitumgebung erstellt werden. Gleichzeitig dürfen die extrem restriktiven Vorgaben von WSNs bzgl. des Ressourcenverbrauchs nicht verletzt werden.

Abbildung 5.7 stellt die Struktur des Codegenerats dar. Sie ist grundsätzlich bei SM4RCD und BPEL gleich und unterscheidet sich lediglich in den Umsetzungsdetails der Laufzeitumgebungen und der Prozessabbildungen. Die weiß dargestellten Elemente beschreiben die vom Codegenerator dynamisch erzeugten Komponenten, während die statisch verknüpften Bausteine grau hinterlegt wurden. Die statischen Komponenten sind zur Ausführung des Generats notwendig. Sie sind jedoch nicht modellspezifisch. Zur Ausführung eines Codegenerats werden als statische Komponenten ein Betriebssystem sowie ein Framework zur Umsetzung der Webservice-Kommunikation zwischen dem Prozess und seinen Kommunikationspartnern benötigt. Zurzeit wird iSense als Betriebssystem für Sensorknoten unterstützt und ermöglicht neben der Ausführung der Prozessmodelle auf den Koten auch ihre Simulation im Simulator *Shawn* [105]. Neben Sensorknoten als Ausführungsplattformen kann das Generat auch in Backend-Systemen auf Windows und Linux als Betriebssystem ausgeführt werden. Die Anforderungen einer transparenten Webservice-Kommunikation im Gesamtsystem wird zurzeit ausschließlich vom Transport-Binding LTP+SMC erfüllt. Deshalb wird als entsprechendes Framework die in Abschnitt 4.5.6 vorgestellte C++-Implementierung von LTP+SMC für ressourcenbeschränkte Geräte verwendet.

Das eigentliche Codegenerat setzt sich aus den dynamischen Komponenten zur Abbildung des Prozessablaufs, der Laufzeitumgebung sowie zur Abbildung und (De-)Serialisierung der Datentypen und Webservice-Nachrichten zusammen. Da sich die Generierung der Datentypen, Nachrichten und Serialisierungsfunktionalität nicht zwischen SM4RCD und BPEL unterscheidet, kann auch in beiden Fällen derselbe Codegenerator verwendet werden. Sowohl bei BPEL als auch bei SM4RCD kommt die in Abschnitt 4.4 beschriebene SMC-Codegenerierung zum Einsatz. Sie übersetzt die aus dem Prozessmodell referenzierten und in einem WSDL- oder XML-Schema-Dokument spezifizierten Daten- und Nachrichtentypen in C-Strukturen und erzeugt zusätzlich C-Code zur entsprechenden (De-)Serialisierung.

Im Gegensatz dazu ist die Codegenerierungsfunktionalität zur Erzeugung der Laufzeitumgebung sowie zur Abbildung des Prozessablaufs abhängig von der verwendeten Prozessreali-

sierungssprache. Die Laufzeitumgebung steht im Zentrum des Codegenerats. Sie verwaltet die Prozessinstanzen und koordiniert die Interaktionen zwischen dem Prozessablauf und dem Webservice-Transport. Sie wird genauso wie der Code zur Beschreibung des Prozessablaufs mit dem Generator der jeweiligen Prozessrealisierungssprache aus einem BPEL- oder SM4RCD-Dokument generiert.

Zur Abbildung des Prozessablaufs wird C++-Code erzeugt, der die Prozesslogik abbildet und ihre direkte Ausführung ermöglicht. Dabei interagiert der Prozessfluss mit der Laufzeitumgebung und verwendet den im Rahmen der SMC-Codegenerierung erzeugten Code zur Abbildung der Daten- und Nachrichtentypen sowie zur (De-)Serialisierung der Webservice-Nachrichten. Während SM4RCD-Modelle lediglich Elemente dieser Sprache enthalten können, die übersetzt werden müssen, können BPEL-Modelle neben Elementen dieser Sprache auch Anweisungen enthalten, die mithilfe anderer Standards formuliert wurden. Dieses sind insbesondere XPath- und XSLT-Ausdrücke[1]. Zur Abbildung der Prozesslogik werden alle Elemente des jeweiligen Modells in C++-Code überführt. Dabei wird sowohl die eigentliche Logik abgebildet als auch Code erzeugt, der die Ausführung der Logik beschreibenden Sprachkonstrukte umsetzt. Das Generat ist also nicht nur eine Repräsentation eines BPEL- oder SM4RCD-Modells in C++. Es enthält zusätzlich Code, der bei einer Modellinterpretation Teil des Interpreters wäre.

Das größte Problem bei der Abbildung der Prozesslogik ist die Behandlung von Parallelität. Während Parallelität bei SM4RCD lediglich die Ausführung paralleler Prozessinstanzen sowie die Verwaltung von Zeitgebern betrifft, unterstützt BPEL zusätzlich eine Reihe von Sprachkonstrukten, die eine massive Parallelität von Abläufen innerhalb einzelner Instanzen erlauben. Dabei können sehr komplexe Abhängigkeiten bzgl. der Synchronisierung paralleler Flüsse entstehen. Problematisch ist in diesem Zusammenhang, dass WSN-Plattformen in der Regel weder die Ausführung paralleler Prozesse (engl.: *Multitasking*) noch die Ausführung paralleler Abläufe innerhalb eines Prozesses (engl.: *Multithreading*) unterstützen. Als Folge muss diese Funktionalität vom Codegenerator als Teil des Generats realisiert werden.

Aufgrund des fehlenden Multithreadings müssen die Prozessabläufe aus der Generatorperspektive sequenziell ausgeführt werden. Um trotzdem eine softwareseitige Quasiparallelität zur erreichen, müssen die Prozessflüsse eines BPEL- bzw. der Zustandsautomat eines SM4RCD-Modells bei der Codegenerierung in Teilprozesse zerlegt werden. Diese müssen dann sequenziell ausgeführt werden. Dabei sollen möglichst keine gegenseitigen Blockierungen und minimale Wartezeiten für die einzelnen Prozessinstanzen sowie eine faire Bearbeitung aller Instanzen realisiert werden.

Ein besonderes Charakteristikum von Geschäftsprozessen ist die Existenz von oft sehr langen Wartezeiten innerhalb des Prozessablaufs. Diese sind darauf zurückzuführen, dass Prozesse in der Regel aus einer Folge an relativ kurz laufenden Aktionen bestehen, die an bestimmten Stellen durch lang laufende Aktionen sowie durch das Warten auf das Eintreten von Ereignissen unterbrochen werden. Auf der technischen Ebene werden in diesem Zusammenhang in dieser Arbeit mit blockierenden und nicht blockierenden Elementen zwei Arten von Sprachkonstrukten bei SM4RCD und BPEL unterschieden. Nicht blockierende Elemente, wie z. B. das Kopieren von Daten, weisen meist nur eine sehr geringe Ausführungsdauer auf. In

[1] XSLT wird zurzeit noch nicht vom Codegenerator unterstützt.

jedem Fall ist die Dauer auf die Ausführung dieser Aktivität beschränkt und enthält keine Wartezeiten, in denen der Prozess nicht weiter ausgeführt wird.

Im Gegensatz zu nicht blockierenden Elementen zwingen blockierende Elemente den Prozessablauf dazu, mit der Fortführung zu warten, bis ein bestimmtes Ereignis eintritt. Das kann z. B. das Eintreffen einer Nachricht oder der Ablauf eines Zeitgebers sein. In BPEL sind das die Ereignisse *receive* und *pick*. Aber auch Aktionen wie *wait* und *invoke* können blockierend sein. Wird *invoke* in einer synchronen Kommunikation verwendet, blockiert die Prozessinstanz an dieser Stelle bis zum Eintreffen der Antwortnachricht. Bei SM4RCD als ereignisgesteuerten Zustandsautomaten blockiert eine Instanz in jedem Zustand bis zum Eintreten eines Ereignisses. Damit stellen alle SM4RCD-Ereignisse blockierende Elemente dar. Analog zu *invoke* als BPEL-Sprachkonstrukt blockiert auch *InvokeWebServiceAction* im Fall einer synchronen Kommunikation.

Bei der Umsetzung von Parallelität bei der Ausführung von Geschäftsprozessen auf Sensorknoten sind blockierende Sprachkonstrukte das Problem und gleichzeitig auch die Lösung. Ohne die Unterstützung von Multithreading durch das Betriebssystem verursachen blockierende Elemente Wartezeiten und verhindern sowohl die Ausführung von parallelen Prozessen und Prozessinstanzen als auch die von parallelen Flüssen innerhalb derselben Instanz. Allerdings stoppt jedes blockierende Element eine Prozessinstanz genau solange, bis ein bestimmtes Ereignis eintritt. Die grundlegende Idee ist es, dass ein Prozess an den blockierenden Elementen so in verschiedene Subprozesse aufgeteilt wird, dass jeder Subprozess mit dem Warten auf ein Ereignis, das die entsprechende Blockierung auflöst, beginnt. Alle folgenden Elemente müssen ausschließlich nicht blockierend sein. Wird nun bei der Abarbeitung der Subprozesse ein ereignisgesteuerter Ansatz verfolgt, kann eine Quasiparallelität realisiert werden. In diesem Ansatz verwaltet eine zentrale Komponente zur Ereignissteuerung eintreffende Ereignisse und dass bestimmte Subprozesse auf das Eintreten eines oder mehrerer Ereignisse warten. Tritt ein Ereignis ein, dann wird der entsprechende Subprozess ausgeführt. Da das Ereignis zur Auflösung der Blockierung bereits eingetreten ist und der Subprozess im Weiteren nur über nicht blockierende Elemente verfügt, ist seine Ausführungsdauer relativ gering, sodass weitere Subprozesse paralleler Instanzen oder Flüsse ohne lange oder zumindest ohne unnötige Wartezeiten abgearbeitet werden können.

In den folgenden beiden Abschnitten wird als Erstes erläutert, wie mithilfe einer ereignisgesteuerten Prozessausführung Parallelität erreicht werden kann, bevor gezeigt wird, wie BPEL- und SM4RCD-Modelle partitioniert werden müssen, um ereignisgesteuert ausgeführt werden zu können.

5.5.3 Ereignisgesteuerte Prozessausführung

Der Ansatz einer ereignisgesteuerten Ausführung paralleler Prozessabläufe ist in Abbildung 5.8 dargestellt. Er sieht die Definition verschiedener Ereignisabarbeiter vor. Diese werden bei einem zentralen Ereignisverteiler auf bestimmte Ereignisse registriert und bei deren Eintreten ausgeführt. Zur Auslösung von Ereignissen sind mit einem Zeitgeber, einem Code- bzw. Methodenaufruf sowie einer eingehenden Webservice-Nachricht die drei möglichen Ereignistypen von BPEL und SM4RCD vorgesehen.

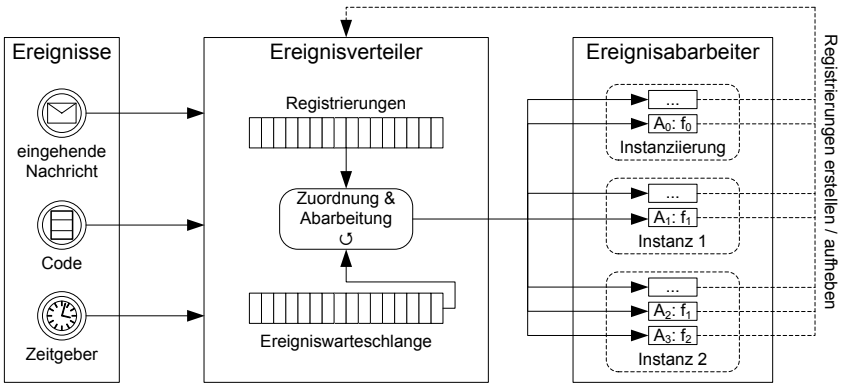


Abbildung 5.8: Ereignisgesteuerte Abarbeitung von parallelen Prozessabläufen

Ereignisse können zu jeder Zeit eintreffen. Diese werden in einer Warteschlange für die spätere Zuordnung zu einem Ereignisabarbeiter und dessen Ausführung zwischengespeichert. Um eine faire Abarbeitung zu garantieren, folgt die Warteschlange dem FIFO-Prinzip (engl.: *First In, First Out*). Die Abarbeitung der Warteschlange erfolgt kontinuierlich und sequenziell. Dabei wird überprüft, ob zu dem jeweils ersten Ereignis eine entsprechende Registrierung eines Ereignisabarbeiters existiert. Ist das der Fall, wird der Abarbeiter ausgeführt. Andernfalls wird das Ereignis verworfen. In jedem Fall werden das Ereignis aus der Warteschlange und (sofern vorhanden) die Registrierung aus der entsprechenden Liste entfernt.

Ereignisabarbeiter stellen einen Teilfluss eines Geschäftsprozesses dar. Die Zuordnung zwischen Ereignis und auszuführenden Teilprozess erfolgt auf der Ebene der Prozessinstanzen. Das bedeutet, dass z. B. eine eingehende Webservice-Nachricht, die eine bestimmte Operation eines Endpunkts adressiert, nicht generell an einen Teilprozess, der diese Nachricht verarbeitet, gerichtet ist. Stattdessen wird diese Subprozessfunktionalität für die entsprechende Instanz ausgeführt. Derselbe Teilprozess kann gleichzeitig für verschiedene Instanzen als Abarbeiter beim Ereignisverteiler registriert sein. Eine Ereignisregistrierung beschreibt also die Kombination aus logischem Ereigniszeitpunkt, Ereignistyp, Ereignisinhalt (z. B. Teile einer konkreten Webservice-Nachricht) sowie die Zuordnung zu einer Funktionalität und Instanz eines Ereignisabarbeiters.

Bei der Initialisierung eines Prozesses wird als Erstes ein oder mehrere Ereignisabarbeiter zur Prozessinstanziierung beim Ereignisverteiler registriert. Sie dienen als Einstiegspunkt in einen Prozess und reagieren auf instanziierende Webservice-Nachrichten oder Codeereignisse (nur SM4RCD) und arbeiten den ersten Subprozess einer Geschäftsprozessinstanz ab. Vor seiner Beendigung erstellt ein instanziierender Abarbeiter für jeden im Prozessverlauf folgenden Teilprozess und die entsprechenden auslösenden Ereignisse für diese Instanz eine Registrierung beim Ereignisverteiler. Um die Instanziierung weiterer Prozessinstanzen zu ermöglichen, wird auch dieser Abarbeiter erneut registriert. Werden nicht nur Ereignisabarbeiter zur Instanziierung eines Prozesses, sondern unterschiedlicher Prozesse registriert, ist auch die parallele Ausführung verschiedener Geschäftsprozesse auf einem Sensorknoten möglich.

Nach demselben Prinzip wie die instanziierenden funktionieren auch nicht instanziierende Ereignisabarbeiter. Bei der Ausführung heben sie die Registrierungen von durch das Eintreten des auslösenden Ereignisses nicht mehr mögliche oder verbotene Ereignisse auf und arbeiten

den entsprechenden Teilprozess für die Instanz ab. Bevor die Ereignisabarbeiter terminieren, führen sie noch die Registrierung aller im Gesamtprozessfluss direkt folgenden Teilprozesse und deren auslösende Ereignisse durch.

Mit dieser ereignisgesteuerten Prozessausführung kann Parallelität bei der Ausführung von verschiedenen Prozessen, Instanzen sowie Abläufen innerhalb einer Instanz realisiert werden. Dabei steuert die ereignisgesteuerte Prozessausführung die Abarbeitung der Logik auf Basis von Ereignissen und registrierten Ereignisabarbeitern. Da sowohl die Registrierung von Ereignissen als auch deren Aufhebung von den Ereignisabarbeitern selbst vorgenommen wird, steuern diese eigenständig die eigentlichen Prozessflüsse sowie die Verwaltung von Prozessinstanzen. Diese Aufgaben bleiben der ereignisgesteuerten Prozessausführung verborgen.

5.5.4 Partitionierung der Prozessmodelle

Zur Verwendung der im vorigen Abschnitt beschriebenen ereignisgesteuerten Prozessausführung muss ein Prozessmodell bei der Codegenerierung, wie in Abschnitt 5.5.2 beschrieben, in Subprozesse bzw. Prozessflüsse zerlegt werden. Die einzelnen Subprozesse stellen Funktionseinheiten dar, die mit einem logischen Ereigniszeitpunkt, -typ und -inhalt sowie mit einer Instanz verknüpft, einen Ereignisabarbeiter der ereignisgesteuerten Prozessausführung darstellen. Sie enthalten neben der eigentlichen Prozessfunktionalität auch Code zur Registrierung neuer Ereignisabarbeiter und der Aufhebung von Registrierungen, deren Ausführung im Gesamtprozess durch die Abarbeitung des aktuellen Subprozesses nicht mehr möglich ist. Befindet sich z. B. eine SM4RCD-Instanz in einem bestimmten Zustand, dann existieren zu diesem Zeitpunkt für jede mögliche Transition eine Registrierung eines Subprozesses als Abarbeiter beim Ereignisverteiler. Löst ein Ereignis nun eine der Transitionen aus, wird der Knoten in einen neuen Zustand überführt. Entsprechend dürfen die restlichen Transitionen des Ausgangszustands nicht mehr ausgeführt werden, sodass die entsprechenden Registrierungen entfernt werden müssen. Gleichzeitig müssen jedoch alle Subprozesse, die die Logik der möglichen Transitionen des neuen Zustands realisieren mit den entsprechenden auslösenden Ereignissen registriert werden. Auf diese Weise steuern die einzelnen Subprozesse den Gesamtprozessfluss.

SM4RCD

SM4RCD-Prozessmodelle weisen von vornherein eine Struktur auf, die ideal für eine ereignisgesteuerte Prozessausführung ist. Jede Instanz wartet in einem Zustand auf das Eintreten eines Ereignisses, das die Ausführung einer der möglichen Transitionen auslöst. Bei jeder Transition, die von einem Ereignis e ausgelöst wird und den Automaten vom Zustand s in einen neuen Zustand t überführt, werden die mit e verknüpfte Aktion sowie die Finalisierungs- und Initialisierungsaktionen f_s bzw. i_t von s und t ausgeführt. Eine direkte Übertragung dieser Transitionen in Subprozesse für Ereignisabarbeiter wird allerdings von *InvokeWebServiceAction* verhindert, da dieser Aktionstyp (als Einziger) blockierend sein kann.

Algorithmus 5.1 Partitionierung von SM4RCD-Prozessmodellen

Input: The SM4RCD process model $P := (S, E, A, T)$ where S is the set of states, E the set of events, A the set of actions and T is the set of transitions of P.

Output: A set of process flows F. Each flow $f \in F$ represents a transition of SM4RCD triggered by an event which can be executed by the event-driven process execution.

1: **for all** $a \in A$ **do**
2: **if** type of a is a synchronous *InvokeWebServiceAction* **then**
3: Split a into $(a_{\text{send}} \rightarrow s_{\text{helper}} \overset{e_{\text{receive}}}{\rightarrow})$ where a_{send} sends the outgoing message of a, s_{helper} is a new helper state with e_{receive} as new event triggered when receiving the answer of a_{send}.
4: Add s_{helper} to a set of helper states S_{helper}, a_{send} to a set of send actions A_{send}, a to a set of blocking actions A_{blocking} and e_{receive} to a set of helper events E_{helper}.
5: **end if**
6: **end for**
7: Remove all $a \in A_{\text{blocking}}$ from original linked states or events.
8: **for all** $e \in E \cup E_{\text{helper}}$ **do**
9: Let $s \in S \cup S_{\text{helper}}$ be the source state of the transition triggered by e and $t \in S \cup S_{\text{helper}}$ its target state.
10: Let $E_s \subseteq E \cup E_{\text{helper}}$ be the events triggering all outgoing transitions of s.
11: Let $(s \overset{e}{\rightarrow} a_e \rightarrow f_s \rightarrow i_t \rightarrow t) \in T$ be the transition triggered by e transferring s to t while executing the actions $a_e, f_s, i_t \in ((A \backslash A_{\text{blocking}}) \cup A_{\text{send}})$ linked to e, s (finalize) and t (initialize).
12: Add $f := (\overset{e}{\rightarrow} unregisterEvents(E_s \backslash \{e\}) \rightarrow a_e \rightarrow f_s \rightarrow i_t \rightarrow setState(t) \rightarrow registerAllEvents(t))$ to F.
13: **end for**

Algorithmus 5.1 zeigt die Überführung eines Gesamtprozessmodells von SM4RCD in eine Menge Subprozesse bzw. Prozessflüsse, die die Funktionalität von Ereignisabarbeitern repräsentieren. Bevor der Gesamtprozess partitioniert werden kann, müssen alle Blockierungen entfernt werden. Dazu werden alle Aktionen vom Typ *InvokeWebServiceAction*, die synchrone Nachrichten austauschen, wie in den Zeilen 1 bis 7 beschrieben, aufgespalten. Von der Aktion a wird im ersten Schritt nur das Versenden der Anfragenachricht a_{send} ausgeführt. Es wird ein neuer Hilfszustand s_{helper} eingefügt, in den der Prozess nach der Ausführung von a_{send} überführt wird. Der Empfang der Antwortnachricht von a wird als Ereignis e_{receive} realisiert, das die Fortführung der ursprünglichen, nun aufgespaltenen Transition auslöst.

Nach der Eliminierung aller blockierenden Aktionen können die Transitionen direkt in einzelne Subprozesse überführt werden (Zeile 8 bis 13). Sie werden nun durch ein Ereignis ausgelöst und führen ausschließlich nicht blockierende Aktionen aus. Am Anfang und Ende eines solchen Subprozesses wird noch Code zur Registrierung von neuen Ereignissen beim Ereignisverteiler bzw. zu deren Aufhebung erstellt und so der weitere Prozessverlauf gesteuert.

BPEL

Im Gegensatz zur Sprache SM4RCD mit ihren wenigen einfachen Aktionen und ihrem ohnehin ereignisgesteuerten Prozessablauf ist der Partitionierungsaufwand bei BPEL bedeutend höher. Sowohl die Anzahl als auch der Komplexitätsgrad der Sprachkonstrukte von BPEL ist wesentlich höher als bei SM4RCD. Durch eine nahezu beliebige rekursive Verschachtelung der Elemente sowie durch die Realisierung paralleler Prozessabläufe innerhalb von Prozessmodellen nimmt ihre Komplexität weiter zu.

Algorithmus 5.2 fasst den Partitionierungsvorgang eines BPEL-Prozessmodells zusammen. Analog zu SM4RCD müssen in einem ersten Transformationsschritt alle *invoke*-Anweisungen zum synchronen Webservice-Aufruf in ein asynchrones Senden der Anfrage gefolgt von einem asynchronen Empfang der Antwortnachricht mit *receive* ersetzt werden (Zeile 1). Zur

Algorithmus 5.2 Partitionierung von BPEL-Prozessmodellen

Input: A BPEL process to partition.
Output: The partitioned process as C++-Code which can be executed by the event-driven process execution.
1: Replace every synchronous *invoke* by an asynchronous *invoke* followed by a *receive*.
2: Go recursively through sub trees and apply the following rules:

receive	Register event to trigger execution of new sub tree f_{new}. End current sub tree f_{cur}. Begin one new sub tree f_{new} with remaining actions of f_{cur}. Insert code into f_{new} to handle messages.
pick	Register an event for each pick-event $e \in E$ to trigger execution of correlating sub tree f_{new}, e. End current sub tree f_{cur}. Begin one new sub tree $f_{\text{new},e}$ for each e and corresponding control flow c_e. Enter deregistration for all $r \in E \backslash \{e\}$ and corresponding $f_{\text{new},r}$ at beginning of each $f_{\text{new},e}$. Create sub tree f_{remain} containing all elements of f_{cur} after *pick*; register immediate execution of f_{remain} in each $f_{\text{new},e}$.
wait	Register timer event with duration defined in *wait* to trigger execution of f_{new}. End current sub tree f_{cur}. Begin one new sub tree f_{new} with remaining actions.
flow or *foreach* ($\|\|$)	Register an immediate timer event for each parallel flow $p \in P$ triggering the execution of $f_{\text{new},p}$. End current sub tree f_{cur}. Begin one new sub tree $f_{\text{new},p}$ for each p containing its control logic c_p. Create a sub tree f_{sync} executing all remaining actions of f_{cur} after *flow* or *foreach* if all $f_{\text{new},p}$ have terminated; register f_{sync} at end of all $f_{\text{new},p}$.
Loops ($\neg \|\|$) containing blocking elements	Split sub tree at every blocking element with each sub tree registering the execution of the succeeding sub tree. The last sub tree registers the execution of the first sub tree. The first sub tree only executes the logic and registers its successor if loop conditions are observed. Otherwise it continues execution of parent tree.
$*$	Generate code to execute the element.

eigentlichen Partitionierung des Prozessmodells müssen die unter Zeile 2 aufgeführten Partitionierungsregeln angewendet werden. Da BPEL selbst eine stark rekursiv verschachtelte Baumstruktur aufweist, müssen auch die Transformationsregeln rekursiv auf den BPEL-Prozess und die generierten Teilprozessflüsse angewendet werden. Während für alle aufgeführten Elemente eine Partitionierung notwendig ist, kann für jedes nicht aufgeführte Element direkt der Code zur Umsetzung der entsprechenden Funktionalität in C++ erstellt werden.

Mit dem Erreichen eines *receive*-Elements wird der aktuelle Teilfluss f_{cur} beendet und ein neuer Teilfluss f_{new} erstellt. Alle vor *receive* liegenden Elemente von f_{cur} bleiben in f_{cur}, während alle auf *receive* folgenden Elemente nach f_{new} verschoben werden. Am Ende von f_{cur} wird das Ereignis, das die Abarbeitung von f_{new} auslöst (also die entsprechende eingehende Nachricht), in Verbindung mit f_{new} beim Ereignisverteiler registriert. Zusätzlich wird am Anfang von f_{new} und Ende von f_{cur} Code zur Verarbeitung der Webservice-Nachrichten eingefügt.

pick beendet ebenfalls den aktuellen Teilfluss f_{cur}. Für jedes *pick*-Ereignis $e \in E$ wird ein neuer Teilfluss $f_{\text{new},e}$ erstellt, der den jeweiligen Kontrollfluss c_e des Ereignisses e enthält. Jeder c_e beginnt mit der Aufhebung der durch die Auslösung von e ausgeschlossenen Registrierungen $E \backslash \{e\}$ und der Verarbeitung von e. In f_{cur} wird die Registrierung aller e und der entsprechenden $f_{\text{new},e}$ erstellt. Mit f_{remain} wird ein zusätzlicher Subbaum erzeugt, der alle hinter *pick* platzierten Elemente von f_{cur} enthält und dessen schnellstmögliche Ausführung (Zeitgeber mit 0 ms Wartezeit) am Ende von jedem $f_{\text{new},e}$ registriert wird.

wait beendet einen Teilfluss f_{cur}, erzeugt genau einen neuen Teilfluss f_{new} und verschiebt die Elemente von f_{cur}, die nach *wait* ausgeführt werden, nach f_{new}. Am Ende von f_{cur} wird ein

Zeitgeberereignis registriert, das nach der von *wait* definierten Wartezeit f_{new} aufruft.

Parallele Teilflüsse (*flow* und paralleles *foreach*) sind selbst nicht blockierend, können aber blockierende Elemente enthalten. Sie beenden ebenfalls einen aktuellen Teilfluss f_{cur}. Dabei wird für jeden parallelen Fluss $p \in P$ ein eigener Subbaum $f_{\text{new},p}$ erstellt, der den entsprechenden Kontrollfluss c_p enthält. Am Ende von f_{cur} werden alle $f_{\text{new},p}$ als Ereignisabarbeiter, die durch einen Zeitgeber mit einer Wartezeit von 0 ms ausgelöst werden, registriert. Zur Synchronisierung aller p wird ein weiterer Teilfluss f_{sync} erstellt und seine schnellstmögliche Ausführung am Ende eines jeden $f_{\text{new},p}$ registriert. f_{sync} überprüft, ob alle $f_{\text{new},p}$ beendet wurden und führt dann alle Prozessschritte von f_{cur}, die nach einem *flow* oder *foreach* definiert wurden, aus.

Enthalten Schleifenkonstrukte blockierende Anweisungen, müssen sie ebenfalls partitioniert werden. Dazu muss rekursiv bei jedem blockierenden Element ein neuer Teilbaum abgespalten werden. Dabei registriert jeweils der Vorgängerteilbaum die Ausführung des Nachfolgers, während der letzte Teilbaum den ersten registriert. Der erste Teilbaum führt die entsprechende Logik nur aus, wenn die Schleifenbedingungen erfüllt sind. Gleiches gilt für die Registrierung des Nachfolgeteilbaums. Ansonsten wird die Ausführung des übergeordneten Teilbaums fortgeführt.

Das Ergebnis der Partitionierung und der entsprechenden Codegenerierung ist eine Menge von Teilprozessabläufen, die von der ereignisgesteuerten Prozessausführung ausgeführt werden können. Genau wie bei SM4RCD steuern sie auch im Fall von BPEL inhärent den Gesamtprozessfluss.

Korrektheit der Partitionierung

Bei der Partitionierung von Prozessmodellen muss gewährleistet werden, dass die erzeugten Teilprozesse im Zusammenspiel dieselbe Funktionalität realisieren, wie das Originalmodell. Die in den vorangegangenen Abschnitten präsentierte Partitionierung von SM4RCD- und BPEL-Modellen basiert darauf, dass an Stellen, an denen ein Prozessfluss blockiert und auf das Eintreten eines oder mehrerer Ereignisse wartet, ein aktueller Teilfluss beendet wird und ein oder mehrere neue Subprozesse erzeugt werden. Diese neuen Subprozesse enthalten die direkt auf das blockierende Element folgenden Anweisungen in derselben zeitlich logischen Folge. Da am Ende des aktuellen Teilprozesses die Ausführung der neuen Teilprozesse mit den auslösenden Ereignissen bei der ereignisgesteuerten Prozessausführung registriert wird und zusätzlich jeder Teilprozess alle Registrierungen von Abarbeitern löscht, die durch seine Ausführung nicht mehr möglich sind, werden die eingefügten Trennstellen indirekt wieder zusammengefügt. So bleibt der Prozessfluss des Originalmodells erhalten. Im Vergleich zu einer direkten, nicht ereignisgesteuerten Ausführung des nicht partitionierten Modells wird dadurch lediglich die Abarbeitung eines Prozessflusses an blockierenden Elementen unterbrochen und der ereignisgesteuerten Prozessausführung die Möglichkeit gegeben, zwischenzeitlich weitere Ereignisabarbeiter auszuführen, anstatt blockierend auf den Ereigniseintritt zu warten. Beim Eintritt eines Ereignisses, das die Blockierung eines Prozessflusses wieder auflöst, wird dieser genau an der Stelle weiter ausgeführt, an der er zuvor unterbrochen wurde.

Damit weisen die Partionierung der Prozesse und ihre ereignisgesteuerte Ausführung bzgl. des Prozessflusses keine Abweichungen im Vergleich zur direkten Ausführung der Originalmodelle

auf. Allerdings kann je nach Systemlast eine Ausführung einzelner Prozessflüsse verzögert werden. Dieses Problem ist jedoch nicht auf den hier verwendeten Ausführungs- und Partitionierungsansatz zurückzuführen. Es besteht bei jeder Art der softwareseitigen Realisierung von Quasiparallelität.

5.6 Analyse der Leistungsmerkmale eines Geschäftsprozessmanagements mit IWFMS

In diesem Abschnitt wird die Leistungsfähigkeit und Praxistauglichkeit von IWFMS nachgewiesen. In diesem Zusammenhang wurde im Rahmen dieser Arbeit der in Abschnitt 3.1 eingeführte Logistikprozess zur Überwachung von Gefahrguttransporten im kombinierten Verkehr mithilfe von IWFMS realisiert.

Viele der Vorteile des Konzepts dieser Arbeit zum ganzheitlichen Geschäftsprozessmanagement wie eine schnelle, einfache und flexible Anpassung von Anwendungen und Prozessen sowie die transparente Integration von Enterprise-IT-Systemen und Sensornetzen sind schwer quantifizierbar. In diesem Abschnitt werden im Rahmen einer quantitativen Evaluation die Vorteile der Nutzung von IWFMS im Entwicklungsprozess anhand der Analyse des Entwicklungsaufwands am Beispiel der Gefahrguttransportüberwachung gezeigt. Gleichzeitig wird nachgewiesen, dass die Ausführung und Interaktion der entsprechenden Workflows extrem effizient ist und problemlos in ressourcenbeschränkten Umgebungen realisiert werden kann.

In Abschnitt 5.6.1 werden die für die folgende Evaluation relevanten technischen Aspekte des Prozesses zum Management von Gefahrguttransporten beschrieben. In den Abschnitten 5.6.2 bis 5.6.5 werden mit der Nachrichten- und Codegröße, dem Arbeitsspeicherverbrauch sowie dem Laufzeitverhalten die für die Ausführung von Geschäftsprozessen mit IWFMS relevanten Kennzahlen diskutiert, bevor in Abschnitt 5.6.6 der Entwicklungsaufwand des Beispielprozesses evaluiert wird.

5.6.1 Beschreibung des Anwendungsfalls

Im Rahmen dieser Arbeit wurde der in Abschnitt 3.1 eingeführte Prozess zur Überwachung von Gefahrguttransporten im kombinierten Verkehr umgesetzt. Abbildung 5.9 beschreibt die Teilprozesse und Dienste zur informationstechnischen Umsetzung des Gefahrgutüberwachungsprozesses aus einer technischen Perspektive. Dabei werden sowohl Orchestrierungs- als auch Choreografieaspekte beschrieben.

Der Hauptprozess h stellt die zentrale Funktionalität zur Überwachung der Gefahrguttransporte dar. Er bildet den kompletten Prozess von der Anmeldung von Gefahrgütern bei den zuständigen Institutionen und der Beladung eines mit einem Sensorknoten ausgestatteten LKW oder Trailers über den Transport auf den jeweiligen Straßenabschnitten, der Abfertigung und Verschiffung des LKW im Hafen bis hin zur Entladung beim Empfänger ab. Dieser Prozess wurde in BPEL modelliert und nutzt das Backend als Ausführungsplattform.

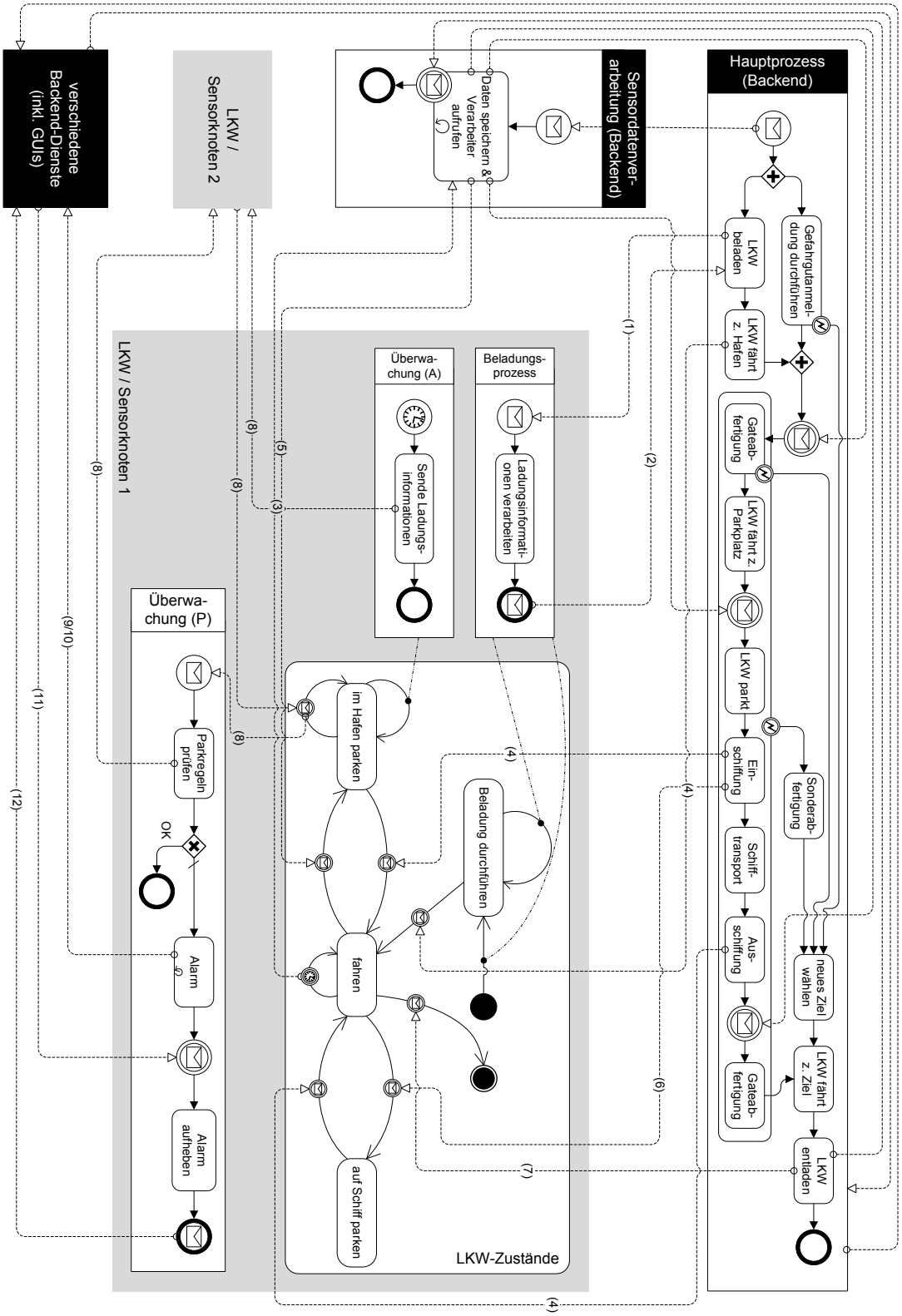


Abbildung 5.9: Technische Betrachtung des Geschäftsprozesses zur Überwachung von Gefahrguttransporten

Der Teilprozess zur Sensordatenverarbeitung s wurde ebenfalls mit BPEL realisiert und wird im Backend ausgeführt. Sein Prozesslebenszyklus ist direkt an den von h gekoppelt. s speichert und analysiert die von den Sensorknoten der LKWs empfangenen Sensor- und Positionsdaten und stellt diese ereignisorientiert über eine *Publish-Subscribe*-Webservice-Schnittstelle weiteren Datenverarbeitern zur Verfügung.

Zusätzlich zu diesen beiden Prozessen werden im Backend weitere Dienste zum Zugriff auf Datenbanken, zur Benutzerinteraktion sowie zur Interaktion mit den Häfen und Reedereien ausgeführt. Diese stehen vor allem mit dem Hauptprozess, z. T. aber auch direkt mit den Sensorknoten in Kommunikationsbeziehungen. Im Rahmen dieser Evaluation werden diese Dienste jedoch nicht analysiert und deshalb in dieser Arbeit auch nicht näher beschrieben.

Auf jedem Sensorknoten werden vier verschiedene Prozesse parallel ausgeführt. Der zentrale Prozess auf einem Knoten ist ein mit SM4RCD modellierter Zustandsautomat z, der die Zustände eines LKW während des kompletten Gefahrguttransports abbildet. Zu Beginn ist der LKW unbeladen. Gesteuert durch h kann z instanziiert und die Beladung durchgeführt werden. Die entsprechenden Transitionen von z werden durch eingehende Webservice-Nachrichten ausgelöst und direkt an den Beladungsprozess weitergeleitet. Mit Abschluss der Beladung fährt der LKW zum Hafen und h überführt z in den entsprechenden Zustand. Während der Fahrt sendet der LKW periodisch die aktuelle Ladungstemperatur und die Position an s. s benachrichtigt h jeweils wenn der LKW das Hafentor und den LKW wenn er innerhalb des Hafens seine Parkposition zum Warten auf die Verschiffung erreicht hat.

Während LKWs im Hafen warten, müssen Gefahrgüter bestimmte Mindestabstände zueinander einhalten. Diese Vorgaben werden durch eine aktive und passive Überwachung der Nachbarschaft eines LKW überprüft. Dazu empfängt ein LKW_1 von einem zweiten LKW_2 dessen Ladungs- und Positionsinformationen, verarbeitet diese mit dem Prozess zur passiven Überwachung (P) u_p und ruft wenn nötig einen Alarmservice im Backend auf. Wurde die Alarmursache behoben, kann das Alarmmanagement den Alarm beim LKW wieder aufheben. Die Nachbarschaftsüberwachung kann parallel erfolgen. z empfängt jeweils Nachrichten, die der Prozess nur asynchron an u_p weiterleitet. Noch vor der Abarbeitung von u_p wird die Transition von z beendet. Für jede Nachricht von einem Nachbarn wird eine neue Instanz von u_p erstellt und parallel ausgeführt. Hat LKW_1 in einem bestimmten Zeitintervall keine entsprechenden Daten erhalten, sendet er aktiv mit dem Teilprozess Überwachung (A) die eigenen Ladungs- und Positionsinformationen per LTP-Broadcast an seine Nachbarschaft.

Der Überwachungsprozess (P) wurde aufgrund seiner höheren Komplexität in BPEL realisiert. Der Beladungsprozess und der Überwachungsprozess (A) können sowohl in BPEL als auch aufgrund ihrer geringeren Komplexität und ihrer Zustandslosigkeit als Teil von z realisiert werden. Während beide Prozesse bei der Umsetzung des Anwendungsfalls Bestandteil von z sind, wurden sie im Rahmen der Evaluation zum Vergleich der Leistungsmerkmale von SM4RCD und BPEL zusätzlich jeweils in diesen beiden Sprachen als eigenständige Prozessmodelle umgesetzt.

Über die Verschiffung, die Abwicklung im Zielhafen, die Fahrt zum Ziel bis hin zur Entladung der Ware beim Empfänger wird z entsprechend des Fortschritts von h benachrichtigt und gesteuert.

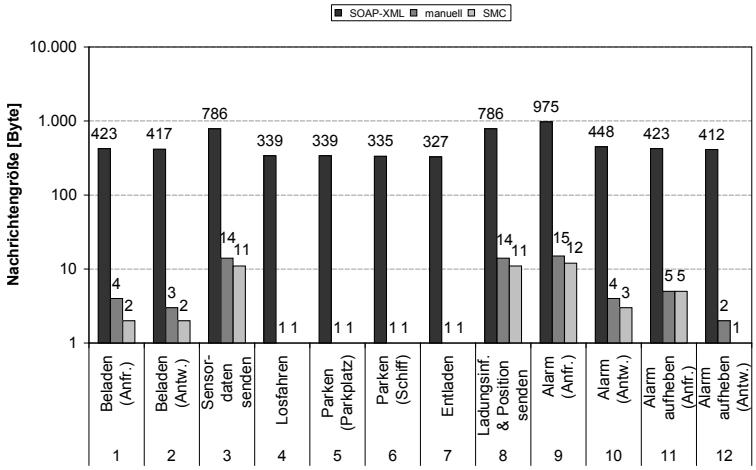


Abbildung 5.10: Vergleich der Nachrichtengrößen einer Gefahrguttransportüberwachung

5.6.2 Evaluation der Nachrichtengröße

Ein entscheidender Erfolgsfaktor bei der Realisierung eines ganzheitlichen Geschäftsprozessmanagements mit IWFMS ist die Kommunikation. Sowohl die Größe der auszutauschenden Nachrichten als auch der zusätzliche Transportprotokoll-Overhead müssen gering sein. IWFMS verwendet LTP+SMC zur ganzheitlichen Webservice-Kommunikation, dessen Kommunikations-Overhead im Folgenden am Beispielprozess zur Überwachung von Gefahrguttransporten analysiert wird.

Bei der Ausführung des in Abbildung 5.9 auf Seite 125 dargestellten Gefahrgutüberwachungsprozesses wird zwischen allen Teilprozessen und Diensten eine Vielzahl an Nachrichten ausgetauscht. Da die Ressourcenbeschränkungen der Sensornetze im Kontext des Anwendungsfalls den limitierenden Faktor darstellen, werden in dieser Evaluation ausschließlich die zwölf Nachrichten betrachtet, bei denen mindestens ein Kommunikationspartner ein Sensorknoten ist. Die Nummern an den Kommunikationspfeilen in Abbildung 5.9 entsprechen den hier betrachteten Nachrichten. Abbildung 5.10 vergleicht die Größen der zwischen den Teilprozessen ausgetauschten SOAP-Nachrichten. Als Bezugspunkte zur Bewertung der Effizienz der von IWFMS verwendeten SMC-Codierung dient die XML-Codierung dieser SOAP-Nachrichten sowie eine manuelle Serialisierung der entsprechenden Datenstrukturen. Im Rahmen dieser Arbeit wird die Annahme getroffen, dass in der Regel aufgrund des erhöhten Aufwands im manuellen Szenario auf eine bitweise Serialisierung verzichtet und byteweise serialisiert wird.

Zur Beladung eines LKW wird die Anfragenachricht 1 mit der Anzahl der zu ladenden Gefahrgüter (ganze 10-Bit-Zahl) und der Gefahrgutklasse (0..15) an den Knoten gesendet, der mit der Anzahl der geladenen Gefahrgüter antwortet (Nachricht 2). Die Größe der komprimierten SOAP-Nachrichten beträgt jeweils 2 Byte. SMC ist damit deutlich kleiner als SOAP-XML (423 bzw. 417 Byte) und sogar kleiner als eine manuelle Serialisierung (4/3 Byte).

Die Nachrichten 4 bis 7 dienen einzig der Änderung der Zustände eines LKW. Als Nutzdaten muss lediglich der Sitzungsschlüssel zur Identifikation der Prozessinstanz übermittelt werden. Da in diesem Anwendungsfall der Zustandsprozess genau eine Instanz ausführt, reicht eine 1-Bit-Zahl zur Codierung des Schlüssels aus. Die gesamte SMC-Nachricht weist eine Größe

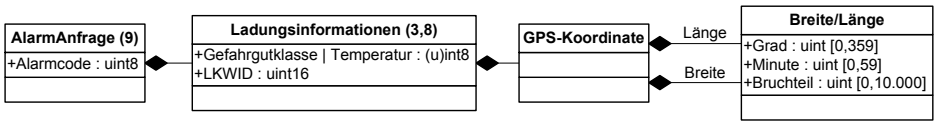


Abbildung 5.11: Datenstruktur der Alarm- sowie Ladungs- und Postionsnachrichten

von einem Byte auf. Während eine manuelle Serialisierung dieselbe Größe erreicht, liegt SOAP-XML mit Nachrichtengrößen zwischen 327 und 339 Byte deutlich darüber.

Wesentlich komplexere Datenstrukturen weisen die Nachrichten zum Versenden der Ladungs- und Positionsdaten (3, 8) sowie zur Meldung von Alarmen (9) auf. Abbildung 5.11 zeigt diese Datenstrukturen. Positionsangaben bestehen aus einem Breiten- und Längengrad mit jeweils drei entsprechend beschränkten ganzen Zahlen. Während der Zustandsprozess als Ladungsinformationen neben der Position die ID des LKW (16-Bit-Zahl) und einen Temperaturwert (8-Bit-Zahl) versendet (3), wird bei der Nachbarschaftsüberwachung die Gefahrgutklasse anstelle der Temperatur übertragen (8). Zusätzlich zu Nachricht 8 enthält die Alarmanfragenachricht 9 noch eine 8-Bit-Zahl als Alarmcode. Die Nachrichten sind mit 11 Byte (3, 8) bzw. 12 Byte (9) mit einer SMC-Serialisierung deutlich kleiner als entsprechende XML-Nachrichten (786/975 Byte). Auch gegenüber einer manuellen Serialisierung (14/15 Byte) ist SMC überlegen.

Als Antwort auf eine Alarmnachricht 10 werden ein Status (1-Bit-Zahl) und die Alarm-ID (16-Bit-Zahl) zurückgegeben. Mit der Nachricht 11, die mit der Alarm-ID und einem Code zwei 16-Bit-Zahlen enthält, wird ein Alarm wieder aufgehoben. Dieser wird mit der Bestätigungsnachricht 12, die einen booleschen Wert überträgt, quittiert. Mit Größen zwischen einem und 5 Byte ist auch für diese Nachrichten SMC effizienter als SOAP-XML (zwischen 412 und 448 Byte) und eine manuelle Serialisierung (zwischen 2 und 5 Byte).

Zusammenfassend kann SMC für die hier aufgeführten Nachrichten mit einer mittleren Kompressionsrate von 99,3 % als deutlich effizienter im Vergleich zu XML-SOAP bewertet werden. Sogar gegenüber einer manuellen Serialisierung ist die Nachrichtengröße von SMC im Mittel 18,4 % geringer. Mit absoluten Größen zwischen 1 Byte und 12 Byte stellen SMC-komprimierte SOAP-Nachrichten keine Probleme für WSNs dar.

Zusätzlich zur Nachrichtenserialisierung erzeugt der Transport einen weiteren Kommunikations-Overhead. Durch LTP entsteht beim Transport der hier aufgeführten Nachrichten bei einer vollständigen Header-Kompression ein Overhead von 6 Byte für alle asynchronen Nachrichten (3-7), 14 Byte für alle Anfrage- (1, 8, 9, 11) und 10 Byte für alle Antwortnachrichten (2, 10, 12) in einer synchronen Kommunikation. Mit einem Gesamt-Overhead zwischen 7 Byte und 26 Byte kann die von IWFMS verwendete LTP+SMC-Webservice-Kommunikation problemlos in WSNs verwendet werden. IWFMS erfüllt damit die Anforderungen bzgl. einer effizienten Kommunikation.

5.6.3 Evaluation der Codegröße

Eine weitere limitierende Ressource stellt in Sensornetzen der verfügbare Programmspeicher dar. Die Logik zum Nachrichtenaustausch, für die Laufzeitumgebungen sowie zur Abbildung

Prozessmodell	**SMC**	**Logik & Laufz.**	$\sum$	**LTP**	$\sum$
Überwachung (P)	4.120	4.476	8.596	5.165	13.761
LKW-Zustände	2.780	3.892	6.672		11.837
	6.900	8.368	15.268	5.165	20.433

Tabelle 5.2: Codegrößen der auf einem Sensorknoten ausgeführten Teilprozesse des Anwendungsfalls (in Byte)

der Prozessabläufe darf nicht zu komplex sein. In diesem Abschnitt wird nachgewiesen, dass die Codegeneratoren von IWFMS aus Prozessmodellen sehr effizienten Code erzeugen, der bzgl. der Größe die Programmspeicherressourcen von Sensorknoten schont. Da der Programmspeicher im Backend keine limitierende Ressource darstellt, wird das Backend bei der Evaluation der Codegröße nicht betrachtet.

Tabelle 5.2 zeigt die Codegrößen[2] der Teilprozesse des Anwendungsfalls, die auf einem Sensorknoten ausgeführt werden (s. Abbildung 5.9 auf Seite 125). Der Beladungs- und der aktive Überwachungsprozess (A) wurden als Teil des SM4RCD-Prozesses zur Abbildung der LKW-Zustände realisiert, während der passive Überwachungsprozess (P) mit BPEL umgesetzt wurde.

Insgesamt erzeugen die verschiedenen Codegeneratoren von IWFMS Code zur Ausführung dieser Prozesse, der kompiliert eine Größe von 15.268 Byte erreicht. Darin ist die Logik zur Ausführung der beiden Prozessmodelle mit 4.476 Byte für den Überwachungsprozess und 3.892 Byte für den Zustandsprozess sowie der SMC-Code zur SOAP-Komprimierung und Abbildung der Daten- sowie Nachrichtentypen (4.120 bzw. 2.780 Byte) enthalten. Zusätzlich wird auf jedem Knoten das LTP-Framework benötigt, das insgesamt zusätzliche 5.165 Byte an Codegröße verursacht[3]. Für die Ausführung der beiden Prozesse auf einem Sensorknoten wird damit von IWFMS Programmspeicher im Umfang von 20.433 Byte verbraucht.

Zur Bewertung des Overheads an Codegröße durch eine Prozessrealisierung mit BPEL und SM4RCD wurden der Beladungs- sowie der passive Überwachungsprozess mit SM4RCD (nur Beladungsprozess) und BPEL realisiert. Zusätzlich wurden die entsprechenden Prozesse manuell mit C++ implementiert. Dabei beschränkt sich die manuelle Implementierung auf die Programmierung der Prozesslogik. Die Kommunikation basiert auch in diesem Szenario auf LTP+SMC-Webservices. Aus diesem Grund unterscheiden sich die Codegrößenanteile von SMC zur Nachrichtenserialisierung sowie zur Abbildung der Daten- und Nachrichtentypen nicht zwischen den verschiedenen Realisierungssprachen, sondern ausschließlich zwischen den Prozessen. Der zusätzliche Overhead des LTP-Frameworks von 5.165 Byte wurde in dieser Messreihe nicht berücksichtigt, da er zwischen den Szenarien nicht variiert.

Abbildung 5.12 fasst die Evaluationsergebnisse zusammen. Sie zeigen, dass der Overhead einer Prozessmodellierung im Vergleich zu einer Programmierung bei einfachen, zustandslosen Prozessen relativ groß ist. Die Codegröße der manuellen Implementierung der LKW-Beladung

[2] Alle Codegrößen in diesem Abschnitt stellen die Größe des Maschinencodes für die Pacemate-Plattform und das iSense-Betriebssystem dar.

[3] Da das LTP-Framework von jedem Teilprozess benötigt wird, ist es bei der Gesamtgröße eines einzelnen Teilprozesses jeweils enthalten. Da es aber insgesamt nur einmal auf einem Knoten existieren muss, ist es in der gemeinsamen Betrachtung beider Teilprozesse nur einmal enthalten.

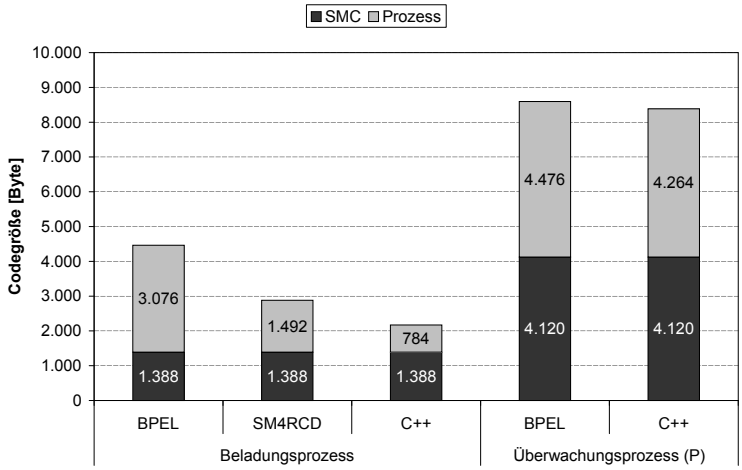


Abbildung 5.12: Codegrößen einer Prozessausführung auf Sensorknoten

beträgt nur 25,5 % der BPEL- und 52,5 % der SM4RCD-Modellierung. Dieses Resultat wird damit begründet, dass bestimmte Teile der Laufzeitumgebungen, wie z. B. die Verwaltung von Prozessinstanzen, zur IWFMS-Prozessausführung unverzichtbar sind. In einer manuellen Implementierung kann dieser und vergleichbarer Overhead in vielen Fällen gezielter vermieden werden. Bei komplexen, zustandsbehafteten Prozessen wird die entsprechende Funktionalität auch in der manuellen Implementierung benötigt, sodass der Overhead einer Umsetzung des Überwachungsprozesses mit BPEL nur noch 5 % gegenüber einer manuellen Implementierung aufweist. SM4RCD weist für den Beladungsprozess nur 48,5 % der Codegröße von BPEL auf. Dieser geringere Overhead resultiert aus der geringeren Komplexität und des geringeren Umfangs der Sprachkonstrukte von SM4RCD.

Zusammenfassend kann der durch die IWFMS-Codegeneratoren erzeugte Code zur Ausführung von Prozessmodellen als sehr effizient und als geeignet für die Ausführung auf Sensorknoten bewertet werden. Sowohl der Overhead im Vergleich zu einer alternativen manuellen Umsetzung von Geschäftsprozessen als auch die absoluten Codegrößen im Rahmen einer Codegenerierung von IWFMS sind sehr gering.

5.6.4 Evaluation des Arbeitsspeicherverbrauchs

Der Arbeitsspeicherverbrauch stellt bei der Prozessausführung im Backend im Gegensatz zum Sensorknoten keine limitierende Ressource dar. Aus diesem Grund wird der RAM-Verbrauch einer Modellinterpretation auch nicht evaluiert. Um den effizienten Umgang von IWFMS mit dem Arbeitsspeicher bei der Ausführung auf Sensorknoten nachzuweisen, wird in diesem Abschnitt der RAM-Verbrauch der Ausführung einer manuellen Prozessrealisierung mit C++ mit dem der Ausführung des durch eine IWFMS-Modellierung entstandenen Codegenerats verglichen. Analog zur Codegrößenbetrachtung dienen auch in diesem Abschnitt der einfache, zustandslose Beladungs- sowie der komplexe, zustandsbehaftete Überwachungsprozess (P) als Evaluationsanwendungsfälle. Bei der Messung wird der komplette durch die Prozessausführung verursachte Speicherverbrauch inkl. der Verarbeitung und Speicherung von SOAP-Nachrichten gemessen. Der durch die Verarbeitung von LTP-Paketen entstehende Overhead wird in diesen

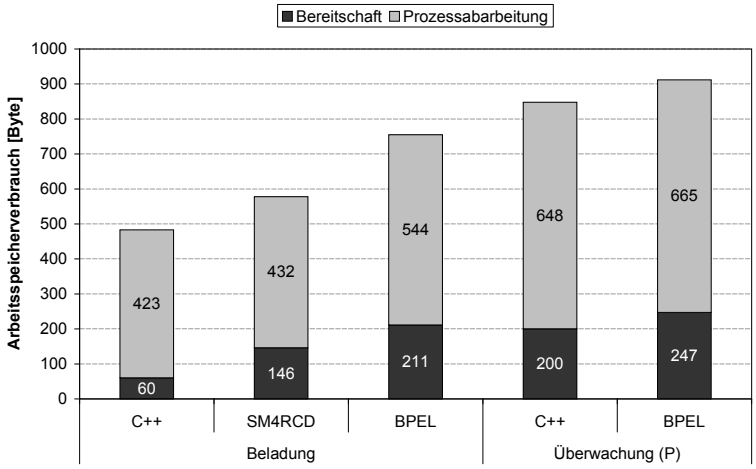


Abbildung 5.13: Arbeitsspeicherverbrauch einer IWFMS-Prozessausführung

Messungen nicht berücksichtigt. Wohl aber der durch die Speicherung der LTP-Pakete verursachte RAM-Verbrauch. Als Ausführungsplattform für die Messungen diente eine 32-Bit-Architektur.

Abbildung 5.13 fasst den Arbeitsspeicherverbrauch bei der Prozessausführung in Abhängigkeit der Realisierungsart zusammen. „Bereitschaft“ beschreibt den Speicherverbrauch nach der Initialisierung des Prozesses. Dieser wird mindestens verursacht, damit eine Prozessinstanz instanziiert und ausgeführt werden kann. „Prozessabarbeitung“ beschreibt den maximalen zusätzlichen Verbrauch, der bei der vollständigen Ausführung einer Prozessinstanz entsteht. Zusammen mit dem minimalen stellt der RAM-Verbrauch der Prozessabarbeitung den maximalen Speicherverbrauch der Prozessausführung dar.

Die Messungen zeigen, dass der Overhead des maximalen RAM-Verbrauchs von IWFMS beim einfachen, zustandslosen Beladungsprozess mit 19,7 % (SM4RCD) und 56,3 % (BPEL) relativ groß ist. Im Gegensatz dazu weicht der maximale RAM-Verbrauch zwischen der Realisierung mit BPEL und C++ mit zunehmender Prozesskomplexität sowie bei zustandsbehafteten Prozessen nur noch geringfügig voneinander ab. Für den Überwachungsprozess beträgt der Overhead von BPEL lediglich 7,5 % gegenüber C++. Genau wie bei der Codegrößenmessung resultieren diese Werte auch bei der RAM-Evaluation aus der Notwendigkeit bestimmter Laufzeitfunktionalität, die zur Ausführung von SM4RCD- und BPEL-Modellen unverzichtbar ist, auf die aber bei einer manuellen Implementierung von einfachen, zustandslosen Prozessen verzichtet werden kann. Diese Aussage wird zusätzlich dadurch begründet, dass der Bereitschafts-Overhead von IWFMS mit 143 % (SM4RCD) und 252 % (BPEL) für den Beladungs- und 23,5 % (BPEL) für den Überwachungsprozess wesentlich größer ist als der Overhead der Prozessabarbeitung. Dieser beträgt für den Beladungsprozess 2,1 % (SM4RCD) bzw. 28,6 % (BPEL) und 2,6 % für den Überwachungsprozess (BPEL). Während der Abarbeitungs-Overhead vor allem durch die Ausführung der Logik sowie die Nachrichtenverarbeitung verursacht wird, wird der Bereitschafts-Overhead größtenteils durch die Prozess- und Instanzverwaltung verursacht. Der Overhead einer Prozessausführung mit IWFMS wird also weniger von der Ausführung Logik sondern vielmehr durch Steuerungs- und Verwaltungsfunktionalität verursacht.

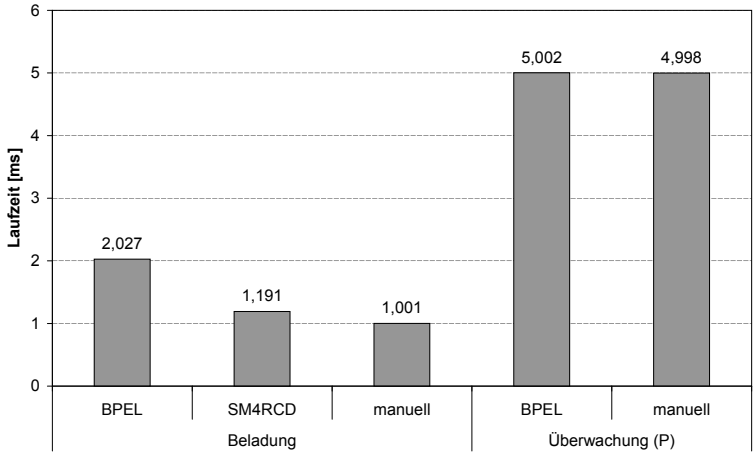


Abbildung 5.14: Vergleich der Geschwindigkeiten einer Prozessausführung auf Sensorknoten

Der RAM-Verbrauch von SM4RCD beträgt für den Beladungsprozess nur 76,6 % im Vergleich zu BPEL. Ebenfalls analog zur Codegröße ist dieser Unterschied auf den größeren Umfang und die größere Komplexität der Sprache BPEL zurückzuführen.

Zusammenfassend lässt sich IWFMS in Bezug auf den Arbeitsspeicherverbrauch als geeignet für die Ausführung im Sensornetz bewerten. Mit einem absoluten maximalen Verbrauch zwischen 578 Byte und 912 Byte können die mit IWFMS realisierten Evaluationsprozesse problemlos auf extrem ressourcenbeschränkten Sensorknoten ausgeführt werden. Gerade für komplexere, zustandsbehaftete Prozesse ist auch der Overhead im Vergleich zu einer manuellen Implementierung vernachlässigbar.

5.6.5 Evaluation des Laufzeitverhaltens

Ein weiteres wichtiges Merkmal der Prozessausführung mit IWFMS ist die Verarbeitungsgeschwindigkeit auf Sensorknoten. Zur Evaluation des Laufzeitverhaltens dienen analog zur Evaluation des Arbeitsspeicherverbrauchs der Beladungs- und Überwachungsprozess (P), die jeweils mit BPEL und SM4RCD (nur der Beladungsprozess) auf Basis von IWFMS sowie manuell mit C++ realisiert wurden. Gemessen wurde jeweils die Dauer der Abarbeitung des kompletten Prozesses auf der Pacemate-Plattform. Das beinhaltet die Verarbeitung von SOAP-Nachrichten (SMC) sowie die Abarbeitung und Verwaltung der Prozessinstanzen.

Abbildung 5.14 fasst die Ausführungsdauer der Prozesse in Abhängigkeit der Realisierungsart zusammen. Die Ergebnisse sind jeweils über 10.000 Iterationen gemittelt und ähneln denen der Evaluation der Codegröße sowie des RAM-Verbrauchs. Auch bei der Laufzeit ist der Overhead von IWFMS für den einfachen, zustandslosen Beladungsprozess mit 18,9 % (SM4RCD) bzw. 102,5 % (BPEL) relativ groß, während er beim zustandsbehafteten, komplexen Überwachungsprozess im Rahmen der Messungenauigkeit quasi nicht existiert. Auch an dieser Stelle resultieren diese Werte aus der Notwendigkeit bestimmter Laufzeitfunktionalität zur Ausführung von SM4RCD- und BPEL-Modellen, die bei einer manuellen Implementierung von einfachen, zustandslosen Prozessen leichter wegoptimiert werden kann. Vergleichbare Funk-

tionalität ist aber auch bei einer manuellen Implementierung komplexer, zustandsbehafteter Prozesse unverzichtbar.

Der Overhead von BPEL gegenüber SM4RCD beträgt für den Beladungsprozess 70,2 %. Dieser Wert resultiert auch bei der Evaluation des Laufzeitverhaltens aus dem größeren Umfang und der größeren Komplexität der Sprache BPEL.

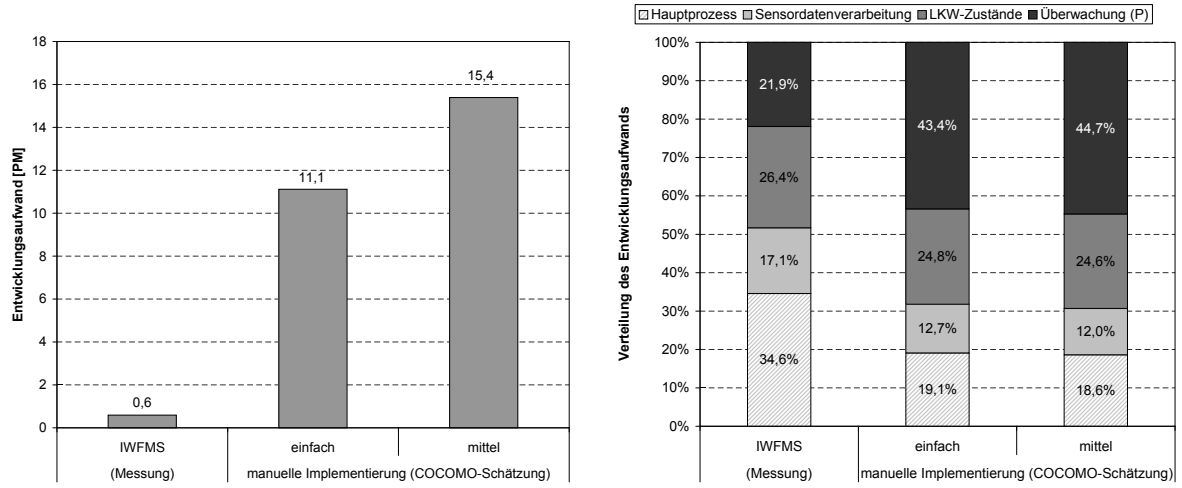
Zusammenfassend lässt sich IWFMS für die Prozessausführung auf Sensorknoten als geeignet bewerten. Mit einer absoluten Ausführungsdauer zwischen 1,191 ms und 5,002 ms können die mit IWFMS realisierten Prozesse problemlos auf Sensorknoten ausgeführt werden. Gerade für komplexe, zustandsbehaftete Prozesse verursacht IWFMS keinen Overhead im Vergleich zu einer manuellen Realisierung.

5.6.6 Evaluation des Entwicklungsaufwands

Mit der Code- und Nachrichtengröße, dem Laufzeitverhalten sowie dem Arbeitsspeicherverbrauch wurden in den vorangegangenen Abschnitten die Eigenschaften einer Prozessrealisierung mit IWFMS evaluiert, die die Kosten während der Ausführung darstellen. Im Gegensatz dazu lässt sich der Nutzen des Einsatzes von IWFMS bei der Prozessrealisierung schwer quantitativ bewerten. Zwar wurde bereits der Nutzen des Konzepts dieser Arbeit zum ganzheitlichen Geschäftsprozessmanagement in Sensornetzen und Backend-Systemen sowie der Verwendung von IWFMS als Umsetzung dieses Konzepts ausführlich qualitativ diskutiert. Diese Argumentation wird jedoch in diesem Abschnitt trotz der Quantifizierungsprobleme auch quantitativ nachgewiesen.

Zur Evaluation des Nutzens der Verwendung von IWFMS bei der Prozessrealisierung wird der Entwicklungsaufwand einer manuellen Prozessrealisierung mit dem der Verwendung von IWFMS verglichen. Eine empirische Studie, die eine repräsentative Menge an Prozessen sowohl manuell als auch mithilfe von IWFMS realisiert, war wegen des hohen Aufwands im Rahmen dieser Arbeit unmöglich. Aus diesem Grund wurde der Entwicklungsaufwand am Beispiel des Prozesses zur Gefahrguttransportüberwachung evaluiert. Dazu wurden die entsprechenden Teilprozesse mit IWFMS modelliert. Gemessen wurde die tatsächliche Entwicklungsdauer, zu der noch ein zusätzlicher Overhead zum Ausgleich von Messfehlern hinzuaddiert wurde. Dem wurde die Entwicklungsdauer der manuellen Implementierung derselben Teilprozesse gegenübergestellt. Da aufgrund des zu hohen zeitlichen Aufwands eine manuelle Implementierung im Rahmen dieser Arbeit nicht möglich war, wurde der Entwicklungsaufwand mit COCOMO (*Constructive Cost Model*, [24, 25]) geschätzt.

COCOMO ist ein algorithmisches Modell der Softwaretechnik zur Schätzung von Entwicklungsaufwänden und -kosten. Es basiert auf Erfahrungswerten und schätzt den Entwicklungsaufwand primär auf Basis der Anzahl der erwarteten Codezeilen eines Softwareentwicklungsprojekts. Als sekundärer Kostenfaktor fließt die Projektkomplexität mit in die Schätzung ein. Dabei wird zwischen drei Komplexitätsausprägungen unterschieden. Einfache Projekte zeichnen sich durch kleine Expertenteams, einen geringen oder keinen Innovationsbedarf, geringen Zeitdruck und geringen Anforderungen bzgl. Änderungen sowie durch einen geringen Projektumfang aus. Schwere Projekte sind durch einen großen Projektumfang, große Teamgrößen und einen hohen Lernaufwand für die Teammitglieder charakterisiert. Zusätzlich unterliegt die Entwicklung starken Beschränkungen, einer hohen Dynamik und



(a) Absoluter Entwicklungsaufwand

(b) Relativer Entwicklungsaufwand

Abbildung 5.15: Vergleich des Entwicklungsaufwands der IWFMS-basierten und manuellen Realisierung des Gefahrguttransportüberwachungsprozesses

hohem Termindruck. Zwischen diesen beiden Extremen liegen mittelschwere Projekte als dritte Komplexitätsausprägung.

Die Umsetzung der Gefahrguttransportüberwachung kann als einfaches, eher sogar als mittelschweres Projekt gewertet werden. Es weist zwar nur eine geringe Teamgröße und einen geringen bis mittleren Umfang auf. Allerdings ist der Innovationsgrad sowie der durch die Ressourcenbeschränkungen und den eingebetteten Charakter von Sensornetzen verursachte Komplexitätsgrad sehr hoch.

Abbildung 5.15a fasst den absoluten Entwicklungsaufwand zusammen. Für die Realisierung des Gefahrguttransportüberwachungsprozesses mit IWFMS wurden 0,6 PM (Personenmonate) gemessen. Je nach Einordnung als einfaches oder mittelschweres Projekt wird der Aufwand für die manuelle Implementierung auf 11 PM bzw. 15 PM geschätzt. Die enorme Zeitersparnis durch IWFMS resultiert vor allem daraus, dass bei einer manuellen Entwicklung auf eine Designphase eine sehr aufwendige Implementierungsphase folgt, die aufgrund der besonderen Charakteristiken von Sensornetzen einen hohen Komplexitätsgrad aufweist. Im Gegensatz dazu entfällt bei der Verwendung von IWFMS dieser Aufwand komplett, da das Modell direkt ausgeführt bzw. aus ihm automatisch der Code für die Ausführung generiert wird. Auch eine zusätzliche Dokumentation ist bei IWFMS weitestgehend unnötig, da das Modell gleichzeitig der Dokumentation dient. Weiterer Aufwand kann in der Testphase vermieden werden. Zwar besteht auch bei der Verwendung von IWFMS ein Testbedarf. Aufgrund der domänenorientierten Modellierung durch Fachexperten und der automatischen Modellausführung kann die Fehlerrate stark reduziert werden. Insbesondere werden Fehler vermieden, die auf die manuelle Überführung des Modells in Software sowie auf eine Fehlinterpretation der fachlichen Modelle durch IT-Personal, insbesondere wegen des Business-IT-Gaps, zurückzuführen sind. Auch erlaubt die umfangreiche Werkzeugunterstützung von IWFMS zur Überwachung von Prozessen ein einfacheres und domänenspezifisches Testen, das insbesondere aufgrund des eingebetteten Charakters von WSNs von großer Bedeutung ist.

Abbildung 5.15b zeigt die Verteilung des Entwicklungsaufwands auf die einzelnen Teilprozesse. Bemerkenswert ist, dass die Verteilung zwischen den Evaluationsszenarien variiert. So nimmt z. B. der Hauptprozess bei der Verwendung von IWFMS mit 34,6 % den größten Anteil am Gesamtaufwand ein, während sein Anteil bei einer manuellen Implementierung auf unter 20 % geschätzt wird. Umgekehrtes gilt für den Überwachungsprozess. Dieses Ergebnis ist mit der Transparenz der Ausführungsplattform von IWFMS zu erklären. Bei IWFMS hängt der Aufwand ausschließlich von der Komplexität des Prozesses ab, während bei einer manuellen Implementierung sowohl die Prozesskomplexität als auch die Ausführungsplattform den Aufwand bestimmen. Entsprechend sinken die relativen Aufwände der Backend-Prozesse trotz ihrer hohen Prozesskomplexität bei einer manuellen Implementierung sehr stark.

Die Evaluation des Entwicklungsaufwands ist zwar ungenau, da zum einen der Aufwand für eine manuelle Implementierung nur mit COCOMO geschätzt wurde und diese Werte zum anderen mit konkreten Messungen verglichen wurden. Die Evaluationsergebnisse zeigen jedoch, dass sich die Aufwände für beide Realisierungsarten in vollkommen anderen Größenordnungen bewegen. Durch die Verwendung von IWFMS kann also der Entwicklungsaufwand sehr stark reduziert und damit auch eine der größten Markteintrittsbarrieren von WSNs beseitigt werden.

5.7 Zusammenfassung

In diesem Kapitel wurde mit dem vom Autor dieser Arbeit entwickelten IWFMS ein umfassendes Workflowmanagementsystem zur Realisierung eines ganzheitlichen Geschäftsprozessmanagements vorgestellt. Es folgt dem WfMC-Referenzmodell zur Umsetzung eines Workflowmanagementsystems und realisiert alle darin geforderten Schnittstellen. Sowohl die Prozessdefinition als auch die -überwachung und -optimierung kann direkt durch Fachpersonal auf einer fachlichen Abstraktionsebene durchgeführt werden.

IWFMS ermöglicht die direkte Ausführung der Prozessmodelle auf Sensorknoten und Backend-Systemen sowie eine Interaktion mit beliebigen externen Prozessen oder Diensten im Gesamtsystem. Sowohl die Ausführung als auch die Kommunikation erfolgt dabei für den IWFMS-Benutzer transparent. Das bedeutet, dass unabhängig von der Ausführungsplattform eines Prozesses die Modellierung und Überwachung über dieselben IWFMS-Schnittstellen und Werkzeuge erfolgt. Mit BPEL und der im Rahmen dieser Arbeit entwickelten und ebenfalls in diesem Kapitel vorgestellten Sprache SM4RCD erfolgt die Prozessdefinition über zwei DSLs. Durch die Verwendung von LTP+SMC auf der Kommunikationsebene wird die o. g. ganzheitliche Verteilungstransparenz erreicht.

Im Rahmen einer Evaluation wurde anhand eines realen Anwendungsfalls die Praxistauglichkeit von IWFMS nachgewiesen und gezeigt, dass der Entwicklungsaufwand einer Geschäftsprozessrealisierung auf Basis von IWFMS im Vergleich zu einer manuellen Umsetzung sehr stark reduziert werden kann. Gleichzeitig ist der Ressourcenverbrauch während der Ausführung kaum größer als bei einer manuellen Implementierung und deutlich geringer als bei heutigen Webservice-basierten Lösungen zur Prozessrealisierung in Backend-Systemen.

6 Eine dynamische Dienstvermittlung und transparente Transport-Binding-Konvertierung

Mit dem in Kapitel 4 beschriebenen Webservice-Transport-Binding LTP+SMC und dem in Kapitel 5 präsentierten Workflowmanagementsystem IWFMS kann sowohl eine Webservice-Kommunikation als auch darauf aufbauend ein Geschäftsprozessmanagement ganzheitlich im Sensornetz und Backend realisiert werden. In den typischen Anwendungsszenarien reichen diese Lösungen zur informationstechnischen Umsetzung von Geschäftsprozessen im Kontext von Sensornetzen aus.

Bezogen auf die Kommunikationsebene realisiert LTP+SMC die Kommunikation zwischen Webservice-Anbieter und -Nutzer. Dabei erfolgt jedoch die Bindung zwischen beiden Kommunikationspartnern statisch zur Designzeit. Wie bereits diskutiert, reicht gerade in durch eine hohe Dynamik und Ad-hoc-Kommunikationsbeziehungen charakterisierten Sensornetzanwendungen eine statische Bindung nicht immer aus. Deshalb muss ein dynamisches Auffinden und Binden zwischen Kommunikationspartnern zur Laufzeit realisiert werden. Zur Umsetzung genau dieser Aufgabe sieht das Konzept dieser Arbeit (s. Abschnitt 3.4.3) das Protokoll LDDE (*Lean Description, Discovery and Exchange*) zur Realisierung der SOA-Rolle einer Dienstvermittlung vor. Es besteht aus effizienten Protokollen zum dezentralen, dynamischen Auffinden von Webservices im Gesamtsystem und zum Austausch ihrer Schnittstellenbeschreibungen sowie aus einem Verfahren zur optimierten Kompression von WSDL-Dokumenten.

Unabhängig von einer statischen oder dynamischen Vermittlung kann eine Webservice-Kommunikation zwischen Kommunikationspartnern nur erfolgen, wenn paarweise jeweils dasselbe Transport-Binding verwendet wird. Aus den in Abschnitt 3.4.3 diskutierten Gründen muss auch eine Kommunikation mit Partnern realisiert werden, die nicht LTP+SMC als Binding verwenden. Um auch Interoperabilität bei der Verwendung unterschiedlicher Bindings zu gewährleisten, sieht das Konzept dieser Arbeit eine transparente und konfigurationsfreie automatische Webservice-Transport-Binding-Konvertierung vor. So kann ohne Aufwand Interoperabilität zwischen andernfalls nicht interoperablen Kommunikationspartnern erreicht und existierende Serviceinventare genutzt werden, ohne zeit- und kostenintensive Integrationsprojekte durchführen zu müssen.

Im ersten Teil dieses Kapitels wird in Abschnitt 6.1 LDDE vorgestellt und seine Leistungsmerkmale analysiert, bevor die Binding-Konvertierung in Abschnitt 6.2 beschrieben und evaluiert wird. Die in diesem Kapitel präsentierten Forschungsergebnisse wurden größtenteils bereits in [68, 71] vorab veröffentlicht.

6.1 Webservice-Vermittlung im Gesamtsystem

Wie das Anwendungsbeispiel der Gefahrgutüberwachung zeigt, können Sensornetze sehr dynamische, ständig wechselnde Ad-hoc-Netze mit Kommunikationsendpunkten verschiedener organisationaler Zugehörigkeit bilden. Gleichzeitig können Teilnetze dauerhaft oder zumindest temporär isoliert sein. Dabei ist der Verlust der globalen Konnektivität nicht auf die WSNs beschränkt. So kann z. B. beim Schiffstransport eine Konnektivität zwischen den LKWs und dem Backend-System des Schiffes existieren, während die Verbindung zum Internet unterbrochen ist.

Für das Auffinden von selbstbeschreibenden Diensten und zum Abruf ihrer Schnittstellenbeschreibungen ergeben sich aus diesen Charakteristiken verschiedene Anforderungen. So muss ein Ansatz zum Auffinden von Diensten dynamisch sein. Das bedeutet, dass permanent und ohne zeitlichen Verzug neue Dienste in das System integriert sowie Dienste aus diesem wieder entfernt werden müssen. Außerdem müssen aufgrund der Ad-hoc-Kommunikationsbeziehungen, der Isoliertheit und der unterschiedlichen organisationalen Zuständigkeiten sowohl das Auffinden von Diensten als auch der Abruf ihrer Schnittstellenbeschreibungen vollständig dezentral erfolgen. Zusätzlich muss die Ganzheitlichkeit des Konzepts dieser Arbeit gewahrt bleiben. Ein entsprechender Ansatz muss also für den Benutzer transparent im Gesamtsystem anwendbar sein. Und als letzte Anforderung muss eine Ressourceneffizienz gewährleistet werden. Neben dem Laufzeitverhalten, dem Arbeitsspeicherbedarf und der Codegröße für die Ausführung der Protokolllogik betrifft die Ressourceneffizienz in diesem Zusammenhang vor allem die Größe und Anzahl der auszutauschenden Nachrichten sowie die Größe der auf den Sensorknoten oder anderen Endgeräten gespeicherten Dienstbeschreibungen.

6.1.1 Verwandte Arbeiten

Protokolle zum Auffinden von Diensten und Geräten sowie zum Austausch von Schnittstellenbeschreibungen haben sowohl in klassischen Enterprise-IT-SOAs als auch in Sensornetzen eine große Bedeutung. Während für viele IP-basierte Netze sowie im Umfeld der Webservice-Technologie bereits eine Vielzahl an Lösungen und Standards existiert, besteht in diesem Bereich für Sensornetze noch ein großer Forschungsbedarf.

Für auf Webservices basierende SOAs werden in Abschnitt 2.3.2 mit UDDI (*Universal Description, Discovery and Integration*, [154]), WS-Inspection [13] und WS-Discovery (*Web Services Dynamic Discovery*, [163]) drei in der Enterprise-IT eingesetzte Standards zur Dienstvermittlung beschrieben. UDDI ermöglicht das Auffinden beliebiger Webservices und den Abruf von Metainformationen wie z. B. Schnittstellenbeschreibungen. Auch die Registrierung, das Auffinden und der Abruf von WSDL-Beschreibungen von beliebigen LTP+SMC-Webservices sind mit UDDI möglich. Allerdings kann ausschließlich aus dem Backend auf das Verzeichnis zugegriffen werden. Folglich kann die vom Konzept dieser Arbeit geforderte Ganzheitlichkeit nicht gewährleistet werden. Auch handelt es sich bei UDDI um einen Ansatz, der mit dem Verzeichnis eine zentrale, immer erreichbare Komponente benötigt, sodass weder das System robust ist, noch dass die einzelnen Kommunikationspartner autark agieren können. Darüber hinaus wird ein zentraler Verwaltungsaufwand verursacht.

Eine ähnliche Bewertung gilt für WS-Inspection. Zwar erlaubt dieser Standard die Referenzierung von LTP+SMC-Webservices und ihrer WSDL-Beschreibungen. Allerdings handelt es sich hier ebenfalls um einen zentral aufgebauten Ansatz, der das Auffinden lediglich aus dem Backend erlaubt. Außerdem ist zwar der Abruf, aber nicht die Registrierung von Diensten standardisiert. Genau wie UDDI verursacht auch WS-Inspection einen enormen manuellen Verwaltungsaufwand, sodass auch dieser Standard die o. g. Anforderungen nicht erfüllt und nicht im Rahmen dieser Arbeit eingesetzt werden kann.

DPWS (*Devices Profile for Web Services*, [42]) stellt eine Sammlung von Standards zur Realisierung selbstbeschreibender und automatisch auffindbarer Dienste dar. Zur dynamischen Dienstsuche wird WS-Discovery verwendet, während zum Austausch von Webservice-Beschreibungen der Einsatz von WS-MetadataExchange (*Web Services Metadata Exchange*, [260]) in Verbindung mit WS-Transfer (*Web Services Transfer*, [243]) von DPWS vorgesehen ist. Durch das Zusammenspiel dieser drei Standards lassen sich ohne eine zentrale Infrastruktur oder einen zentralen Verwaltungsaufwand selbstbeschreibende Webservices realisieren, die dynamisch gefunden und deren Dienstbeschreibungen direkt abgerufen werden können. Damit eignet sich DPWS besonders gut für dynamische Anwendungsszenarien mit einer autarken Ad-hoc-Kommunikation und erfüllt sehr gut die o. g. funktionalen Anforderungen. DPWS ist zwar gezielt für den Einsatz auf eingebetteten und ressourcenschwachen Geräten entworfen, allerdings bedeutet Ressourcenbeschränktheit in diesem Kontext eine Leistungsstärke von PDAs oder Druckern. Für WSNs ist der Ressourcenverbrauch von DPWS jedoch bei Weitem zu groß.

Einen Ansatz zur Verwendung von DPWS in WSNs präsentieren Moritz et al. [142]. Dieser Ansatz versucht die Anwendbarkeit von DPWS durch Einschränkungen der einzelnen Standards zu erreichen. Mit Nachrichtengrößen zwischen 584 Byte und 5.217 Byte sowie einem Mittelwert von 1.507 Byte für den von Moritz et al. gewählten Anwendungsfall ist auch dieser Ansatz zu ressourcenverbrauchend für die meisten WSNs. Auch ist nicht die Speicherung der Schnittstellenbeschreibungen auf den Sensorknoten vorgesehen, sodass eine autarke Selbstbeschreibung nicht realisiert werden kann.

Einen weiteren Ansatz zum dezentralen Auffinden von Diensten stellt UPnP (*Universal Plug and Play*, [88]) dar. Die Funktionsweise von UPnP ähnelt der von WS-Discovery, basiert jedoch nicht auf der Webservice-Technologie. Dobrescu et al. [41] beschreiben wie UPnP in WSNs verwendet werden kann. Allerdings sieht der Ansatz die zwingende Verwendung von IP vor. Aufgrund der fehlenden Kompatibilität zu Webservices und dem Zwang zur Verwendung von IP, das im Rahmen dieser Arbeit zwar optional in WSNs verwendet werden kann, dessen Existenz aber nicht vorausgesetzt werden darf, ist der Einsatz von UPnP im Rahmen dieser Arbeit nicht möglich. Aus denselben Gründen scheidet auch das SLP (*Service Location Protocol*, [74]) aus, das in einigen Sensornetzen verwendet wird [268].

Die o. g. verwandten Arbeiten beschäftigen sich ausschließlich mit dem Auffinden von Diensten sowie z. T. mit dem Austausch von Dienstbeschreibungen. Um eine autarke Selbstbeschreibung zu realisieren, ist jedoch zusätzlich die Speicherung der Schnittstellenbeschreibungen auf den Sensorknoten unverzichtbar. Da die Webservice-Schnittstellen gegenwärtig ausschließlich mit WSDL beschrieben werden und entsprechende Dokumente sehr viel Speicherplatz einnehmen, müssen sie in komprimierter Form auf den Sensorknoten gespeichert werden. Zu ihrer Kompression lassen sich die in Abschnitt 4.2.2 beschriebenen allgemeinen Kompressoren sowie die schemalosen und schemabehafteten XML-Kompressoren einsetzen. Allerdings weisen diese

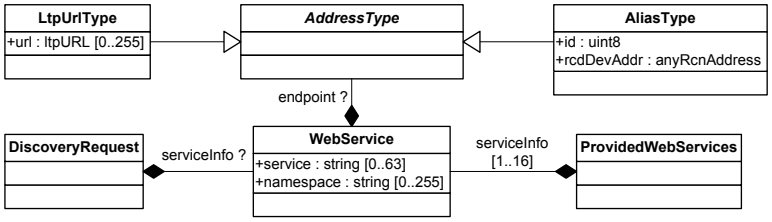


Abbildung 6.1: Aufbau der SOAP-Nachrichten des Discovery-Webservices von LDDE

Verfahren zu schlechte Kompressionsraten auf, da sie nicht auf die besonderen Merkmale von WSDL ausgerichtet sind. Speziell optimierte Kompressoren existieren jedoch zurzeit nicht.

6.1.2 Lean Description, Discovery and Exchange

Das LDDE-Protokoll (*Lean Description, Discovery and Exchange*) realisiert im Gesamtsystem das dynamische Auffinden und die Selbstbeschreibung von Webservices und setzt damit die SOA-Rolle der Dienstvermittlung (s. Abschnitt 2.1.4) informationstechnisch um. Abbildung 3.3 auf Seite 60 zeigt die Einordnung von LDDE in den Webservice-Technologiestapel. LDDE baut auf der Beschreibungsebene auf WSDL sowie auf der Nachrichtenebene auf SOAP auf. Zwar können konzeptionell auch XML-serialisierte SOAP-Nachrichten verwendet werden. Aufgrund der Ressourcenbeschränkungen von Sensornetzen ist jedoch eine SMC-Kompression unverzichtbar. Zum Transport verwendet LDDE das LTP-Protokoll.

Mit dem Auffinden (engl.: *Discovery*) und der Beschreibung (engl.: *Description*) von Webservices sowie dem Austausch (engl.: *Exchange*) der Schnittstellenbeschreibungen wird die Dienstvermittlung von LDDE in drei verschiedenen Teilaufgaben realisiert. Dabei handelt es sich um einen vollständig dezentral aufgebauten Ansatz, der weder eine zentrale Systemkomponente noch einen zentralen Verwaltungsaufwand verlangt und in dynamischen sowie isolierten Kommunikationsbeziehungen verwendet werden kann.

Auffinden von Diensten

Die erste Aufgabe bei einer dynamischen Dienstvermittlung ist das Auffinden von Webservices bzw. Webservice-Instanzen. Wie zuvor beschrieben, erfüllt WS-Discovery sehr gut die funktionalen Anforderungen dieser Aufgabe. Allerdings kann dieser Standard aus zwei Gründen nicht verwendet werden. WS-Discovery sieht für eine Multicast-Kommunikation UDP und HTTP für Unicast-Verbindungen vor. Während HTTP keinesfalls in Sensornetzen verwendet werden kann, steht UDP nur zum Teil und nicht flächendeckend in WSNs zur Verfügung. Wie aber die Evaluation in Abschnitt 6.1.3 zeigt, ist auch der Kommunikations-Overhead bei der Verwendung von LTP+SMC als Transport-Binding zu groß. Aus diesem Grund wurde im Rahmen dieser Arbeit ein eigenes, auf LTP+SMC basierendes Discovery-Protokoll entwickelt. Konzeptionell ist dieses an WS-Discovery angelehnt. Jedoch sind die verwendeten Nachrichtenstrukturen sowie das Transport-Binding wesentlich effizienter.

Bei der Dienstsuche wird bei LDDE zwischen einer aktiven und passiven Suche unterschieden. In beiden Fällen werden mit SMC komprimierte SOAP-Nachrichten über LTP in Unicast-

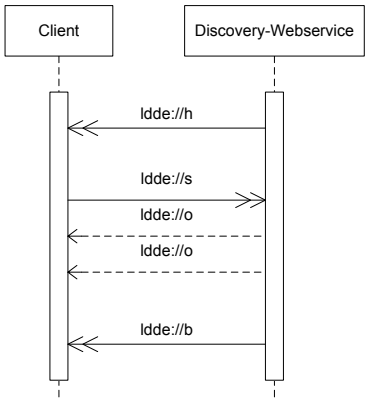


Abbildung 6.2: Nachrichtenfolge einer Webservice-Suche mit LDDE

und Multicast- bzw. Broadcast-Kommunikationsbeziehungen ausgetauscht. Abbildung 6.1 stellt die Struktur der SOAP-Nachrichten dar, die zwischen den Dienstsuchern und den tatsächlichen sowie potenziellen Dienstanbietern ausgetauscht werden, während Abbildung 6.2 die Nachrichtenfolge zeigt.

Bei einer aktiven Suche schickt ein suchender Kommunikationspartner ein *DiscoveryRequest*-Paket mit LTP per Broad-/Multicast an das Teilnetz, in dem nach Webservices gesucht werden soll. Dazu enthält die *to*-URL als *authority*-Eintrag die Broad-/Multicast-Geräteadresse des Teilnetzes. Die Adressierung des Webservices, der die Suchanfragen auf dem jeweiligen Gerät verarbeitet (Discovery-Webservice), erfolgt über die Pfadangabe „/“. Neben der Adresse zum Empfang der Antwortnachrichten und einer Nachrichten-ID muss über das *action*-Feld mit dem Eintrag „ldde://s“ festgelegt werden, dass diese Nachricht eine LDDE-Suchanfrage enthält. Die eigentliche Suchanfrage ist in der SOAP-Nachricht codiert. Diese ist vom Typ *DiscoveryRequest* und erlaubt durch die Angabe eines Webservice-Bezeichners und -Namensraums optional eine gezielte Suche nach Instanzen eines Webservices sowie durch Weglassen dieser Informationen die Suche nach allen Webservices eines Teilnetzes. Bei Suchanfragen dürfen *DiscoveryRequest*-Pakete keine konkreten Endpunktadressen enthalten. Jedes Endgerät, das diese Anfrage erhält, antwortet mit einer Liste an Diensten, die es anbietet. Diese sind in SOAP-Nachrichten vom Typ *ProvidedWebServices* codiert. Zu jedem Webservice kann optional die konkret angebotene Instanz durch die Nennung ihres Endpunkts angeben werden. Analog zu LTP kann jeder Endpunkt als Alias oder URL beschrieben werden (s. Abschnitt 4.5.2). Da Aliase jedoch ausschließlich innerhalb eines Teilnetzes gültig sind, dürfen sie nur verwendet werden, wenn sich der Anbieter und Sucher im selben Teilnetz befinden. Zwar wäre auch die Ersetzung von Aliasen an den Routern denkbar und sehr einfach möglich. Um aber die Unabhängigkeit des LTP-Protokolls von LDDE zu wahren, wurde auf diese Option verzichtet. Auf der Ebene von LTP werden die Antwortnachrichten auf Suchanfragen durch den Bezeichner „ldde://o“ im *action*-Feld gekennzeichnet. Zusätzlich wird im *from*-Feld die Unicast-Endpunktadresse des Discovery-Webservices des antwortenden Endgeräts angegeben.

Damit nicht jeder suchende Kommunikationspartner permanent Suchanfragen stellen muss, um neu in das Netz eintretende Dienste zu ermitteln, wird neben der aktiven auch eine passive Suche umgesetzt. Jedes Endgerät, das in ein Teilnetz eintritt, schickt eine Nachricht vom Typ *ProvidedWebServices* mit den von ihm angebotenen Webservices an die Broad- bzw. Multicast-

Adresse des Teilnetzes. Diese *Hello*-Nachricht wird mit „ldde://h“ im *action*-Feld des LTP-Pakets gekennzeichnet. Verlässt ein Endgerät das Teilnetz wieder, versendet es ein LTP-Paket mit leerem Body an das Teilnetz. Dieses Paket enthält im *action*-Feld den Bezeichner „ldde://b“. Beide Nachrichten enthalten im *from*-Feld die Unicast-Endpunktadresse des eigenen Discovery-Webservices.

Mit dieser Lösung zur Dienstsuche können vollständig dezentral und ohne Verwaltungsaufwand dynamisch Webservice-Instanzen im Gesamtsystem gefunden werden.

Selbstbeschreibung von Webservices

Nach dem Auffinden von Webservice-Instanzen im Gesamtsystem können Dienstnutzer und -anbieter dynamisch zur Laufzeit gebunden werden. Dieses setzt allerdings die Kenntnis der WSDL-Dokumente der entsprechenden Webservices zur Designzeit voraus. Ein vollständiges dynamisches Binden kann nur erfolgen, wenn auch die Dienstbeschreibungen zur Laufzeit ermittelt und interpretiert werden. Nur so können Webservice-Nutzer mit Diensten interagieren, von denen sie weder die Schnittstelleninformationen noch die Endpunktadressen zur Designzeit kennen. Zusätzlich setzt auch die in Abschnitt 6.2 präsentierte transparente und konfigurationsfreie automatische Binding-Konvertierung selbstbeschreibende Webservices voraus. Zwar ist die Interpretation von WSDL-Dokumenten aufgrund der Ressourcenanforderungen nicht auf Sensorknoten, sondern ausschließlich im Backend möglich. Durch die ganzheitliche Realisierung von selbstbeschreibenden Webservices können jedoch sowohl WSN- als auch Backend-Webservices vollständig dynamisch zur Laufzeit an Webservice-Nutzer im Backend gebunden werden.

Aufgrund der Dezentralität von LDDE und der Notwendigkeit der Funktionsfähigkeit in isolierten Kommunikationsbeziehungen können selbstbeschreibende Webservices nur realisiert werden, wenn die Schnittstellenbeschreibungen direkt auf den Endgeräten gespeichert und von diesen abgerufen werden können. WSDL stellt gegenwärtig den einzigen Standard zur Definition von Webservice-Schnittstellen dar und wird auch im Rahmen dieser Arbeit zur Beschreibung von LTP+SMC-Webservices verwendet (s. Abschnitt 4.6). WSDL-Dokumente von Sensornetzdiensten weisen zwar im Vergleich zu Backend-Diensten aufgrund des meist kleineren Dienstumfangs eine geringere Größe auf. Trotzdem sind sie zur Speicherung auf Sensorknoten im XML-Format zu groß. Für eine effiziente Anwendung von generischen Kompressoren wie GZIP oder schemalosen XML-Kompressoren wie XMill sind sie jedoch zu klein. Auch schemabehaftete XML-Kompressoren erreichen trotz einer klaren Spezifikation der WSDL-Grammatik keine guten Ergebnisse. Aus diesen Gründen wurde im Rahmen dieser Arbeit mit WSDLC (*Web Services Description Language Compression*) ein spezielles Kompressionsverfahren für WSDL-Dokumente entwickelt, das die Speicherung der Schnittstellenbeschreibungen auf Sensorknoten erlaubt.

WSDLC realisiert grundsätzlich eine schemabasierte Kompression von WSDL-Dokumenten. Allerdings darf der Kompression nicht die Standardgrammatik von WSDL zugrunde gelegt werden. Zwar ist die Struktur von WSDL-Dokumenten exakt formal spezifiziert. Problematisch ist jedoch, dass diese nicht sehr restriktiv ist, um einen flexiblen Einsatz sowie die Erweiterbarkeit von WSDL zu gewährleisten. So können WSDL-Dokumente an verschiedenen Stellen beliebige XML-Strukturen enthalten, deren Aufbau nicht näher bekannt ist oder

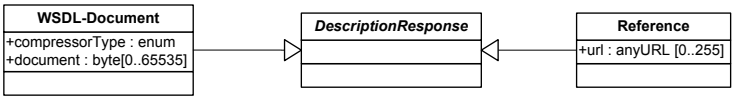


Abbildung 6.3: Struktur der SOAP-Antwortnachricht zum Abruf der WSDL-Beschreibungen von Webservice-Endpunkten

zumindest außerhalb des WSDL-Standards definiert wurde. Für diese Inhalte kann aufgrund der fehlenden Strukturinformationen keine schemabasierte XML-Kompression angewendet werden, sodass sie als Strings interpretiert werden müssen.

Um den Anteil unbekannter Strukturen zu reduzieren, verwendet WSDLC einen statistischen Ansatz. Im Rahmen dieser Arbeit wurden 300 WSDL-Dokumente aus dem Internet betrachtet[1]. Dabei wurden die Strukturen der in den Dokumenten verwendeten XML-Elemente und -Attribute analysiert, die nicht Teil von WSDL sind. Die Schemabeschreibung zur Kompression von WSDL wurde um besonders häufig verwendete Strukturen erweitert, sodass auch nicht in WSDL spezifizierte Inhalte effizient schemabasiert komprimiert werden können. Für immer noch unbekannte Strukturen kann weiterhin eine Codierung als Strings als Ersatzlösung verwendet werden.

Um auch für die Ersatzlösung der String-Codierung ganzer XML-Blöcke eine bessere Kompressionsrate zu erreichen, werden Strings Huffman-codiert. Dabei wird ein für XML-Inhalte optimierter Huffman-Baum verwendet. Dieser entstand ebenfalls aus einer statistischen Analyse der o. g. 300 WSDL-Dokumente.

Um die Selbstbeschreibung von Webservices zu vervollständigen, müssen diese Dokumente zusätzlich von beliebigen WSN- und Backend-Endgeräten abrufbar sein. Der Abruf erfolgt in LDDE über eine Webservice-Kommunikation. Dazu wird ein LTP-Paket an die Endpunkt-URL der entsprechenden Webservice-Instanz gesendet. Dieses enthält im *action*-Feld „ldde://d“ zur Kennzeichnung einer Anfrage zum Abruf der Schnittstellenbeschreibung. Abbildung 6.3 stellt die Struktur der SOAP-Nachricht des Antwortpakets dar. Diese überträgt entweder direkt das WSDL-Dokument Form oder eine URL, über die das Dokument abgerufen werden kann. Zwar wird der Einsatz von WSDLC als WSDL-Kompressionsverfahren empfohlen, allerdings können auch weitere Ansätze verwendet oder auf eine Kompression verzichtet werden. Diese Information wird über das Feld *compressorType* identifiziert. Als *action*-Feld von LTP muss „ldde://w“ in der Antwort gesetzt werden.

Durch die Speicherung der WSDL-Dokumente auf Enterprise-IT-Servern und die ausschließliche Versendung von URLs, die auf diese Dokumente verweisen, können sowohl die Speicherressourcen der Sensorknoten als auch die Datenrate der Sensornetze geschont werden. Allerdings besteht in diesem Fall die Abhängigkeit von der Erreichbarkeit des Servers, der die Schnittstellenbeschreibung bereitstellt. So wird der Einsatz von LDDE in Ad-hoc-Szenarien eingeschränkt. Hier muss eine Abwägung zwischen beiden Selbstbeschreibungsoptionen auf Grundlage der Anwendungsanforderungen getroffen werden.

[1] Die WSDL-Dokumente wurden von der Internetseite `http://www.xmethods.net` geladen.

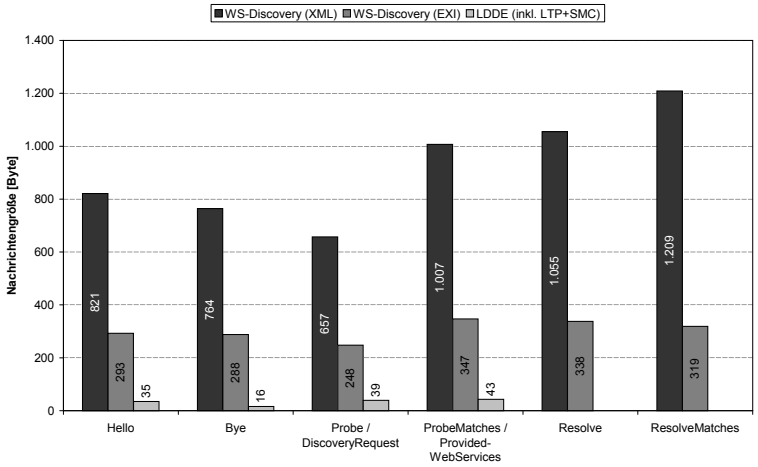


Abbildung 6.4: Vergleich der Nachrichtengrößen von WS-Discovery und LDDE

6.1.3 Analyse der Leistungsmerkmale

In diesem Abschnitt werden die Leistungsmerkmale von LDDE untersucht. Als Evaluationsplattformen dienten die Pacemate-Sensorknoten sowie der Mess-PC, die bereits in den Analysen der Leistungsmerkmale in den vorigen Kapiteln verwendet wurden.

Auffinden von Diensten

Die wichtigste Eigenschaft eines Protokolls zum Auffinden von Webservices im WSN ist die Nachrichtengröße. Zur Untersuchung dieses Merkmals wurden die Nachrichtengrößen des Auffindens des Überwachungsprozesses (P) (s. Abschnitt 5.6.1) verwendet. Zum Vergleich der LDDE-Ergebnisse diente WS-Discovery mit einer XML-Serialisierung sowie mit einer schemabasierten EXI-Kompression. Während die *Hello*- und *Bye*-Nachrichten bei LDDE und WS-Discovery eine weitestgehend identische Semantik haben, weicht die aktive Suche in beiden Ansätzen voneinander ab. WS-Discovery realisiert diese Suchart mit *Probe* und *Resolve* über zwei verschiedene Anfragen und entsprechende Antworten, während LDDE dazu lediglich eine Nachricht vom Typ *DiscoveryRequest* bzw. *ProvidedWebServices* als Anfrage- und Antwortnachricht verwendet.

Abbildung 6.4 fasst die Ergebnisse zusammen. Die Nachrichtengrößen bei WS-Discovery wurden inkl. der notwendigen Verwendung von WS-Addressing aber ohne den Transportprotokoll-Overhead gemessen. Bei LDDE wurde dieser Overhead bereits mit berücksichtigt, da die Informationen, die bei der Verwendung von WS-Discovery im SOAP-Header mit WS-Addressing definiert werden müssen, im Fall von LDDE in LTP codiert werden. Mit Werten zwischen 657 Byte und 1.209 Byte verursacht WS-Discovery sowohl mit unkomprimierten als auch mit komprimierten (zwischen 248 Byte und 347 Byte) Nachrichten einen sehr großen Overhead. Zusätzlich erzeugt WS-Discovery Overhead auf der Transportebene. Eine Übertragung von WS-Discovery über LTP erzeugt einen Gesamt-Overhead zwischen 747 Byte und 1.374 Byte (XML) bzw. 293 Byte und 407 Byte (EXI). Der Overhead eines Transports mit HTTP und UDP ist noch einmal wesentlich höher. Im Vergleich dazu erreichen die Nachrichten von

Nachricht	Serialisierung	Deserialisierung
DiscoveryRequest ohne serviceInfo	0,344 ms	0,812 ms
ProvidedWebServices ohne serviceInfo	0,346 ms	0,818 ms
ProvidedWebServices mit serviceInfo & Endpunkt-URL	0,930 ms	2,230 ms
ProvidedWebServices mit serviceInfo & Endpunkt-Alias	0,390 ms	0,848 ms

Tabelle 6.1: Geschwindigkeit der Nachrichtenverarbeitung von LDDE auf Sensorknoten

LDDE im Mittel eine Kompressionsrate von 96,3 % gegenüber der XML-Serialisierung und 89,9 % gegenüber der EXI-Kompression von WS-Discovery. Mit einem absoluten Overhead für dieses Evaluationsbeispiel zwischen 16 Byte und 43 Byte kann LDDE problemlos in WSNs auch bei einem hohen Nachrichtenaufkommen verwendet werden.

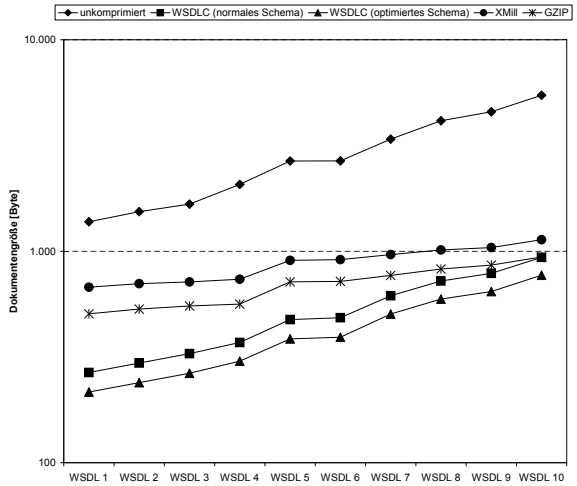
Zur Analyse des Speicherverbrauchs wurden der RAM-Verbrauch und die Codegröße betrachtet. Die Codegröße der Nachrichtenverarbeitung für die Pacemate-Plattform beträgt exklusive Huffman-Baum 1.640 Byte, während der RAM-Verbrauch der Serialisierung und Deserialisierung eine Größe von 640 Byte nicht überschreitet.

Zur Evaluation des Laufzeitverhaltens wurden die in Tabelle 6.1 aufgeführten *DiscoveryRequest*- und *ProvidedWebService*-Nachrichten auf der Pacemate-Plattform serialisiert und deserialisiert. Je nach Szenario enthielten die Nachrichten keine Dienstinformationen, Dienstinformationen ohne konkreten Endpunkt sowie mit einem konkreten Endpunkt, der entweder als Alias oder als URL codiert wurde. Der verwendete Namensraum hatte eine Länge von 11 Zeichen, die URL war 41 Zeichen und der Name 12 Zeichen lang. Jede Messung wurde in 10.000 Iterationen durchgeführt und die Ergebnisse gemittelt. Mit einer Verarbeitungsdauer zwischen 0,3 ms und 2,2 ms können die *DiscoveryRequest*- und *ProvidedWebService*-Nachrichten problemlos auf Sensorknoten verarbeitet werden. Auffällig ist, dass die Verarbeitungsgeschwindigkeit, der Nachrichten ohne Endpunkt-URL im Mittel nur 38 % der Nachricht mit Endpunkt-URL beträgt. Das zeigt den hohen Einfluss von Strings auf die Verarbeitungsgeschwindigkeit.

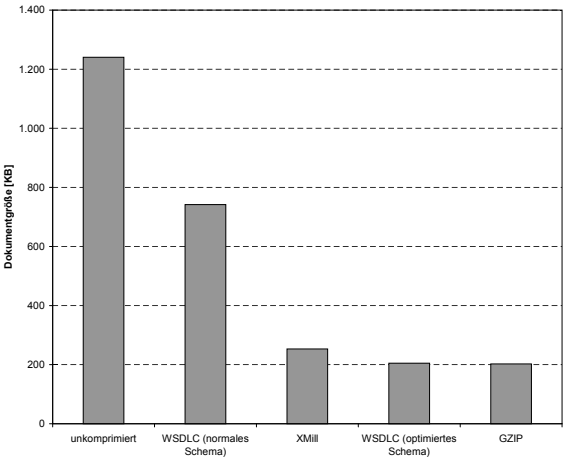
Die Evaluation hat gezeigt, dass LDDE problemlos zum Auffinden von Webservices im WSN eingesetzt werden kann. Außerdem ist dieses Protokoll wesentlich effizienter als WS-Discovery, das sich nicht für den Einsatz im WSN eignet.

Kompression von WSDL-Dokumenten

Das Ziel beim Design von WSDLC war die Erreichung einer effizienten Kompression von relativ kleinen Schnittstellenbeschreibungen. Aus diesem Grund wird in dieser Evaluation das WSDL-Dokument eines typischen Sensornetz-Webservices verwendet. Dieser Dienst realisiert die Abfrage von Temperatur-, Feuchtigkeits-, Bewegungs- und Beschleunigungssensoren sowie das Setzen der Zeit und des Modus eines Knotens. Um das Skalierbarkeitsverhalten von WSDLC mit zunehmender Schnittstellenkomplexität und -größe zu betrachten, wurde als Erstes eine Version des Webservices untersucht, die lediglich die Abfrage des Temperatursensors realisiert. In weiteren neun Szenarien wurden sukzessive die weiteren Funktionen hinzugefügt, sodass die Komplexität und Größe der WSDL-Dokumente in jedem Szenario steigt. Verglichen wird WSDLC mit den beiden Kompressionsverfahren XMill und GZIP. WSDLC wird dabei in



(a) Größenvergleich von Sensornetz-Webservice-Schnittstellen



(b) Kumulative Größe von Enterprise-IT-Webservice-Schnittstellen

Abbildung 6.5: Vergleich der Kompressionsraten einer WSDL-Komprimierung

zwei Varianten betrachtet. Zum einen basiert das zur Kompression verwendete Schema auf der normalen WSDL-Grammatik, während zum anderen das statistisch erweiterte, optimierte Schema verwendet wird. WSDLC kann konzeptionell mit beliebigen schemabasierten XML-Kompressoren umgesetzt werden. Zurzeit wird microFibre für diese Aufgabe eingesetzt.

Abbildung 6.5a zeigt die Kompressionsergebnisse. Die Größe von kleinen mit WSDLC (optimiertes Schema) komprimierten WSDL-Dokumenten weist nur 32,0 % bzw. 42,7 % im Vergleich zu einer XMill- bzw. GZIP-Kompression auf. Mit zunehmender Dokumentengröße nimmt zwar der Vorteil von WSDLC ab. Für die größte WSDL-Beschreibung beträgt er jedoch immer noch 32,1 % bzw. 18,1 %. Im Vergleich zur Größe der unkomprimierten WSDL-Dokumente erreicht WSDLC eine Kompressionsrate zwischen 84,1 % und 85,9 %. Mit absoluten Größen zwischen 216 Byte und 770 Byte können die Beispieldokumente problemlos auf Sensorknoten gespeichert werden. Mit einer ähnlichen Größenordnung wie die o. g. EXI-komprimierten WS-Discovery-Nachrichten erzeugt auch der Austausch komprimierter WSDL-Dokumente eine hohe Belastung für das WSN. Da diese Dokumente aber wesentlich seltener ausgetauscht werden als Discovery-Nachrichten, ist der Overhead im Fall von WSDLC vertretbar.

Zusätzlich zur Analyse spezieller WSDL-Dokumente für Sensornetz-Webservices wurden die Kompressionsergebnisse von 179 aus dem Internet ermittelten WSDL-Dokumenten von Enterprise-IT-Diensten untersucht. Abbildung 6.5b zeigt die kumulative Größe aller Dokumente bei der Verwendung derselben Kompressoren wie im ersten Messszenario. Zwar ist die Kompressionsrate von WSDLC (optimiertes Schema) immer noch mindestens so gut wie die der Mitbewerber. Jedoch zeigen die Ergebnisse, dass die generischen Kompressoren XMill und GZIP mit zunehmender Dokumentengröße effektiver werden. WSDLC mit normalem Schema steht als Repräsentant für eine nicht optimierte schemabasierte XML-Kompression. Das schlechte Abschneiden zeigt, dass eine schemabasierte XML-Kompression nur sinnvoll einsetzbar ist, wenn die XML-Grammatik restriktiv ist.

	Mittelwert	**Maximum**
WSDL	25,94 Byte	36,00 Byte
URL	36,97 Byte	48,00 Byte

Tabelle 6.2: Arbeitsspeicherverbrauch bei der Verarbeitung der SOAP-Antwortnachricht zum Abruf der WSDL-Beschreibungen von Webservice-Endpunkten

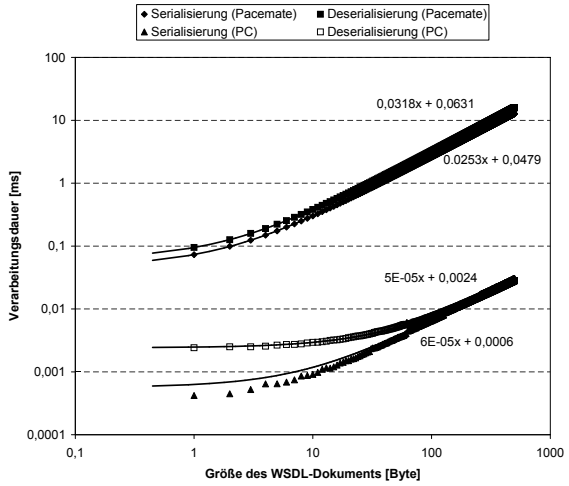
Austausch von WSDL-Dokumenten

Als erstes Merkmal des Austauschs von WSDL-Dokumenten wird der Nachrichten-Overhead betrachtet. Gleichung 6.1 beschreibt die Größe der in Abbildung 6.3 dargestellten SOAP-Nachricht zur direkten Übertragung einer WSDL-Datei bzw. der auf sie verweisenden URL. Sei l_{URL} die Länge der URL in Zeichen, l_{WSDL} die Größe des Feldes *document* in Byte, dann ist s die Größe der resultierenden SOAP-Nachricht vom Typ *DescriptionResponse* in Byte. Die Funktion „stringSize“ ist in Gleichung 4.7 auf Seite 93 definiert und darf in diesem Kontext ausschließlich mit einer Huffman-Codierung verwendet werden. Zusätzlich zur SOAP-Nachricht wird weiterer Overhead durch LTP verursacht. Das Paket zur Übertragung der o. g. Nachricht verursacht zusätzlich zu dieser einen Overhead von 17 Byte. Das Anfrage-Paket überträgt keine Nutzdaten. Es hat eine Größe von 20 Byte. Bei beiden angegebenen LTP-Größen werden alle Header-Felder außer *action* komprimiert und keine Fragmentierung verwendet.

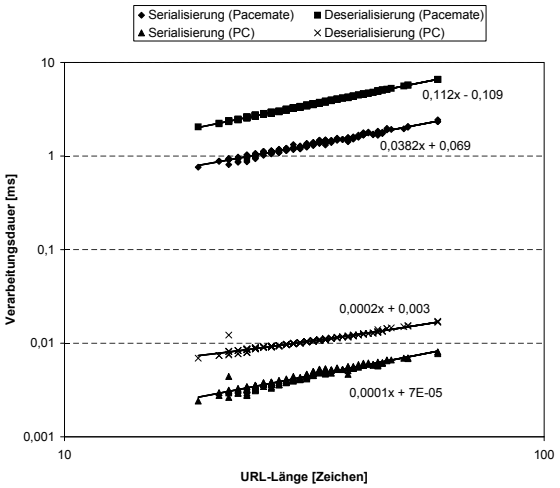
$$s = \left\lceil \frac{1}{8} \cdot \left(1 + \begin{cases} 8 + \text{stringSize}(l_{\text{URL}}) & \text{URL} \\ 21 + 8 \cdot l_{\text{WSDL}} & \text{WSDL} \end{cases} \right) \right\rceil \tag{6.1}$$

Als weitere Evaluationseigenschaft wurde der Speicherverbrauch gemessen. Die Codegröße exklusive Huffman-Code beträgt für die Pacemate-Plattform 1.052 Byte. Zur Analyse des RAM-Verbrauchs wurden die URLs verwendet, von denen die o. g. 179 WSDL-Dokumente abgerufen wurden. Zur Betrachtung der direkten Übertragung von Dokumenten wurden Größen von 0 Byte bis 500 Byte für das Feld *document* serialisiert und deserialisiert. Tabelle 6.2 fasst die Ergebnisse zusammen. Nicht berücksichtigt ist in diesen Messungen der Speicherverbrauch eines Puffers zur Speicherung der übertragenen URL bzw. WSDL auf dem Knoten, sondern ausschließlich die reine Nachrichtenverarbeitung.

Als letztes Merkmal des Austauschs von WSDL-Dokumenten wurde die Verarbeitungsgeschwindigkeit untersucht. Dabei wurde derselbe Testaufbau wie bei der RAM-Messung verwendet. Jede Serialisierung und Deserialisierung wurde auf den Pacemate-Sensorknoten sowie auf dem Mess-PC in 10.000 Iterationen durchgeführt. Die Abbildungen 6.6a und 6.6b fassen die Ergebnisse gemittelt zusammen. Zusätzlich sind die Regressionsgeraden durch die Messpunkte dargestellt. Sie erlauben eine Abschätzung der Verarbeitungsgeschwindigkeit in Abhängigkeit der zu serialisierenden SOAP-Nachricht. Die optische Abweichung der Geraden des Szenarios „Serialisierung PC“ im unteren Wertebereich von den Messpunkten ist auf die logarithmische Skalierung zurückzuführen. Mit einem Korrelationskoeffizienten von 0,999 besteht auch in dieser Messreihe eine starke Abhängigkeit zwischen der WSDL-Größe und der Serialisierungsdauer.



(a) Direkte Übertragung des WSDL-Dokuments



(b) Übertragung einer Referenz auf ein WSDL-Dokument

Abbildung 6.6: Geschwindigkeit der Verarbeitung von SOAP-Nachrichten zur Übertragung von WSDL-Beschreibungen

Zusammenfassend kann der Austausch von WSDL-Dokumenten bzw. ihrer Verweise in LDDE als sehr effizient und als geeignet für den Einsatz in WSNs bewertet werden.

6.2 Transparente Konvertierung von Webservice-Transport-Bindings

Wie bereits beschrieben, setzt eine Webservice-Kommunikation voraus, dass alle Kommunikationspartner paarweise dasselbe Transport-Binding verwenden. Während Webservice-Transport-Bindings wie HTTP+SOAP oder SMTP+SOAP (jeweils XML-serialisiert) zumindest in Enterprise-IT-Umgebungen omnipräsent sind, muss bei alternativen Bindings wie LTP+SMC aus den bereits genannten Gründen zumindest temporär von einem begrenzten Verbreitungsgrad ausgegangen werden. Um trotzdem eine uneingeschränkte Interoperabilität zu gewährleisten und beliebigen LTP+SMC-Endpunkten die direkte Webservice-Kommunikation mit Webservice-Anbietern und -Nutzern zu ermöglichen, die dieses Binding nicht unterstützen, wurde im Rahmen dieser Arbeit ein Ansatz für eine Binding-Konvertierung zwischen LTP+SMC und weiteren Webservice-Transport-Bindings entwickelt (s. Abbildung 6.7). Dabei erfolgt die Konvertierung ohne Konfigurationsaufwand vollständig automatisch und für alle Kommunikationspartner transparent.

6.2.1 Verwandte Arbeiten

Als verwandte Arbeiten aus dem Forschungsbereich Sensornetze sind die bereits in den Abschnitten 3.3.4 und 4.2.1 betrachteten Arbeiten von Amundson et al. [6] bzw. Kushwaha et al. [107] sowie Souza et al. [197] zu nennen. Alle Ansätze setzen eine Nachrichten- und

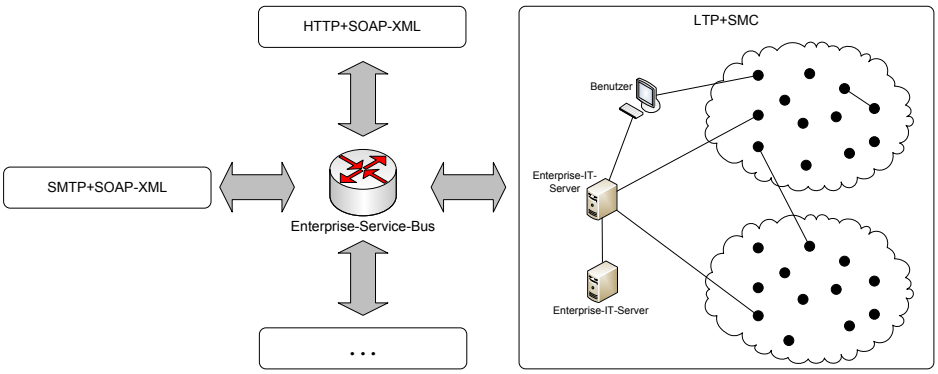


Abbildung 6.7: Ein Enterprise-Service-Bus zur automatischen, transparenten und konfigurationsfreien Webservice-Transport-Binding-Konvertierung

Protokollkonvertierung ein. Diese erfolgt jeweils an einem Gateway, das das Sensornetz mit dem Backend-System verbindet. Dabei wird zwischen den proprietären Sensornetzformaten und einer Webservice-Kommunikation im Backend konvertiert. Keiner der Ansätze verwendet jedoch eine Webservice-Kommunikation im WSN sowie eine konfigurationsfreie, applikationsunabhängige Konvertierung, sodass diese Ansätze nicht im Rahmen dieser Arbeit verwendet werden können.

In serviceorientierten Architekturen ist die Konvertierung zwischen Transportprotokollen und Nachrichtenformaten eine wesentliche Aufgabe von Enterprise-Service-Bussen (ESBs) [94]. Den Kern eines jeden ESB stellt ein einheitlicher Nachrichtenbus dar. Jede eingehende Nachricht wird zuerst in dieses Format übersetzt, bevor es vor dem Weiterleiten in das Zielformat transformiert wird. Aktuelle ESB-Implementierungen unterstützen eine Vielzahl an Webservice-Transport-Bindings sowie Konvertierungen zwischen diesen. Die schemabasierte SOAP-Kompression von LTP+SMC verhindert jedoch die direkte konfigurationsfreie Integration einer entsprechenden Konvertierung in diese ESBs, da für jede Konvertierung die jeweilige WSDL-Beschreibung eines aufgerufenen Webservices dem ESB bekannt sein muss.

6.2.2 Konvertierung zwischen LTP+SMC und weiteren Webservice-Transport-Bindings

In diesem Abschnitt wird ein Enterprise-Service-Bus präsentiert, der die automatische, transparente und konfigurationsfreie Binding-Konvertierung zwischen LTP+SMC und weiteren Bindings ermöglicht. Er umfasst sowohl die Konvertierung zwischen unterschiedlichen Transportprotokollen als auch zwischen verschiedenen Serialisierungen bzw. Komprimierungen der Webservice-Nachrichten.

Zurzeit unterstützt der ESB die Webservice-Transport-Bindings LTP+SMC sowie HTTP+SOAP. Um jedoch auch in Zukunft einfach um weitere Transport-Bindings erweiterbar zu sein, wurde analog zu klassischen ESBs ein Zwischenformat definiert, sodass nur zwischen diesem Format und jedem unterstützten Binding ein Konverter realisiert werden muss.

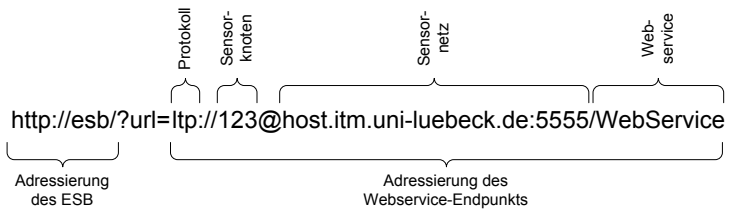


Abbildung 6.8: Adressierungsschema des Enterprise-Service-Busses zur transparenten Webservice-Transport-Binding-Konvertierung

Adressierung und Routing

Um die beiden Kernaufgaben einer Transport- und Nachrichtenkonvertierung realisieren zu können, müssen die Webservice-Nachrichten über den ESB geroutet werden. Dabei muss die Adressierung des ESB sowie der tatsächlichen Webservice-Endpunkte für alle Kommunikationspartner transparent erfolgen. Das bedeutet in diesem Zusammenhang, dass das Adressierungsschema eines jeden Bindings unverändert bleiben muss. Da Webservices URLs zur Adressierung von Endpunkten verwenden, muss auch bei der Realisierung des ESB diese Adressierungsart verwendet werden. Dabei müssen alle notwendigen Adressierungs- und Routing-Informationen für alle Kommunikationspartner in URLs codiert werden. Gleichzeitig muss die Tatsache, dass Webservice-Anbieter und -Nutzer unterschiedliche Bindings verwenden, verborgen bleiben.

Abbildung 6.8 zeigt ein Beispiel einer URL. In diesem Fall wird von einem Webservice-Nutzer, der HTTP+SOAP als Binding verwendet, ein Webservice aufgerufen, der über LTP+SMC angeboten wird. Wie Abbildung 6.9 zeigt, kann die Ende-zu-Ende-Kommunikation zwischen Client und Webservice in zwei Teilkommunikationen aufgeteilt werden. So kommuniziert der Client ausschließlich mit dem ESB, der wiederum direkt mit dem Webservice Nachrichten austauscht. Auf beiden Teilkommunikationsabschnitten wird jeweils die zu dem entsprechenden Transportprotokoll konforme URL-Notation verwendet. Um trotzdem alle für den gesamten Nachrichtenaustausch notwendigen Routing-Informationen ausschließlich über URLs zu codieren, haben die zwischen Webservice-Client und ESB verwendeten URLs den in Abbildung 6.8 dargestellten zweiteiligen Aufbau. In der Notation des Client-Protokolls wird der ESB adressiert. Um dem ESB über dieselbe URL die Anweisungen zur Weiterleitung der Nachricht sowie zum Quell-Binding zu übermitteln, wird an diese URL im Abfrageteil der Parameter „url“ angefügt. Dieser enthält die Endpunkt-URL des tatsächlichen Endpunkts in der URL-Notation des zwischen dem ESB und dem Webservice-Anbieter verwendeten Protokolls[2]. Im Parameter „compr“ können zusätzliche Informationen über die Kompression bzw. Serialisierung der vom Client empfangenen Webservice-Nachrichten angegeben werden. Zur Reduktion des Overheads werden für dieses Feld Standardwerte definiert, die bei dessen Fehlen verwendet werden. So wird beim Transportprotokoll HTTP XML-serialisiertes SOAP 1.2 (*document/literal*) und bei LTP SMC-komprimiertes SOAP 1.2 (*document/literal*) verwendet. Die Serialisierungsinformationen für den Nachrichtenaustausch zwischen ESB und Webservice werden über den dynamischen Abruf der entsprechenden WSDL-Beschreibung (s. u.) bezogen.

[2] Um diese URL als Wert eines Abfrageparameters zu codieren, muss sie maskiert werden. Aus Gründen der Lesbarkeit wurde darauf jedoch in Abbildung 6.8 verzichtet.

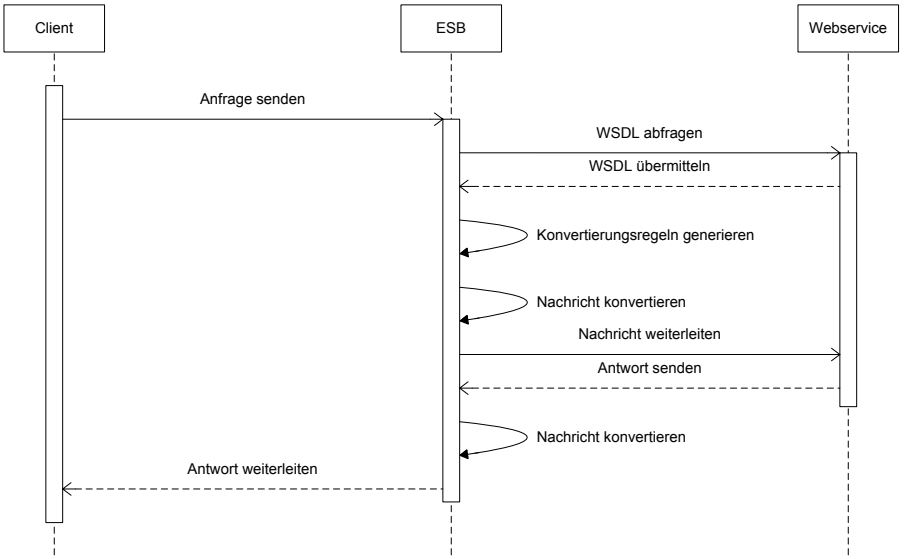


Abbildung 6.9: Ablauf einer Ende-zu-Ende-Webservice-Kommunikation über den Enterprise-Service-Bus

Auf Basis dieser Informationen sendet der Client, wie in Abbildung 6.9 dargestellt, die Anfrage-Nachricht an den ESB. Dieser verwendet die im Pfad der ursprünglichen URL codierte zweite URL und leitet die Nachricht nach der Binding-Konvertierung mit dem Zielprotokoll zum tatsächlichen Webservice-Anbieter weiter. Für den Webservice-Nutzer erfolgt das Routing vollständig transparent, da die von ihm verwendeten URLs der Standardnotation des von ihm eingesetzten Protokolls entsprechen. Für ihn stellt der ESB den Webservice-Anbieter dar. Dass dieser nicht selbst den Dienst erbringt, sondern lediglich die Nachricht an den tatsächlichen Webservice weiterleitet, bleibt verborgen. Gleiches gilt für den Webservice-Anbieter. Für ihn stellt der ESB den Nutzer dar, sodass ihm der tatsächliche Client unsichtbar bleibt.

Transportprotokollkonvertierung

Die Konvertierung zwischen verschiedenen Transportprotokollen betrifft zwei Aspekte. So müssen die Protokollinhalte, also sowohl die Header- als auch die Body-Informationen vom Quell- in das Zielprotokoll überführt werden. Diese Transformation kann relativ einfach durchgeführt werden, da die Transformationsregeln statisch und zum Zeitpunkt der Bereitstellung des ESB bekannt sind.

Die reine Transformation der Nachrichten eines Protokolls reicht jedoch bei der Realisierung einer Protokollkonvertierung nicht aus. Zusätzlich muss die Protokollsemantik berücksichtigt werden. Während es sich z. B. bei HTTP um ein synchrones Request-Response-Protokoll handelt, realisiert SMTP einen asynchronen unidirektionalen Nachrichtenaustausch.

Um eine synchrone Request-Response-Anfrage über ein asynchrones Protokoll an den Webservice weiterzuleiten, fügt der ESB den entsprechenden Nachrichten Antwortendpunkte sowie Nachrichten-IDs bei und verwaltet den Zustand dieser Transaktionen. Über dieses Verfahren

werden asynchrone Antwortpakete einer synchronen Kommunikation zugeordnet. Darüber hinaus verzögert der ESB die Antworten zu synchronen Anfragen bis zum Eintreffen der asynchronen Antwortnachrichten.

Nachrichtenkonvertierung

Bei einer vollständigen Binding-Konvertierung müssen neben der Konvertierung zwischen Transportprotokollen auch die Nutzdaten, also die Webservice-Nachrichten konvertiert werden. Im Gegensatz zu den statischen Konvertierungsregeln einer Transportprotokollkonvertierung ist eine Konvertierung von Webservice-Nachrichten für jeden Webservice individuell. Sie hängt von der konkreten Nachrichtenserialisierung sowie von der Struktur der Nachrichten ab. Insbesondere bei der Verwendung einer schemabasierten SOAP-Nachrichtenkomprimierung kann die Nachrichtenkonvertierung nur erfolgen, wenn die entsprechenden Schnittstellenbeschreibungen bekannt sind. Um eine transparente, dynamische und konfigurationsfreie Konvertierung zu ermöglichen, müssen diese Dokumente dem ESB automatisch zur Laufzeit bereitgestellt werden, sodass die Konvertierungsregeln ebenfalls zur Laufzeit automatisch aus diesen Informationen generiert werden können.

Der im Rahmen dieser Arbeit entwickelte ESB erfüllt diese Anforderungen, indem er auf selbstbeschreibenden Webservices aufbaut. Wie in Abbildung 6.9 dargestellt ruft der ESB bei der erstmaligen Konvertierung einer Nachricht eines Webservices dessen WSDL-Beschreibung automatisch ab. Auf Basis dieser Informationen generiert er die Konvertierungsregeln für diesen Dienst und wendet diese zur Nachrichtenkonvertierung an. Zur Reduzierung sowohl des Kommunikations-Overheads als auch der Laufzeit werden die Konvertierungsregeln für weitere Aufrufe desselben Dienstes zwischengespeichert.

Für den Abruf der WSDL-Dokumente von LTP+SMC-Webservices wird das in Abschnitt 6.1.2 beschriebene Protokoll LDDE eingesetzt. Selbstbeschreibende HTTP+SOAP-Webservices werden zurzeit auf Basis der Konvention realisiert, dass sich das WSDL-Dokument eines Webservices über eine HTTP-GET-Anfrage abrufen lässt, wenn „?WSDL“ an die Endpunkt-URL angefügt wird. Zur Abfrage der Schnittstellenbeschreibungen von im Vergleich zu WSNs ressourcenstarken Geräten stellt auch das in DPWS verwendete WS-MetadataExchange in Verbindung mit WS-Transfer eine für die Zukunft wichtige Alternative dar.

6.2.3 Analyse der Leistungsmerkmale

Zur Analyse der Leistungsmerkmale des im vorigen Abschnitt beschriebenen Enterprise-Service-Busses wurde das Laufzeitverhalten am Beispiel von zwei Webservices aus der Logistikdomäne untersucht. Der erste Webservice (WS 1) realisiert die Überprüfung der Bedingungen in einem Lagerhaus. Dazu übermittelt der Webservice-Client zwei Zeitstempel an den Dienst, der mit der maximalen Lagertemperatur der entsprechenden Zeitspanne antwortet. Im Gegensatz zu diesem eher einfachen Dienst weist der zweite Webservice (WS 2) einen höheren Komplexitätsgrad auf. Dieser erwartet als Eingabe die ID eines Gefahrgutes und antwortet mit dem Zeitpunkt der Einlagerung, der ID des Spediteurs, der Anzahl eingelagerter Einheiten sowie mit einer URL, die auf eine Webseite mit weiteren Informationen verweist. Beide Webservices wurden mit HTTP+SOAP bereitgestellt und von einem Nutzer verwendet,

Aufgabe		Dauer WS 1	Dauer WS 2
Erzeugung der Regeln		2.600,95	1.185,00
Anfrageverarbeitung	LTP-Paket deserialisieren	0,02	0,03
	SOAP-Nachricht dekomprimieren	0,54	0,55
	Konvertierung in Zwischenformat	0,02	0,02
Anfrageweiterleitung	Konvertierung nach HTTP+SOAP	0,01	0,01
Anwortempfang	Konvertierung in Zwischenformat	0,05	0,06
Antwortweiterleitung	Konvertierung nach LTP	1,70	1,70
	Komprimierung der SOAP-Nachricht	9,60	11,58
	Serialisierung des LTP-Pakets	0,01	0,01
Summe (ohne Erzeugung der Regeln)		11,95	13,96

Tabelle 6.3: Dauer der Konvertierung zwischen Webservice-Transport-Bindings (in ms)

der LTP+SMC verwendet. Gemessen wurde die Ausführungsdauer der einzelnen Schritte der Binding-Konvertierung auf dem bereits in den vorangegangenen Evaluationen verwendeten Mess-PC, jeweils über 10.000 Iterationen gemittelt.

Tabelle 6.3 zeigt die Ergebnisse der Laufzeitmessungen. Den bei weitem größten Zeitaufwand bei der Konvertierung verursacht mit 2,6 s (WS 1) bzw. 1,2 s (WS 2) die Erzeugung der dynamischen Konvertierungsregeln. Dieser Overhead wird dadurch verursacht, dass bei der Regelerzeugung zur Laufzeit ein SMC-Codeerzeugungsprozess durchgeführt und die generierten Klassen zur Abbildung der Daten- und Nachrichtentypen sowie zu deren Serialisierung kompiliert werden müssen. Obwohl entsprechende Wartezeiten bei der Verwendung dieser Webservices nicht akzeptabel sind, stellen diese Werte kein Problem dar. Das ist damit zu begründen, dass die Konvertierungsregeln lediglich beim ersten Aufruf eines Dienstes erzeugt und zwischengespeichert werden und so im weiteren Verlauf direkt wiederverwendet werden können.

Ohne die Berücksichtigung der Regelgenerierung verursacht die Komprimierung der SOAP-Nachrichten in beiden Messreihen mit etwas über 80 % den größten Zeitverbrauch. Dieses ist auf die Notwendigkeit von aufwendigen XML-Verarbeitungsvorgängen zurückzuführen. Insgesamt kann jedoch die Webservice-Transport-Binding-Konvertierung mit 11,95 ms bzw. 13,96 ms für die Beispiel-Webservices als sehr performant betrachtet werden. Sie kann problemlos in der Webservice-Kommunikation zwischen zwei Endpunkten, die unterschiedliche Bindings verwenden, eingesetzt werden.

6.3 Zusammenfassung

In diesem Kapitel wurde mit LDDE ein vom Autor dieser Arbeit entwickeltes Protokoll präsentiert, das die Rolle einer Webservice-Vermittlung informationstechnisch im Gesamtsystem umsetzt. Es besteht zum einen aus einem Protokoll zum dynamischen und dezentralen Auffinden von Webservices. Zum anderen realisiert LDDE selbstbeschreibende Webservices. Dazu enthält das Protokoll ein spezielles Verfahren zur Kompression von WSDL-Dokumenten, das deren Speicherung auf Sensorknoten erst ermöglicht, sowie ein Protokoll zum Abruf der

Schnittstellenbeschreibungen von den Endgeräten. LDDE zeichnet sich durch einen extrem geringen Ressourcenverbrauch aus, sodass der Einsatz in WSNs kein Problem darstellt.

Als weiteren im Rahmen dieser Arbeit entstandenen Forschungsbeitrag wurde in diesem Kapitel aufbauend auf LDDE ein Enterprise-Service-Bus vorgestellt. Dieser realisiert eine automatische, transparente und konfigurationsfreie Konvertierung von Webservice-Transport-Bindings. So kann ohne Aufwand Interoperabilität zwischen andernfalls nicht interoperablen Kommunikationspartnern erreicht und existierende Serviceinventare auch im Kontext von Sensornetzen genutzt werden, ohne zeit- und kostenintensive Integrationsprojekte durchführen zu müssen.

7 Zusammenfassung und Ausblick

In der Vision des zukünftigen Internets werden verschiedenartige Geräte, von ressourcenschwachen Sensorknoten bis zu leistungsstarken Unternehmensservern, das Internet der Dinge bilden und das Internet mit der realen Welt verbinden. Gerade drahtlose Sensornetze stellen in diesem Zusammenhang eine innovative Technologie dar, die ein sehr großes Potenzial hat, durch ihre Integration in Unternehmensanwendungen und Workflows völlig neuartige Geschäftsprozesse und Unternehmensstrategien zu ermöglichen. Trotzdem sind sie bisher kein Bestandteil von Unternehmensanwendungen. Dieses ist mit ihrer komplexen und monolithischen Anwendungsentwicklung sowie der Verwendung von proprietären, zur Enterprise-IT inkompatiblen Technologien und der daraus entstehenden hohen Entwicklungs- und Integrationskosten zu begründen.

Im Rahmen dieser Arbeit wurden konzeptionelle und technische Lösungen für o. g. Probleme entwickelt. Sie realisieren ein *dynamisches und ganzheitliches Geschäftsprozessmanagement in Sensornetzen und Enterprise-IT-Systemen.* Damit erlauben sie eine schnelle und flexible Entwicklung und Anpassung von Anwendungen und Prozessen sowie eine nahtlose Integration von Sensornetzen in Unternehmensanwendungen.

Zur informationstechnischen Ausführung von Geschäftsprozessen wird im Rahmen dieser Arbeit eine SOA als Systemarchitektur vorgeschlagen. Entscheidend ist dabei die ganzheitliche und transparente Anwendung des SOA-Konzepts sowohl im Backend als auch im Sensornetz. Das bedeutet, dass zum einen jegliche Anwendungs- und Prozessfunktionalität als in sich abgeschlossene Dienste bereitgestellt und genutzt werden. Zum anderen ist sowohl für den Dienstbereitsteller bzw. -entwickler als auch für den Dienstnutzer die jeweilige Ausführungsplattform vollständig transparent.

Die Umsetzung dieser ganzheitlichen SOA ist erst durch die technologischen Forschungsbeiträge dieser Arbeit möglich. So wird mit dem vom Autor dieser Arbeit entwickelten *Lean Transport Protocol* (LTP) eine ganzheitliche und transparente Ende-zu-Ende-Kommunikation im Sensornetz und Backend realisiert. Darauf setzt das ebenfalls im Rahmen dieser Arbeit entwickelte SOAP-Transport-Binding *LTP+SMC* auf. Es verwendet LTP zum Austausch von speziell komprimierten SOAP-Nachrichten (*SOAP Message Compression*, SMC) sowie WSDL zur Schnittstellenbeschreibung und realisiert eine ganzheitliche und transparente Interaktion zwischen Diensten im Gesamtsystem auf Basis einer Webservice-Kommunikation. Wie die in dieser Arbeit präsentierte Analyse der Leistungsmerkmale von LTP+SMC zeigt, zeichnet sich dieses Transport-Binding durch eine bedeutend höhere Effizienz als existierende Webservice-Transport-Bindings aus und kann im Gegensatz zu diesen in ressourcenbeschränkten Umgebungen verwendet werden.

Da Sensornetze durch eine hohe Dynamik und Ad-hoc-Kommunikationsbeziehungen gekennzeichnet sind, ist eine statische Bindung zwischen Dienstanbieter und -nutzer oft nicht sinnvoll.

Aus diesem Grund wurde im Rahmen dieser Arbeit das Protokoll LDDE (*Lean Description, Discovery and Exchange*) entwickelt. Es ermöglicht ein dynamisches Auffinden und Binden zwischen Kommunikationspartnern zur Laufzeit. LDDE besteht aus einem Protokoll zum dezentralen, dynamischen Auffinden von Webservices im Gesamtsystem. Zusätzlich realisiert LDDE selbstbeschreibende Webservices. Dazu enthält das Protokoll ein spezielles Verfahren zur Kompression von WSDL-Dokumenten, das ihre Speicherung auf Sensorknoten erst ermöglicht, sowie ein Protokoll zum Austausch entsprechender Dokumente. Wie die Analyse der Leistungsmerkmale zeigt, ist LDDE wesentlich effizienter als existierende Webservice-Lösungen zur Dienstvermittlung und im Gegensatz zu diesen in Sensornetzen einsetzbar.

Unabhängig von einer statischen oder dynamischen Bindung zwischen den Kommunikationspartnern kann eine Interaktion nur erfolgen, wenn beide Partner paarweise dasselbe Transport-Binding verwenden. Das Konzept dieser Arbeit sieht zwar den primären Einsatz von LTP+SMC vor. Um aber auch Interoperabilität bei der Verwendung davon abweichender Bindings zu gewährleisten und die Nutzung bereits bestehender Serviceinventare ohne die Durchführung zeit- und kostenintensiver Integrationsprojekte zu erlauben, wurde im Rahmen dieser Arbeit eine transparente und konfigurationsfreie automatische *Transport-Binding-Konvertierung* entwickelt. Sie ermöglicht sowohl die Konvertierung zwischen verschiedenen Transportprotokollen wie LTP und HTTP als auch zwischen unterschiedlichen Serialisierungsarten von SOAP-Nachrichten. Eine Evaluation der Leistungsmerkmale zeigt, dass eine Binding-Konvertierung nur einen geringen Laufzeit-Overhead verursacht.

Aufbauend auf einer ganzheitlichen und transparenten Realisierung einer SOA mit den eben genannten Webservice-Lösungen sieht das Konzept dieser Arbeit die Realisierung eines ebenfalls ganzheitlichen und transparenten Geschäftsprozessmanagements vor. Zur Realisierung einer Modellierung, Ausführung, Simulation, Überwachung und Optimierung von Prozessen aus einer fachlichen Perspektive durch Fachpersonal wurde im Rahmen dieser Arbeit mit IWFMS (*Integral Workflow Management System*) ein umfassendes Workflowmanagementsystem entwickelt. Die entscheidende Forschungsherausforderung bei der Umsetzung von IWFMS stellte die Prozessrealisierung dar. Mit BPEL und SM4RCD (*State Machine for Resource Constrained Devices*) werden zwei Sprachen zur domänenspezifischen Geschäftsprozessmodellierung verwendet. Beide realisieren eine Prozessbeschreibung durch eine rekursive Aggregation von Webservices und sind durch Fachpersonal verwendbar. Während BPEL eine flussorientierte Aggregation realisiert und einen bereits verbreiteten Standard darstellt, wurde SM4RCD im Rahmen dieser Arbeit entwickelt und verwirklicht eine Aggregation auf Basis von Zustandsautomaten. Sowohl mit BPEL als auch mit SM4RCD modellierte Prozesse erlauben eine transparente Ausführung auf Sensorknoten und Backend-Systemen.

Neben der Prozessmodellierung und -ausführung sind die Überwachung und Optimierung von Geschäftsprozessen aus fachlicher Sicht jedoch nicht weniger bedeutend. IWFMS erlaubt über verschiedene Schnittstellen die Anbindung von Überwachungs- und Optimierungswerkzeugen. Zu diesem Zweck können sowohl existierende Werkzeuge eines fachlichen Geschäftsprozessmanagements als auch verschiedene im Rahmen dieser Arbeit entwickelte Programme eingesetzt werden. In jedem Fall ist ihr Einsatz dahin gehend transparent, dass es auch bei einer Überwachung bzw. Optimierung keinen Unterschied macht, auf welcher Plattform ein Prozess ausgeführt wird.

Am Beispiel eines neuartigen Logistikprozesses zur Überwachung von Gefahrguttransporten wurden die Leistungsmerkmale sowie die Praxistauglichkeit von IWFMS analysiert. IWFMS ist wesentlich effizienter als aktuelle Webservice-basierte Workflowmanagementsysteme und erlaubt im Gegensatz zu diesen eine Prozessausführung auf Sensorknoten. Auch gegenüber einer manuellen Prozessimplementierung ist der Overhead von IWFMS bei der Prozessausführung minimal. Der größte Vorteil des Einsatzes des Konzepts dieser Arbeit und der entsprechenden Umsetzungstechnologien zeigt sich im Vergleich des Entwicklungsaufwands für den Beispielprozess unter der Verwendung von IWFMS und einer manuellen Implementierung, wie sie bisher zur Anwendungsentwicklung für Sensornetze verwendet wird. Während der Aufwand bei IWFMS 0,6 Personenmonate beträgt, erreicht er bei einer manuellen Implementierung über 15 Personenmonate.

Die Forschungsergebnisse dieser Arbeit ermöglichen ein Geschäftsprozessmanagement in Sensornetzen und Backend-Systemen. Für zukünftige Forschungen ergeben sich zwei Schwerpunkte. Als Erstes ist das Thema Sicherheit zu nennen. Bei jeglicher Kommunikation wird zurzeit weder auf Transport- noch auf Nachrichtenebene die Integrität, Vertraulichkeit oder Authentizität der Daten gewährleistet. Hier gilt es, Erweiterungsmöglichkeiten von LTP um Sicherheitsmechanismen sowie den Einsatz von WS-Security auf der Nachrichtenebene zu erforschen.

Ein weiteres Forschungsthema betrifft die Prozessmodellierung. Zurzeit werden ausschließlich Sprachen zur Orchestrierung eingesetzt. Aufbauend auf diesen sollte der Einsatz von Choreographiesprachen untersucht werden. Sie ermöglichen die Beschreibung der Kollaboration zwischen Diensten und erlauben eine weitere Abstraktion der Modellierung. Da Choreographien zur Designzeit in verschiedene interagierende Orchestrierungen überführt werden, liegt die Vermutung nahe, dass entsprechende Prozessmodelle mit den im Rahmen dieser Arbeit entwickelten Laufzeitumgebungen ohne größere Anpassungen ausgeführt werden können. Dieses gilt es jedoch zu prüfen.

Mit den im Rahmen dieser Arbeit entwickelten konzeptionellen und technischen Lösungen wird ein dynamisches, ganzheitliches Geschäftsprozessmanagement in Sensornetzen und Backend-Systemen realisiert. Es ermöglicht analog zu Abbildung 1.1 auf Seite 1 eine schnelle und flexible Entwicklung und Anpassung von Geschäftsprozessen im Gesamtsystem. Drahtlose Sensornetze oder andere ressourcenbeschränkte Geräte können so nahtlos als gleichwertige Ausführungsplattform für Geschäftsprozesse in die Unternehmensanwendungssysteme integriert werden. Damit ermöglicht die IT völlig neuartige Geschäftsprozesse, die aktiv die Unternehmensstrategie beeinflussen und Wettbewerbsvorteile für Unternehmen generieren.

A Technische Ergänzungen

Bitweise LTP-Paketserialisierung

Datenfeld	Bitfeld
huffmanEncodedStrings	0
messageID.is_present	1
messageID.encoding (if present)	2
messageID.uri.length	3
(if URI-encoded & present)	4
(in characters)	5
	6
	7
	8
messageID.uri.value	9
(if URI-encoded & present)	10
...	11
messageID.integer	12
(if integer-encoded & present)	13
	14
	15
	16
	17
	18
	19
	20
	21
	22
	23
	24
	25
	26
	27
	28
	29
	30
	31
	32
	33

	34
	35
	36
	37
	38
	39
	40
	41
	42
	43
relatesTo.is_present	44
relatesTo.encoding (if present)	45
relatesTo.uri.length	46
(if URI-encoded & present)	47
(in characters)	48
	49
	50
	51
relatesTo.uri.value	52
(if URI-encoded & present)	53
...	54
relatesTo.integer	55
(if integer-encoded & present)	56
	57
	58
	59
	60
	61
	62
	63
	64
	65
	66
	67
	68
	69
	70
	71
	72
	73
	74
	75
	76
	77
	78
	79
	80

	81
	82
	83
	84
	85
	86
to.encoding	87
to.alias.id	88
(if alias-encoded)	89
	90
	91
	92
	93
	94
	95
to.alias.rcdDevAddr	96
(if alias-encoded)	97
(field size depends on rcnAddress length)	98
to.url.length	99
(if URL-encoded)	100
(in characters)	101
	102
	103
	104
	105
	106
to.url.value	107
(if URL-encoded)	108
...	109
from.is_present	110
from.encoding (if present)	111
from.alias.id	112
(if alias-encoded & present)	113
	114
	115
	116
	117
	118
	119
from.alias.rcdDevAddr	120
(if alias-encoded & present)	121
(field size depends on rcnAddress length)	122
from.url.length	123
(if URL-encoded & present)	124
(in characters)	125
	126

	127
	128
	129
	130
from.url.value	131
(if URL-encoded & present)	132
...	133
replyTo.is_present	134
replyTo.encoding (if present)	135
replyTo.alias.id	136
(if alias-encoded & present)	137
	138
	139
	140
	141
	142
	143
replyTo.alias.rcdDevAddr	144
(if alias-encoded & present)	145
(field size depends on rcnAddress length)	146
replyTo.url.length	147
(if URL-encoded & present)	148
(in characters)	149
	150
	151
	152
	153
	154
replyTo.url.value	155
(if URL-encoded & present)	156
...	157
faultTo.is_present	158
faultTo.encoding (if present)	159
faultTo.alias.id	160
(if alias-encoded & present)	161
	162
	163
	164
	165
	166
	167
faultTo.alias.rcdDevAddr	168
(if alias-encoded & present)	169
(field size depends on rcnAddress length)	170
faultTo.url.length	171
(if URL-encoded & present)	172
(in characters)	173

	174
	175
	176
	177
	178
faultTo.url.value	179
(if URL-encoded & present)	180
...	181
fragmentProperties.is_present	182
fragmentProperties.fragmentNumber	183
(if present)	184
	185
	186
	187
	188
	189
fragmentProperties.isLastFragment (if present)	190
action.is_present	191
action.length	192
(if present)	193
(in characters)	194
	195
	196
	197
action.value	198
(if present)	199
...	200
body.length	201
(in byte)	202
	203
	204
	205
	206
	207
	208
	209
	210
	211
	212
	213
	214
Alignment	215
body.value	216
...	217
	218
	219

	220
	221
	222
	223

Literaturverzeichnis

[1] Aalst, W. M. P. d. ; Lassen, K. B.: *Translating Workflow Nets to BPEL4WS*. http://is.tm.tue.nl/staff/wvdaalst/BPMcenter/reports/2005/BPM-05-16.pdf. Version: 2005. – Abruf: 05.10.2010

[2] Aalst, W. M. P. d. ; Lassen, K. B.: Translating Unstructured Workflow Processes to Readable BPEL – Theory and Implementation. In: *International Journal of Information and Software Technology* 50 (2006), Dezember, S. 131–159

[3] Abrach, H. ; Bhatti, S. ; Carlson, J. ; Dai, H. ; Rose, J. ; Sheth, A. ; Shucker, B. ; Deng, J. ; Han, R. : MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors. In: *2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, 2003, S. 50–59

[4] Agrawal, A. ; Amend, M. ; Das, M. ; Ford, M. ; Keller, C. ; Kloppmann, M. ; König, D. ; Leymann, F. ; Müller, R. ; Pfau, G. ; Plösser, K. ; Rangaswamy, R. ; Rickayzen, A. ; Rowley, M. ; Schmidt, P. ; Trickovic, I. ; Yiu, A. ; Zeller, M. : *Web Services Human Task (WS-HumanTask), Version 1.0*. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf. Version: Juni 2007. – Abruf: 06.10.2010

[5] Agrawal, A. ; Amend, M. ; Das, M. ; Ford, M. ; Keller, C. ; Kloppmann, M. ; König, D. ; Leymann, F. ; Müller, R. ; Pfau, G. ; Plösser, K. ; Rangaswamy, R. ; Rickayzen, A. ; Rowley, M. ; Schmidt, P. ; Trickovic, I. ; Yiu, A. ; Zeller, M. : *WS-BPEL Extension for People (BPEL4People), Version 1.0*. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/BPEL4People_v1.pdf. Version: Juni 2007. – Abruf: 06.10.2010

[6] Amundson, I. ; Kushwaha, M. ; Koutsoukos, X. ; Neema, S. ; Sztipanovits, J. : Efficient Integration of Web Services in Ambient-aware Sensor Network Applications. In: *3rd IEEE/CreateNet International Workshop on Broadband Advanced Sensor Networks (BaseNets 2006)*, 2006

[7] Andrews, T. ; Curbera, F. ; Dholakia, H. ; Goland, Y. ; Klein, J. ; Leymann, F. ; Liu, K. ; Roller, D. ; Smith, D. ; Thatte, S. ; Trickovic, I. ; Weerawarana, S. : *Business Process Execution Language for Web Services Version 1.1*. http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel/ws-bpel.pdf. Version: Mai 2003. – Abruf: 04.10.2010

[8] Apache Software Foundation: *Apache Axis2*. http://ws.apache.org/axis2/. – Abruf: 03.04.2011

[9] Apache Software Foundation: *Apache Tomcat*. http://tomcat.apache.org/. – Abruf: 22.05.2011

[10] Apache Software Foundation: *Apache ODE*. http://ode.apache.org/. Version: 2009. – Abruf: 06.08.2009

[11] Avancha, S. ; Undercoffer, J. ; Joshi, A. ; Pinkston, J. : Secure Sensor Networks for Perimeter Protection. In: *Computer Networks* 43 (2003), Nr. 4, S. 421–435

[12] Backhaus, K. ; Voeth, M. : *Industriegütermarketing*. 8. Auflage. Verlag Franz Vahlen, 2007

[13] Ballinger, K. ; Brittenham, P. ; Malhotra, A. ; Nagy, W. A. ; Pharies, S. : *Web Services Inspection Language (WS-Inspection) 1.0*. http://www.ibm.com/developerworks/library/specification/ws-wsilspec/. Version: November 2001. – Abruf: 11.09.2010

[14] Barker, A. ; Besana, P. ; Robertson, D. ; Weissman, J. B.: The Benefits of Service Choreography for Data-Intensive Computing. In: *CLADE '09: Proceedings of the 7th International Workshop on Challenges of Large Applications in Distributed Environments*. New York, NY, USA, 2009, S. 1–10

[15] Becker, J. ; Delfmann, P. ; Knackstedt, R. : Konstruktion von Referenzmodellierungssprachen – Ein Ordnungsrahmen zur Spezifikation von Adaptionsmechanismen für Informationsmodelle. In: *Wirtschaftsinformatik* Jahrgang 46 (2004), Nr. 4, S. 251–264

[16] Berkau, C. : Instrumente der Datenverarbeitung für das effiziente Prozesscontrolling. In: *Kostenrechnungspraxis* (1998), Nr. 2, S. 27–32. – Sonderheft

[17] Berners-Lee, T. ; Fielding, R. ; Frystyk, H. : *Hypertext Transfer Protocol – HTTP/1.0*. RFC 1945 (Informational). http://www.ietf.org/rfc/rfc1945.txt. Version: Mai 1996 (Request for Comments)

[18] Berners-Lee, T. ; Fielding, R. ; Masinter, L. : *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986 (Standard). http://www.ietf.org/rfc/rfc3986.txt. Version: Januar 2005 (Request for Comments)

[19] Berners-Lee, T. ; Masinter, L. ; McCahill, M. : *Uniform Resource Locators (URL)*. RFC 1738 (Proposed Standard). http://www.ietf.org/rfc/rfc1738.txt. Version: Dezember 1994 (Request for Comments). – Obsoleted by RFCs 4248, 4266, updated by RFCs 1808, 2368, 2396, 3986

[20] Bieberstein, N. ; Laird, R. ; Jones, K. ; Mitra, T. : *Executing SOA – A Practical Guide for the Service-Oriented Architect*. IBM Press, 2008

[21] Bloomberg, J. ; Schmelzer, R. : *Service Oriented or Be Doomed – How Service Orientation Will Change Your Business*. John Wiley & Sons, 2006

[22] Blow, M. ; Goland, Y. ; Kloppmann, M. ; Leymann, F. ; Pfau, G. ; Roller, D. ; Rowley, M. : BPELJ – BPEL for Java / BEA und IBM. Version: März 2004. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-bpelj/ws-bpelj.pdf. – Forschungsbericht. – Abruf: 06.10.2010

[23] Blumenthal, J. ; Handy, M. ; Golatowski, F. ; Haase, M. ; Timmermann, D. : Wireless Sensor Networks – New Challenges in Software Engineering. In: *Proceedings of Emerging Technologies and Factory Automation*, 2003, S. 551–556

[24] Boehm, B. W.: *Software Engineering Economics.* Prentice Hall, 1981

[25] Boehm, B. W. ; Abts, C. ; Brown, A. W.: *Software Cost Estimation with Cocomo II.* Prentice Hall, 1990

[26] Buschmann, C. : *Zeitliches und räumliches Kontextbewusstsein in drahtlosen Sonsornetzen*, Universität zu Lübeck, Diss., 2008

[27] Buschmann, C. ; Pfisterer, D. : iSense: A Modular Hardware and Software Platform for Wireless Sensor Networks / 6. Fachgespräch Drahtlose Sensornetze der GI/ITG-Fachgruppe Kommunikation und Verteilte Systeme. Version: 2007. http://ds.informatik.rwth-aachen.de/events/fgsn07/fgsn07proc.pdf. – Forschungsbericht. – Abruf: 06.11.2010

[28] Business Process Management Initiative (BPMI): *Business Process Modeling Language (BPML).* http://www.bpmi.org/BPML. – Abruf: 14.10.2010

[29] Cearley, D. W. ; Fenn, J. ; Plummer, D. C.: *Gartner's Positions on the Five Hottest IT Topics and Trends in 2005.* http://www.gartner.com/DisplayDocument?id=480912. Version: 2005. – Abruf: 06.11.2008

[30] Cerf, V. : *ASCII format for network interchange.* RFC 20. http://www.ietf.org/rfc/rfc20.txt. Version: Oktober 1969 (Request for Comments)

[31] Chu, D. ; Popa, L. ; Tavakoli, A. ; Hellerstein, J. M. ; Levis, P. ; Shenker, S. ; Stoica, I. : The design and implementation of a declarative sensor network system. In: *Proceedings of the 5th international conference on Embedded networked sensor systems.* New York, NY, USA : ACM, 2007 (SenSys '07), S. 175–188

[32] coalesenses GmbH: *Product brief – iSense solar power system.* http://www.coalesenses.com/uploads/pdf/ProductBriefSolarModule.pdf. Version: 2009. – Abruf: 06.11.2010

[33] Crossbow Technology Inc.: *TELOSB - TelosB Mote Platform.* http://www.xbow.com/. Version: 2009. – Abruf: 06.11.2010

[34] Decker, G. ; Kopp, O. ; Borros, A. : An Introduction to Service Choreographies. In: *Information Technologie* 2 (2008), S. 122–127

[35] Decker, G. ; Kopp, O. ; Leymann, F. ; Weske, M. : BPEL4Chor – Extending BPEL for Modeling Choreographies. In: *Proceedings of the IEEE 2007 International Conference on Web Services (ICWS 2007).* IEEE Computer Society, S. 296–303

[36] Deering, S. ; Hinden, R. : *Internet Protocol, Version 6 (IPv6) Specification.* RFC 2460 (Draft Standard). http://www.ietf.org/rfc/rfc2460.txt. Version: Dezember 1998 (Request for Comments). – Updated by RFC 5095

[37] Delicato, F. C. ; Pires, P. F. ; Pirmez, L. ; Carmo, L. F.: A Service Approach for Architecting Application Independent Wireless Sensor Networks. In: *Cluster Computing* 8 (2005), Juli, S. 211–221

[38] Deutsch, P. : *GZIP file format specification version 4.3.* RFC 1952 (Informational). http://www.ietf.org/rfc/rfc1952.txt. Version: Mai 1996 (Request for Comments)

[39] Dierks, T. ; Rescorla, E. : *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). `http://www.ietf.org/rfc/rfc5246.txt`. Version: August 2008 (Request for Comments)

[40] Distributed Management Task Force (DMTF): *Web Services for Management (WS-Management) Specification*. `http://www.dmtf.org/sites/default/files/standards/documents/DSP0226_1.0.0.pdf`. Version: Februar 2008. – Abruf: 03.09.2010

[41] Dobrescu, R. ; Dobrescu, M. ; Nicolae, M. ; Popescu, D. : Using UPnP services with an intelligent sensor network node. In: *AIC'07: Proceedings of the 7th Conference on 7th WSEAS International Conference on Applied Informatics and Communications*. Stevens Point, Wisconsin, USA : World Scientific and Engineering Academy and Society (WSEAS), 2007, S. 371–374

[42] Driscoll, D. ; Mensch, A. : *Devices Profile for Web Services V 1.1*. `http://docs.oasis-open.org/ws-dd/dpws/wsdd-dpws-1.1-spec.html`. Version: 2009

[43] Dunkel, J. ; Eberhart, A. ; Fischer, S. ; Kleiner, C. ; Koschel, A. : *Systemarchitekturen für verteilte Anwendungen – Client-Server, Multi-Tier, SOA, Event-Driven Architectures, P2P, Grid, Web 2.0*. Hanser, 2008

[44] Dunkels, A. : Full TCP/IP for 8-bit architectures. In: *MobiSys '03: Proceedings of the 1st international conference on Mobile systems, applications and services*, 2003, S. 85–98

[45] Dunkels, A. ; Grönvall, B. ; Voigt, T. : Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors. In: *Proceedings of the First IEEE Workshop on Embedded Networked Sensors (Emnets-I)*. Tampa, Florida, USA, Nov. 2004

[46] Durvy, M. ; Abeillé, J. ; Wetterwald, P. ; O'Flynn, C. ; Leverett, B. ; Gnoske, E. ; Vidales, M. ; Mulligan, G. ; Tsiftes, N. ; Finne, N. ; Dunkels, A. : Making sensor networks IPv6 ready. In: ACM (Hrsg.): *Proceedings of the 6th ACM Conference on Networked Embedded Sensor Systems (ACM SenSys 2008), Raleigh, North Carolina, USA*, 2008, S. 421–422

[47] Eclipse Foundation: *Eclipse IDE*. `http://www.eclipse.org/`. – Abruf: 02.05.2011

[48] Eclipse Foundation: *Eclipse Modeling Framework Project (EMF)*. `http://www.eclipse.org/modeling/emf/`. – Abruf: 02.05.2011

[49] Eclipse Foundation: *Graphical Modeling Project (GMP)*. `http://www.eclipse.org/modeling/gmp/`. – Abruf: 02.05.2011

[50] Eisentraut, C. ; Spieler, D. : Fault, Compensation and Termination in WS-BPEL 2.0 – A Comparative Analysis. In: *Web Services and Formal Methods: 5th International Workshop, WS-FM 2008, Milan, Italy, September 4-5, 2008, Revised Selected Papers* (2009), S. 107–126

[51] Elsaify, A. ; Padhy, P. ; Martinez, K. ; Zou, G. : GWMAC – A TDMA Based MAC Protocol for a Glacial Sensor Network. In: *PE-WASUN '07: Proceedings of the 4th ACM Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks*. New York, NY, USA : ACM, 2007, S. 54–61

[52] Erl, T. : *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall, 2005

[53] Erl, T. : *SOA – Entwurfsprinzipien für serviceorientierte Architektur*. Addison-Wesley, 2008

[54] Fielding, R. ; Gettys, J. ; Mogul, J. ; Frystyk, H. ; Masinter, L. ; Leach, P. ; Berners-Lee, T. : *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). `http://www.ietf.org/rfc/rfc2616.txt`. Version: Juni 1999 (Request for Comments). – Updated by RFC 2817

[55] Fielding, R. T.: *Architectural styles and the design of network-based software architectures*, Diss., 2000

[56] Free Software Foundation, Inc.: *Gnu Compiler Collection (GCC)*. `http://gcc.gnu.org/`. – Abruf: 03.04.2011

[57] Frijters, J. : *IKVM.NET Home Page*. `http://www.ikvm.net/`. – Abruf: 21.04.2011

[58] Fuchs, G. ; German, R. : UML2 Activity Diagram based Programming of Wireless Sensor Networks. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops*, 2010

[59] Gadatsch, A. ; Knuppertz, T. ; Schnägelberger, S. : Geschäftsprozessmanagement – Umfrage zur aktuellen Situation in Deutschland. In: *Schriftenreihe des Fachbereiches Wirtschaft Sankt Augustin*. Band 9. Sankt Augustin : Fachhochschule Bonn-Rhein-Sieg, 2004

[60] Gadatsch, A. ; Knuppertz, T. ; Schnägelberger, S. : Geschäftsprozessmanagement – Umfrage zur aktuellen Situation in Deutschland, Österreich und der Schweiz. In: *Schriftenreihe des Fachbereiches Wirtschaft Sankt Augustin*. Band 14. Sankt Augustin : Fachhochschule Bonn-Rhein-Sieg, 2005

[61] Gadatsch, A. : *Grundkurs Geschäftsprozess-Management – Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker*. 5., erweiterte und überarbeitete Auflage. Wiesbaden : Friedr. Vieweg & Sohn Verlag, 2008

[62] Galler, J. ; Scheer, A.-W. : Workflow-Projekte – Vom Geschäftsprozessmodell zur unternehmensspezifischen Workflow-Anwendung. In: *Information-Management* 1 (1995), S. 20–27

[63] Gay, D. ; Levis, P. ; Behren, R. von ; Welsh, M. ; Brewer, E. ; Culler, D. : The nesC language: A holistic approach to networked embedded systems. In: *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*. New York, NY, USA : ACM, 2003 (PLDI '03), S. 1–11

[64] Gehring, H. ; Gadatsch, A. : Ein Rahmenkonzept für die Prozessmodellierung. In: *Information Management & Consulting* (1999), Nr. 4, S. 69–74

[65] Glombitza, N. ; Buschmann, C. ; Pfisterer, D. ; Fischer, S. ; Pahl, H. : Towards Service Orientation on Resource Constrained Devices. In: *WEBIST 2009 – 5th International Conference on Web Information Systems and Technologies*, 2009, S. 72–77

[66] Glombitza, N. ; Ebers, S. ; Pfisterer, D. ; Fischer, S. : Using BPEL to Realize Business Processes for an Internet of Things. In: *10th International Conference on Ad Hoc Networks and Wireless*. Paderborn, Germany, Juli 2011

[67] Glombitza, N. ; Lipphardt, M. ; Werner, C. ; Fischer, S. : Using Graphical Process Modeling for Realizing SOA Programming Paradigms in Sensor Networks. In: *Proceedings of the 6th International Conference on Wireless On demand Network Systems and Services*. Snowbird, Utah, USA, Februar 2009, S. 61–70

[68] Glombitza, N. ; Mietz, R. ; Römer, K. ; Fischer, S. ; Pfisterer, D. : Self-Description and Protocol Conversion for a Web of Things. In: *Proceedings of 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2010)*, 2010, S. 229–236

[69] Glombitza, N. ; Pfisterer, D. ; Fischer, S. : Integrating Wireless Sensor Networks into Web Service-Based Business Processes. In: *MidSens '09: Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks*. New York, NY, USA : ACM, Dezember 2009, S. 25–30

[70] Glombitza, N. ; Pfisterer, D. ; Fischer, S. : On the Integration of Service-Oriented Architectures and Sensor Networks. In: *Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS Workshop)*. Marina Del Rey, California, USA, Juni 2009

[71] Glombitza, N. ; Pfisterer, D. ; Fischer, S. : A Comprehensive Approach to Integrating Sensor Networks and Enterprise IT. In: *International Journal of Next-Generation Computing* 1 (2010), Juli, Nr. 1, S. 16–32

[72] Glombitza, N. ; Pfisterer, D. ; Fischer, S. : LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices. In: *Seventh Annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (IEEE SECON' 10)*. Boston, Massachusetts, USA, Juni 2010, S. 199–207

[73] Glombitza, N. ; Pfisterer, D. ; Fischer, S. : Using State Machines for a Model Driven Development of Web Service-Based Sensor Network Applications. In: *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops*. Cape Town, South Africa, Mai 2010, S. 2–7

[74] Guttman, E. ; Perkins, C. ; Veizades, J. ; Day, M. : *Service Location Protocol, Version 2*. RFC 2608 (Proposed Standard). `http://www.ietf.org/rfc/rfc2608.txt`. Version: Juni 1999 (Request for Comments). – Updated by RFC 3224

[75] Hackmann, G. ; Gill, C. ; Roman, G.-C. : Extending BPEL for Interoperable Pervasive Computing. In: *Proceedings of the 2007 IEEE International Conference on Pervasive Services*, 2007, S. 204–213

[76] Hammer, M. ; Champy, J. : *Business Reengineering – Die Radikalkur für das Unternehmen*. 2. Auflage. Frankfurt/Main : Campus-Verlag, 1994

[77] Han, C.-C. ; Kumar, R. ; Shea, R. ; Kohler, E. ; Srivastava, M. : A dynamic operating system for sensor nodes. In: *Proceedings of the 3rd international conference on Mobile systems, applications, and services (MobiSys '05)*. New York, NY, USA : ACM, 2005, S. 163–176

[78] Harmon, P. : Analyzing Activities. In: *Business Process Trends* 1 (2003), April, Nr. 4, S. 1–12

[79] Hoeller, N. ; Reinke, C. ; Neumann, J. ; Groppe, S. ; Werner, C. ; Linnemann, V. : Efficient XML data and query integration in the wireless sensor network engineering process. In: *International Journal of Web Information Systems* 6 (2010), Nr. 4, S. 319–358

[80] Horstmann, M. ; Kirtland, M. : *DCOM Architecture*. `http://msdn.microsoft.com/de-de/library/ms809311%28en-us%29.aspx`. Version: Juli 1997. – Abruf: 05.09.2009

[81] Huffman, D. A.: A Method for the Construction of Minimum-Redundancy Codes. In: *Proceedings of the Institute of Radio Engineers* 40 (1952), Nr. 9, S. 1098–1101

[82] Hui, J. W. ; Culler, D. E.: IP is dead, long live IP for wireless sensor networks. In: *SenSys'08: Proceedings of the 6th ACM conference on Embedded network sensor systems*. New York, NY, USA : ACM, 2008, S. 15–28

[83] IBM, BEA Systems, Microsoft, Arjuna, Hitachi, IONA: *Web Services Transactions Specifications*. `http://www.ibm.com/developerworks/library/specification/ws-tx/`. Version: August 2005. – Abruf: 09.10.2010

[84] IDS Scheer AG: *Große Umfrage bei deutschen IT-Entscheidern zum Stand des Geschäftsprozessmanagements*. `http://www.ids-scheer.de/de/Meldungen/Investitionen_fuer_Geschaeftsprozessoptimierung_steigen/2225.html?referer=8731`. Version: 2003. – Abruf: 03.04.2010

[85] IEEE 802.15 Working Group: *IEEE 802.15 WPAN Task Group 4 (TG4)*. `http://www.ieee802.org/15/pub/TG4.html`. Version: März 2004. – Abruf: 26.10.2010

[86] Institut für Telematik, Universität zu Lübeck; TraDaV GmbH; coalesenses GmbH: *Lübecker Logistik Datendrehscheibe – Meilensteinbericht 2*. Projektbericht, Juni 2010

[87] International Organization for Standardization: *Information technology – Syntactic meta language – Extended BNF*. `http://standards.iso.org/ittf/PubliclyAvailableStandards/s026153_ISO_IEC_14977_1996(E).zip`. Version: Dezember 1996. – Abruf: 31.03.2011

[88] International Organization for Standardization: *ISO/IEC 29341: UPnP Device Architecture – Part 1: UPnP Device Architecture Version 1.0*. Dezember 2008

[89] International Organization for Standardization: *Structured Query Language (ISO/IEC 9075-1:2008)*. 2008

[90] International Telecommunication Union: *ITU-T Recommendation Z.120: Formal Description Techniques (FDT) – Message Sequence Chart (MSC).* http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-Z.120-200404-I!!PDF-E&type=items. Version: April 2004. – Abruf: 15.10.2010

[91] International Telecommunication Union (ITU): *Recommendation X.891: Generic Applications of ASN.1 – Fast Infoset.* Mai 2005

[92] Johnson, D. ; Perkins, C. ; Arkko, J. : *Mobility Support in IPv6.* RFC 3775 (Proposed Standard). http://www.ietf.org/rfc/rfc3775.txt. Version: Juni 2004 (Request for Comments)

[93] Johnstone, I. ; Nicholson, J. ; Shehzad, B. ; Slipp, J. : Experiences From a Wireless Sensor Network Deployment in a Petroleum Environment. In: *IWCMC '07: Proceedings of the 2007 International Conference on Wireless Communications and Mobile Computing.* New York, NY, USA : ACM, 2007, S. 382–387

[94] Josuttis, N. : *SOA in der Praxis – System-Design für verteilte Anwendungen.* dpunkt.verlag, 2008

[95] Keller, G. ; Nüttgens, M. ; Scheer, A.-W. : *Semantische Prozeßmodellierung auf der Grundlage Ereignisgesteuerter Prozeßketten (EPK).* Saarbrücken : Institut für Wirtschaftsinformatik, Universität Saarbrücken, 1992 (Veröffentlichungen des Instituts für Wirtschaftsinformatik ; 89)

[96] Keller, G. ; Teufel, T. : *SAP R/3 prozeßorientiert anwenden – iteratives Prozeß-Prototyping zur Bildung von Wertschöpfungsketten.* 2. korrigierte Auflage. Bonn : Addison-Wesley, 1998

[97] Kim, J.-H. ; Huemer, C. : From an ebXML BPSS Choreography to a BPEL-based Implementation. In: *ACM SIGecom Exchanges* 5 (2004), Nr. 2, S. 1–11

[98] Kim, S. ; Pakzad, S. ; Culler, D. ; Demmel, J. ; Fenves, G. ; Glaser, S. ; Turon, M. : Wireless Sensor Networks for Structural Health Monitoring. In: *SenSys '06: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems.* New York, NY, USA : ACM, 2006, S. 427–428

[99] Kim, S. ; Pakzad, S. ; Culler, D. ; Demmel, J. ; Fenves, G. ; Glaser, S. ; Turon, M. : Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In: *IPSN '07: Proceedings of the 6th International Conference on Information Processing in Sensor Networks.* New York, NY, USA : ACM, 2007, S. 254–263

[100] Klensin, J. : *Simple Mail Transfer Protocol.* RFC 5321 (Draft Standard). http://www.ietf.org/rfc/rfc5321.txt. Version: Oktober 2008 (Request for Comments)

[101] Kloppmann, M. ; König, D. ; Pfau, G. : Business Process Standards – Current Landscape. In: *it - Information Technology* 50 (2008), Nr. 2, S. 93–98

[102] Kloppmann, M. ; Koenig, D. ; Leymann, F. ; Pfau, G. ; Rickayzen, A. ; Riegen, C. von ; Schmidt, P. ; Trickovic, I. : WS-BPEL Extension for Sub-processes – BPEL-SPE / IBM und SAP. Version: September 2005. http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-bpelsubproc/ws-bpelsubproc.pdf. – Forschungsbericht. – Abruf: 06.10.2010

[103] Kopp, O. : *Abbildung von EPKs nach BPEL anhand des Prozessmodellierungswerkzeugs Nautilus*, Institut für Architektur von Anwendungssystemen, Universität Stuttgart, Diplomarbeit, 2005

[104] Krcmar, H. : *Informationsmanagement.* 4. Auflage. Berlin : Springer, 2005

[105] Kröller, A. ; Pfisterer, D. ; Buschmann, C. ; Fekete, S. P. ; Fischer, S. : Shawn: A new approach to simulating wireless sensor networks. In: *Design, Analysis, and Simulation of Distributed Systems 2005, Part of the SpringSim 2005*

[106] Kushalnagar, N. ; Montenegro, G. ; Schumacher, C. : *IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals.* RFC 4919 (Informational). `http://www.ietf.org/rfc/rfc4919.txt`. Version: August 2007 (Request for Comments)

[107] Kushwaha, M. ; Amundson, I. ; Koutsoukos, X. ; Neema, S. ; Sztipanovits, J. : OASiS: A Programming Framework for Service-Oriented Sensor Networks. In: *Proceedings of the Second International Conferences of Communication System Software and Middleware (Comsware 2007).* Bengalore, India, 2007

[108] Leach, P. ; Mealling, M. ; Salz, R. : *A Universally Unique IDentifier (UUID) URN Namespace.* RFC 4122 (Proposed Standard). `http://www.ietf.org/rfc/rfc4122.txt`. Version: Juli 2005 (Request for Comments)

[109] Lee, I. ; Sokolsky, O. : Research challenges in embedded and hybrid systems. In: *SIGBED Rev.* 1 (2004), April, S. 1–5

[110] Levis, P. ; Madden, S. ; Polastre, J. ; Szewczyk, R. ; Whitehouse, K. ; Woo, A. ; Gay, D. ; Hill, J. ; Welsh, M. ; Brewer, E. ; Culler, D. : TinyOS: An Operating System for Sensor Networks. In: *Ambient Intelligence* (2005), S. 115–148

[111] Levis, P. ; Culler, D. : Maté: a tiny virtual machine for sensor networks. In: *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems (ASPLOS-X).* New York, NY, USA : ACM, 2002, S. 85–95

[112] Leymann, F. : Web Services Flow Language (WSFL) / IBM Corporation. Version: 2001. `http://www-01.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`. – Forschungsbericht. – Abruf: 14.10.2008

[113] Liefke, H. ; Suciu, D. : XMill: An Efficient Compressor for XML Data. In: Chen, W. (Hrsg.) ; Naughton, J. F. (Hrsg.) ; Bernstein, P. A. (Hrsg.): *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data.* Dallas, Texas, USA : ACM, Mai 2000, S. 153–164

[114] Lipphardt, M. ; Glombitza, N. ; Neumann, J. ; Werner, C. : Demo Abstract: A Service-oriented Operating System and an Application Development Infrastructure for Wireless Sensor Networks. In: *The 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*, 2009, S. 309–310

[115] Lipphardt, M. ; Glombitza, N. ; Neumann, J. ; Werner, C. ; Fischer, S. : A Service-Oriented Operating System and an Application Development Infrastructure for Distributed Embedded Systems. In: *17. Fachtagung "Kommunikation in Verteilten Systemen 2011"(KiVS'11)*, 2011, S. 26–37

[116] Lipphardt, M. ; Hellbrueck, H. ; Pfisterer, D. ; Ransom, S. ; Fischer, S. : Practical Experiences on Mobile Inter-Body-Area-Networking. In: *Proceedings of the Second International Conference on Body Area Networks (BodyNets'07)*, 2007

[117] Lorincz, K. ; Malan, D. ; Fulford-Jones, T. ; Nawoj, A. ; Clavel, A. ; Shnayder, V. ; Mainland, G. ; Welsh, M. ; Moulton, S. : Sensor Networks for Emergency Response – Challenges and Opportunities. In: *IEEE Pervasive Computing* 3 (2004), Oktober–Dezember, Nr. 4, S. 16–23

[118] Losilla, F. ; Vicente-Chicote, C. ; Álvarez, B. ; Iborra, A. ; Sánchez, P. : Wireless Sensor Network Application Development: An Architecture-Centric MDE Approach. In: *ECSA* Bd. 4758, Springer, 2007 (Lecture Notes in Computer Science), S. 179–194

[119] Luckenbach, T. ; Gober, P. ; Arbanowski, S. ; Kotsopoulos, A. ; Kim, K. : TinyREST - A Protocol for Integrating Sensor Networks into the Internet. In: *Proceedings of REALWSN*, 2005

[120] Luckham, D. : *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley, 2002

[121] Madden, S. R. ; Franklin, M. J. ; Hellerstein, J. M. ; Hong, W. : TinyDB: an acquisitional query processing system for sensor networks. In: *ACM Trans. Database Syst.* 30 (2005), Nr. 1, S. 122–173

[122] Mainwaring, A. ; Culler, D. ; Polastre, J. ; Szewczyk, R. ; Anderson, J. : Wireless Sensor Networks for Habitat Monitoring. In: *WSNA '02: Proceedings of the 1st ACM International Workshop on Wireless Sensor Networks and Applications*. New York, NY, USA : ACM Press, 2002, S. 88–97

[123] Malan, D. ; Fulford-jones, T. ; Welsh, M. ; Moulton, S. : CodeBlue – An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care. In: *International Workshop on Wearable and Implantable Body Sensor Networks*, 2004

[124] Margaria, T. ; Steffen, B. : Service Engineering – Linking Business and IT. In: *30th Annual IEEE/NASA Software Engineering Workshop SEW '06*, 2006, S. 33–36

[125] Margaria, T. ; Steffen, B. : Service Engineering – Linking Business and IT. In: *IEEE Computer* Oktober (2006), S. 45–55

[126] Marin-Perianu, R. ; Scholten, H. ; Havinga, P. : Prototyping Service Discovery and Usage in Wireless Sensor Networks. In: *Proceedings of the 32nd IEEE Conference on Local Computer Networks*. Washington, DC, USA : IEEE Computer Society, 2007, S. 841–850

[127] Mattern, F. ; Floerkemeier, C. : Vom Internet der Computer zum Internet der Dinge. In: *Informatik-Spektrum* 33 (2010), April, Nr. 2, S. 107–121

[128] Mattern, F. ; Römer, K. : Drahtlose Sensornetze. In: *Informatik-Spektrum* 26 (2003), S. 191–194

[129] Mealy, G. H.: A Method to Synthesizing Sequential Circuits. In: *Bell Systems Technical Journal* 34 (1955), Nr. 5, S. 1045–1079

[130] Melzer, I. ; Eberhard, S. ; Thiele, A. H.: *Service-orientierte Architekturen mit Web Services*. 3. Auflage. Spektrum-Verlag, 2008

[131] Microsoft Corporation: *Microsoft Excel*. http://office.microsoft.com/en-us/excel. – Abruf: 02.05.2011

[132] Microsoft Corporation: *Microsoft Visio*. http://office.microsoft.com/de-de/visio/. – Abruf: 02.05.2011

[133] Microsoft Corporation: *Microsoft Visual Studio*. http://www.microsoft.com/germany/visualstudio/. – Abruf: 02.05.2011

[134] Microsoft Corporation: *Microsoft Windows Mobile*. http://www.microsoft.com/windowsmobile/de-de/default.mspx. – Abruf: 11.12.2009

[135] Microsoft Corporation: Orchestration Developer Reference. http://msdn.microsoft.com/en-us/library/ee268148%28v=BTS.10%29.aspx. – Forschungsbericht. – Abruf: 06.10.2010

[136] Microsoft Corporation: *Microsoft Message Queuing (MSMQ)*. http://msdn.microsoft.com/en-us/library/ms711472.aspx. Version: Juli 2009. – Abruf: 14.10.2010

[137] Microsoft Corporation: *.NET Framework 4*. http://msdn.microsoft.com/de-de/netframework/default.aspx. Version: May 2009. – Abruf: 14.10.2010

[138] Microsoft Corporation: *Windows Workflow Foundation*. http://msdn.microsoft.com/en-us/netframework/aa663328.aspx. Version: 2010. – Abruf: 14.10.2010

[139] Mockapetris, P. : *Domain names - implementation and specification*. RFC 1035 (Standard). http://www.ietf.org/rfc/rfc1035.txt. Version: November 1987 (Request for Comments). – Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2845, 3425, 3658, 4033, 4034, 4035, 4343

[140] Molina, F. J. ; Barbancho, J. ; Luque, J. : Automated Meter Reading and SCADA Application for Wireless Sensor Network. In: Pierre, S. (Hrsg.) ; Barbeau, M. (Hrsg.) ; Kranakis, E. (Hrsg.): *Ad-Hoc, Mobile, and Wireless Networks* Bd. 2865. Berlin / Heidelberg : Springer, 2003, S. 223–234

[141] Moore, E. F.: Gedanken-experiments on Sequential Machines. In: *Automata Studies,Annals of Mathematical Studies* 34 (1956), S. 129–153

[142] Moritz, G. ; Zeeb, E. ; Prüter, S. ; Golatowski, F. ; Timmermann, D. ; Stoll, R. : Devices Profile for Web Services in Wireless Sensor Networks: Adaptations and Enhancements. In: *IEEE Conference on Emerging Technologies & Factory Automation*, 2009, S. 1–8

[143] Naumowicz, T. ; Schröter, B. ; Schiller, J. : Demo Abstract: Software Factory for Wireless Sensor Networks. In: *6th European Conference on Wireless Sensor Networks (EWSN 2009)*, 2009

[144] Naumowicz, T. ; Schröter, B. ; Schiller, J. : Poster Abstract: Prototyping a Software Factory for Wireless Sensor Networks. In: *7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009)*, 2009

[145] Oberweis, A. : *Modellierung und Ausführung von Workflows mit Petri-Netzen*. Stuttgart, 1996

[146] Object Management Group (OMG): *Unified Modeling Language (UML)*. http://www.omg.org/spec/UML/. – Abruf: 22.07.2010

[147] Object Management Group (OMG): *Common Object Request Broker Architecture: Core Specification*. http://www.omg.org/docs/formal/04-03-12.pdf. Version: März 2004

[148] Object Management Group (OMG): *Business Process Model and Notation (BPMN) – Version 1.2*. http://www.omg.org/spec/BPMN/1.2/PDF/. Version: Januar 2009. – Abruf: 24.04.2010

[149] Oracle: *Metro User Guide*. http://metro.java.net/guide/. Version: März 2011. – Abruf: 20.03.2011

[150] Oracle Corporation: *NetBeans IDE*. http://netbeans.org/. – Abruf: 02.05.2011

[151] Oracle Corporation: *JSR 220 – Enterprise JavaBeans™, Version 3.0*. http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html. Version: Mai 2006. – 14.10.2010

[152] Organization for the Advancement of Structured Information Standards (OASIS): *OASIS-Internetpräsenz*. http://www.oasis-open.org/. – Abruf: 02.09.2010

[153] Organization for the Advancement of Structured Information Standards (OASIS): *RELAX NG Specification*. http://www.oasis-open.org/committees/relax-ng/spec-20011203.html. Version: Dezember 2003. – Abruf: 09.09.2010

[154] Organization for the Advancement of Structured Information Standards (OASIS): *UDDI Version 3.0.2*. http://uddi.org/pubs/uddi-v3.0.2-20041019.htm. Version: Oktober 2004. – Abruf: 11.09.2010

[155] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Distributed Management: Management Using Web Services (MUWS 1.0) Part 1*. http://docs.oasis-open.org/wsdm/2004/12/wsdm-muws-part1-1.0.pdf. Version: März 2005. – Abruf: 03.09.2010

[156] Organization for the Advancement of Structured Information Standards (OASIS): *Reference Model for Service Oriented Architecture 1.0 – OASIS Standard, 12 October 2006*. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf. Version: 2006. – Abruf: März 2010

[157] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Distributed Management: Management of Web Services*. http://docs.oasis-open.org/wsdm/wsdm-mows-1.1-spec-os-01.htm. Version: August 2006. – Abruf: 03.09.2010

[158] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Security – SOAP Message Security 1.1*. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf. Version: Februar 2006. – Abruf: 03.09.2010

[159] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Business Process Execution Language Version 2.0*. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. Version: April 2007. – Abruf: 03.10.2010

[160] Organization for the Advancement of Structured Information Standards (OASIS): *WS-SecurityPolicy 1.2*. http://docs.oasis-open.org/ws-sx/ws-securitypolicy/200702/ws-securitypolicy-1.2-spec-os.html. Version: Juli 2007. – Abruf: 03.09.2010

[161] Organization for the Advancement of Structured Information Standards (OASIS): *SOAP-over-UDP Version 1.1*. http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/os/wsdd-soapoverudp-1.1-spec-os.html. Version: Juli 2009. – Abruf: 03.09.2010

[162] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Atomic Transaction (WS-AtomicTransaction) Version 1.2*. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os/wstx-wsat-1.2-spec-os.html. Version: Februar 2009. – Abruf: 03.09.2010

[163] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Dynamic Discovery (WS-Discovery) Version 1.1*. http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html. Version: Juli 2009. – Abruf: 11.09.2010

[164] Organization for the Advancement of Structured Information Standards (OASIS): *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2*. http://docs.oasis-open.org/ws-rx/wsrm/v1.2/wsrm.html. Version: Februar 2009. – Abruf: 03.09.2010

[165] Owen, M. ; Raj, J. : *BPMN and Business Process Management – Introduction to the New Business Process Modeling Standard*. http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf. Version: 2003. – Abruf: 22.08.2010

[166] Pandey, K. ; Patel, S. : A Novel Design of Service Oriented and Message Driven Middleware for Ambient Aware Wireless Sensor Network. In: *International Journal of Recent Trends in Engineering (IJRTE)*, 2009

[167] Papazoglou, M. P.: *Web Services – Principles and Technology*. Pearson, 2008

[168] Peltz, C. : Web Services Orchestration and Choreography. In: *IEEE Computer* 36 (2003), S. 46–52. – ISSN 0018–9162

[169] Perkins, C. : *IP Mobility Support for IPv4*. RFC 3344 (Proposed Standard). http://www.ietf.org/rfc/rfc3344.txt. Version: August 2002 (Request for Comments). – Updated by RFC 4721

[170] Petri, C. A.: *Kommunikation mit Automaten*. Bonn : Mathematisches Institut der Universität Bonn, 1962

[171] Pfisterer, D. : *Comprehensive Development Support for Wireless Sensor Networks*, Universität Lübeck, Technisch-Naturwissenschaftliche Fakultät, Diss., 2007

[172] Pfisterer, D. ; Wegner, M. ; Hellbrück, H. ; Werner, C. ; Fischer, S. : Energy-optimized Data Serialization for Heterogeneous WSNs Using Middleware Synthesis. In: *Proceedings of The Sixth Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net' 07)*, 2007, S. 180–187

[173] Pister, K. ; Kahn, J. ; Boser, B. : *SMART DUST – Autonomous sensing and communication in a cubic millimeter*. `http://robotics.eecs.berkeley.edu/~pister/SmartDust/`. – Abruf: 06.11.2010

[174] Postel, J. : *User Datagram Protocol*. RFC 768 (Standard). `http://www.ietf.org/rfc/rfc768.txt`. Version: August 1980 (Request for Comments)

[175] Postel, J. : *Internet Protocol*. RFC 791 (Standard). `http://www.ietf.org/rfc/rfc791.txt`. Version: September 1981 (Request for Comments). – Updated by RFC 1349

[176] Postel, J. : *Transmission Control Protocol*. RFC 793 (Standard). `http://www.ietf.org/rfc/rfc793.txt`. Version: September 1981 (Request for Comments). – Updated by RFCs 1122, 3168

[177] Postel, J. ; Reynolds, J. : *File Transfer Protocol*. RFC 959 (Standard). `http://www.ietf.org/rfc/rfc959.txt`. Version: Oktober 1985 (Request for Comments). – Updated by RFCs 2228, 2640, 2773, 3659

[178] Priyantha, N. B. ; Kansal, A. ; Goraczko, M. ; Zhao, F. : Tiny Web Services: Design and Implementation on Interoperable and Evolvable Sensor Networks. In: *SenSys'08: Proceedings of the 6th ACM conference on Embedded network sensor systems*, 2008

[179] Raghunathan, V. ; Kansal, A. ; Hsu, J. ; Friedman, J. ; Srivastava, M. : Design considerations for solar energy harvesting wireless embedded systems. In: *In IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, IEEE Press, 2005, S. 457–462

[180] Ratnasamy, S. ; Karp, B. ; Shenker, S. ; Estrin, D. ; Govindan, R. ; Yin, L. ; Yu, F. : Data-centric storage in sensornets with GHT, a geographic hash table. In: *Mob. Netw. Appl.* 8 (2003), August, S. 427–442

[181] Rothenpieler, P. ; Krüger, D. ; Pfisterer, D. ; Fischer, S. ; Dudek, D. ; Haas, C. ; Kuntz, A. ; Zitterbart, M. : FleGSens – Secure Area Monitoring Using Wireless Sensor Networks. In: *Proceedings of the International Conference on Sensor Networks, Information, and Ubiquitous Computing (ICSNIUC 2009)* Bd. 56, World Academy of Science, Engineering and Technology, 2009, S. 81–92

[182] Rothenpieler, P. ; Krüger, D. ; Pfisterer, D. ; Fischer, S. ; Dudek, D. ; Haas, C. ; Kuntz, A. ; Zitterbart, M. : FleGSens – Secure Area Monitoring Using Wireless Sensor Networks. In: *Proceedings of the 4th Safety and Security Systems in Europe*, Micro Materials Center Berlin at Fraunhofer Institute IZM and Fraunhofer ENAS, 2009, S. 136–139

[183] Rothenpieler, P. ; Zwingelberg, H. ; Carlson, D. ; Schrader, A. ; Fischer, S. : Datenschutz im AAL Service System SmartAssist. In: *4. Deutscher AAL-Kongress Innovative Assistenzsysteme im Dienste des Menschen - Von der Forschung für den Markt (AAL 2011)*. Berlin, Germany, 2010

[184] Rump, F. J.: *Geschäftsprozeßmanagement auf der Basis ereignisgesteuerter Prozeßketten – Formalisierung, Analyse und Ausführung von EPKs*. Stuttgart [u.a.], Diss., 1999

[185] Sadilek, D. A.: Domain-Specific Languages for Wireless Sensor Networks. In: *Modellierung*, 2008, S. 237–241

[186] Sadilek, D. A.: Prototyping domain-specific language semantics. In: *OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. New York, NY, USA : ACM, 2008, S. 895–896

[187] SAP AG: *SAP R/3 Enterprise Release 4.70 – Documentation for SAP R/3 and R/3 Enterprise 4.70*. `http://help.sap.com/content/documentation/r3/docu_sbs_r3_470.htm`. Version: 2004. – Abruf: 24.04.2010

[188] Scheer, A.-W. : *ARIS – Modellierungsmethoden, Metamodelle, Anwendungen*. 4. Auflage. Berlin : Springer, 2001

[189] Scheer, A.-W. : *Architektur integrierter Informationssysteme – Grundlagen der Unternehmensmodellierung*. 2., verb. Auflage. Berlin [u.a.] : Springer, 1992

[190] Scheer, A.-W. : *ARIS –House of business engineering*. IWI (Veröffentlichungen des Instituts für Wirtschaftsinformatik (IWI) im Institut für Empirische Wirtschaftsforschung an der Universität des Saarlandes ; 133). `http://www.iwi.uni-sb.de/Download/iwihefte/heft133.pdf`. – Abruf: 24.04.2010

[191] Scheer, A.-W. : *ARIS – Vom GeschäftsProzess zum Anwendungssystem*. 3. Auflage. Berlin : Springer, 1998

[192] Scheer, A.-W. ; Jost, W. : Geschäftsprozessmodellierung innerhalb einer Unternehmensarchitektur. In: Vossen, G. (Hrsg.) ; Becker, J. (Hrsg.): *Geschäftsprozessmodellierung und Workflow-Management, Modelle, Methoden, Werkzeuge*. Bonn, 1996, S. 29–46

[193] Schlüter, F. ; Schneider, H. : Produktionsplanung und -steuerung. In: Schneider, H. (Hrsg.): *Produktionsmanagement in kleinen und mittleren Unternehmen*. Stuttgart, 2000, S. 227–286

[194] Shapiro, R. : *A Technical Comparison of XPDL, BPML and BPEL4WS*. `http://xml.coverpages.org/Shapiro-XPDL.pdf`. Version: August 2002. – Abruf: 14.10.2010

[195] Shelby, Z. ; Hartke, K. ; Bormann, C. ; Frank, B. : *Constrained Application Protocol (CoAP)*. `https://datatracker.ietf.org/doc/draft-ietf-core-coap/?include_text=1`. Version: März 2011. – Abruf: 20.03.2011

[196] Sixsmith, A. ; Johnson, N. : A Smart Sensor to Detect the Falls of the Elderly. In: *IEEE Pervasive Computing* 3 (2004), April–Juni, Nr. 2, S. 42–47

[197] Souza, L. M. S. ; Spiess, P. ; Guinard, D. ; Köhler, M. ; Karnouskos, S. ; Savio, D. : SOCRADES: A Web Service Based Shop Floor Integration Infrastructure. In: *IOT* Bd. 4952, Springer, 2008

[198] Spiess, P. ; Karnouskos, S. : Maximizing the Business Value of Networked Embedded Systems through Process-Level Integration into Enterprise Software. In: *2nd International Conference on Pervasive Computing and Applications*, 2007, S. 536–541

[199] Spiess, P. ; Nguyen, D. K. ; Weber, I. ; Markovic, I. ; Beigl, M. : Modelling, Simulation, and Performance Analysis of Business Processes Involving Ubiquitous System. In: *Advanced Information Systems Engineering* Bd. 5074/2008, Springer Berlin / Heidelberg (Lecture Notes in Computer Science)

[200] Spiess, P. ; Vogt, H. ; Jutting, H. : Integrating sensor networks with business processes. In: *Real-World Sensor Networks Workshop at ACM MobiSys*, 2006

[201] Stahl, T. ; Völter, M. ; Efftinge, S. ; Haase, A. : *Modellgetriebene Softwareentwicklung – Techniken, Engineering, Management.* 2. Auflage. dpunkt.verlag, 2007

[202] Steffen, B. ; Narayan, P. : Full Life-Cycle Support for End-to-End Processes. In: *IEEE Computer* 40 (2007), Nr. 11, S. 64–73

[203] Stein, S. ; Ivanov, K. : EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In: *Software Engineering*, 2007, S. 75–82

[204] Österle, H. : *Business Engineering – Prozeß- und Systementwicklung.* Bd. 1. Berlin : Springer, 1995

[205] Sun Microsystems: *OpenJDK.* `http://openjdk.java.net/`. – Abruf: 22.05.2011

[206] Sun Microsystems: *Remote Method Invocation (RMI) Documentation.* `http://java.sun.com/products/jdk/rmi/`

[207] Sun Microsystems: *Java Message Service Specifcation.* `http://java.sun.com/products/jms/docs.html`. Version: April 2002. – Abruf: 03.09.2010

[208] Sun Microsystems: *SOAP/TCP v1.0.* `http://java.sun.com/webservices/reference/apis-docs/soap-tcp-v1.0.pdf`. Version: Mai 2007. – Abruf: 03.09.2010

[209] Szewczyk, R. ; Mainwaring, A. ; Polastre, J. ; Anderson, J. ; Culler, D. : An Analysis of a Large Scale Habitat Monitoring Application. In: *In Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004, S. 214–226

[210] Szewczyk, R. ; Polastre, J. ; Mainwaring, A. ; Culler, D. : Lessons From A Sensor Network Expedition. In: *Proceedings of the First European Workshop on Sensor Networks (EWSN)*, 2004, S. 307–322

[211] Tanenbaum, A. S.: *Computernetzwerke.* 4., überarb. Auflage. München : Pearson, 2009

[212] Tanenbaum, A. S.: *Moderne Betriebssysteme.* 3., aktualis. Aufl. München : Pearson Studium, 2010

[213] Tanenbaum, A. S. ; Steen, M. v.: *Verteilte Systeme – Prinzipien und Paradigmen*. 2., aktualisierte Auflage. München : Pearson Studium, 2008

[214] Thomas, O. ; Leyking, K. ; Dreifus, F. : Prozessmodellierung im Kontext service-orientierter Architekturen. In: *HMD: Praxis der Wirtschaftsinformatik*. Heidelberg : dpunkt-Verlag, Februar 2007 (Band 44.2007, 253), S. 37–46

[215] Tolani, P. M. ; Haritsa, J. R.: XGRIND: A Query-Friendly XML Compressor. In: *Proceedings of the 18th International Conference on Data Engineering*. San Jose, CA, USA : IEEE Computer Society, Februar 2002, S. 225–234

[216] United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT); Organization for the Advancement of Structured Information Standards (OASIS): *ebXML Business Process Specification Schema Version 1.01*. Mai 2001. – Abruf: 15.10.2010

[217] Universität zu Lübeck, Institut für Telematik: *Lübecker Logistik Datendrehscheibe (L2D2)*. `http://www.itm.uni-luebeck.de/projects/l2d2/`. Version: 2009. – Abruf: 27.10.2010

[218] Web Services Interoperability Organization (WS-I): *WS-I-Internetpräsenz*. `http://www.ws-i.org/`. – Abruf: 02.09.2010

[219] Werner, C. : *Optimierte Protokolle für Web Services mit begrenzten Datenraten*, Universität zu Lübeck, Diss., Juni 2006

[220] Werner, C. ; Buschmann, C. ; Brandt, Y. ; Fischer, S. : Compressing SOAP Messages by using Pushdown Automata. In: *ICWS '06: Proceedings of the IEEE International Conference on Web Services*. Washington, DC, USA : IEEE Computer Society, 2006, S. 19–28

[221] Werner, C. ; Buschmann, C. ; Fischer, S. : Advanced Data Compression Techniques for SOAP Web Services. In: Zhang, L.-J. (Hrsg.): *Modern Technologies in Web Services Research*. IGI Publishing, 2007. – ISBN 978–1–59904–280–0, S. 76–97

[222] Werner, C. ; Buschmann, C. ; Jäcker, T. : Enhanced Transport Bindings for Efficient SOAP Messaging. In: *IEEE International Conference on Web Services* (2005), S. 193–200

[223] White, S. A.: *Introduction to BPMN*. `http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf`. Version: 2004. – Abruf: 24.04.2010

[224] White, S. A.: *Using BPMN to Model a BPEL Process*. `http://www.bpmn.org/Documents/Mapping_BPMN_to_BPEL_Example.pdf`. Version: 2005. – Abruf: 27.08.2010

[225] Wittenburg, G. ; Terfloth, K. ; Villafuerte, F. ; Naumowicz, T. ; Ritter, H. ; Schiller, J. : Fence Monitoring – Experimental Evaluation of a Use Case for Wireless Sensor Networks. In: Langendoen, K. (Hrsg.) ; Voigt, T. (Hrsg.): *Wireless Sensor Networks* Bd. 4373. Berlin / Heidelberg : Springer, 2007, S. 163–178

[226] Workflow Management Coalition: *The Workflow Reference Model*. `http://www.wfmc.org/index.php?option=com_docman&task=doc_download&gid=92&Itemid=72`. Version: Januar 1995. – Abruf: 20.08.2010

[227] Workflow Management Coalition: *The Workflow Management Coalition Specification – Terminology & Glossary*. WWW. http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf. Version: 1999. – Abruf: 05.04.2010

[228] Workflow Management Coalition: *Process Definition Interface – XML Process Definition Language*. http://www.wfmc.org/index.php?option=com_docman&task=doc_download&Itemid=72&gid=132. Version: Oktober 2008. – Abruf: 14.10.2010

[229] World Wide Web Consortium (W3C): *W3C Recommendation: XML Path Language (XPath) Version 1.0*. http://www.w3.org/TR/xpath/. Version: November 1999. – Abruf: 09.10.2010

[230] World Wide Web Consortium (W3C): *W3C Recommendation: XSL Transformations (XSLT) Version 1.0*. http://www.w3.org/TR/xslt. Version: November 1999. – Abruf: 09.10.2010

[231] World Wide Web Consortium (W3C): *Note: Simple Object Access Protocol (SOAP) 1.1*. http://www.w3.org/TR/2000/NOTE-SOAP-20000508/. Version: Mai 2000. – Abruf: 07.09.2010

[232] World Wide Web Consortium (W3C): *Note: Web Services Description Language (WSDL) 1.1*. http://www.w3.org/TR/wsdl.html. Version: 2001. – Abruf: 03.09.2010

[233] World Wide Web Consortium (W3C): *Note: SOAP Version 1.2 Email Binding*. http://www.w3.org/TR/soap12-email. Version: Juli 2002. – Abruf: 03.09.2010

[234] World Wide Web Consortium (W3C): *From SOAP/1.1 to SOAP Version 1.2 in 9 points*. http://www.w3.org/2003/06/soap11-soap12.html. Version: Juni 2003. – Abruf: 07.09.2010

[235] World Wide Web Consortium (W3C): *Recommendation: XML Information Set (Second Edition)*. http://www.w3.org/TR/xml-infoset/. Version: Februar 2004. – Abruf: 07.09.2010

[236] World Wide Web Consortium (W3C): *W3C Working Draft: Web Services Choreography Description Language Version 1.0*. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/. Version: Dezember 2004. – Abruf: 15.10.2010

[237] World Wide Web Consortium (W3C): *W3C Working Draft: Web Services Choreography Requirements*. http://www.w3.org/TR/ws-chor-reqs/. Version: März 2004. – Abruf: 15.10.2010

[238] World Wide Web Consortium (W3C): *Web Services Architecture*. http://www.w3.org/TR/ws-arch/. Version: 2004

[239] World Wide Web Consortium (W3C): *XML Schema*. http://www.w3.org/XML/Schema. Version: Oktober 2004. – Abruf: 03.09.2010

[240] World Wide Web Consortium (W3C): *XML Schema Part 0: Primer Second Edition*. http://www.w3.org/TR/xmlschema-0/. Version: Oktober 2004. – Abruf: 03.09.2010

[241] World Wide Web Consortium (W3C): *XML Schema Part 1: Structures Second Edition*. http://www.w3.org/TR/xmlschema-1/. Version: Oktober 2004. – Abruf: 03.09.2010

[242] World Wide Web Consortium (W3C): *XML Schema Part 2: Datatypes Second Edition*. http://www.w3.org/TR/xmlschema-2/. Version: Oktober 2004. – Abruf: 03.09.2010

[243] World Wide Web Consortium (W3C): *Member Submission: Web Services Transfer (WS-Transfer)*. http://www.w3.org/Submission/WS-Transfer/. Version: September 2006. – Abruf: 11.09.2010

[244] World Wide Web Consortium (W3C): *Recommendation: Extensible Markup Language (XML) 1.1*. http://www.w3.org/TR/2006/REC-xml11-20060816/. Version: September 2006. – Abruf: 03.09.2010

[245] World Wide Web Consortium (W3C): *Recommendation: Web Services Addressing 1.0*. http://www.w3.org/2002/ws/addr/. Version: 2006. – Abruf: 23.09.2010

[246] World Wide Web Consortium (W3C): *W3C Candidate Recommendation: Web Services Addressing 1.0 – WSDL Binding*. http://www.w3.org/TR/ws-addr-wsdl/. Version: Mai 2006. – Abruf: 26.03.2011

[247] World Wide Web Consortium (W3C): *W3C Candidate Recommendation: Web Services Addressing 1.0 - WSDL Binding*. http://www.w3.org/TR/2006/CR-ws-addr-wsdl-20060529/. Version: Mai 2006. – Abruf: 09.04.2011

[248] World Wide Web Consortium (W3C): *W3C Member Submission: WSDL 1.1 Binding Extension for SOAP 1.2*. http://www.w3.org/Submission/wsdl11soap12/. Version: April 2006. – Abruf: 10.04.2011

[249] World Wide Web Consortium (W3C): *Recommendation: SOAP Version 1.2 Part 0: Primer*. http://www.w3.org/TR/2007/REC-soap12-part0-20070427/. Version: April 2007

[250] World Wide Web Consortium (W3C): *Recommendation: SOAP Version 1.2 Part 1: Messaging Framework*. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. Version: April 2007

[251] World Wide Web Consortium (W3C): *Recommendation: SOAP Version 1.2 Part 2: Adjuncts*. http://www.w3.org/TR/2007/REC-soap12-part2-20070427/. Version: April 2007

[252] World Wide Web Consortium (W3C): *Recommendation: Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. http://www.w3.org/TR/wsdl20-primer/. Version: Juni 2007. – Abruf: 09.09.2010

[253] World Wide Web Consortium (W3C): *Recommendation: Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. http://www.w3.org/TR/wsdl20/. Version: Juni 2007. – Abruf: 09.09.2010

[254] World Wide Web Consortium (W3C): *Recommendation: Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*. http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626/. Version: Juni 2007. – Abruf: 09.09.2010

[255] World Wide Web Consortium (W3C): *Recommendation: Web Services Policy 1.5 - Framework*. http://www.w3.org/TR/ws-policy/. Version: September 2007. – Abruf: 03.09.2010

[256] World Wide Web Consortium (W3C): *SOAP Specifications*. http://www.w3.org/TR/SOAP/. Version: April 2007. – Abruf: 03.09.2010

[257] World Wide Web Consortium (W3C): *W3C Working Draft: Web Services Addressing 1.0 – Metadata*. http://www.w3.org/TR/2007/WD-ws-addr-metadata-20070202/. Version: Februar 2007. – Abruf: 19.04.2011

[258] World Wide Web Consortium (W3C): *Candidate Recommendation: SOAP over Java Message Service 1.0*. http://www.w3.org/TR/2009/CR-soapjms-20090604/. Version: Juni 2009. – Abruf: 03.09.2010

[259] World Wide Web Consortium (W3C): *Recommendation: Namespaces in XML 1.0 (Third Edition)*. http://www.w3.org/TR/xml-names/. Version: Dezember 2009. – Abruf: 03.09.2010

[260] World Wide Web Consortium (W3C): *Working Draft: Web Services Metadata Exchange (WS-MetadataExchange)*. http://www.w3.org/TR/ws-metadata-exchange/. Version: August 2010. – Abruf: 11.09.2010

[261] World Wide Web Consortium (W3C): *Recommendation: Efficient XML Interchange (EXI) Format 1.0*. http://www.w3.org/XML/EXI/. Version: März 2011. – Abruf: 25.05.2011

[262] Yao, Y. ; Gehrke, J. : The cougar approach to in-network query processing in sensor networks. In: *SIGMOD Rec.* 31 (2002), Nr. 3, S. 9–18

[263] Yazar, D. ; Dunkels, A. : Efficient Application Integration in IP-based Sensor Networks. In: *Proceedings of ACM BuildSys 2009, the First ACM Workshop On Embedded Sensing Systems For Energy-Efficiency In Buildings*. Berkeley, CA, USA, 2009

[264] Yick, J. ; Mukherjee, B. ; Ghosal, D. : Wireless Sensor Network Survey. In: *Computer Networks* 52 (2008), Nr. 12, S. 2292–2330

[265] Yu, L. ; Wang, N. ; Meng, X. : Real-Time Forest Fire Detection with Wireless Sensor Networks. In: *Procceding of th International Conference on Wireless Communications, Networking and Mobile Computing*, 2005, S. 1214–1217

[266] Zaha, J. M. ; Barros, A. ; Dumas, M. ; ter Hofstede, A. : Let's Dance: A Language for Service Behavior Modeling. In: *The Fourteenth International Conference on Cooperative Information Systems (CoopIS)*. Montpellier, Frankreich, 2006

[267] Zhang, P. ; Sadler, C. M. ; Lyon, S. A. ; Martonosi, M. : Hardware Design Experiences in ZebraNet. In: *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 2004

[268] Zhou, X. L. ; Wu, M. : Service Discovery Protocol in Wireless Sensor Networks. In: *Second International Conference on Semantics, Knowledge and Grid*. Los Alamitos, CA, USA : IEEE Computer Society, 2006, S. 101

[269] Ziemann, J. ; Mendling, J. : EPC-Based Modelling of BPEL Processes – a Pragmatic Transformation Approach. In: *Proceedings of the 7th International Conference on Modern Information Technology in the Innovation Processes of the Industrial Enterprises*. Italien, September 2005

[270] ZigBee Alliance: *The ZigBee 1.0 Specification*. `http://www.zigbee.org/`. – Abruf: 26.10.2010

Eigene Publikationen

Journalbeiträge

1

Titel: A Comprehensive Approach to Integrating Sensor Networks and Enterprise IT

Autoren: Glombitza, N.; Pfisterer, D. & Fischer, S.

Publikation: International Journal of Next-Generation Computing, Juli 2010, Volume 1, S. 16-32

Kurzfassung: It is envisioned that in the future, all kinds of devices ranging from resource-constraint wireless sensor nodes to powerful server-class computers will interact to form an Internet of Things (IoT). In such a setting, tiny devices will extend the Internet to the physical world and allow for a completely new class of applications. However, until today, no widespread deployment of such IoT applications can be observed. Major challenges to master issues of embedded programming, massive distribution, resource constraints, heterogeneity, and seamless integration with traditional Internet technologies. Orchestrating such a number of different devices to form an application can be arbitrarily complex. In the Internet and especially in Enterprise IT, the concept of Service-Oriented Architectures (SOA) has been applied successfully to address this orchestration problem. However, the technologies used to realize SOAs (such as Web Services, HTTP, XML, or BPEL) are to heavyweight to be used in resource-constraint networks. In this paper, we present a comprehensive, Web Service-compliant approach to use SOA technologies in WSNs. Our approach comprises self-description of sensor nodes, a light-weight Web Service transport protocol (called Lean Transport Protocol, LTP), an approach for the model-based programming of sensor nodes, and a solution for the integration with the Internet. We present an exhaustive simulation showing the first-rate performance of the approach.

Konferenz- und Workshopbeiträge

2

Titel: Using BPEL to Realize Business Processes for an Internet of Things

Autoren: Glombitza, N.; Ebers, S.; Pfisterer, D. & Fischer, S.

Publikation: 10th International Conference on Ad Hoc Networks and Wireless, Juli 2011, S. 294-307

Kurzfassung: In the vision of an IoT, trillions of tiny devices extend the Internet to the physical world and enable novel applications that have not been possible before. Such applications emerge out of the interaction of these devices with each other and with more powerful server-class computers on the Internet. Programming such applications is challenging due to the massively distributed nature of these networks combined with the challenges of embedded programming. In addition, resource constraints, device heterogeneity, and the integration with the Internet further complicate this situation. In this paper, we present a programming-in-the-large approach for resource-constraint devices such as wireless sensor nodes. Our approach is to model such applications using the Business Process Execution Language (BPEL), which is successfully and widely used in the Internet to model complete applications and business processes. However, BPEL and its associated technologies are too resource-demanding to be directly applied in resource-constraint environments. We therefore use the BPEL model as input to a code generation process that generates custom-tailored, lean code for different target platforms. The resulting code is fully standard-compliant and allows a seamless integration of IoT devices in enterprise IT environments. We present an exhaustive evaluation on real hardware showing the first-rate performance of the approach.

3

Titel: LTP: An Efficient Web Service Transport Protocol for Resource Constrained Devices

Autoren: Glombitza, N.; Pfisterer, D. & Fischer, S.

Publikation: Seventh Annual IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks (IEEE SECON' 10), Juni 2010, S. 199-207

Kurzfassung: Wireless Sensor Networks (WSNs) are envisioned to become an integral part of the Future Internet. Together with countless other embedded appliances, such resource-constraint devices will form an Internet of Things (IoT) where all kinds of devices extend the Internet to the physical world. In this vision, the seamless and flexible integration of IoT devices ranging from simple sensor nodes to large scale Enterprise IT servers are the basis for novel applications and business processes not possible before. A major challenge is to master the arising challenges of scale, low resources, and heterogeneity. In the Internet and especially in Enterprise IT, heterogeneity is addressed using Service-Oriented Architectures (SOA). However, today's technologies used to realize SOAs are to heavyweight to be used in resource-constraint networks (RCNs). In this paper, we introduce a novel, versatile, and light-weight Web Service transport protocol (called Lean Transport Protocol, LTP) that allows the transparent exchange of Web Service messages between all kinds of resource-constrained devices and server or PC class systems. We describe LTP in detail and show by real-world measurements that LTP has the potential to serve as standard Web Service transport protocol in the Internet of Things.

4

Titel: Self-Description and Protocol Conversion for a Web of Things

Autoren: Glombitza, N.; Mietz, R.; Römer, K.; Fischer, S. & Pfisterer, D.

Publikation: Proceedings of 2010 IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC 2010), Juni 2010, S. 229-236

Kurzfassung: In the upcoming Internet of Things a plethora of mostly extremely resource constrained devices are integrated into the Internet which is extended to the physical world. The corresponding systems provide all kinds of services which are characterized by a highly decentralized organization. Thus, to flexibly use these services, a decentralized service discovery and self description is needed while under-running the devices' resource constrains. As further problem, since resource constrained devices are not capable of using classical communication protocols, a protocol conversion to integrate the Internet of Things into today's networks is required. In this paper, we propose an approach to efficiently realize standard compliant self description and dynamic discovery of Web Services using a schema based WSDL compression. Furthermore, we present an approach to convert between Web Service transport bindings which are allowed to use schema based SOAP message compression.

5

Titel: Using State Machines for a Model Driven Development of Web Service-Based Sensor Network Applications

Autoren: Glombitza, N.; Pfisterer, D. & Fischer, S.

Publikation: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering and ICSE Workshops, Mai 2010, S. 2-7

Kurzfassung: In the Internet of Things, all kinds of devices will extend the Internet to the physical world. In that vision, even extremely resource constrained sensor nodes can be triggered by as well as trigger business processes and are not limited to sense-and-send anymore. Despite the large potential, due to the time consuming, inflexible, and error prone development of sensor network applications, sensor networks are rarely integrated into today's enterprise IT. In this paper, we present an approach using state machines for a Model Driven Development of Web Service-based sensor network applications. We show how Web Services can be realized on sensor nodes and present a domain-specific language called State Machine for Resource Constrained Devices (SM4RCD) to orchestrate these services.

6

Titel: Integrating Wireless Sensor Networks into Web Service-Based Business Processes

Autoren: Glombitza, N.; Pfisterer, D. & Fischer, S.

Publikation: MidSens '09: Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks, ACM, Dezember 2009,

S. 25-30

Kurzfassung: Wireless Sensor Networks (WSNs) are envisioned to become an integral part of the Future Internet where they extend the Internet to the physical world. Yet, while Service-Oriented Architectures (SOA) are prospering in Enterprise-IT, wireless sensor networks have - despite contrary prognoses - not found their way into enterprises. A major obstacle is certainly the different and resource-constraint nature of this class of devices. We argue that approaches for the seamless integration with existing, widely deployed SOA technologies such as XML, Web Services, and the Business Process Execution Language (BPEL) are key to the success of WSNs in enterprises. In this paper, we present our approach to integrate WSNs into SOA environments using these technologies in a resource-efficient but fully standard-compliant way. We evaluate our implementation and present a case study.

7

Titel: On the Integration of Service-Oriented Architectures and Sensor Networks

Autoren: Glombitza, N.; Pfisterer, D. & Fischer, S.

Publikation: Proceedings of the 5th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS Workshop), Juni 2009

Kurzfassung: Currently, two major trends are gaining momentum: Service-Oriented Architectures (SOA) based on Internet standards are widely deployed in industry and Wireless Sensor Networks (WSNs) are becoming an integral part of the Future Internet. Combined, these two trends lay the groundwork for a new class of applications where all kinds of devices ranging from simple sensor nodes to large-scale application servers interact to drive business processes not possible before. That way, data stemming from a WSN monitoring assets in a port area influence the control flow of a business process in real-time or even trigger a business process. To achieve this level of integration, WSNs must seamlessly interoperate with existing widely deployed SOA technologies such as XML, Web Services, and the Business Process Execution Language (BPEL) to name only a few. However, due to their high ressource demand, these technologies are hardly applicable in resource-constrained WSN environments. In this paper, we present a survey of existing approaches and introduce our technique to seamlessly integrate WSNs into SOA environment using only very few resources.

8

Titel: Towards Service Orientation on Resource Constrained Devices

Autoren: Glombitza, N.; Buschmann, C.; Pfisterer, D.; Fischer, S. & Pahl, H.

Publikation: WEBIST 2009 – 5th International Conference on Web Information Systems and Technologies, März 2009, S. 72-77

Kurzfassung: For the flexible integration of enterprise applications and business processes, the Service Oriented Architecture (SOA) concept and the web service technology are the state of the art today. Especially for powerful hardware, a lot of web service and related technologies were developed during the last years. But with the development of Future Internet technologies, there is a demand for integrating all kinds of devices into a SOA. This includes especially devices with extremely limited resources, such as wireless sensor nodes, which are not capable of running today's web service technologies. In this paper we disclose the need for research action on different layers of the web service technology stack. We discuss promising solutions for running standard compliant web services in sensor networks and integrating sensor network and enterprise IT web services as well as BPEL business processes seamlessly. We introduce the L2D2 project in which our concept will be realized and proven.

9

Titel: Using Graphical Process Modeling for Realizing SOA Programming Paradigms in Sensor Networks

Autoren: Glombitza, N.; Lipphardt, M.; Werner, C. & Fischer, S.

Publikation: Proceedings of the 6th International Conference on Wireless On demand Network Systems and Services, Februar 2009, S. 61-70

Kurzfassung: Designing and modifying sensor network applications demand for IT expertise in the field of distributed systems. Programming paradigms used in application development for sensor networks like object orientation do not refer to the distributed nature of a sensor network application. This represses the usage of sensor network technology as part of industrial applications. In enterprises graphical tools like BPEL and BPMN are used to coordinate distributed processes and overcome the complexity of the interaction among different components. In this work we introduce the GWELS toolbox as graphical process modeling tool to realize the service oriented programming paradigm for sensor networks and ease the development and integration of sensor network applications. We exemplarily design and deploy a sensor network application with GWELS to demonstrate the applicability of our approach.

10

Titel: FRED – An Application for a Real-Life Large Scale Multihop Ad Hoc Network

Autoren: Glombitza, N.; Lipphardt, M.; Hellbrück, H. & Fischer, S.

Publikation: Proceedings of the 5th Annual IEEE/IFIP Conference on Wireless On demand Network Systems and Services (WONS'08), Januar 2008, S. 73-76

Kurzfassung: Simulations were standard in the Mobile Ad Hoc Network (MANET) research community in the past. New protocols and algorithms were validated via simulations by network simulators such as ns2 or GloMoSim. However, even with the increasing number of published simulation papers, contradictory results are decreasing the confidence in simulations today. First experimental results show that simulation models for Ad Hoc

networks are far from being realistic and protocols developed in simulations failed in real word environments. As a result, the demand for experimental results and application deployments within the research community increases. In this paper, we introduce FRED as a suitable application to conduct protocol evaluations as well as user acceptance surveys. FRED (Flexible Radio Enabled Dialog) is an enhanced TED-System that allows asynchronous surveys or quizzes with a multi hop radio network, where individual participants conduct the survey independent of others. FRED is based on the pacemate sensor network platform that provides a comfortable lightweight housing, a simple radio interface and an intuitive GUI that allows long term evaluations of user acceptance and network protocols.

Poster- und Demonstrationsbeiträge

11

Titel: Poster Abstract: Integrating Wireless Sensor Network and Enterprise IT Web Services

Autoren: Glombitza, N.; Mietz, R.; Römer, K.; Fischer, S. & Pfisterer, D.

Publikation: 7th European Conference on Wireless Sensor Networks, Februar 2010

Kurzfassung: In the emerging Internet of Things a plethora of mostly resource constrained devices (e.g., sensor nodes) offering different services are integrated into the Internet. Thus, to flexibly use these services, which are often characterized by ad-hoc communication links, decentralized service discovery and self-description is needed. Due to the existence of different protocols, a protocol conversion to integrate WSNs into today's (IP-based) networks is essential. We propose an approach to efficiently realize self-description and dynamic discovery of embedded Web Services using a schema-based WSDL compression. Furthermore, we present an approach to convert between standard Web Service transport bindings such as SOAP+HTTP and optimized binding for WSNs using schema-based SOAP message compression.

12

Titel: Evaluating Ad Hoc Networks with FRED[1]

Autoren: Glombitza, N.; Hellbrück, H. & Fischer, S.

Publikation: The 9th ACM International Symposium on Mobile Ad Hoc Networking and Computing (ACM Mobihoc 2008), Demo Session, Mai 2008

Kurzfassung: –

[1] Demonstrationsbeiträge dieser Konferenz durchliefen einen Auswahlprozess, wurden jedoch nicht in den Konferenzbänden veröffentlicht.

Publikationen als Coautor

13

Titel: A Service-Oriented Operating System and an Application Development Infrastructure for Distributed Embedded Systems

Autoren: Lipphardt, M.; Glombitza, N.; Neumann, J.; Werner, C. & Fischer, S.

Publikation: 17. Fachtagung "Kommunikation in Verteilten Systemen 2011", März 2011, S. 26-37

Kurzfassung: The paradigm of service-orientation promises a significant ease of use in creating and managing distributed software systems. A very important aspect here is that also application domain experts and stakeholders, who are not necessarily skilled in computer programming, get a chance to create, analyze, and adapt distributed applications. However, up to now, service-oriented architectures have been mainly discussed in the context of complex business applications. In this paper we will investigate how to transfer the benefits of a service-oriented architecture into the field of embedded systems, so that this technology gets accessible to a much wider range of users. As an example, we will demonstrate this scheme for sensor network applications. In order to address the problem of limited device resources we will introduce a minimal operating system for such devices. It organizes all pieces of code running on a sensor node in a service-oriented fashion and also features the relocation of code to a different node at runtime. We will demonstrate that it is possible to design a sensor network application from a set of already existing services in a highly modular way by employing already existing technologies and standards.

14

Titel: Demo Abstract: A Service-oriented Operating System and an Application Development Infrastructure for Wireless Sensor Networks

Autoren: Lipphardt, M.; Glombitza, N.; Neumann, J. & Werner, C.

Publikation: The 7th ACM Conference on Embedded Networked Sensor Systems (SenSys 2009), November 2009, S. 309-310

Kurzfassung: Due to the highly distributed nature and special basic conditions such as limited resources, implementing and maintaining a sensor network application is a tedious task. In this demonstration we present a service-oriented operating system for sensor nodes and a framework that allows a composition of services on the nodes. We show how sensor network applications can be composed and modified by using different services.