

Entwicklung situationsabhängig adaptierbarer Web-Service basierter Software

Veröffentlichung in: Ortner, E. (Hrsg.) Proceedings Symposium Entwicklung Web-Service basierter Anwendungen im Rahmen der 33. Jahrestagung der Gesellschaft für Informatik e.V., Frankfurt (Main), 2003. ISSN: 1432-3613. S. 1-22.

Inhalt:

Die Intelligenz hinter einer Anwendung und das Verständnis für die Intensionen der Anwender wird für betriebliche Software immer wichtiger. Für die Neu- und Weiterentwicklung verteilter Software werden Ansätze benötigt, die eine situationsabhängige Anpassung der Software an die speziellen Bedürfnisse der Anwender und ihrer Tätigkeiten unterstützen. In dieser Arbeit wird untersucht, wie Web-Service basierte Software adaptierbar gestaltet werden kann.

Leistungsbereich: Projektmanagement

Ansprechpartner: Jens Wehrmann

Dokumentart: Artikel

Kontakt

Safari GmbH Office Mannheim
Goethestraße 18 D-68161 Mannheim

Safari GmbH Office München
Reitmorstraße 4 D-80358 München

Tel: +49 - 621 - 18 144 720

Fax: +49 - 621 - 18 144 740

info@safari-gmbh.de
www.safari-gmbh.de

Dieses Werk ist urheberrechtlich geschützt. Alle Rechte, auch die der Übersetzung, des Nachdrucks und der Vervielfältigung vorbehalten. Kein Teil des Werkes darf ohne schriftliche Genehmigung der Safari GmbH in irgendeiner Form (Fotokopie, Mikrofilm, Datenträger oder einem anderen Verfahren) reproduziert oder unter Verwendung elektronischer Systeme verarbeitet, vervielfältigt oder verbreitet werden.

Entwicklung situationsabhängig adaptierbarer Web-Service basierter Software

Michael Amberg, Shota Okujava, Jens Wehrmann

Lehrstuhl für BWL, insbes. Wirtschaftsinformatik III
Friedrich-Alexander-Universität Erlangen-Nürnberg

Lange Gasse 20,

90403 Nürnberg

amberg@wiso.uni-erlangen.de

shota.okujava@wiso.uni-erlangen.de

jens.wehrmann@wiso.uni-erlangen.de

Abstract: Die Intelligenz hinter einer Anwendung und das Verständnis für die Intensionen der Anwender wird für betriebliche Software immer wichtiger. Für die Neu- und Weiterentwicklung verteilter Software werden Ansätze benötigt, die eine situationsabhängige Anpassung der Software an die speziellen Bedürfnisse der Anwender und ihrer Tätigkeiten unterstützen. In dieser Arbeit wird untersucht, wie Web-Service basierte Software adaptierbar gestaltet werden kann.

Zunächst wird der Gesamtkontext adaptierbarer Software für Web-Service basierte Software diskutiert, bevor anhand von Szenarien Nutzenspotenziale adaptierbarer Software aufgezeigt werden. Anschließend wird ein Komponenten-Framework vorgestellt, das die situationsabhängige Adaption Web-Service basierter Software unterstützt. Die wesentlichen Bestandteile des Komponenten-Frameworks sind die Grundkonzeption der situationsabhängigen Adaption und Protokollierung mit Web-Services, die zentralen Komponenten des Adaptionprozesses, sowie eine Gesamtarchitektur, die das Zusammenspiel der verwendeten Komponenten beschreibt. Abschließend wird auf die Realisierung adaptierbarer Software eingegangen und anhand der zuvor eingeführten Szenarien die Eignung der Frameworks beurteilt.

1. Motivation

Die Entwicklung der Informationstechnologien ist rasant. Die technischen Möglichkeiten sind kaum noch ein limitierender Faktor in der Softwareentwicklung. Für zukünftige Entwicklungen wird die Intelligenz hinter Anwendungen und das Verständnis für die Intensionen der Anwender ein immer bedeutenderer Wettbewerbsfaktor. Die Steuerung der Nutzungshäufigkeit und die Steigerung der Nutzungseffektivität einer Software stellen übergeordnete Ziele dar.

Eine Möglichkeit, derartig intelligente Software zu realisieren, besteht in der Analyse des Benutzungskontextes der Software und des Nutzungsverhaltens der Anwender sowie in der Speicherung bzw. Verwendung von Nutzungsinformationen [Or99]. Nutzungsinformationen werden typischerweise als Nutzungsprofile und Protokolldaten gespeichert. Diese können sehr detaillierte Informationen über das Verhalten der Anwender bei der Nutzung einer Software beinhalten. Mit Hilfe derartiger Informationen ist es möglich, Situationen zu erkennen und aus den erkannten Situationen Informationen abzuleiten, die einen Mehrwert sowohl für den Anwender als auch für die Entwickler und Administratoren einer Software generieren.

Die Verwendung von Nutzungsinformationen bei Internetanwendungen und mobilen Anwendungen kann bereits als gebräuchlich bezeichnet werden, jedoch fehlen unmittelbar anwendbare Standards sowie standardisierte Softwarekomponenten für Web-Service basierte Software. Derzeit wird erwartet, dass Web-Services eine besonders effiziente Form der Softwareentwicklung ermöglichen. Hierbei können Standardmodule flexibel kombiniert werden. Standardschnittstellen ermöglichen die Entwicklung von standardisierten und wiederverwendbaren Komponenten.

In dieser Arbeit soll ein Beitrag zur Beantwortung folgender Fragestellungen geleistet werden:

- Was ist unter situationsabhängig adaptierbarer Software zu verstehen?
- Welche Arten von situationsabhängiger Adaption lassen sich bei Web-Service basierter Software unterscheiden?
- Wie sieht ein Komponenten-Framework aus, welches die situationsabhängige Adaption von Web-Service basierter Software unterstützt?
- Wie lässt sich situationsabhängig adaptierbare Software mit dem Komponentenframework realisieren? Wie leistungsfähig ist das Framework?

Der hier vorgestellte Beitrag wurde im Umfeld eines durch die Bayerische Staatsregierung geförderten Projektes erarbeitet. Im Projektkontext entwickelt ein KMU-Unternehmen eine Controllingsoftware als betriebliche Standardsoftware mit Multi-Tier-Architektur. Um sich von Wettbewerbern abzuheben, wurde beabsichtigt, die situationsabhängige Protokollierung und Adaption unter Nutzung von Web-Services zu integrieren. Unter Zuhilfenahme von Standardkomponenten des Komponenten-Frameworks soll die Adaption möglichst flexibel realisiert werden können und in Bezug zur Kernsoftware weitgehend lose gekoppelt sein.

2. Grundlagen situationsabhängig adaptierbarer Web-Service basierter Software

Unter **adaptierbarer Software** wird die Software verstanden, die an ihre Umwelt angepasst werden kann. Finden diese Anpassungen automatisch statt, spricht man auch von intelligenter bzw. (selbst-) **adaptiver Software** [He98], [La00].

In diesem Beitrag wird von der Annahme ausgegangen, dass eine adaptierbare Software, die Informationen über die Nutzungssituation berücksichtigt, sehr viel besser zur Problemlösung beitragen kann als dies ohne Situationsinformationen möglich wäre. Der Anpassungsprozess der Software an die Nutzungssituation des Anwenders wird als **situationsabhängige Adaption** bezeichnet. Dabei werden Informationen verarbeitet, die auch für ergänzende Aufgaben und weitere Interessensgruppen hilfreich sind. Art und Umfang einer situationsabhängigen Protokollierung und Adaption sind maßgeblich davon abhängig, wie der Situationsbegriff systematisiert und konkretisiert wird.

Ursprünglich entstammt die **Diskussion über den Situationsbegriff** aus dem Umfeld der mobilen Dienste. In der Literatur finden sich unterschiedliche Ansätze, die den Situationsbegriff bearbeiten und jeweils eigenständig interpretieren (z.B. [Am02], [AW02], [GJ00], [Hi02] und [Sc02]). Bei [Am02] wird eine Einteilung in Ort, Zeit und Person vorgenommen. Bei [AW02] und [Sc02] wird die letztere Dimension in einen persönlichen und einen situationsabhängigen Teil untergliedert. [GJ00] nehmen eine weitere Unterteilung des situationsabhängigen Teils vor. Hitz et al. [Hi02] betrachtet in ihrer Arbeit explizit die Modellierung von ubiquitären Web-Anwendungen. Sie nehmen die Unterscheidung zwischen natürlichem (Ort, Zeit), technischem (Endgerät, Browser, Netzwerk, Status) und sozialem Kontext (Benutzerprofil und -verhalten) vor. Für die situationsabhängige Adaption wird diese Differenzierung zugrunde gelegt.

Betrachtet man den Gesamtanwendungskontext einer Software, können verschiedene Interessensgruppen unterschieden werden, die individuelle Zielsetzungen verfolgen. Der Gesamtanwendungskontext adaptierbarer Software ist in Abbildung 1 vereinfacht dargestellt (in Anlehnung an das Compass-Kooperationsmodell [Am02]). Externe oder interne Entwickler stellen eine betriebliche Software bereit und führen die Weiterentwicklung und Pflege der Software durch. Das (Unternehmens-) Management sowie Administratoren sind u. a. für die unternehmensspezifische Anpassung, die Systemeinführung und den Systembetrieb verantwortlich. Die Software wird im Allgemeinen von mehreren Mitarbeitern und sonstigen Anwendern (z.B. Kunden und Lieferanten) verwendet, wobei verschiedene Anwender in der Regel unterschiedliche Softwarefunktionalitäten nutzen. Als wesentliche Inter-

essensgruppen lassen sich demnach (End-) Anwender, Administratoren, Entwickler und das (Unternehmens-) Management unterscheiden.

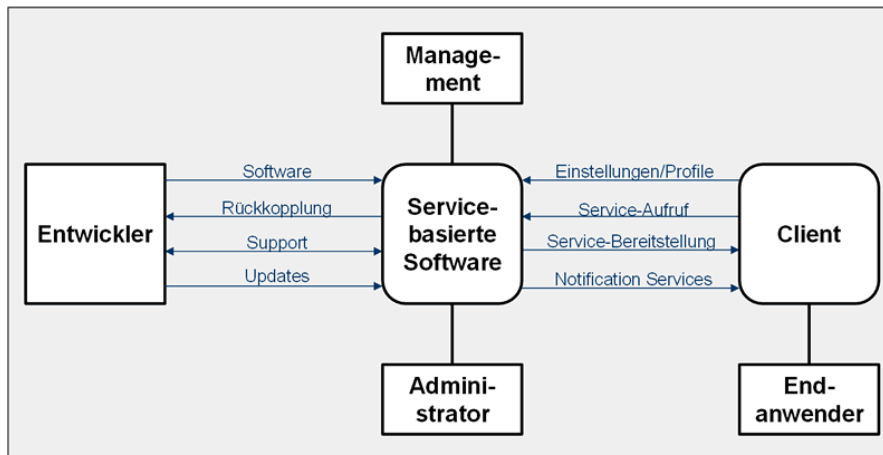


Abbildung 1: Gesamtanwendungskontext adaptierbarer Software

3. Anwendung situationsabhängig adaptierbarer Software

Die situationsabhängige Adaption kann in zwei Arten unterteilt werden:

1) Verbesserung der Bedienbarkeit: In der besseren Bedienung der Software wird ein direkter Mehrwert für den Endanwender einer Software gesehen. Es können drei Kategorien der Adaption unterschieden werden:

- Gestaltung adaptiver **Pull-Services**,
- Gestaltung adaptiver **Push-Services**,
- Gestaltung adaptiver **Benutzeroberfläche (GUI)**.

2) Verbesserung der Bereitstellung der Software: Die bessere Erbringung der Software stellt weniger einen direkten Mehrwert für den Endanwender einer Software, als für den Administrator, den Entwickler und das Management dar (siehe Tabelle 1). Es können drei weitere Kategorien der Adaption unterschieden werden:

- Nutzungsabhängige **Leistungsverteilung** und Performanzmanagement,
- Nutzungsabhängige **Softwareanalyse**, -wartung und -pflege,
- Nutzungsabhängige **interne Leistungsverrechnung** und ASP.

In Tabelle 1 ist der Zusammenhang zwischen den in Kapitel 2 vorgestellten Interessengruppen und den Kategorien dargestellt.

Kategorien Inter- essengruppen	Verbesserung der Bedienbarkeit der Software			Verbesserung der Bereitstellung der Software		
	Adaptive Pull-Services	Adaptive Push-Services	Adaptive Benutzeroberfläche (GUI)	Leistungsver- & Performance-management	Analyse, Wartung & Pflege	Interne Leistungsverrechnung & ASP
Endanwender	X	X	X	X		
Administrator				X	X	
Entwickler					X	
Management						X

Tabelle 1: *Interessensgruppen der Adaptionskategorien*

Im Folgenden soll jede Kategorie anhand von zwei Beispielen aus dem Kontext einer Controllingsoftware erläutert werden. Diese Beispiele werden im Kapitel 5 zur Diskussion über Realisierungsaspekte herangezogen.

3.1. Adaptive Pull-Services

Ein Serviceaufruf eines Anwenders wird interpretiert und das Ergebnis entsprechend der Nutzungsinformationen modifiziert. Der Anwender erhält eine auf seinen Nutzungskontext angepasste Softwarefunktionalität.

Szenario 1) Wertehistory: Bei einem Controllingbericht, der mehrfach von einem Endanwender aufgerufen wird, können die Veränderungen, die sich seit dem letzten Aufruf ergeben haben, farblich hervorgehoben dargestellt werden.

Szenario 2) Clientabhängige Funktion: Das Ergebnis eines Dienstaufwurfes kann unmittelbar von der Art des genutzten Endgerätes abhängen. Bei einem mobilen Endgerät wird der Controllingbericht so verdichtet, dass nur die wichtigsten Daten auf dem Display dargestellt werden.

3.2. Adaptive Push-Services

Aufgrund der Erkennung einer speziellen Situation wird ein Service von sich aus aktiv und sendet eine Nachricht an den Endanwender. Es können Hinweise auf anstehende Tätigkeiten, über geänderte Datenbestände oder Verbesserungsvor-

schläge für ein effizienteres Arbeiten gegeben werden. Diese Form von Services wird hier als Notification-Service bezeichnet.

Szenario 3) Workflow-Notification: Die Software kann einen Endanwender beim Eintreten eines Bearbeitungsereignisses durch Nachricht auf die grafische Oberfläche (GUI) des Client-Rechners aufmerksam machen. Ist der Endanwender nicht über das GUI erreichbar, bekommt er abhängig von der Dringlichkeit der Nachricht eine Email oder eine SMS.

Szenario 4) Wert-Notification: Ein Manager eines Unternehmens wird informiert, wenn sich für ihn kritische Unternehmenswerte negativ entwickeln.

3.3. Adaptive Benutzeroberfläche (GUI)

Die Ergebnisse eines Serviceaufrufes werden hinsichtlich der individuellen Bedürfnisse (individuelle Einstellungen, technische Möglichkeiten des Endgerätes) in der Darstellung angepasst.

Szenario 5) Personalisiertes GUI: Auf der Basis von Benutzerprofilen und des Benutzerverhaltens lassen sich die Menüs einer Software im Laufe der Benutzung an das individuelle Verhalten des Endanwenders anpassen.

Szenario 6) Dynamische Menüs: Die Menüs eines Endanwenders passen sich daran an, welche Funktionen häufig bzw. zuletzt verwendet wurden (vgl. Microsoft Produkte).

3.4. Nutzungsabhängige Leistungsverteilung und Performanzmanagement

Die Protokollierung des Nutzungsverhaltens ermöglicht die Messung und Analyse der nutzungsabhängigen Systemlast. Entsprechende Ergebnisse erlauben es der Software, nutzungsspezifisch (Zwischen-) Ergebnisse gegebenenfalls im Vorfeld einer Anfrage zu berechnen und zwischenspeichern. Dies kann Engpasssituationen vermeiden und helfen, das Systemverhalten zu verbessern. Als Interessensgruppe sind hier der Endanwender und Administrator des Systems zu sehen.

Szenario 7) Adaptive Caching: Häufig genutzte Daten können vom Server in die Cache gehalten werden.

Szenario 8) Vorberechnung: Rechenintensive Serviceaufrufe, die einem zeitlichen Muster folgen (z.B. Berichte werden immer montags aufgerufen), werden im Voraus berechnet.

3.5. Nutzungsabhängige Softwareanalyse, -wartung und -pflege

Die Nutzungsinformationen einer Software stellen für die Entwickler und Administratoren einer Software eine wertvolle Informationsquelle dar. So erlauben entsprechende Analysen (Welche Funktionen der Software werden wie und wie häufig genutzt?) eine Bewertung der Nützlichkeit von Softwarefunktionen und liefern Hinweise für zu ergänzende oder zu überarbeitende Funktionen. Eine Schnittstelle für den regelmäßigen Austausch von Nutzungsinformationen zwischen Entwickler und Nutzer schafft ein stetiges Verbesserungspotenzial, damit die Software permanent weiterentwickelt und ohne lange Versionszyklen verbessert werden kann.

Szenario 9) Prioritätsmanagement: Beispielsweise können mittels Nutzungsstatistiken der Softwarefunktionen festgestellt werden, welche Fehler vorrangig zu beheben bzw. welche Funktionen zu verbessern sind.

Szenario 10) Fehlermanagement: Tauchen Fehler in der Software auf, wird der Entwickler darüber informiert und erhält den spezifischen Anwendungskontext, in dem der Fehler aufgetreten ist, zur Verfügung gestellt.

3.6. Interne Leistungsverrechnung und ASP (Application Service Providing)

Die Protokollierung des Nutzungsverhaltens kann als Grundlage für eine Verrechnung der Software genutzt werden. Für eine interne Leistungsverrechnung kann festgestellt werden, welcher Mitarbeiter was, wann, wie oft und insbesondere in welcher Art und Weise einsetzt. Dies kann sowohl für die nutzungsabhängige Aufteilung auf innerbetriebliche Kostenstellen als auch für die Auswertung zu Controllingzwecken verwendet werden.

Szenario 11) Interne Leistungsverrechnung: Beispielsweise können die Entwicklungskosten einer Software nach dem Grad der Nutzung abgerechnet und auf die Kostenstellen verteilt werden.

Szenario 12) Nutzungsanalyse: Derartige Informationen helfen weiterhin, die Aktualität der Datenbestände zu sichern sowie den Bedarf für Schulungsmaßnahmen und Anreizsysteme zur Steigerung der Softwarenutzung zu identifizieren.

4. Ein Komponenten-Framework für die situationsabhängige Adaption

Im Allgemeinen stellt ein Komponentenanwendungsframework nach [Tu99] den Komponenten die Dienste zur Steuerung und Datenübertragung bereit. Im Folgenden wird ein Komponenten-Framework dargestellt, welches auf eine standardisierte Unterstützung der situationsabhängigen Adaption in Web-Service basierender Software ausgerichtet ist. Es wurde erstmals in [Am03] vorgestellt. Das Komponenten-Framework soll folgenden Designkriterien genügen:

- **Lose Kopplung zwischen Adaptionskomponenten und Kernsoftware:** Adaptionskomponenten und Kernsoftware sollen möglichst unabhängig voneinander sein.
- **Flexibilität hinsichtlich Adaption:** Inhalt und Umfang der situationsabhängigen Adaption sollen auch zur Laufzeit beliebig angepasst und verändert werden können.
- **Gewährleistung hoher Performanz:** Einbußen in der Performanz durch die Adaption sollen weitestgehend vermieden bzw. minimiert werden. Eine direkte Kommunikation sowie eine gegebenenfalls parallelisierte Bearbeitung werden bevorzugt.
- **Realisierung mit Standardkomponenten:** Der Adaptionsprozess soll weitgehend automatisiert und standardisiert werden. Der Aufwand für die softwareindividuelle Adaption ist weitgehend zu reduzieren.
- **Orientierung an offenen Standards:** Aufbauend auf der Entwicklungsplattform Java 2 Enterprise Edition sollen möglichst offene Standards (z.B. XML, XSLT) und Open-Source-Komponenten eingesetzt werden.

Im Folgenden werden wesentliche Bestandteile des Komponenten-Frameworks vorgestellt. Zunächst wird die Grundkonzeption der situationsabhängigen Adaption mit Web-Services aufgezeigt (Kapitel 4.1). Anschließend werden zentrale Komponenten des Adaptionsprozesses eingeführt (Kapitel 4.2). Die Erweiterungen werden in eine Gesamtarchitektur integriert und das Zusammenspiel der verwendeten Komponenten beschrieben (Kapitel 4.3).

4.1. Konzeption der situationsabhängigen Protokollierung und Adaption mit Web-Services

Um langfristige Aussagen über die Situation der Anwender treffen zu können, kann gemäß der Einteilung nach [Hi02] (vgl. Kapitel 2) der soziale Kontext betrachtet werden. Ziel ist die Erkennung von relevanten Verhaltensmustern, um die Benutzung der Software durch Adaption zu vereinfachen. Für diese Form der situationsabhängigen Adaption wird eine Protokollierung benötigt. Für kurzfristige Adaption (z.B. Anpassung auf den spezifischen Client) sind darüber hinaus der technische und der natürliche Kontext relevant.

Die Grundkonzeption der situationsabhängigen Adaption mit Web-Services zielt auf eine lose Kopplung zwischen Adaptions- und sonstiger Kernsoftware-funktionalität ab, um so eine nachträgliche Berücksichtigung bzw. Erweiterung der situationsabhängigen Protokollierung und Adaption zu unterstützen.

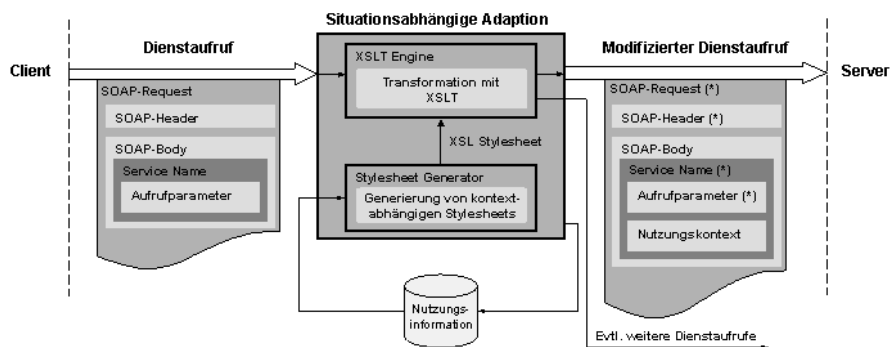


Abbildung 2: Grundkonzept der situationsabhängigen Protokollierung und Adaption mit Web-Services

Die Grundkonzeption ist in *Abbildung 2* dargestellt. Hierzu wurde der Aufruf eines Standard-Web-Services [Su03] um Aspekte der situationsabhängigen Adaption erweitert. Im Folgenden wird davon ausgegangen, dass eine Client-Komponente bei einem Serviceaufruf keine Kenntnisse über Art und Umfang der Adaption hat bzw. erhält. Die aufgerufene Server-Komponente benötigt derartige Informationen lediglich, wenn eine inhaltliche Anpassung der Softwarefunktionalität erfolgen soll. Jede andere Form der Adaption erfolgt durch die Analyse, Interpretation und gegebenenfalls Modifikation der Kommunikationsnachrichten durch eine Standardkomponente für die situationsabhängige Adaption (SDA-Komponente). Die SDA-Komponente besteht im Wesentlichen aus einer XSLT-Engine für die Transformation von XML-Nachrichten mit Stylesheets und einem Stylesheet-Generator, der die Stylesheets abhängig vom Situationskontext generiert.

Der grundsätzliche Ablauf lässt sich wie folgt beschreiben: Ein Serviceaufruf (SOAP-Request) wird von der SDA-Komponente entgegengenommen, mit Hilfe einer Standard-XML-Transformation modifiziert und gegebenenfalls um Informationen zum relevanten Nutzungskontext ergänzt. Im modifizierten Serviceaufruf können sowohl der Header (z.B. Ergänzung von Metadaten für den Adaptionprozess), der Servicename (z.B. Nutzung eines anderen Services) als auch die Aufrufparameter (z.B. Erweiterung des Aufrufs um den Nutzungskontext) verändert werden. Das für den Transformationsprozess notwendige Stylesheet wird unter Nutzung eines Stylesheet-Generators gegebenenfalls dynamisch aus gespeicherten Nutzungsinformationen generiert. Der Transformationsprozess wird zudem verwendet, um Informationen über das Nutzungsverhalten abzuleiten bzw. weitere Serviceaufrufe für weiterführende bzw. ergänzende Adaptionen zu generieren.

Die gleichen Adaptionsschritte können auch auf Ebene der Aufrufergebnisse (SOAP-Response) durchgeführt werden. Die Ergebnisse eines Aufrufs werden entsprechend des Kontextes und der Situation modifiziert bzw. erweitert. Der beschriebene Adaptionprozess kann neben der Transformation genutzt werden, um das Nutzungsverhalten zu protokollieren oder sonstige Aktionen zu initiieren. So ist beispielsweise beim Eintreten eines vorgegebenen Nutzungsereignisses eine Benachrichtigung von Endanwendern über die Nutzeroberfläche, per Email oder SMS möglich.

4.2. Komponenten des situationsabhängigen Adaptionprozesses

Für den situationsabhängigen Adaptionprozess lassen sich grob folgende Komponentengruppen unterscheiden:

- **Komponenten der Kernsoftware:** Umfassen die Komponenten der Kernsoftware unterteilt in Client-Komponenten und Server-Komponenten inklusive Web-Service-Interfaces.
- **Softwareindividuelle Adaptionskomponenten:** Umfassen die Komponenten, die eine situationsabhängige Adaption softwareindividuell realisieren. Sie können in serverseitige sowie clientseitige Adaptionskomponenten gegliedert werden.
- **Standard-Adaptionskomponenten:** Umfassen die Standardkomponenten zur Adaption, die unmittelbar auf den situationsabhängigen Adaptionprozess ausgerichtet sind und unabhängig von einer konkreten Software eingesetzt werden können.

Im Folgenden wird insbesondere auf die dritte Komponentengruppe fokussiert und das Zusammenspiel der Komponenten aus dem Blickwinkel der Standardkomponenten betrachtet. Die zentralen Designcharakteristika für das Zusammenspiel der benötigten Komponenten sind:

- **Mehrstufiger Adaptionprozess:** Für die umfassende Unterstützung der situationsabhängigen Adaption eines Serviceaufrufes wird ein mehrstufiger Bearbeitungsprozess zugrundegelegt. Die Bearbeitungsschritte können beispielsweise nach der Zielsetzung der Adaption oder dem Bearbeitungsaufwand differenziert werden. Die Bearbeitung erfolgt zunächst auf dem Client (z.B. für die Ergänzung des Nutzungskontextes und für eine clientseitige Adaption), anschließend durch Standardkomponenten, die allgemeine Adaptionfunktionalität bereitstellen (z.B. für eine serviceunabhängige Adaption) und abschließend auf dem Server (z.B. für eine serverseitige Adaption).
- **Performanz und Lastverteilung:** Um der Entstehung von Systemengpässen vorzubeugen, sollen die Anzahl der Bearbeitungskomponenten sowie die Ablauffolge variabel gehandhabt werden. Abhängig vom konkreten Serviceaufruf und allgemeinen Adaptionseinstellungen sollen lediglich die benötigten Bearbeitungskomponenten angesprochen werden (variable Kommunikationsrouten). Wenn möglich, werden voneinander unabhängige Bearbeitungskomponenten parallel aufgerufen (autonome Bearbeitung). Informationen (z.B. Protokollierungsinformationen), die für den Echtzeitbetrieb nicht zwingend erforderlich sind, werden asynchron mit Messages übertragen. Hierzu werden Komponenten für die Administration benötigt.

Das resultierende Komponentendesign ist in *Abbildung 3* dargestellt. Es basiert auf den Standards Java 2 Enterprise Edition (J2EE) und Web-Services [Su00] und erweitert die Architektur für Web-Service basierte Software um Komponenten für die situationsabhängige Adaption. Eine Web-Service basierte Software ist in der Regel als Multi-Tier-Architektur realisiert, die im Allgemeinen vier grundlegende Komponenten unterscheidet: (Thin-) Client, Presentation-Server, Application-Server und Datenbank-Server. Für die situationsabhängige Protokollierung und Adaption sind insbesondere die in der Abbildung dargestellten Komponenten von Interesse, die um einen weiteren Server für die Adaption ergänzt werden.

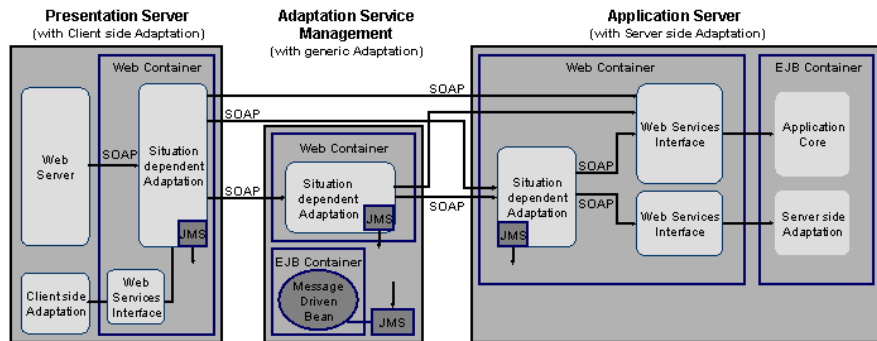


Abbildung 3: Um Komponenten für die situationsabhängige Adaption erweiterte Multi-Tier-Architektur

Der **Präsentations-(Web-)Server** soll verschiedenartigen Clients die Möglichkeit bieten, Web-Services eines Applikations-Servers nutzen zu können. Dabei soll der technische Kontext (gemäß [Hi02] vgl. Kapitel 2) verwendet werden, um die Darstellung an die verschiedenartigen Clients (Webbrowser, Thin- und Rich-Client) zu adaptieren. Der Präsentations-Server ist als Webserver konzipiert und um eine Clientseitige Adaptionkomponente (CSA) ergänzt. Der Web-Server nimmt die Anfrage eines Client entgegen, wandelt diese in einen Web-Service-Aufruf (SOAP-Request) um und leitet den Aufruf an eine situationsabhängige Adaptionkomponente weiter. Diese Adaptionkomponente erweitert den Web-Service-Aufruf um spezifische Informationen über den Nutzungskontext und legt die weitere Bearbeitungsfolge fest. Ein Web-Service eines Applikations-Servers kann sowohl direkt, als auch indirekt über einen speziellen Adaption-Server (Adaptation Service Management) aufgerufen werden. Ein Anwendungsbeispiel für eine clientseitige Adaption ist die Hervorhebung der Werte eines Berichtes, die sich seit der letzten Anzeige des Berichtes verändert haben.

Der **Applikations-Server** stellt die funktionalen Dienste der Software als Web-Services zentral zur Verfügung. Er besteht aus einem Web-Container und einem Enterprise Java Beans (EJB)-Container. Der Web Container beinhaltet eine Komponente zur situationsabhängigen Adaption sowie Web-Service-Schnittstellen zum EJB-Container. Dieser beinhaltet die Kernfunktionalität der Software und wird um eine Serverseitige Adaptionkomponente (SSA) erweitert. Ein Anwendungsbeispiel hierfür ist die Vorverdichtung von zeit- und ressourcenintensiven Berechnungen, die mittels einer Mustererkennung zeitlich vorgelagert ausgeführt werden können.

Für die situationsabhängige Adaption wird weiterhin ein spezieller **Adaptions-Server** (Adaptation Service Management) benötigt. Dieser besteht aus einem Web-Container für situationsabhängige Adaptionen und einem EJB-Container. Ersterer ist für rechenintensive oder serviceunabhängige Adaptionen hilfreich. Als Anwendungsbeispiel kann das Modifizieren eines Nutzerprofils genannt werden. Letzterer dient der Bearbeitung nicht zeitkritischer Adaptionen, z.B. dem Protokollieren der Web-Service-Aufrufe für die interne Leistungsverrechnung. Hierzu nimmt eine Java Messaging Service (JMS)-Komponente entsprechende Anfragen aller Komponenten als Messages entgegen. Message-Driven-Beans im EJB-Container führen die entsprechenden Adaptionen durch.

4.3. Gesamtarchitektur des Komponenten-Frameworks

Die Gesamtarchitektur des Komponenten-Frameworks geht insbesondere auf die für das Management der Adaptionen benötigten Komponenten ein und zeigt das Zusammenspiel mit weiteren administrativen Komponenten verteilter Software auf. Es werden die folgenden Designkriterien zugrundegelegt:

- **Standardisiertes Kommunikations- und Informationsmanagement:** Die notwendigen Informationen zur Durchführung eines Adaptionprozesses sind in der Regel in den entsprechenden Komponenten gespeichert. Lediglich bei der ersten Durchführung und bei Veränderung der Adaption ist eine Synchronisation der Komponenten notwendig. Über einen Synchronisationsbus werden die entsprechenden Komponenten mit den benötigten Informationen einheitlich versorgt.
- **Erweiterbarkeit auf mehrere Softwarekomponenten:** Die Architektur soll eine einheitliche Plattform darstellen, um die situationsabhängige Adaption bei mehreren Komponenten gegebenenfalls unterschiedlicher Softwaresysteme zu unterstützen. Der Entwicklungsaufwand für die Unterstützung neuer Softwarekomponenten soll reduziert werden.

In der Gesamtarchitektur werden Komponenten zum Management des Gesamtsystems ergänzt (vgl. Abbildung 4). Neutrale Komponenten der Administration in verteilter Software sind Komponenten für die Benutzerverwaltung (User-Management), zum Verwalten von Services (Service Directory) sowie zur Beschreibung der Services (Meta Data Repository). Diese werden auch für die situationsabhängige Adaption benötigt. Für das Adaptionmanagement (Adaptation Service Management) werden mehrere Komponenten benötigt. Diese werden in administrative und operationale Komponenten unterteilt.

Als **administrative Komponenten** werden hier die Komponenten für das übergeordnete Adaptionsmanagement (Adaptation Management), für das Synchronisationsmanagement der SDA-Komponenten (Synchronization Management) und zur Verwaltung der Transformationsvorschriften (Transformation Service Management) bezeichnet. Die wesentlichen Interaktionsbeziehungen sind in *Abbildung 4* dargestellt. Die gestrichelten Kanten zeigen die Synchronisation im Gesamtsystem. Die Synchronisations-Management-Komponente aktualisiert die Daten (z.B. Profile) und synchronisiert andere Komponenten.

Als **operationale Komponenten** werden im Wesentlichen Komponenten zur Protokollierung von Transaktionsdaten, Benutzerverhalten, Antwortzeiten und sonstigen Nutzungsinformationen (Logging) sowie zur Mustererkennung in Nutzungsinformationen und für die automatische Profilvergenerierung (Profiling) benötigt. Die generierten Nutzungssituationen und Profile werden von den administrativen Komponenten verwendet. Weiterhin werden generische Adaptionen hinzugezählt.

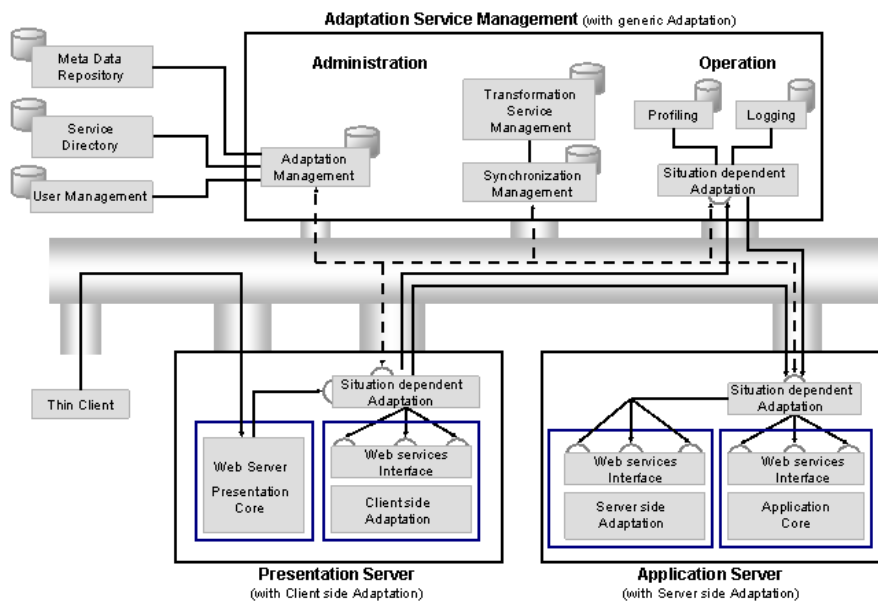


Abbildung 4: Gesamtarchitektur des Komponenten-Frameworks

Die Vorteile dieser Architektur liegen neben der losen Kopplung zur Kernsoftware insbesondere in der Performanz durch Reduktion des Verwaltungsaufwandes sowie in der Flexibilität hinsichtlich der Adaptionsprozesse. Durch die Komponentenorientierung wird das Gesamtsystem skalierbar und leichter erweiterbar. Die Aufteilung in Präsentations-Server und Applikations-Server bietet die nötige Flexibilität und die leichte Erweiterbarkeit für unterschiedliche Clients. So genannte Thin-Clients (Web Browser, PDA-Browser,...) können über den Präsentations-Server flexibel eingebunden werden.

5. Realisierung situationsabhängig adaptierbarer Software

Im Folgenden soll aufgezeigt werden, inwieweit die in Kapitel 3 beschriebenen Anwendungsszenarien mit dem in Kapitel 4 vorgestellten Framework realisiert werden können. Zu diesem Zweck werden die für den Bearbeitungsprozess innerhalb des Komponenten-Frameworks erforderlichen Komponenten kurz beschrieben. Danach wird betrachtet, inwieweit die in Kapitel 4 eingeführten Designkriterien für die Realisierung der Szenarien berücksichtigt werden konnten. Abschließend steht eine Betrachtung des Nutzens.

Damit eine Adaption der Software leicht und schnell durchgeführt werden kann, werden generische Komponenten benötigt, die durch einfache Parametrisierung flexibel gesteuert werden. Im beschriebenen Framework wird diese Flexibilität dadurch erreicht, dass für jede generische Komponente die Konfigurationsdaten in XML-Format definiert werden. Zudem werden Datenfluss und Transformationen der Datenstrukturen durch XSL-Stylesheets festgelegt. Um das Verhalten der Komponenten zu ändern, müssen lediglich die XML- und XSL-Dokumente verändert werden. Zur Laufzeit werden die Daten aktualisiert und die Komponenten mit neuen Parametern initialisiert.

Abbildung 5 zeigt die Bearbeitungsprozesse des Komponenten-Frameworks. Das Framework benötigt für die Ankopplung an die Kernsoftware eine Proxy-Komponente (ADFWProxy), die die Web-Service-Requests entgegennimmt und an die CallProcessor-Komponente weiterleitet. Der CallProcessor ist die Hauptkomponente, die anhand der Call-ID (z.B. Web-Service-URL) entscheiden kann, was mit dem Aufruf passieren soll und an welche anderen Komponenten die Daten weitergeleitet werden sollen. Das Call-Processor-Config-Dokument beinhaltet die dazu notwendigen Einstellungen. Dieses Dokument hält fest, welche Aufrufe wie verarbeitet werden sollen und welche Aufrufinformationen für die aktuelle Bearbeitung benötigt werden. Die so vorbereiteten Aufrufe werden unter Verwendung von XSL-Stylesheets transformiert. Danach wird die so erzeugte XML-Struktur an die zugehörige Engine weitergeleitet.

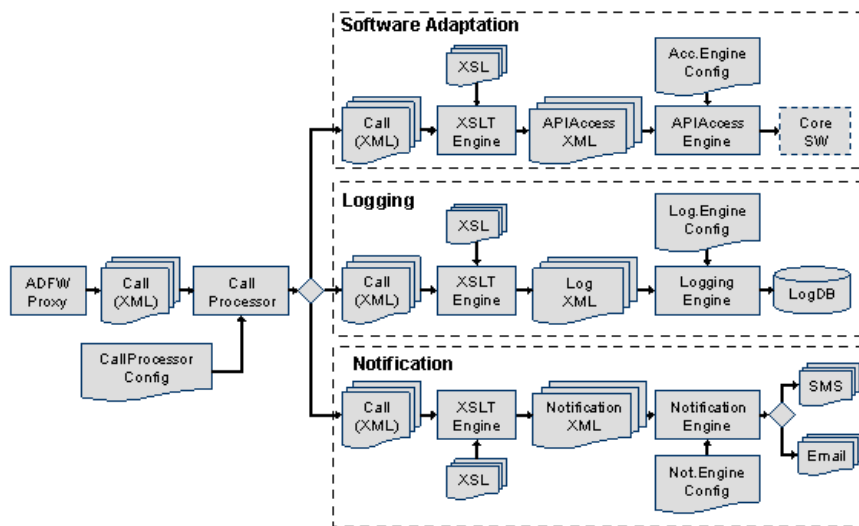


Abbildung 5: Bearbeitungsprozesse des Komponenten-Frameworks

In Abbildung 5 sind die drei wesentlichen Bearbeitungsprozesse des Komponenten-Frameworks dargestellt:

- **Software Adaptation:** Die API-Access-Engine unterstützt die individuelle Adaption von Kernsoftwarefunktionen. Dafür stellt die Kernsoftware in der Regel eine eigene API bereit. Diese Engine impliziert die engste Kopplung zwischen Adaptionsframework und Kernsoftware. Für die Adaption ist die Auswertung der Protokolldaten und Erkennung der auffälligen Muster notwendig.
- **Logging:** Die Logging-Engine loggt die (Protokollierungs-) Daten für spätere Auswertungen. Die notwendigen Einstellungen (Welche Informationen müssen wie gespeichert werden?) werden in Logging-Engine-Config-Dokument spezifiziert und können auch zur Laufzeit modifiziert werden. Damit ist eine flexible Speicherung von Protokolldaten möglich.
- **Notification:** Für die asynchrone Kommunikation mit dem Endanwender wird die Notification-Engine eingesetzt. Die Daten für die Notification-Engine werden durch XSL-Transformation generiert. Je nach Anforderung können statische oder dynamisch zur Laufzeit (unter Einbeziehung von Protokolldaten) generierte Stylesheets verwendet werden. Die Stylesheets beinhalten auch Information über die zu nutzenden Kommunikationskanäle (SMS, Email, GUI). Die Einstellungen der einzelnen Kanäle (Server, Absender, Accountinformationen) finden sich im Notification-Engine-Config-Dokument.

Die zentralen Komponenten für die dargestellten Abläufe stellen die oben als Engines bezeichneten Komponenten dar, die je nach Ausrichtung Daten protokollieren, Notifications erzeugen oder die Rückkopplung zur Kernsoftware ermöglichen. Für diese Komponenten sind feste XML-Strukturen definiert. Diese Strukturen werden erzeugt, in dem ein Aufruf (Call) abhängig von der Aufrufart und unter Verwendung von speziellen XSL-Stylesheets transformiert wird. Die Abläufe können parallel zueinander oder auch sequenziell durchgeführt werden.

In *Tabelle 2* ist dargestellt, welche der zuvor beschriebenen Komponenten für die Realisierung der Szenarien benötigt wurden. Alle Szenarien waren realisierbar. Die Logging-Komponente wird in den meisten Szenarien benötigt. Die Notification-Komponente wird nur für Realisierung der ausgewählten Szenarien benötigt. Die Komponente *Software Adaptation* passt die Kernsoftware entsprechend zu den Nutzungsinformationen des Endanwenders an.

Szenario \ verwendete Komponenten	1) Werthehistory	2) Clientabh. Funktion	3) Workflow Notification	4) Wert-Notification	5) Personalisiertes GUI	6) Dynamische Menüs	7) Adaptives Caching	8) Vorberechnung	9) Prioritätsmanagement	10) Fehlermanagement	11) Interne Leistungsverr.	12) Nutzungsanalyse
Software Adaptation		X			X	X	X	X				
Logging	X		X	X	X	X	X	X	X	X	X	X
Notification			X	X					X	X		

Tabelle 2: Verwendung der Komponenten zur Realisierung der Szenarien

Der Nutzen des hier vorgestellten Ansatzes ist gegeben, wenn die Realisierung der Adaptionprozesse mithilfe des Komponenten-Frameworks deutlich effizienter und flexibler als bei einer „konventionellen“ Realisierung erfolgen kann. Anhand der in Kapitel 4 eingeführten Designkriterien sollen nun Aussagen über die Leistungsfähigkeit des Adaptionframeworks getroffen werden.

- **Lose Kopplung zwischen Adaptionskomponenten und Kernsoftware:** Die Komponenten der Adaption wurden als Standardkomponenten realisiert, die unabhängig von der Kernsoftware sind, solange nicht die Kernsoftwarefunktionen selbst adaptiert werden. Die Komponenten *Notification* und *Logging* sind von der Kernsoftware lose gekoppelt. Für die Verwendung des Adaptionsframeworks mit einer anderen Software sind nur geringfügige Modifikationen an den verwendeten Schnittstellen nötig. Die Komponente *Software Adaptation* hingegen greift direkt auf die Funktionen der Kernsoftware zu. Die Autorisierungsroutinen benötigen eine enge Kopplung.
- **Flexibilität:** Die Anforderung hoher Flexibilität wurde in dem Framework dadurch erreicht, dass der Datenfluss zwischen einzelnen Adaptionskomponenten durch den Austausch von XML-Dokumenten realisiert wurde. Die Struktur und Inhalt der XML-Dokumente können zur Laufzeit mittels XSL-Transformationen frei geändert werden. Die benötigten XSL-Stylesheets werden bei Bedarf automatisch zur Laufzeit, unter Berücksichtigung der Nutzungsinformationen des Endanwenders, generiert.
- **Performanz:** Um die zusätzliche Belastung der Kernsoftware durch die Adaptionprozesse zu vermeiden, wurden diese in der Regel parallel zu den Prozessen der Kernsoftware aufgerufen. Durch die Parallelisierung der Prozesse blieb die von den Komponenten der Adaption verursachte zusätzliche Belastung bei allen Szenarien deutlich unter 1%. Bei der Realisierung der Szenarien 1 (Wertehistory) und 2 (Clientabhängige Funktion) war erforderlich, die Adaptionprozesse nach den Prozessen der Kernsoftware zu starten (sequentielle Bearbeitung). In den beiden sequentiell realisierten Szenarien ergab sich eine erkennbare Zusatzlast (bis zu 15%).
- **Realisierung mit Standardkomponenten:** Zur Realisierung der vorgestellten Szenarien reichten die drei Standard-Adaptionskomponenten *Software Adaptation*, *Logging* und *Notification* aus. Bei Adaptionarten, die enger in die Kernsoftware eingreifen, als in den hier gewählten Szenarien, sind manuelle Anpassungen in der Kernsoftware erforderlich.
- **Offene Standards:** Das Adaptionsframework und alle vorgestellten Szenarien konnten mit offenen Standards (HTML, Web-Services, WML, XML, XPATH, XSL, XSLT) und Open-Source-Komponenten (Apache Xerces und Xalan, MySQL DBMS) realisiert werden.

Abschließend soll eine Betrachtung des Nutzens erfolgen: Im Rahmen des Projektes hat sich herausgestellt, dass in den Szenarien 1 (Wertehistory), 2 (Clientabhängige Funktion), 4 (Wert-Notification) und 10 (Fehlermanagement) auch über die Sicht der jeweiligen Interessensgruppen hinaus, ein überdurchschnittlicher Nutzen gesehen wird. Im Szenario 6 (dynamische Menüs) wurde kein bzw. geringer Nutzen erkannt. Dies ist darauf zurückzuführen, dass die Adaptionen bereits in Microsoft-Produkten verwendet wird und von vielen Endanwendern nicht als Mehrwert eingestuft wird. In den anderen Szenarien wurde ein Nutzen nur von den in Tabelle 1 zugeordneten Interessensgruppen erkannt.

6. Zusammenfassung und Ausblick

Im Rahmen dieser Arbeit wurde zunächst der Gesamtkontext der situationsabhängigen Adaption der Software beschrieben und wesentliche Anwendungsszenarien aufgezeigt. Der Kern der Arbeit beschreibt den systematischen Aufbau eines Komponenten-Frameworks für die situationsabhängige Protokollierung und Adaption Web-Service basierter Software. Die Grundkonzeption zielt auf eine lose Kopplung zwischen Kernsoftware- und Adaptionen funktionalität ab, um so eine nachträgliche Berücksichtigung bzw. Erweiterung der situationsabhängigen Adaption zu unterstützen. Für einen gegebenenfalls mehrstufigen Adaptionsprozess sind spezielle Komponenten erforderlich. Diese werden in die Architektur des Gesamtsystems integriert. Sie stellt einen, gegebenenfalls auf mehrere Anwendungssysteme erweiterbaren, standardisierten Rahmen für adaptierbare Web-Service basierte Software bereit.

Bei der Evaluierung anhand von Beispielszenarien zeigte sich, dass die zugrundegelegten Designkriterien *lose Kopplung, hohe Flexibilität, hohe Performanz, Realisierung mit Standardkomponenten* und *Verwendung offener Standards* weitgehend erreicht werden konnten. Über Web-Service basierte Software hinaus, ist das vorgestellte Framework auch auf Client-Server-basierter Software übertragbar. Hierfür sind spezielle Adapter zu realisieren, die Client-Server-Aufrufe in Web-Service-Aufrufe transformieren. Im Projekt zeigte sich, dass bei der Realisierung entsprechender Adapter die Performanz des Gesamtsystems nur geringfügig beeinträchtigt wurde.

Untersuchungsaspekte, die in dieser Arbeit nicht weiter betrachtet wurden, sind: Inwieweit sind die beschriebenen Komponenten für den Einsatz in kommerziellen Produkten geeignet? Können weitere Formen der Adaptivität durch das Framework unterstützt werden? Wie ist mit personenbezogenen Daten umzugehen?

7. Literatur

- [Am02] Amberg, M.; Figge, S.; Wehrmann, J. (2002): Compass – Ein Kooperationsmodell für situationsabhängige mobile Dienste: in Hampe, J. F.; Schwabe, G. (Hrsg.), Proceedings zur Teilkonferenz Mobile and Collaborative Business der Multikonferenz Wirtschaftsinformatik (MKWI 2002), Nürnberg.
- [AW02] Amberg, M.; Wehrmann, J. (2002): A Framework for the Classification of Situation Dependent Services: Eighth Americas Conference on Information Systems Proceedings (AMCIS 2002), pp. 1838-1843, Dallas, USA.
- [Am03] Amberg, M.; Okujava, S.; Wehrmann, J. (2003): Ein Komponenten--Framework für die situationsabhängige Adaption Web-Service basierter Standardsoftware. In: Turowski, K. (Hrsg.): Tagungsband des fünften Workshop für Komponentenorientierte betriebliche Anwendungssysteme (WKBA5), Augsburg.
- [Ba00] Badrinath, B.; Fox, A.; Kleinrock, L.; Popek, G. (2000): A Conceptual Framework for Network and Client Adaptation. ACM MONET Journal, Vol. 5, No. 4, pp. 221-231.
- [Bl00] Blair, G.; Blair, L.; Issarny, V.; Tuma, P.; Zarras, A. (2000): The Role of Software Architecture in Containing Adaptation in Component-based Middleware Platforms. Proceedings of Middleware 2000, IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing, Hudson River Valley (NY), USA.
- [GJ01] Gessler, S.; Jesse, K. (2001): Advanced Location Modeling to enable sophisticated LBS Provisioning in 3G networks. In: Beigl, M.; Gray, P.; Salber, D. (Hrsg.): Proceedings of the Workshop on Location Modeling for Ubiquitous Computing. Atlanta, USA.
- [He98] Heinemann, G. T. (1998): Adaptation and Software Architecture. In: 3rd Annual International Workshop on Software Architecture (ISAW-3), Seiten: 61-64, November 1998. Orlando, Florida.
- [Hi02] Hitz, M.; Kappel, G.; Retschitzegger, W.; Schwinger, W. (2002): Ein UML-basiertes Framework zur Modellierung ubiquitärer Web-Anwendungen. In Wirtschaftsinformatik 44 (3/2002), S.225-235, Wiesbaden.
- [Ke02] Ketfi, A.; Belkhatir, N.; Cunin, P.-Y. (2002): Automatic Adaptation of Component-based Software. PDPTA'02, Las Vegas, USA.
- [La00] Laddaga, R. (2000): Active Software. In First International Workshop on Self-Adaptive Software (IWSAS2000), USA.

- [Or99] Oreizy, P.; Gorlick, M.M.; Taylor, R.N.; Heimbigner, D.; Johnson, G.; Medvidovic, N.; Quilici, A.; Rosenblum, D.S.; Wolf, A.L. (1999): An Architecture-Based Approach to Self-Adaptive Software in: IEEE Intelligent Systems, May/June 1999 (Vol. 14, No. 3), pp 54-62.
- [Or96] Orfali, R.; Harkey, D.; Edwards, J. (1996): The Essential Distributed Objects Survival Guide. John Wiley & Sons, New York, USA.
- [Sc02] Scheer, A. W.; Feld, T.; Göbl, M.; Hoffmann, M. (2002): Das mobile Unternehmen. In: Silberer, G.; Wohlfahrt, J.; Wilhelm, T. (Hrsg.) Mobile Commerce – Grundlagen, Geschäftsmodelle, Erfolgsfaktoren, Gabler Verlag, Wiesbaden.
- [Su00] Sun Microsystems (2000): http://java.sun.com/j2ee/sdk_1.2.1/techdocs/guides/ejb/html/Overview3.html.
- [Su03] Sun Microsystems (2003): <http://developer.java.sun.com/developer/technicalArticles/WebServices/WSPack2/techno.html#xrpcmsg>.
- [Tu99] Turowski, K. (1999): Ordnungsrahmen für komponentenbasierte betriebliche Anwendungssysteme. In: Turowski, K. (Hrsg.): Tagungsband des 1. Workshops Komponentenorientierte betriebliche Anwendungssysteme (WKBA 1), S. 3-14, Magdeburg.
- [Wa00] Wang, X.; Schulzrinne, H. (2000): An Integrated Resource Negotiation, Pricing, and QoS Adaptation Framework for Multimedia Applications. IEEE Journal on Selected Areas in Communications (JSAC), Vol. 18, No. 12, pp. 2514-2529.