Prozessunterstützung für eGovernment-Services mit Java und XML am Beispiel von www.hamburg.de

Timmy Blank

Diplomarbeit

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Softwaretechnik

Juni 2001

Erstbetreuer:
Dr. Ralf Klischewski

Zweitbetreuer:
Prof. Dr. Arno Rolf

Erklärung:

Ich versichere hiermit, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

Börnsen, den 18.06.2001

Timmy Blank
Auf der Haide 11
21039 Börnsen
Matr. Nr.: 4628041
Tel.: (040) 720 39 58
EMail: 3blank@informatik.uni-hamburg.de

Betreuung:

Dr. Ralf Klischewski (Erstbetreuung)

Prof. Dr. Arno Rolf (Zweitbetreuung)

Dr. Ralf Klischewski
Arbeitsbereich Softwaretechnik (SWT)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22175 Hamburg

Prof. Dr. Arno Rolf
Arbeitsbereich Angewandte und Sozialorientierte Informatik (ASI)
Fachbereich Informatik
Universität Hamburg
Vogt-Kölln-Straße 30
22175 Hamburg

Inhaltsverzeichnis

1 Einführung

1.1 Einleitung und Motivation

Das WWW (World Wide Web) hat sich in den letzten Jahren zu einer Plattform entwickelt, die von vielen Unternehmen dazu genutzt wird, mit ihren Geschäftspartnern zu kommunizieren (Extranet), die betriebsinterne Kooperation zu realisieren (Intranet) und Produkte zu verkaufen (E-Commerce im Internet). Zunehmend gewinnt das WWW auch für den öffentlichen Sektor an Bedeutung und wird u.a. als zusätzlicher Kanal betrachtet, dem Bürger *öffentliche Dienstleistungen* (*Public Services*) zugänglich zu machen (vgl. [Sch00]).

In der Wirtschaft ist im Hinblick auf die Steigerung von Kundenorientierung allgemein der Trend zu beobachten, das Leistungsangebot an der vom Kunden in seiner als *Kundenprozess* bezeichneten komplexen Problemsituation auszurichten und gebündelt möglichst über eine Kontaktstelle anzubieten (vgl. [ÖstFleAlt00]). Dieser Ansatz spielt auch in der öffentlichen Verwaltung zunehmend eine Rolle (vgl. z.B. [Mei01]). Verwaltungsdienstleistungen sollen bedarfsgerecht und in einer hohen Qualität erbracht werden. Hierbei werden zwei „Qualitätsstufen“ unterschieden: zum einen die Art und Weise, wie man Verwaltungsleistungen erlangt und zum anderen, in welchen Abläufen die Dienstleistung an sich produziert wird. (vgl. [Sau99])

Zur Qualitätssteigerung des Zugangs zu Verwaltungsdienstleistungen wird dem Konzept *Self-Service* eine entscheidende Bedeutung beigemessen. Ziel ist es, dem Bürger über *Bürgerprozessportale*[1] im WWW genau die Leistungen zugänglich zu machen, die er zur Erledigung seines Anliegens benötigt und die er selbstständig, rund um die Uhr und von beliebigen Standorten aus abrufen kann.

Nicht jede Dienstleistung kann jedoch vollständig als Self-Service realisiert werden. Dieser Anteil erfordert immer noch einen persönlichen Kontakt zwischen Leistungsnehmer und Leistungsanbieter, wenn z.B. spezielle Fähigkeiten oder Kenntnisse gefordert sind. Zur Qualitätssteigerung des Prozesses der Dienstleistungserstellung an sich bedarf es somit ebenfalls einer angemessenen Softwareunterstützung mit dahinterstehenden Prinzipien, die den Besonderheiten des Einsatzgebietes gerecht wird und die (öffentlichen) Dienstleistungserbringer in ihrem kundenorientierten Handeln unterstützt. Der Ansatz *Serviceflow Management (SfM)* [KliWet00] hat zum Ziel, sich genau diesen Anforderungen zu stellen.

SfM fokussiert hierbei v.a. auf die Unterstützung komplexer Dienstleistungen, die sich aus einer Reihe aufeinander aufbauender Teilleistungen zusammensetzen (z.B. Reiseplanung oder Wohnungswechsel), die evtl. organisationsübergreifend erbracht werden. Solche Dienstleistungsprozesse werden als Folgen von sogenannten *Servicepoints* verstanden, an denen sich der Leistungsnehmer mit seinem Anliegen dem Leistungserbringer direkt (z.B. im Front-Office eines Reisebüros) oder mediiert durch Informations- und Kommunikationstechnologie (z.B. im Back-Office) präsentiert und Teilleistungen erbracht werden.

Eine Softwareunterstützung an Servicepoints muss der Anforderung gerecht werden, den jeweiligen Dienstleistungserbringer in die Lage zu versetzen, kundenorientiert, d.h. flexibel zu handeln und gleichzeitig am Dienstleistungsprozess zu partizipieren. Zusätzlich sind in übergreifenden Kontexten meist heterogene Systemlandschaften zu überwinden, d.h. die zur Zusammenarbeit notwendigen Informationen sind möglichst technologieneutral zwischen den einzelnen Leistungserbringern auszutauschen. Zur Lösung dieses Problems wird der

[1] Zum Begriff des Prozessportals siehe [Öst00].

Datenbeschreibungssprache *eXtensible Markup Language (XML)* zur Repräsentation der auszutauschenden Daten eine besondere Bedeutung beigemessen. Meist wird die Programmiersprache Java zur Entwicklung von Anwendungen verwendet, die den Umgang mit diesen Informationen implementieren. Als Gründe können z.B. das gemeinsame Herkunftsfeld (WWW), ähnliche Eigenschaften (z.B. Portabilität: „Java + XML = Portable Code + Portable Data“ [McL00]) und die mittlerweile erreichte Bedeutung auch im Unternehmensumfeld angeführt werden.

Die Qualitätssteigerung von Verwaltungsdienstleistungen kann also dadurch erreicht werden, dass eine Prozessunterstützung, die sowohl Self-Service über Bürgerprozessportale als auch die evtl. organisationsübergreifende Leistungserstellung auf Produzentenseite unterstützt, angeboten wird. Hierbei ist im Hinblick auf die Umsetzung auf Technologieneutralität und Portabilität zu achten, wodurch sich die Verwendung von XML und Java empfiehlt.

1.1 Zielsetzung und Vorgehen

Aus dem dargestellten Anwendungskontext ergibt sich folgende in dieser Arbeit zu beantwortende Problemstellung:

Wie sollte eine Softwarearchitektur für webbasierte Servicepoints gestaltet sein, auf deren Basis eine Prozessunterstützung für eGovernment-Services unter Verwendung der Technologien Java und XML realisiert werden kann?

Zur Beantwortung der Fragestellung wird zunächst grundlegend geklärt, was unter den Begriffen Service bzw. Dienstleistung[2], eGovernment und Prozessunterstützung in dieser Arbeit verstanden wird.

Da in dieser Arbeit SfM das grundlegende Leitbild[3] zur Entwicklung einer Prozessunterstützung darstellt, werden anschließend die sich aus diesem Ansatz ergebenen Anforderungen an eine Prozessunterstützung abgeleitet, die an jedem Servicepoint anzubieten ist.

Self-Service über Bürgerprozessportale stellt ein wesentliches im eGovernment-Bereich verfolgtes Konzept dar, um die Leistungserstellung stärker an den Bedürfnissen der Bürger auszurichten zu können (vgl. oben). Werden Verwaltungsprozesse durch den Einsatz von Software nach dem SfM-Ansatz unterstützt, so bietet es sich an, den Ansatz auch dazu zu verwenden, diese direkt zum Bürger zu „verlängern“. Hierzu ist das Konzept Self-Service über Bürgerprozessportale in den SfM-Ansatz einzuordnen. Es werden zunächst die Prinzipien herausgearbeitet, die Self-Service über Bürgerprozessportale zugrunde liegen. Anhand eines Beispielportals (Bremer-Online-Service) und der Merkmale, die einen Servicepoint im SfM-Ansatz auszeichnen, wird die Einordnung vorgenommen.

Zur Beantwortung der zentralen Fragestellung der Arbeit, wird in einem weiteren Schritt ein funktionaler Prototyp für einen webbasierten Servicepoint vorgestellt. Dieser wird im Hinblick auf die Erfüllung der ermittelten Anforderungen sowie der Einhaltung allgemeiner softwaretechnischer Entwurfsprinzipien evaluiert, um so eine allgemeine Empfehlung zur Ausgestaltung einer Softwarearchitektur für webbasierte Servicepoints sowohl auf grob- als auch feinstruktureller Ebene geben zu können. Der Prototyp ist im Rahmen einer Lehrveranstaltung am Arbeitsbereich Softwaretechnik der Universität Hamburg im Wintersemester 2000/2001 entstanden. Er diente zur teilweisen Umsetzung eines Anwendungsbeispiels für einen eGovernment-Service, namentlich der „elektronische Antrag

[2] Die Begriffe Service und Dienstleistung werden in dieser Arbeit synonym verwendet.

[3] Zum Begriff des Leitbildes siehe z.B. [Rol98].

auf Briefwahlunterlagen über www.hamburg.de“, das auch in dieser Arbeit Verwendung findet.

Neben der Einhaltung allgemeiner softwaretechnischer Entwurfsprinzipien mussten bei der Entwicklung des webbasierten Servicepoints auch die Besonderheiten beachtet werden, die sich aus der Verwendung des WWW als Basistechnologie für eine Softwarearchitektur ergeben. Diese werden herausgearbeitet. Anhand einer auf die Verwendung im Webumfeld ausgerichteten Entwurfsmetapher namens Fachlicher Service (siehe [LipWolZül01]), die den Entwurf der Grobstruktur des funktionalen Prototypen angeleitet hat, wird ihre Ausgestaltung als vierschichtige Softwarearchitektur begründet.

Als weitere Ausgangslage zur Entwicklung des Prototypen dienten die im SfM-Ansatz verwendeten Modellierungstechniken und Entwurfsmetaphern[4] für softwaretechnische Prozessrepräsentationen. Sie werden vorgestellt und anschließend Umgangsformen abgeleitet, die notwendig sind, um dem Benutzer eines Servicepoints einen adäquaten Umgang mit den Prozessrepräsentationen anbieten zu können. Da die beiden Technologien Java und XML zur Realisierung der fachlichen Funktionalität des Prototypen eingesetzt worden sind, wird dargestellt, welche Möglichkeiten grundlegend durch ihre Kombination angeboten werden, um die Prozessrepräsentationen und ihre Umgangsformen technisch abzubilden.

Zur Realisierung der webbasierten Benutzungsoberfläche des Prototypen wurde das Produkt StoryServer 5.0 der Firma Vignette eingesetzt. Dessen Zusammenspiel mit den anderen verwendeten Technologien wird beispielhaft für weitere mögliche Produkte, die zur Präsentation von Servicepoints im Webumfeld eingesetzt werden könnten, untersucht.

1.2 Überblick über die Arbeit

Kapitel 2, *Prozessunterstützung für eGovernment-Services im Rahmen von Bürgerprozessportalen*, behandelt zunächst grundlegend den für diese Arbeit relevanten Begriff der (öffentlichen) Dienstleistung und stellt das Anwendungsfeld eGovernment vor.

Kapitel 2.3 beschäftigt sich hiernach zunächst mit dem Prozessbegriff und stellt dar, was in dieser Arbeit unter Prozessunterstützung verstanden wird. Anschließend wird auf den SfM-Ansatz eingegangen und seine wesentlichen Merkmale herausgestellt. Das Kapitel schließt mit der Erstellung eines Anforderungskatalogs, den Serviceflow Management Systeme erfüllen sollten, um im Rahmen arbeitsteiliger Prozesse an Servicepoints eine adäquate Prozessunterstützung anbieten zu können.

In Abschnitt 2.4 wird dann auf den Self-Service-Ansatz eingegangen und seine wesentlichen Merkmale herausgearbeitet. Hiernach wird das Bürgerprozessportal-Konzept zunächst allgemein und dann anhand eines Beispielportals (Bremer-Online-Service) als mögliche Ausprägung eines webbasierten Self-Service-Zugangssystem zu öffentlichen Verwaltungsprozessen vorgestellt. Zum Abschluss wird der Ansatz Self-Service über Prozessportale in den SfM-Ansatz eingeordnet.

Kapitel 2 schließt mit der Vorstellung des Anwendungsbeispiels „elektronische Beantragung von Briefwahlunterlagen über www.hamburg.de“.

Kapitel 3, *Prozessunterstützung mit Java und XML*, gibt zunächst einen Überblick über den Entstehungskontext des in dieser Arbeit verwendeten funktionalen Prototypen.

Anschließend wird in Kapitel 3.2 auf die Besonderheiten von Webanwendungen und die Entwurfsmetapher Fachlicher Service eingegangen. Zusätzlich werden allgemeine softwaretechnische Entwurfsprinzipien herausgearbeitet, die grundsätzlich bei der Ausgestaltung von Softwarearchitekturen beachtet werden müssen.

---

[4] Zum Begriff Entwurfsmetapher siehe z.B. [Zül98, S. 78].

Hiernach beschäftigt sich dieses Kapitel in Abschnitt 3.3 unter Bezugnahme auf das Anwendungsbeispiel damit, wie Prozesse im SfM-Ansatz modelliert werden und stellt die verwendeten Entwurfsmetaphern sowie die zu unterstützenden Umgangsformen an den Prozessrepräsentationen vor.

Anschließend wird in Kapitel 3.4 und 3.5 auf die zur Realisierung des funktionalen Prototypen eingesetzten Technologien XML, Java und StoryServer 5.0 eingegangen. Zunächst werden die der XML zugrundeliegenden Basiskonzepte und Möglichkeiten der Handhabung von XML-Dokumenten aufgezeigt, die für die Programmiersprache Java angeboten werden. Es folgt eine kurze Vorstellung des StoryServers und des Zusammenspiels der verwendeten Technologien.

Danach wird der funktionale Prototyp in Kapitel 3.6 präsentiert und in Kapitel 3.7 im Hinblick auf die Erfüllung des Anforderungskatalogs sowie der allgemeinen softwaretechnischen Entwurfsprinzipien evaluiert. Zusätzlich wird geprüft, ob die Besonderheiten webbasierter Anwendungen ausreichend berücksichtigt wurden. Auf Basis der hierbei erzielten Ergebnisse schließt das Kapitel mit einer Empfehlung, wie eine Softwarearchitektur für webbasierten Servicepoints gestaltet werden sollte.

Kapitel 4, *Zusammenfassung und Ausblick*, fasst die Arbeit zusammen und benennt nicht beantwortete Fragestellungen, die als Ausgangspunkt für weitere Untersuchungen im behandelten Themengebiet dienen können.

1.3 Danksagung

Ich möchte mich an dieser Stelle bei allen Personen bedanken, die mir bei der Erstellung dieser Arbeit behilflich waren.

Mein Dank geht zunächst an Herrn Dr. Ralf Klischewski für die Erstbetreuung, der mich während des Entstehungsprozesses dieser Arbeit jederzeit mit Rat und Tat unterstützt hat. Bei Herrn Prof. Dr. Arno Rolf bedanke ich mich für die Zweitbetreuung.

Des weiteren möchte ich mich bei den Teilnehmern des Projektseminars „Service als Leitbild: Softwareunterstützung für Dienstleister“ bedanken, in dessen Rahmen der in dieser Arbeit verwendete funktionale Prototyp entstanden ist und das Anwendungsbeispiel erarbeitet wurde. Insbesondere Nol Shala danke ich für die Zusammenarbeit bei der Implementation des Prototypen.

Wolf-Gideon Bleek schulde ich Dank für seine Unterstützung bei der Installation des StoryServers.

Bei meiner Schwester bedanke ich mich für das Korrekturlesen dieser Arbeit.

2 Prozessunterstützung für eGovernment-Services im Rahmen von Bürgerprozessportalen

In diesem Kapitel wird zunächst der Begriff der Dienstleistung aus verschiedenen Perspektiven diskutiert, um darzustellen wie dieser Begriff in der vorliegenden Arbeit verstanden wird.

Hiernach wird auf das Anwendungsfeld eGovernment eingegangen und motiviert, dass im Hinblick auf die informationstechnische Unterstützung von eGovernment-Services sowohl der Aspekt der Selbstbedienung, z.B. über Bürgerprozessportale, als auch der manuellen Leistungserstellung auf Anbieterseite berücksichtigt werden muss.

Als Leitbild[5] zur Entwicklung von Software für Prozessunterstützung wird dieser Arbeit der SfM-Ansatz zugrundegelegt. Dessen grundlegende Konzepte werden nach Klärung des Begriffs Prozessunterstützung vorgestellt. Anschließend wird ein Anforderungskatalog erstellt, den ein Softwaresystem erfüllen sollte, um eine adäquate Prozessunterstützung an einem Servicepoint anbieten zu können.

Hiernach wird der Ansatz Self-Service über Bürgerprozessportale eingeführt und in den SfM-Ansatz eingeordnet. Zum Schluss dieses Kapitels wird das in dieser Arbeit verwendete Anwendungsbeispiel vorgestellt.

2.1 Dienstleistungen

Der Begriff der *Dienstleistung* ist in der Literatur bisher nicht eindeutig definiert worden (vgl. [Bod99, S. 1], [Sau99, S. 12]). Laut Bodendorf [Bod99, S. 1] scheint dieses Vorhaben auch schlechthin nicht möglich. Er arbeitet statt dessen spezifische und in der betriebswirtschaftlichen Literatur weitgehend anerkannte Merkmale von Dienstleistungen heraus, die im folgenden dargestellt werden.

2.1.1 Merkmale von Dienstleistungen

Die beiden wichtigsten und von Bodendorf als Basismerkmale bezeichneten Eigenschaften von Dienstleistungen zur Abgrenzung gegenüber Sachleistungen sind ihr *immaterieller Charakter* und die Notwendigkeit der *Integration des externen Faktors* in die Leistungsproduktion (vgl. Abbildung 1). Obwohl heutzutage eine scharfe Trennung der Sparten Sach- und Dienstleistungsproduktion aufgrund derer zunehmender Verschmelzung[6] eher schwer fällt, überwiegt bei *Dienstleistungsprodukten*[7] die Immaterialität der Leistung (vgl. auch [DIN98, S. 16]). Der Blick wird auf den nutzenstiftenden Charakter der Leistung gerichtet. Ist dieser vorwiegend immateriell, dann spricht man von Dienstleistungen. Dadurch wird es auch besonders schwer, die Güte einer Dienstleistung zu beurteilen, da eine physische

[5] Unter einem Leitbild wird die grundsätzliche Sichtweise verstanden, anhand derer ein Ausschnitt der Realität wahrgenommen, verstanden und gestaltet wird (vgl. [Gry96, S. 99]). Zum Begriff des Leitbildes siehe z.B. auch [Rol98], [Zül98].

[6] Dienstleistungen werden oft mit Sachgütern gekoppelt oder materielle Objekte durch Dienstleistungen aufgewertet. Bsp.: Verkauf von Problemlösungen anstelle einzelner Produkte oder Überlassung eines Sachgutes auf Zeit (Leasing). [Bod99, S. 2]

[7] Der Begriff des Dienstleistungsproduktes weist auf eine ergebnisorientierte Betrachtungsweise von Dienstleistungen in Form von Endprodukten (mit vorwiegend immateriellen Charakter) hin. Daneben kann man eine Dienstleistung auch als Aktivität oder Folge von Aktivitäten betrachten, die ihren zeitlichen Verlauf und die Kombination von notwendigen Handlungen in den Vordergrund stellt (vgl. [Bod99, S. 6]). Dienstleistungen können also als Produkt, d.h. als Ergebnis von Tätigkeiten oder als diese Tätigkeiten selbst verstanden werden (vgl. [Ert86, S. 17]).

Messung nicht möglich ist. Nur der Kunde selber kann die Qualität einer Serviceleistung beurteilen (vgl. auch [Kli00]).

Dies leitet zu dem zweiten Basismerkmal von Dienstleistungen, nämlich die Integration des Kunden oder eines in seinem Besitz befindlichen Objektes (als mögliche Ausprägungen des externen Faktors) in die Leistungsproduktion (z.B. Finanzberatung in einer Bank oder KFZ-Reparatur). Dadurch wird der Kunde in die Lage versetzt, in einem gewissen Maße Einfluss auf die Leistungserstellung auszuüben, was zu einer Auftragsindividualität führen kann[8], die sich von der Sachgüterproduktion dadurch abhebt, dass dort die Einflussmöglichkeit meist nur am Anfang der Produktion gegeben ist (vgl. auch [KliWet00]).

Als weiterer Aspekt von Dienstleistungen wird die Gleichzeitigkeit der Erstellung und Verwertung angeführt, was nach [Bod99, S. 3] als *Uno-Actu-Prinzip* bezeichnet wird. Gemeint ist hiermit, dass Erbringung und Konsum der Leistung zeitlich als auch räumlich gekoppelt sind, die Erstellung der Leistung praktisch die Dienstleistung selbst ausmacht (vgl. auch [DIN98, S. 16]). Dies trifft heutzutage allerdings nur noch auf den Teil der Dienstleistungen zu (z.B. Haarschnitt, ärztliche Behandlung), bei denen die körperliche Anwesenheit des Kunden erforderlich ist. Besonders Dienstleistungen, die auf die Vermittlung von Informationen aufbauen, können aufgrund der informationstechnischen Errungenschaften auch asynchron erbracht werden (vgl. auch Abschnitt 2.4).

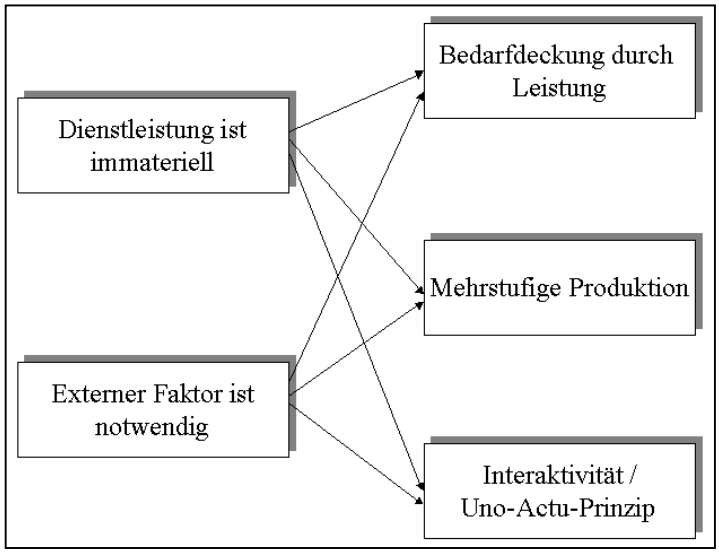

Abbildung 1: Merkmale von Dienstleistungen nach [Bod99, S. 2]

Das Merkmal der mehrstufigen Leistungsproduktion bezieht sich darauf, dass vor Absatz der Dienstleistung zunächst ein Angebot, die sogenannte Vorkombination bzw. Leistungsbereitschaft des Anbieters, produziert werden muss. Nach Absatz der Dienstleistung findet erneut ein Produktionsprozess, die sogenannte Endkombination, am externen Faktor statt. Für materielle Güter gilt i.d.R. dagegen, dass der Produktionsprozess nur bis zum Absatz läuft.

Als letztes abgeleitetes Merkmal wird von Bodendorf angeführt, dass die Bedarfsdeckung durch Leistung erfolgt. Hierdurch wird nochmals betont, dass der Anbieter seine Leistung direkt in Verbindung mit dem externen Faktor erbringt, dieser also sofort einen Nutzen daraus zieht. Der Nutzen kann bei einem Sachgut erst nach Erwerb durch einen entsprechenden Gebrauch realisiert werden.

[8] „Dienstleistungen sind keineswegs gleichzusetzen mit individuellen Leistungen. Wer etwa ins Kino geht, erlebt keineswegs eine individuelle Leistung.“ [Ber86]

2.1.2 Dienstleistungen als Beziehungen

Betrachtet man Dienstleistungen aus der sozialwissenschaftlichen Perspektive, so können diese als soziale Beziehungen zwischen Anbieter und Nachfrager aufgefasst werden (vgl. [Gut95], [Kli00]). Produzent und Konsument lassen sich auf diese Verbindung ein, da der Kunde ein Bedürfnis hat, welches der Dienstleister zu befriedigen verspricht. Art und Umfang der Serviceleistung wird meist vor Leistungserbringung in Form von (mündlichen oder schriftlichen) Verträgen festgehalten. Leistungsnachfrager und –anbieter gehen also ein Angebot/Nachfrage-Verhältnis ein, das durch einen Service-Vertrag beschrieben wird, der die Rechte und Pflichten der Vertragspartner festlegt.

Oft spielen soziale Einflüsse eine Rolle, wenn es um die Etablierung einer Servicebeziehung geht. So hängt die Wahl des konkreten Dienstleisters z.B. davon ab, ob es schon im Vorwege zu Kontakten kam oder welchen Ruf der Serviceanbieter (Öffentlichkeit/Freunde) hat. Der Anbieter einer Leistung versucht diese so weit wie möglich an die tatsächlichen Bedürfnisse des Kunden anzupassen, damit die Beurteilung seitens des Kunden positiv ausfällt, mit dem Ziel einer langfristigen Kundenbindung[9]. Da sich Bedürfnisse auch während der Leistungserstellung (z.B. während eines Urlaubs) ändern können, werden Dienstleistungen (oft) auch bei der konkreten Erbringung an den jeweiligen Kunden angepasst, d.h. von der eigentlichen Vereinbarung wird abgewichen (vgl. [Kli00]).

Dienstleistungen als Beziehungen aufzufassen resultiert somit daraus, dass zwischen Servicenehmer und Serviceanbieter über eine längere Zeit und verschiedene Dienstleistungen hinweg eine Bekanntschaft existiert. Dies gilt auch innerhalb komplexer Dienstleistungen, die sich aus verschiedenen aufeinander aufbauenden Teilleistungen zusammensetzen und über einen längeren Zeitraum ablaufen. Versucht der Dienstleister die eigene Leistung aufgrund der mit dem Kunden gemachten Erfahrungen und dessen besonderen Wünschen, die sich aus der jeweiligen Situation ergeben, kontinuierlich auf Basis der Leistungsvereinbarung anzupassen, so spricht man von Dienstleistungen als Beziehungen. (vgl. [KliWetBah01], [Gut95])

Steht das Erkennen und Befriedigen von Bedürfnissen unter Beachtung der konkreten Situation des Kunden und des bisherigen Dienstleistungsverlaufs nicht im Vordergrund, sondern wird nur eine Standarddienstleistung erbracht, so spricht man auch von *Begegnungen (Encounter)* (vgl. [Gut95]). Bei Begegnungen werden Schnelligkeit, Effizienz und Einheitlichkeit der Serviceleistung angestrebt, mit dem Ziel, die aus der Herstellung von Sachgütern bekannte Massenproduktion auch im Servicebereich zu ermöglichen.

2.1.3 Öffentliche Dienstleistungen

Im Hinblick auf den Begriff der öffentlichen Dienstleistung wird oft die Frage gestellt, ob man hier überhaupt von Dienstleistungen, geschweige denn kundenorientierten Dienstleistungen sprechen kann. Bogumil & Kißler [BogKiß95, S. 2] weisen darauf hin, dass öffentliche Leistungen wie z.B. die verschiedenen Ausweispapiere, Baugenehmigungen oder Kfz-Kennzeichen eher als *aufgezwungene Leistungen* zu bezeichnen sind und kaum als Dienst im eigentlichen Sinne interpretiert werden können. Somit könnte man in diesen Fällen auch von einer *aufgezwungenen Bedürfnisbefriedigung* oder von *Zwangserwerb* sprechen (vgl. auch [Sau99, S. 28]). Ebenso ist im Bereich der öffentlichen Verwaltung die Individualisierung von Leistungen mit dem Ziel einer erhöhten Kundenbindung kein Ziel, da der „Bürger oft keine Wahl zwischen verschiedenen öffentlichen Leistungen“ [Sau99, S. 28]

[9] Dies verschafft Marktvorteile und sichert die Position des Unternehmens. Der Dienstleister kennt die Vorlieben und Eigenschaften des Kunden und kann somit seine Leistung und sein Verhalten individuell anpassen. Ebenso kennt der Kunde den Serviceerbringer und kann sich ebenfalls auf diesen einstellen.

hat und einem Monopolisten gegenübersteht, dessen „Angestellte“ im Hinblick auf ihre Bezahlung und ihren Arbeitsplatz vollständig unabhängig von der Zufriedenheit des Abnehmers ihrer Leistungen sind. Des weiteren sind die Möglichkeiten auf besondere Bedürfnisse der Bürger zu reagieren aufgrund der Vielzahl von Gesetzen, welche die Leistungserbringung meist genau vorschreiben, sehr begrenzt.

Eine weitere Besonderheit öffentlicher Dienstleistungen stellt der *hohe Aufwand* für den Kunden dar, öffentliche Leistungen in Anspruch zu nehmen (vgl. [Sch98, S. 18]). Aufgrund der sich aus der Organisationsstruktur der Verwaltung ergebenen hohen Arbeitsteilung ist der Bürger oft gezwungen, „von Tür zu Tür zu gehen“, um sein Anliegen bearbeiten zu lassen bzw. sich über den Bearbeitungszustand zu informieren.

Trotz der Tatsache, dass sich das Verständnis von öffentlichen Leistungen als Service eher schwierig erweist, kann man die Bezeichnung als Dienstleistung darüber rechtfertigen, dass Public Services einen Grossteil der oben herausgearbeiteten wesentlichen Dienstleistungsmerkmale wie Immaterialität, Integration des externen Faktors oder Gleichzeitigkeit von Produktion und Konsum aufweisen (vgl. auch [Sau99, S. 11 ff.]). Ebenso liegt zwischen Bürger und Verwaltungsangestellten ein Anbieter/Nachfrager-Verhältnis trotz eventueller Aufgezwungenheit vor, dass durch Gesetze und Vorschriften geregelt wird, die somit die Funktion eines Vertrages übernehmen (vgl. [BogKiß95, S. 6]).

Saueressig arbeitet zwei weitere Merkmale öffentlicher Dienstleistungen heraus, die sie zusätzlich zu den „normalen“ Services aufweisen: der *nichtrivalisierende Konsum* und das *fehlende Ausschlussprinzip* (vgl. [Sau99, S. 9]). Nichtrivalisierender Konsum bezieht sich darauf, dass mehrere Personen die Leistung konsumieren können, ohne sich gegenseitig zu beeinträchtigen. Nichtausschließbarkeit meint, dass niemand aus technischen, ökonomischen oder sozialen Gründen an der Nutzung einer Leistung gehindert werden kann. Öffentliche Leistungen sind also allgemein zugänglich und eine größere Nachfrage führt z.B. nicht zu einem höheren Preis.

Public Services können somit als Dienstleistungen verstanden werden. Mittlerweile wird dem Prinzip der Kundenorientierung auch in der öffentlichen Verwaltung ein höherer Stellenwert zugeordnet und es wird v.a. mit Hilfe elektronischer Medien versucht, den Kundenservice zu verbessern und den Bürger auch tatsächlich als Kunden wahrzunehmen, um das Verhältnis zwischen Staat und Bürger zu verbessern (vgl. z.B. [Sch00]). Kapitel 2.2 geht auf diesen Punkt näher ein und stellt Konzepte vor, wie dies erreicht werden soll.

2.1.4 Was sind Dienstleistungen?

Obige Ausführungen verdeutlichen, dass sich eine allgemeine Begriffsbildung dessen, was eine Dienstleistung ist, als sehr schwierig wenn nicht sogar unmöglich erweist. Je nach Blickwinkel und Verwendungskontext kann ein anderes Dienstleistungsverständnis aufgebaut werden.

So findet der Servicebegriff z.B. auch in der Informatik unter einem anderen Verständnis Verwendung, wobei sich allerdings auch in diesem Bereich noch kein allgemeiner Konsens etablieren konnte (siehe. z.B. [Ker93], [Mer99b], [Zül98] und Kapitel 3). Im Rahmen der Softwaretechnik oder auch der verteilten Systeme sind mit Dienstleistungen meist die Leistungen gemeint, die ein Softwarebaustein (Server) seinen benutzenden Softwarebausteinen (Clients) über eine Schnittstelle (Menge von Operationen) anbietet. Wie die konkrete Leistung erbracht wird, bleibt den Leistungsnehmern entsprechend des Geheimnisprinzips nach Parnas [Par72] verborgen. Zur eigenen Leistungserbringung kann der Server wiederum auf andere Server zugreifen und wird somit selbst zum Client. Server können z.B. Objekte, Module oder Komponenten eines lokalen oder auch verteilten Systems sein. Dieses Verständnis der Verbindung zwischen Softwarebausteinen wird auch als

Dienstleistungsprinzip bezeichnet (vgl. z.B. [KilGryZül94, S. 48]) und entspricht dem Verständnis von Dienstleistungen als Dienstleistungsprodukte (vgl. Abschnitt 2.1.1).

Der Versuch, (soziale) Dienstleistungen durch die Aufzählung charakteristischer Merkmale fassbar zu machen, weist den Nachteil der Allgemeinheit auf und sagt nur wenig z.B. über die Qualität der Dienstleistung, ihren Inhalt oder die Organisation der Dienstleistungserstellung aus (vgl. [Ert86, S. 16]). Der Ansatz, Services als soziale Beziehungen zwecks Bedürfnisbefriedigung mit personalisiertem, also auf das Individuum zugeschnittenem, Charakter zu sehen, fokussiert im Gegensatz auf die vernachlässigten Aspekte einer Merkmalsaufzählung und beschäftigt sich gerade mit der Frage, was die Qualität einer Dienstleistung ausmacht. Zusätzlich müssen auch betriebswirtschaftliche Gesichtspunkte beachtet werden, so dass eine schnelle und kosteneffiziente Leistungserbringung gewährleistet werden kann. Um wirklich kundenorientierte Dienstleistungen erbringen zu können, ist es wichtig, diesen Punkten besondere Aufmerksamkeit zukommen zu lassen.

Für den weiteren Verlauf der Untersuchung wird unter Berücksichtigung des Umstandes, dass nicht von einem allgemeingültigen Dienstleistungsbegriff ausgegangen werden kann, das bisher herausgearbeitete Verständnis von Dienstleistungen zugrunde gelegt, welches auch z.B. mit der im Brockhaus [Bro88] zu findenden Definition größtenteils übereinstimmt[10], und sich auf eine Merkmalsaufzählung abstützt. Dort, wo eine andere Auffassung des Servicebegriffs von Nöten ist, wird explizit darauf hingewiesen. Der Punkt, dass kundenorientierte Dienstleistungen auch während der Leistungserstellung an individualisierte Bedürfnisse angepasst werden können (sollen), wird zusätzlich in die Merkmalsaufzählung aufgenommen. Da diese weitreichende Einflussmöglichkeit des Kunden bei materiellen Gütern meist nicht gegeben ist (die Einflussmöglichkeit endet meist bei der Leistungsvereinbarung bzw. Vorauswahl; vgl. [KliWet00]), stellt dies eine weitere Besonderheit von Dienstleistungen dar.

Dienstleistungen zeichnen sich also für den weiteren Verlauf der Arbeit durch folgende Charakteristika aus:

- (Überwiegende) Immaterialität
- Integration des externen Faktors
- Gleichzeitigkeit von Produktion und Konsum (Uno-Actu-Prinzip)
- Bedarfsdeckung erfolgt durch Leistung
- Mehrstufige Leistungsproduktion
- Kundenorientierte Dienstleistungen können auch nach der Leistungsvereinbarung an individualisierte Bedürfnisse angepasst werden

[10] „Dienstleistungen können allgemein als ökonomische Güter aufgefasst werden, die wie Waren (Sachgüter) der Befriedigung menschlicher Bedürfnisse dienen. Im Unterschied zu Sachleistungen zeichnen sich Dienstleistungen als an Personen gebundene nutzenstiftende Leistungen durch mangelnde Dauerhaftigkeit und Lagerfähigkeit, durch Standortgebundenheit, durch Gleichzeitigkeit von Produktion und Konsum sowie durch vergleichsweise arbeitsintensive Produktion aus; sie werden oft auch als immaterielle Güter bezeichnet.“ [Bro88]

2.2 Das Anwendungsfeld eGovernment

Der Begriff des *electronic Government (eGovernment)* ist bis jetzt noch nicht eindeutig definiert worden. Oft ist auch nicht klar, was darunter zu verstehen ist (vgl. [Sch00]). Sicher ist jedoch, dass elektronische Medien, insbesondere das *Internet* bzw. das darauf basierende *World Wide Web (WWW)*[11], auch im administrativen Bereich eine immer größer werdende Rolle spielt. Schedler [Sch00, S. 34] hebt hervor, dass das wirklich neue an eGovernment nicht die „Informatisierung der verwaltungsinternen Abläufe, sondern der Aspekt der Kommunikation zwischen Verwaltung und Dritten über elektronische Medien und die daraus abgeleiteten organisatorischen Änderungen in den öffentlichen Institutionen“ darstellt.

Vor diesem Hintergrund definiert er eGovernment folgendermaßen:

> *„Electronic Government ist eine Organisationsform des Staates, welche die Interaktionen und Wechselwirkungen zwischen dem Staat und den Bürgern, privaten Unternehmen, Kunden und öffentlichen Institutionen durch den Einsatz von modernen Informations- und Kommunikationstechnologien (IKT) integriert.“ [Sch00, S. 35]*

eGovernment meint also die Summe der neuen Möglichkeiten, die sich den staatlichen Institutionen bieten, sich mit Anderen auf elektronischem Wege auszutauschen. Erklärtes Ziel des eGovernment ist die interne und externe Leistungsverbesserung sowie die konsequente Ausrichtung der öffentlichen Institutionen auf die Abnehmer ihrer Leistungen. Auch im Verwaltungsbereich wird dem Konzept Kundenorientierung (vgl. Kapitel 1) somit steigende Bedeutung beigemessen und die Bedürfnisse der Bürger in den Vordergrund gestellt. Damit einher geht meist eine Veränderung der internen Leistungserstellung und die Aufbrechung etablierter Organisationsstrukturen, d.h. die Anpassung der Aufbau- und Ablauforganisation[12] an diese Strategie.

Schedler identifiziert drei Elemente des eGovernment, die im folgenden kurz dargestellt werden (vgl. Abbildung 2):

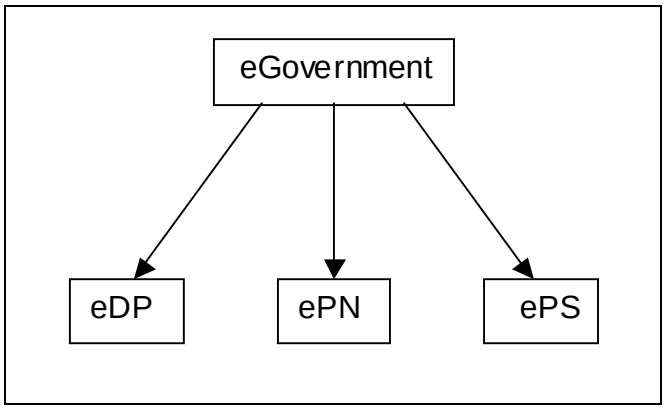

Abbildung 2: Die drei Elemente des eGovernment

[11] Zum Begriff des Internet bzw. WWW siehe z.B. [RecPom, S. 839 ff.].

[12] Zum Begriff der Ablauf- bzw. Aufbauorganisation siehe z.B. [JabBöhSch97].

- *Electronic Democracy and Participation (eDP):* Elektronische Abbildung demokratisch legitimierender Entscheidungsverfahren bzw. deren Vorbereitung. (z.B. *eVoting*)
- *Electronic Production Networks (ePN):* Formen der Zusammenarbeit zwischen öffentlichen und privaten bzw. öffentlichen Institutionen via elektronische Medien. (z.B. *eProcurement*[13], *Aufbau virtueller Produktionsnetzwerke*)
- *Electronic Public Services (ePS):* Abgabe von öffentlichen Leistungen an die Leistungsempfänger, Privatpersonen oder Unternehmen über lokale, regionale oder nationale Portale. (z.B. elektronische Eingabe der Steuererklärung)

Von Relevanz für diese Arbeit sind primär die beiden letztgenannten Elemente des eGovernment, nämlich die ePN und ePS. Bei ePS handelt es sich um den für den Kunden bzw. Bürger sichtbaren Teil des eGovernment und zielt auf die Verbesserung (und Erleichterung) der Kommunikation zwischen Verwaltung und Leistungsempfänger ab. ePN bezieht sich mehr auf den für den Bürger unsichtbaren Teil und beschäftigt sich mit der Verbesserung der internen Leistungserstellung, was allerdings Rückwirkungen auf das externe Leistungsangebot hat, z.B. in Form effizienterer Leistungsproduktion.

Wie aus der Begriffsbildung von Schedler schon ersichtlich, geht es im ePS-Bereich vornehmlich darum, Public Services von den Leistungsempfängern über elektronische Medien, insbesondere dem WWW, zugänglich zu machen, und somit die Möglichkeit anzubieten, Behördengänge quasi von zuhause erledigen zu können. Hier gewinnen sogenannte Portale (siehe Abschnitt 2.4) eine immer größere Bedeutung, über die verschiedene, nach intendierten Benutzergruppen ausgerichtete Leistungen nach bestimmten Kriterien gebündelt angeboten werden (vgl. auch [Mei01]). Die Bürger können die ePS je nach Bedarf und rund um die Uhr über diese eine Schnittstelle abrufen und sich quasi selbst bedienen was durch das Schlagwort *„Non-Stop-Government"* beschrieben wird.

Hinter diesen Entwicklungen steht die Philosophie des *„One-Stop-Shops"* [Sch00] oder auch *„One-Stop-Service"* [Käs00]. Der Bürger soll nicht mehr gezwungen sein, verschiedenste Institutionen der Verwaltung aufzusuchen, wenn er Verwaltungsangelegenheiten erledigen möchte. Es geht um die „(...) Bündelung von inhaltlich zusammenhängenden Dienstleistungen in einer einzigen Kontaktstelle (...)" [Käs00] um die notwendigen Behördengänge auf ein Minimum zu reduzieren und somit bürgerfreundlicher zu werden.

Allerdings handelt es sich bei Portalen nur um eine weitere (virtuelle) Möglichkeit, die es dem Bürger ermöglicht, Public Services an einem Ort in Anspruch zu nehmen. Es muss weiterhin die (reale) Möglichkeit bestehen, auf öffentliche Dienstleistungen direkt, also durch persönlichen Kontakt mit Mitarbeitern der Verwaltung, zuzugreifen. Neben der notwendigen Wahrung des Nichtausschließbarkeitsprinzips (vgl. Abschnitt 2.1.3) resultiert diese Forderung insbesondere daraus, dass spezielle Anliegen der Bürger und die daraus entstehende besondere Situation von Antragsstellern der Beurteilung durch qualifizierte Mitarbeiter bedarf. Dem Gedanken des „One-Stop-Service" kann dadurch entsprochen werden, dass die verschiedenen Dienstleistungen der Behörden an einem Schalter in sog. *Bürgerbüros* oder *Bürgerämtern*[14] angeboten werden (vgl. [Käs00], [Bod99, S. 158]). Hier ist IKT anzubieten, die speziell den öffentlichen Dienstleister unterstützt und ihm ermöglicht, alle benötigten Informationen abzurufen und Leistungsprozesse[15] anzustoßen, die sowohl von internen als

[13] eProcurement steht für elektronische Beschaffung (siehe z.B. [BTZZ00]).

[14] „Das Konzept des Bürgeramtes zielt darauf ab, bei der Bearbeitung von Anliegen, bei denen ein persönlicher Kontakt mit dem Bürger notwendig ist, möglichst viele Dienstleistungen an einer einzigen Anlaufstelle zu erbringen." [Bod99, S. 158]

[15] Zum Prozessbegriff siehe Abschnitt 2.3.1.

auch von externen Partnern der Verwaltung erbracht werden, um das Bedürfnis des Kunden zu befriedigen.

Die (digital) angebotenen Services lassen sich prinzipiell in folgende drei Kategorien unterteilen (vgl. [Käs00]):

- *Informationsdienste*: Dem Bürger werden Informationen über Öffnungszeiten der Ämter, Zuständigkeiten, Ansprechpartner, Planungs- und Entscheidungsverfahren, usw. angeboten.
- *Kommunikationsdienste*: Dem Bürger werden Möglichkeiten zur direkten Kontaktaufnahme mit zuständigen Ansprechpartnern der Verwaltung angeboten, beispielsweise über eMail oder Chat-Rooms.
- *Transaktionsdienste*: Hier wird dem Bürger die Möglichkeit angeboten, Verwaltungsprozesse direkt anzustoßen.

Dienstleistungen der ersten beiden Kategorien werden mittlerweile von vielen behördlichen Institutionen angeboten[16]. Transaktionsdienste sind dagegen noch rar. Sie implizieren meist die Teilnahme mehrerer räumlich getrennter administrativer Einheiten (bedingt durch die meist tayloristische Organisationsstruktur[17]), zusätzlich können auch private Unternehmen in die Leistungserbringung involviert sein.

Unter dem Begriff eGovernment-Service wird in dieser Arbeit somit folgendes verstanden:

> *eGovernment-Services sind öffentliche Dienstleistungen (Information, Kommunikation, Transaktion). Sie können vom Leistungsempfänger direkt oder von Angestellten der Verwaltung auf elektronischem Wege zugegriffen werden. Die Leistungsproduktion erfolgt unter Nutzung von IKT und kann auch private Organisationen integrieren.*

Die wesentlichen Vorteile, die man sich von eGovernment verspricht, sind zusammengefasst also zum einen die Verbesserung der Verwaltungsleistung, denn der Bürger hat 24 Stunden lang, ohne Wartezeiten und unabhängig vom Ort einen Zugang zu den Behörden („Non Stop“). Zum anderen wird die Bürgerfreundlichkeit dadurch erhöht, dass die benötigten Services nach bestimmten Kriterien gebündelt und über eine Schnittstelle zugänglich gemacht („One-Stop“) und im Rahmen der gesetzlichen Vorgaben an die speziellen Bedürfnisse der Bürger angepasst werden. Der „hohe Aufwand“ des Bürgers soll reduziert werden (vgl. Abschnitt 2.1.3). Des weiteren erhofft man sich Kosteneinsparungen und eine Effizienzsteigerung, insbesondere durch die Optimierung und Vereinfachung von Arbeitsprozessen innerhalb der Verwaltung, aber auch organisationsübergreifend mit ihren Partnern und Zulieferern. (vgl. [Käs00])

eGovernment wird oft als „eBusiness of the state“ bezeichnet (vgl. [SchHäu01]). Schubert und Häusler definieren *eBusiness* folgendermaßen:

> *„eBusiness is a business model and focuses on the support of processes and relationships between business partners, employees and customers by means of electronic media. (…) eBusiness is the electronic support of business relationships from the perspective of an enterprise.” [SchHäu01]*

Vergleicht man diese Definition mit den bisherigen Ausführungen, so lassen sich eine Reihe von Gemeinsamkeiten erkennen. Ein wesentlicher Unterschied besteht jedoch darin, dass bei eGovernment der kommerzielle Aspekt nicht im Vordergrund steht und die Art und

[16] Z.B. der „Direkter-Bürger-Informations-Service“ der Stadt Hamburg (http://dibis.dufa.de) (02.2001)

[17] Einen Überblick zu Optionen der Aufbauorganisation liefert z.B. [Rol98].

Weise, wie Leistungen produziert werden und welche Form das Ergebnis zu haben hat, zumeist eindeutig durch Gesetze und Vorschriften geregelt und somit standardisiert ist (für weitere Unterschiede siehe Kapitel 2.1.3). Im Hinblick auf eine informationstechnische Unterstützung sind jedoch ähnliche Herausforderungen zu bewältigen und die Nutzung des Internets als kostengünstige und allgemein verfügbare Basisplattform wird in beiden Bereichen favorisiert. Aus diesem Grund werden die untersuchten Konzepte im weiteren Verlauf der Arbeit hauptsächlich auf einer allgemeinen Basis betrachtet, wobei immer wieder Bezug auf die Besonderheiten des administrativen Bereiches genommen wird.

Angemerkt sei an dieser Stelle, dass auf Aspekte, welche die notwendigen strukturellen Veränderungen innerhalb der Verwaltungen (oder auch Unternehmen im kommerziellen Bereich) betreffen, aus Platzgründen, wenn überhaupt, nur oberflächlich eingegangen werden kann. Ebenso werden sicherheitsrelevante Aspekte nur am Rande behandelt, und es sei auf die Literatur verwiesen[18]. Das untersuchte Anwendungsfeld beschränkt sich weiterhin auf den Bereich des eGovernment, der in Verbindung mit der Abgabe öffentlicher Dienstleistungen an Bürger steht. Dies betrifft zum einen die Unterstützung von Selbstbedienung über Internetportale und zum anderen die Unterstützung der (organisationsübergreifenden) Dienstleistungserstellung im Rahmen von Transaktionsdienstleistungen, die z.B. über Internetportale zugänglich gemacht werden.

Im nächsten Abschnitt wird u.a. der Ansatz Serviceflow Management vorgestellt, der in dieser Arbeit gewählt worden ist, um die Softwareentwicklung zur Unterstützung von Transaktionsdienstleistungen anzuleiten.

[18] z.B. [Mer99] oder [Sau99], die jeweils einen Überblick über die relevanten Konzepte bieten.

2.3 Prozessunterstützung im Service-Bereich

Der Serviceflow Management Ansatz und die Anforderungen, die an eine Prozessunterstützung im Bezug auf die softwaregestützte Durchführung von Transaktionsdienstleistungen im Rahmen dieses Ansatzes gestellt werden, stehen im Mittelpunkt dieses Kapitels.

Das Kapitel beginnt mit der Diskussion darüber, was unter den Begriffen Prozess und Prozessunterstützung in dieser Arbeit verstanden wird.

Hiernach wird auf den Serviceflow Management Ansatz eingegangen und abschließend ein Anforderungskatalog erstellt, den Anwendungssoftware erfüllen sollte, um eine adäquate Prozessunterstützung nach diesem Ansatz anbieten zu können.

2.3.1 Der Prozessbegriff und Optionen der Prozessunterstützung

Oft wird der Begriff *Prozess* synonym zum Begriff *Vorgang* verwendet. Nach Mittelstraß [Mit96] kann ein Vorgang (allgemein) folgendermaßen definiert werden:

> *„Bezeichnung für ein Ereignis unter ausdrücklicher Berücksichtigung seines Verlaufs, also seiner dynamischen Binnenstruktur; synonym mit Prozess. [...]“*

In [CurKelOve92] wird ein Prozess als „a set of partially ordered steps intended to reach a goal“ definiert. Die Komponenten eines Prozesses werden als Prozesselemente (*process element*) bezeichnet und ein Prozessschritt (*process step*) als eine atomare Aktion, die keine außen sichtbare interne Sub-Struktur aufweist. Ob ein Prozesselement als ein Prozessschritt bezeichnet wird, hängt davon ab, ob eine weitere Dekomposition des Prozesselementes im Hinblick auf die Zielerreichung notwendig bzw. sinnvoll ist, d.h. ob das Prozesselement selbst wieder einen Prozess darstellt oder nicht. Nichtatomare Prozesselemente werden auch als *Teilprozesse* bezeichnet (vgl. z.B. [Rol98]).

Andere Autoren sprechen im Zusammenhang mit Prozessen, an denen menschliche Akteure[19] beteiligt sind, auch von Aufgaben, Arbeitsabläufen und Aktivitäten:

> *„Eine Aufgabe ist eine Vorgabe zum zielorientierten Handeln. Diese Vorgabe kann von außen oder von den Handelnden selbst kommen. Die Erfüllung der Aufgaben erfolgt in Arbeitsabläufen, die in einzelne Aktivitäten strukturiert sind. Die Ausübung von Aktivitäten erfolgt unter gewissen Vorraussetzungen und liefert Ergebnisse. (...)“[Pae00, S. 21]*

Aktivitäten bilden hierbei das eigentliche Geschehen der Aufgabenerfüllung, die Aufgabe beinhaltet zusätzlich noch die zugrundeliegenden Ziele. Prozesse als Arbeitsabläufe sind Folgen von Aktivitäten, die von Akteuren durchgeführt werden, um ein bestimmtes Ziel zu erreichen. Jede Aktivität kann selber wieder einen Prozess darstellen.

Arbeitsabläufe in Wirtschaftseinheiten werden oft als *Geschäftsprozesse* bezeichnet. Jablonski et al. [JabBöhSch97] definieren einen Geschäftsprozess entsprechend:

> *„Ein Geschäftsprozess ist ein Vorgang in Wirtschaftseinheiten (Unternehmen, Verwaltungen, etc.), der einen wesentlichen Beitrag zu einem nicht notwendigerweise ökonomischen Unternehmenserfolg leistet. Dabei läuft er in der Regel funktions-, hierarchie- und standortübergreifend ab, kann die*

[19] Zum Akteursbegriff siehe z.B. [Rol98]. Akteure können Mitarbeiter, Abteilungen, das Unternehmen als ganzes, seine Geschäftspartner, usw. sein. Technische Akteure sind z.B. Softwaresysteme.

Unternehmensgrenzen überschreiten und erzeugt einen messbaren, direkten Kundennutzen." [JabBöhSch97, Glossar]

Diese Definition macht zunächst deutlich, dass als Geschäftsprozesse nur solche Vorgänge bezeichnet werden, die im hauptsächlichen Tätigkeitsfeld eines Unternehmens liegen, d.h. die Organisation hat den Vorgang zu ihrem Geschäftszweck (Geschäftsaufgabe) erklärt[20]. So würde z.B. der Prozess „Reisekosten abrechnen" nicht als Geschäftsprozess bezeichnet werden, es sei denn, ein Unternehmen wendet sich diesem Vorgang geschäftsmäßig zu. Der Begriff des Vorgangs oder Prozesses ist also im Gegensatz zum Geschäftsprozess allgemeiner zu fassen[21]. (vgl. [JabBöhSch97, S. 24])

An Geschäftsprozessen sind meist eine Vielzahl von Organisationseinheiten beteiligt, die ihren Anteil an der Erfüllung der Gesamtaufgabe, also der Erreichung des übergeordneten Zieles des Geschäftsprozesses, übernehmen. Arbeiten mindestens zwei Akteure (Personen) arbeitsteilig zusammen um ein gemeinsames Ziel zu erreichen, so spricht man davon, dass sie kooperieren. Grundlage für *Kooperation* ist zum einen *Kommunikation*, also die Verständigung zwischen mehreren Akteuren durch Informationsaustausch und zum anderen die *Koordination* (vgl. [Teu95]). Unter Koordination ist die Kommunikation zu verstehen, die zusätzlich notwendig ist, um Aktivitäten aufeinander abzustimmen. Malone et al. [Mal99] definieren Koordination wie folgt :

„(…) coordination can be defined as managing dependencies among activities." [Mal 99]

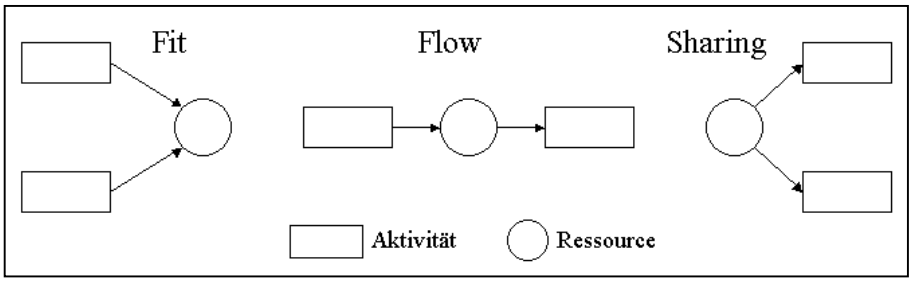

Abbildung 3: Basis-Abhängigkeitstypen zwischen Aktivitäten [Mal99]

Sie identifizieren grundsätzlich drei verschiedene Abhängigkeitstypen. So ist z.B. zur Durchführung einer Aktivität das Leistungsergebnis einer vorhergehenden Aktivität notwendig (z.B. bestimmte Informationen). Weiterhin kann eine Ressource von mehreren Aktivitäten gleichzeitig benötigt werden (z.B. bestimmte Dokumente) oder Leistungsergebnisse sind zusammenzuführen (z.B. bestimmte Teile eines Dokuments), um eine bestimmte Ressource zu produzieren. Malone et al. bezeichnen die Abhängigkeitstypen als „flow dependencies", „sharing dependencies" und „fit dependencies" (vgl. Abbildung 3). Sie weisen darauf hin, dass es sicherlich noch weitere Abhängigkeitstypen geben kann, vermuten jedoch, dass es sich hierbei nur um Spezialisierungen oder Kombinationen der genannten handelt.

[20] Je nach Anwendungsgebiet und zu produzierender Leistung kann nach [BBKKSS99] statt von einem Geschäftsprozess spezifischer z.B. auch von einem *Softwareprozess* (Entwicklung und Wartung von Software) oder *Designprozess* (Entwurf von Objekten, z.B. Schaltungen) gesprochen werden. Bei vorwiegend Dienstleistungen produzierenden Organisationen findet auch der Begriff *Dienstleistungsprozess* Verwendung (vgl. [Bod99]).

[21] Andere Autoren unterscheiden aus diesem Grund auch zwischen Prozessen, die direkt den Unternehmenserfolg ausmachen (Führung-, Haupt- oder Kernprozess) und Prozessen, welche die Durchführung der Hauptprozesse (Unterstützungs-, Service- oder Supportprozess) unterstützen (vgl. [Kru97], [GRVR94])

Relevant für diese Arbeit sind v.a. Abhängigkeitsbeziehungen entsprechend des „flow dependencies“-Typs. Dieser Typ kann in den meisten Prozessen wiedergefunden werden (vgl. [Mal99]). Er ist insbesondere dann von Relevanz, wenn die einzelnen Aktivitäten von relativ unabhängig voneinander agierenden Akteuren an verschiedenen Orten und zu verschiedenen Zeitpunkten (nacheinander) durchgeführt werden, deren Leistungen aufeinander aufbauen und schließlich das Leistungsendergebnis ergeben. Zur Durchführung der Teilaufgaben oder zu Informationszwecken müssen die Teilleistungsergebnisse zwischen den Akteuren ausgetauscht werden. Die Beziehung der einzelnen Aktivitäten untereinander kann im Sinne eines Kunden/Lieferanten-Verhältnisses interpretiert werden, das durch vereinbarte Anforderungen im Hinblick auf zu erbringende Vorleistungen und garantierten Leistungsergebnisse, im Sinne eines Vertrages, geprägt ist. Krumbiegel spricht im Zusammenhang mit Geschäftsprozessen hier auch von *Geschäftsprozessschnittstellen*, die aus den Transaktionen zum Austausch der Leistungsergebnisse und Lenkungsnachrichten zwischen Aktivitäten und den hierzu zusätzlich notwendigen Aktionen bestehen (vgl. [Kru97, S. 97]).

Zum Austausch der notwendigen Koordinationsinformationen und zu bearbeitenden Ressourcen (soweit es sich hierbei wieder um Informationen bzw. Daten handelt), sowie zur zeitlichen Ordnung von Aktivitäten und deren Zuordnung zu den jeweiligen Akteuren, wird zunehmend IKT eingesetzt, sobald zeitliche und räumliche Grenzen zu überwinden sind (vgl. [BBKKSS99]). Die auszutauschenden Informationen und Abhängigkeitsbeziehungen werden in sogenannten *Prozessmodellen* beschrieben, welche die Grundlage zur Realisierung einer Computerunterstützung in diesem Bereich darstellen.

In [Mer96, S. 334] werden *Prozessmodelle*[22] folgendermaßen definiert:

> *„Ein Prozessmodell ist ein immaterielles Abbild eines Prozesses oder mehrerer gleichgearteter Prozesse [...]“.*

Prozessmodelle dienen dazu (vgl. [SchMac00]),

- eine eindeutige Beschreibung dessen zu liefern, was zu tun ist,
- festzulegen, wer was wann wie und womit zu erledigen hat,
- und sicherzustellen, dass die Ergebnisse wiederholbar werden.

Prozessbeschreibungen können somit als Verfahrens- oder Handlungsvorschriften aufgefasst werden. Sie beschreiben routinisierte[23] Arbeitsabläufe, die in einem ausreichenden Maße strukturiert und „festgeklopft“ werden können.

Allerdings existiert neben der Auffassung eines Prozessmodells als Handlungsvorschrift auch eine andere Interpretation. Man kann Beschreibungen routinisierter Vorgänge auch im Sinne von Plänen oder Mustern auffassen, die als Handlungsanleitungen zu verstehen sind (vgl. z.B. [Gry96]). Sie vergegenständlichen menschliche Erfahrungen und repräsentieren etablierte und bewährte Arbeitsabläufe, die als Ressource dienen, um in der konkreten Situation das Handeln anzuleiten, es aber nicht vorzuschreiben. Hierbei wird berücksichtigt, dass trotz der Existenz einer allgemeingültigen Arbeitsvorgangsbeschreibung konkrete Vorgänge aufgrund aktueller Anforderungen auch eine abweichende Durchführung erfordern können.

---

[22] In Kapitel 3.2 wird am Beispiel der Serviceflow-Modellierung (zum Begriff der Serviceflow siehe weiter unten) aufgezeigt, wie Prozessmodellierung konkret aussehen kann.

[23] Unter einer Routine kann „(...) eine wiederkehrende Handlungsfolge mit bekanntem Ergebnis (...)“ [Gry96] verstanden werden.

Die beiden Interpretationsmöglichkeiten der Aufgabe von Modellen routinisierter Arbeitsabläufe spiegeln sich auch darin wieder, wie die Entwicklung und der Einsatz von Software in diesem Bereich verstanden wird. Sie manifestieren sich in den Begriffen *ablaufsteuernde* und *unterstützende Sichtweise*.

Wird Anwendungssoftware nach der ablaufsteuernden Sichtweise entwickelt, so bedeutet dies, die wiederholbaren Anteile der Arbeitsabläufe im Anwendungsbereich zu identifizieren, daraus einen allgemeingültigen Ablauf zu bilden und diesen zu algorithmisieren, also auf den Computer zu übertragen. (vgl. [Zül98, S. 78]) Als Beispiel für Anwendungssoftware, die dieser Sichtweise folgt, können traditionelle WfMS (siehe [JabBöhSch97]) genannt werden. Dort wird die gesamte Logik des Arbeitsablaufs im System implementiert, d.h. die Prozessmodelle so weit verfeinert und formalisiert, dass sie von einer Ausführungsmaschine interpretiert werden können und diese dann die Steuerung des jeweiligen konkreten Arbeitsprozesses und die Verantwortung für dessen korrekte Ausführung übernehmen kann ([BBKKSS99]). Nachteilig ist an diesem Ansatz, dass alle möglichen Abweichungen und Sonderfälle, die bei der Prozessausführung entstehen könnten, vorausgeplant und implementiert werden müssen. Dies ist jedoch meist nicht möglich. Somit ist es erforderlich, dass der Ablauf bzw. das ihm zugrundeliegende Modell zur Ausführungszeit modifiziert werden kann, was allerdings bei gängigen WfMS oft nicht der Fall ist (vgl. z.B. [BlaTes99]).

Wird Anwendungssoftware dagegen nach der unterstützenden Sichtweise entwickelt, so ist das Ziel, dass die Verantwortung und Kontrolle der Arbeitabläufe weiterhin bei den Benutzern des Softwaresystems verbleibt (vgl. [Zül98, S. 80 ff.]). Um im Sinne dieses Verständnisses auch im Kontext der Unterstützung von Routinevorgängen unvorhersehbare Modifikationen am zugrundeliegenden Prozessmodell zu ermöglichen, wird der Ansatz verfolgt, das Prozessmodell selber explizit und bearbeitbar zu machen (zu vergegenständlichen) und zwischen den Akteuren zusätzlich zu den Arbeitsgegenständen auszutauschen (vgl. [Zül98, S. 448]). Dadurch wird es möglich, die Kontrolle über den weiteren Verlauf des Vorgangs in die Verantwortung des bearbeitenden Akteurs zu legen. Dieser Variante wird auch im Serviceflow Management Ansatz gefolgt, der in Abschnitt 2.3.2 vorgestellt wird.

Wesentliche Voraussetzung hierbei ist jedoch, dass der jeweilige Akteur ein sicheres Verständnis über die Zusammenhänge der Aktivitäten des Prozesses hat. D.h. z.B., dass ihm Sinn und Zweck sowie die Dringlichkeit der eigenen und auch der weiteren Arbeitsschritte bewusst ist, die durch andere Akteure wahrgenommen werden und dass er dies in seiner Entscheidung berücksichtigen kann (vgl. [Zül98, S. 448]). In komplexen Arbeitsabläufen, an denen viele Akteure beteiligt sind, kann ein solches umfassendes Verständnis jedoch nur schwerlich vorausgesetzt werden. Der Notwendigkeit, auch die Geschäftsprozessschnittstellen explizit zu machen, kommt hierbei eine entscheidende Bedeutung zu. Dies wird ebenfalls im Serviceflow Management Ansatz berücksichtigt.

Die Verwendung des Begriffs Prozessunterstützung im Titel dieser Arbeit weist schon darauf hin, dass Prozesse und ihre softwaretechnische Kooperations- und Koordinationsunterstützung grundsätzlich von der unterstützenden Sichtweise ausgehend betrachtet werden. Die Kontrolle über den Gesamtablauf als auch die Kontrolle über den Prozess des jeweiligen (menschlichen) Akteurs soll möglichst weitgehend bei den Akteuren selbst verbleiben und nicht in die Verantwortung einer Software gelegt werden. Akteuren ist also grundsätzlich die Freiheit zuzugestehen, den routinemäßigen Ablauf der gemeinsamen oder auch individuellen Leistungserbringung auf konkrete abweichende Situationen „einstellen“ zu können. Im Servicebereich ist diese Möglichkeit essentiell, denn nur so kann situativ auf veränderte und besondere Kundenwünsche eingegangen werden.

Dies gilt auch für den eGovernment-Bereich. Obwohl, wie in Abschnitt 2.2 bereits erwähnt, der Ablauf der arbeitsteiligen Leistungserstellung im öffentlichen Bereich meist eindeutig

durch Gesetze und Vorschriften geregelt und standardisiert ist, kann ein allgemeiner feststehender Prozessablauf im voraus meist nicht eindeutig bestimmt werden. Auch hier kann es z.B. nötig sein, den standardmäßigen Prozessverlauf situativ dahingehend abzuändern, dass vorherige Bearbeiter eine Aufgabe wiederholen müssen, nachfolgende Prozessteilnehmer geändert oder neue hinzugefügt werden (vgl. [PriKol96]).

Unter Prozessunterstützung wird in dieser Arbeit mit Fokus auf arbeitsteilige Prozesse, deren Teilleistungen aufeinander aufbauen, somit folgendes verstanden:

> *Prozessunterstützung heißt, Akteure sowohl in der Koordination des gesamten arbeitsteiligen Prozesses als auch in der Koordination von Aktivitäten innerhalb von Teilprozessen mit Hilfe von IKT zu unterstützen. Hierzu gehört mindestens die Unterstützung des Weiterleitens von zur Durchführung der jeweiligen Aktivitäten benötigten Ressourcen als auch des Austauschs zur Abstimmung notwendiger Informationen. Die Kontrolle über die Durchführung, sowohl des gesamten als auch der Teilprozesse, verbleibt soweit wie möglich und sinnvoll „in den Händen" der Prozessbeteiligten.*

2.3.2 Serviceflow Management

Klischewski und Wetzel [KliWet00] haben unter dem Begriff *Serviceflow Management (SfM)* ein Leitbild vorgestellt, dass darauf abzielt, für die Entwicklung und den Einsatz von Software zur Prozessunterstützung speziell im Dienstleistungssektor eingesetzt zu werden. Sie berücksichtigen mit ihrem Ansatz die Besonderheiten des Servicebereichs, und fokussieren darauf, Servicemitarbeiter in ihrem kundenorientierten Handeln zu unterstützen und gleichzeitig eine effiziente Leistungsproduktion zu ermöglichen.

Der SfM-Ansatz orientiert sich an einer eigenen Servicedefinition, welche die Sicht des Dienstleistungsanbieters betont:

> *„Service ist eine unternehmerische Leistung, die ihre Wertschöpfung darauf gründet, die Bedürfnisse des Kunden zu erkennen und zu befriedigen. Dafür werden nach Möglichkeit betriebliche Standardabläufe auf die Erfordernisse der jeweiligen Dienstleistungssituation angepasst." [KliWet00]*

Dem SfM-Ansatz liegt eine Servicedefinition zugrunde, die Dienstleistung in einem Spannungsfeld zwischen Routinisierung und individueller Kundenorientierung begreift. Die Notwendigkeit des situativen Erkennens und Befriedigens der Bedürfnisse des Kunden ist zentral für das Serviceverständnis in diesem Ansatz. Dem Beziehungscharakter von kundenorientierten Dienstleistungen (vgl. Abschnitt 2.1.2) wird also auch im SfM-Ansatz Beachtung geschenkt (vgl. auch [KliWetBah01]). Gleichzeitig werden betriebswirtschaftliche Aspekte, d.h. die Notwendigkeit von Standardabläufen zur zeit- und kostenoptimierten Leistungserstellung, beachtet.

SfM zielt auf eine bestimmte Art von Dienstleistung ab. Es handelt sich hierbei um „(...) Serviceleistungen, die aus aufeinander aufbauenden Teilservices bestehen. Die Gesamtdienstleistung kann nur über die Durchführung einer Reihe von Teilleistungen erfolgen." [KasKliWet01] Dienstleistungen, die praktisch anonym (z.B. Schnellrestaurant) oder die wiederholt und gerne in Anspruch genommen werden (z.B. Friseur), aber eben keine aufeinander aufbauenden Teilservices aufweisen, werden nicht von SfM adressiert.

Services werden als arbeitsteilige, personalisierbare Prozesse verstanden, wobei die in den einzelnen Teilschritten erbrachten Leistungen aufeinander aufbauen, d.h. der zeitliche Verlauf und die notwendigen Kombinationen von Teilleistungen wird in den Vordergrund gestellt (vgl. Abschnitt 2.3.1).

Im Rahmen des SfM-Ansatzes werden zwei Arten von betrieblichen Standardabläufen unterschieden. Zum einen werden darunter im Sinne der unterstützenden Sichtweise (vgl. Abschnitt 2.3.1) auf einem Muster basierende Abläufe der Leistungserbringung verstanden, die zwar routinemäßig durchgeführt werden, jedoch die Möglichkeit bieten, je nach Situation vom vorgesehenen Ablauf abzuweichen. Sie beziehen sich auf die Leistungen, die direkt für den Kunden produziert werden. Die Möglichkeit des Abweichens vom Standardverlauf ist notwendig, da diese Leistungserstellungsprozesse im Sinne der Kundenorientierung durch das individuelle Anliegen des Kunden in ihrem Ablauf beeinflussbar sein sollen. Zum anderen werden hierunter Prozesse verstanden, die zwar ebenfalls routinemäßig ablaufen, jedoch keine Abweichungen erlauben. Diese Vorgänge werden auch als Supportprozesse bezeichnet, da sie im Hintergrund ablaufen und zur Unterstützung der Leistungsprozesse, die sich direkt auf den Kunden beziehen, dienen. Ihr Ablauf kann nicht durch besondere Anliegen des Kunden beeinflusst werden. (vgl. [KliWet00])

Prozesse werden im SfM-Ansatz als *Serviceflows* bezeichnet, die als eine Folge von Teilleistungen, die zu einer Gesamtleistung zusammengefasst werden, zu verstehen sind (vgl. [KliWet00]). Die Teilleistungen des Serviceflows repräsentieren die Aktivitäten, die dem Kunden standardmäßig angeboten werden. Die Erbringung der Teilleistungen erfolgt an sogenannten *Servicepoints*, die als Kontaktpunkte aufzufassen sind, an denen Servicenehmer und –geber zwecks Leistungsaustausch zu einem bestimmten Zeitpunkt aufeinandertreffen. Dieser Ort muss kein realer Ort im Sinne eines tatsächlich existierenden Kontaktpunktes (z.B. ein Bankschalter oder ein Bürgeramt) sein. Ein Servicepoint kann auch die computerisierte Arbeitsumgebung eines Dienstleistungsmitarbeiters sein, dem sich der Kunde mit seinem Anliegen nicht persönlich, sondern mediiert durch Informationstechnologie präsentiert (z.B. im Back-Office einer Bank oder einer Verwaltung). (vgl. [KliWetBah01]) Abbildung 4 illustriert die bisherigen Ausführungen.

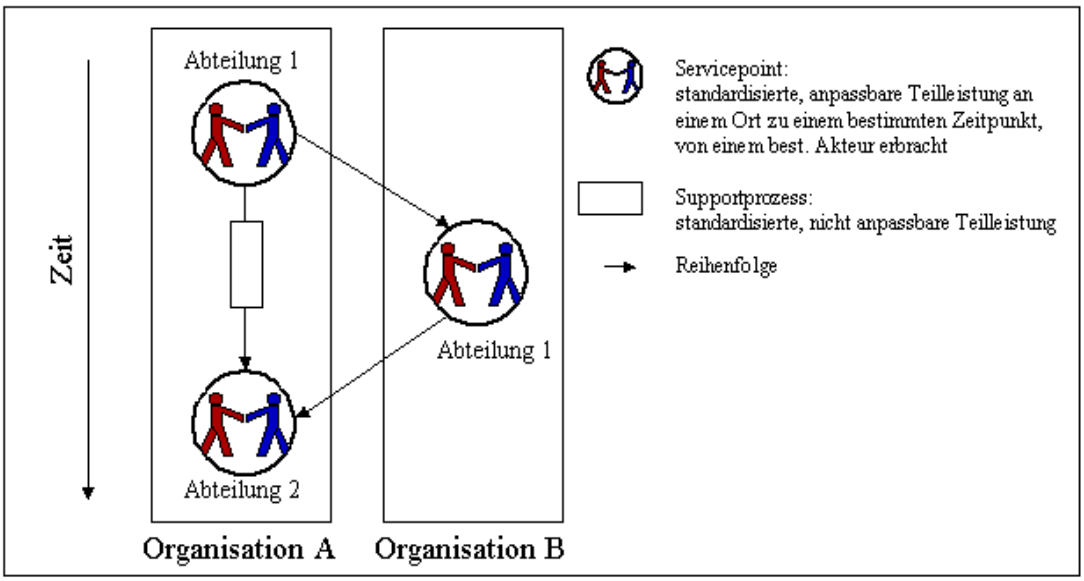

Abbildung 4: Serviceflow mit Supportprozess

An einem Servicepoint werden dem Leistungserbringer alle Hilfsmittel zur Verfügung gestellt, die er zur Erledigung seiner Aufgabe benötigt, z.B. relevante Informationen über den bisherigen Serviceverlauf oder zur Verfügung stehende Supportprozesse, die angestoßen und/oder deren Leistungsergebnisse in Empfang genommen werden können. Signifikant für einen Servicepoint ist, dass das Anliegen des Kunden aufgenommen und situativ darauf reagiert werden kann, d.h. nicht nur eine (oder mehrere) Standardleistung(en) im Sinne einer Routine erbracht wird, was dem Encounter-Gedanken (siehe Abschnitt 2.1.2) entsprechen würde, sondern auch von diesem Standard abgewichen werden kann.

Da Serviceflows sich aus einer Menge von Teilleistungen zusammensetzen, die von unterschiedlichen Akteuren erbracht werden können, sind der situativen Abweichung vom Standard allerdings auch Grenzen gesetzt, da der jeweilige Teilleistungserbringer den Beschränkungen seiner Befugnisse bzw. Fähigkeiten oder den Erwartungen die an ihn von anderen Dienstleistungspartnern gestellt werden, gerecht werden muss (vgl. [KasKliWet01]). Die Geschäftsprozessschnittstellen müssen eingehalten werden (vgl. Abschnitt 2.3.1). Findet eine situative Anpassung trotzdem statt, so resultieren daraus evtl. Abstimmungserfordernisse mit den Partnern, was an jedem Servicepoint zu berücksichtigen ist, damit nachfolgende Servicepoints ihre Leistung auf veränderte Geschäftsprozessschnittstellen anpassen können. Es kann somit auch Servicepoints geben, an denen die prinzipiell mögliche Flexibilität eingeschränkt wird, z.B. aus Gründen, die sich aus dem übergeordneten Serviceflow ergeben (Geschäftsprozessschnittstellen dürfen nicht verändert werden) oder Leistungen abgefordert werden, die aus Kostengründen oder firmenpolitischen Vorgaben nicht erbracht werden können. Eine Abweichung von Standard ist also nur in einem vorgegebenen Rahmen möglich.

So weit möglich werden alle Teilleistungen, die einem Kunden im Prozessverlauf angeboten werden, an einem Servicepoint gebündelt („alles aus einer Hand“). Verschiedene Servicepoints sind z.B. dann erforderlich, wenn bestimmte Ausstattungen bzw. Geräte notwendig sind, um die Dienstleistung am Kunden zu erbringen (z.B. verschiedene Abteilungen in einem Krankenhaus), der Kunde die Leistung an verschiedenen Orten und Zeitpunkten in Anspruch nehmen will (z.B. während einer Urlaubsreise) oder die Teilleistungen nur von speziellen Organisationseinheiten mit besonderer Sachkenntnis erbracht werden können. Es ist ebenso möglich, dass der gleiche Ort zu verschiedenen Zeitpunkten auch verschiedene Servicepoints repräsentiert, da sich z.B. im fortschreitenden Serviceflow die am Servicepoint zu erbringende Leistung von der zuvor am selben Ort erbrachten Leistung unterscheiden kann.

Servicepoints zeichnen sich zusammengefasst somit hauptsächlich durch folgende Merkmale aus:

- Es sind Bestandteile von Serviceflows, also Prozesselemente.
- Sie repräsentieren zusammenfassbare, standardmäßige Teilleistungen der Gesamtserviceleistung. Vom Standard kann im Rahmen abgewichen werden.
- Sie sind einem Akteur zugeordnet.
- Sie sind mit einem Ort und einem Zeitpunkt assoziiert.
- Es sind Kontaktpunkte zwischen Leistungsnachfrager und Leistungsanbieter, an denen sich der Leistungsnachfrager dem Leistungsanbieter mit seinem Anliegen präsentiert.

Klischewski & Wetzel [KliWet00] definieren den Begriff Serviceflow Management folgendermaßen:

> *„Serviceflow Management organisiert und unterstützt die Durchführung der Leistungserbringung gemäß der situativen Erfordernisse im Verlauf des Serviceflows auf der Basis von erprobten Standardabläufen. Es unterstützt sowohl individuelle Serviceflows als auch ihre parallele Erbringung.“ [KliWet00]*

Unter den Begriff Management subsummiert man nach allgemeiner Auffassung Handlungen wie Organisieren, Planen, Entscheiden, Kontrollieren, Steuern und Führen. Die Ausübung dieser Handlungen im Zusammenhang mit Vorgängen wird beim Einsatz eines Workflow-Management-Systems Workflow-Management (WfM) genannt. (vgl. [JabBöhSch97, Glossar]) Daran angelehnt verwenden Klischewski & Wetzel in dem hier diskutierten Ansatz den Begriff SfM, wobei die eingesetzten Systeme als *Serviceflow Management Systeme (SfMS)* betitelt werden (vgl. [KliWet00]). Genau wie ein Vorgang (vgl.

Abschnitt 2.3.1) zu einem Workflow wird, wenn die zur Leistungserbringung notwendigen Arbeitsschritte mit Hilfe eines WfMS durchgeführt werden (vgl. [JabBöhSch97, S. 68]), wird ein Vorgang zu einem Serviceflow, wenn die Arbeitsschritte durch ein SfMS unterstützt werden. Allerdings fällt auf, dass in der SfM-Definition bis auf Organisation keiner der im Zusammenhang mit WfM aufgeführten Begriffe verwendet wird. Dies hat auch einen guten Grund, denn Sinn und Zweck von SfM bzw. den dazugehörigen SfMS ist nicht die Planung, Entscheidung und Steuerung, sondern „lediglich“ die Unterstützung der Benutzer in diesen Aktivitäten. D.h., diese Management-Aufgaben sollen weitestgehend den Leistungserbringern im Sinne der unterstützenden Sichtweise (vgl. Abschnitt 2.3.1) überlassen werden, denn nur so ist die erforderliche Flexibilität im Serviceprozess sicherzustellen.

Der Begriff SfMS wird im Rahmen dieser Arbeit allerdings nicht im Sinne eines zentralen Systems verstanden, das von allen Prozessteilnehmern gemeinsam genutzt wird, wie es bei WfMS meist der Fall ist, die eine zentrale „Engine“ zur Steuerung der Workflows verwenden (vgl. [JabBöhSch97]). Unter einem SfMS wird ein dezentrales System verstanden, welches an jedem Servicepoint eine Prozessunterstützung nach dem SfM-Ansatz für einen Akteur zur Verfügung stellt. Dieses Verständnis resultiert daraus, dass SfM auf die Unterstützung übergreifender Leistungserstellungsprozesse abzielt. Da verschiedene Organisationen oder auch Abteilungen innerhalb einer Organisation unterschiedlichste IT-Infrastrukturen aufweisen können, sollte keine Organisation dazu gezwungen sein, ein neues System einzuführen oder vorhandene Systeme gegenüber Fremdanwendungen zu öffnen, nur um an einem Serviceflow teilnehmen zu können. Erforderlich ist lediglich, dass ein Mechanismus gefunden wird, die zur Koordination und Kooperation notwendigen Informationen möglichst technologieneutral zwischen den Servicepoints auszutauschen. Wie die zur Prozessunterstützung notwendige Funktionalität am jeweiligen Servicepoint technisch realisiert wird, sollte jedem Prozessteilnehmer selbst überlassen werden. Ein Vorschlag, wie eine solche Umsetzung aussehen könnte, wird im Rahmen dieser Arbeit in Kapitel 3.6 gegeben.

Unter einem SfMS wird also im folgenden eine dezentrale Softwareanwendung verstanden, die an jedem Servicepoint die benötigte Funktionalität zur Verfügung stellt, um dem dort aktiven Akteur die Teilnahme an einem arbeitsteiligen Prozesses nach dem SfM-Ansatz zu ermöglichen. Welche Funktionalitäten an jedem Servicepoint zur Verfügung gestellt werden müssen, wird im nächsten Abschnitt diskutiert.

2.3.3 Anforderungen an Prozessunterstützung nach dem Serviceflow Management Ansatz

In diesem Abschnitt wird ein Anforderungskatalog erstellt, den Softwaresysteme erfüllen sollten, um Akteuren im Kontext arbeitsteiliger Dienstleistungsprozesse, deren Leistungen aufeinander aufbauen, eine adäquate Prozessunterstützung entsprechend der Philosophie des SfM-Ansatzes an einem Servicepoint anbieten zu können. Die dargestellten Anforderungen orientieren sich an den in [KliWet00] erzielten Ergebnissen, die dort ihr Konzept mit dem WfM-Ansatz kontrastieren.

Im Sinne der unterstützenden Sichtweise soll die Verantwortung und Kontrolle der Prozessdurchführung im SfM-Ansatz weitestgehend bei den Dienstleistungsmitarbeitern verbleiben. Ihnen ist an jedem Servicepoint die Möglichkeit anzubieten, den weiteren Prozessverlauf beeinflussen zu können, um so auf veränderte Bedürfnisse des Kunden reagieren zu können (vgl. auch Abschnitt 2.1.2). Das bedeutet, dass sie entweder der vorgedachten Prozesslogik oder der „Logik der Situation“, also einem aktuell modifizierten Prozessplan, folgen können. Hieraus resultiert die Anforderung, dass es möglich sein muss, an jedem Servicepoint eine Bearbeitung des dem jeweiligen Serviceflow zugrundeliegenden vergegenständlichten Prozessmodells zu ermöglichen, und z.B. neue Servicepoints

aufzunehmen, Reihenfolgebeziehungen oder die Geschäftsprozessschnittstelle zu ändern. Dies wird von [KasKliWet01] als *dynamische Modellierung* bezeichnet, im Gegensatz zur *statischen Modellierung*, was sich auf die Erstellung des zugrundeliegenden Prozessmodells bezieht[24]. Eine Veränderung kann hier natürlich nur in einem festgelegten Rahmen erfolgen, d.h. es können z.B. nur Servicepoints hinzugefügt werden, die verfügbar sind. Somit muss das System auch dafür „Sorge tragen", dass dem Benutzer die zur Modifikation zur Verfügung stehenden Optionen zugänglich gemacht werden. Modifizierbarkeit ist nicht nur auf der Ebene des Gesamtprozesses, sondern auch auf der Ebene der einzelnen Prozesselemente, also der Aktivitäten, die an einem Servicepoint durchzuführen sind, notwendig.

Zusätzlich ist es erforderlich, dass sich Veränderungen unmittelbar auf das jeweilige Prozessexemplar auswirken. Dies ist bei traditionellen WfMS wie z.B. *FlowMark* der Firma IBM nicht möglich. Hier machen sich Anpassungen am zugrundeliegenden Prozessmodell erst bei nachfolgend erzeugten Prozessexemplaren bemerkbar. (vgl. [BlaTes99, S. 51])

Anforderung: dynamische Modellierung und unmittelbare Adaptivität

Ein SfMS muss die situative Adaption des Prozessmodells des jeweiligen Prozessexemplars an jedem Servicepoint sowohl auf Gesamt- als auch auf Teilprozessebene erlauben (dynamische Modellierung) und unmittelbar darauf reagieren.

Eine situative Optimierung der Leistungserbringung erfordert die Kenntnis der Prozesshistorie bzw. der bisherigen Beziehung zwischen Dienstleistungsorganisation und Leistungsnehmer (vgl. Abschnitt 2.1.2). Bei Bedarf sollten dem jeweiligen Leistungserbringer entsprechende Informationen vom System zur Verfügung gestellt werden (vgl. [KliWet00]). Hierzu gehören neben Informationen über durchlaufene Servicepoints auch Informationen über dort erbrachte Leistungen, produzierte Ergebnisse (z.B. Dokumente), wer die jeweiligen Leistungen erbracht hat, wie dieser zum Zwecke evtl. Nachfragen erreicht werden kann, usw. Es ist außerdem notwendig, den Teilleistungserbringern das dem jeweiligen Prozessexemplar zugrundeliegende Prozessmodell zugänglich zu machen, so dass sie sich den Prozesszustand, das zu erreichende Ziel und weitere noch zu erledigende Aktivitäten und zu durchlaufende Servicepoints „vor Augen führen" können. Aus diesen Informationen sollte auch eindeutig hervorgehen, welche Teilleistungen an dem jeweiligen Servicepoint zu erbringen sind, damit nachfolgende Leistungsträger sich auf das „Vorhandensein" des jeweiligen Leistungsergebnisses verlassen können. Eine Prozessunterstützung am Servicepoint muss also zu jedem Servicepoint die dort erwarteten Vorleistungen und resultierenden Leistungsergebnisse für jeden Prozessteilnehmer transparent machen, quasi das Geschäftsprozessprotokoll der Leistungsübernahme und -übergabe vermitteln (vgl. Abschnitt 2.3.1). Das System darf allerdings die Prozessausführung nicht verhindern, weil evtl. „versprochene" Leistungen nicht erbracht wurden, denn das würde der geforderten Flexibilität zuwiderlaufen (vgl. [KliWetBah01]). Der Umgang mit solchen Situationen ist (soweit vertretbar und gewollt) in die Verantwortung der Leistungserbringer zu legen. Weiterhin sind bisherige und zu erwartende Abweichungen vom Standardablauf des einzelnen Prozessexemplars und Optionen zur Fortführung des jeweiligen Serviceflows (vgl. oben) zu vermitteln. (vgl. [KliWet00])

[24] Die Möglichkeit zur statischen Modellierung, d.h. die Vorgabe einer (textuellen oder grafischen) Modellierungssprache zur eindeutigen Beschreibung der zugrundeliegenden Prozessmodelle, ist natürlich auch zu fordern. Da dies allerdings nicht im direkten Zusammenhang mit der Benutzung eines SfMS zur Durchführung von Prozessen steht, wird diese Anforderung nicht in den hier konstruierten Anforderungskatalog aufgenommen.

Des weiteren sollte an jedem Servicepoint festgestellt werden können, an welchem Servicepoint sich ein Prozess gerade befindet, so dass z.B. einem Kunden Informationen über den Zustand der Leistungserbringung an jedem Servicepoint vermittelt werden können.

Im Hinblick auf den Zugriff eben aufgeführter Informationen müssen natürlich Zugriffrechte Beachtung finden, denn oftmals ist nur eine beschränkte Transparenz angemessen (vgl. z.B. [Gry96, S. 175]).

Anforderung: Prozesstransparenz[25]

Eine Softwareunterstützung am Servicepoint muss dem Benutzer verschiedenste Informationen über aktuelle Prozessexemplare, unter Beachtung von Zugriffsrechten, transparent machen. Hierzu gehören Informationen, die den Prozessverlauf betreffen (Historie, Zustand, Zukunft, Abweichungen, usw.) als auch Informationen über produzierte Ergebnisse (Informationen, Dokumente, usw.), zu erreichende Ziele, von Prozesselementen zu erwartende Leistungen, zu erbringende Vorleistungen und evtl. nicht erbrachte Leistungen sowie verfügbare Optionen zur Prozessfortführung.

Die Organisation der Zusammenarbeit der Prozessbeteiligten und die Verknüpfung der von den Arbeitsträgern erbrachten Teilleistungen wird sowohl im SfM-Ansatz als auch im WfM-Ansatz dadurch erreicht, dass Prozesse modelliert, instantiiert und dann abgearbeitet werden (vgl. [KliWet00]).

An einem Servicepoint muss die Möglichkeit zur Verfügung gestellt werden, neue Prozesse zu instantiieren. Dies bedeutet im SfM-Ansatz, dass aus Prozessvorlagen (den Mustern; vgl. Abschnitt 2.3.1) neue Prozessexemplare erzeugt werden. Die hierzu benötigten Prozessvorlagen und die Prozessexemplare selbst sind an einem Servicepoint geeignet zu verwalten und den Akteuren zugänglich zu machen. Welche Aktivitäten an einem Servicepoint bereits durchgeführt und welche Servicepoints durchlaufen wurden, ist in den jeweiligen Prozessexemplaren zu dokumentieren, indem die Zustände der Prozessexemplare bzw. der ihnen zugrundeliegenden vergegenständlichten Prozessmodelle entsprechend des Zustandes des real ablaufenden Prozesses fortgeschrieben werden. Ebenso muss es möglich sein, an einem Servicepoint erzeugte Informationen bzw. Dokumente, die ebenfalls den Zustand des Leistungserstellungsprozesses dokumentieren und an Folge-Servicepoints benötigt werden, in die Prozessrepräsentationen einzufügen (bzw. einen Verweis wo diese gelagert werden, falls Dokumente selber nicht direkt transportiert werden sollen).

Nach Abarbeitung aller an einem Servicepoint zu erledigenden Aufgaben muss der Transport der zur Koordination der Aufgabenträger relevanten Informationen und Leistungsergebnisse zum nächsten im Modell des Gesamtprozesses eingetragenen Servicepoint eingeleitet und durchgeführt werden. .

Anforderung: Prozessinstantiierung, –abarbeitung und -verwaltung

Eine Softwareunterstützung am Servicepoint muss die Erzeugung von Prozessexemplaren aus Vorlagen, deren Abarbeitung inklusive Versendung sowie deren Verwaltung unter Beachtung der Vorgaben des zugrundeliegenden Prozessmodells ermöglichen.

An einem Servicepoint können gleichzeitig mehrere Servicemitarbeiter tätig sein, wenn z.B. eine Abteilung mit einem Servicepoint assoziiert ist. Die Angehörigen der Abteilung können sowohl parallel verschiedene Kunden bedienen als auch gleichzeitig nur einen. Eine Softwareunterstützung an einem Servicepoint muss also ermöglichen, dass zur selben Zeit auf die verschiedenen verwalteten Prozessrepräsentationen als auch von verschiedenen Servicemitarbeitern auf eine Prozessrepräsentation zugegriffen wird.

[25] Transparenz wird hier im Sinne von durchsichtig/erkennbar verwendet (vgl. z.B. [Zül98, S. 433]).

Anforderung: Mehrbenutzerfähigkeit

Eine Softwareunterstützung am Servicepoint muss in der Lage sein, unterschiedlichen Akteuren den gleichzeitigen Zugriff auf die verwalteten Prozessexemplare zu ermöglichen. Dies gilt sowohl für den Zugriff auf verschiedene als auch für den konkurrenten Zugriff auf ein Exemplar.

Eine Softwareunterstützung am Servicepoint muss das Anstoßen von Arbeitsabläufen zur Abarbeitung von Hintergrundvorgängen (Supportprozesse) ermöglichen (vgl. [KliWet00]). Supportprozesse können voll- (z.B. Hostanwendung) oder auch nur teilautomatisiert sein (z.B. WfM-Anwendung) bzw. vollständig manuell durch einen (oder mehrere) menschliche Akteur(e) erbracht werden. Erforderlich ist die Integration der entsprechenden Anwendungen bzw. benötigten Kooperationsmittel (sofern eine direkte Interaktion nicht möglich ist) mit dem System. Andererseits sollte das System sich selbst auch in bestehende Arbeitsumgebungen integrieren, so dass gewohnte Arbeitsmittel weiterverwendbar sind, keine Zusatzaufgaben geschaffen und Medienbrüche vermieden werden (vgl. [Gry96, S. 175]). Dadurch wird die Benutzerakzeptanz erhöht und zusätzlich die Investitionssicherung des Unternehmens gewährleistet (vgl. [BlaTes99, S. 55]).

Anforderung: Integration

Eine Softwareunterstützung am Servicepoint muss sich in die vorhandene Arbeitsumgebung einfügen, indem es von anderen Anwendungen benutzt werden kann bzw. vorhandene Anwendungsfunktionalitäten selbst verwendet.

Eine weitere von Klischewski & Wetzel [KliWet00] genannte Anforderung an eine adäquate Prozessunterstützung bezieht sich auf die Analysierbarkeit der vom SfMS verwalteten Vorgänge zum Zweck des Redesigns. Da sich Kundenwünsche ständig ändern können, resultiert daraus die Notwendigkeit, die zugrundeliegenden vergegenständlichten standardisierten Prozessmodelle den veränderten Kundenwünschen kontinuierlich anzupassen. Im SfM-Ansatz kann jedes Prozessexemplar grundsätzlich eine eigene Variante des grundlegend angenommenen Prozessablaufs darstellen. Somit sind entsprechende Protokollierungs-Mechanismen zur Verfügung zu stellen, so dass Abweichungen nachvollziehbar und auswertbar werden und so die evolutionäre Anpassung und Optimierung der Prozessvorlagen erleichtert wird.

Anforderung: Analysierbarkeit

Ein SfMS muss die Aufzeichnung und Auswertung der von ihm unterstützten Vorgänge ermöglichen.

Wie bereits dargestellt, unterstützen SfMS die Koordination zeitlich und räumlich verteilter Arbeitsgruppen, die unterschiedlichen Organisationen angehören können. Grundlegend für eine Koordination ist der Austausch von Informationen (vgl. 2.3.1). Informationsaustausch gestaltet sich insbesondere in interorganisationalen Bereichen aufgrund der anzutreffenden heterogenen Systemlandschaften jedoch oftmals als problematisch. Die zu überwindenden Schwierigkeiten werden von Merz [Mer99, S. 73] unter den Stichworten *Interoperabilität* und *Kohärenz* beschrieben. Interoperabilität bezieht sich auf die technische Integration der beteiligten Kommunikationsdienste und meint, dass eine gemeinsame Sprache bzw. entsprechende Transformationsprotokolle geschaffen werden müssen (Syntax), damit eine Kommunikation über unterschiedliche Systeme hinweg stattfinden kann. Zusätzlich ist es erforderlich, dass die übertragenen Daten einheitlich interpretiert werden (Semantik), worauf sich der Begriff Kohärenz bezieht. Dies führt zur letzten in diesem Abschnitt an SfMS gestellten Anforderung:

Anforderung: Konnektivität

SfMS müssen den Informationsaustausch der Prozessbeteiligten über heterogene Systemlandschaften hinweg unterstützen. Dabei sind Interoperabilitäts- und Kohärenzprobleme zu überwinden und vor dem Benutzer zu verbergen.

2.3.4 Zusammenfassung und Ausblick

In diesem Kapitel wurden zunächst die Begriffe Prozess und Prozessmodell diskutiert und dargestellt, was in dieser Arbeit unter Prozessunterstützung verstanden wird.

Hiernach wurde der SfM-Ansatz vorgestellt, der darauf abzielt, ein Leitbild für die Entwicklung von Software zur Unterstützung evtl. die Organisationsgrenzen überschreitender arbeitsteiliger Abläufe, deren Teilleistungsergebnisse aufeinander aufbauen, vorzugeben. Abschließend wurde ein Anforderungskatalog erstellt, den Softwaresysteme erfüllen sollten, um Akteuren eine Prozessunterstützung entsprechend des SfM-Ansatzes anbieten zu können. Dieser Anforderungskatalog dient in Kapitel 3.7 zur Evaluation des funktionalen Prototypen für einen webbasierten Servicepoint, der detailliert in Kapitel 3.6 vorgestellt wird.

SfM beachtet also die Besonderheiten des Servicebereichs, die Notwendigkeit der Integration menschlicher Akteure in die Leistungserstellung und des Managements evtl. verschiedener Kontaktstellen zum Kunden und aufeinander aufbauender Teilleistungen auf der Basis von Standardabläufen. Es stellt somit ein passendes Leitbild für die Entwicklung von Software für den Transaktionsdienstleistungsbereich dar. Ob es auch dazu geeignet ist, zur Verlängerung von Transaktionsdienstleistungen direkt zum Kunden eingesetzt zu werden, wird u.a. im folgenden Abschnitt untersucht.

2.4 Self-Service und Prozessportale

Wie bereits in der Einleitung dieser Arbeit dargestellt, gehen Behörden sowie private Unternehmen aktuell unter dem Deckmantel der Kundenorientierung dazu über, die zur Lösung des jeweiligen Kunden- bzw. Bürgerproblems notwendigen Leistungen zu bündeln und als Pakete möglichst über eine einzige Kontaktstelle anzubieten. Hier spielt insbesondere das WWW als Basisplattform zur Realisierung sogenannter *Kundenprozessportale* (vgl. [Öst00]) als Zugangssysteme zu Dienstleistungen eine Rolle, über die sich Kunden selbst bedienen können.

Nachfolgend wird zunächst das dem *Self-Service-Ansatz* zugrundeliegende Konzept und Kriterien dafür vorgestellt, die für die Beurteilung der Eignung einer Dienstleistung für Self-Service vorgeschlagen werden. Anschließend wird auf das Kunden bzw. Bürgerprozessportal-Konzept eingegangen. Anhand des *Bremer-Online-Service* wird ein Beispiel für ein Prozessportal vorgestellt, das speziell darauf ausgerichtet ist, Bürger in ihren Vorgängen zu unterstützen. Das Kapitel schließt mit der Einordnung des Konzeptes Self-Service über Prozessportale in den SfM-Ansatz.

2.4.1 Das Self-Service Konzept

Self-Service stammt vom klassischen Konzept der *Selbstbedienung* ab, ist aber in seiner Bedeutung weiter zu fassen (vgl. [Sau99, S. 36]). Saueressig sieht die Erweiterung vor allem darin, dass der Leistungsnachfrager nun aktiv an der Leistungserstellung teilnimmt, und nicht nur ein vorab produziertes Gut erwirbt. Neben dem Ersatz des Bedienvorgangs durch die Eigenleistung des Kunden, steht beim Self-Service auch die Mitwirkung an der Leistungsproduktion im Vordergrund, was dazu führt, dass der Leistungsnachfrager einen Teil der Dienstleistung selbst erbringt und somit einzelne Aufgaben übernimmt, die sonst vom Leistungsanbieter erledigt werden müssten. Die vom Kunden durchgeführten Aufgaben können dabei vom einfachen Informationsabruf bis hin zur selbstständigen Durchführung ganzer Dienstleistungstransaktionen reichen (vgl. [Sau99, S. 35]). Vor dem Hintergrund dieser Entwicklung spricht man auch vom *„prosumer“* [Sau99, S. 14], was den Umstand betont, dass der Konsument zum Mitproduzenten der Leistung wird. Beispielsweise kann im Rahmen einer automatisierten Antragsstellung die Bearbeitung nur erfolgen, wenn der Nutzer des Self-Service-Systems seine persönlichen Angaben übermittelt (siehe hierzu auch Abschnitt 2.5), also quasi bei der Dienstleistungserstellung mitwirkt. Das Leistungspotential, also die Dienstleistung an sich, wird nach wie vor natürlich vom Dienstleister zur Verfügung gestellt und verbleibt in seinem Verfügungsbereich. Ziel ist es jedoch, den Leistungserstellungsprozess von Absatz bis Leistungsendkombination möglichst weitgehend auf den Kunden „überzuweltzen“ (vgl. Abschnitt 2.1.1). Self-Service ist also als *Mitwirkung bei der Dienstleistungserstellung* zu verstehen.

Als Gegenleistung erhält der Kunde allerdings den Vorteil, dass die von ihm benötigten Leistungen über eine Schnittstelle angeboten und „rund um die Uhr“ abgerufen werden können. Somit braucht er nicht mehr persönlich beim Leistungsanbieter zu erscheinen und muss evtl. auch noch nicht mal mehr wissen, welcher Anbieter die Leistung tatsächlich erbringt. Kann eine Dienstleistung vollständig als Self-Service realisiert werden, so ist der Nutzer des Self-Service-Angebots zusätzlich davon unabhängig, wann der Dienstleister in der Lage ist, sich mit seinem Anliegen zu beschäftigen (vgl. [Bod99, S. 159]). Die Einflussmöglichkeit auf die Leistungserstellung wird erhöht.

Nicht alle Dienstleistungen sind jedoch gleichermaßen dafür geeignet, als Self-Service realisiert zu werden. Als Entscheidungskriterien hierfür können die *Self-Service-Fähigkeit* und der *Self-Service-Grad* herangezogen werden, auf die nachfolgend eingegangen wird.

Unter Self-Service-Fähigkeit wird die prinzipielle Eignung einer Dienstleistung für Self-Service verstanden. Der Self-Service-Grad bestimmt das Ausmaß, in dem eine technikgestützte Self-Service-Konzeption umgesetzt werden kann. (vgl. [Sau99, S. 48]) Der Self-Service-Grad ist dann am höchsten, wenn sich die gesamte Dienstleistung durchgängig durch den Kunden gesteuert abwickeln lässt und der Leistungsanbieter nur noch die Leistungsvorkombination erbringen muss (vgl. Abschnitt 2.1.1).

In [Bod99, S. 24] und [Sau99, S. 49 ff.] werden wesentliche Merkmale genannt, die eine Dienstleistung aufweisen sollte, um im Rahmen einer Self-Service-Anwendung zugänglich gemacht werden zu können. Der Self-Service-Grad ist umso höher, wenn:

- sich die Dienstleistung durch einen hohen Informationsanteil auszeichnet, wodurch die Leistung durchgängig durch automatisierte Informationsverarbeitung- und bereitstellung erbracht werden kann.
- der externe Faktor nicht direkt Objekt der Leistungserstellung ist bzw. dies auch nicht ausdrücklich gewünscht wird, d.h. das Uno-Actu-Prinzip kann aufgehoben werden.
- der Leistungserstellungsprozess weitestgehend standardisiert werden kann, d.h. die einzeln durchzuführenden Prozessschritte für den Leistungsnachfrager vorgegeben und informationstechnisch unterstützt werden können.
- kein spezifisches Know-How der Nutzung der Dienstleistung erforderlich ist, was den potentiellen Nutzerkreis einer Self-Service-Anwendung einschränken würde.

Abbildung 5 illustriert das Prinzip des Self-Service-Ansatzes unter Berücksichtigung der wichtigsten Eigenschaften.

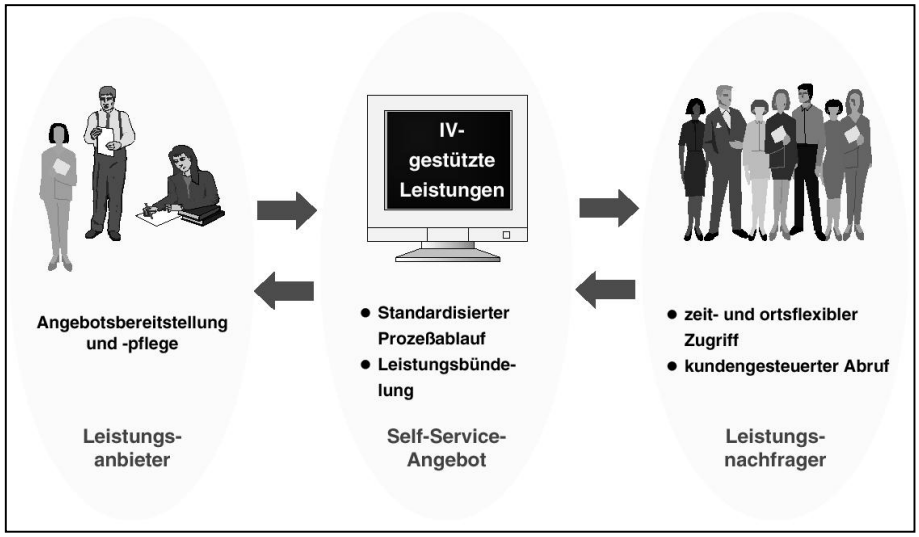

Abbildung 5: Prinzip des Self-Service-Ansatzes (vgl. [Sau99, S. 38])

Aus den vorgestellten Merkmalen wird deutlich, dass nur wenige kundenorientierte Dienstleistungsvorgänge einen sehr hohen Self-Service-Grad aufweisen können. Insbesondere in der Phase der Leistungserstellung selbst sind auf Seiten der Leistungsanbieter meist weiterhin Personen beteiligt, da eine vollständige Automatisierung nicht möglich bzw. erstrebenswert ist. In diesen Fällen reduziert sich der Self-Service-Anteil auf das Anstoßen des jeweiligen Leistungserstellungsprozesses, also die Auswahl der gewünschten Leistung, die Eingabe notwendiger Parameter, dem Nachverfolgen des Vorgangs und dem eventuellen in Empfang nehmen des Leistungsergebnisses, sofern es aus Informationen besteht. Ebenso kann das persönliche Zusammentreffen des Leistungsnachfragers und –anbieters erforderlich sein und der Self-Service nur der Vorbereitung dienen. Die Rolle eines Self-Service-

Zugangssystem kann bei den meisten Transaktionsdienstleitungen also nur in der Ergänzung der Schnittstelle zum Kunden gesehen werden.

Bodendorf [Bod99, S. 26] schlägt aus diesem Grunde vor, zur Personen integrierenden Leistungserbringung im Dienstleistungssektor WfMS einzusetzen, die zum Kunden „verlängert“ werden sollen, um ihn so in die elektronische Vorgangsbearbeitung einbinden zu können. Diese Systeme weisen jedoch den Nachteil auf, dass sie nicht explizit für den Servicebereich entwickelt wurden, sondern auf die Optimierung und Steuerung organisationsinterner Vorgänge abzielen. Kundenorientierung und individualisierte Leistungserbringung sind nur mittelbar mit diesem Ziel verknüpft (vgl. [Kliwet00]).

Somit bietet sich hier eher der Einsatz von SfMS an, da sie auf die Besonderheiten des Servicebereichs ausgerichtet sind (vgl. Abschnitt 2.3.2). Wie der Ansatz Self-Service über Prozessportale in den SfM-Ansatz eingeordnet werden kann, wird nachfolgend nach Vorstellung des Prozessportalkonzeptes untersucht.

2.4.2 Kundenprozess und Prozessportale

Bevor das Prozessportal-Konzept vorgestellt wird, wird in diesem Abschnitt zunächst der Begriff des Kunden- bzw. Bürgerprozesses geklärt, der die konzeptuelle Grundlage für diesen Portaltyp darstellt.

Österle et al. verstehen unter einem Kundenprozess folgendes (vgl. [ÖstFleAlt00]):

> *Der Kundenprozess ist der Prozess, den ein Kunde zur Befriedigung eines Bedürfnisses durchläuft.*

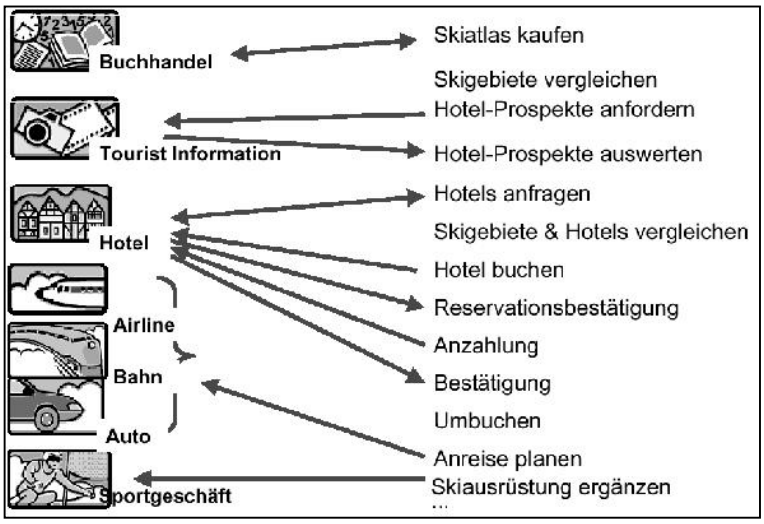

Abbildung 6: Der Kundenprozess „Reiseplanung“ (vgl. [Öst99])

Kunden können hier Endkunden im Sinne von Endverbrauchern als auch andere Unternehmen sein. Ein Beispiel für einen Kundenprozess ist der in Abbildung 6 dargestellte Vorgang der „Reiseplanung“. An diesem Beispiel lässt sich sehr gut ein Problem des Kunden erkennen: Er muss, wenn er sich in einer komplexeren Problemsituation befindet, zur Bedürfnisbefriedigung viele verschiedene Kontakte mit unterschiedlichen Leistungsanbietern eingehen, was evtl. auch impliziert, dass er sich immer wieder auf andere Protokolle[26] einzustellen hat. Außerdem ist der Kunde dazu gezwungen, seinen Prozess selbst zu organisieren, d.h. er muss selbst herausfinden, welche Aktivitäten er durchführen muss, welche Leistungen er hierzu benötigt und wo er sie bekommen kann. Er macht sich sozusagen

[26] bzw. Geschäftsprozessschnittstellen, siehe Abschnitt 2.3.1.

selbst zum Experten in diesem Prozess, ohne tatsächlich einer werden zu können (vgl. [ÖstFleAlt00]).

Diese Problematik trifft insbesondere auch auf Kunden der öffentlichen Verwaltung zu, die zur „Erlangung“ öffentlicher Dienstleistungen aufgrund der hohen Arbeitsteilung meist auf viele verschiedene Zuständigkeiten treffen, um ihr Anliegen bearbeiten zu lassen. Oftmals sind sie gezwungen, die Erfüllung ihres Anliegens selbst zu koordinieren: „Der Aufwand für die Koordination der Arbeitsteilung wird (...) auf den Bürger verlagert.“ [Sch98, S. 18]

Österle et al. [ÖstFleAlt00] beobachten allerdings aktuell den Trend, dass moderne Unternehmen dazu übergehen, ihre Produktion nicht mehr nur auf einzelne Dienstleistungen oder Produkte auszurichten, sondern sich auf die Lösung des gesamten Kundenproblems zu konzentrieren: „The enterprise of the information age focuses on the customer process.“ [ÖstFleAlt00] Sie bieten dem Kunden „aus einer Hand“ jedes Produkt, jede Information und jede Dienstleistung die er braucht. Sie werden zu sogenannten Leistungsintegratoren und Spezialisten für den Kundenprozess, indem sie ihre interne produktzentrierte Sicht durch eine externe Problemlösungssicht ersetzen. Können Teilleistungen nicht selbst produziert werden, so werden diese von anderen, auf diese Teilleistung spezialisierten Anbietern, eingekauft und integriert, d.h. zu einem Gesamtprozess verknüpft und möglichst über eine Kontaktstelle angeboten. Die Koordination des Gesamtprozesses wird von den Leistungsanbietern übernommen.

Zur Bestimmung der Leistungen, die ein Kunde zur Bedürfnisbefriedigung in seinem Prozess benötigt, schlägt Österle die Orientierung am *Customer Ressource Life Cycle* vor. Dieser wird von Ives und Learmonth folgendermaßen definiert:

> *„The Customer Resource Life Cycle is the sequence of tasks for which the customer needs the services of a supplier". [IveLea84]*

Der Ansatz, die im Kundenprozess benötigten Leistungen zu identifizieren und gebündelt möglichst über eine Kontaktstelle anzubieten, wird auch im Kontext von eGovernment verfolgt, wie in Abschnitt 2.2 bereits angedeutet wurde. Diese Philosophie spiegelt also einen allgemeinen Trend in Wirtschaft und Verwaltung wieder.

Zur Ausgestaltung der Kontaktstelle zum Kunden bzw. Bürger werden im Kontext des WWW Prozessportale eingesetzt.

Im allgemeinen werden unter Webportalen[27] Angebote im WWW verstanden, die den Nutzer auf weiterführende Seiten entsprechend seiner Vorlieben lenken. Man kann Portale als Vermittler (sog. *Intermediäre*) auffassen, welche die Nachfrage (eines Nutzers) auf Angebote (im WWW) lenken (vgl. [HeHe99]). Es sind sowohl Einstiegspunkte ins WWW als auch Informationsfilter. Geht es primär nicht um die Bereitstellung von WWW-Adressen und somit der Navigationsunterstützung, sondern um die Zusammenfassung von Inhalten zu einem Angebot, so spricht man auch von *Aggregatoren* (vgl. [HeHe99]).

Grundsätzlich kann zwischen drei Portaltypen unterschieden werden (vgl. [BTZZ00, S. 186]):

- *Vertikale Portale*: Hierbei handelt es sich um Portale, die sich mit speziellen Themen auseinandersetzen. Sie werden oft von Anbietern betrieben, die selbst in der Branche tätig sind oder mit deren Anbietern direkt kooperieren.

[27] I.A. versteht man unter dem Begriff Portal so etwas wie einen Eingang oder Übergang (vgl. [BTZZ00, S. 185]).

- *Horizontale Portale*: Dies sind Portale, die nicht auf ein bestimmtes Thema festgelegt sind. Sie bieten Personalisierungsfunktionen an, die es dem Benutzer erlauben, die Inhalte seiner Startseite nach Belieben selber zusammenzustellen.
- *Unternehmensportale*: Hierbei handelt es sich um Portale, die dem Benutzer über den Browser einen personalisierten Zugang zu Unternehmensinformationen und –anwendungen ermöglichen (vgl. [McH00, S. 6]). Unternehmensportale richten sich im Gegensatz zu den beiden anderen Portaltypen, die auf den Massenmarkt fokussieren, an Angestellte, Kunden und Geschäftspartner einer Organisation. Somit werden hier bei der Personalisierung ergänzend zu den Interessen, auch die Rechte des Besuchers berücksichtigt, die sich aus seiner Position relativ zum Unternehmen ergeben.

Portale können also zusammengefasst als webbasierte, personalisierbare und integrierte Zugangssysteme zu elektronischen Diensten (Information, Kommunikation, Transaktion) verstanden werden (vgl. auch [ALPR01]).

Österle präsentiert in [Öst00] einen weiteren Portaltyp, der als *Kundenprozessportal* bezeichnet wird und sich am vorgestellten Konzept des Kundenprozesses orientiert.

> *„Ein Kundenprozessportal fasst alle Services für einen Kundenprozess auf einer Website zusammen.“[Öst00, S. 6]*

Dieser Portaltyp zeichnet sich dadurch aus, dass eine Menge fachlich abgestimmter Leistungen angeboten werden, die ein Kunde zur Durchführung von Aktivitäten in seinem Kontext bzw. Prozess nutzen kann. Kundenprozessportale stellen somit Aggregatoren im weiter oben angeführten Sinne dar.

Kundenprozessportale sind nicht eindeutig einem der oben unterschiedenen Portaltypen zuzuordnen. Sie können z.B. Bestandteil von Unternehmensportalen sein, und in diesem Fall als spezielle Sicht auf Unternehmensleistungen realisiert werden (vgl. [Öst00, S. 14]). Sie können aber auch den vertikalen Portalen zugeordnet werden, wenn ein Anbieter den Kundenprozess zu seinem Thema macht. Kundenprozessportale sind allerdings keine horizontalen Portale, da dort der eindeutige Themenbezug fehlt.

Das Konzept des Prozessportals ist je nach intendierter Benutzergruppe variabel einsetzbar. Somit wird in Bezug auf den dieser Arbeit zugrundeliegenden Anwendungskontext eGovernment von Bürgerprozessportalen gesprochen, um den Umstand zu betonen, dass Inhalte und Konzeption genau auf die Bedürfnisse der Bürger als Kunden der Verwaltung zugeschnitten werden. D.h. es werden elektronisch zugängliche Leistungen von Verwaltungseinheiten an einer Schnittstelle gebündelt, unter Beachtung der durchzuführenden Prozessschritte eines Bürgers in einer bestimmten Situation. In Anlehnung an die Definition von Österle wird in dieser Arbeit somit folgendes unter einem Bürgerprozessportal verstanden:

> *Ein Bürgerprozessportal fasst alle Services für einen (oder mehrere) Bürgerprozess(e) auf einer Website zusammen.*

Aktuell existieren schon einige Ansätze, eGovernment über Bürgerprozessportale zu betreiben, die mehr oder weniger fortgeschritten sind. Im folgenden Abschnitt wird exemplarisch ein Bürgerprozessportal, namentlich der *Bremer-Online-Service*, vorgestellt.

2.4.3 Beispiel eines Bürgerprozessportals: Der Bremer-Online-Service

Die Leitidee des Bremer-Online-Service[28] ist Kundenorientierung durch Zusammenfassung von Geschäftsprozessen von öffentlichen und privaten Dienstleistern für Bürger nach Lebenslagen zu erreichen (vgl. [KHKS99]). Lebenslagen können als spezifische Problemsituationen oder bestimmte Ereignisse bzw. Vorhaben im Leben eines Bürgers aufgefasst werden, zu dessen Lösung er bestimmte Kombinationen von Leistungen benötigt. Dieses Verständnis entspricht somit grundsätzlich dem vom Kundenprozess (vgl. Abschnitt 2.4.2).

Wie auf Abbildung 7 zu sehen ist, werden für mehrere Lebenslagen, die auf der linken Seite aufgelistet sind, elektronische Dienstleistungen angeboten. Zu jedem dieser Services sind wiederum Teilleistungen verfügbar, die auf der rechten Seite dargestellt sind. In diesem Fall handelt es sich um die Lebenslage „Umzug und Wohnen“ mit den Teilleistungen „Adresse ändern“, „Elektronische Meldebestätigung“, „Nachsendeauftrag erteilen“, usw.

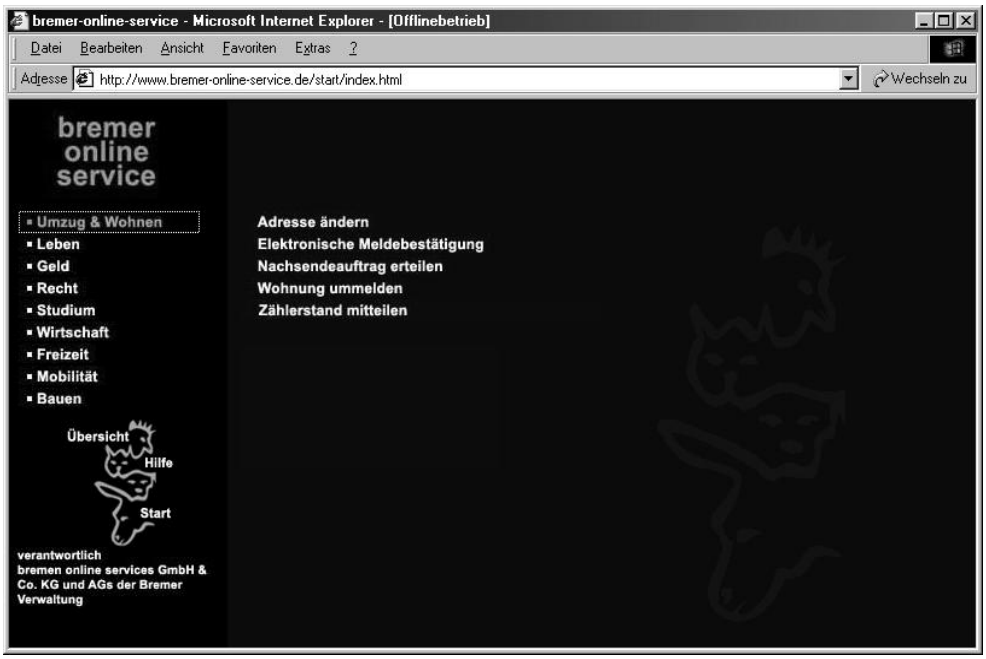

Abbildung 7: Der Bremer-Online-Service

Im Rahmen des Bremer-Online-Services werden zu jeder Teilleistung Informationen darüber vermittelt, wer den konkreten Service erbringt, wie der Dienstleister direkt erreicht werden kann und zu welchen Zeiten dies möglich ist. Man kann unmittelbar auf die Seite des Anbieters weitergehen (Intermediär), die benötigte Dienstleistung aber auch direkt über den Bremer-Online-Service abwickeln. Der Vorteil, den man durch letztere Option hat ist der, dass man evtl. schon einmal angegebene persönliche Informationen wie z.B. die Wohnadresse oder die Bankverbindung nicht noch einmal eingeben muss. Um diese Funktionalität nutzen zu können, ist der Besitz einer Chipkarte[29] notwendig, der die eindeutige Identifikation des Benutzers erlaubt (vgl. Abbildung 8). Nachdem die erforderlichen Informationen an den jeweiligen Dienstleister weitervermittelt wurden, werden dort die entsprechenden Geschäftsprozesse angestoßen und die gewünschte Leistung produziert.

Neben Informationen über angebotene Serviceleistungen und der Möglichkeit, Geschäftsprozesse anzustoßen, soll dem Benutzer in Zukunft auch noch die Option zur Verfügung gestellt werden, den Geschäftsprozesszustand zu beobachten. So ist z.B. für die

[28] http://www.bremer-online-service.de (02.2001)

[29] Zum Konzept der Chipkarte siehe z.B. [Sau99].

Lebenslage „Kauf eines Autos“ geplant, dass sich der Bürger online über den jeweiligen Bearbeitungszustand in der Kfz-Zulassungsstelle informieren kann[30]. Zusätzlich wird der Autobesitzer per email darüber informiert werden, wann und wo die Kfz-Papiere zur Abholung bereitliegen.

Abbildung 8: Informationen zum Prozess „Adressänderung“ des Dienstleisters „Bremer Straßenbahn AG“ über den Bremer-Online-Service.

2.4.4 Self-Service über Prozessportale und Serviceflow Management

In diesem Kapitel wird unter Bezugnahme auf den eben vorgestellten Bremer-Online-Service der Frage nachgegangen, wie Self-Service über Prozessportale in den SfM-Ansatz eingeordnet werden kann.

Diese Frage wird beantwortet, indem das Verhältnis von Kunden- bzw. Bürgerprozessportal zum Konzept Servicepoint untersucht wird. Hierzu werden die Merkmale benutzt, durch die sich ein Servicepoint auszeichnet (vgl. Abschnitt 2.3.2).

An einem Prozessportal werden standardmäßige Teilleistungen der Gesamtserviceleistung zusammengefasst, die sich durch einen hohen Self-Service-Grad auszeichnen, also vom Selbstbediener durchgeführt werden können. Beim Bremer-Online-Service handelt es sich hierbei um die Auswahl und das Starten der innerhalb der Lebenslagen angeboten Dienstleistungsprozesse. Zur Durchführung werden dem Benutzer die hierzu benötigten Hilfsmittel zur Verfügung gestellt.

Vom Standard kann an Prozessportalen ebenfalls nur im Rahmen abgewichen werden. Dies bezieht sich zum einen darauf, dass nur neue Teilleistungen in einen Kundenprozess aufgenommen werden können, die auch verfügbar sind. Zum anderen können z.B. nur die Aktivitäten übersprungen werden, die nicht zwingend ausgeführt werden müssen, damit vom Selbstbediener Leistungen produziert werden, die wiederum von nachfolgenden Dienstleistungserbringern benötigt werden. So können die Dienstleistungsprozesse des Bremer-Online-Service nur angestoßen werden, wenn die für den jeweiligen Bearbeitungsvorgang notwendigen Informationen vorliegen. Andernfalls wird das Anstoßen eines Prozesses verhindert.

[30] Vgl. http://www.bremer-online-service.de/mobil/auto.htm (02.2001)

Einem Prozessportal ist mit dem Selbstbediener ein Akteur zugeordnet, der die dort zu erbringenden Standardteilleistungen durchführt. Die Hilfsmittel hierzu werden vom Portalbetreiber, also einem weiteren Akteur, zur Verfügung gestellt. Dieser ist somit ebenfalls (indirekt) in den Leistungserstellungsprozess involviert, indem er die benötigte Leistungsvorkombination (vgl. Abschnitt 2.1.1) durch zur Verfügung Stellung der benötigten Anwendungen übernimmt. Hierzu stimmt er sich mit den anderen am jeweiligen Prozess beteiligten Akteuren dahingehend ab, welche Leistungen am Prozessportal produziert werden sollen.

Prozessportale repräsentieren einen virtuellen Ort, an dem zu einem bestimmten Zeitpunkt Leistungen vom Selbstbediener erbracht bzw. vom Serviceanbieter abgefordert werden. Servicenehmer und Serviceanbieter treffen hier vollständig mediiert durch IKT aufeinander. Sie ergänzen die Schnittstelle zum Kunden und können im Rahmen eines Serviceflows als Servicepoints, also als Prozesselemente, verstanden werden. Ist ein Prozessportal Bestandteil verschiedener Serviceflows, die hier aggregiert werden, so kann ein Prozessportal auch unterschiedliche Servicepoints repräsentieren bzw. zusammenfassen. Da ein Servicepoint auch immer mit einem Zeitpunkt assoziiert ist, kann ein Prozessportal auch verschiedene Servicepoints innerhalb eines Serviceflows darstellen, die sich durch unterschiedliche Teilleistungen auszeichnen, die an verschiedenen Zeitpunkten erbracht bzw. abgefordert werden und sich aus dem Prozesszustand ergeben[31].

Prozessportale repräsentieren also ein passendes Konzept zur Ausgestaltung eines Zugangssystems zu Transaktionsdienstleistungen für Selbstbediener, die nach dem SfM-Ansatz unterstützt werden. Sie stellen somit einen adäquaten Mechanismus zur „Verlängerung“ der softwaregestützten Vorgangsbearbeitung nach dem SfM-Ansatz zum Kunden dar, ohne zu einem konzeptionellen Bruch zu führen. Komplexe Kundenprozesse, welche sowohl Self-Service fähige Teilleistungen mit ausreichendem Self-Service-Grad und Teilleistungen, auf die dies nicht zutrifft, aufweisen, können folgernd auf Serviceflows mit Self-Service-Anteil abgebildet werden, wobei der Self-Service-Anteil im Rahmen eines Prozessportals zugänglich gemacht wird. Abbildung 9 illustriert diesen Zusammenhang.

Hierbei repräsentiert Organisation C den Prozessportalbetreiber, der die Hilfsmittel zur Verfügung stellt, die es einem Kunden erlaubt, einen Teil der von ihm zur Erledigung seines Kundenprozesses durchzuführenden Aufgaben (Aufgabe 1 und 2) selbständig abzuwickeln. Hierbei könnte am ersten Servicepoint (SP1) dem Kunden z.B. die Möglichkeit angeboten werden, den gesamten Serviceflow anzustoßen und benötigte Vorleistungen zu erbringen, z.B. sein Anliegen zu formulieren und eine für ihn passende Serviceflow-Variante auszusuchen. Anschließend würden Teilleistungen von Organisation B und A erbracht werden, deren Leistungsergebnis an SP4 in Empfang genommen werden können. SP5 erfordert den direkten Kontakt zwischen Kunde und Dienstleister (Organisation B), z.B. in einem Beratungsgespräch, um die dritte Aufgabe zu erledigen.

Organisation C fungiert hier somit als Aggregator (vgl. Abschnitt 2.4.2), der in Kooperation mit den Organisationen A und B ausgehend vom Kundenprozess für die geeignete Verknüpfung der vom Kunden benötigten Teilleistungen zu einem Gesamtprozess sorgt. Die zur Kooperation und Koordination notwendigen Informationen werden zwischen den Prozessteilnehmern ausgetauscht, so dass an jedem Servicepoint die zur Erbringung der jeweiligen Teilleistung notwendigen Unterlagen vorliegen.

Ein konkretes Beispiel für einen Ablauf dieser Art wird im nachfolgenden Kapitel 2.5 anhand des in dieser Arbeit verwendeten Anwendungsbeispiels gegeben.

---

[31] Das in dieser Arbeit verwendete Anwendungsbeispiel zeichnet sich durch eine solche Konstellation aus (vgl. Abschnitt 2.5).

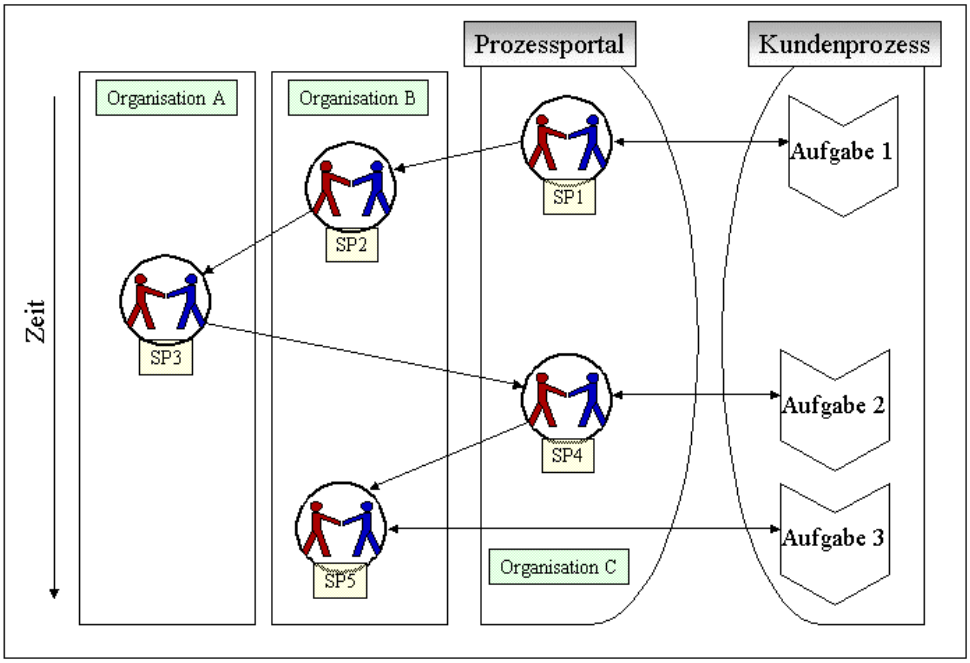

Abbildung 9: Serviceflow mit Self-Service-Anteil im Rahmen eines Kundenprozessportals

2.4.5 Zusammenfassung

In diesem Kapitel wurde zunächst das Konzept Self-Service vorgestellt. Unter Self-Service wird die Mitwirkung bei der Dienstleistungserstellung verstanden, d.h. der Kunde übernimmt Aufgaben, die sonst vom Leistungsanbieter ausgeführt werden müssten. Dadurch entstehen für den Self-Service-Nutzer jedoch Vorteile in dem Sinne, dass er z.B. selbst entscheiden kann, wann die Dienstleistung zu ihm gelangt und er alle benötigten Leistungen über eine Schnittstelle abrufen kann, ohne den konkreten Leistungserbringer kennen zu müssen.

Nicht jede Dienstleistung ist allerdings als Self-Service realisierbar. Sie sollte spezifische Merkmale aufweisen, die unter den Stichworten Self-Service-Fähigkeit und Self-Service-Grad vorgestellt wurden. Hierdurch konnte verdeutlicht werden, dass nur sehr wenige Dienstleistungen sich für eine durchgängige selbständige Abwicklung durch den Kunden eignen, was insbesondere für Transaktionsdienstleistungen gilt.

Anschließend wurde das Kundenprozessportal-Konzept als mögliche Ausgestaltungsform von Self-Service-Zugangssystemen im WWW vorgestellt. Dieser Portaltyp zeichnet sich durch Orientierung am Kundenprozess aus und fasst alle Services zusammen, die vom Kunden zur Erledigung seiner Aufgaben und somit zur Bedürfnisbefriedigung benötigt werden. Je nach intendierter Benutzergruppe kann im Kontext von eGovernment auch von Bürgerprozessportalen gesprochen werden, die dem „One-Stop“-Gedanken (vgl. Abschnitt 2.2) entsprechen.

Anschließend wurde mit dem Bremer-Online-Service ein Beispiel-Bürgerprozessportal vorgestellt, dass Geschäftsprozesse von öffentlichen und privaten Dienstleistern nach Lebenslagen zusammenfasst und über diese Schnittstelle zugänglich macht. Lebenslagen entsprechen grundsätzlich dem Verständnis vom Kundenprozess.

Zum Abschluss dieses Kapitels wurde der Ansatz Self-Service über Prozessportale in den SfM-Ansatz eingeordnet. Über ein Prozessportal können die standardmäßig auszuführenden Teilleistungen eines Serviceflows direkt zugänglich gemacht werden, die sich durch einen ausreichend hohen Self-Service-Grad auszeichnen.

2.5 Das Anwendungsbeispiel: Elektronische Beantragung von Briefwahlunterlagen

Bei dem dieser Arbeit zugrunde liegenden Anwendungsbeispiel handelt es sich um die elektronische Beantragung von Briefwahlunterlagen über das Bürgerprozessportal www.hamburg.de. Das Anwendungsbeispiel wurde in einem Projektseminar, dass im Wintersemester 2000/2001 am Arbeitsbereich Softwaretechnik des Fachbereichs Informatik der Universität Hamburg stattfand, grundlegend analysiert und modelliert. Der Titel der Lehrveranstaltung lautete[32] „Service als Leitbild: Softwareunterstützung für Dienstleister." Die Projektverantwortlichen waren Ralf Klischewski und Ingrid Wetzel.

Auf Basis dieses Anwendungsbeispiels ist auch der in dieser Arbeit beschriebene funktionale Prototyp für einen webbasierten Servicepoint entstanden, der im Kapitel 3.6 vorgestellt wird. Auf dessen Konstruktionsprozess und die Vorraussetzungen und Ziele, die seine Entwicklung im Rahmen des Seminars beeinflusst haben, wird in Abschnitt 3.1 näher eingegangen. An dieser Stelle wird zunächst das Anwendungsbeispiel selbst beschrieben, zu dessen Unterstützung der funktionale Prototyp eingesetzt worden ist.

Das Anwendungsbeispiel wurde von der Stadt Hamburg bzw. dem Senatsamt für Bezirksangelegenheiten (SfB) vorgegeben, mit dem im Rahmen des Seminars kooperiert wurde. Vom SfB wurde das Ziel verfolgt, den Bürgern der Stadt Hamburg zur Wahl 2001 die Möglichkeit anzubieten, einen Antrag auf Briefwahlunterlagen auch über das WWW[33] stellen zu können[34].

Zur Bestimmung des Ist-Zustandes der Stellung eines Antrags auf Briefwahlunterlagen und deren Bearbeitung wurde ein Interview mit zwei Verantwortlichen des SfB durchgeführt[35]. Auf Basis der Ist-Situation, wurde eine Soll-Konzeption entwickelt, die nachfolgend anhand eines Serviceflow Modells und eines Soll-Szenarios[36], das die antizipierte Nutzung des Self-Service beschreibt, dargestellt wird. Die Soll-Konzeption wurde mit dem SfB rückgekoppelt.

Unter Verwendung der in Abschnitt 2.4.1 genannten Merkmale, wird zusätzlich die Eignung des Anwendungsbeispiels zur Umsetzung als Serviceflow mit Self-Service-Anteil begründet.

2.5.1 Soll-Prozess des elektronischen Briefwahlantrags

Abbildung 10 stellt den Soll-Prozess für den elektronischen Briefwahlantrag dar, der als Folge von Servicepoints im Sinne eines Serviceflows modelliert ist[37]. Reihenfolgebeziehungen werden durch Pfeile angedeutet, Supportprozesse werden als Kästen dargestellt.

Zunächst erstellt der Bürger einen Briefwahlantrag im Rahmen des Bürgerportals www.hamburg.de (siehe Abschnitt 2.5.2). Der Antrag wird dann an das SfB gesandt. Dort wird zunächst automatisch geprüft, ob alle benötigten Informationen korrekt eingetragen sind und ob der Antragsteller sich im Wählerverzeichnis befindet, was ihn zur Teilnahme an der Wahl berechtigt. Anschließend wird die für den Wähler zuständige Wahldienststelle ermittelt

---

[32] LV-Nr.: HPJS 18.363

[33] Über die Plattform http://www.hamburg.de

[34] Die Ausführungen zum Anwendungsbeispiel beziehen sich auf den Stand Dezember 2000. Veränderungen, die sich z.B. für den Ablauf der Briefwahlbeantragung nach diesem Zeitpunkt ergeben haben, werden in dieser Arbeit nicht berücksichtigt, da dem Autor keine weiteren Informationen hierüber vorliegen.

[35] Die Interviewer waren: Ralf Klischewski, Jürgen Köster, Ulrike Najmi, Wolfgang Riese und der Autor.

[36] Zum Begriff des Szenarios siehe z.B. [Zül98].

[37] Auf Serviceflow Modellierung wird detaillierter in Abschnitt 3.3 eingegangen.

und ein Eintrag in der Bearbeitungsliste der Wahldienststelle erzeugt. Zum Schluss erfolgt die Generierung einer Rückmeldung, die an hamburg.de zu übertragen ist, damit dort dann eine email an den Antragsteller mit Informationen über die erfolgreiche Bearbeitung und die zuständige Wahldienststelle erzeugt werden kann (siehe Abschnitt 2.5.2). Tritt ein Fehler auf, z.B. ist der Wähler nicht im Wählerverzeichnis, so nimmt sich ein Sachbearbeiter dem Fall an und kümmert sich um eine sinnvolle Fehlerbehandlung. Wenn der Antragsteller z.B. seine email-Adresse für Rückfragen angegeben hat, dann schreibt der Sachbearbeiter eine Nachricht an hamburg.de, so dass dort dann eine entsprechende Mail an den Antragsteller verschickt werden kann.

Anschließend (ab drei Wochen vor der Hamburg-Wahl; sieben Wochen vor Europa-Wahlen) beginnt die Arbeit in den Wahldienststellen. Dort greifen Sachbearbeiter über das DiWa[38] auf die Briefwahlanträge zu. Sie prüfen zunächst noch einmal, ob der Antragsteller im Wählerverzeichnis geführt wird und dort kein Sperrvermerk[39] existiert. Entspricht alles den Vorschriften, werden die Wahlunterlagen erstellt, ein Vermerk im DiWa gemacht und der Versand eingeleitet. Hierbei wird beachtet, ob die Wahlunterlagen an die Meldeadresse des Antragstellers geschickt werden sollen, oder an eine zusätzlich angegebene Zustelladresse (vgl. Abschnitt 2.5.2). Zusätzlich wird eine elektronische Rückmeldung aufgrund des Eintrags im DiWa generiert, die über das Portal an den Bürger übertragen wird und ihm mitteilt, dass seine Wahlunterlagen nun ausgestellt sind und sich auf dem Postweg zu ihm befinden. Bei Problemen und Sonderfällen wird der Wahldienststellenleiter hinzugezogen. Er entscheidet, ob z.B. trotz Sperrvermerk erneut Wahlunterlagen ausgestellt werden oder statt der Wahlunterlagen eine Benachrichtigung verschickt wird, die dem Antragsteller mitteilt, dass er nicht wahlberechtigt ist.

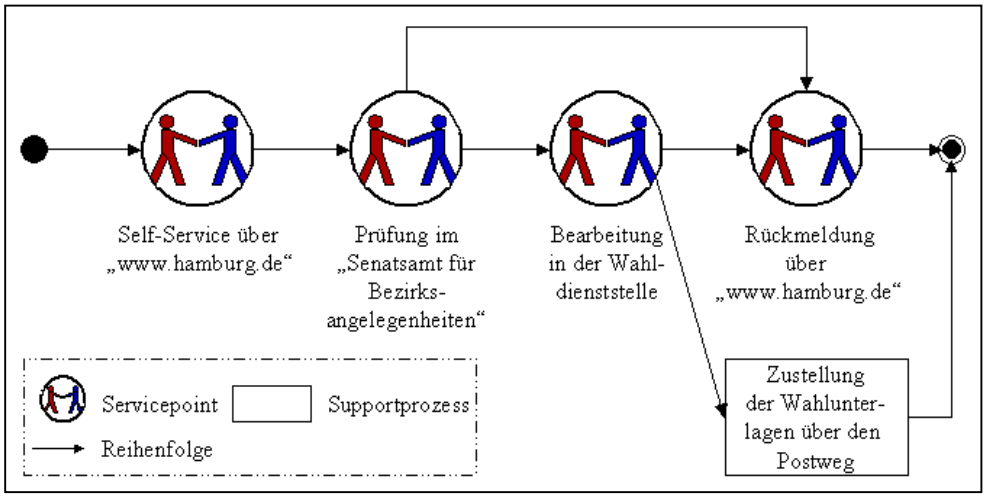

Abbildung 10: Serviceflow Modell: „Elektronische Beantragung von Briefwahlunterlagen“

Wesentlich im Hinblick auf eine Computerunterstützung des gesamten Prozesses ist, dass es kein gemeinsames System geben kann. Das von der Behörde verwendete Informationssystem ist als „geschlossenes“ zu verstehen, d.h. der Zugriff von außen ist nicht gestattet.

Die Eignung des Anwendungsbeispiels als Serviceflow mit Self-Service-Anteil, kann anhand der in Abschnitt 2.4.1 aufgezählten Merkmale motiviert werden.

[38] DiWa steht für Dialogverfahren Wahlen und bezeichnet eine Software, welche die Sachbearbeiter in den Wahldienststellen unterstützt.

[39] Kennzeichnet, das für den Wähler schon einmal Briefwahlunterlagen ausgestellt wurden.

Die Dienstleistung zeichnet sich durch einen hohen Informationsanteil aus. Um Briefwahlunterlagen erhalten zu können, muss der Antragsteller seine persönlichen Daten der Behörde mitteilen. Diese Informationen werden dann zur Berechtigungsüberprüfung und Ausstellung der Briefwahlunterlagen verwendet. Eine vollständige Realisierung durch automatische Informationsverarbeitung und –bereitstellung ist allerdings nicht möglich, da die Briefwahlunterlagen weiterhin „per Hand“ erstellt werden müssen und Sonderfälle auftreten können, die z.B. vom Wahldienststellenleiter behandelt werden. Da zur Leistungserbringung allerdings nicht der persönliche Kontakt mit dem externen Faktor erforderlich ist, erfüllt der Briefwahlantrag das zweite Merkmal vollständig: Der Kunde oder ein in seinem Besitz befindliches materielles Produkt ist nicht direkt Objekt der Leistungserstellung. Das Anliegen des Bürgers wird den Dienstleistungserbringern elektronisch präsentiert, das Uno-Actu-Prinzip kann somit außer Kraft gesetzt werden.

Die Übergabe der erforderlichen Informationen, die Ausstellung der Briefwahlunterlagen und deren Auslieferung erfolgt bei jedem Briefwahlantrag nach dem gleichen Schema. Der Leistungserstellungsprozess kann also zu weiten Teilen standardisiert werden. Allerdings kann es zu einigen Ausnahmesituationen kommen, die das Reagieren auf individualisierte Bedürfnisse auf Leistungsanbieterseite erforderlich macht. Beispielsweise kann der Antragssteller umziehen, nachdem er seinen Antrag gestellt hat und bevor die Arbeit in den Wahldienststellen beginnt. Hierdurch entsteht eine sich aus der persönlichen Situation des Antragstellers ergebende Inkonsistenz, die vom Sachbearbeiter situativ berücksichtigt und evtl. abweichend vom Standard behandelt werden muss. Weitere Beispiele wären, dass der Antragsteller seine schon ausgestellten Wahlunterlagen verloren hat und sie erneut beantragt oder am Wahltag persönlich im Wahlbüro erscheint und dort wählen möchte, was ebenfalls eine Sonderbehandlung erfordert.

Ebenso ist kein spezifisches Know-how zur Nutzung der Dienstleistung erforderlich. Der Leistungsempfänger muss nur seine persönlichen Daten kennen und wissen, wie er sie dem Dienstleister mitteilen kann. Er braucht im Prinzip auch noch nicht einmal zu wissen, ob er wahlberechtigt ist oder nicht, denn das wird vom Dienstleister überprüft und gegebenenfalls dem Antragsteller mitgeteilt.

Das Anwendungsbeispiel zeichnet sich also durch Self-Service-Fähigkeit aus (vgl. Abschnitt 2.4.1). Der Self-Service-Grad ist jedoch gering, da eine durchgängige Abwicklung durch den Kunden nicht möglich ist. Die Briefwahlunterlagen müssen von Mitarbeitern der Behörde ausgestellt werden und Fälle, die von Standardablauf abweichen, sind von speziellen Dienstleistungsmitarbeitern zu bearbeiten. Eine passende Computerunterstützung erfordert also eine Variante, die Mischformen zwischen der automatisierten und der manuellen Dienstleistungserstellung ermöglicht (vgl. [Sau99, S. 58]).

Der Einsatz des Leitbildes SfM zur Entwicklung einer Computerunterstützung für das Anwendungsbeispiel rechtfertigt sich dadurch, dass dem Beispiel ein organisationsübergreifender Ablauf zugrunde liegt (Portalbetreiber von www.hamburg.de, SfB, Wahldienststelle), die Zusammenarbeit somit zeit-, orts- und teamübergreifend stattfindet, die Teilleistungen aufeinander aufbauen und an den einzelnen Servicepoints situativ auf Besonderheiten, die sich aus der persönlichen Situation des Antragstellers ergeben, reagiert werden muss. Der Ablauf ist durch ein Muster beschreibbar, das den Standard vorgibt und als Basis zur Kooperation und Koordination der beteiligten Akteure dienen kann.

2.5.2 Unterstützung des Vorgangs der Antragsstellung von Briefwahlunterlagen aus Bürgersicht

In diesem Abschnitt wird anhand eines Soll-Szenarios die antizipierte Nutzung des Self-Servicepoints im Rahmen des Bürgerprozessportals www.hamburg.de vorgestellt. Dieses Szenario wurde ebenfalls im Rahmen des oben genannten Projektseminars vom Autor erstellt. Bei der hier abgedruckten Fassung handelt es sich um eine gekürzte Version, die so weit wie möglich auf die für die weitere Untersuchung relevanten Aspekte beschränkt ist. Es beschreibt die Folge von Aktivitäten, die ein Bürger in Interaktion mit dem zu entwickelnden System durchzuführen hat, um den eben beschriebenen Gesamtprozess der Erstellung von Briefwahlunterlagen über diese Plattform anzustoßen. Dieses Szenario diente als Vorlage zur Entwicklung der Benutzungsoberfläche des im Abschnitt 3.6 vorgestellten Prototypen.

„[...] Zunächst wird Herr Meier eine Spiegelstrichliste präsentiert, welche die Vorraussetzungen benennt, die zu erfüllen sind, damit man an der Hamburg-Wahl teilnehmen kann. Herr Meier stellt fest, dass er alle Anforderungen erfüllt. Als nächstes müsste Herr Meier seinen Namen und seine Meldeadresse (Straße, Hausnummer, PLZ) eintragen. Da jedoch diese Daten aufgrund des in myhamburg.de[40] hinterlegten persönlichen Profils von Herrn Meier bekannt sind, wurden die Informationen schon automatisch in das Formular eingetragen. Herr Meier wird zusätzlich darauf hingewiesen, dass die Wahlunterlagen standardmäßig an diese Adresse geschickt werden. Falls er dies nicht wünscht, wird er dazu aufgefordert, weiter unten im Formular eine andere Zustelladresse anzugeben.

Nachfolgend trägt er seine Wählerverzeichnisnummer (Nr.-WVZ)[41] in das vorgesehene Formularfeld ein, die er der Vorderseite seiner Wahlbenachrichtigungskarte[42] entnimmt. Er muss diese Nummer zwar nicht angeben, es wird ihm jedoch empfohlen, da so eine schnellere und fehlerfreie Bearbeitung möglich ist. Herr Meier könnte auch auf die Angabe seines Namens und seiner Adresse verzichten, und nur die Wählerverzeichnisnummer angeben, wodurch allerdings die gleichen Probleme bei der Bearbeitung entstehen könnten. Es ist also erforderlich, dass mindestens eine der beiden Informationen in dem Formular vorhanden sind. [...]

Da Herr Meier einige Zeit in München verbringen wird, gibt er seine dortige Adresse als Zustelladresse an. [...]

Das Feld für die email-Adresse wurde schon automatisch ausgefüllt. Hierzu findet Herr Meier noch den Hinweis, dass die Angabe seiner email-Adresse wichtig ist, wenn er darüber informiert werden möchte, ob sein elektronischer Briefwahlantrag erfolgreich an die zuständigen Stellen weitergeleitet werden konnte. Zusätzlich kann er so eine Nachricht darüber erhalten, dass seine Briefwahlunterlagen ausgestellt wurden und sich auf dem Postwege zu ihm befinden. Falls er dies nicht möchte, wird Herr Meier dazu aufgefordert, das Feld wieder zu löschen. Herr Meier lässt das Feld unverändert.

Nach Drücken auf den Absenden-Button erscheint eine Dank-Seite mit dem Hinweis, dass die Wahllokale drei Wochen vor der Wahl öffnen und sein Antrag ab dann bearbeitet werden wird. Seine Wahlunterlagen wird er einige Tage später auf dem Postwege zugesendet bekommen. Weiterhin wird Herrn Meier mitgeteilt, dass sein Antrag zunächst an das SfB

---

[40] www.hamburg.de wird Personalisierungsfunktionen anbieten, die auch in diesem Kontext Verwendung finden sollen

[41] Eine Wählerverzeichnisnummer hat die Form xxx yy zzzz, dabei ist xxx die Schlüsselnummer für den Ortsteil, yy die Schlüsselnummer für den Wahlbezirk, zzzz die lfd. Nummer des Wählers im Wählerverzeichnis.

[42] Wird den wahlberechtigten Bürgern zugesandt und berechtigt sie zur Teilnahme an der entsprechenden Wahl bzw. zur Beantragung von Briefwahlunterlagen

weitergeleitet wurde und von dort dann im nächsten Schritt an die für ihn zuständige Wahldienststelle gelangt. Er wird am nächsten Tag eine Mail bekommen, die ihn darüber informiert, ob der Antrag erfolgreich weitergeleitet werden konnte und welche Wahldienststelle für ihn zuständig ist. Außerdem wird ihm eine Ansprechperson mitgeteilt werden, die im Fehlerfall oder bei anderen Fragen kontaktiert werden kann. Anschließend wird Herr Meier darauf hingewiesen, dass eine Kopie seines Antrags im System zugriffssicher gespeichert wurde. Zum Zeitpunkt der letzten email-Versendung wird die Kopie dann wieder gelöscht werden. [...]”

2.5.3 Zusammenfassung

In diesem Abschnitt wurde das in dieser Arbeit verwendete Anwendungsbeispiel für einen eGovernment-Service vorgestellt. Anhand seiner Charakteristika wurde die Eignung des Anwendungsbeispiels zur Umsetzung als Serviceflow mit Self-Service-Anteil begründet. Anschließend ist mittels eines Soll-Szenarios die antizipierte Nutzung des Service über das Portal www.hamburg.de beschrieben worden. Auf Basis dieses Szenarios wurde die Benutzungsoberfläche des funktionalen Prototypen, der in Abschnitt 3.6 vorgestellt wird, im Rahmen des ebenfalls in diesem Abschnitt vorgestellten Projektseminars entworfen.

2.6 Zusammenfassung und Ausblick

Ziel dieser Arbeit ist die Beantwortung der Fragestellung, wie eine Softwarearchitektur für webbasierte Servicepoints gestaltet sein sollte, auf deren Basis eine Prozessunterstützung für eGovernment-Services unter Verwendung der Technologien Java und XML realisiert werden kann. In Kapitel 2 wurde die Fragestellung unter anwendungsfachlichen Gesichtspunkten untersucht und grundlegende Begriffe geklärt, die mit dem fokussierten Anwendungsbereich in Verbindung stehen.

Zunächst wurde der Dienstleistungsbegriff aus verschiedenen Perspektiven betrachtet und die charakteristischen Merkmale herausgearbeitet, durch die sich (öffentliche) Dienstleistungen auszeichnen. Es wurde betont, dass im Hinblick auf die Qualitätssteigerung von Dienstleistungen neben Schnelligkeit und Effizienz insbesondere die situative Anpassbarkeit der Leistungserstellung auch nach der Leistungsvereinbarung notwendig ist.

Anschließend wurde der Begriff eGovernment diskutiert und dargestellt, dass es sich bei eGovernment-Services um elektronisch zugängliche Verwaltungsleistungen (Information, Kommunikation, Transaktion) handelt, die z.B. über Webportale vom Bürger direkt zugegriffen werden können. Dahinter steht die Philosophie des „One-Stop-Services“, also die Bündelung von zusammengehörigen Leistungen an einer Kontaktstelle, um die Verwaltung bürgerfreundlicher und effizienter zu gestalten.

Hiernach wurde dargestellt, wie der Begriff Prozessunterstützung in dieser Arbeit in Bezug auf die Unterstützung routinisierter arbeitsteiliger Prozesse mit Hilfe von IKT grundsätzlich interpretiert wird. Wesentlich ist, dass die Kontrolle über den Arbeitsprozess so weit wie möglich „in den Händen“ der Prozessbeteiligten verbleibt, so dass situativ auf unvorhergesehene Anforderungen im Prozessverlauf reagiert werden kann.

Diese Anforderung wird vom anschließend vorgestellten SfM-Ansatz berücksichtigt, der darauf abzielt, im Kontext arbeitsteiliger Dienstleistungsprozesse, die orts-, zeit und –teamübergreifend ablaufen und deren Teilleistungen aufeinander aufbauen, zur Entwicklung von Anwendungssoftware eingesetzt zu werden. Unter Berücksichtigung der grundlegenden Konzepte dieses Leitbildes wurden dann Anforderungen abgeleitet, denen eine diesem Ansatz folgende Softwareunterstützung gerecht werden muss.

Hiernach wurde das Konzept Self-Service über Prozessportale vorgestellt. Self-Service ist als Mitwirkung bei der Dienstleistungserstellung zu verstehen, d.h. der Selbstbediener übernimmt Aufgaben, die sonst vom Leistungsanbieter erledigt werden müssten. Nur sehr wenige, insbesondere Transaktionsdienstleistungen, sind dafür geeignet, vollständig als Self-Service mit Hilfe von IKT umgesetzt zu werden, was durch Darstellung der beiden Konzepte Self-Service-Fähigkeit und –Grad verdeutlicht werden konnte. Es wurde gezeigt, dass der Ansatz Self-Service über Prozessportale dazu geeignet ist, arbeitsteilige Prozesse nach dem SfM-Ansatz ohne konzeptionellen Bruch zum Kunden zu „verlängern“.

Als Beispiel für einen Serviceflow mit Self-Service-Anteil aus dem eGovernment-Bereich wurde die elektronische Beantragung von Briefwahlunterlagen über das Portal www.hamburg.de vorgestellt.

Ziel des nächsten Kapitels ist es, auf Basis der in diesem Kapitel herausgearbeiteten Anforderungen, die von einem SfMS beachtet werden müssen, zu ermitteln, wie eine geeignete Softwarearchitektur für webbasierte Servicepoints auf Basis der gewählten Technologien gestaltet sein könnte.

3 Prozessunterstützung mit Java und XML

In diesem Kapitel wird die Fragestellung nach der Softwarearchitektur eines webbasierten Servicepoints vornehmlich aus technischer Sicht beantwortet. Die wesentliche Grundlage zur Beantwortung dieser Fragestellung stellt ein funktionaler Prototyp dar, der im Rahmen des in Abschnitt 2.5 bereits angesprochenen Projektseminars auf Basis der Technologien Java 2 Standard Edition, XML und StoryServer 5.0 entstanden ist.

Um zu vermitteln, warum der funktionale Prototyp so realisiert worden ist, wie er in dieser Arbeit präsentiert wird, wird zunächst ein Überblick über den Projektverlauf gegeben, in dessen Kontext der Prototyp entstanden ist. Hierbei wird auf die vorgefundene Ausgangslage, das gewählte Vorgehen, getroffene Entwurfsentscheidungen und die Ziele, die bei seiner Entwicklung im Vordergrund standen, eingegangen.

Bei der Konstruktion eines webbasierten Servicepoints sind neben allgemeinen softwaretechnischen Entwurfsprinzipien auch die Besonderheiten zu beachten, die sich aus diesem Einsatzkontext ergeben. Kapitel 3.2 geht auf diesen Aspekt näher ein, indem neben allgemein etablierten Entwurfsprinzipien die wesentlichen Eigenschaften webbasierter Anwendungen herausgearbeitet werden. Zusätzlich wird die Entwurfsmetapher Fachlicher Service vorgestellt, da sie die Grobstrukturierung der Softwarearchitektur des funktionalen Prototypen angeleitet hat.

Das Leitbild SfM (siehe Abschnitt 2.3.2) und die für diesem Ansatz charakteristischen Modellierungskonzepte sowie die verwendeten Entwurfsmetaphern für Prozessrepräsentationen bildeten die Grundlage dafür, was auf technischer Ebene durch den funktionalen Prototypen umgesetzt wurde. Abschnitt 3.3 stellt zunächst die bei der Serviceflow Modellierung benutzen Techniken vor. Anschließend werden die Umgangsformen[43] präsentiert, die an den softwaretechnischen Prozessrepräsentationen zu implementieren waren.

Hiernach wird in Abschnitt 3.4 und 3.5 auf die zur Realisierung des Prototypen verwendeten Technologien Java, XML und StoryServer 5.0 sowie deren Verbindung über das Produkt TclBlend eingegangen. Kapitel 3.6 stellt die Konstruktion des funktionalen Prototypen vor.

In Kapitel 3.7 wird der funktionale Prototyp im Hinblick auf die Erfüllung des in Abschnitt 2.3.3 erstellten Anforderungskatalogs evaluiert. Zusätzlich wird geprüft, ob neben den allgemeinen softwaretechnische Entwurfskriterien auch die Besonderheiten webbasierter Anwendungen ausreichend berücksichtigt wurden. Die Ergebnisse der Evaluation werden dazu genutzt, eine allgemeine Empfehlung dahingehend abzugeben, wie eine Softwarearchitektur für webbasierte Servicepoints sowohl im Hinblick auf ihre Grobstrukturierung als auch in Bezug auf deren Ausgestaltung konstruiert werden sollte und welche Technologien am besten einzusetzen sind.

[43] „Eine Umgangsform ist eine charakteristische Handlung an oder mit einem Gegenstand.“ [Zül98, S. 150]

3.1 Projektkontext: Entwicklung eines webbasierten Servicepoints

In diesem Abschnitt wird ein Überblick darüber gegeben, welche konzeptionellen und technischen Rahmenbedingungen die Entwicklung des funktionalen Prototypen beeinflusst haben, der größtenteils im Rahmen des bereits in Abschnitt 2.5 angesprochenen Projektseminars entstanden ist. Hierbei wird auch auf den Entwicklungsverlauf und die während des Konstruktionsprozesses gefällten Entwurfsentscheidungen eingegangen. Dieser Abschnitt dient dazu, dem Leser ein Verständnis darüber zu vermitteln, warum der Prototyp so gestaltet worden ist, wie er in Kapitel 3.6 präsentiert wird und welche Ziele hierbei im Vordergrund standen. Auf die angesprochenen Punkte wird detaillierter in den nachfolgenden Kapiteln eingegangen.

3.1.1 Projektvorgaben und Grobstruktur des webbasierten Servicepoints

Die wesentlichste Grundannahme, die der Entwicklung des Prototypen zugrundegelegt wurde, ist die, dass zur informationstechnischen Unterstützung arbeitsteiliger Prozesse nach dem SfM-Ansatz kein gemeinsames System aller Prozessbeteiligten vorrausgesetzt werden kann (vgl. Abschnitt 2.3.2). Diese Annahme wurde auch durch das Anwendungsbeispiel (vgl. Abschnitt 2.5.1) bestätigt, bei dem ein direkter Zugriff auf die Systeme des SfB von www.hamburg.de zum einen aus sicherheitstechnischen Gründen nicht in Betracht gezogen werden konnte. Zum anderen verwenden die beiden teilnehmenden Organisationen grundsätzlich verschiedene IT-Infrastrukturen, die unter dem Aspekt der Investitionssicherheit auch weiterhin verwendet werden sollen. Hierbei handelt es sich bei www.hamburg.de um eine webbasierte Umgebung, die auf dem Web Content Management System StoryServer 5.0 aufsetzt, die Behörde verwendet Großrechner.

Somit wurde im Rahmen des Seminars entschieden, dass eine Softwareunterstützung lediglich für einen webbasierten Self-Servicepoint, der im Kontext des Bürgerprozessportals eingesetzt werden kann, konstruiert werden soll. Allerdings sollte die Softwarelösung so generisch gestaltet werden, dass sie auch zur Unterstützung von Dienstleistern, die ebenfalls mit einer webbasierten Benutzungsoberfläche umgehen, verwendbar ist.

Die technische Verbindung zwischen dem Self-Servicepoint und den anderen im Anwendungsbeispiel beschriebenen Servicepoints sollte lediglich darin bestehen, dass XML-Dokumente[44] als Repräsentanten der Serviceflows zwischen den Servicepoints ausgetauscht werden. Dadurch kann vermieden werden, dass dauerhaft Verbindungen zwischen den einzelnen Servicepoints notwendig sind bzw. auf einem gemeinsamen System gearbeitet werden muss. Im Rahmen des SfM-Ansatzes werden diese Prozessrepräsentationen als *Servicefloats* bezeichnet. Davon unterschieden werden sogenannte *Servicepointscripts*, welche die Aktivitäten beschreiben, die an einem Servicepoint im Rahmen eines Serviceflows standardmäßig ausgeführt werden. (Servicefloats und Servicepointscript werden detaillierter in Abschnitt 3.3.2 diskutiert)

Neue Prozessrepräsentationen sollten an einem Servicepoint dadurch erzeugt werden, dass Kopien aus Prozessvorlagen generiert und weitere Aktionen dann auf diesen Kopien durchgeführt werden. Die Prozessvorlagen sollten ebenfalls in Form von XML-Dokumenten am Servicepoint vorliegen. Die Struktur der XML-Dokumente wurde von den Projektverantwortlichen vorgegeben.

Zur Realisierung der zur Manipulation und Verwaltung der Prozessrepräsentationen und ihrer Vorlagen benötigte Anwendungsfunktionalität am zu konstruierenden webbasierten Servicepoint wurde die Programmiersprache Java 2 Standard Edition (Version 1.3) in

[44] Zur eXtensible Markup Language (XML) siehe Abschnitt 3.4.1.

Kombination mit der vom XML-Parser Xerces[45] (Version 1.2.3) angebotenen Java-XML-API eingesetzt.

Um die Tauglichkeit des funktionalen Prototypen zum Einsatz im Bürgerprozessportal www.hamburg.de zu überprüfen, ist zur Generierung der webbasierten Benutzungsoberfläche der StoryServer 5.0[46] verwendet worden. Hierbei wurde jedoch das Ziel verfolgt, dass die Benutzungsoberfläche möglichst ohne großen Aufwand gegen eine andere Weboberflächen generierende Technologie wie Beispielsweise Java Server Pages (JSP[47]) ausgetauscht werden kann. Des weiteren sollten die Auswirkungen auf die Realisierung der fachlichen Funktionalität, die sich aus der Verwendung einer webbasierten Benutzungsoberfläche ergeben, so minimal wie möglich gehalten werden[48]. Diese Anforderungen konnten durch Orientierung an der Entwurfsmetapher Fachlicher Service, die in Abschnitt 3.2.2 vorgestellt wird, eingehalten werden. Sie fokussiert auf die von Präsentation und Handhabung unabhängige Entwicklung fachlicher Funktionalität auch im Web-Umfeld (vgl. [LipWolZül01]).

Auf Basis der aus dem SfM-Ansatz resultierenden Konzepte und der einzusetzenden Technologien sowie der Orientierung an der sich aus der Anwendung der Entwurfsmetapher Fachlicher Service ergebenden Softwarearchitektur fiel die Entscheidung für die Grobstrukturierung des zu entwickelnden webbasierten Servicepoints als vierschichtige Softwarearchitektur[49], entsprechend Abbildung 11.

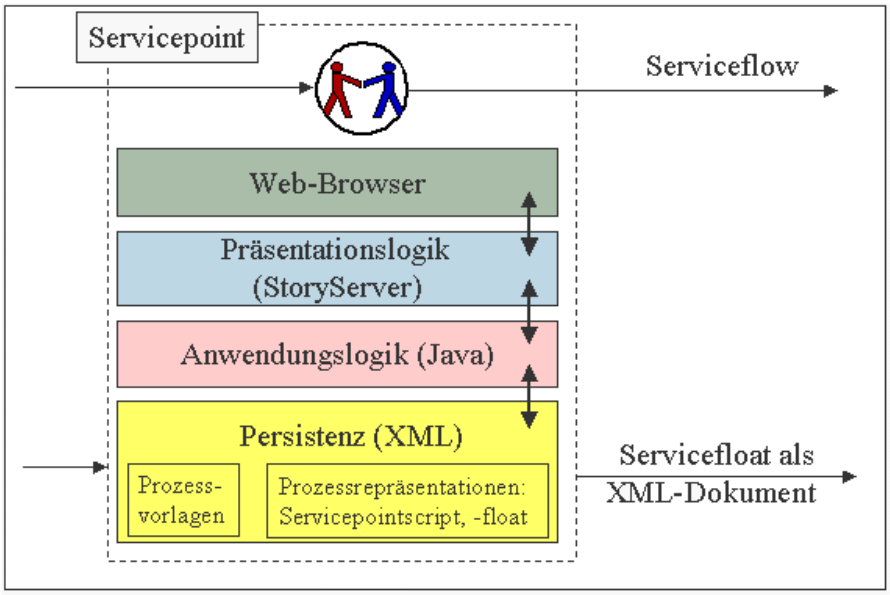

Abbildung 11: Grundlegender Aufbau des webbasierten Servicepoints

[45] Dieser Parser ist kostenlos unter http://www.apache.org verfügbar (03.2001).

[46] Grundlegendes zu Web Content Management Systemen und dem StoryServer 5.0 findet sich in Abschnitt 3.5.

[47] Zu JSP siehe z.B. ftp://ftp.java.sun.com/pub/jsp/11final-87721/jsp1_1-spec.pdf (04.2001)

[48] Auf die Besonderheiten, die sich aus der Verwendung einer webbasierten Benutzungsoberfläche ergeben, wird in Abschnitt 3.2.1 eingegangen.

[49] Zum Begriff der Softwarearchitektur siehe Abschnitt 3.2.3.

3.1.2 Entwicklungsverlauf

Auf Basis der im letzten Abschnitt dargestellten Vorraussetzungen begann dann im Rahmen des Projektseminars die Implementation der erforderlichen Anwendungslogik zum Umgang mit den Prozessrepräsentationen am Servicepoint, die im Rahmen der Grobstruktur in der dritten Ebene von oben angesiedelt wurde (vgl. Abbildung 11). Hierzu sind zunächst die zur Handhabung der Prozessrepräsentationen erforderlichen Umgangsformen identifiziert worden, die in Abschnitt 3.3.2 dieser Arbeit vorgestellt werden.

Anschließend wurden daraus die von den Prozessrepräsentationen anzubietenden Schnittstellen abgeleitet, gegen die im weiteren Verlauf der Implementation programmiert wurde[50]. Zusätzlich ist die Schnittstelle einer am Servicepoint benötigten Verwaltungseinheit (im folgenden als Servicepointmanager bezeichnet) definiert worden, an die sich Präsentationsbausteine wenden können, um z.B. bestimmte Prozessrepräsentationen anzufordern bzw. die Versendung von Servicefloats zum nächsten Servicepoint zu initiieren. Die Schnittstellendefinition wurden von Nol Shala, den beiden Projektverantwortlichen und mir (dem Autor) festgelegt. Die Implementation habe ich dann mit Nol Shala zusammen durchgeführt.

Wir haben uns dazu entschlossen, den Servicepointmanager nach dem Singleton-Entwurfsmuster (vgl. [GHJV96, S. 139 ff.]) umzusetzen. Ein wesentlicher Grund hierfür war, dass dadurch der Entwurf vereinfacht werden konnte. Andernfalls hätte ein Mechanismus gefunden und implementiert werden müssen, der es mehreren Servicepointmanager-Exemplaren an einem Servicepoint erlaubt, sich z.B. dahingehend zu koordinieren, wer welche Prozessrepräsentation verwaltet (siehe hierzu auch Abschnitt 3.6.2 und 3.6.5).

Zur Umsetzung der fachlich motivierten Umgangsformen an den Prozessrepräsentationen haben wir uns dafür entschieden, diese direkt auf die technische DOM-API[51] abzubilden, die vom verwendeten Java-XML-Parser zum Umgang mit XML-Dokumenten angeboten wird (siehe hierzu Abschnitt 3.4.2). Wir haben uns gegen den Entwurf und gegen eine Abbildung der Struktur der XML-Dokumente (der Prozessrepräsentationen) in ein eigenes Objektmodell entscheiden, da in diesem Fall ein Transformationsprotokoll zu implementieren gewesen wäre, was aus Zeitgründen zu aufwendig erschien. Hierbei haben wir das Ziel verfolgt, diese Entwurfentscheidung so weit wie möglich vor dem Servicepointmanager zu verbergen, um so eine leichte Austauschbarkeit im Hinblick auf eine mögliche andere Realisierungsvariante gewährleisten zu können.

Außerdem stand im Vordergrund, zusätzlich zu den Prozessrepräsentationen mit beliebig anderen, bis dato noch nicht bekannten Materialien, am Servicepoint umgehen zu können (z.B. Dokumente), ohne dass dies zu einem großen Änderungsaufwand im Hinblick auf die Implementation des Servicepointmanagers führen würde.

Da uns keine XML-fähige Datenbank zur Verfügung stand, haben wird uns dafür entschieden, als Persistenzmedium für die XML-Dokumente das File-System des Web-Servers zu verwenden. Hierbei wurde ebenfalls das Ziel verfolgt, diese Entwurfsentscheidung so weit wie möglich vor dem Servicepointmanager zu verbergen, um den Umstieg auf ein anderes Persistenzmedium zu erleichtern.

Nach Festlegung dieser Ziele haben wir die Implementationsarbeit dahingehend aufgeteilt, dass Nol Shala die Realisierung des Servicepointscripts und ich die des Servicefloats übernommen habe. Zusätzlich wurde von mir der Servicepointmanager umgesetzt. Ich habe mich hierbei weitestgehend an schon existierenden Entwürfen orientiert, die ebenfalls auf die

[50] Die Schnittstellendefinitionen finden sich im Anhang dieser Arbeit.

[51] DOM steht für Document Object Model.

Einhaltung der genannten Entwurfsziele ausgerichtet sind. Hierdurch konnten diese auch erfüllt werden, was in Abschnitt 3.6.2 gezeigt wird.

Die auf dem StoryServer basierende Benutzungsschnittstelle, die in Abschnitt 3.6.5 vorgestellt wird, wurde von Dorina Gumm und Beate Orlowski umgesetzt.

Die einzelnen Teile der Implementation sind vor dem Präsentationstermin des Projektseminars im Januar 2001 zusammengeführt worden. Nach Abschluss des Seminars habe ich noch einige Modifikationen an der Verwaltungseinheit vorgenommen.

Die Beschreibung des funktionalen Prototypen in Kapitel 3.6 basiert auf der überarbeiteten Version. Die Veränderungen bezogen sich hauptsächlich auf die interne Strukturierung des Servicepointmanagers und die Erweiterung seiner Schnittstelle. Dies war notwendig, da ich im Rahmen eines „Reviews“ des Servicepointmanagers unnötige Codeverdopplungen in einigen Klassen festgestellt hatte. Außerdem habe ich die Schnittstelle und die Funktionalität der Verwaltungseinheit dahingehend erweitert, dass auch mit mehreren Prozessexemplaren für ein und denselben Kunden am Servicepoint umgegangen werden kann. Diese Funktionalität war vorher größtenteils schon vorhanden, konnte aus Zeitgründen jedoch nicht vollständig im Rahmen des Projektseminars umgesetzt und getestet werden.

3.1.3 Zusammenfassung und Ausblick

In diesem Kapitel wurden zunächst die konzeptuellen und technischen Rahmenbedingungen dargestellt, welche die Entwicklung des funktionalen Prototypen im Projektkontext beeinflusst haben. Hieraus wurde die für den webbasierten Servicepoint gewählte Grobstruktur als vierschichtige Softwarearchitektur abgeleitet.

Anschließend wurde der Entwicklungsverlauf und die während des Konstruktionsprozesses der Ausgestaltung der Grobstruktur getroffenen Entwurfsentscheidungen sowie die hierbei verfolgten Ziele dargestellt. Angestrebt wurde v.a., dass ohne großen Änderungsaufwand

- die Prozessrepräsentationen gegen eine andere Realisierungsvariante ausgetauscht werden können,
- zusätzliche Materialien am Servicepoint verwaltet werden können,
- auf ein anderes Persistenzformat bzw. –medium umgestiegen werden kann.

Im weiteren Verlauf der Arbeit werden die in diesem Kapitel angesprochenen Aspekte detaillierter behandelt. Ob und wie die bei der Konstruktion verfolgten Ziele eingehalten werden konnten, wird in Kapitel 3.6 im Rahmen der detaillierteren Vorstellung der Architektur des Prototypen aufgezeigt.

3.2 Webbasierte Anwendungen, Fachlicher Service und Entwurfsprinzipien

Die im vorangehenden Abschnitt dargestellte Softwarearchitektur des Prototypen für webbasierte Servicepoints ist durch die Anforderung motiviert worden, den Einfluss der Besonderheiten webbasierter Benutzungsoberflächen auf den fachlichen Anteil der Anwendung zu minimieren. Die Entwurfmetapher Fachlicher Service diente hierbei als Anleitung zur Erreichung dieses Zieles und zur Grobstrukturierung der Architektur des Servicepoints.

Im folgenden wird zunächst dargestellt, wie webbasierte Anwendungen normalerweise aufgebaut sind und wodurch sie sich gegenüber herkömmlichen, z.B. Desktop-Anwendungen, unterscheiden. Anschließend wird die Entwurfsmetapher Fachlicher Service vorgestellt.

Dieses Kapitel schließt mit der Diskussion allgemeiner softwaretechnischer Kriterien, die bei der Konstruktion von Softwareanwendungen berücksichtigt werden müssen. Diese Kriterien dienen in Abschnitt 3.7 zur Evaluation des funktionalen Prototypen.

3.2.1 Charakteristische Merkmale webbasierter Anwendungen

Webbasierte Anwendungen werden zumeist in Anlehnung an das nachfolgend dargestellte Interaktionsschema realisiert (vgl. auch Abbildung 12). Client und Server kommunizieren über einen einfachen Anfrage/Antwort-Mechanismus. Hierbei sendet ein HTTP[52]-Client (meist ein Web-Browser), z.B. als Reaktion des Drückens eines Formular-Buttons durch den Benutzer, zunächst eine codierte Anfrage an einen HTTP-Server (der meist als Web-Server bezeichnet wird), der auf diese reagiert, indem er eine HTML[53]-Antwortseite zurücksendet. Die Antwortseite kann entweder statisch oder dynamisch sein. Man spricht von statischen Seiten, wenn sie „vorgefertigt“ z.B. im Dateisystem des Web-Browsers abgelegt sind und nur noch ausgeliefert werden müssen. Wird die Antwortseite erst zur Laufzeit (vollständig oder nur zu Teilen) generiert, indem auf Serverseite eine Anwendung zur Ausführung kommt, die z.B. Datenbankabfragen durchführt oder mit einem Host-System kommuniziert und die Resultate dann den Aufbau der Seite beeinflussen, so spricht man von dynamischen bzw. semidynamischen Seiten. Zur eindeutigen Identifikation der von dem Browser angeforderten Objekte werden sogenannte *Uniform Resource Locators (URL)* verwendet (siehe [BarMasMcC94]).

Abbildung 12 illustriert den grundsätzlichen Ablauf einer Interaktion zwischen HTTP-Client und HTTP-Server mit eingebundener Anwendung, die zusätzlich auf eine Datenbasis, z.B. eine Datenbank oder das Dateisystem, zugreift.

Grundsätzlich kann man zwischen server- und clientseitigen webbasierten Anwendungen unterscheiden (vgl. [Tur99]). Bei serverseitigen Anwendungen ist die Anwendung für die Realisierung der Anwendungslogik sowie die Erzeugung ihrer Repräsentation verantwortlich, der Browser wird im Prinzip nur zur Ein- und Ausgabe verwendet. Werden neben reinem HTML noch weitere Techniken, wie beispielsweise Skriptsprachen (z.B. JavaScript, VBScript) zur Validierung von Benutzereingaben oder Java-Applets[54] zur Realisierung komplexer Benutzungsoberflächen, eingesetzt, so spricht man auch von clientseitigen Webanwendungen (vgl. [Tur99]). Im folgenden wird in dieser Arbeit der Einfachheit halber unter einer Webanwendung eine Anwendung verstanden, die serverseitig abläuft und

[52] HTTP steht für Hypertext Transfer Protocol. Siehe z.B. [Fie97].

[53] HTML steht für Hypertext Markup Language und bezeichnet die Auszeichnungssprache des WWW (siehe z.B. [RecPom97, S. 846], [RagHorJac99]).

[54] Zu Java-Applets siehe z.B. [OrfHar97]

clientseitig lediglich HTML interpretiert wird. An dieser Konstellation können sehr gut die wesentlichen Charakteristika webbasierter Anwendungen dargestellt werden.

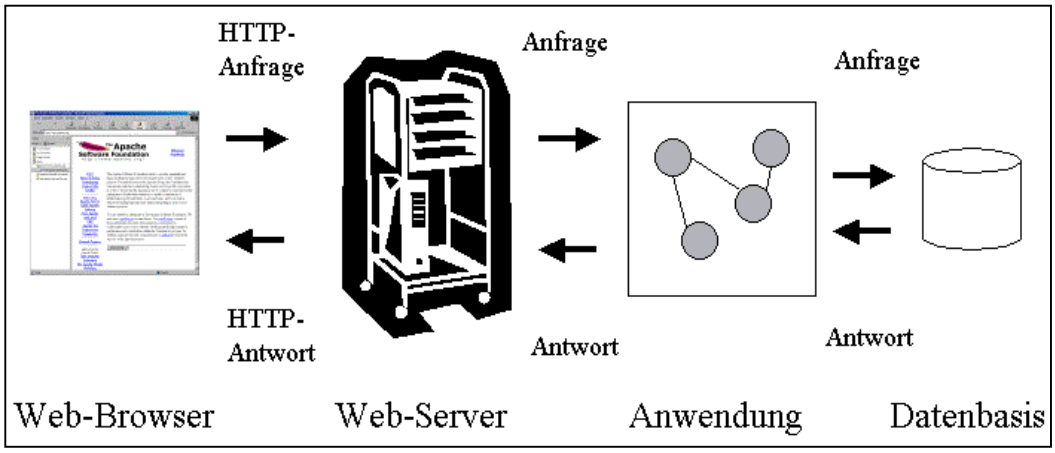

Abbildung 12: Interaktionsschema von webbasierten Anwendungen (in Anlehnung an [And00, S. 37],[ShaEar98, S. 192])

Zur Realisierung serverseitiger Anwendungen werden unterschiedliche Ansätze verfolgt[55], wobei die beiden wichtigsten an dieser Stelle kurz vorgestellt werden. Zum einen werden für einfache Anwendungen sogenannte serverseitige Skripte wie z.B. Java Server Pages eingesetzt, die in HTML-Seiten eingebettet und serverseitig ausgewertet werden, d.h. das Resultat wird in die HTML-Seite an der Stelle des Skriptes eingefügt, die erst dann zum Client übertragen wird. Hierzu wird der Web-Server um einen Interpreter für die jeweilige Skript-Sprache erweitert. Zum anderen gibt es auch die Möglichkeit, komplexere Anwendungen direkt über CGI (Common Gateway Interface) anzusprechen, die in einer herkömmlichen Programmiersprache wie z.B. C++ implementiert sind, sogenannte externe Programme (vgl. [Tur99]). Der Web-Server leitet in diesem Ansatz die Anfrage direkt an die Programme weiter, die nach ihrer Ausführung mit einer HTML-Seite antworten, welche vom Web-Server an den Client übertragen wird.

Signifikant an beiden Ansätzen ist, dass sie nicht per Konstruktionsprinzip die Vermischung von Anwendungslogik und die Präsentation betreffende Teile trennen, was die Wiederverwendung einschränkt (vgl. [OttSch00]). Eine Anleitung zur Vermeidung dieses Dilemmas liefert die im nächsten Abschnitt vorgestellte Entwurfsmetapher Fachlicher Service.

Im folgenden werden die wesentlichen Merkmale, an denen die Unterschiede bzw. Einschränkungen webbasierter gegenüber herkömmlicher Anwendungen verdeutlicht werden können, kurz vorgestellt.

- Zustandslosigkeit: Das HTT-Protokoll ist zustandslos. Das bedeutet, dass jede Anfrage eines Clients einen Server im exakt gleichen Zustand vorfindet bzw. der Server sozusagen „alles vergessen hat“, nachdem er eine Anfrage abgearbeitet hat (vgl. z.B. [GirLeeSch96]). Zur Überbrückung dieses Hemmnisses haben sich Techniken durchgesetzt, die z.B. unter den Begriffen „Cookies“ (siehe z.B. [SchEar98]) und „URL-Erweiterungen“ (siehe z.B. [Sch00b]) bekannt sind. Sie dienen dazu, den Zustand einer Anwendung über eine Folge von Client-/Server-Interaktionen ausgehend von ein und demselben Client (sog. Session; vgl. [Sch00b]) aufrechtzuerhalten.

[55] Für eine detailliertere Diskussion der grundsätzlichen Möglichkeiten zur Realisierung server- und clientseitiger webbasierter Anwendungen sei der Artikel von Turau [Tur99] empfohlen, der einen Überblick über die meisten relevanten Technologien gibt.

- Einseitige Interaktion: Eine Client-/Server-Interaktion kann nur vom Client initiiert werden, d.h. der Server kann von sich aus nicht die Kommunikation anstoßen (vgl. [GirLeeSch96]). So ist es dem Server z.B. nicht möglich, einen Neuaufbau der Darstellung einer Datenbanktabelle zu veranlassen, wenn sich die gespeicherten Daten verändert haben. Der Benutzer muss explizit z.B. auf den „Refresh-Button“ des Browsers drücken, damit er die aktuellste Darstellung erhält. Ein Benachrichtigungsmechanismus, wie er z.B. im MVC-Paradigma[56] (siehe z.B. [GHJV96]) der objektorientierten Programmierung verwendet wird, um die Bildschirmrepräsentation über Änderungen des zugrundeliegenden Datenmodells zu informieren, würde hier also nicht zum gewünschten Ergebnis führen.

- Eingeschränkte Rückkopplung: Hiermit ist gemeint, dass auf Aktionen des Benutzers erst mit einer aus dem prinzipiellen Interaktionsschema (vgl. oben) resultierenden zeitlichen Verzögerung reagiert werden kann (vgl. [GirLeeSch96]). Sollte z.B. das Anklicken eines Eingabefeldes in einer veränderten Darstellung desselbigen resultieren, würde dies bedeuten, dass die entsprechenden Informationen zunächst zum Server geschickt werden müssten, der dann eine neue Seite mit der modifizierten Präsentation generiert und zum Client zurücksendet. Hierdurch wird die Nützlichkeit solcher feingranularer Reaktionen in Frage gestellt, bzw. auf ihre Umsetzung verzichtet. Dadurch unterscheidet sich eine webbasierte Anwendung in den Möglichkeiten ihrer Handhabung[57] wesentlich von denen einer „normalen“ Desktop-Anwendung. Zur Überbrückung dieses Hemmnisses werden vornehmlich die weiter oben schon angesprochenen und zur Realisierung clientseitiger Anwendungen verwendeter Technologien, wie Skriptsprachen oder Java-Applets, eingesetzt. Letztere eignen sich auch zur Lösung des Problems der einseitigen Interaktion.

- Erhöhte Ausfallwahrscheinlichkeit und Gefahr inkonsistenter Zustände: Da sich eine Webanwendung aus vielen unterschiedlichen Komponenten zusammensetzt, die auf verschiedenen physikalischen Maschinen installiert sein können, ist auch die Wahrscheinlichkeit höher, dass einzelne Teile des Gesamtsystems ausfallen (vgl. [MeiSch01]) oder die Verbindung absichtlich abgebrochen wird. So könnte z.B. der Web-Server ausfallen oder der Client auf Initiative des Benutzers die Verbindung vorzeitig abbrechen, ohne dass die Anwendung in einen vorgesehenen Endzustand überführt wurde. Aufgrund der Zustandslosigkeit ergibt sich hier auch das Problem, zu entscheiden, wann eine Verbindung zwischen Client und Server abgebrochen ist, da es keine dauerhafte Verbindung gibt, sondern diese nur simuliert wird (vgl. oben). Des weiteren werden Webanwendungen meist von mehreren Benutzern (Web-Browsern) gleichzeitig verwendet. Hierdurch ist die Gefahr, dass es auf technischer Ebene zu den bekannten Problemen kommen kann, die aus konkurrenten Zugriffssituationen entstehen können, sehr viel höher als z.B. in einer auf einen Einzelplatz bezogenen Desktop-Umgebung (Inkonsistenz, nichtwiederholbares Lesen, Phantom Problem; vgl. [DenPet00, S. 162]).

---

[56] MVC steht für Model, View und Controller. Dieses Muster wird v.a. zur Konstruktion von Benutzungsschnittstellen verwendet. Es entkoppelt das Anwendungsobjekt (Modell) von seiner Darstellung (View) und den Möglichkeiten, mit denen die Benutzungsschnittstelle auf Benutzereingaben reagieren kann (Controller). Durch die Entkopplung wird u.a. erreicht, dass die Komponenten unabhängig voneinander variiert werden können. (vgl. [GJHV96, S. 4 ff.])

[57] Unter Handhabung versteht man nach [Zül98, S. 71] die Möglichkeiten der Interaktion mit einem Anwendungssystem.

Webbasierte Anwendungen zeichnen sich also durch eine Reihe spezifischer Merkmale aus, von denen die signifikantesten hier vorgestellt wurden[58]. Zur Überbrückung der Nachteile, denen ein Anwendungsentwickler in diesem Umfeld begegnen muss, sind eine Reihe von Technologien entwickelt worden, von denen einige ebenfalls genannt wurden.

Das Problematische hierbei ist jedoch, dass der Browserbenutzer die meisten der clientseitig eingesetzten Technologien abschalten kann und sich ein Anwendungsentwickler somit z.B. nicht darauf verlassen kann, dass Cookies verwendet werden können. Der Einsatz von Java-Applets wird v.a. im Internet aufgrund der damit verbundenen längeren Downloadzeiten vermieden.

Aus diesen Gründen wird zunehmend der reine HTML-Client insbesondere im Internet wieder favorisiert und die restlichen Anwendungsbestandteile auf dem Server positioniert. Die oben angesprochenen Nachteile werden damit in Kauf genommen. Der Einsatz dieser Art webbasierter Anwendung zur Realisierung von Self-Service Zugangssystemen zu eGovernment-Services scheint allerdings angemessen zu sein. Durch Verzicht auf spezielle Technologien auf dem Client-Rechner und die einfache, durch HTML vorgegebene Gestaltung der Benutzungsoberfläche, kann z.B. der Forderung nach Nichtausschließbarkeit durch technische Gründe nachgekommen werden (vgl. Abschnitt 2.1.3). Zur Realisierung des funktionalen Prototypen wurde diese Variante gewählt (vgl. Abschnitt 3.6.5).

In geschlossenen Umgebungen, wie sie z.B. für Servicepoints vorrausgesetzt werden können, die im Intra- oder Extranet verwendet werden, könnte jedoch das Vorhandensein entsprechend benötigter Technologien sichergestellt werden. Hieraus ergibt sich ein Hinweis darauf, auf welche Art und Weise ein System gestaltet werden sollte, das eine Prozessunterstützung für beide Einsatzgebiete übernimmt: Unabhängigkeit von der Benutzungsoberfläche bzw. die Präsentation und Handhabung realisierenden Technologien, die sich aus dem Einsatzkontext ergeben. Diesem Aspekt und anderen bei der Konstruktion zu beachtenden softwarearchitektonischen Prinzipien wird im weiteren Verlauf dieses Kapitels nachgegangen.

3.2.2 Fachliche Services

In Abschnitt 3.1 wurde bereits erwähnt, dass sich die grundlegend für den webbasierten Servicepoint gewählte schichtenbasierte Softwarearchitektur an der Entwurfsmetapher Fachlicher Service (vgl. z.B. [LipWolZül01], [OttSch00]) orientiert hat. Diese wird im folgenden kurz vorgestellt.

Fachliche Services haben zum Ziel, die fachliche Funktionalität eines Anwendungsbereichs unter beliebigen Benutzungsschnittstellentypen zugänglich zu machen. Beispiele für Benutzungsschnittstellentypen sind der weiter oben dargestellte Webtop, der klassische Desktop, aber auch neuere Schnittstellen wie Mobiltelefone (Waptop).

Ein Fachlicher Service wird in [Wol00] folgendermaßen definiert:

> *„Fachliche Services sind fachliche und technische Einheiten eines großen verteilten Anwendungssystems. Sie vergegenständlichen Geschäftslogik derart, dass nachvollziehbare und in einem Anwendungskontext zusammengehörige Leistungen gekapselt werden. Fachliche Services sind so realisiert, dass die konkrete Art und Weise offen bleibt, wie diese Leistungen erbracht und wie sie dem Anwender an der Benutzungsschnittstelle präsentiert werden.“*

Ein fachlicher Service orientiert sich am Dienstleistungsprinzip, wie es im Umfeld der objektorientierten Programmierung verstanden wird und in Abschnitt 2.1.4 bereits dargestellt

[58] Für eine detailliertere Diskussion siehe z.B.[GirLeeSch96] oder [OttSch00].

wurde. Fachliche Services werden unabhängig von der die Interaktion mit dem Benutzer betreffenden Teile einer Anwendung entwickelt und entsprechen so dem softwaretechnischen Entwurfskriterium der Trennung von Funktion und Interaktion (siehe hierzu auch Abschnitt 3.2.3). Ein Fachlicher Service macht keine Annahmen über seine Präsentation und Handhabung. Er kapselt und vergegenständlicht fachliches Wissen, z.B. über Geschäftsprozesse oder die Verwaltung von Materialien des jeweiligen Gegenstandsbereichs. Fachliche Services bieten Funktionalität an, die benötigt wird, um den Benutzer bei der Erledigung einer kleinen Menge zusammengehöriger Aufgaben zu unterstützen. Sie repräsentieren nicht das gesamte Anwendungssystem (vgl. [LipWolZül01]), sondern sind Teile von verteilten und großen Anwendungssystemen.

Die Unabhängigkeit von Präsentation und Handhabung wird durch Einführen einer weiteren Schicht oberhalb der Schicht, der Fachliche Services zugeordnet sind, erreicht. Der Grund hierfür ist, dass die Entscheidung für einen bestimmten Benutzungsschnittstellentyp (z.B. Webtop oder Desktop) auch meist die Realisierung der Anwendungslogik beeinflusst, da sich verschiedene Client-Typen durch unterschiedliche Handhabungen auszeichnen (vgl. z.B. [LipWolZül01]). Dies wurde in Abschnitt 3.2.1 z.B. unter dem Punkt „eingeschränkte Rückkopplung“ schon angesprochen. Lippert et al. geben zur Verdeutlichung ein Beispiel:

> *„For example, a Java Servlet[59] always transfers an entire browser page, whereas a normal desktop application requests only those services and data needed to respond to a specific user action. A WAP application has the same interaction style as the Servlet, but it has to take care of the very limited display and interaction options.“[LipWolZül01]*

Die zusätzliche Schicht wird als *Interaktionsschicht* bzw. *Interaktionsservice* bezeichnet. Diese übernimmt die Aufgabe, die Interaktionsspezifika für den jeweils gewählten Oberflächentyp umzusetzen (vgl. z.B. [LipWolZül01]). Dadurch kann sichergestellt werden, dass der Fachliche Service tatsächlich keine Annahmen über die jeweils angebundene Benutzungsschnittstelle machen muss. Die Interaktionsschicht setzt den Teil der Anwendung um, der sich in den Varianten für die verschiedenen Benutzungsschnittstellen unterscheidet. Hier wird das Wissen um den Umgang mit dem Fachlichen Service, die Koordination der Zusammenarbeit mehrerer Fachlicher Services, die Übersetzung von Eingaben der Benutzungsschnittstelle an die Schnittstelle des Fachlichen Services, bzw. umgekehrt und die Ansteuerung der jeweiligen Benutzungsschnittstelle implementiert (vgl. [OttSch00, S. 147 ff.]). In dieser Schicht kann z.B. auf die in Abschnitt 3.2.1 angeführten und sich auf die Präsentation und Handhabung beziehenden Besonderheiten webbasierter Anwendungen reagiert werden, so dass ihre Auswirkungen auf die eigentliche Fachanwendung minimiert werden können. Für einen webbasierten Servicepoint bedeutet dies somit, dass auf Basis derselben fachlichen Funktionalität unterschiedliche webbasierte Benutzungsoberflächen realisierbar sind. So könnte z.B. neben einem reinen HTML-Client im Internet (diese Variante wurde für den funktionalen Prototypen gewählt; siehe Abschnitt 3.6.5) auch ein Client, der auf einem Java-Applet basiert und im Intranet eingesetzt wird, ohne Änderungsaufwand für die Fachanwendung umgesetzt werden, wodurch ihre Wiederverwendbarkeit erhöht wird.

[59] Zu Java Servlets siehe z.B. ftp://ftp.java.sun.com/pub/servlet/22final-182874/servlet2_2-spec.pdf (04.2001)

3.2.3 Entwurfsprinzipien

In diesem Abschnitt werden grundsätzlich bei der Softwarekonstruktion einzuhaltende Entwurfsprinzipien diskutiert, damit eine Softwarearchitektur Qualitätskriterien wie Änderbarkeit und Wiederverwendbarkeit (siehe z.B. [Mey97]) auch auf der Ebene der Mikroarchitektur (siehe unten) gerecht werden kann. Diese dienen in Abschnitt 3.7 als eine der Grundlagen zur Beurteilung des funktionalen Prototypen, um auf dieser Basis eine allgemeine Empfehlung im Hinblick darauf geben zu können, wie eine Softwarearchitektur für Servicepoints im Web-Umfeld gestaltet werden sollte.

Der Begriff Softwarearchitektur wird z.B. von Shaw & Garlan folgendermaßen definiert:

> *„The architecture of a software system defines that system in terms of computational components and interactions among these components." [ShaGar96, S. 3]*

Komponenten einer Softwarearchitektur können z.B. Clients und Servers, Datenbanken, Module, Klassen oder Ebenen eines hierarchischen Systems sein. Interaktionen zwischen diesen Komponenten, die auch als *Konnektoren (connectors)* bezeichnet werden (vgl. [ShaGar96, S. 20]), sind beispielsweise durch einfache Prozedur- oder Methodenaufrufe, gemeinsame Variablen oder komplexere Verbindungen wie Client-/Server-Protokolle oder Ereignismechanismen repräsentiert (vgl. [ShaGar96, S. 3]). Eine Softwarearchitektur bestimmt also sowohl den statischen Aufbau als auch die Möglichkeiten zur dynamischen Interaktion der Komponenten, die ein Softwaresystem ausmachen (vgl. [FloGryMac99]).

Im erweiterten Sinne umfasst der Architekturbegriff neben anwendungssystembezogenen Aspekten aber auch eine Reihe von Entwurfsprinzipien und Organisationsformen von Software, die nicht an anwendungsfachlichen Inhalten orientiert sind (vgl. [RecPom, S. 652]). Zu den Organisationsformen zählen z.B. die Organisation in Programmbibliotheken, Anwendungsrahmenwerken oder Komponenten (siehe [RecPom, S. 652 ff.]). Hierauf wird im weiteren Verlauf allerdings nicht weiter eingegangen, und es sei auf die Literatur verwiesen[60].

Grundlage von Verständlichkeit, Änderbarkeit und Wiederverwendung ist das klassische Konzept der Modularisierung (sieh [Par72]), also der Zerlegung eines Softwaresystems in kleinere Einheiten. Modularisierung allein reicht allerdings nicht aus, um diese Ziele zu erreichen. Im Laufe der Zeit haben sich verschiedene Entwurfsprinzipien oder Kriterien guter Modularisierung herauskristallisiert, die auch um Zeitalter objektorientierter Programmierung nichts an ihrer Bedeutung eingebüßt haben. Für größere Softwareentwürfe ist die Einhaltung von Entwurfsprinzipien wesentlich (vgl. [RecPom97, S. 646]). Eine umfassende Übersicht findet sich z.B. in [BMRSS96, S. 397 ff.]. Im folgenden werden die für diese Arbeit relevantesten kurz beschrieben.

Kapselung

Kapselung ist eines der wichtigsten Prinzipien modularer als auch objektorientierter Programmierung. Es bezieht sich darauf, dass die konkrete Repräsentation des Zustands eines Objektes, d.h. seine interne Struktur, gegenüber benutzenden Objekten verborgen bleibt. Dieses Konzept wird traditionell als *Geheimnisprinzip* [Par72] bezeichnet. Dem Konzept der Kapselung wird auch das Prinzip der *Trennung von Schnittstelle und Implementation* zugeordnet. Die Schnittstelle definiert die Funktionalität, die von einer Komponente erbracht wird und wie sie zu benutzen ist. Die Implementation ist der Teil, in dem die versprochene Funktionalität realisiert wird. Die benutzenden Komponenten brauchen nur noch die

[60] Zum Thema komponentenorientierte Softwareentwicklung siehe z.B. [Gri98]; Anwendungsrahmenwerke werden z.B. in [Zül98] behandelt.

Schnittstelle zu kennen, die Implementation kann, sofern die Semantik gleich bleibt, problemlos gegen andere Implementationen ausgetauscht werden.

„Separation of Concerns“

Hierunter versteht man, dass die Aufteilung eines Systems anhand von Zuständigkeiten vorgenommen werden sollte. Komponenten, die an der gleichen Aufgabe beteiligt sind, sollen gruppiert und von denen abgegrenzt werden, die für eine andere Aufgabe zuständig sind. Wenn eine Komponente verschiedene Rollen in unterschiedlichen Kontexten spielt, d.h. unterschiedliche Aufgaben übernimmt, dann sollten diese verschiedenen Verantwortlichkeiten innerhalb der Komponente ebenfalls getrennt voneinander realisiert werden.

Unter diesen Punkt fällt auch das Prinzip der *Trennung von Funktion und Interaktion*, das insbesondere bei der Entwicklung von Benutzungsschnittstellen eine Rolle spielt. Hierunter versteht man, dass die anwendungsfachliche Funktionalität eines Softwaresystems unabhängig von dessen Darstellung entwickelt wird. Dadurch wird u.a. erreicht, dass eine Veränderung der Benutzungsschnittstelle keine Rückwirkung auf die Funktionalität hat, wodurch wiederum die Änderbarkeit erhöht wird. Dieses Prinzip wurde erstmalig in Smalltalk-Systemen verwendet und ist allgemein unter dem Begriff des MVC-Paradigmas bekannt (vgl. z.B. [GHJV96]).

Das Prinzip, Komponenten in einer Entwurfeinheit zu gruppieren, die gemeinsam eine bestimmte abgrenzbare Aufgabe erfüllen, wird auch als *funktionale Kohäsion* oder innerer Zusammenhalt bezeichnet. Komponenten sollen eine möglichst hohe fachliche und logische Kohäsion aufweisen (vgl. [RecPom97, S. 653]).

Minimierte Kopplung

Kopplung bezeichnet das Maß der Abhängigkeit einer Konstruktionseinheit von anderen Konstruktionseinheiten, z.B. gemeinsam benutzte oder ausgetauschte Daten. Um austauschbare, verständliche und wiederverwendbare Komponenten zu entwickeln ist eine minimale Kopplung anzustreben. Insbesondere sind zyklische Abhängigkeitsbeziehungen zu vermeiden.

Meyer [Mey97, S. 47 ff.] weist im Zusammenhang mit der Minimierung der Kopplung von Entwurfs- und Konstruktionseinheiten darauf hin, dass möglichst wenige, kleine und explizite Schnittstellen verwendet werden sollten. D.h. jede Komponente sollte mit möglichst wenigen anderen interagieren und dabei so wenig Informationen wie möglich und nötig austauschen, wobei der Austausch explizit (nur über öffentliche Schnittstellen) sein sollte.

Verwenden Konstruktionseinheiten andere nur unter einem Ausschnitt ihrer gesamten Schnittstelle, so spricht man auch von *loser Kopplung* ([Zül98, S. 56]). Eine Komponente verwendet eine andere dann quasi nur unter einer Sub-Schnittstelle, die meist einen bestimmten Typ repräsentiert. Alle Komponenten, die diesen Typ implementieren, können dann von der benutzenden Komponente verwendet werden, wodurch die Austauschbarkeit weiter erhöht wird. Ein Beispiel hierfür ist der Beobachtermechanismus. Jede Komponente, die eine andere beobachtet, muss die Beobachter-Schnittstelle implementieren, die dann zur Kommunikation mit beobachteten Komponenten verwendet wird, unabhängig vom konkreten Typ der implementierenden Komponente (vgl. [GHJV96]). Dieser Mechanismus wird v.a. zur Reduzierung zyklischer Abhängigkeitsbeziehungen eingesetzt.

Architektur- und Entwurfsmuster

Zur Umsetzung der Prinzipien in konkreten Softwarearchitekturen ist z.B. der Einsatz objektorientierter Technologien, die „von Haus aus“ z.B. Kapselung unterstützen und Schnittstellenkonzepte anbieten (z.B. Java) alleine nicht ausreichend. Favorisiert werden deswegen Standardisierungsmaßnahmen, die Anleitungen im Hinblick auf die Festlegung der

Grob- und Feinstruktur eines Softwaresystems, der Aufteilung von Zuständigkeiten und der Realisierung von Interaktionsbeziehungen zwischen Entwurfseinheiten geben. Neben Komponententechnologien und Rahmenwerken, auf die hier nicht weiter eingegangen wird, spielen insbesondere Architektur- und Entwurfsmuster eine große Rolle. (vgl. [Pae00, S. 156])

Architektur- und Entwurfsmuster beschreiben wiederkehrende Entwurfsprobleme und zugeordnete Lösungen (Problem-Lösungspaare) und werden typischerweise anhand eines Beschreibungsschemas dokumentiert, das seine charakteristischen Eigenschaften darlegt (vgl. [GHJV96], [BMRSS96, S. 19 ff.]). Dazu zählen z.B. der Zweck, die Motivation, die Anwendbarkeit, eine Beispielimplementierung oder bekannte Verwendungen. Der Nutzen von Architektur- und Entwurfsmustern liegt somit v.a. darin, dass sie eine Anleitung bei der softwaretechnischen Realisierung eines Softwaresystems bieten können (vgl. [Zül98, S. 325]). Durch die Wiederverwendung von guten und erprobten Architekturen kann der Entwurf neuer Softwaresysteme stark vereinfacht werden (vgl. [Bäu98, S. 88]).

Architekturmuster beschreiben Softwaresysteme auf einem sehr groben Level. Sie bestimmen die grundlegenden Strukturierungsprinzipien und legen fest, wie ein Softwaresystem in Subsysteme (Komponenten) aufgeteilt wird und wie diese miteinander interagieren. Die Subsysteme sind oft von großer Granularität, so dass sie selbst und ihre Verbindungen wieder aus kleineren Architektureinheiten zusammengesetzt werden (vgl. [BMRSS96, S. 12]). Zur Strukturierung dieser Einheiten wird der Einsatz von Entwurfsmustern empfohlen, die auch als Mikroarchitekturen bezeichnet werden (vgl. [FloGryMac99]). Sie sind im Gegensatz zu Architekturmustern feingranularer und beeinflussen nicht die fundamentale Struktur des gesamten Softwaresystems sondern nur die Architektur des jeweiligen Subsystems (vgl. [BMRSS96, S. 13]).

Ein Beispiel für Architekturmuster ist das schichtenbasierte Architekturmuster (vgl. [BMRSS96, S. 31 ff.]). Ein Beispiel für Entwurfsmuster ist das bereits erwähnte Beobachtermuster.

3.2.4 Zusammenfassung und Ausblick

In diesem Kapitel wurden zunächst die Besonderheiten herausgearbeitet, die webbasierte Anwendungen grundsätzlich von anderen, insbesondere Desktop-Anwendungen, unterscheiden. Dies sind hauptsächlich die Zustandslosigkeit des HTT-Protokolls, die einseitige Interaktion und die eingeschränkte Rückkopplung, die v.a. Auswirkungen auf die Möglichkeiten der Handhabung fachlicher Funktionalität haben. Zusätzlich wurde die hohe Gefahr inkonsistenter Zustände hervorgehoben.

Abschließend ist die Entwurfsmetapher Fachlicher Service vorgestellt worden, welche die Entscheidung für die Grobstrukturierung des webbasierten Servicepoints als Vier-Schichten-Architektur grundlegend angeleitet hat und darauf abzielt, die Auswirkungen der Verwendung einer webbasierten Benutzungsoberfläche auf den fachlichen Anteil der Anwendung zu reduzieren, um ihre Wiederverwendbarkeit zu erhöhen.

Hiernach sind allgemeine Entwurfsprinzipien diskutiert worden, die beim Entwurf von Softwaresystemen beachtet werden müssen. Sie stellen eine wesentliche Grundlage für die Evaluation der Ausgestaltung des funktionalen Prototypen in Abschnitt 3.7 dar und sind nachfolgend noch einmal aufgelistet:

- Kapselung (Geheimnisprinzip und Trennung von Schnittstelle und Implementation)
- Trennung von Zuständigkeiten („Separation of Concerns“)
- Anstreben von minimierter Kopplung
- Verwendung von Architektur- und Entwurfsmustern

Im nächsten Abschnitt werden die für die Entwicklung von Serviceflow Management Systemen vorgeschlagenen Entwurfsmetaphern und verwendeten Modellierungstechniken unter Berücksichtigung des Anwendungsbeispiels vorgestellt. Sie stellen, wie in Abschnitt 3.1.2 bereits angedeutet, die Grundlage für die Realisierung des funktionalen Prototyps dar.

3.3 Serviceflow Modellierung

Arbeitsteilige Prozesse werden im SfM-Ansatz als sogenannte Serviceflows modelliert, die aus einer Folge von Servicepoints bestehen, an denen der Leistungsempfänger auf den Leistungsanbieter zum Zwecke des Leistungsaustausches, entweder direkt oder mediiert durch Informationstechnologie (Self-Servicepoint oder Back Office), trifft (vgl. Abschnitt 2.3.2 und 2.4.4). Im folgenden werden diese Konzepte konkretisiert und anhand von Ausschnitten der Modellierung des in Kapitel 2.5 vorgestellten Anwendungsbeispiels illustriert. Hierzu wird zunächst dargestellt, welche Aspekte im Rahmen einer Dienstleistung überhaupt modelliert werden können, welche Modellierungstechniken im SfM-Ansatz Verwendung finden und in welcher Beziehung sie zueinander stehen. Anschließend wird in Abschnitt 3.3.2 darauf eingegangen, aus welchen Elementen sich konkrete Prozessrepräsentationen im SfM-Ansatz zusammensetzen sollten und welche Umgangsformen mit ihnen im Hinblick auf eine softwaretechnische Realisierung vorzusehen sind.

3.3.1 Servicemodellierung im Serviceflow Management Ansatz

Im Hinblick auf die Entwicklung einer Softwareunterstützung an Servicepoints stellt sich zunächst die Frage, wie Dienstleistungen sinnvoll modelliert werden können. Versteht man kundenorientierte Dienstleistungen, wie in Kapitel 2.1.2 dargestellt, als soziale Beziehungen, die sich basierend auf einer Vereinbarung situativ entfalten, so wird deutlich, dass Dienstleistungen an sich, aufgrund ihres individuellen Charakters, einer Abstraktion nur schwer zugänglich sind (vgl. [Kli00]). Hingegen können diejenigen Teile modelliert werden, die wiederkehrend in Dienstleistungssituationen auftauchen und standardmäßig zur Leistungserbringung verwendet werden, bzw. den üblichen Weg der Leistungserstellung beschreiben. Klischewski schreibt hierzu:

> *„Services als Erkennen und situatives Befriedigen subjektiver Bedürfnisse sind (...) als solche nicht modellierbar, wohl aber die beobachtbar wiederkehrenden Vorgänge (...) sowie die Materialien, Produkte oder Werkzeuge (...), die im Rahmen der Serviceleistung eine wichtige Rolle spielen (...).“ [Kli00]*

Die besondere „Natur“ von kundenorientierten Dienstleistungen und deren Rückwirkung auf die Servicemodellierung wird im SfM-Ansatz berücksichtigt, indem dort „lediglich“ die wiederkehrenden und somit standardisierbaren Aspekte einer Dienstleistung modelliert werden. Nach [KasKliWet01] ist für einen Serviceflow folgendes zu modellieren:

- *„die an einzelnen Servicepunkten in der Regel zu erbringenden standardisierten Teilleistungen als eine Menge von einzelnen Aktivitäten.*
- *die jeweiligen Vor- und Nachbedingungen dieser Menge von Aktivitäten (bezogen auf einen Servicepunkt).*
- *die sinnvolle bzw. mögliche Reihenfolge dieser Teilleistungen als ausgewiesenen Standardprozess (ggf. mit Varianten).“*

Ein Serviceflow besteht aus einer Folge von Servicepoints. Diese repräsentieren die Teilleistungen, die dem Kunden zu einem bestimmten Zeitpunkt im Leistungserstellungsprozess angeboten wird und die zusammengefasst die Gesamtleistung ergeben. Sie stellen sozusagen die verallgemeinerbaren und somit modellierbaren Anteile des Dienstleistungsprozesses dar. Ein modellierter Serviceflow repräsentiert den Standardablauf der Leistungserbringung und ist als Prozessmodell nach der unterstützenden Sichtweise (vgl. Kapitel 2.3.1) aufzufassen, d.h. er gibt grundsätzlich keine Verfahrensweise vor, sondern wird als Prozessmuster (vgl. [Gry96]) verstanden, welches die konkrete Leistungserbringung

anleitet. In einem Serviceflowmodell werden Reihenfolgebeziehungen zwischen Servicepoints angegeben, die als sinnvoll erachtet werden oder aufgrund von Abhängigkeiten nicht anders möglich sind. Ein Serviceflow vergegenständlicht Erfahrungen mit dem Umgang eines kooperativen Arbeitsprozesses und repräsentiert den „Normalfall“. Sollte dieser in der konkreten Situation nicht mehr „passen“, dann ist grundsätzlich eine Abweichung möglich. Bekannte Abweichungen (z.B. zusätzliche Servicepoints) werden als Varianten modelliert, unbekannte können situativ (soweit vertretbar und realisierbar) von den Prozessteilnehmern hinzugefügt, also quasi nachmodelliert werden.

Ein Servicepoint zeichnet sich durch die bereits in Abschnitt 2.3.2 dargestellten Merkmale aus. Entsprechend des Serviceflow Modells gibt es auch ein Modell für den Servicepoint (vgl. Abschnitt 3.1), das aus der Angabe der normalerweise von der (den) Organisationseinheit(en) durchzuführenden Menge von Aktivitäten besteht, die notwendig sind, um die versprochene(n) Teilleistung(en) des Servicepoints zu erbringen. Dieses Modell wird, wie auch auf der Ebene des Serviceflows, im Sinne eines Musters verstanden. Zu einer Aktivitätsbeschreibung gehört mindestens ein Name, eine Beschreibung der Aufgabe und eine Zuständigkeitsangabe.

Zu jedem Servicepoint wird eine Menge von Vor- und Nachbedingungen modelliert. Die Vorbedingungen geben an, welche Vorleistungen an einem Servicepoint erwartet werden, damit dort die versprochene Leistung erbracht werden kann. Die Nachbedingungen beschreiben, was nachfolgende Servicepoints an erbrachten Teilleistungen erwarten können. Hierdurch wird das Geschäftsprozessprotokoll des Servicepoints festgelegt (vgl. Kapitel 2.3.1).

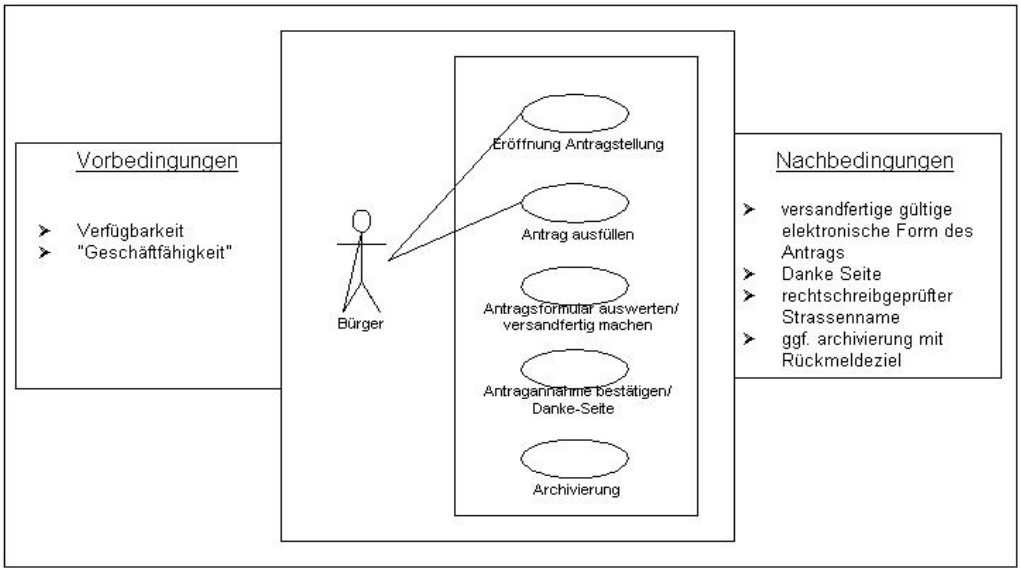

Abbildung 13: UseCase-Diagramm: „Anwendungsunterstützung der Antragsannahme durch www.hamburg.de“

Zur Identifikation der einzelnen Servicepoints eines Serviceflows, der im Rahmen eines Servicepoints standardmäßig durchzuführenden Aktivitäten und der sich dadurch ergebenden Vor- und Nachbedingungen einschließlich der Bestimmung der zur übergreifenden Kooperation notwendigen Dokumente, wird im SfM-Ansatz auf etablierte Modellierungs- und Analysetechniken zurückgegriffen.

Neben der bereits in Abschnitt 2.5 präsentierten Darstellung für Servicefolgen als Serviceflow Diagramm, die sich an Modellierungstechniken wie sie für Workflows verwendet werden, orientiert (zur Workflow Modellierung siehe z.B. [JabBöhSch97]), werden v.a. auch Techniken aus dem Umfeld der *Unified Modeling Language (UML)* (zu UML siehe z.B. [Oes01]), insbesondere *UseCases (Anwendungsfälle)* mit den entsprechenden Diagrammen

sowie Aktivitätsdiagramme und weitere benutzerorientierte Modellierungstechniken wie Kooperationsdiagramme und Szenarien mit Glossaren eingesetzt (hierzu siehe z.B. [Zül98, S. 601 ff.], [KraWetRat96]).

Serviceflow Diagramme beschreiben arbeitsteilige Prozesse auf einem sehr hohen Abstraktionsniveau, aus dem hauptsächlich Reihenfolgebeziehungen von Servicepoints, vorgesehene Organisationseinheiten für Servicepoints und zusätzliche Verbindungen zwischen Servicepoints in Form von Supportprozessen dargestellt werden (siehe [KliWet00] und Abschnitt 2.5). Zur detaillierteren Modellierung wird jedem Servicepoint ein UseCase-Diagramm zugeordnet, das die Aktivitäten in Form von UseCases[61] benennt, welche an einem Servicepoint von den jeweils zugeordneten Akteuren in Rechnerarbeit standardmäßig erledigt werden. Zusätzlich werden Vor- und Nachbedingungen benannt, die zur Ausführung der durch die Aktivitäten dargestellten Serviceteilleistung normalerweise notwendig sind, bzw. hinterlassen werden. Abbildung 13 präsentiert z.B. ein UseCase-Diagramm, das die UseCases darstellt, die im Rahmen einer Selbstbedienung zum Anstoßen des Gesamtprozesses über das Hamburger Bürgerprozessportal normalerweise vom Bürger durchgeführt werden müssen. Die drei unteren UseCases beschreiben detaillierter, welche Aktionen das System ausführt, nachdem der Bürger die Versendung des Antragsformulars eingeleitet hat.

UseCases können zur Beschreibung einer anwendungsorientierten Soll-Situation (Rechnereinbettung) verwendet werden (vgl. [FloGryMac99]). Neben einer grafischen Darstellung werden die an einem Servicepoint durchzuführenden Aktionen, also der antizipierten Rechnerinteraktionen im Rahmen einer Serviceleistung, detaillierter, z.B. auch mit Szenarien beschrieben. Das in Abschnitt 2.5.2 präsentierte Soll-Szenario korrespondiert mit dem in Abbildung 13 dargestellten UseCase-Diagramm und bietet für jede der dort aufgeführten Aktivitäten eine textuelle Erläuterung. Jede Aktivität (UseCase) kann selbst wieder verfeinert werden, soweit sie kein atomares Prozesselement (vgl. Kapitel 2.3.1) darstellt. Hierzu werden z.B. Aktivitätsdiagramme eingesetzt (vgl. [KliWetBah01]). Aktivitätsdiagramme beschreiben ablaufartig Folgen von Aktivitäten (mit Verzweigungen und Synchronisationen), die selbst wieder verfeinert und als Aktivitätsdiagramme dargestellt werden können.

Strukturierte Vorgänge und teamübergreifende Kooperationen, die neben dem eigentlichen Serviceflow ablaufen, werden ebenfalls mit Aktivitätsdiagrammen und Kooperationsbildern abgebildet (vgl. [KliWetBah01]). Mit Hilfe von Kooperationsbildern können synchrone (z.B. Telefonanruf) und asynchrone (z.B. zeitversetzter Zugriff auf gemeinsame Informationsräume) Kooperationsbeziehungen zwischen verschiedenen Servicepoints und die ausgetauschten Arbeitsmittel illustriert werden. Aktivitätsdiagramme beschreiben hier workflowartige Abläufe zwischen Servicepoints. Servicepoints stellen somit auch Start- und Endpunkte für diese unterstützenden Prozesse dar.

Abbildung 14 stellt überblicksartig die Zusammenhänge zwischen einigen der angesprochenen Modellierungstechniken dar, die auf den verschiedenen Modellierungsebenen Verwendung finden.

Angemerkt sei an dieser Stelle, dass bei der Realisierung des funktionalen Prototypen (Abschnitt 3.6) die Notwendigkeit der Integration von Supportprozessen bzw. der Zugriff auf gemeinsame Informationsräume nicht berücksichtigt wurde. Aus diesem Grund wird auf diese Aspekte im weiteren Verlauf nicht detaillierter eingegangen.

[61] „Ein Anwendungsfall beschreibt eine Menge von Aktivitäten eines Systems aus der Sicht seiner Akteure, die für die Akteure zu einem wahrnehmbaren Ergebnis führen. (...)“ [Oes01, S. 5]

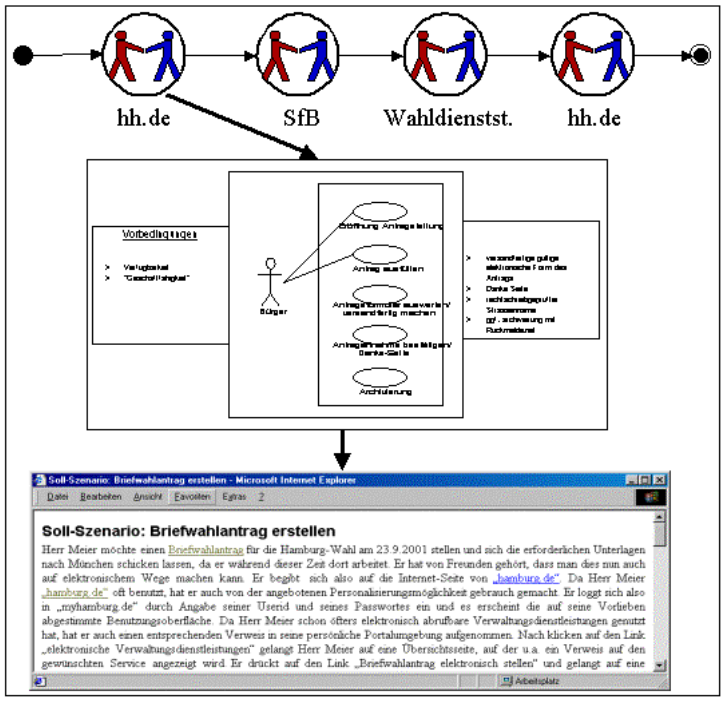

Abbildung 14: Zusammenhang der Modellierungsebenen im SfM-Ansatz am Beispiel von Serviceflow Diagramm, UseCase-Diagramm und Szenario

3.3.2 Prozessrepräsentationen und deren Umgangsformen

Konkrete softwaretechnische Repräsentationen von Serviceflows werden als Servicefloats bezeichnet (vgl. [KliWet00]). Servicefloats können als Transportbehälter aufgefasst werden, die von Servicepoint zu Servicepoint geschickt werden. Sie beinhalten für die Prozessteilnehmer notwendige Informationen, damit diese ihre eigene Leistung erbringen können. Hierzu gehört z.B. das Serviceflowmodell, der Zustand des Serviceflows oder die benötigten und erzeugten Arbeitsmittel (Dokumente). Letztere müssen nicht direkt im Servicefloat mitgeführt werden. Es könnte auch stellvertretend ein Verweis auf die relevanten Dokumente transportiert werden, die sich physikalisch an einem ausgezeichneten Ort befinden und erst bei Bedarf zum Servicepoint übertragen werden.

Eine konkrete softwaretechnische Repräsentation eines Servicepoints wird als Servicepointscript bezeichnet. Ein Servicepointscript ist im Sinne eines Prozessskriptes zu verstehen, das laut [CurKelOve92] „(...) a process model to be performed by a human (...)“ darstellt. Es beschreibt eine Menge von Aktivitäten, die durch einen (oder mehrere) (menschliche) Akteur(e) durchgeführt werden sollten[62]. Ein Servicepointscript kann ebenfalls als ein Behälter aufgefasst werden, der jedoch nicht zwischen Servicepoints transportiert wird, sondern an einem verbleibt und die Informationen über die Leistungserbringung an einem Servicepoint beinhaltet. Es beinhaltet z.B. das Servicepointscriptmodell, den Zustand des Servicepointscripts und die Vor- und Nachbedingungen.

Durch Beibehaltung der schon durch die verschiedenen Modellierungsebenen angedeuteten Aufteilung des Serviceflows in Servicefloat und Servicepointscript resultiert v.a. ein Flexibilitätsgewinn. Das verwendete Prozessmodell am Servicepoint kann dadurch unabhängig vom Prozessmodell des übergeordneten Serviceflow variiert und ausführt werden. Es können auch verschiedene Servicepointscript-Varianten an einem Servicepoint im gleichen

[62] Sobald eine Aktivität einen ausreichend hohen Automatisierungsgrad aufweist, kann die Ausführung natürlich auch von einem technischen Akteur übernommen werden.

Servicefloat Verwendung finden, wodurch eine Reihe verschiedener Serviceflow-Typen auf der Basis des gleichen Servicefloats realisiert werden können. Andererseits betrifft z.B. eine Veränderung der Reihenfolgebeziehungen in Servicefloats nicht die Servicepointscripts. Dadurch wird die Wiederverwendung bereits existierender Prozessmodelle gefördert.

Grundsätzlich könnte auch auf die informationstechnische Abbildung von Servicepointscripts verzichtet werden, wenn an einem Servicepoint z.B. nur eine Aktivität auszuführen ist und hierzu lediglich Informationen des bisherigen Prozessverlaufs benötigt werden. Für den weiteren Verlauf der Arbeit wird jedoch davon ausgegangen, dass Servicepointscripts am Servicepoint verwendet werden, da diese Annahme auch der Umsetzung des funktionalen Prototypen, der in Abschnitt 3.6 vorgestellt wird, zugrundegelegt wurde.

Im Hinblick auf die Prozessunterstützung an einem Servicepoint sind nachfolgend dargestellte Aktivitäten softwaretechnisch zu unterstützen, die der Anwender in Interaktion mit dem System durchführt. Hieraus lassen sich notwendige Umgangsformen ableiten, die an den Prozessrepräsentationen implementiert werden müssen. Abbildung 15 stellt die Aktivitäten als UseCase-Diagramm aus Sicht eines Dienstleistungsmitarbeiters dar. Die Aktivitäten und die sich daraus ergebenden Umgangsformen werden nachfolgend beschrieben.

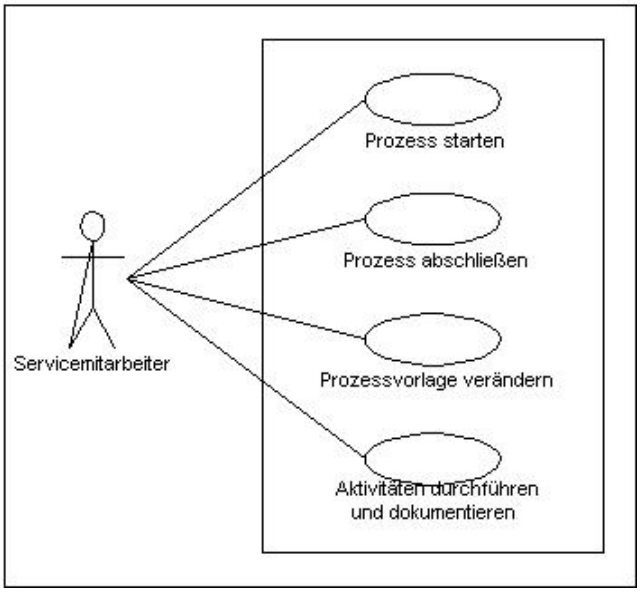

Abbildung 15: UseCase-Diagramm: Aktivitäten am Servicepoint

Prozess starten

Soll ein neuer Serviceflow „zum Laufen gebracht werden“, so wählt der Benutzer die zum jeweiligen Fall passende Variante aus einer Sammlung von Vorlagen (Masters) aus. Stehen verschiedene Servicepointscript-Varianten zum ausgewählten Servicefloat zur Verfügung, so kann der Benutzer hier ebenfalls die passende bestimmen. Eine andere Möglichkeit wäre, das zugehörige Servicepointscript automatisch zu bestimmen. Konkrete Prozessrepräsentationen werden durch Generierung einer Kopie vom Master erzeugt, die dann zur Bearbeitung am Servicepoint zur Verfügung gestellt werden. Nun können z.B. persönliche Informationen über den Kunden eingetragen oder der Prozess an spezielle Erfordernisse angepasst werden (vgl. hierzu auch Umgang Prozessvorlage verändern). Der Serviceflow, also der Gesamtprozess, gilt als gestartet, sobald eine im Servicepointscript beschriebene Aktivität durchgeführt wird.

Die Aktivität Prozess starten wird an jedem Servicepoint auf der Ebene der Servicepointscripts immer wieder neu ausgeführt. Um während eines laufenden Serviceflows

den Teilprozess am Servicepoint zu starten, wählt der Benutzer in diesem Fall das korrespondierende Servicefloat aus einer Sammlung für laufende Prozesse aus. Im Hinblick auf die Erzeugung eines passenden Servicepointscripts ergeben sich hier wiederum die oben genannten Optionen. Eine weitere Möglichkeit wäre, dass das zu verwendende Servicepointscript vom Servicefloat selbst transportiert wird.

Hieraus ergeben sich folgende Umgangsformen:

- Sondierung des Typs einer Prozessrepräsentation, so dass festgestellt werden kann, um welche Art von Prozess oder Teilprozess es sich handelt (z.B. „Briefwahlantrag“ oder „Einkommenssteuererklärung“).
- Zuweisung eines eindeutigen Identifikators zu Servicefloat und Servicepointscript und dessen Abfrage. Dem Servicepointscript muss zusätzlich ein Identifikator zugewiesen werden können, um es eindeutig dem zugehörigen Servicefloat zuordnen zu können.
- Setzen von kundenbezogenen Informationen (z.B. Kundennummer, Name, Adresse). Diese Informationen müssen auch abgefragt werden können, um bei laufenden Prozessen am Servicepoint z.B. eindeutig feststellen zu können, zu welchem Kunden welcher Prozess gehört.
- Erzeugen einer Kopie, um den Master in eine konkrete Prozessrepräsentation transformieren zu können.

Prozess abschließen

Ein Teilprozess wird abgeschlossen, indem die Beendigung des Servicepointscripts vom Benutzer eingeleitet wird. Dies kann nicht automatisch, z.B. nach Abschluss der letzten im Servicepointscript hinterlegten Aktivität geschehen, da es konzeptuell dem Benutzer überlassen wird, ob er beispielsweise Aktivitäten auslässt oder die Reihenfolge der Aufgabenerledigung ändert.

Der Abschluss des Servicepointscripts impliziert, dass die Versendung des korrespondierenden Servicefloats zum nächsten eingetragenen Servicepoint initiiert wird. Vor Versand werden alle relevanten Informationen, die zum nächsten Servicepoint zu übertragen sind, aus dem Servicepointscript ausgelesen (z.B. Dokumente und erreichte Nachbedingungen) und in das Servicefloat übertragen. Anschließend wird das Servicefloat „weitergeschaltet“, d.h. der aktuelle Servicepoint wird als durchlaufen markiert und der nächste „anzulaufende“ Servicepoint wird als solcher gekennzeichnet.

Handelt es sich bei dem aktuellen Servicepoint um den letzen im Servicefloat eingetragenen, so gilt auch das Servicefloat bzw. der Gesamtprozess als beendet.

Hieraus ergeben sich folgende Umgangsformen:

- Abfragen der Dokumente und erreichten Nachbedingungen aus dem Servicepointscript.
- Hinzufügen der Dokumente und erreichten Nachbedingungen in das Servicefloat.
- „Weiterschalten“ des Servicefloats.
- Abfragen des nächsten „anzusteuernden“ Servicepoints am Servicefloat (z.B. von einem Transportautomaten).

Prozessvorlage verändern

Um eine konkrete Prozessrepräsentation auf besondere Erfordernisse anzupassen, können die Benutzer das im Servicefloat bzw. Servicepointscript vorhandene Prozessmodell, welches den Standardablauf beschreibt, dynamisch verändern, wobei sich die Veränderungen sofort auf das Prozessexemplar auswirken. Die Prozesshistorie, also die bereits durchlaufenen

Servicepoints oder erledigten Aktivitäten, dürfen im Nachhinein nicht verändert werden, da sie der Dokumentation dienen (siehe auch Aktivitäten durchführen und Dokumentieren).

Zur Modifikation der Prozessmodelle müssen folgende Umgangsformen zur Verfügung gestellt werden:

- Hinzufügen oder Entfernen von Prozesselementen (Servicepoints oder Aktivitäten).
- Veränderung von Reihenfolgebeziehungen zwischen Prozesselementen.
- Zurückspringen zu einer bereits durchgeführten Aktivität.
- Hinzufügen oder Entfernen von Vor- und Nachbedingungen am Servicepointscript.
- Neuzuweisung von Zuständigkeiten.

Aktivitäten durchführen und dokumentieren

Zur Aufgabenerledigung führen die Mitarbeiter am Servicepoint die im Servicepointscript beschriebenen und als nächstes durchzuführenden Aktivitäten aus. Welche Aktivität als nächstes durchzuführen ist, kann am Zustand des jeweiligen Servicepointscripts abgelesen werden. Die betreffende Aktivität ist als aktuelle Aktivität markiert. Bereits durchgeführte Aktivitäten werden als erledigt gekennzeichnet. Letztere können nicht mehr modifiziert werden, da sie den Prozesszustand dokumentieren. Soll eine Aktivität im nachhinein doch noch mal bearbeitet werden, so ist diese explizit erneut „zu öffnen“. Dies entspricht jedoch einer Veränderung des Prozessmodells bzw. des vorgesehenen Ablaufs und ist somit dem vorhergehenden Punkt zugeordnet worden.

Zusätzlich wird dokumentiert, ob Nachbedingungen eingehalten werden konnten. Dies kann automatisch oder auch manuell durch den Bearbeiter geschehen. Ebenso wird dokumentiert, ob die an einem Servicepoint erwarteten Vorleistungen im bisherigen Verlauf des Serviceflows erbracht wurden. Wie auf nicht eingehaltene Vorbedingungen zu reagieren ist, wird dem Akteur am Servicepoint überlassen, d.h. er kann entscheiden, ob Aktivitäten trotzdem durchgeführt werden können oder ob der Serviceflow z.B. erneut zu einem vorherigen Servicepoint gesendet werden muss, damit die fehlenden Vorleistungen erbracht werden. Vor- und Nachbedingungen dienen also ebenfalls der Dokumentation und besitzen keine technische Interpretation in dem Sinne, dass bei nicht erfüllten Nachbedingungen die Weiterleitung verhindert wird[63].

An Servicepoints werden oft Dokumente als Ergebnis bzw. Dokumentation der durchgeführten Aktivitäten erstellt, z.B. ausgefüllte Formulare oder Gesprächsprotokolle. Diese werden zum nächsten Servicepoint transportiert, wenn sie dort benötigte Arbeitsmittel im Sinne der „flow dependencies“ (vgl. Abschnitt 2.3.1) darstellen. Die im Servicefloat enthaltenen Arbeitsmittel werden somit von den Akteuren zugegriffen und modifiziert. Neue werden grundsätzlich zunächst im Servicepointscript hinterlegt.

Um Aktivitäten in Servicepointscripts bzw. das Servicefloat selber Aufgabenträgern zuweisen zu können, müssen entsprechende Informationen an den Prozessrepräsentationen abgefragt werden können. Informationen über Zuständigkeiten können von einem Aufgabenträger auch dazu genutzt werden, direkten Kontakt mit anderen Aufgabenträgern außerhalb des Systems aufzunehmen, um z.B. Nachfragen zu bereits erbrachten Leistungen stellen zu können.

Hieraus ergeben sich folgende Umgangsformen:

- Kennzeichnen und Abfragen des Prozesszustandes (Historie, aktuelle Aktivität bzw. Servicepoint, weiterer Verlauf)

[63] Diese Freiheit könnte natürlich je nach Bedarf eingeschränkt werden.

- Kennzeichnen und Abfragen des Zustands von Vor- und Nachbedingungen.
- Modifikation und Abfragen von Dokumentinhalten in Servicefloat und Servicepointscript und Hinzufügen neuer Dokumente.
- Abfragen von Zuständigkeiten.

Die Prozessrepräsentationen Servicefloat und Servicepointscript setzen sich aus den nachfolgend dargestellten Elementen zusammen, die im Rahmen des in Abschnitt 3.1 vorgestellten Projektseminars identifiziert wurden und sich aus den beschriebenen Umgangsformen ergeben. Sie sind an dieser Stelle aufgeführt, da sie in Verbindung mit den eben diskutierten Umgangsformen die Grundlage für die konkrete Realisierung der Prozessunterstützung mit den im nächsten Abschnitt vorgestellten Technologien bilden. Auf eine detailliertere Darstellung der einzelnen Elemente, wie z.B. Dokumente oder Servicepoint, die sich selbst wieder aus einer Reihe weiterer Elemente zusammensetzen, wird hier aus Platzgründen verzichtet.

Servicefloats setzen sich aus folgenden Bestandteilen zusammen:

- Eindeutiger Identifikator
- Einem Typ, der die Art des Serviceflows repräsentiert
- Basisinformationen über den Kunden (z.B. Kundennummer, Name, Adresse)
- Basisinformationen über den (oder die) Leistungserbringer an den vorgesehenen Servicepoints
- Aktueller Servicepoint
- Liste der nächsten „anzulaufenden“ Servicepoints
- Liste der schon „durchlaufenen“ Servicepoints
- Liste von akkumulierten Nachbedingungen, die von den schon „durchlaufenen“ Servicepoints hinterlassen wurden
- Liste von Dokumenten, die im Laufe des Serviceflows im Rahmen der Leistungserbringung an den Servicepoints produziert wurden.

Servicepointscripts setzen sich aus folgenden Elementen zusammen:

- Eindeutiger Identifikator
- Einen Namen, der den Servicepoint identifiziert, an dem das Servicepointscript ausgeführt wird
- Einen Typ, der die Art des Servicepointscripts repräsentiert
- Eindeutiger Identifikator des Servicefloats, in dessen Rahmen das Servicepointscript zum Einsatz kommt
- Basisinformationen über den (oder die) Leistungserbringer
- Basisinformationen über den Kunden (z.B. Kundennummer)
- Aktuelle Aktivität
- Liste der noch auszuführenden Aktivitäten
- Liste der schon ausgeführten Aktivitäten
- Liste der Vorbedingungen der Menge der Aktivitäten

- Liste der Nachbedingungen der Menge der Aktivitäten
- Liste von Dokumenten, die im Rahmen der Leistungserbringung an dem Servicepoint erstellt werden.

Die grundlegende Struktur konkreter Servicefloats und Servicepointscripts ist also für den jeweiligen Typ identisch. Exemplare der Prozesstypen unterscheiden sich nur im Hinblick auf die Inhalte, welche die Struktur ausfüllen. Ebenso sind die zu realisierenden Umgangsformen unter den angesprochenen Aspekten gleich. Ein SfMS kann sich also im Hinblick auf die Erfüllung seiner Aufgabe, dem Management von Serviceflows, auf die allgemeine Struktur der Prozessrepräsentationen verlassen und von konkreten Inhalten abstrahieren.

3.3.3 Zusammenfassung und Ausblick

In diesem Kapitel wurde zunächst aufgezeigt, dass nur die beobachtbar wiederkehrenden und somit standardisierbaren Aspekte von Dienstleistungen einer Abstraktion zugänglich und somit modellierbar sind. Dies wird im SfM-Ansatz berücksichtigt, indem Serviceflows als Folgen von Servicepoints modelliert werden, wobei die Servicepoints standardisierbare Teilleistungen im Rahmen der Gesamtserviceleistung repräsentieren. Servicepoints selber werden dadurch modelliert, dass Aktivitäten identifiziert werden, die standardmäßig im Rahmen der Teilleistung von zuständigen Servicemitarbeitern ausgeführt werden. Zusätzlich werden Vor- und Nachbedingungen abstrahiert, die von den Aktivitäten an einem Servicepoint erwartet bzw. hinterlassen werden. Beziehungen, die neben dem eigentlichen Serviceflow zwischen den Servicepoints existieren werden mittels Supportprozessen abgebildet. Zur Modellierung werden eine Reihe etablierter Modellierungstechniken aus dem UML-Umfeld und der Kooperationsmodellierung eingesetzt.

Konkrete Prozessrepräsentationen werden im SfM-Ansatz als Servicefloats und Servicepointscripts bezeichnet. Anhand von UseCases wurden die zur Entwicklung einer Softwareunterstützung benötigten Umgangsformen mit den Prozessrepräsentationen an Servicepoints abgeleitet. Abschließend sind Bausteine genannt worden, aus denen sich die Prozessrepräsentationen zusammensetzen sollten, damit der vorgesehene Umgang realisiert werden kann. Die Umgangsformen und die Elemente der Prozessrepräsentationen dienten als Ausgangslage zur Realisierung der fachlichen Funktionalität des in Abschnitt 3.6 vorgestellten funktionalen Prototypen (vgl. Abschnitt 3.1.2).

Im nächsten Abschnitt werden die zur Realisierung der Prozessunterstützung an einem Servicepoint eingesetzten Technologien Java und XML vorgestellt. Unter Beachtung der in Kapitel 3.1.1 vorgestellten anzustrebenden Ebenenarchitektur sind diese Technologien auf den unteren beiden Ebenen einzuordnen, d.h. zur Realisierung der Persistenzebene und der Anwendungslogik.

Aspekte, welche die Darstellung und Handhabung der Prozessrepräsentationen im Kontext des Bürgerprozessportals www.hamburg.de und die Integration der Funktionskomponenten betreffen, werden im darauffolgenden Kapitel 3.4 diskutiert.

3.4 XML und Java

Der Werbespruch der Firma Sun „Java + XML = Portable Code + Portable Data“ [McL00, S. 1] deutet schon an, dass der größte Vorteil, der aus der Kombination der Programmiersprache Java mit der Datenbeschreibungssprache XML entsteht, die vollständige Portabilität auf diesen Technologien basierender Anwendungen darstellt. Java-Anwendungen können überall dort ausgeführt werden, wo eine virtuelle Maschine installiert ist und mit Hilfe von XML können die von den Anwendungen benötigten Daten plattformneutral repräsentiert werden. (vgl. [McL00, S. 15])

Im Kontext von SfM ist dieser Aspekt von besonderer Bedeutung. Wie weiter oben schon dargestellt, ist die zentrale Idee zur Unterstützung von Kooperation und Koordination über Organisationsgrenzen hinweg, Servicefloats zwischen den Organisationseinheiten auszutauschen, die den Stand der gemeinsamen Leistungserbringung dokumentieren. Da damit zu rechnen ist, dass die Prozessteilnehmer in den seltensten Fällen die gleiche Hard- und Software verwenden werden und ihre Systeme (z.B. aus Sicherheitsgründen) geschlossen halten wollen, ist die Verwendung von plattformunabhängigen und keine dauerhaften Verbindungen erfordernder Technologien zur Repräsentation der Servicefloats von hoher Priorität. XML empfiehlt sich somit als Datenaustauschformat in diesem Kontext (vgl. [McL00, S. 20]). Da sich Exemplare von Servicefloats und Servicepointscripts, wie am Ende des vorherigen Abschnitts aufgezeigt, durch eine identische logische Struktur auszeichnen, sind die erforderlichen Umgangsformen mit den Exemplaren an jedem Servicepoint gleich. Somit könnte, sofern gewollt, eine in Java programmierte Anwendung an alle am Prozess teilnehmenden Organisationseinheiten verteilt werden und es müsste lediglich das Vorhandensein einer virtuellen Maschine sichergestellt werden, um am Serviceflow teilzunehmen.

Die folgende Untersuchung erfolgt v.a. im Hinblick darauf, ein grundlegendes Verständnis von XML als Datenbeschreibungssprache und den grundlegenden Optionen der Manipulation von XML-Dokumenten mit Java zu vermitteln. Aufgrund des beschränkten Rahmens dieser Arbeit kann nur auf einen kleinen Teil der prinzipiellen Möglichkeiten, die für die Kombination der beiden Technologien mittlerweile angeboten wird, eingegangen werden.

3.4.1 XML-Grundlagen

XML steht für *eXtensible Markup Language*. Unter Markup versteht man laut [And00, S.34] folgendes:

> *„Markup ist eine Methode zur Übermittlung von Metadaten, d.h. Daten über Daten. Auszeichnungssprachen benutzen Literale oder sogenannte Tags, um Metadaten zu beschreiben und vom eigentlichen Inhalt zu trennen.“*

Ein berühmtes Beispiel für eine Auszeichnungssprache ist HTML, die zur Beschreibung von HTML-Dokumenten eingesetzt wird, um dem Browser Informationen darüber zu vermitteln, wie die Inhalte der Dokumente darzustellen sind. Die Semantik der Metadaten und deren Umfang ist im Falle von HTML vom W3C (World Wide Web Consortium) festgelegt. Dies ist bei XML allerdings nicht so. Mit XML hat man die Möglichkeit, eigene Tags zu definieren und deren Semantik selbst zu bestimmen.

Bei XML handelt es sich um eine Auszeichnungssprache, die einen Mechanismus bereithält, mit dem man neue Auszeichnungssprachen anwendungsspezifisch definieren kann. Tolksdorf schreibt z.B. hierzu:

> *„XML ist (...) eine standardisierte* (vom W3C; A. d. Autors) *Sprache in der sich die Syntax von Auszeichnungssprachen notieren lässt. Formaler ausgedrückt ist*

XML eine Metagrammatik für kontextfreie Grammatiken – also die Grammatik einer Sprache mit der sich die Regeln von Grammatiken notieren lassen.“[Tol99]

Grammatikdefinitionen werden im XML-Umfeld *DTD (Document Type Definition)* genannt[64]. DTDs dienen dazu, den Typ von XML-Dokumenten festzulegen. Sie bestimmen, wie mögliche Sätze der festgelegten Grammatik aussehen dürfen und legen so die Struktur der zur jeweiligen DTD konformen Dokumente fest. Beispiele für standardisierte DTDs sind XHTML, die eine DTD für HTML darstellt oder MathML, die der Beschreibung mathematischer Ausdrücke dient (vgl. z.B. [And00]). Der Zusammenhang zwischen XML, verschiedenen DTDs und zu diesen passenden XML-Dokumenten wird in Abbildung 16 illustriert.

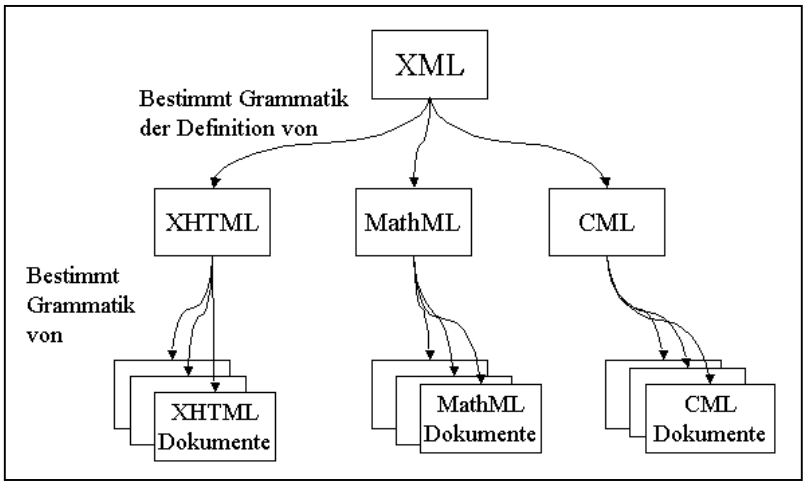

Abbildung 16: XML als Metagrammatik für Auszeichnungssprachen [Tol99]

DTDs legen das Vokabular fest, mit deren Hilfe die Inhalte von XML-Dokumenten beschrieben werden können. Durch DTDs wird im Rahmen von XML ein standardisierter Mechanismus zur Definition neuer anwendungsspezifischer Tags und deren Anwendung vorgegeben. Dieser Mechanismus wird durch das Wort „eXtensible“ in XML betont (vgl. [And00, S. 40 ff.])

DTDs bestehen aus *Elementtypen*, deren *Attributen* und *Regeln*, welche die möglichen Kombinationen der Elemente des Dokumenttyps festlegen. Zusätzlich werden *Entitäten* verwendet. (siehe z.B. [GolPre99, S. 63])

Elementtypen definieren einerseits Terminale der beschriebenen Sprache, die in XML-Dokumenten als Tags auftreten und legen gleichzeitig Produktionen als Regeln für den von ihnen umschlossenen Inhalt fest (vgl. [Tol99]). Anders ausgedrückt bestimmen Elementtypen neben dem Elementnamen, der später als Tag im XML-Dokument verwendet wird, auch das sogenannte *Inhaltsmodell* des Elements (vgl. [And00, S. 98 ff.]). Im Inhaltsmodell wird die Verschachtelung der Elementtypen festgelegt, woraus sich die logische Struktur der zur DTD konformen Dokumente ergibt. Ein Elementtyp kann wieder andere Elementtypen enthalten oder leer sein. Zur Spezifikation des Inhaltsmodells bietet XML verschiedene Operatoren an, welche die Beschreibung von Gruppierungen, Alternativen, Sequenzen und Kardinalität der Wiederholungen erlaubt.

Jedem Element können Attribute zugeordnet werden, die das Element genauer spezifizieren. Zu jedem Attribut kann ein *Modifikator* angegeben werden, der bestimmt, ob

[64] Das neuerdings verwendete XML-Schema, das ebenfalls der Definition von XML-Sprachen dient, wird in dieser Arbeit nicht behandelt, da es im Kontext der Realisierung des Anwendungsbeispiels keine Verwendung fand. Zu XML-Schema siehe z.B. [And00].

das Attribut immer gesetzt sein muss, der Wert unspezifiziert bleiben kann oder ob es immer einen vorgegebenen Wert besitzt. Zusätzlich wird zu jedem Attribut ein sogenannter *Attributtyp* angegeben, der den Datentyp des Attributes bestimmt, z.B. Zeichendaten (CDATA), eindeutige Elementkennzeichnungen (ID) und Referenzen auf diese (IDREF). Weitere Datentypen sind z.B. in [And00, S. 103 ff.] zu finden.

Entities können als Ersetzungstexte verstanden werden, die es erlauben, einmal definierte Textpassagen an beliebigen Stellen im XML-Dokument oder der DTD wiederzuverwenden. Dieser Mechanismus ist vergleichbar mit Text-Makros, wie sie z.B. in C++ verwendet werden (vgl. [And00, S. 95]). Der Mechanismus wird durch Einsatz von Kürzeln realisiert, die dann expandiert werden. Der Ersetzungstext muss nicht in der jeweiligen Datei selbst stehen, sondern kann auch Bestandteil einer anderen Datei oder diese Datei selbst sein. Deshalb unterscheidet man verschiedene Kategorien von Entities, die aus Platzgründen an dieser Stelle nur genannt werden können: Interne (gleiche Datei) und externe (andere Datei), geparste (Text) und nicht geparste (z.B. Bilder), allgemeine (nur XML-Dokument) und Parameter-Entities (nur DTD). Für eine tiefergehende Beschreibung sei auf die Literatur verwiesen, z.B. [And00], [Eck00] oder [PreGol99].

Es sind noch zwei wesentliche Konzepte anzusprechen, die im Zusammenhang mit XML immer wieder auftauchen und auch in dem in dieser Arbeit untersuchten Zusammenhang von Bedeutung sind. Es handelt sich hierbei um *Wohlgeformtheit* und *Gültigkeit* von XML-Dokumenten.

Ein XML-Dokument ist wohlgeformt (well formed), wenn es der XML Spezifikation [BPSM00] und der darin festgelegten Syntax entspricht. Dies bedeutet v.a., dass die Elemente des Dokuments korrekt geschachtelt sind, d.h. eine logische Baumstruktur bilden. Zu jedem öffnenden Tag muss es ein korrespondierendes schließendes Tag geben. (vgl. z.B. [And00, S. 81 ff.]) Ein XML-Dokument ist gültig (valid), wenn es wohlgeformt ist und zusätzlich den Regeln entspricht, die durch eine DTD vorgegeben wird (vgl. z.B. [Eck00, S. 28]).

Validität spielt insbesondere in organisationsübergreifenden Abläufen eine Rolle. Wenn sich alle beteiligten Einheiten auf eine bestimmte Form von Prozessrepräsentation einigen, deren Prozessexemplare als XML-Dokumente zwischen den Prozessteilnehmern ausgetauscht werden, dann können sich die XML verarbeitenden Anwendungen auf die Einhaltung der in der DTD formulierten Regeln verlassen. Nicht gültige XML-Dokumente können so von vorneherein abgelehnt oder an ausgezeichnete Stellen zur Sonderbehandlung delegiert werden.

Auch im SfM-Ansatz wird diese Eigenschaft von XML ausgenutzt. Die Struktur von Servicefloats und Servicepointscripts wird mit Hilfe von DTDs beschrieben. Die Vorlagen zur Erzeugung von konkreten Exemplaren der jeweiligen Prozessrepräsentationen sind gültige XML-Dokumente, die kopiert und personalisiert werden. Diese Kopien werden dann von Servicepoint zu Servicepoint geschickt und dokumentieren den Zustand der kooperativen Leistungserbringung im Rahmen des Serviceflows. Die benötigten DTDs sind an jedem Servicepoint vorhanden. (vgl. Abschnitt 3.1)

Abbildung 17 präsentiert zur Verdeutlichung des Zusammenhangs zwischen DTD und XML einen Ausschnitt aus der DTD des Servicefloats und eines korrespondierenden XML-Dokuments, die als eine der Grundlagen der Implementation des funktionalen Prototypen dienten (vgl. Abschnitt 3.1). Die Semantik der Elemente ist durch Kommentare oberhalb des jeweiligen Elements erklärt.

Um an jedem Servicepoint die relevanten Informationen eintragen bzw. auslesen zu können und eine Zustandsveränderung zu initiieren, bedarf es einer angemessenen Softwareunterstützung, die dies anwendungsorientiert unter Beachtung der in Abschnitt 3.3.2 aufgeführten Umgangsformen realisiert. Im nächsten Abschnitt werden die grundlegenden

Möglichkeiten vorgestellt, die Java zum Umgang mit XML-Dokumenten bereithält und auf deren Basis eine Anwendung zur Prozessmanipulation und Verwaltung umgesetzt werden kann.

```
DTD:
<!-- servicefloat beschreibt das Dokument-Element, also das Eltern-Element aller anderen Elemente in zu
     dieser DTD konformen Dokumente; das Inhaltsmodell des servicefloat-Elements ist eine Sequenz, die alle
     Elemente in der angegebenen Reihenfolge beinhalten muss -->
<!ELEMENT servicefloat (sf_type, sf_name, current_servicepoint,
                        schedule, history, postcondition_list, document_list)>
<!-- dem servicefloat-Element muss ein Attribut sf_id zugeordnet sein -->
<!ATTLIST servicefloat:servicefloat
     sf_id ID #REQUIRED>
<!-- das Inhaltsmodell der Elemente sf_type und sf_name besteht aus parsed chracter data (PCDATA)
     dies bedeutet, dass alle Zeichendaten, die kein Element sind, zwischen den Element-Tags auftauchen
     dürfen -->
<!ELEMENT sf_type (#PCDATA)>
<!ELEMENT sf_name (#PCDATA)>
<!-- das Element current_servicepoint beeinhaltet optional ein oder kein weiteres Element namens
     servicepoint, was durch das ? spezifiziert wird -->
<!ELEMENT current_servicepoint (servicepoint?)>
...
<!ELEMENT servicepoint (sp_type, sp_name, provider*, sp_xml_address)>
<!ATTLIST servicepoint
     sp_id ID #REQUIRED>
<!ELEMENT sp_type (#PCDATA)>
<!ELEMENT sp_name (#PCDATA)>
<!-- die Definition des Elements provider wird aus einer anderen DTD importiert  (externe Parameter-Entity);
     es können beliebig viele Provider-Elemente vorhanden sein (0..*) -->
<!ENTITY % provider.dtd SYSTEM "provider.dtd">
<!ELEMENT sf_xml_address (#PCDATA)>
...
-------------------------------------------------------------------------------------------------------------------------------------------
XML-Dokument:
<!-- XML-Deklaration mit Kodierungsanweisung -->
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- DOCTYPE legt fest, dass das XML-Dokument den Sprachumfang servicefloat, also den
     Dokumenttyp servicefloat hat; zusätzlich wird angegeben, wo sich die dazugehörige DTD befindet-->
<!DOCTYPE servicefloat SYSTEM "http://swt5.informatik.uni-hamburg.de:8800/DTDs/servicefloat.dtd">
<!-- servicefloat ist das Dokument-Element -->
<servicefloat sf_id="sf.0074711">
<sf_type>Briefwahlantrag2001</sf_type>
<sf_name>Briefwahlantrag.hh.de</sf_name>
...
<current_servicepoint>
     <servicepoint sp_id="sps.008998">
          <sp_type>Self Servicepoint</sp_type>
          <sp_name>Antragsunterstützung Briefwahl hh.de</sp_name>
          <provider> ... </provider>
          <sp_xml_address>www.hamburg.de</sp_xml_address>
     </servicepoint>
</current_servicepoint>
...
<servicefloat>
```

Abbildung 17: Ausschnitte aus der „servicefloat“-DTD und korrespondierendem XML-Dokument[65]

[65] Auf die Darstellung von Namensräumen (siehe z.B. [And00]), die bei den vom funktionalen Prototypen , der in Abschnitt 3.6 beschrieben wird, verwendeten Vorlagen Verwendung fanden, wurde hierbei aus Gründen der Übersichtlichkeit verzichtet.

3.4.2 Document Object Model: Java-API zur Manipulation von XML-Dokumenten

Mit der Programmiersprache Java hat man im Wesentlichen zwei Möglichkeiten, mit XML-Dokumenten umzugehen. Dies ist zum einen das *Simple API*[66] *for XML (SAX)* (siehe z.B. [And00, S. 197 ff.]) und zum anderen das *Document Object Model (DOM)* [Woo98]. Im Gegensatz zu SAX definiert das DOM einen vom W3C standardisierten Umgang mit XML-Dokumenten, SAX stellt kein Produkt eines Standardisierungsgremiums dar. Des weiteren weist DOM gegenüber SAX die Möglichkeit auf, die Inhalte der XML-Dokumente zu modifizieren; SAX erlaubt nur lesenden Zugriff (vgl. [And00, S. 201]). Im Hinblick auf die Realisierung des funktionalen Prototypen (siehe Abschnitt 3.6.3) ist DOM von größerer Bedeutung. Aus diesem Grund wird im folgenden nur auf diese API näher eingegangen. Für weitere Informationen zum Thema SAX sei auf die Literatur verweisen (z.B. [And00], [McL00]).

Das DOM spezifiziert eine standardisierte API für HTML- und XML-Dokumente. Es definiert die logische Struktur von Dokumenten und die Art und Weise, mit der man auf Dokumente zugreifen und sie manipulieren kann. (vgl. [Woo98]) Beim DOM handelt es sich um eine programmiersprachen- und plattformunabhängige Spezifikation. Es gibt Implementationen in einer Vielzahl von Programmier- und Scriptsprachen, wie z.B. Perl, JavaScript, C++, CORBA, und eben auch Java (vgl. [McL00, S. 178]).

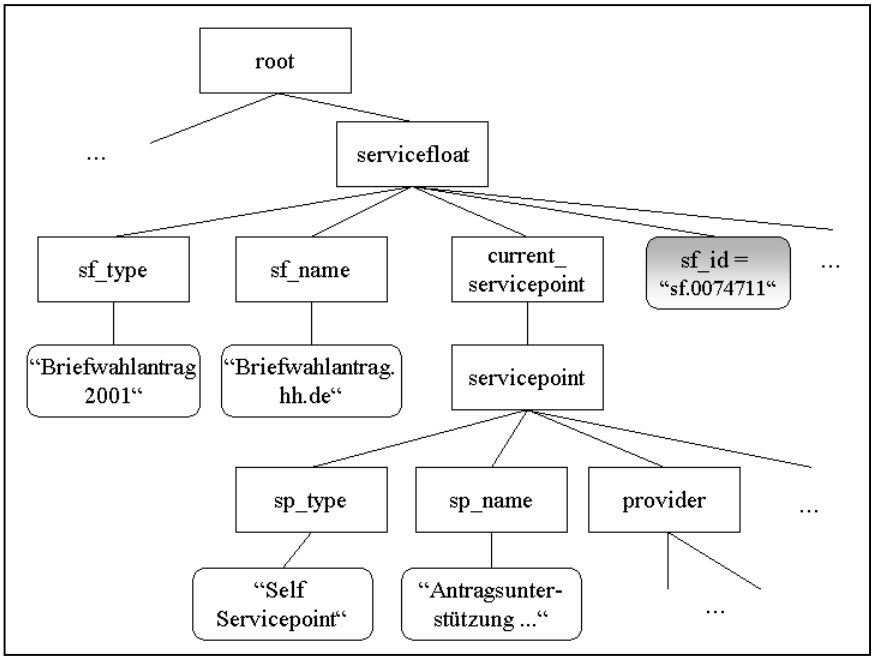

Abbildung 18: DOM-Strukturmodell

Im DOM werden Dokumente auf eine baumartige Struktur abgebildet, die stark mit der logischen Struktur wohlgeformter XML-Dokumente korrespondiert (vgl. Abschnitt 3.4.1). Wie der Name schon andeutet, handelt es sich hierbei um ein Model, das dem Paradigma der Objektorientierung folgt. Dokumente werden als Bäume von Objekten modelliert, deren Knoten die Elemente des korrespondierenden XML-Dokuments repräsentieren. Das DOM spezifiziert, welche Schnittstellen zur Repräsentation und Manipulation eines Dokumentes verwendet werden, deren Semantik und die Beziehungen zwischen diesen Schnittstellen (vgl. [Woo98]). Abbildung 18 stellt ein DOM-Strukturmodell des Beispiel-Dokumentes aus

[66] API steht für Application Programming Interface (vgl. z.B. [And00, S. 169]).

Abschnitt 3.4.1 dar. „root“ bezeichnet dabei die Dokument-Wurzel (Typ: Document; siehe unten), die dem Dokument-Element „servicefloat“ (Typ: Element; siehe unten) übergeordnet ist. Die Dokument-Wurzel ermöglicht neben dem Zugriff auf den eigentlichen Inhalt eines XML-Dokuments beispielsweise auch den Zugriff auf die dazugehörige DTD.

Die DOM-Implementationen für Java befinden sich standardmäßig im Package[67] `org.w3c.dom` (vgl. [McL00]). Eine Implementation wird meist mit einem sogenannten *Parser* ausgeliefert. Parser[68] sind Programme, mit deren Hilfe XML-Dokumente gelesen und z.B. in das DOM überführt werden können[69]. Grundsätzlich kann zwischen validierenden und nicht validierenden Parsern unterschieden werden. Nicht validierende Parser überprüfen lediglich, ob ein XML-Dokument wohlgeformt ist, validierende prüfen zusätzlich, ob es der korrespondierenden DTD entspricht. Das Auftreten ungültiger XML-Dokumente kann beispielsweise von validierenden Parsern über entsprechende Fehlermeldungen an die sie benutzende Programme mitgeteilt werden, so dass geeignet darauf reagiert werden kann[70]. (vgl. z.B. [And00, S. 82 ff.]) Nachfolgende Abbildung 19 stellt anhand eines Programmausschnitts beispielhaft dar, wie mit Hilfe eines DOM-Parsers ein DOM aus einem XML-File erstellt werden kann[71]. Auf die Darstellung der Ausnahmebehandlung (Exception-Handlings[72]) wurde hierbei verzichtet.

```
import javax.xml.parsers.*;
import org.w3c.dom.*;
import java.io.*;
…
public Document createDocumentFromXMLFile (File theFile) {
  // Um ein DOM zu erzeugen, braucht man einen DocumentBuilder. Diesen
  // erhält man über eine DocumentBuilderFactory, die zusätzlich
  // die Konfiguration der DocumentBuilder erlaubt, z.B. ob sie einen
  // validierenden Parser verwenden sollen. Nach Konfiguration und Erzeugung
  // wird der DocumentBuilder dazu aufgefordert, das übergebene File zu
  // parsen. Das Ergebnis dieses Vorgangs stellt ein Document (DOM) dar.
  DocumentBuilderFactory dbf =  DocumentBuilderFactory.newInstance();
  dbf.setValidating(true);
  DocumentBuilder builder = dbf.newDocumentBuilder();
  FileInputStream fis = new FileInputStream(theFile);
  Document doc = builder.parse(fis);
  return doc;
}
```

Abbildung 19: Codebeispiel: Erzeugung eines DOM aus einem XML-File

[67] Zum Package-Konzept von Java siehe z.B. [MidSin99, S. 86 ff.]

[68] Beispiele für Parser sind: Xerces (www.apache.org) oder IBM XML4J (www.ibm.com).

[69] SAX-Parser hingegen überführen ein Dokument nicht in eine bestimmte Darstellung, sondern bieten den sie nutzenden Programmen lediglich die Möglichkeit, sich für bestimmte Ereignisse, die beim Einlesen auftreten können, anzumelden, wie z.B. das Auftreten eines Elementes oder Attributes (vgl. [McL00, S. 49 ff.]).

[70] Diese Eigenschaft wurde bei der Realisierung des funktionalen Prototypen ausgenutzt. Es werden dort nur XML-Dokumente eingelesen, die der DTD entsprechen.

[71] Das Beispiel impliziert die Verwendung der Xerces 1.2.3 API.

[72] Zum Exception-Mechanismus von Java siehe z.B. [MidSin99, S. 135 ff.]

Abbildung 20 zeigt einen Ausschnitt wichtiger Schnittstellen des DOM-Packages und einen Teil der von ihnen spezifizierten Methoden. Für eine detailliertere Diskussion sei z.B. auf [McL00, S. 179 ff.] oder die API-Dokumentation eines Parsers verwiesen. Im folgenden wird kurz auf die dargestellten Schnittstellen eingegangen.

Nahezu alle Interfaces des DOM-Packages sind von `Node` abgeleitet. `Node` spezifiziert die wichtigsten Operationen, die zur Traversierung des DOM-Baumes und zu dessen Manipulation notwendig sind. Der konkrete Typ eines Knotens kann durch die Methode `getNodeType` an jedem Knoten abgefragt werden. `Element` als Spezialisierung von `Node` repräsentiert die Elemente und `Attr` die Attribute der Elemente. Attribute werden als Attribute im objektorientierten Sinne verstanden, d.h. sie sind keine Kindknoten ihrer Elemente sondern Elementbestandteile[73]. `Document` repräsentiert das gesamte XML-Dokument und fungiert konzeptuell als Wurzel des Baumes. Hier werden die Methoden spezifiziert, die z.B. zur Erzeugung neuer Elemente verwendet werden können. Jeder Knoten des DOM ist mit dem Dokument über ein Attribut assoziiert, in dessen Kontext er existiert.

`NamedNodeMap` und `NodeList` repräsentieren Sammlungen für Knoten. Diese Sammlungen erhält man, wenn man sich z.B. alle Attribute eines Knotens geben lässt (Methode `getAttributes` im Interface `Node`) oder alle Kindelemente eines Elements, die einen bestimmten Tag-Namen haben (`getElementsByTagName` im Interface `Element`). Die Knoten innerhalb der Sammlungen können über einen Index zugegriffen werden.

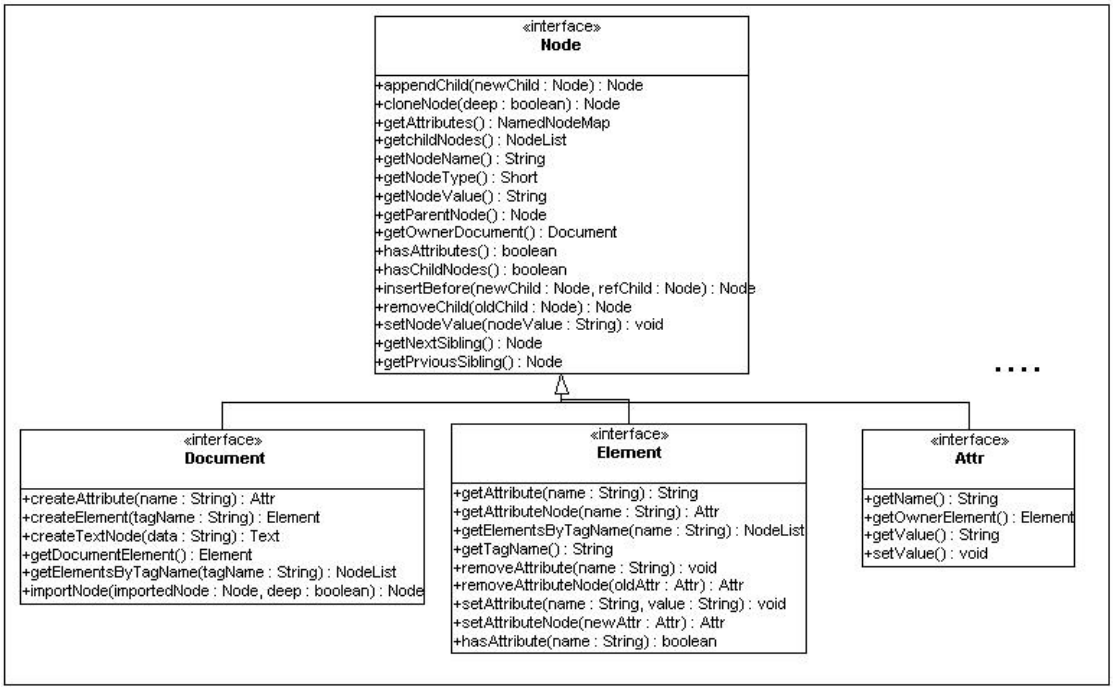

Abbildung 20: UML-Klassendiagramm: Ausschnitt der DOM-Schnittstellen Node, Document, Element und Attr

Grundsätzlich sind folgende Operationensarten auf dem DOM möglich (vgl. auch Abbildung 20):

- *Lesen der Inhalte*: Die Werte, die den einzelnen Knoten zugeordnet sind, können einfach mit sogenannten „getter"-Methoden, wie sie z.B. im `Node`-Interface spezifiziert sind, ausgelesen werden. Weiterhin können Zusatzinformationen wie z.B. der Name eines Knotens festgestellt werden. Außerdem ist es möglich, sich alle Kindknoten eines

[73] Deswegen auch die besondere Darstellung des Attributes „c_id" in Abbildung 18.

Elements bzw. dessen Attribute geben zu lassen und darauf weitere Operationen anzuwenden. Methoden, mit denen das Vorhandensein von Attributen oder Kindelementen geprüft werden kann, sind auch verfügbar. Werte von Elementen bzw. Attributen sind grundsätzlich vom Typ `java.lang.String`. Benutzerdefinierte Datentypen, wie z.B. ein Datum, sind auf dieser Ebene nicht verfügbar.

- *Ändern der Inhalte*: Knoten eines DOM sind im Hinblick auf ihren Inhalt modifizierbar. Hierzu gibt es einfache „setter"-Methoden. Zusätzlich können weitere Knoten und Attribute angehängt bzw. gelöscht werden.
- *Manipulieren der Struktur*: Es können neue Knoten bestimmter Typen erzeugt und dem DOM hinzugefügt, entfernt oder durch andere ersetzt werden. Es ist weiterhin möglich, Knoten bzw. auch ganze Teilbäume innerhalb der Struktur umzuordnen. Zusätzlich sind Knoten eines DOM in ein anderes DOM übertragbar. Hierzu muss lediglich der Kontext-Owner (die Referenz auf `Document`; siehe oben) umgesetzt und der Knoten importiert werden.

Neben dem Standard-Package werden oft eine Reihe weiterer Packages mit den Parsern ausgeliefert, die einen komfortableren Umgang mit XML-Dokumenten erlauben. Außerdem wird der Standard ständig erweitert. Die Standardisierung erfolgt schrittweise in sogenannten Levels. Zum Zeitpunkt der Niederschrift der Arbeit ist DOM Level 2 bereits verabschiedet.

3.4.3 Zusammenfassung und Ausblick

In diesem Kapitel wurde grundlegend dargestellt, was XML ist und welche Basisoptionen im Rahmen der Programmiersprache Java zur Manipulation von XML-Dokumenten angeboten werden.

Mit XML ist es möglich, beliebige auf den jeweiligen Anwendungskontext zugeschnittene Auszeichnungssprachen zu definieren, die dann zur Erzeugung der anwendungsspezifischen Dokumente verwendet werden können. Mit Hilfe von Parsern können die Dokumente eingelesen und gleichzeitig auf ihre Wohlgeformtheit und Validität geprüft werden. DOM-Parser transformieren die XML-Dokumente in eine mit der logischen Dokumentenstruktur korrespondierende baumartige Darstellung, die dann über standardisierte APIs traversiert und manipuliert werden kann. So ist es möglich, Inhalte auszulesen, zu verändern und auch die Struktur zu manipulieren.

Die Kombination von Java und XML scheint also eine gute Wahl, als Basistechnologie im untersuchten Anwendungskontext eingesetzt zu werden. Im Hinblick auf die Definition von organisationsübergreifenden Prozessen nach dem SfM-Ansatz auf der Basis von XML ist lediglich sicherzustellen, dass die Prozessbeteiligten sich auf eine gemeinsame DTD einigen und diese an jedem Servicepoint verfügbar ist.

Für den Prototyp, der in Abschnitt 3.6 vorgestellt wird, wird davon ausgegangen, dass der Servicepoint im Rahmen der Bürgerprozessportals www.hamburg.de eingesetzt wird (vgl. Abschnitt 3.1). Die zur Realisierung des Portals eingesetzte Technologie ist das Web Content Management System StoryServer 5.0 der Firma Vignette. Im nächsten Abschnitt wird daher darauf eingegangen, welche grundlegenden Konzepte von diesem Produkt zur Erzeugung von HTML-Seiten und zur Realisierung der Handhabung und Integration eines in Java programmierten Servicepoints angeboten werden.

3.5 StoryServer 5.0 und Java

Der StoryServer 5.0 der Firma Vignette ist ein Web Content Management System (WCMS), das sich v.a. durch seine hohe Leistungsfähigkeit auszeichnet. Unter dem Begriff WCMS werden Systeme verstanden, die den Inhalt (Content) von Webseiten, wie Texte, Bilder, Videos, aber auch anwendungsgebundene und transaktionsbezogene Daten verwalten (Management) (vgl. z.B. [BTZZ00, S. 99 ff.]). Es können sowohl statische als auch dynamische Inhalte gehandhabt werden. Zusätzlich wird die Möglichkeit angeboten, Workflows zu definieren und auszuführen, welche die notwendigen Schritte zur Erstellung, Bearbeitung, Verwaltung, Publikation und Archivierung von Websites betreffen. Ohne den Einsatz von WCMS sind große Websites wie Stadtinformationssysteme oder Internet-Portale kaum mehr zu handhaben. Die wichtigste Eigenschaft von WCMS ist, dass sie die Trennung von Inhalt und Form ermöglichen, d.h. das Layout der Webseiten kann unabhängig von den eigentlichen Inhalten produziert und verwaltet werden. Hierzu werden sogenannte *Templates* eingesetzt, die anhand des vom StoryServer angebotenen Template-Konzeptes nachfolgend vorgestellt werden.

Der StoryServer 5.0 selber bietet neben den Grundfunktionen eines WCMS auch noch weitere Möglichkeiten, die z.B. unter die Schlagworte Customer Relationship Management, Personalisierung und User-Profiling fallen. Zur Überwindung der Zustandslosigkeit des HTT-Protokolls (vgl. Abschnitt 3.2.1) werden URL-Erweiterungen und die Verwendung von Cookies angeboten. Außerdem kann durch Benutzung eines StoryServer-eigenen Hyperlink-Verwaltungskonzeptes vermieden werden, dass Verweise auf nicht existierende Seiten entstehen können. Diese speziellen Referenzen werden als *CURLs (Customized URLs)* bezeichnet. Auf diese Punkte soll im weiteren allerdings nicht näher eingegangen werden und es sei auf die Dokumentation z.B. [Vig99b] und [Vig99c] verwiesen.

Zusätzlich existieren Schnittstellen zu anderen Systemen, beispielsweise zu Datenbanken, C++-Programmen über eine API und zu Java über ein spezielles Produkt namens TclBlend. Da die Integration mit Java im Kontext dieser Arbeit von Relevanz ist, wird nur dieser Aspekt näher untersucht.

3.5.1 Das Template-Konzept des StoryServers

Templates (Schablonen) stellen im StoryServer das Konzept dar, mit dessen Hilfe die Form der Webseiten beschrieben werden kann. Eine Webseite kann sich grundsätzlich aus mehreren ineinander geschachtelten Templates zusammensetzen. Um dies zu realisieren wird ein bestimmter Templatetyp angeboten, der als *Komponente (Component)* bezeichnet wird. Templates dieser Art können an beliebiger Stelle in beliebigen anderen Templates importiert werden und selber wieder Templates enthalten. Ein wesentlicher Vorteil, der aus ihrer Verwendung entsteht ist, dass sich im Falle einer Änderung der Komponente diese Modifikation an allen Stellen bemerkbar macht, an denen sie verwendet wird.

Der StoryServer unterscheidet grundsätzlich noch weitere Templatetypen, die hauptsächlich jedoch nur zur Orientierung dienen und im Hinblick auf technologische Besonderheiten keine neuen Gesichtspunkte aufzeigen. Somit wird hier aus Platzgründen auf eine detailliertere Beschreibung verzichtet und es sei z.B. auf [Vig99b] verwiesen.

Neben der Beschreibung der Form der Webseiten ist in Templates auch die Formulierung von Programmlogik möglich. Somit könnten Templates z.B. nur dazu benutzt werden Berechnungen durchzuführen, Datenbanken zu aktualisieren oder das Layout zu bestimmen. Eine Vermischung von Programmlogik und Layout ist ebenfalls möglich. Die eingebetteten Anweisungen werden bei der Seitenerzeugung vom StoryServer ausgewertet und das Resultat in der vorgesehenen Stelle eingetragen. Der StoryServer verwendet somit serverseitige

Skripte zur Realisierung einfacher Webanwendungen (vgl. Abschnitt 3.2.1). Als Skriptsprache wird *Tcl (Tool Command Language)* verwendet (zu Tcl siehe z.B. [Ost93]). Abbildung 21 zeigt ein Beispiel, bei dem Tcl-Anweisungen in die Definition einer HTML-Tabelle eingefügt wurden und präsentiert in der unteren Hälfte das Resultat der Auswertung des markierten Abschnitts. Tcl-Kommandos werden durch öffnende und schließende eckige Klammern vom Markup getrennt. Der StoryServer verwendet eigene Tcl-Kommandos, die dadurch erkannt werden können, dass sie im Gegensatz zu Standard-Tcl-Anweisungen groß geschrieben werden.

Da es auch möglich ist, Templates zu formulieren, die keinerlei Präsentationsaufgaben übernehmen, können z.B. komplexe Verzweigungsalgorithmen oder Berechungen getrennt von den Präsentationstemplates entwickelt und gepflegt werden. Dies führt dazu, dass Templates übersichtlicher und verständlicher gestaltet werden können. Die Notwendigkeit, Ablaufinformationen (welches Template ist als nächstes aufzurufen) und erforderliche Parameter in Templates bereitzustellen, die eigentlich nur eine Präsentationsaufgabe wahrnehmen, kann auch im StoryServer nicht umgangen werden, da die Basistechnologie weiterhin HTML darstellt und somit die in Abschnitt 3.2.1 genannten Einschränkungen auch hier Gültigkeit besitzen.

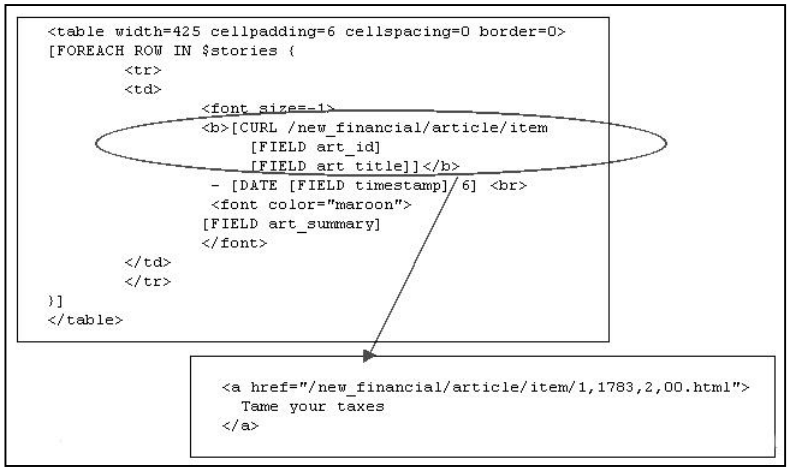

Abbildung 21: Beispiel der Einbettung von Tcl-Anweisungen in ein StoryServer Template mit Auswertung

3.5.2 TclBlend und der StoryServer

TclBlend (siehe z.B. [Bal99]) ist ein Produkt, das grundsätzlich unabhängig vom StoryServer existiert und mit dem Ziel entwickelt wurde, eine Brücke zwischen Tcl-Programmen und Java-Anwendungen herzustellen. Bei Tcl selber handelt es sich um eine interpretierte Skriptsprache, die hauptsächlich dazu eingesetzt wird, Programme, die in verschiedenen Programmiersprachen realisiert worden sind, zusammenarbeiten zu lassen. Man spricht in diesem Zusammenhang auch von „Glue-Sprachen“ (vgl. [Ost93]). Somit ist es nur natürlich, dass auch für Java eine entsprechende Verbindung existiert. Die Brücke wird technisch so umgesetzt, dass TclBlend eine dynamisch ladbare Erweiterung für den Tcl-Interpreter darstellt, der normalerweise in C realisiert ist, und einem Tcl-Interpreter so die dynamische Initialisierung der virtuellen Maschine von Java und den Zugriff auf dort enthaltene Objekte und Klassen erlaubt (vgl. [Bal99, S. 292]). Der StoryServer benutzt TclBlend, um so auch in Java realisierte, komplexere Anwendungsfunktionalität einbinden zu können und über die von ihm generierte Benutzungsoberfläche zugänglich zu machen.

Tcl bietet ein Namensraumkonzept an, in dem zusammengehörige Kommandos gruppiert werden können (vgl. [Bal99]). Wird TclBlend verwendet, so steht ein eigener Namensraum für Java-Kommandos zur Verfügung. Die Java-Kommandos bilden die Standard-Java-

Operatoren wie z.B. `new` oder `instanceof` ab und werden in der Art „java::new signature *arg arg ...*“ (z.B. java::new service2000.Servicefloat “cId007”) verwendet. Sobald ein neues Exemplar einer Java-Klasse erzeugt wird, generiert TclBlend automatisch ein Tcl-Kommando, das ein sog. „Object Handle“ auf das Java-Objekt, das sich in der virtuellen Maschine befindet, darstellt. Das neue Kommando trägt einen eindeutigen Bezeichner, der das neue Exemplar identifiziert. Dieser eindeutige Identifikator wird einer Tcl-Variablen als Wert zugeordnet und stellt somit eine Referenz auf das „Object Handle“ dar. Es muss immer mindestens eine Referenz auf ein Java-Objekt existieren, da es sonst vom Garbage-Collektor gelöscht werden würde. Deswegen ist die Erzeugung einer Tcl-Variable notwendig. Über die Variable kann dann wie in Abbildung 22 dargestellt in StoryServer-Templates auf Java-Objekte zugegriffen und öffentliche Methoden an ihnen aufgerufen werden.

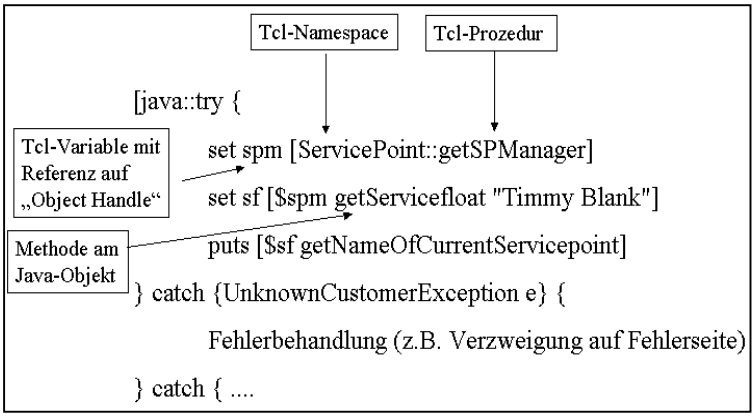

Abbildung 22: TCL-Blend Codebeispiel

Im Hinblick auf die Benutzung eines Java-Objektes über TclBlend im Rahmen des StoryServers ist noch eine wesentliche Besonderheit des StoryServers zu beachten[74]. Der StoryServer benutzt zur Auswertung der Tcl-Kommandos in den Templates einen Tcl-Interpreter, der grundsätzlich zwei verschiedene Namensräume kennt (vgl. [Vig99b]). Einen sogenannten „Global Namespace“ und einen sogenannten „VgnDefaultNamespace“. In letzteren werden alle Prozeduren und Variablen abgelegt, die innerhalb eines Templates benutzt werden. Im globalen Namensraum befinden sich alle StoryServer-eigenen Kommandos. Das besondere ist hierbei, dass der „VgnDefaultNamespace“ nach jeder Seitengenerierung gelöscht wird, was beim globalen Namensraum nicht der Fall ist. Möchte man also Variablen oder eben auch Referenzen auf Java-Objekte über mehrere Seiten hinweg halten, so müsste man sie im globalen Namensraum ablegen. Da dieser jedoch für den StoryServer reserviert ist und im globalen Namensraum angelegte Variablen ebenfalls nach Auswertung des jeweiligen Templates gelöscht werden, muss man sich zu diesem Zwecke einen eigenen Namensraum erzeugen. Dieser wird nicht gelöscht und somit können dort Variablen über einen beliebigen Zeitraum gehalten werden. Im Codebeispiel ist dieser Zusammenhang an dem selbstdefinierten Namensraum `ServicePoint` erkennbar. `getSPManager` repräsentiert eine Prozedur, die den Zugriff auf eine dort abgelegte Variable erlaubt.

Abbildung 23 stellt die Beziehungen der Namensräume zueinander und die Verbindung zur virtuellen Maschine von Java über TclBlend noch mal grafisch dar.

[74] Für eine weitere Besonderheit des StoryServers im Hinblick auf die Erzeugung von HTML-Seiten aus Templates in Verbindung mit Java siehe Abschnitt 3.7.2.

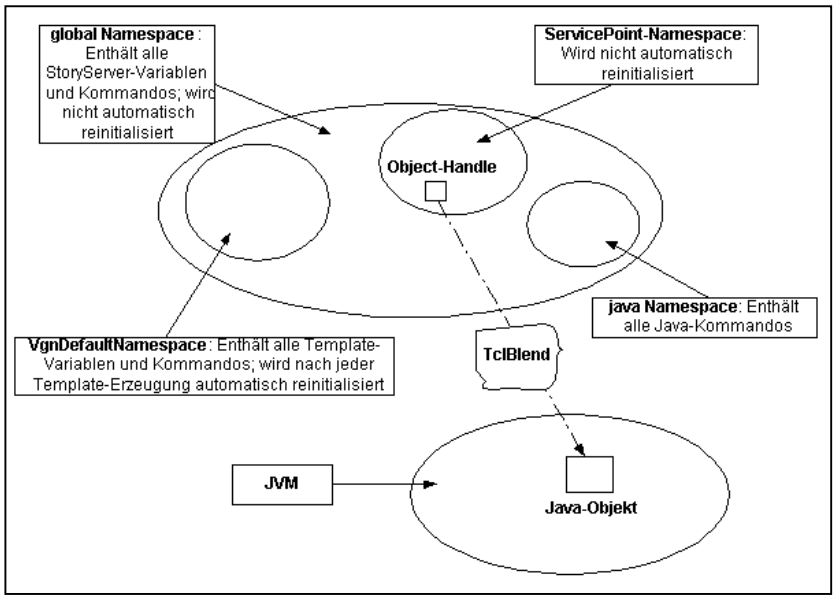

Abbildung 23: Namensräume im StoryServer im Zusammenhang mit TclBlend

3.5.3 Zusammenfassung und Ausblick

In diesem Kapitel wurde zunächst kurz dargestellt, was WCMS sind und wodurch sie sich hauptsächlich auszeichnen. Anschließend wurde am Beispiel des StoryServers das in WCMS verfolgte Template-Konzept zur Beschreibung von HTML-Seiten, die sowohl dynamische als auch statische Inhalte enthalten können, erläutert. Hierbei ist insbesondere das Komponenten-Konzept des StoryServers hervorgehoben worden, dass es ermöglicht, Templates ineinander zu schachteln und Änderungen sofort an allen Stellen sichtbar werden zu lassen, an denen die Komponente Verwendung findet.

Im Rahmen des StoryServers 5.0 wird die Skript-Sprache Tcl zur Einbettung von Programmlogik in Templates verwendet. Um auch auf die von Java-Programmen angebotene Funktionalität zugreifen zu können, muss das Produkt TclBlend eingesetzt werden, dass eine Brücke zwischen Tcl-Scripts und Java-Anwendungen darstellt. Neu erzeugte Java-Objekte können dann über Tcl-Variablen in den Templates zugegriffen werden. Sollen Java-Objekte über mehrere Seitenfolgen hinweg existieren, so ist das vom StoryServer verfolgte Namensraumkonzept zu beachten und ein eigener Namensraum für diese Objekte zu erzeugen.

Die Anbindung von Java ist also auf leichte Art und Weise möglich und somit kann der StoryServer zur Realisierung der Handhabung und Präsentation einer in Java programmierten fachlichen Funktionalität an einem webbasierten Servicepoint eingesetzt werden. Diese Möglichkeit wurde bei der Realisierung des funktionalen Prototypen genutzt, auf dessen konkrete Ausgestaltung im nachfolgenden Kapitel eingegangen wird.

3.6 Der funktionale Prototyp

Im diesem Kapitel wird der funktionale Prototyp[75] für eine Softwareunterstützung an einem webbasierten Servicepoint vorgestellt, der auf Basis der in den vorhergehenden Abschnitten vorgestellten Technologien realisiert wurde.

Zunächst wird eine Übersicht über die gewählte Makroarchitektur gegeben und die verwendeten Technologien eingeordnet. Hiernach fokussiert die Diskussion auf der Präsentation des Herzstückes der Implementation, das in Anlehnung an die Entwurfsmetapher des Fachlichen Service umgesetzt wurde. Dessen dynamisches Verhalten wird danach anhand eines Beispiels illustriert. Zum Abschluss wird auf die Umsetzung der Prozessrepräsentationen und des Interaktionsservices eingegangen.

Im Anhang dieser Arbeit findet sich eine Übersicht über die Schnittstellendefinitionen der im folgenden vorgestellten Komponenten. Aus diesem Grund werden in diesem Kapitel nur einige Ausschnitte der Schnittstellen präsentiert. Die Diskussion fokussiert auf einer allgemeinen Beschreibung der Komponenten.

3.6.1 Architekturübersicht

Die grundlegende Architektur des webbasierten Servicepoints wurde bereits in Abschnitt 3.1 vorgestellt. Abbildung 24 stellt die gewählte Architektur noch mal überblicksartig dar und benennt die in den einzelnen Schichten eingesetzten Technologien und zusätzlich die dort realisierten Aufgaben. Die drei unteren Schichten wurden vollständig auf einem Server umgesetzt, nur die Darstellung wird zum Client in Form von HTML-Seiten übertragen, die dort vom Web-Browser interpretiert und angezeigt werden. Als Web-Server wurde der Apache Web-Server[76] eingesetzt, der mit dem StoryServer zusammenarbeitet.

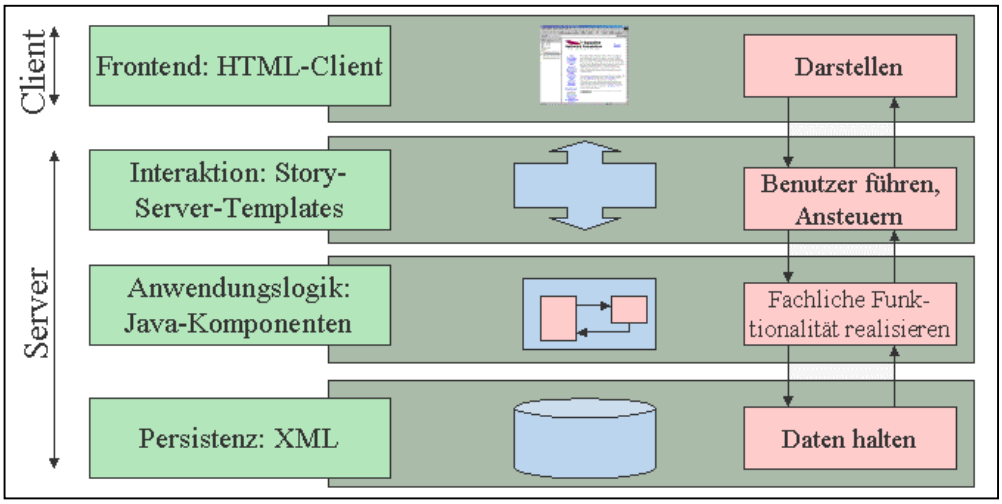

Abbildung 24: Schema der Servicepoint-Architektur

Die zweite Schicht wurde mit Hilfe des vom StoryServer angebotenen Template-Konzeptes realisiert (vgl. Abschnitt 3.5.1). Die hauptsächlichen Aufgaben dieser Schicht beziehen sich darauf, den Benutzer durch die Anwendung zu führen und die richtige Anwendungslogik der darrunterliegenden Schicht anzusteuern. Konkret bedeutet dies, die richtigen Folgen von HTML-Seiten aufgrund von Benutzeraktionen zu generieren (z.B. im Anwendungsbeispiel

[75] „Ein Prototyp ist eine spezielle Ausprägung eines ablauffähigen Softwaresystems. (...) Funktionale Prototypen modellieren in der Regel Ausschnitte aus der Benutzungsschnittstelle und Teile der Funktionalität.“ [RecPom97, S. 660].

[76] http://www.apache.org

die Auslieferung der Dankesseite aufgrund des Drückens des Absenden-Buttons; vgl. Kapitel 2.5.2) und das Wissen um die Benutzung der Anwendungslogik zu implementieren. Letztere ist in der dritten Ebene realisiert, welche grob aus einer Verwaltungseinheit, die als Servicepointmanager bezeichnet wird, und den Komponenten, welche den zustandsabfragenden und -verändernden Umgang mit den Prozessrepräsentationen Servicefloat und Servicepointscript implementieren, besteht. Die Verbindung zwischen StoryServer-Templates und der in Java implementierten Anwendungslogik wurde über das in Kapitel 3.5.2 vorgestellte Produkt TclBlend hergestellt. Als Persistenzformat wird XML, insbesondere aufgrund seiner Portabilitätseigenschaften, eingesetzt (vgl. Kapitel 3.4.1). Der Prototyp verwendet das Dateisystem des Servers, um die einzelnen XML-Dokumente (Servicefloats als auch Servicepointscripts) persistent zu halten. Ankommende und abgehende Servicefloats werden auch im Filesystem abgelegt, ebenso finden sich die Prozessvorlagen und DTDs auf diesem Medium. Hier ist natürlich der Einsatz anderer Persistenzmedien, wie z.B. relationaler oder XML-fähiger Datenbanken, denkbar. Abbildung 25 gibt eine Übersicht zum Zusammenspiel der benannten Komponenten, auch über die serverseitigen Ebenen hinweg.

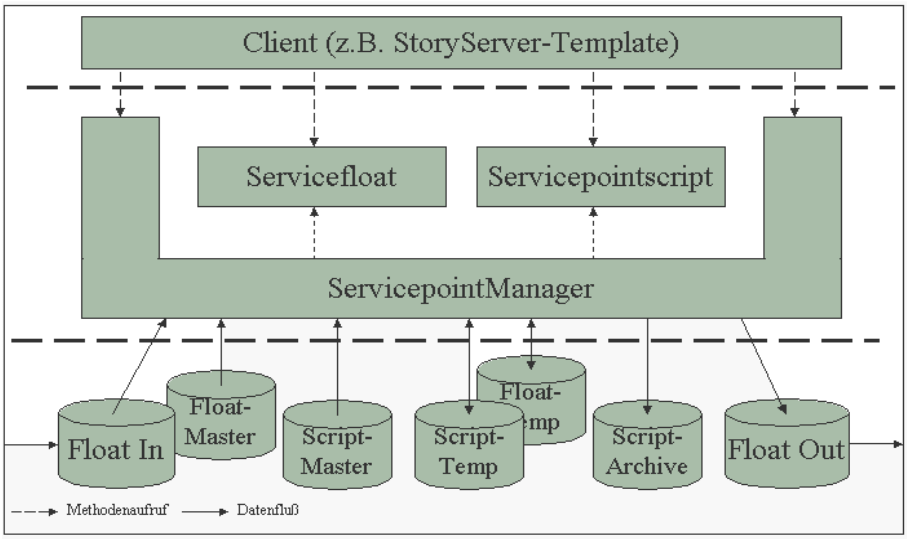

Abbildung 25: Schichtenübergreifendes Zusammenspiel der Komponenten am Servicepoint

3.6.2 Realisierung des Servicepointmanagers

Der Servicepointmanager stellt die zentrale Einheit an einem Servicepoint dar. Er stellt folgende Dienstleistungen zur Verfügung:

- Verwalten von Servicefloat- und Servicepointscripts sowie der zur Erzeugung notwendigen Prozessvorlagen
- Erzeugen neuer Servicefloat- und Servicepointscripts aus Prozessvorlagen
- Empfang und Versand von Servicefloats
- Archivierung abgearbeiteter Servicepointscripts

Der Servicepointmanager fasst die Anwendungslogik zusammen, die neben den exemplarbezogenen abfragenden und verändernden Umgangsformen (vgl. Abschnitt 3.3.2) zusätzlich notwendig ist, um die Prozessrepräsentationen an einem Servicepoint zu handhaben.

Abbildung 26 präsentiert einen Ausschnitt der Schnittstelle des Servicepointmanagers, die von seinen Klienten genutzt werden kann, um die aufgeführten Services in Anspruch zu

nehmen. Auf die Darstellung anwendungsfachlicher Exceptions wurde hier aus Platzgründen verzichtet.

```
public Servicefloat createNewServicefloat (String cId, String type);
public Servicefloat getServicefloat(String cId, String fId);
public Servicepointscript getServicepointscript(String cId, String fId);
public List getListOfAvailableServicefloatTypes ();
public List getCustomerIds ();
public List getServicefloatIds (String cId);
public void prepareForDispatch (String cId, String fId);
public void sendServicefloat(String cId, String fId);
```

Abbildung 26: Ausschnitt aus der Schnittstelle des Servicepointmanagers

Die Methode `createNewServicefloat` ermöglicht das Erzeugen eines neuen Servicefloats für einen Kunden aus einer Vorlage, die durch Angabe des Typs identifiziert wird. Welche Vorlagen am Servicepoint zur Verfügung stehen, kann durch die Methode `getListOfAvailableServicefloatTypes` erfahren werden, die eine aus Strings bestehende Liste der verfügbaren Typen liefert. Der Prototyp erzeugt automatisch zu einem neuen Servicefloat ein passendes Servicepointscript. Diese Variante wurde zunächst der Einfachheit halber gewählt. Im Hinblick auf eine flexible Unterstützung sollte jedoch die Möglichkeit zur Verfügung gestellt werden, dass Servicepointscripts getrennt von Servicefloats erzeugt werden können (vgl. Abschnitt 3.3.2).

Auf ein am Servicepoint verfügbares Servicefloat oder Servicepointscript kann durch die Methoden `getServicepointscript` bzw. `getServicefloat` zugegriffen werden. Hierzu ist die Angabe der Kundennummer und des Identifikators des Servicefloats notwendig. Da es zu einem Servicefloat immer nur genau ein Servicepointscript am Servicepoint geben kann, erfolgt der Zugriff auf das Servicepointscript ebenfalls über diesen Identifikator.

Um feststellen zu können, welche Servicefloats für einen Kunden sich gerade am Servicepoint befinden, wird die Methode `getServicefloatIds` angeboten. Für welche Kunden überhaupt Servicefloats am Servicepoint vorliegen, ist durch die Methode `getCustomerIds` feststellbar.

Die Methode `prepareForDispatch` erlaubt es, das angegebene Servicefloat versandfertig zu machen. Dies bezieht sich auf den Übertrag der Dokumente und erreichten Nachbedingungen aus dem Servicepointscript und das Weiterschalten des Servicefloats (auf den Ablauf wird genauer in Abschnitt 3.6.4 eingegangen). `sendServicefloat` initiiert dann die tatsächliche Versendung des Servicefloats. Vor Aufruf dieser Methode können die Prozessrepräsentationen am Servicepoint vom Benutzer immer noch zugegriffen werden, so dass nachträglich evtl. zusätzliche Änderungen vorgenommen werden können. Anschließend ist dies nicht mehr möglich. `sendServicefloat` sorgt auch dafür, dass das benutzte Servicepointscript zu Auswertungszwecken am Servicepoint archiviert wird.

Zur Erbringung seiner eigenen Leistung greift der Servicepointmanager auf eine Reihe weiterer Komponenten zurück, die im folgenden vorgestellt werden. Diese sind für den Client nicht sichtbar, d.h. der Servicepointmanager präsentiert sich als „Black Box“, deren Services nur über seine öffentliche Schnittstelle zugänglich sind. Die Diskussion dieser Komponenten orientiert sich an den oben genannten Aspekten. Jede der Komponenten fokussiert auf die Lösung eines bestimmten Konstruktionsproblems. Bei der Implementation wurde Wert auf die Wiederverwendung von Design [Mey97, S. 70] gelegt. In [Zül98] und [Gry96] finden sich eine Reihe von Entwurfsmustern, die passende Lösungen für die genannten Problembereiche anbieten. Diese Mikroarchitekturen (vgl. Abschnitt 3.2.3) zielen auf die Problemlösung im

Umfeld von Einzelarbeitsplatzsystemen ab, sind aber auch geeignet, im dem hier besprochenen Kontext eine Anleitung zu geben.

Prozessexemplare verwalten

In [Zül98, S. 299 ff.] wird das Entwurfsmuster der Materialverwaltung vorgestellt. Eine Materialverwaltung ist eine zentrale Stelle, an die sich Softwarewerkzeuge einer Arbeitsumgebung wenden können, um die von ihnen benötigten Materialien zur Bearbeitung zu erhalten. Die Materialverwaltung hat Kenntnis davon, welche Materialien zur Bearbeitung zur Verfügung gestellt werden können. Das Entwurfsmuster der Materialverwaltung adressiert zugleich das Problem, die für die Materialien verwendeten Persistenzmedien und -formate gegenüber den bearbeitenden Komponenten zu verbergen, d.h. den Ort als auch die Art der Speicherung transparent, im Sinne von nicht sichtbar, zu halten. Hierdurch wird erreicht, dass ein neues Persistenzmedium ohne Änderungsauswirkungen auf die bearbeitenden Komponenten eingesetzt werden kann.

Die Lösung entsprechender Problemstellungen wurde auch bei der Realisierung der Prozessexemplarverwaltung angestrebt (vgl. Abschnitt 3.1). Oben wurde schon angedeutet, dass als Speichermedium in anderen Versionen auch relationale Datenbanken statt des Dateisystems eingesetzt werden könnten. Es wäre auch möglich, dass man statt XML ein anderes Format zur Prozessrepräsentation einsetzen möchte. Des weiteren ist es sinnvoll, eine ausgezeichnete Komponente mit der Aufgabe der Verwaltung der verfügbaren Materialien zu betrauen, dessen angebotene Dienstleistung vom Servicepointmanager verwendet werden kann, da dadurch die Kohäsion des Servicepointmanagers erhöht wird.

Das Entwurfsmuster Materialverwaltung besteht in seiner einfachsten Form aus drei Teilnehmern (vgl. auch Abbildung 27). Den einzelnen Komponenten kommen folgende Aufgaben zu:

- *Materialverwalter* (`MaterialManager`): Realisiert die Komponente, an die sich andere Softwarekomponenten wenden können, um Materialien anzufordern. Hierzu ist die Identifikation des Materials notwendig. Ergebnisse der Anfragen werden in Form von *Materialsammlungen* an den Anfrager geliefert.
- *Materialmagazin* (`MaterialStore`): Hält die Information über bereits geladene Materialien, so dass der Materialverwalter den Import desselben Materials nicht mehrfach veranlasst.
- *Materialversorger* (`MaterialProvider`): Beschafft die Materialien aus einer externen Datenquelle. Er implementiert das Protokoll, dass die Materialien aus ihrer externen Repräsentation in eine intern verwendete Darstellung transformiert (und umgekehrt). Er hat als einziger Kenntnis darüber, wo die Materialien abgespeichert werden.

Materialien werden vom Materialverwalter und den anderen Komponenten nur unter einer bestimmten Schnittstelle, die als „verwaltbar“ (`Manageable`) bezeichnet wird, verwendet. Dadurch wird erreicht, dass der Materialmanager mit beliebigen Objekten umgehen kann, die diese Schnittstelle implementieren. Auch der Materialsammlung sind ihre Inhalte nur unter der Schnittstelle `Manageable` bekannt. Auftraggeber der Anfrage an den Materialmanager müssen auf der Ergebnismenge eine entsprechende Typkonversion vornehmen um die Materialien unter der von ihnen benötigten Schnittstelle bearbeiten zu können.

Das in Abschnitt 3.1 formulierte Ziel, das der Servicepointmanager ohne großen Änderungsaufwand mit weiteren Materialien umgehen können soll, kann also durch Orientierung an diesem Entwurfsmuster ebenfalls eingehalten werden.

Im Entwurfsmuster werden die vom Materialverwalter zu benutzenden Materialversorger bei der Initialisierung von der Arbeitsumgebung gesetzt. Im hier diskutierten Entwurf gibt es allerdings kein entsprechendes Konzept. Diese Aufgabe muss somit vom Servicepointmanager übernommen werden. In der aktuellen Version werden drei Materialversorger verwendet, einer für am Servicepoint vorhandene Servicefloats, einer für Servicepointscripts und einer für mögliche Folge-Servicepoints[77]. Der Materialmanager kann jedoch mit beliebig vielen Providern umgehen, z.B. könnten spezielle Provider für Dokumente eingeführt werden.

Durch die Verwendung von zwei Providern für die Prozessrepräsentationen wird der Möglichkeit Rechnung getragen, dass die Prozessrepräsentationen grundsätzlich in verschiedenen Persistenzmedien abgespeichert sein können. So könnten Servicefloats in XML-Form im Filesystem abgelegt sein, die Servicepointscripts jedoch in einem Mainframe gehalten werden. Der Typ der angebotenen Materialien kann über die Schnittstelle der Versorger abgefragt werden. Die Provider selber erhält der Servicepointmanager über eine Fabrik (siehe [GHJV96]), welche die Methode `createProvider (String location, String type)` zum Erzeugen von Providern anbietet. Die Parameter identifizieren den Material-Typ, den der Provider anbieten soll (z.B. Servicefloat) und den Ort, auf dem ein Provider arbeitet (z.B. einem temporären Verzeichnis oder den Vorlagen; vgl. Abbildung 25). Die Fabrik ist als Singleton (siehe [GHJV96] und weiter unten) realisiert. Das Wissen darüber, ob es sich bei dem Ort technisch um eine Datenbank oder das Dateisystem handelt, und welcher konkrete Provider folglich hierfür erzeugt werden muss, steckt in der Fabrik und kann dort konfiguriert werden. Somit konnte auch das in Abschnitt 3.1 verfolgte Ziel, auf ein anderes Persistenzmedium bzw. –format umzusteigen, ohne den Servicepointmanager ändern zu müssen, erreicht werden. Es müsste nur ein neuer Provider konstruiert und in der Fabrik registriert werden.

Der funktionale Prototyp verwendet lediglich eine Providerklasse, die auf dem Dateisystem arbeitet. Diese benutzt selbst eine Klasse namens `XMLTransformer`, welche die Transformation von XML-Files in das DOM und wieder zurück übernimmt (aus dieser Klasse stammt auch das Codebeispiel aus Abbildung 19; vgl. auch Abbildung 27). Diese Klasse verhindert zusätzlich das Einlesen nicht gültiger XML-Dokumente durch Verwendung eines validierenden Parsers (vgl. Abschnitt 3.4.2).

Der Materialmanager kann explizit dazu aufgefordert werden, die von ihm verwalteten Materialien im momentanen Bearbeitungszustand auf dem gewählten Persistenzmedium abzuspeichern. So kann gewährleistet werden, dass die verwalteten Prozessrepräsentationen bei eventuellen Ausfällen auf Serverseite den letzen abgespeicherten Bearbeitungszustand aufweisen. Beim starten liest der Materialmanager alle im temporären Verzeichnis abgelegten Prozessrepräsentationen ein.

Verwaltung von Prozessvorlagen und Erzeugung neuer Prozessexemplare

Für die Prozessrepräsentationen Servicefloat und Servicepointscript kann es an einem Servicepoint eine Reihe unterschiedlicher Vorlagen geben, die zur Erzeugung neuer Prozessexemplare verwendet werden können. Die Vorlagen müssen ebenfalls geeignet verwaltet werden. Gryczan [Gry96, S. 191] schlägt für solche Problemstellungen die Verwendung von Repertoires vor. Ein Repertoire ist technisch ein Behälter, in dem alle zur Verfügung stehenden Prozessvorlagen abgelegt sind.

[77] An jedem Servicepoint befindet sich eine Liste möglicher Servicepoints, die in Servicefloats aufgenommen werden können, um den Gesamtablauf zu verändern. Es handelt sich hierbei ebenfalls um ein XML-File, das im Dateisystem abgelegt ist. Aus Gründen der Übersichtlichkeit wurde auf eine Darstellung in Abbildung 25 verzichtet.

Da alle Prozessvorlagen im Hinblick auf den mit ihnen zu realisierenden Umgang und ihre interne Struktur identisch sind (vgl. Abschnitt 3.3.2), kann die Erzeugung neuer Prozessexemplare aus den Vorlagen durch Generierung einer Kopie entsprechend des Prototyp-Musters aus [GRJV96, S. 127 ff.] realisiert werden. Hierzu bieten die Prozessrepräsentationen die Methode `getCopy` an, die eine tiefe Kopie der jeweilige Vorlage liefert.

Der Servicepointmanager verwendet ein Repertoire. Um neue Prozessrepräsentationen vom Repertoire erhalten zu können, muss der Servicepointmanager die Methode `getProcessrepresentation (String mainType, String subType)` verwenden. Hierbei identifiziert `mainType` den Haupttyp (Servicefloat oder Servicepointscript) und `subType` einen Untertyp, der die jeweilige Prozessvorlage eindeutig identifiziert (z.B. „Briefwahlantrag 2001“). Welche Typen verfügbar sind, kann am Repertoire abgefragt werden. Intern werden die Vorlagen in einem assoziativen Speicher gehalten. Neu erzeugte Prozessexemplare werden mit einem eindeutigen Identifikator (der Prototyp verwendet hierfür die Kundennummer) versehen und dann dem Materialmanager übergeben. Hiernach können diese dann vom Benutzer zugegriffen werden um z.B. weitere Informationen zum Kunden einzutragen oder den vorgesehenen Ablauf zu modifizieren.

Die Verwaltung der Vorlagen wird nicht vom Materialmanager übernommen, da dies durch die zusätzliche Verantwortlichkeit zu einer Verringerung seiner Kohäsion führen würde. Außerdem müssen die konkreten Typen der zu erzeugenden Prozessexemplare bekannt sein (Servicefloat oder Servicepointscript mit entsprechenden Varianten bzw. Untertypen). Ein Aspekt des Materialmanagers ist allerdings von den konkret verwalteten Typen zu abstrahieren.

Die Vorlagen selber sind am Servicepoint auch als XML-Files abgelegt. Zur Transformation in die benötigte Objektdarstellung verwendet ein Repertoire ebenfalls Provider, die von Servicepointmanager direkt nach der Erzeugung gesetzt werden und mit deren Hilfe das Repertoire dann die am Servicepoint verfügbaren Vorlagen einliest.

Empfang und Versand von Servicefloats

Das Konzept der Postkörbe wird in [Zül98, S. 440 ff.] vorgeschlagen, um eine Verbindung von Arbeitsplätzen zum expliziten Austausch von Arbeitsmaterialien zu realisieren. Es wird zwischen Eingangs- und Ausgangspostkörben unterschieden, die benannt sind und in die die auszutauschenden Arbeitsmaterialien hineingelegt oder aus ihnen entnommen werden können. Ein Postversandsystem [Zül98, S. 442 ff.] sorgt für den Transport der auszutauschenden Arbeitsgegenstände. Technisch werden Postkörbe als Behälter realisiert, die eine feste Versandadresse haben und bei ihrer Erzeugung einem bestimmten Arbeitsplatz zugeordnet werden.

Diese Idee wurde auch bei der Realisierung des Servicepointmanagers übernommen. Servicefloats sind zwischen den einzelnen Servicepoints auszutauschen. Damit dies umgesetzt werden kann, müssen ausgezeichnete Komponenten für den Empfang und den Versand der Prozessexemplare existieren.

Es gibt zwei Komponenten, die als `FloatReceiver` und `FloatSender` bezeichnet werden, und entsprechend ihrer Namen die Zuständigkeit der Versendung und des Empfangs von Servicefloats übernehmen. Ein Servicepointmanager erzeugt bei seiner Initialisierung jeweils ein Exemplar der benannten Klassen. Technisch sind Receiver und Sender als Behälter umgesetzt. Receiver und Sender haben immer eine eindeutige Adresse, so dass ein Servicefloatversandsystem den richtigen Servicepoint eindeutig als Empfänger oder Versender identifizieren kann.

Zusätzlich sorgen Receiver und Sender für die Transformation des Materials aus und in das Transportformat und stoßen den eigentlichen Transport an bzw. nehmen die transportierten Prozessrepräsentationen entgegen. Ein Servicefloatversandsystem ist nicht realisiert worden, so dass ankommende und abgehende Servicefloats ebenfalls im Dateisystem abgelegt sind.

Nachfolgende Abbildung 27 stellt anhand eine UML-Klassendiagramms[78] dar, aus welchen Komponenten sich der Servicepointmanager zusammensetzt.

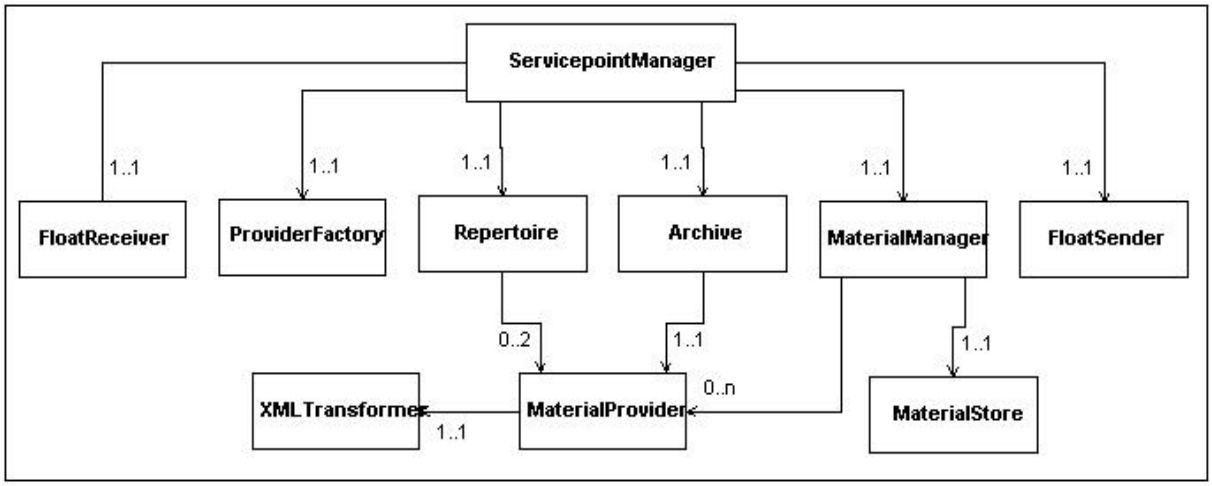

Abbildung 27: UML-Klassendiagram: Struktur des Servicepointmanagers

Sollen Servicefloats zu einem anderen Servicepoint übertragen werden, so ist sicherzustellen, dass sich die in einem Servicepointscript befindenden Dokumente und hinterlassenen Nachbedingungen zu Dokumentationszwecken in das Servicefloat übertragen werden. Diese Aufgabe übernimmt ebenfalls der Servicepointmanager, da er als einzige Komponente die Zusammenhänge zwischen Servicepointscript und Servicefloat kennt.

Zusätzlich verwendet der Servicepointmanager ein Archiv (siehe Abbildung 27) für Servicepointscripts, das die an einem Servicepoint abgearbeiteten Scripts z.B. für spätere Auswertungszwecke an einem ausgezeichneten Ort abspeichert. Hierdurch wird die Dienstleistung der Archivierung abgearbeiteter Servicepointscripts des Servicepointmanagers umgesetzt (vgl. oben).

Der Servicepointmanager ist nach dem Singleton-Entwurfsmuster (vgl. [GHJV96, S. 139 ff.]) realisiert. Über dieses Entwurfmuster kann sichergestellt werden, dass eine Klasse genau ein Exemplar besitzt und dies über einen wohldefinierten globalen Zugriffpunkt angesprochen werden kann. Alle Objekte anderer Klassen benutzen zur Laufzeit das gleiche Exemplar.

Die Verwendung dieses Entwurfsmusters ist hauptsächlich dadurch motiviert, dass das mehrfache Auftreten von Servicepointmanager-Exemplaren an einem Servicepoint einen zusätzlichen Kommunikationsaufwand zwischen ihnen erforderlich machen würde. Sie müssten sich z.B. darüber informieren, dass neue Servicefloats an einem Servicepoint angekommen sind oder dass gerade ein neues Servicepointscript für einen bestimmten Kunden erzeugt wurde. Des weiteren müssten alle vorhandenen Exemplare dahingehend abgefragt werden, ob z.B. eine zu bearbeitende Prozessrepräsentation für einen Kunden am Servicepoint schon existiert. Durch die Verwendung einer zentralen Instanz wird der Entwurf vereinfacht (siehe hierzu auch Abschnitt 3.6.5).

Zur Benachrichtigung des Servicepointmanagers über neu angekommene Servicefloats an einem Servicepoint und zur Benachrichtigung evtl. Transportautomaten über zum Versand

[78] Zu UML-Klassendiagrammen siehe z.B. [Oes01].

bereite Servicefloats wird das Beobachter-Entwurfsmuster (vgl. [GHJV96, S. 257 ff.]) eingesetzt. Das Entwurfmuster zielt darauf ab, eine 1:n-Abhängigkeit zwischen Objekten zu definieren, so das eine Änderung des Zustandes eines Objektes dazu führt, dass alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden.

Der Floatreceiver ist hierbei z.B. das Objekt, dessen Zustandsänderung den Servicepointmanager interessiert. Sobald ein neues Servicefloat dem Receiver übergeben wurde, informiert er den Servicepointmanager und teilt ihm einen eindeutigen Identifikator mit. Dieser wird vom Manager genutzt, um das neue Servicefloat aus dem Receiver zu entnehmen. Der Servicepointmanager ist dem Floatreceiver nur unter der abstrakten Beobachter-Schnittstelle bekannt. Die zyklische Abhängigkeit der beiden Komponenten kann somit minimal gehalten werden[79].

Nach Entnahme übergibt der Servicepointmanager das neue Servicefloat an den Materialmanager. Hierbei wird gleichzeitig ein neues zum Servicefloat passendes Servicepointscript erzeugt, was ebenfalls dem Materialmanager übergeben wird. Danach stehen beide Prozessrepräsentationen zur Bearbeitung am Servicepoint zur Verfügung. Diese Variante der Zuordnung von Servicefloat und Servicepointscript (vgl. Abschnitt 3.3.2) wurde zunächst der Einfachheit halber gewählt. Im Hinblick auf eine flexible Prozessunterstützung an Servicepoints, empfiehlt sich jedoch die Orientierung an der Variante, die dem Benutzer des Systems die Möglichkeit bietet, das passende Servicepointscript selbst auszuwählen.

3.6.3 Realisierung der Prozessrepräsentationen

Die Prozessrepräsentationen Servicefloat und Servicepointscript sind auf Basis des DOM (vgl. Abschnitt 3.4.2) implementiert worden (siehe hierzu auch Abschnitt 3.1). Ihre Realisierung wird anhand der in Abschnitt 3.3.2 aus den Aktivitäten Prozessvorlage verändern und Aktivitäten durchführen und Dokumentieren abgeleiteten Umgangsformen verdeutlicht.

Abbildung 28 präsentiert ein UML-Klassendiagramm, dass einen Überblick über die grundsätzliche Realisierung der Prozessrepräsentationen gibt. Servicefloat und Servicepointscript sind Interfaces, zu denen jeweils eine korrespondierende Implementation (`ServicefloatImpl` und `ServicepointscriptImpl`) existiert. Die beiden Interfaces spezifizieren die Methoden, die vom Interaktionsservice, also in diesem Fall den StoryServer-Templates, verwendet werden können. Beide Implementationen besitzen intern eine Referenz auf die DOM-Repräsentation der XML-Files vom Typ `Document` und nutzen die dort angebotenen Methoden zur Realisierung der fachlich motivierten Methoden. Das Interface `Manageable` spezifiziert die Methoden die vom Materialmanager benötigt werden. `ManageableServicepointscript` und `ManageableServicefloat` spezifizieren zusätzlich die Operationen, die vom Servicepointmanager verwendet werden, um geeignet mit den Prozessrepräsentationen umgehen zu können. Hierzu gehören z.B. Methoden um die Dokumente und erreichten Nachbedingungen aus dem Servicepointscript in das Servicefloat zu übertragen, bevor das Servicefloat zum nächsten Servicepoint versendet wird (siehe hierzu auch Abschnitt 3.6.4).

[79] Die beidseitige Bekanntschaft wird in Abbildung 27 dadurch angedeutet, dass die Assoziation zwischen `FloatReceiver` und `ServicepointManager` als ungerichtete dargestellt ist.

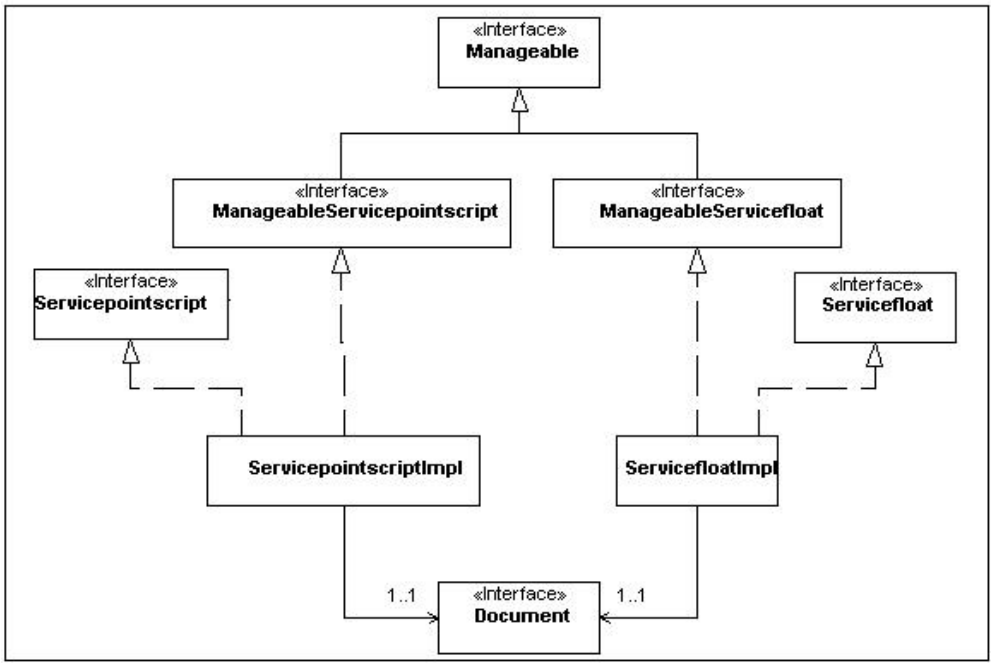

Abbildung 28: UML-Klassendiagramm: Überblick über die Realisierung der Prozessrepräsentationen

Prozessvorlage verändern

Die Operationen, die Prozessvorlage zu verändern, beziehen sich hauptsächlich darauf, neue Prozesselemente hinzuzufügen oder zu entfernen, Reihenfolgebeziehungen zu verändern, Vor- und Nachbedingungen hinzuzufügen und Zuständigkeiten neu zuzuweisen. Diese Operationen können leicht auf das DOM abgebildet werden. Es ist möglich neue Elemente oder Attribute zu erzeugen und an vorhandene anzuhängen, Elemente umzupositionieren oder zu löschen (vgl. Abschnitt 3.4.2). Diese von den Java-APIs angebotenen Methoden wurden genutzt, um die fachlich motivierten und auf die Veränderung des Prozessmodells bezogenen Operationen umzusetzen.

Nachfolgende Abbildung 29 stellt einen Ausschnitt der Methoden der Prozessrepräsentationen vor, die von ihnen angeboten werden, um den weiteren Verlauf der Prozessausführung anpassen zu können.

Servicepointscript:

```
public void addScheduledActivity(String nameOfPredecessorActivity, String
                                                              nameOfNewActivity);
public void deleteScheduledActivity(String nameOfScheduledActivity);
public void reopenLastActivity();
```

Servicefloat:

```
public void addScheduledServicepoint(String nameOfPredecessorServicepoint,
                                              String nameOfNewServicepoint);
public void deleteScheduledServicepoint(String nameOfScheduledServicepoint);
```

Abbildung 29: Ausschnitt aus den Schnittstellen der Prozessrepräsentationen

Mit Hilfe der Methoden `addScheduledActivity` bzw. `addScheduledServicepoint` können neue Prozesselemente in das jeweilige Prozessmodell aufgenommen werden. Die Position wird durch Angabe des Namens des Vorgängerelements bestimmt. Wird kein Name angegeben, so wird das neue Prozesselement als erstes Element in der Liste der noch zu

durchlaufenden Prozesselemente eingetragen. Prozesselemente können durch Angabe ihres Namens gelöscht werden. `reopenLastActivity` ermöglicht es, zur letzten bearbeiteten Aktivität im Servicepointscript zurückzuspringen.

Aktivitäten durchführen und Dokumentieren

Um die Aktivität, die als nächstes durchgeführt werden soll, auswählen zu können, muss festgestellt werden können, welche schon erledigt wurden und welche noch zu erledigen sind. Zusätzlich muss es nach Durchführung der jeweiligen Aktivität möglich sein, diese als erledigt zu kennzeichnen. Mit Hilfe der Java-API kann man durch das DOM Navigieren und die benötigten Informationen an den Elementen und Attributen abfragen und so den Prozesszustand bestimmen. Aktivitäten werden als erledigt gekennzeichnet, indem sie dem die Liste der bereits erledigten Aktivitäten repräsentierenden Teilbaum hinzugefügt werden. Dies geschieht durch umhängen des die betroffene Aktivität repräsentierenden Teilbaums. Die als nächstes auszuführende Aktivität wird durch umhängen der ersten Aktivität in der Liste der noch auszuführenden Aktivitäten an die Position der aktuellen Aktivität realisiert. Äquivalent, nur umgekehrt, wird beim Zurückspringen zu einer bereits durchgeführten Aktivität vorgegangen (Methode `reopenLastActivity` in Abbildung 29). Entsprechend wird auch auf der Ebene des Servicefloats mit Servicepoints umgegangen. Die Kennzeichnung des Einhaltens von Bedingungen wird durch Modifikation entsprechender Attributwerte realisiert.

Das Hinzufügen neuer Dokumente zu einem Servicepointscript ist nicht realisiert worden. Es ist lediglich möglich, vorhandene Dokumente bzw. Dokumentvorlagen zu modifizieren und deren Inhalte auszulesen.

Nachfolgende Abbildung 30 zeigt einen Ausschnitt aus den Methoden des Servicepointscripts, die sich auf die Feststellung des Prozesszustandes und des Weiterschaltens beziehen.

Mit Hilfe der ersten drei Methoden kann sich ein Klient des Servicepointscripts die Namen der im Servicepointscript vorhandenen Aktivitäten geben lassen und so feststellen, welche Aktivitäten bereits erledigt wurden, welches die aktuelle ist und welche noch erledigt werden müssen. Durch Verwendung des jeweiligen Namens können dann weitere Informationen zu den einzelnen Aktivitäten abgefragt werden. So liefert z.B. `getDescriptionOfActivity` eine Beschreibung der zur Erledigung der angegebenen Aktivität normalerweise auszuführenden Aktionen. Entsprechend wird auch mit Vor- und Nachbedingungen verfahren, wobei hier beispielhaft Methoden für die Nachbedingungen angegeben sind. `endActivity` bewirkt das Weiterschalten des Servicepointscripts, indem die aktuelle Aktivität der Liste der erledigten Aktivitäten hinzugefügt wird und die nächste auszuführende die Position der aktuellen Aktivität einnimmt.

```
public List getListOfNamesOfScheduledActivities();
public List getListOfNamesOfPassedActivities();
public String getNameOfCurrentActivity();
public String getDescriptionOfActivity(String nameOfActivity);
public List getListOfSpPostconditions();
public String getStateOfSpPostcondition(String postconditionName);
public void endActivity();
```

Abbildung 30: Ausschnitt aus der Schnittstelle `Servicepointscript`

3.6.4 Vorbereitung zum Versand eines Servicefloats

Um das dynamische Zusammenspiel der in den vorangegangenen Abschnitten dieses Kapitels präsentierten Komponenten am Servicepoint zu verdeutlichen, wird im folgenden dargestellt, welche Aktionen der Servicepointmanager in Kollaboration mit Materialmanager, Servicepointscript und –float durchführt, um ein Servicefloat zum Versand zum nächsten Servicepoint vorzubereiten. Der Vorgang ist in Abbildung 31 anhand eines UML-Sequenzdiagramms[80] illustriert.

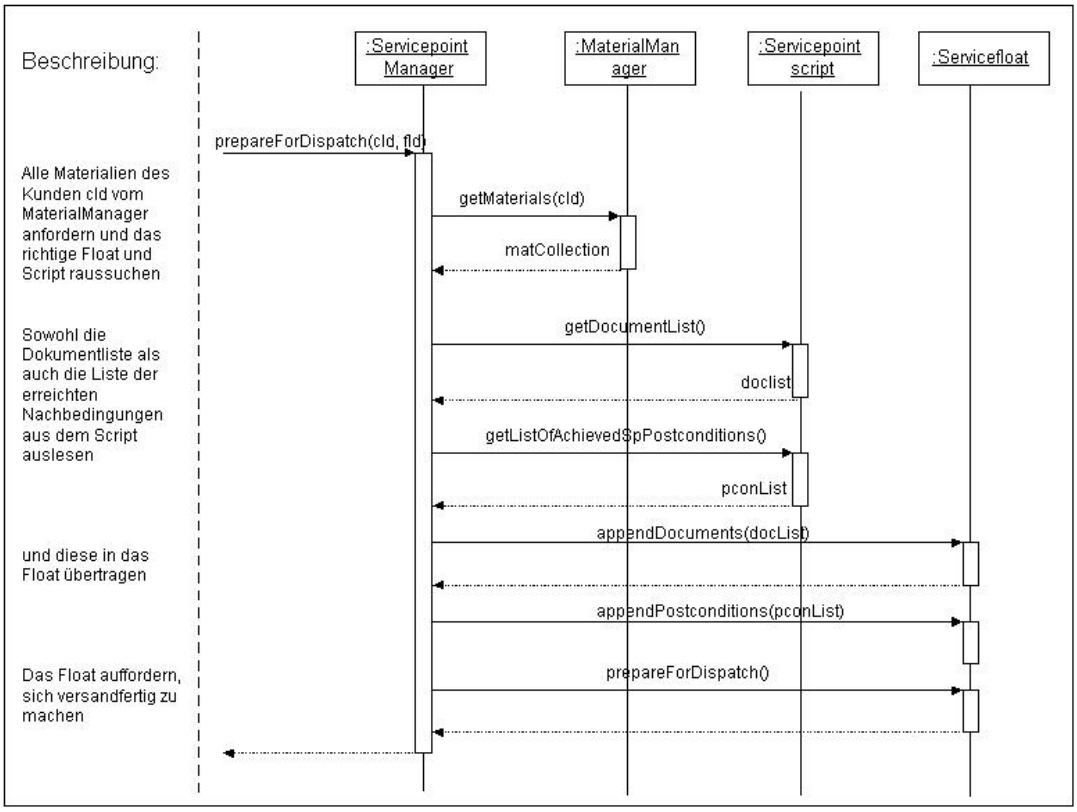

Abbildung 31: UML-Sequenzdiagramm: Vorbereitung zum Versand eines Servicefloats

Der `ServicepointManager` bietet seinen Klienten hierzu die Methode `prepareForDispatch` an, die zwei Argumente erwartet (vgl. Abschnitt 3.6.2). Das erste Argument (`cId`) identifiziert eindeutig den Kunden, für den die Serviceteilleistung am Servicepoint erbracht wurde und das zweite Argument (`fId`) das Servicefloat, das versandt werden soll. Die Identifikation des Servicefloats ist notwendig, da es grundsätzlich mehrere Servicefloats für einen Kunden an einem Servicepoint geben kann (vgl. Abschnitt 3.3.2). Nach Aufruf der Methode wendet sich der `ServicepointManager` an den `MaterialManager`, um das Servicefloat mit korrespondierendem Servicepointscript (falls vorhanden) zu erhalten. Es wird an dieser Stelle davon ausgegangen, dass ein Servicepointscript vorhanden ist. Der `ServicepointManager` benutzt hierzu die vom `MaterialManager` zur Verfügung gestellte Methode `getMaterials(cId)`. Letzterer sucht daraufhin die unter diesem Identifikator verwaltete Materialsammlung aus dem Materialmagazin heraus[81] und liefert sie als Ergebnis des Methodenaufrufs

[80] Zu UML-Sequenzdiagrammen siehe z.B. [Oes01].

[81] Auf die Darstellung des Materialmagazins ist in Abbildung 31 aus Gründen der Übersichtlichkeit verzichtet worden.

(`matCollection`) zurück. Anschließend durchsucht der `ServicepointManager` die `MaterialCollection` nach den beiden Prozessrepräsentationen unter Nutzung des angegebenen Identifikators `fId`. Da jedes Servicepointscript ebenfalls den Identifikator für das Servicefloat enthält, dem es zugeordnet ist (vgl. Abschnitt 3.3.2), wird dieser zur eindeutigen Bestimmung des Servicepointscripts verwendet.

Anschließend werden die im Servicepointscript enthaltenen Dokumente und erreichten Nachbedingungen vom Servicepointmanager ausgelesen und in das Servicefloat übertragen. Zum Auslesen bietet das `Servicepointscript` die Methoden `getDocumentList` und `getListOfAchievedSpPostconditions` an, die in der Schnittstelle `ManageableServicepointscript` spezifiziert sind. Diese werden dann in das `Servicefloat` unter Nutzung der von diesem angebotenen Methoden `appendDocuments` und `appendPostconditions`, die in der Schnittstelle `ManageableServicefloat` spezifiziert sind, eingetragen (vgl. auch Abschnitt 3.6.3). Zur Realisierung dieser Methoden ist auf der Ebene des DOM die Möglichkeit genutzt worden, dass Teilbäume eines DOMs in ein anderes DOM übertragen werden können (vgl. Abschnitt 3.4.2).

Zum Schluss fordert der Servicepointmanager das Servicefloat auf, sich versandfertig zu machen. Hierzu wird ebenfalls eine Methode namens `prepareForDispatch` vom `ManageableServicefloat` angeboten. Diese Methode bewirkt, dass der als aktueller Servicepoint eingetragene in die Liste der bereits durchlaufenen Servicepoints verschoben wird und der erste Servicepoint der Liste der noch zu durchlaufenen Servicepoints die Position des aktuellen Servicepoints einnimmt. Diese Operationen sind auf der Ebene des DOM durch Umhängen der entsprechenden Teilbäume realisiert (vgl. Abschnitt 3.4.2).

3.6.5 Realisierung des Interaktionsservices

Der Interaktionsservice basiert auf dem StoryServer, insbesondere dem Templatetyp „Component“. Die Verbindung zwischen Templates und dem Servicepointmanager wurde über TclBlend hergestellt. Im folgenden wird darauf eingegangen, wie die Realisierung des Interaktionsservice des Prototyps ausgestaltet wurde.

Abbildung 32 präsentiert einen Ausschnitt der vom StoryServer generierten HTML-Oberfläche, wie sie zum Browser des Bürgers übertragen wird, der die Self-Service-Anwendung im Rahmen des Bürgerprozessportals www.hamburg.de benutzt. Die Oberfläche setzt sich aus vier StoryServer-Komponenten zusammen, die in einer HTML-Tabelle, die sich in einem weiteren Template (im folgenden Basistemplate genannt) befindet, angeordnet sind. Die beiden oberen Templates sind als reine Präsentationstemplates umgesetzt worden. Das Template links oben gibt den Namen des Serviceflows an.

Das linke untere Template stellt den Zustand des Teilprozesses der Erstellung des Briefwahlantrags am Self-Servicepoint dar. Es präsentiert die Namen der einzeln am Servicepoint auszuführenden Aktivitäten, wobei die aktuell auszuführende Aktivität durch einen Pfeil gekennzeichnet ist. Rechts daneben befindet sich das Hauptfenster (im folgenden Interaktionstemplate genannt), in dem Benutzern Informationen zu den einzelnen Aktivitäten präsentiert werden. Zusätzlich erhält er hier die Möglichkeit, die zur Prozessausführung benötigten Daten einzugeben, wie es in Abbildung 32 dargestellt ist. In diesem Bereich werden verschiedene Templates eingesetzt, die jeweils mit dem Zustand der Antragserstellung korrespondieren. Wird der Prozess weitergeschaltet, so wird jedes Mal das Basistemplate mit erforderlichen Parametern aufgerufen, die an die „inneren“ Templates weitergeleitet werden. Über einen solchen Parameter wird z.B. von einem zusätzlich verwendeten Template bestimmt, welches Interaktionstemplate verwendet werden soll, z.B. das Template, aus dem das in Abbildung 32 dargestellte Formular generiert wird.

Abbildung 32: Screenshot der HTML-Oberfläche des Self-Servicepoint

Während des Durchlaufs des Antragerstellungsprozesses am Self-Servicepoint findet nur nach Drücken des Button „Antrag senden“ eine Interaktion mit dem Servicepointmanager und den Prozessrepräsentationen statt. Hier wird ein Template aufgerufen, das keine Präsentationsaufgaben übernimmt (im folgenden Verarbeitungstemplate genannt). In diesem Template wird zunächst der Servicepointmanager über TclBlend angesprochen und zur Erzeugung eines neuen Servicefloats mit korrespondierendem Servicepointscript für den jeweiligen Kunden aufgefordert (entsprechend des Codebeispiels aus Abbildung 22). In das Servicepointscript werden dann die in den Formularfeldern eingetragenen Informationen, die dem Verarbeitungstemplate per URL-Parameter übergeben werden, eingetragen. Anschließend wird der Servicepointmanager dazu aufgefordert, die erreichten Nachbedingungen und die Dokumente (das Briefwahlantragsformular) vom Servicepointscript in das Servicefloat zu übertragen und das Servicefloat zu versenden. Im letzten Schritt verzweigt das Template wieder auf das Basistemplate und präsentiert die Dankeseite, die den Antragsteller über den weiteren Verlauf des Briefwahlantrags informiert (vgl. Szenario in Abschnitt 2.5.2).

Bis auf das Verarbeitungstemplate interagiert kein Template mit dem Servicepointmanager oder den Prozessrepräsentationen. Alle anderen Templates arbeiten mit Datenbanktabellen, die in einer vom StoryServer verwendeten Datenbank angelegt sind. Dort sind die darzustellenden Informationen feststehend hinterlegt, z.B. die Bezeichnungen für die einzelnen Aktivitäten.

Eine andere mögliche Realisierungsvariante, die zunächst auch im Rahmen des in Abschnitt 3.1 erwähnten Projektseminars angestrebt wurde, stellt sich so dar, dass die Informationen über den Prozesszustand dynamisch aus einem korrespondierenden Servicepointscript ausgelesen und in HTML kodiert werden. Ebenso könnte die Bezeichnung des Gesamtprozesses (Template links oben) dynamisch aus einem korrespondierenden Servicefloat bestimmt werden. Die Interaktion mit dem Servicepointmanager und den

Prozessrepräsentationen würde in diesem Fall also auch in Präsentationstemplates stattfinden. Der wesentliche Vorteil dieser Variante ist, dass dann auch verschiedene Prozesse (nicht nur der Briefwahlantrag) auf der gleichen Menge von Templates realisiert werden könnte. Man bräuchte nur andere Servicepointscript- und Servicefloattypen definieren. Allerdings konnte diese Variante aus Zeitgründen nicht mehr umgesetzt werden.

Die Variante ist an dieser Stelle zusätzlich dargestellt worden, da sie die Entscheidung, den Servicepointmanager nach dem Singleton-Muster zu entwickeln, zusätzlich beeinflusst hat (vgl. auch Abschnitt 3.6.2). Um an ein Prozessexemplar zu gelangen, muss ein Template zunächst den Servicepointmanager ansprechen. Um zu verhindern, dass dadurch beliebig viele Exemplare des Servicepointmanagers während der Generierung der gesamten HTML-Seite erzeugt werden, schien das Singleton Muster eine geeignet Lösung zu sein.

3.6.6 Zusammenfassung

In diesem Kapitel wurde der funktionale Prototyp eines webbasierten Servicepoints vorgestellt, der auf Basis der Technologien Java, XML, StoryServer 5.0 und TclBlend entstanden ist. Zunächst ist anhand einer Architekturübersicht verdeutlicht worden, welche Technologie in welcher Ebene eingesetzt wird und welche Aufgaben den einzelnen Komponenten zukommt. Der StoryServer ist für die Realisierung des Interaktionsservice zuständig, Java wird für die Umsetzung der Umgangsformen, die in Kapitel 3.3.2 benannt wurden, eingesetzt, XML ist das Persistenzmedium. TclBlend verbindet StoryServer-Templates mit den Java-Komponenten.

Anschließend wurde die Realisierung des Servicepointmanagers vorgestellt. Die ihm zugewiesenen Aufgaben (Verwalten von Prozessexemplaren und Vorlagen, Erzeugen neuer Prozessrepräsentationen, Weiterleiten von Servicefloats und Archivierung von Servicepointscripts) konnten vorwiegend durch Orientierung an Entwurfsmustern, die aus dem Desktop-Bereich stammen, umgesetzt werden und bestimmen die interne Architektur des Servicepointmanagers. Hierdurch konnten auch die in Abschnitt 3.1 formulierten Entwurfsziele eingehalten werden. Der Servicepointmanager selber wurde nach dem Singleton-Entwurfsmuster realisiert.

Danach ist auf die Umsetzung der Prozessrepräsentationen eingegangen worden. Die fachlich motivierten Methoden wurden direkt durch Abbildung auf die von der DOM-API angebotenen Operationen realisiert.

Hiernach wurde anhand des Beispiels der Vorbereitung zum Versenden eines Servicefloats das dynamische Zusammenspiel des Servicepointmanagers mit einigen der von ihm benutzten Komponenten verdeutlicht.

Zum Abschluss wurde dargestellt, wie die Benutzungsoberfläche des Self-Servicepoints mit Hilfe des StoryServers implementiert worden ist.

3.7 Evaluation des funktionalen Prototypen

Im folgenden wird der eben präsentierte funktionale Prototyp im Hinblick auf die Erfüllung des in Kapitel 2 erstellten Anforderungskatalogs und der in Kapitel 3.2.3 ermittelten allgemeinen softwaretechnischen Kriterien, die beim Entwurf von Softwaresystemen beachtet werden müssen, evaluiert[82]. Zusätzlich wird auf technischer Ebene untersucht, ob bei der Realisierung des Prototypen die Besonderheiten, die sich aus seiner Verwendung im Webumfeld ergeben, adäquat berücksichtigt wurden. Im Anschluss an die Untersuchung des Prototypen werden die gewonnenen Ergebnisse im Rahmen einer Empfehlung zur Gestaltung einer Softwarearchitektur für webbasierte Servicepoints zusammengefasst.

3.7.1 Erfüllung des Anforderungskatalogs

Die Diskussion der Erfüllung des Anforderungskatalogs durch den funktionalen Prototypen strukturiert sich entsprechend der in Kapitel 2 genannten Punkte.

Dynamische Modellierung und unmittelbare Adaptivität

An einem Servicepoint soll die situative Anpassung des dem jeweiligen Prozessexemplar zugrundeliegenden Prozessmodells sowohl auf Gesamt- als auch auf Teilprozessebene (Servicefloat und Servicepointscript) möglich sein und unmittelbar darauf reagiert werden können. Die hierzu notwendigen Umgangsformen wurden in Abschnitt 3.3.2 unter dem Punkt „Prozessvorlage verändern“ beschrieben.

Diese Anforderung ist vom Prototyp teilweise umgesetzt worden. So ist es z.B. möglich, neue zu erfüllende Aktivitäten in Servicepointscripts aufzunehmen oder noch zu erledigende Aktivitäten zu löschen. Das gleiche gilt auf der Ebene der Servicefloats, d.h. man kann neue Servicepoints aufnehmen und noch nicht erreichte löschen. Neue Servicepoints stammen aus einer Liste verfügbarer Servicepoints, die an jedem Servicepoint vorhanden sein muss. Diese neuen Einträge beeinflussen den weiteren Prozessverlauf. Rücksprünge im Prozess sind durch Neueintrag der schon durchlaufenen Servicepoints möglich, falls z.B. Teilleistungen nur ungenügend erbracht wurden. Auf Scriptebene kann schrittweise zu den vorherigen Aktivitäten zurückgesprungen werden.

Reihenfolgebeziehungen können jedoch nicht geändert werden. Dies ist allerdings auf einfache Art und Weise zu implementieren, denn es ist hierzu auf der Ebene des zugrundeliegenden DOMs nur eine Umordnung der entsprechenden Elemente erforderlich. Da die Erfüllung von Vor- und Nachbedingungen nicht formal interpretiert, sondern in die Verantwortung der Benutzer gelegt wird, würde eine Umordnung auch nicht zu einem Eingriff seitens des SfMS führen.

Die Möglichkeit, neue Vor- und Nachbedingungen in das Servicepointscript aufzunehmen, d.h. die Geschäftsprozessschnittstelle hierdurch zu modifizieren, wurde ebenfalls nicht realisiert. Es ist lediglich möglich, den Zustand vorhandener Bedingungen zu modifizieren. Ebenso können Zuständigkeiten, z.B. für Aktivitäten, nicht neu zugewiesen werden. Es werden lediglich die in Abschnitt 3.6.3 in Abbildung 29 dargestellten Methoden angeboten.

Die unmittelbare Adaptivität des jeweiligen Prozessexemplars wird dadurch sichergestellt, dass die Änderungen direkt am vergegenständlichten Prozessmodell durchgeführt werden, welches zur Bestimmung des weiteren Prozessverlaufs verwendet wird.

---

[82] Unter dem Begriff Evaluieren versteht man nach [RecPom97, S. 871]: „Das zielbezogene Beurteilen von Objekten auf der Grundlage eines Systems von Bewertungskriterien.“

Prozesstransparenz

Eine Softwareunterstützung am Servicepoint soll dem Benutzer alle für ihn relevanten Informationen über den Prozess, an dem er teilnimmt, unter Beachtung von Zugriffsrechten zugänglich machen.

Diese Anforderung wird vom Prototypen zum größten Teil erfüllt. Die hierzu notwendigen Umgangsformen wurden in Abschnitt 3.3.2 unter dem Punkt „Aktivitäten durchführen und dokumentieren“ beschrieben. Ein Teil der hierzu zur Verfügung gestellten Methoden wurde in Abschnitt 3.6.3 in Abbildung 30 dargestellt.

Mit Hilfe der Prozessrepräsentationen Servicefloat und Servicepointscript werden dem Benutzer sowohl Informationen über den Prozesszustand (Historie, aktuelle Aktivität bzw. Servicepoint, weiterer Verlauf) als auch die zur Aufgabenerfüllung benötigten Informationen und Dokumente, als Ergebnis vorheriger Teilleistungen, zur Verfügung gestellt. Vor- und Nachbedingungen beschreiben die Geschäftsprozessschnittstellen. Zusätzlich sind Informationen zu den Leistungserbringern am jeweiligen Servicepoint verfügbar.

Nicht realisiert sind jedoch Vergleichsfunktionen, die den aktuellen Prozessverlauf mit dem vorgesehenen Standardablauf vergleichen. Hier ergibt sich auch ein Problem im Hinblick auf das Design des Servicefloats, denn es ist nicht vorgesehen, den Standardablaufplan im Servicefloat in Kopie mitzuführen. Es ist immer nur der aktuelle, eventuell modifizierte Plan verfügbar. Da allerdings nicht an jedem Servicepoint der Servicefloat-Master verfügbar sein muss, könnte eine Vergleichsoperation somit nur schwer realisiert werden. Für Servicepointscripts gilt dieses Problem nicht, da ihre Master sich an dem Servicepoint befinden, an dem sie auch ausgeführt werden (außer man entscheidet sich für die Variante, dass ein Servicefloat seine Servicepointscripts selber mitführt; vgl. Abschnitt 3.3.2).

Eine Einschränkung der Transparenz ist ebenfalls nicht möglich, da ein hierzu benötigtes Zugriffskonzept nicht umgesetzt wurde.

Des weiteren kann nicht festgestellt werden, wo sich ein Prozess gerade befindet. Dies resultiert aus dem dezentralen Charakter der gewählten Architektur. Wie ein solcher Mechanismus realisiert werden könnte, ist eine offene Fragestellung.

Prozessinstantiierung, -abarbeitung und –verwaltung

Eine Softwareunterstützung am Servicepoint soll die Erzeugung von Prozessexemplaren aus Vorlagen, deren Abarbeitung inklusive Versendung, sowie die Verwaltung der Exemplare und Vorlagen ermöglichen. Die hierzu notwendigen Umgangsformen wurden in Abschnitt 3.3.2 unter den Punkten „Prozess starten“, „Prozess abschließen“ sowie „Aktivitäten durchführen und Dokumentieren“ beschrieben.

Der funktionale Prototyp erfüllt diese Anforderungen ebenfalls nur teilweise. Dies resultiert daraus, dass der Transport zwischen Servicepoints nicht realisiert worden ist, es kann lediglich die Versendung eingeleitet werden. Die zur Versendung erforderlichen Umgangsformen wurden vollständig umgesetzt („Prozess abschließen“). Wie der Transport selber am besten umzusetzen ist, stellt eine offene Fragestellung dar.

Neue Prozessrepräsentationen werden durch Generierung einer Kopie aus Vorlagen mit Hilfe des Repertoires erzeugt. Dieses stellt auch Informationen darüber zur Verfügung, welche Prozessvorlagen-Typen am Servicepoint vorhanden sind (vgl. Abschnitt 3.6.2).

Die Abarbeitung von Prozessvorlagen, d.h. die Aufgabenerledigung am Servicepoint, wird nahezu vollständig unterstützt, da die hierzu notwendigen Umgangsformen („Aktivitäten durchführen und dokumentieren“) fast alle umgesetzt wurden. Lediglich das Erzeugen und Hinzufügen neuer Dokumente in das Servicepointscript ist nicht möglich.

Ebenso ermöglicht der Prototyp die Verwaltung laufender Prozesse und deren Vorlagen am Servicepoint sowie den Zugriff auf diese (vgl. Abschnitt 3.6.2). Welches Prozessexemplar zur Bearbeitung gewünscht wird, kann dem Servicepointmanager über einen eindeutigen Identifikator mitgeteilt werden. Welche Prozessexemplare überhaupt für einen Kunden am Servicepoint vorhanden sind, kann durch Angabe einer Kundennummer vom Servicepointmanager erfahren werden. Diese ist auch in den Prozessrepräsentationen eingetragen, wodurch bestimmt werden kann, welche Prozessrepräsentationen zu welchem Kunden gehören. Zusammengehörige Servicefloats und Servicepointscripts sind durch den Identifikator des Servicefloats im Servicepointscript gekennzeichnet.

Mehrbenutzerfähigkeit

Eine Softwareunterstützung am Servicepoint muss den gleichzeitigen Zugriff auf verwaltete Prozessexemplare ermöglichen.

Dieser Aspekt wurde im Rahmen der Realisierung des Prototypen nur sehr sporadisch betrachtet. Es ist lediglich möglich, mehrere Prozessexemplare verschiedener Kunden an einem Servicepoint zu handhaben und dem Benutzer Zugriff darauf zu gewähren. Ein passendes Benutzungsmodell, dass den Umgang mit konkurrenten Zugriffssituationen regelt, wurde jedoch nicht umgesetzt.

Integration

Eine Softwareunterstützung am Servicepoint sollte sich in vorhandene Arbeitsumgebungen einfügen, indem es von anderen Anwendungen benutzet werden kann bzw. vorhandene Anwendungsfunktionalitäten selbst verwendet. Diese Anforderung ist ebenfalls nur teilweise umgesetzt worden.

Durch die Orientierung an der Entwurfsmetapher Fachlicher Service und der damit verbundenen Unabhängigkeit des Prototypen von der Präsentation und Handhabung, ist eine Integration in vorhandene, auf Java basierende Umgebungen, wie z.B. beim Anwendungsbeispiel gesehen, leicht möglich.

Der Aspekt der Integration vorhandener Anwendungsfunktionalität, z.B. WfMS zur Realisierung von Supportprozessen, wurde allerdings noch nicht betrachtet. Eine Möglichkeit wäre, diese Systeme ebenfalls in Form eines Fachlichen Services anzubieten, der entweder direkt über die Benutzungsoberfläche oder vom Servicepointmanager verwendet wird.

Analysierbarkeit

Ein SfMS sollte die Aufzeichnung und Auswertung der von ihm unterstützten Vorgänge ermöglichen.

Diese Anforderung wird dahingehend erfüllt, dass zum einen die Servicepointscripts an jedem Servicepoint archiviert werden (vgl. Abschnitt 3.6.2), und zum anderen der Fortgang des gesamten Serviceflows in den einzelnen Servicefloats dokumentiert wird. Somit ist die Grundlage für eine Auswertung und kontinuierliche Verbesserung der Prozesse geschaffen. Bei der Realisierung des Prototypen wurde jedoch nicht das Problem berücksichtigt, dass abgearbeitete Servicefloats ebenfalls an einem geeigneten Ort archiviert werden müssen, damit diese zu einem späteren Zeitpunkt ausgewertet werden können.

Konnektivität

SfMS sollen den Informationsaustausch der Prozessbeteiligten unabhängig von heterogenen Systemlandschaften unterstützen.

Das Interoperabilitätsproblem wird durch die Verwendung von XML gelöst, das den plattformunabhängigen Datenaustausch „von Haus aus“ ermöglicht. Das Kohärenzproblem ist damit jedoch noch nicht gelöst, denn XML legt nicht die Semantik der Daten fest, die

beschrieben werden. D.h., es ist mindestens erforderlich, dass sich die beteiligten Organisationen auf eine gemeinsame DTD und die Bedeutung der darin definierten Tags einigen, um so auch dieses Problem überwinden zu können. Dies stellt jedoch eine organisatorische Fragestellung dar. Somit ist die Grundlage dafür geschaffen, dieser Anforderung erfolgreich zu begegnen.

Nachfolgende Tabelle 1 fasst die gewonnenen Erkenntnisse im Hinblick auf die Erfüllung der an SfMS gestellten Anforderungen durch den funktionalen Prototypen zusammen. Erkennbar ist, dass die meisten der Anforderungen nur zum Teil erfüllt worden sind. Hier ergeben sich Ansätze zur Weiterentwicklung des funktionalen Prototypen.

Anforderung	***Erfüllt***
Dynamische Modellierung und unmittelbare Adaptivität des Prozessmodells	Teilweise - keine Umordnung von Reihenfolgebeziehungen - die Aufnahme neuer Vor- und Nachbedingungen in das Servicepointscript ist nicht möglich - keine Neuzuweisung von Zuständigkeiten
Prozesstransparenz	Teilweise - keine Vergleichsfunktionen - kein Zugriffskonzept - keine Feststellung des Aufenthaltsortes eines Servicefloats
Prozessinstantiierung, -abarbeitung und –verwaltung	Teilweise - kein Transport - es können keine neuen Dokumente dem Servicepointscript hinzugefügt werden
Mehrbenutzerfähigkeit	Teilweise - kein Benutzungsmodell für gleichzeitigen Zugriff auf Prozessrepräsentationen
Integration	Teilweise - keine Integration mit anderen Systemen
Analysierbarkeit	Teilweise - keine Archivierung von Servicefloats
Konnektivität	Ja

Tabelle 1: Erfüllung des Anforderungskatalogs durch den funktionalen Prototypen

3.7.2 Erfüllung softwaretechnischer Entwurfsprinzipien

Die Diskussion der Erfüllung softwaretechnischer Kriterien durch den funktionalen Prototypen orientiert sich an den in Kapitel 3.2.3 genannten Entwurfsprinzipien, die bei der Softwareentwicklung allgemein beachtet werden müssen.

Kapselung

Das Prinzip der Kapselung bezieht sich auf die Beachtung des Geheimnisprinzips und der Trennung von Schnittstelle und Implementation.

Letzteres Entwurfskriterium wurde vollständig eingehalten. Zu jeder verwendeten Komponente existiert ein Interface, welches die angebotenen Leistungen spezifiziert. Dies konnte mit Hilfe des von Java angebotenen Interface-Konzeptes[83] leicht umgesetzt werden. Somit kann jede Komponente problemlos gegen eine andere Implementation ausgetauscht werden. So wäre es z.B. möglich, die DOM-basierende Implementationen der Prozessrepräsentationen gegen eine Variante auszutauschen, die mit anwendungsfachlich motivierten Objekten und nicht der eher technisch motivierten DOM-API arbeitet.

Das Geheimnisprinzip wird auf der Ebene des Servicepointmanagers ebenfalls vollständig eingehalten. Benutzende Komponenten haben keine Kenntnisse darüber, mit welchen Mitteln der Servicepointmanager seine Dienstleistungen erbringt und welches Persistenzmedium, bzw. Persistenzformat verwendet wird. Dieses wird zusätzlich gegenüber dem Servicepointmanager durch die Verwendung der Versorger verborgen.

Das Geheimnisprinzip wird jedoch an den Prozessrepräsentationen selbst durchbrochen. Erkennbar ist dies daran, dass die verwendeten Rückgabewerte der meisten Operationen lediglich Strings oder Listen von Strings liefern, was auch für Parameter gilt. Dies entspricht den Datentypen, die auf der Persistenzebene zur Speicherung der Attribute verwendet werden (die XML-API kennt hier nur Strings; vgl. Abschnitt 3.4.2). Dadurch wird die Realisierung der inneren Struktur der Prozessexemplare, also die Abbildung ihres Zustandes, nach außen sichtbar. Abhilfe würde hier die Verwendung weiterer anwendungsfachlich motivierter Klassen bzw. Objekte und benutzerdefinierte Werte (Fachwerte) schaffen (vgl. [Zül98, S. 316 ff.]).

Trennung von Zuständigkeiten

Diesem Entwurfsprinzip wird zunächst durch die Verwendung einer Ebenenarchitektur entsprochen, die einen Rahmen für die Aufteilung der Verantwortlichkeiten vorgibt. Durch Orientierung an der Entwurfsmetapher Fachlicher Service konnten die den einzelnen Schichten zuzuordnenden Aufgaben klar benannt werden. Diese werden vom Prototyp unter Verwendung der jeweils eingesetzten Technologien in den einzelnen Ebenen auch erfüllt.

Bei der Realisierung des Servicepointmanagers, der mehrere Aspekte des Umgangs mit den verwendeten Prozessrepräsentationen implementiert, ist ebenfalls auf eine Trennung der Verantwortlichkeiten geachtet worden. Seine Hauptaufgaben beziehen sich auf das Verwalten, das Erzeugen und das Versenden und Empfangen von Prozessrepräsentationen. Diese Aufgaben werden an andere Komponenten delegiert, deren Services der Servicepointmanager zur Erbringung seiner eigenen Leistung verwendet. Ihm selbst verbleibt die Aufgabe, die benötigten Komponenten zu initialisieren und die Koordination zu übernehmen. Die Zuständigkeiten der Teilnehmer der Entwurfsmuster ergeben sich aus den Musterbeschreibungen.

[83] Zum Interface-Konzept von Java siehe z.B. [MidSin99, S. 111 ff.]

Die Trennung von Zuständigkeiten auf der Ebene des Interaktionsservices kann teilweise durch Nutzung des Template-Komponenten-Konzeptes des StoryServers realisiert werden. Allerdings ist nahezu in jedem Template entweder Ablauflogik oder Funktionslogik einzubauen. Reine Präsentationskomponenten sind zwar möglich, sobald jedoch Gesichtspunkte ins Spiel kommen, die den Ablauf oder die Handhabung von Anwendungsfunktionalität betreffen, ist eine Vermischung von Präsentations- und Funktionslogik unumgänglich. Somit sollte weitestgehend darauf geachtet werden, dass komplexe Verzweigungslogik in eigenen Templates realisiert wird und die Anwendungslogik sich in Präsentationskomponenten maximal auf die Iteration von Ergebniswerten beschränkt. Somit ist bei der Realisierung der richtige Weg eingeschlagen worden, indem zustandsverändernde und komplexere Verzweigungsoperationen jeweils in einem Template gebündelt wurden, das keine Präsentationsaufgaben übernimmt.

Minimierte Kopplung

Die Kopplung der Komponenten des Servicepointmanagers ist als gering zu bezeichnen. Dort wo eine zyklische Abhängigkeitsbeziehung notwendig ist, wird das Beobachtermuster eingesetzt und so eine lose Kopplung erreicht (`FloatReceiver`). Alle anderen Komponenten stehen in einer einfachen einseitigen Benutztbeziehung zueinander, was auch für den ebenenübergreifenden Aspekt gilt. Die Prozessrepräsentationen werden unter verschiedenen, auf die jeweils benutzende Komponente zugeschnittenen Aspekte verwendet, wodurch die Schnittstellen kleiner und somit die Kopplung weiter reduziert wird.

Die Schnittstellen, unter denen die Prozessrepräsentationen von den Klienten (StoryServer-Templates) verwendet werden, sind allerdings zu breit. So muss z.B. eine Präsentationskomponente, die nur mit der Historie eines Servicefloat umgeht, die volle Schnittstelle des Servicefloats kennen. Dies wiederspricht dem Gedanken der Verwendung kleiner Schnittstellen. Dieser Aspekt macht schon auf ein grundlegendes Fehldesign der Prozessrepräsentationen aufmerksam, auf das am Ende der Diskussion noch einmal näher eingegangen wird.

Verwendung von Architektur- und Entwurfsmustern

Wie aus der Vorstellung der Realisierung des funktionalen Prototypen ersichtlich, basiert der Entwurf sehr stark auf der Verwendung von Architektur- und Entwurfsmustern. Zur Grobstrukturierung des Entwurfs wurde das schichtenbasierte Architekturmuster verwendet, wobei die den einzelnen Schichten zugewiesenen Aufgaben aus der Entwurfsmetapher Fachlicher Service abgeleitet wurde. Der Servicepointmanager selber ist weitestgehend auf der Basis auch auf diesen Einsatzkontext passender etablierter Entwurfsmuster aus dem Desktopbereich realisiert, wodurch u.a. die Zuordnung von Verantwortlichkeiten und die Reduktion der Kopplung wesentlich vereinfacht wurde.

Offen ist allerdings die Frage, ob noch weitere Entwurfsmuster angewandt, bzw. verwendete an anderen Stellen eingesetzt werden können. So wäre es z.B. denkbar, dass sich weitere Fachliche Services beim Servicepointmanager für bestimmte Ereignisse über das Beobachtermuster registrieren und so deren Zusammenarbeit realisiert wird. Ebenso könnte z.B. der Servicepointmanager mit einem Fachlichen Service zur Benutzerverwaltung zusammenarbeiten.

Die Verwendung des Singleton-Musters zur Realisierung des Servicepointmanagers wird weiter unten noch einmal diskutiert.

Der funktionale Prototyp entspricht in weiten Teilen den in Kapitel 3.2.3 genannten softwaretechnischen Entwurfsprinzipien. Tabelle 2 fasst die gewonnenen Ergebnisse zusammen.

Entwurfsprinzip	*Erfüllt*
Kapselung	Teilweise - interne Struktur der Prozessrepräsentationen „scheint durch“
Trennung von Zuständigkeiten	Ja
Minimierte Kopplung	Teilweise - zu breite Schnittstellen der Prozessrepräsentationen
Verwendung von Architektur- und Entwurfsmustern	Ja

Tabelle 2: Erfüllung allgemeiner softwaretechnischer Entwurfsprinzipien durch den funktionalen Prototypen

Diskussion der Realisierung der Prozessrepräsentationen

Bei der Diskussion der Einhaltung allgemeiner softwaretechnischer Kriterien wurde schon darauf hingewiesen, dass die Realisierung der Prozessrepräsentationen einige Schwächen aufweist, die an dieser Stelle noch einmal detaillierter dargestellt werden sollen.

Bemängelt wurde v.a., dass die Schnittstellen der Prozessrepräsentationen, die den Templates zur Benutzung angeboten werden, zu breit sind und das Geheimnisprinzip verletzt wird. Der Grund hierfür ist, dass die Klassen `ServicefloatImpl` und `ServicepointscriptImpl` zu ihrer eigenen Realisierung nicht auf weitere anwendungsfachlich motivierbare Klassen zurückgreifen, d.h. Zuständigkeiten an diese delegieren. Gemeinsame fachliche Eigenschaften und Umgangsformen der Prozessrepräsentationen wurden nicht dazu genutzt, daraus weitere Klassen abzuleiten bzw. in gemeinsamen Oberklassen zu abstrahieren. Die Prozessrepräsentationen implementieren jeweils die gesamten an ihnen benötigten Umgangsformen und operieren hierzu direkt auf der technischen DOM-API. Außerdem werden keine Fachwerte verwendet. Nachfolgend sind einige Methoden aus dem Interface `Servicefloat` aufgelistet, die diese Aspekte verdeutlichen (vgl. auch Abschnitt 3.6.3). Auf die Angabe von Exceptions ist hierbei verzichtet worden.

```
public List getListOfNamesOfPassedServicepoints();
public String getNameOfServicepointProvider(String nameOfServicepoint)
public void modifyContentOfDocument(String nameOfDocument, String newContent,
                                                        String contentName)
public void modifyValidUntilDateOfScheduledServicepoint(String
                                    nameOfScheduledServicepoint, String date);
```

Abbildung 33: Einige Methoden aus dem Interface `Servicefloat`

Erkennbar sind hier auf den ersten Blick mindestens drei weitere Klassen, die implementiert werden müssten: `Servicepoint`, `ServicepointProvider`, `Document`. Da es in einem Servicefloat mehrere Servicepoints, an einem Servicepoint mehrere Servicepointprovider (im Sinne von Anbietern der Dienstleistung) und auch mehrere Dokumente geben kann, gehören dazu noch entsprechende Behälterklassen (z.B. ein Dokumenten-Ordner), in denen die Objekte aufbewahrt werden können. Vergleichbare Methoden existieren auch am Servicepointscript.

Somit könnte eine Verkleinerung der Schnittstelle dadurch erreicht werden, dass ein Servicefloat nur Methoden anbietet, die diese Behälter herausgeben, an denen dann wiederum

entsprechende Methoden realisiert sind, die der benutzenden Komponente erlauben, die jeweils benötigten Objekte (z.B. ein Dokument) zu entnehmen. In dieser Realisierungsvariante würden sich die Schnittstellen von Servicefloat und Servicepointscript signifikant verkleinern. Außerdem würde sie der anwendungsfachlichen Bedeutung z.B. eines Servicefloats eher entsprechen, denn ein Servicefloat repräsentiert nichts anderes als einen fachlichen Behälter, in dem weitere fachliche Behälter (z.B. die Historie oder die Liste der noch zu durchlaufenden Servicepoints) und zusätzlich benötigte Informationen abgelegt sind.

Der Durchbruch der Kapsel kann sehr schön an der letzten beispielhaft angeführten Methode erkannt werden. Dort wird kein benutzerdefinierter Wert Datum übergeben, sondern stellvertretend ein String. Dies entspricht dem im DOM abgelegten Wert, die interne Realisierung „scheint“ also durch. Somit ist die Methode `modifyValidUntilDateOfScheduledServicepoint` dazu aufgefordert, sicherzustellen, dass es sich auch tatsächlich um ein Datum handelt. Außerdem muss die benutzende Komponente „wissen“ welches Format ein Datum haben soll, um Ausnahmen zu umgehen.

Der funktionale Prototyp weist an diesen Stellen somit erhebliche Schwächen auf und bedarf hier einer gründlichen Überarbeitung. Wie ein adäquates Design konkret aussehen könnte, kann aus Zeitgründen in dieser Arbeit nicht mehr umfassend geklärt werden und wird somit als offene Fragestellung festgehalten.

3.7.3 Beachtung der Besonderheiten webbasierter Anwendungen

In Kapitel 3.2.1 wurden die Besonderheiten aufgezeigt, durch die sich webbasierte Anwendungen gegenüber „normalen“ auszeichnen. An dieser Stelle werden die genanten Aspekte noch einmal aufgegriffen, um zu überprüfen, ob sie vom funktionalen Prototypen adäquat beachtet wurden.

Durch Verwendung der vierschichtigen Makroarchitektur wurden die Teile, welche die Präsentation und Handhabung des Prototypen betreffen, von den Teilen getrennt, die sich auf die Umsetzung des fachlichen Anteils der Anwendung beziehen. Somit sind die Aspekte, die sich auf die besondere Art der Präsentation und Handhabung im Webumfeld beziehen, beachtet worden. Die Zustandslosigkeit wurde durch Nutzung des URL-Erweiterungsmechanismus überwunden (vgl. Abschnitt 3.6.5). Auf den Einsatz weiterer Technologien zur Verbesserung der Präsentation und Handhabung (z.B. JavaScript) wurde allerdings verzichtet. Für den vorgesehenen Einsatzkontext stellte dies jedoch eine adäquate Lösung dar, was in Abschnitt 3.2.1 bereits begründet wurde.

Nicht beachtet wurde jedoch die Gefahr inkonsistenter Zustände die aus der hohen Wahrscheinlichkeit von Verbindungsabbrüchen und des konkurrenten Zugriffs auf gemeinsam genutzte Ressourcen resultiert. Dies ist daran erkennbar, dass der Servicepointmanager Servicefloat und –script als Originale an die Interaktionsservices herausgibt (vgl. Abbildung 26 in Abschnitt 3.6.2). Somit kann es zu Situationen kommen, in denen z.B. gleichzeitig zustandsverändernd auf ein und dasselbe Prozessexemplar zugegriffen wird. Dadurch wird der eindeutige Zustand der Prozesse gefährdet. Dieses Problem wird zusätzlich durch den Umstand verstärkt, dass DOM-Implementationen, wie z.B. JAXP von SUN nicht „Thread-Safe“ sind (vgl. [MorDavBoa01]).

Die Gefahr inkonsistenter Zustände wird weiterhin dadurch erhöht, dass z.B. im Falle des Weiterschaltens des Gesamtprozesses eine Folge von Operationen, sowohl am Servicepointscript als auch am Servicefloat, ausgeführt werden müssen, z.B. Auslesen der hinterlassenen Nachbedingungen aus dem Servicepointscript und Hineinschreiben in das Servicefloat. Mit dem von der Programmiersprache Java standardmäßig zur Verfügung

gestellten Möglichkeit der Synchronisation[84] von Methodenaufrufen, kann lediglich eine Methode ununterbrochen ausgeführt werden.

Diese Probleme könnten zum einen z.B. dadurch umgangen werden, dass Klienten nur mit Kopien arbeiten und diese nach Bearbeitung explizit an den Servicepointmanager zurückgeben, der die Originale dann nach einer bestimmten Strategie ersetzt (vgl. [LipWolZül01]).

Sollen mehrere Methoden auch an verschiedenen Objekten unteilbar ausgeführt werden, muss zum anderen auf mächtigere Konzepte wie z.B. Transaktionen (siehe z.B. [DenPet00, S. 161 ff.]) zurückgegriffen werden. Hiermit ist es dann auch möglich, den vor Beginn der unteilbar zusammengehörigen Folge von Aktionen vorhandenen Zustand wieder herzustellen und somit inkonsistente Zustände zu verhindern.

Ein weiteres Problem entsteht aus dem Zusammenhang der Verwendung des StoryServers und der Integration einer einfachen Java-Umgebung mittels TclBlend. Beim StoryServer ist es grundsätzlich möglich, mehrere sogenannte Delivery-Server (siehe [Vig99a]) einzusetzen, welche die Seitengenerierung und Auslieferung übernehmen. Jeder dieser Server hat seinen eigenen Seitengenerator. Da sie auf verschiedenen physikalischen Maschinen betrieben werden können, gibt es auch verschiedene virtuelle Maschinen und somit auch entsprechend viele verschiedene Exemplare des Servicepointmanagers. Hinzu kommt noch, dass ein Delivery-Server selbst noch einmal mehrere Tcl-Interpreter starten kann, um auf hohe Anfragevolumen reagieren zu können. Jedem Tcl-Interpreter wird eine virtuelle Maschine zugeordnet. Da die Testumgebung nur mit einem Server betrieben und die Reaktion auf hohe Benutzerzahlen nicht getestet wurde, hat sich dieses Problem zunächst nicht bemerkbar gemacht. Des weiteren wurde das Problem dadurch „verschattet“, dass nur ein einziges Template mit dem Servicepointmanager interagiert (vgl. Abschnitt 3.6.5), so dass die Reaktion auf die gleichzeitige Benutzung des Servicepointmanagers durch mehrere Templates nicht getestet werden konnte. Das Singleton-Muster stellt somit keine adäquate Lösung in diesem Kontext dar.

3.7.4 Empfehlung

Die Evaluation des funktionalen Prototypen in den vorangegangenen Abschnitten hat aufgezeigt, dass er an einigen Stellen noch nicht ausgereift ist und einer Verbesserung bedarf, bzw. wichtige Funktionen noch gar nicht implementiert sind, die benötigt werden, um eGovernment-Services sowohl auf Selbstbediener als auch Dienstleistungsanbieterseite zu unterstützen. Die Schwachpunkte des Prototypen verdeutlichen jedoch, wo weitere Ausbaustufen ansetzen können und welche Richtung im Hinblick auf den Entwurf einschlagen werden sollte. Im folgenden werden die wichtigsten Punkte noch einmal hervorgehoben. Dies ist im Sinne einer Empfehlung zu verstehen, wie eine Softwarearchitektur für Servicepoints zu gestalten ist.

Die Grobstrukturierung des Servicepoint in Anlehnung an das schichtenbasierte Architekturmuster stellt grundsätzlich eine gute Wahl dar. Da jede Schicht eine wohldefinierte Abstraktion anbietet und über eine wohldefinierte Schnittstelle (bzw. der Schnittstellen der innerhalb der Schicht realisierten Komponenten) mit ihrer Umwelt kommuniziert (Trennung von Schnittstelle und Implementation; Geheimnisprinzip), resultiert daraus eine leichtere Austausch- und Änderbarkeit, z.B. im Hinblick auf die Verwendung eines anderen Persistenzmediums oder eines zusätzlichen Interaktionsservices. Die Funktionalität des Servicepointmanagers kann leicht durch Hinzufügen neuer Komponenten erweitert und über seine Schnittstelle Klienten zur Verfügung gestellt werden, ohne dass diese

[84] Zum „synchronized“-Konstrukt der Programmiersprache Java siehe z.B. [MidSin99, S. 399 ff.]

Kenntnisse über die konkrete Realisierung haben müssen oder existierende Klienten davon beeinflusst werden.

Wie viele Schichten (mindestens) benötigt werden und welche Abstraktion in jeder Schicht vorgenommen wird, kann durch Orientierung an der Entwurfsmetapher Fachlicher Service bestimmt werden. Somit ist die Positionierung der einzelnen Softwarekomponenten in den Schichten leicht möglich. Durch die Ausrichtung an dieser Entwurfsmetapher ist es möglich, die grundlegende fachliche Funktionalität eines Servicepoints unabhängig von Handhabung und Präsentation und somit zur Wiederverwendung in verschiedenen Einsatzkontexten (Self-Servicepoint und Servicepoint) zu entwickeln.

Die Verwendung von schichtenbasierten Architekturen als Makroarchitektur für webbasierte Servicepoints und die Orientierung an der Entwurfsmetapher Fachlicher Service ist somit grundsätzlich zu empfehlen:

> *Empfehlung*: Verwende schichtenbasierte Softwarearchitekturen und orientiere Dich an der Entwurfsmetapher Fachlicher Service zur Bestimmung der Verantwortlichkeiten.

Die verwendeten Entwurfsmuster zur Ausgestaltung der Mikroarchitektur des Servicepointmanagers haben sich ebenfalls bewährt. Sie haben zu einer übersichtlichen Strukturierung und Aufteilung der Verantwortlichkeiten beigetragen und somit die Kohäsion des Servicepointmanagers erhöht. Des weiteren haben sie dabei geholfen, die in Abschnitt 3.1.2 formulierten Ziele einzuhalten. So können ohne großen Änderungsaufwand neue Materialien am Servicepoint verwaltet werden oder das verwendete Persistenzmedium sowie das Persistenzformat gegen ein anderes ausgetauscht oder um ein zusätzliches erweitert werden (vgl. Abschnitt 3.6.2). Durch obige Diskussion ist jedoch deutlich geworden, dass die Wahl des Singleton-Musters, dessen Verwendung in Kontexten mit mehreren Adressräumen ungeeignet ist, nicht zu empfehlen ist.

> *Empfehlung*: Orientiere Dich zur Ausgestaltung der Mikroarchitektur des Servicepointmanagers an den verwendeten Entwurfsmustern. Vermeide jedoch das Singleton-Muster zur Realisierung des Servicepointmanagers.

Sollen dagegen mehrere Servicepointmanager-Exemplare verwendet werden, hätte dies grundsätzliche Auswirkungen auf die Architektur. Entweder müssten sich die Exemplare darüber austauschen, welche Prozessrepräsentationen gerade von welchem Servicepointmanager verwaltet werden, oder die Kommunikation wird über ein gemeinsames Persistenzmedium realisiert. Letztere Variante wird für hoch skalierende Umgebungen vorgeschlagen (vgl. z.B. [MeiSch01]). Die Autoren dieses Artikels empfehlen die Benutzung zustandsloser Objekte, die bei jeder Anfrage (einem Methodenaufruf) das jeweils betroffene Objekt z.B. aus einer Datenbank holen, eine Modifikation durchführen und anschließend wieder in die Datenbank zurückstellen. Welches der existierenden Server die jeweilige Anfrage gerade bearbeitet wäre somit irrelevant; die Last könnte besser verteilt werden.

Die Verwendung des Materialmanagers bzw. dessen Interpretation wäre im Hinblick auf die Modifikation des Prototypen unter den angesprochenen Gesichtspunkten anzupassen. Er ist so ausgelegt, dass alle Prozessrepräsentationen, die an einem Servicepoint existieren, in den Hauptspeicher bzw. das Materialstore geladen werden (vgl. Abschnitt 3.6.2). Das bedeutet zwar im Hinblick auf die Verwendung eines Servicepointmanager-Exemplares einen Performanzgewinn, da die jeweiligen Prozessexemplare nicht jedes Mal erst aus dem XML-File rekonstruiert werden müssen und so quasi ein Cache zur Verfügung steht. Werden jedoch mehrere Servicepointmanager verwendet, resultiert dies, wie bereits angedeutet, in einen zusätzlichen Abstimmungsaufwand zwischen den Exemplaren im Hinblick darauf, welcher Manager welchen Prozess verwaltet. Aus diesem Grund ist unter dem Aspekt einer einfachen Realisierung zu empfehlen, entsprechend der Idee der zustandslosen Objekte die Prozessrepräsentationen im Persistenzmedium zu belassen und so eine Kooperation

verschiedener Servicepointmanager-Exemplare über dieses Medium zu ermöglichen. Der Materialstore würde in dieser Variante nicht mehr benötigt werden.

Welche weiteren Auswirkungen die Verwendung dieses Ansatzes auf die gesamte Architektur des Servicepointmanagers hat, kann an dieser Stelle jedoch nicht mehr diskutiert werden.

> *Empfehlung*: Verwende zustandslose Objekte zur Realisierung des Umgangs mit den Prozessrepräsentationen in hoch-skalierenden Umgebungen.

Eine Orientierung an dem Entwurf der Prozessrepräsentationen selber kann nicht empfohlen werden. Hier ist ein Design erforderlich, das den Umstand beachtet, dass sich beide Prozessrepräsentationen durch gemeinsame Umgangsformen und Eigenschaften auszeichnen. Durch gemeinsame Oberklassen und zusätzliche anwendungsfachlich motivierte Klassen (z.B. für Aktivitäten und Servicepoints) würden die Schnittstellen für die Klienten zum einen signifikant verkleinert werden. Zum anderen würde dadurch die Erweiterbarkeit der Prozessrepräsentationen erleichtert und der Entwurf verständlicher werden.

Als gemeinsame Oberklasse könnte z.B. eine abstrakte Klasse[85] vorgesehen werden, die beispielsweise `Processrepresentation` heißen könnte, welche das Zustandsmodell der Prozessrepräsentationen (Historie, aktuelles Prozesselement, zukünftige Prozesselemente) und den hierauf bezogenen Umgang mit den Prozesselementen (Aktivität und Servicepoint) implementiert. Aktivität und Servicepoint könnten selber wieder von einer Oberklasse erben (z.B. `Processelement`), die von `Processrepresentation` verwendet wird. Historie und zukünftige Prozesselemente stellen z.B. geeignete Kandidaten für zusätzliche Behälterklassen dar. Sie unterscheiden sich lediglich darin, dass Elemente der Historie nicht mehr verändert werden dürfen. Die meisten ihrer Eigenschaften und Umgangsformen könnten somit ebenfalls in einer Oberklasse abstrahiert werden.

> *Empfehlung*: Siehe zur Realisierung der Prozessrepräsentationen zusätzliche anwendungsfachlich motivierte Klassen vor. Abstrahiere gemeinsame Eigenschaften und Umgangsformen in (abstrakten) Oberklassen.

Für die sinnvolle Strukturierung des Interaktionsservices empfiehlt sich, Templates, die auch Informationen aus den Prozessrepräsentationen darstellen, mit so wenig Anwendungs- und Ablauflogik wie möglich auszustatten, um die „Vermischung“ dieser eigentlich getrennt zu realisierenden Bereiche so minimal wie möglich zu halten. Ablauflogik sollte sich maximal auf die Iteration von Ergebniswerten beschränken. Für zustandsverändernde Operationen und komplexere Verzweigungen sollten eigene Templates verwendet werden.

> *Empfehlung*: Statte Templates, die Präsentationsaufgaben übernehmen, mit so wenig Anwendungslogik wie möglich aus. Verwende für zustandsverändernde Operationen und komplexere Verzweigungen eigene Templates.

Zur Realisierung der fachlichen Funktionalität des funktionalen Prototypen wurde auf die Technologien Java 2 Standard Edition und XML zurückgegriffen. Grundsätzlich ist die Wahl von XML in Kombination mit Java zu empfehlen, da so die Portabilitätseigenschaft beider Technologien ausgenutzt werden kann und ein Servicepoint, der auf diesen Technologien aufsetzt, lediglich das Vorhandensein einer virtuellen Maschine, die für nahezu alle Plattformen existiert, erfordert. Java als objektorientierte Sprache bringt neben den allgemeinen Vorteilen einer objektorientierten Programmiersprache[86] zusätzlich ein Interface-Konzept mit, das im Rahmen des Prototypen gut verwendet werden konnte. XML stellt

---

[85] Zum Thema abstrakte Klassen in Java siehe z.B. [MidSin99].

[86] z.B. Kapselung, Vererbung, Polymorphismus, usw. [MidSin99, S. 6 ff.]

mittlerweile den DeFacto-Standard zur Realisierung von organisationsübergreifenden Geschäftsprozessen dar, bei denen geschlossene Systeme verwendet werden müssen. Es entstehen immer mehr Produkte und Standards, die auf Basis der Kombination beider Technologien aufsetzen und somit spricht z.B. im Hinblick auf Investitionssicherheit alles für den kombinierten Einsatz.

Im Rahmen der Programmiersprache Java werden mittlerweile zusätzlich eine Reihe von Technologien angeboten, die z.B. zum Zwecke der Integration vorhandener Anwendungsfunktionalitäten an einem Servicepoint Verwendung finden können, beispielsweise RMI, CORBA oder JDBC[87] (vgl. z.B. [OrfHar97]). Viele Produkte, deren Services im Rahmen eines Serviceflows genutzt werden sollen, bieten auch standardmäßig eine Java-API an, über die eine Integration realisiert werden kann. So wäre es über diesen Weg z.B. möglich, auf Supportprozesse zugreifen zu können, die im Rahmen des WfMS *MQSeries Workflow* von IBM ablaufen (vgl. [IBM98]).

Nachteilig ist jedoch, dass von der Standard Ausgabe von Java kein Transaktionskonzept angeboten wird und die Performanz oftmals zu wünschen übrig lässt. Da die Wahrscheinlichkeit von Ausfällen im Untersuchungskontext recht groß ist, sollte statt dessen die Enterprise-Ausgabe von Java (Java 2 Enterprise Edition, J2EE; siehe z.B. [DenPet00]) Verwendung finden, in dessen Rahmen neben Transaktionen z.B. auch das Konzept der *Stateless Session Beans* angeboten wird, das als Grundlage zur Realisierung zustandsloser Objekte verwendet werden kann (vgl. [MidSch01]).

Die Verwendung des DOM zur Implementation der zustandssondierenden und zustandsverändernden Operationen an den Prozessrepräsentationen hat sich zum einen Aufgrund der recht einfachen Handhabung der DOM-API bewährt. Ein weiterer Vorteil ist die Einfachheit der Transformation vom XML-File in das DOM und wieder zurück. Dies ist zwar nicht standardisiert, wird aber von den meisten Parser-Herstellern angeboten und in zukünftigen DOM-Levels berücksichtigt werden[88]. Somit entfällt die Notwendigkeit der Implementationen eines aufwendiges Transformationsprotokolls.

DOM-Implementationen weisen allerdings den Nachteil auf, dass sie nicht „Thread-Safe“ sind, wie es z.B. bei der Implementation von Sun der Fall ist (vgl. [MorDavBoa01]). Somit sollte, wie weiter oben schon angedeutet, von Klienten nur auf Kopien des Originals gearbeitet werden, denn dann hat ein Klient immer nur ein DOM und somit kann das Problem vermieden werden.

Die Verwendung von Java und XML als Basistechnologie zur Realisierung von Servicepoints ist also grundsätzlich zu empfehlen. Wird eine große Benutzerzahl erwartet, so dass man sich auch mit Nebenläufigkeitsaspekten und Lastverteilung auseinandersetzen muss, sollte der Einsatz der Enterprise-Ausgabe von Java „ins Auge gefasst“ werden.

> *Empfehlung*: Verwende Java und XML. Sind Nebenläufigkeitsaspekte und Lastverteilung von Relevanz, dann verwende eine Implementation der J2EE.

Die Vermeidung der Herausgabe von Originalen, die bis jetzt lediglich technisch motiviert wurde, scheint auch ein gutes Benutzungsmodell des Servicepoints aus anwendungsfachlicher Perspektive darzustellen. Laut [Zül98, S. 432] sollte man ein solches Benutzungsmodell wählen, wenn der konkurrierende Zugriff sich meist auf lesende und selten auf schreibende Zugriffe bezieht. Kopien werden mit Zeitstempeln versehen, die bei der Ersetzung des Originals nach einer expliziten Änderungsoperation „zu Rate gezogen“ werden. Kommt es zu

---

[87] RMI steht für Remote Method Invocation; CORBA steht für Common Object Request Broker Architecture; JDBC steht für Java Database Connection.

[88] http://www.w3.org/TR/2001/WD-DOM-Level3-CMLS-20010209 (03.2001)

Konflikten, d.h. ist die das Original ersetzende Kopie älter als das Original, dann wird der Benutzer über diesen Konflikt informiert und die Benutzer können sich dann außerhalb des Systems über den Umgang mit solchen Situationen abstimmen.

Dieses Benutzungsmodell empfiehlt sich, da die Wahrscheinlichkeit, dass gleichzeitig an einem Servicepoint auf das selbe Prozessexemplar verändernd zugegriffen wird, nicht sehr hoch ist. An Self-Servicepoints kann diese Situation nahezu ausgeschlossen werden, da hier davon ausgegangen werden kann, dass lediglich der Selbstbediener auf das seinen Prozess repräsentierende Prozessexemplar zugreift. Auch an Servicepoints kann damit gerechnet werden, dass ein gleichzeitiger verändernder Zugriff sehr selten stattfinden wird. Dies resultiert daraus, dass davon ausgegangen wird, dass zu einem Zeitpunkt immer nur eine Aktivität (die aktuelle) durchgeführt wird, was normalerweise nur von einem Akteur gemacht wird. Es sind jedoch auch Situationen denkbar, in denen ein Servicepointscript im Back-Office bearbeitet wird, und ein anderer Akteur das Servicepointscript im Front-Office zugreift, um einem Kunden Informationen über den aktuellen Bearbeitungszustand mitzuteilen. Hier findet normalerweise nur ein lesender Zugriff statt. Evtl. könnte es jedoch zu einer Modifikation des Prozessmodells aufgrund veränderter Kundenwünsche kommen, woraus eine gleichzeitige ändernde Zugriffssituation entstehen könnte. Des weiteren könnte auch von der starren Reihenfolge der Durchführung der Aktivitäten abgewichen werden (immer nur eine aktuelle Aktivität), wenn Aktivitäten an einem Servicepoint nicht in einer Abhängigkeitsbeziehung entsprechend der „flow dependencies“ stehen und auch die Möglichkeit der gleichzeitigen Durchführung von Aktivitäten vorgesehen wird. In einer solchen Variante (die nicht vom Prototyp realisiert ist) könnte es dann ebenfalls zu den angesprochenen Konfliktsituationen kommen.

Empfehlung: Siehe ein Benutzungsmodell dahingehend vor, dass der Zugriff auf Prozessrepräsentationen immer nur auf Kopien möglich ist.

Der grundlegende Ansatz, Prozesse durch Servicefloats und Servicepointscripts zu vergegenständlichen und Servicefloats von Servicepoint zu Servicepoint zu verschicken, wobei von jedem Servicepoint die zur Bearbeitung notwendige Funktionalität angeboten wird, ist grundsätzlich eine gute Ausgangsbasis, um die im Servicebereich geforderte Flexibilität auf der Basis von Standardabläufen anbieten zu können. Allerdings wurde der Aspekt des Transports zwischen Servicepoints bei der Realisierung des funktionalen Prototypen nur „am Rande“ durch Einführen der Komponenten `FloatReceiver` und `FloatSender` berücksichtigt.

Im Hinblick auf die Umsetzung des Transports von Servicefloats zwischen den Servicepoints bietet sich auf technischer Ebene die Verwendung von XML-Massaging-Systemen an, die zunehmend im Kontext organisationsübergreifender Geschäftsprozesse entwickelt werden. Hier könnte man z.B. auf JAXM[89] der Firma Sun zurückgreifen. JAXM ist ein prototypisches XML-basiertes Massaging System, mit deren Hilfe XML-Dokumente auf Basis der Transportmechanismen des Internets, z.B. FTP oder http, zwischen unterschiedlichen Teilnehmern ausgetauscht werden können[90].

Beim Transport von Servicefloats, bzw. beim Zugriff auf Prozessrepräsentationen, müssen auch Sicherheitsaspekte berücksichtigt werden. Diese Frage ist in der bisherigen Diskussion ausgeklammert worden. Auch hier gibt es erste Ansätze, wie z.B. der von [GreSchHae01]. Dort werden sogenannte „Secure XML Document Containers“ verwendet, die ebenfalls auf den Technologien XML und Java aufsetzen, zusätzlich aber noch Digitale Signaturen verwenden. Das Konzept zielt auch auf die Realisierung von eGovernement-Prozessen ab und

[89] JAXM steht für „Java API for XML Massaging“.

[90] http://developer.java.sun.com/developer/earlyAccess/mproject (03.2001)

ist grundsätzlich mit dem in dieser Arbeit vorgestellten Konzept vergleichbar. Für eine Weiterentwicklung unter diesem Gesichtspunkt empfiehlt sich somit eine Orientierung an dem Ansatz.

Empfehlung: Setze zur Realisierung des Transports zwischen Servicepoints ein XML-basiertes Messaging-System ein, das auf den Protokollen des Internets aufsetzt. Beachte auch Sicherheitsaspekte.

3.7.5 Zusammenfassung

In diesem Kapitel wurde der funktionale Prototyp für einen webbasierten Servicepoint im Hinblick auf die Erfüllung des in Kapitel 2.3.3 erstellten Anforderungskatalog, den in Kapitel 3.2.3 gestellten allgemeinen softwaretechnischen Anforderungen und der Berücksichtigung der Besonderheiten webbasierter Anwendungen evaluiert.

Die wesentlichen Ergebnisse der Evaluation werden im folgenden stichpunktartig noch einmal zusammengefasst:

- Die anwendungsfachlichen und technischen Anforderungen konnten nur zum Teil erfüllt werden.
- Vom Anforderungskatalog konnten folgende Punkte nicht erfüllt werden:
  - Es ist keine Umordnung der Reihenfolge von Servicepoints in Servicefloats bzw. keine Umordnung der Reihenfolge von Aktivitäten in Servicepointscripts möglich.
  - Es können keine neuen Bedingungen in Servicepointscripts aufgenommen werden.
  - Die Neuzuweisung von Zuständigkeiten ist nicht möglich.
  - Es können keine neuen Dokumente in Servicepointscripts aufgenommen werden.
  - Es werden keine Vergleichsoperationen angeboten, die eine Gegenüberstellung des aktuellen Prozessverlaufs mit dem geplanten Verlauf erlauben.
  - Es fehlt ein Zugriffskonzept.
  - Der Aufenthaltsort von Servicefloats kann nicht festgestellt werden.
  - Der Transport von Servicefloats zwischen Servicepoints ist nicht möglich.
  - Es existiert kein Benutzungsmodell für den gleichzeitigen Zugriff auf ein Prozessexemplar.
  - Die Integration anderer Systeme ist nicht betrachtet worden.
  - Es ist nicht geklärt, wie mit „durchgelaufenen“ Servicefloats umgegangen werden soll.
- Aus technischer Sicht ergeben sich folgende Mängel:
  - Die Realisierung der Prozessrepräsentationen muss im Hinblick auf das Design überarbeitet werden. Es sind zusätzliche Klassen erforderlich, die von den Prozessrepräsentationen zur internen Strukturierung und zum Umgang mit ausgewählten Aspekten seitens der Klienten (Templates) verwendet werden können.
  - Die Besonderheit des Webs im Hinblick auf eine hohe Wahrscheinlichkeit von Verbindungsabbrüchen wurde nicht berücksichtigt. Dadurch, dass Prozessrepräsentationen als Originale vom Servicepointmanager

herausgegeben werden, wird die Möglichkeit von inkonsistenten Zuständen unnötig erhöht.

Unter Beachtung der identifizierten Probleme und des vorliegenden Entwurfs, wurden folgende Empfehlungen gegeben:

- Verwende schichtenbasierte Softwarearchitekturen für webbasierte Servicepoints und orientiere Dich an der Entwurfsmetapher Fachlicher Service zur Bestimmung der Verantwortlichkeiten.
- Orientiere Dich zur Ausgestaltung der Mikroarchitektur des Servicepointmanagers an den verwendeten Entwurfsmustern. Vermeide jedoch das Singleton-Muster zur Realisierung des Servicepointmanagers.
- Verwende zustandslose Objekte zur Realisierung des Umgangs mit den Prozessrepräsentationen in hoch-skalierenden Umgebungen.
- Siehe zur Realisierung der Prozessrepräsentationen zusätzliche anwendungsfachlich motivierte Klassen vor. Abstrahiere gemeinsame Eigenschaften und Umgangsformen in (abstrakten) Oberklassen.
- Statte Templates, die Präsentationsaufgaben übernehmen, mit so wenig Anwendungslogik wie möglich aus. Verwende für zustandsverändernde Operationen und komplexere Verzweigungen eigene Templates.
- Verwende Java und XML. Sind Nebenläufigkeitsaspekte und Lastverteilung von Relevanz, dann verwende eine Implementation der J2EE.
- Siehe ein Benutzungsmodell dahingehend vor, dass der Zugriff auf Prozessrepräsentationen immer nur auf Kopien möglich ist.
- Setze zur Realisierung des Transports zwischen Servicepoints ein XML-basiertes Messaging-System ein, das auf den Protokollen des Internets aufsetzt. Beachte auch Sicherheitsaspekte.

Trotz der aufgeführten Mängel des funktionalen Prototypen kann die Orientierung an seinem Entwurf zur Gestaltung der Architektur eines Servicepoints für Anwendungssituationen, in denen nicht mit einer hohen Benutzerzahl zu rechnen ist, empfohlen werden (bis auf das Singleton-Muster in Verbindung mit dem StoryServer). Der Servicepointmanager zeichnet sich durch eine übersichtliche Strukturierung aus. Der Interaktionsservice kann ohne größere Probleme gegen eine andere Technologie, z.B. JSP ausgetauscht werden. Ebenso ist das Persistenzmedium leicht durch andere, z.B. Datenbanken, ersetz- bzw. ergänzbar. Die Prozessrepräsentationen bedürfen jedoch einer gründlichen Überarbeitung.

4 Zusammenfassung und Ausblick

Gegenstand der vorliegenden Arbeit war die Beantwortung der Fragestellung, wie eine Softwarearchitektur für webbasierte Servicepoints gestaltet sein sollte, auf deren Basis eine Prozessunterstützung für eGovernment-Services unter Verwendung der Technologien Java und XML realisiert werden kann.

Zu Beginn dieser Arbeit wurde zunächst grundlegend auf die Begriffe Dienstleistungen und eGovernment-Services eingegangen und das Anwendungsfeld auf den Bereich eingeschränkt, der sich auf die Unterstützung der Zugänglichmachung und der Durchführung öffentlicher Transaktionsdienstleistungen mit Hilfe von IKT bezieht.

Als in dieser Arbeit gewählter Ansatz zur Entwicklung von Software zur flexiblen Unterstützung arbeitsteiliger Prozesse wurde SfM vorgestellt. SfM zielt auf routinisiert ablaufende komplexe Dienstleistungsprozesse ab, die auch die Organisationsgrenzen überschreiten können und deren Teilleistungen aufeinander aufbauen. Ausgehend von diesem Ansatz wurde ein Anforderungskatalog erstellt, den Softwaresysteme erfüllen müssen, um Prozessteilnehmern eine adäquate Unterstützung an Servicepoints anbieten zu können.

Es wurde gezeigt, dass SfM auch dazu geeignet ist, eGovernment-Prozesse direkt zum Kunden zu verlängern, ohne dass dies zu einem konzeptionellen Bruch führen würde. Hierzu wurde der Ansatz Self-Service über Kunden- bzw. Bürgerprozessportale in den SfM-Ansatz eingeordnet, der im eGovernment-Bereich zur Steigerung der Bürgerfreundlichkeit verfolgt wird. Die Teilleistungen im Rahmen eines Serviceflows, die sich durch Self-Service-Fähigkeit und einen ausreichend hohen Self-Service-Grad auszeichnen, können über diese Kontaktstelle dem Kunden direkt zur Verfügung gestellt werden. Als Beispiel für eine solche eGovernment-Transaktionsdienstleistung wurde das Anwendungsbeispiel „elektronische Beantragung von Briefwahlunterlagen über www.hamburg.de“ vorgestellt.

Als wesentlicher Gegenstand zur Beantwortung der Frage nach einer geeigneten Softwarearchitektur für webbasierte Servicepoints aus technischer Sicht diente ein funktionaler Prototyp, der zur Umsetzung des Self-Service-Anteils des Anwendungsbeispiels konstruiert worden ist. Zunächst wurde der Entstehungskontext des Prototypen und die während seiner Konstruktion getroffenen Entwurfsentscheidungen, sowie die hierbei verfolgten Ziele dargestellt. Die Ziele (z.B. leichte Austauschbarkeit der Prozessrepräsentationen) konnten durch Orientierung an bestehenden Entwurfsmustern, die zur Realisierung der Verwaltungseinheit am Servicepoint (Servicepointmanager) eingesetzt worden sind, eingehalten werden.

Für die Makroarchitektur des Prototypen wurde eine vierschichtige Softwarearchitektur gewählt, die es erlaubt, den Einfluss der Besonderheiten webbasierter Benutzungsoberflächen (Zustandslosigkeit, einseitige Interaktion, eingeschränkte Rückkopplung) auf die eigentliche Fachanwendung zu minimieren, so dass diese ohne großen Änderungsaufwand in verschiedenen Einsatzkontexten wiederverwendet werden kann. Die Anleitung hierzu lieferte die Entwurfsmetapher Fachlicher Service.

Als Grundlage zur softwaretechnischen Abbildung arbeitsteiliger Prozesse werden im SfM-Ansatz die standardisierbaren Teilleistungen im Rahmen eines Dienstleistungsprozesses auf Serviceflows als Folgen von Servicepoints abgebildet, wobei die standardmäßig an einem Servicepoint durchzuführenden Aktivitäten selbst wieder als Teilprozesse modelliert werden. Diese Prozessmodelle werden in Form von Servicefloats und Servicepointscripts vergegenständlicht, um den Prozessteilnehmern eine flexible Prozessunterstützung an Servicepoints anbieten zu können. Auf Basis dieses Konzeptes und nach Vorstellung der im SfM-Ansatz verwendeten Modellierungs- und Analysemittel wurden die Umgangsformen abgeleitet, die von den Prozessrepräsentationen angeboten werden müssen, um den Akteuren

einen adäquaten IKT-gestützten Umgang mit ihnen zu ermöglichen. Diese dienten als Ausgangsbasis zur softwaretechnischen Realisierung der vom Prototypen verwendeten Prozessrepräsentationen.

Nach Vorstellung der zur Realisierung des webbasierten Servicepoints eingesetzten Technologien XML, Java, StoryServer 5.0 und TclBlend wurde die Ausgestaltung der Mikroarchitektur des Servicepointmanagers, die Umsetzung der Prozessrepräsentationen und der Benutzungsoberfläche für das Anwendungsbeispiel präsentiert.

Der funktionale Prototyp ist abschließend im Hinblick auf die Erfüllung des Anforderungskatalogs, allgemeiner softwaretechnischer Entwurfsprinzipien sowie der Beachtung der Besonderheiten webbasierter Anwendungen evaluiert worden. Hierdurch konnten wesentliche Schwächen und noch fehlende Funktionalitäten aufgedeckt werden. Diese Punkte wurden genutzt, um eine Empfehlung im Hinblick auf die Gestaltung einer Softwarearchitektur für webbasierte Servicepoints sowohl auf Makro- als auch Mikroebene zu geben. Als wesentliches Ergebnis lässt sich anführen, dass eine Orientierung an der vierschichtigen Makroarchitektur und der Ausgestaltung des Servicepointmanagers (sofern nicht mit hohen Benutzerzahlen zu rechnen ist) empfohlen werden kann. Um auch mit konkurrenten Zugriffsituationen umgehen zu können, ist die Umsetzung eines passenden Benutzungsmodells notwendig. Die Orientierung an der Realisierung der Prozessrepräsentationen ist hingegen nicht empfehlenswert.

Aus den Ergebnissen und offenen Fragestellungen, die im Verlauf der Arbeit aufgetreten sind, lassen sich einige Ansatzpunkte ableiten, die als Grundlage für weitere Arbeiten in diesem Bereich dienen könnten. Diese sind zum Abschluss der Arbeit nachfolgend in loser Zusammenstellung aufgeführt.

Als grundsätzliches Problem im Kontext webbasierter Anwendungen wurde die hohe Gefahr inkonsistenter Zustände resultierend aus der hohen Wahrscheinlichkeit von Verbindungsabbrüchen und konkurrenter Zugriffsituationen hervorgehoben. Zusätzlich wurde betont, dass die gewählte Architektur bei hohen Benutzerzahlen schnell an ihre Grenzen stößt. Es empfiehlt sich hier der Einsatz der J2EE. Hieraus resultiert die Fragestellung, wie eine Softwarearchitektur für Servicepoints auf der Basis von J2EE aussehen könnte, die darauf ausgelegt ist, diesen Besonderheiten zu begegnen. Dieser Fragestellung wird z.B. in der sich in der Entstehung befindlichen Diplomarbeit von Nol Shala [Sch01] nachgegangen.

Weitere offene Fragestellungen betreffen die Aspekte des Transports von Servicefloats zwischen Servicepoints und der Nachverfolgung, also der Bestimmung des Aufenthaltsortes von Servicefloats, wenn sich z.B. ein Selbstbediener über den Zustand der Bearbeitung einer angestoßenen Transaktionsdienstleistung informieren möchte. In diesem Zusammenhang könnte untersucht werden, was XML-Messaging-Systeme anbieten. Interessant wäre auch, wie ein Auswertungswerkzeug aussehen könnte, das zur Analyse von Servicepointscripts und Servicefloats eingesetzt werden soll.

Weiterhin wurde der Aspekt der nebenläufigen Durchführung von Transaktionsdienstleistungen, also die Aufteilung und Synchronisation von Servicefloats, aus der Diskussion ausgeklammert. Hierzu kann auch der Aspekt der Einbindung von Supportprozessen, die beispielsweise auf der Basis von WfMS realisiert sind, die neben dem eigentlichen Serviceflow ablaufen, gezählt werden. Wie diese Anforderungen bewältigt werden können, stellt ebenfalls ein interessantes Themengebiet für weitere Arbeiten dar.

Literaturverzeichnis

[ALPR01] R. Alt, F. Leser, T. Puschmann, C. Reichmayr: *Business Networking Architektur*. Universität St. Gallen. http://ccbn. iwi.unisg.ch/Results/files/Business Networking Architektur fls35.pdf (03.2001)

[And00] R. Anderson et al.: *XML Professionell*. MITP-Verlag, 2000.

[Bäu98] D. Bäumer: *Softwarearchitekturen für die rahmenwerksbasierte Konstruktion großer Anwendungssysteme*. Dissertationsschrift zur Vorlage am Fachbereich Informatik der Universität Hamburg, 1998.

[Bal99] S. Ball: *Web TCL Complete*. McGraw-Hill, 1999.

[BarMasMcC94] T. Barners-Lee, L. Masinter, M. McCahill: *Uniform Resource Locators (URL)*. RfC 1738, 1994. http:// www.cis.ohio-state.edu/htbin/rfc/ rfc1738.html (02.2001)

[BBKKSS99] W. Becker, C. Burger, J. Klarmann, O. Kulendik, F. Schiele, K. Schneider: *Rechnerunterstützung für Interaktionen zwischen Menschen*. In: Informatik Spektrum, 22, Dezember, 1999.

[Ber86] L. Berekoven: *Der Dienstleistungsmarkt – Sachliche Besonderheiten und empirische Befunde"*. In: E. Pestel (Hrsg.): *Perspektiven der Dienstleistungs-wirtschaft*. Vandenhoeck & Ruprecht, 1986.

[BlaTes99] T. Blank, O. Tesmer: *Workflow-Unterstützung für den universitären Lehrplanungsprozess: Möglichkeiten und Grenzen bei der Umsetzung mit „FlowMark"*. Studienarbeit am Fachbereich Informatik, Universität Hamburg, Arbeitsbereich Softwaretechnik, 1999.

[BMRSS96] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal: *Pattern-Oriented Software Architecture – A System of Patterns*. Wiley, 1996.

[Bod99] F. Bodendorf: *Wirtschaftsinformatik im Dienstleistungsbereich*. Springer-Verlag, 1999.

[BogKiß95] J. Bogumil, L. Kißler: *Bediente Kunden?* 1995. http://www.fernuni-hagen.de/POLAD/download/1995-2.doc (12.2000)

[BPSM00] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler (Eds.): *Extensible Markup Language (XML) 1.0 (Second Edition)*. W3C Recommendation, 6. Oktober, 2000. http://www.w3c.org/TR/2000/REC-XML-20001006.pdf (03.2001)

[Bro88] Brokhaus Enzyklopädie, 5. Band, COT-DR, 19. Auflage, F.A. Brokhaus Verlag, 1988.

[Bro93] Brokhaus Enzyklopädie, 2. Band, APU-BEC, 19. Auflage, F.A. Brokhaus Verlag, 1993.

[BTZZ00] H. Büchner, D. Traub, R. Zahradka, O. Zschau: *Web Content Management – Websites professionell entwickeln*. Galileo Business, Bonn, 2000.

[DenPet00] S. Denninger, I. Peters: *Enterprise JavaBeans*. Addison-Wesley, 2000.

[DIN98] DIN-Fachbericht: *Service Engeneering – Entwicklungsbegleitende Normung für Dienstleistungen*. Beuth Verlag GmbH, 1998.

[Eck00] R. Eckstein: *XML – Kurz und gut*. O'Reilly Verlag, 2000.

[Ert86] R. Ertel: *Was sind Dienstleistungen? Definitorische Anmerkungen*. In: E. Pestel: *Perspektiven der Dienstleistungs-wirtschaft*. Vandenhoeck & Ruprecht, 1986.

[Fie97] R. Fielding, UC Irvine, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee: *Hypertext Transfer Protocol – HTTP/1.1, RfC 2068, 1997.* http://www.cis.ohio-state.edu/rfc/rfc2068.txt (02.2001)

[FloGryMac99] C. Floyd, G. Gryczan, J. Mack: *Einführung in die Softwaretechnik.* Scriptum zur gleichnamigen Lehrveranstaltung, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, Universität Hamburg, 1999.

[GHJV96] E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Entwurfsmuster – Elemente wiederverwendbarer objektorientierter Software.* Addison-Wesley, 1996.

[GirLeeSch96] A. Girgensohn, A. Lee, K. Schlueter: *Experiences in Developing Applications Using the World Wide Web "Shell"*. In: Hypertext'96, Seventh ACM Conference on Hypertext, pp. 246-255, 1996.

[GolPre99] C. F. Goldfarb, P. Prescod: *XML Handbuch.* Prentice Hall, 1999.

[GreSchHae01] M. Greunz, B. Schopp, J. Haes: *Integrating e-Government Infrastructures through Secure XML Document Containers.* In: Proceedings of the 34th Hawaii International Conference on System Sciences 2001, HICSS 34, IEEE Computer Society, 2001.

[Gri98] F. Griffel: *Componentware – Konzepte und Techniken eines Softwareparadigmas.* dpunkt-Verlag, 1998.

[Gry96] G. Gryczan: *Prozessmuster zur Unterstützung kooperativer Tätigkeit.* Deutscher Universitätsverlag GmbH, 1996.

[GRVR94] M. Gaitanides, R. Scholz, A. Vrohlings, M. Raster (Hrsg.): *Prozessmanagment: Konzepte, Umsetzungen und Erfahrungen des Reengineering.* München Wien, 1994.

[Gut95] B.A. Gutek: *The Dynamics of Service.* Jossey Bass, 1995.

[HeHe99] T. Hess, V. Herwig: *Portale im Internet*. Wirtschaftsinformatik, Heft 6, 1999.

[IBM98] International Business Machines Corporation: *IBM MQSeries Workflow: Concepts and Architecture.* Version 3.1, 1998.

[IveLea84] B. Ives, G.P. Learmonth: *The Information System as a Competitive Weapon*. In: Communications of the ACM, 27 (1984) 12, pp. 1193-1201

[JabBöhSch97] S. Jablonski, M. Böhm, W. Schulze (Hrsg.): *Workflow Management: Entwicklung von Anwendungen und Systemen; Facetten einer neuen Technologie.* Dpunkt-Verlag, 1997.

[KasKliWet01] R. Klaschek, R. Klischewski, I. Wetzel: *A Virtual Debate on Serviceflows.* In: EMISA Forum (GI-FG 2.5.2), Heft 1, S. 16 –23, 2001.

[Käs00] K. Kästner: *E-Government – Wege zur elektronischen Verwaltung der Zukunft.* 2000. http://www.z-punkt.de/download/e-gov.pdf (01.2001)

[Ker93] H. Kerner: *Rechnernetze nach OSI.* Addison-Wesley, 2. Auflage, 1993.

[KHKS99] H. Kubicek, M. Hagen, S. Klein, G. Schwellenbach: *Kundenorientierung durch Integration elektronischer Dienstleistungen für Bürger und Wirtschaft aus einer Hand. Die Bewerbung der Freien Hansestadt Bremen beim Multimedia Städtewettbewerb MEDIA@Komm,* 1999. http://www.bos-bremen.de/bremer online service/bremer konzept/pdf/kurzkonzeptMK.dpf (09.2000)

[KilGryZül94] K. Kilberth, G. Gryczan, H. Züllighoven: *Objektorientierte Anwendungsentwicklung – Konzepte, Strategien, Erfahrungen.* Vieweg, 2. verbesserte Auflage, 1994

[Kli00] R. Klischewski: *Abstrakte Bedürfnisse und konkrete Beziehungen oder: Wie man Services (nicht) modelliert.* In: J. Ebert, U. Frank (Hrsg.): Modelle und Modellierungssprachen in Informatik und Wirtschaftsinformatik. Proceedings Modellierung 2000, Koblenz: Fölbach, S. 19-22, 2000.

[KliWet00] R. Klischewski, I. Wetzel: *Serviceflow Management.* Informatik Spektrum, Bd. 23, Heft 1, S. 38-46, 2000.

[KliWetBah01] R. Klischewski, I. Wetzel, A. Baharami: *Modeling Serviceflow.* In: *Proceedings of the 1st International Conference on Information Systems Technology and Applications*, GI Lecture Notes in Informatics, Köllen Verlag, Bonn 2001.

[KraWetRat96] A. Krabbel, I. Wetzel, S. Ratuski: *Objektorientierte Analysetechniken für übergreifende Aufgaben.* In: *Beiträge der GI-Fachtagung Softwaretechnik 1996.* Koblenz, S. 65 – 72, 12.-13. September 1996.

[Kru97] J. Krumbiegel: *Integrale Gestaltung von Geschäftsprozessen und Anwendungssystemen in Dienstleistungsbetrieben.* Deutscher Universitäts-Verag, Wiesbaden, 1997.

[LipWolZül01] M. Lippert, H. Wolf, H. Züllighoven: *Domain Services for Multichannel Application Software.* In: Proceedings of the Hawaii Interantional Conference On System Sciences 2001, HICSS 34, IEEE Computer Society, 2001.

[Mal99] T. W. Malone et al.: *Tools for inventing organizations: Towards a handbook of organizational processes.* Management Science 45(3) S. 425-443, März 1999.

[McH00] S. McHale: *Partner Portals: Networking at the Speed of Business.* IDC, 2000. http://www.idc.com (07.2000)

[McL00] B. McLaughlin: *Java and XML.* O'Reilly & Associates, 2000.

[MeiSch01] J. Meier, T. Scharf: *Regeln für die Anwendungsarchitektur verteilter Systeme.* In: ObjektSpektrum, 2, 2001.

[Mer96] P. Mertens et al. (Hrsg.): *Lexikon der Wirtschaftsinformatik.* Springer-Verlag, dritte Auflage, 1996.

[Mer99] M. Merz: *Electronic Commerce – Marktmodelle, Anwendungen und Technologien.* Dpunkt.verlag, 1999.

[Mer99b] M. Merz: *Elektronische Dienstemärkte: Modelle und Mechanismen des Electronic Commerce.* Springer Verlag, Heidelberg, New York, 1999.

[Mei01] J. Meir: *Prozessmanagement als Grundlage für integriertes eGovernment.* In: Bulletin des Kompetenzzentrums eGovernment der Berner Fachhochschule, „eGov Präsenz", Ausgabe 01, 2001.

[Mey97] B. Meyer: *Object-Oriented Software Construction.* Second Edition, Prentice Hall, 1997.

[MidSin99] S. Middendorf, R. Singer: *Java Programmierhandbuch und Referenz für die Java-2-Plattform.* 2. Auflage, dpunkt-Verlag, 1999.

[Mit96] J. Mittelstraß (Hrsg.): *Enzyklopädie Philosophie und Wissenschaftstheorie 4.* J.B. Metzler-Verlag, Sp – Z, 1996.

[MorDavBoa01] R. Mordani, J.D. Davidson, S. Boag: *Java API for XML Processing.* Version 1.1, Final Release. http://www.javasoft.com/xml/ download.html#prodspec (03.2001)

[Oes01] B. Oestereich: *Die UML-Kurzreferenz für die Praxis.* Oldenbourg-Verlag, 2001.

[OrfHar97] R. Orfali, D. Harkey: *Client/Server Programming with Java and CORBA.* Wiley Computer Publishing, John Wiley & Sons Inc., 1997.

[Ost93] J. K. Osterhout: *Tcl and the Tk Toolkit.* Addison-Wesley, 1993.

[Öst99] H. Österle: *Banking im Kundenprozess – Ein Versuch zum Andersdenken.* http://www.iwi.unisg.ch/iwiwebdaten/Publications/euroforum_banking_anders_07_hoe31MAY99.pdf (12.2000)

[Öst00] H. Österle: *Geschäftsmodell des Informationszeitalters.* 2000. http://www.iwi.unisg.ch/Publications/publication_detail.asp?id=602 (07.2000)

[ÖstFleAlt00] H. Österle, E. Fleisch, R. Alt: *Business Networking – Shaping Enterprise Relationships on the Internet.* Springer Verlag, 2000.

[OttSch00] M. Otto, N. Schuler: *Fachliche Services: Geschäftslogik als Dienstleistung für verschiedene Benutzungsschnittstellen-Typen.* Diplomarbeit am Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 2000.

[Pae00] B. Paech: *Aufgabenorientierte Softwareentwicklung – Integrierte Gestaltung von Unternehmen, Arbeit und Software.* Springer-Verlag, 2000.

[Par72] D. L. Parnas: *On the Criteria to be Used in Decomposing Systems into Modules.* In: Communications of the ACM, 15, S. 1053 – 1058, 1972.

[RagHorJac99] D. Raggett, A. L. Hors, I. Jacobs (Editors): *HTML 4.01 Specification W3C Recommendation.* 1999. http://www.w3.org/TR/html4/html40.txt (02.2001)

[RecPom97] P. Rechenberg, G. Pomberger: *Informatik-Handbuch.* (Eds.), Carl Hanser Verlag, 1997.

[PriKol96] W. Prinz, S. Kolvenbach: *Support for Workflows in a Ministerial Environment.* In: S. Ackermann (Hrsg.): Proceedings of the Conference on Computer Supported Cooperative Work. ACM-Press, S. 109 – 208, 1996.

[Sch01] N. Shala: *XML basiertes Serviceflow-Management mit Enterprise Java Beans.* (Arbeitstitel) Diplomarbeit am Fachbereich Informatik der Universität Hamburg, Arbeitsbereich Softwaretechnik, voraussichtlich Ende 2001.

[SchHäu01] P. Schubert, U. Häusler: *E-Government meets E-Business: A Portal Site for Startup Companies in Switzerland*. In: Proceedings of the Hawaii Interantional Conference On System Sciences 2001, HICSS 34, IEEE Computer Society, 2001.

[ShaGar96] M. Shaw, D. Garlan: *Software Architecture – Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[Rol98] A. Rolf: *Grundlagen der Organisations- und Wirtschaftsinformatik.* Springer-Verlag, 1998.

[Sau99] G. Saueressig: *Internetbasierte Self-Service-Systeme für kundenorientierte Dienstleistungsprozesse in öffentlichen Verwaltungen.* dissertation.de Verlag im Internet, 1999. (http://www.dissertation.de) (07.2000)

[Sch98] F. Schiedner: *Verwaltung gestalten – Orientierungen für die Praxis – Organisationsstrukturen und Arbeitsprozesse.* Raabe Fachverlag für Öffentliche Verwaltung, 1998.

[Sch00] K. Schedler: *eGovernment und neue Servicequalität in der Verwaltung?*. In: M. Gisler, D. Spahni: eGovernment – Eine Standortbestimmung, Verlag Paul Haupt Bern, 2000.

[Sch00b] U. Schreier, M. Gerschwiler, D. Girardin, J. Jungjohann, T. Wüst: *Adaptierbare Frontend-Architekturen*. In: Objekt Spektrum 4/2000, S. 18 - 26, 2000.

[SchMac00] G. Schulmeyer, G.R. MacKenzie: *Verification & Validation of modern Software-Intensive Systems.* Prentice Hall, 2000.

[ShaEar98] Y.-P. Shan, R. H. Earle: *Enterprise Computing with Objects – From Client/Server Environments to the Internet.* Addison-Wesley, 1998.

[Teu95] S. Teufel, C. Sauter, T. Mühlherr, K. Bauknecht: *Computerunterstützung für die Gruppenarbeit.* Addison-Wesley, 1995.

[Tol99] R. Tolksdorf: *XML und darauf basierende Standards: Die neuen Auszeichnungssprachen des Web.* In: Informatik Spektrum, 22, Dezember 1999.

[Tur99] V. Turau : *Techniken zur Realisierung Web-basierter Anwendungen.* In: Informatik Spektrum 22, 3-12, Springer-Verlag, 1999.

[Vig99a] Vignette Corporation: *Vignette StoryServer: Overview*. Version 5.0, 1999.

[Vig99b] Vignette Corporation: *Vignette StoryServer: Template Cookbook.* Version 5.0, 1999.

[Vig99c] Vignette Corporation: *Vignette StoryServer: Business Center Guide.* Version 5.0, 1999.

[Wol00] H. Wolf: *Java Server Pages und Fachliche Services.* Vortrag im Rahmen des Projektseminars „Service als Leitbild - Softwareunterstützung für Dienstleister“ am Fachbereich Informatik der Universität Hamburg, Arbeitsbereich Softwaretechnik, WS2000/2001, 8.11.2000.

[Woo98] L. Wood: *Document Object Model (DOM) Level 1 Specification.* W3C Recommendation, 1. Oktober 1998. http://www.w3c.org/DOM/dom.pdf (03.2001)

[Zül98] H. Züllighoven: *Das objektorientierte Konstruktionshandbuch nach dem Werkzeug- & Material-Ansatz.* Dpunkt-Verlag, 1998.

Anhang

Im folgenden sind die Schnittstellen der wichtigsten vom funktionalen Prototypen verwendeten Komponenten aufgeführt. Das Zusammenspiel der Komponenten und ihre Aufgaben sind in Kapitel 3.6 der vorliegenden Arbeit beschrieben. Aus diesem Grund wird hier auf eine weitere Kommentierung im Hinblick auf diese Aspekte verzichtet. Des weiteren sei darauf hingewiesen, dass ein durchgängiger Ausnahmebehandlungsmechanismus aus Zeitgründen nicht vollständig implementiert werden konnte.

Archive

```
/** Archiviert das angegebene Material */
public void archivate (Manageable material);

/**
* Setzt den Provider, um die Scripts abzuspeichern
*/
public void setProvider (MaterialProvider provider);
```

FloatReceiver

```
/**
* Ermöglicht die Übergabe eines neuen Serviceflats an den Receiver. Diese Methode
* kann z.B. von einem Transportautomaten aufgerufen werden.
*/
public void receiveFloat (Object theFloat);

/**
* Liefert das im Receiver vorhandene Float, dass über dem Paramter id
* identifiziert wird. Der FloatReceiver benachrichtigt seine Beobachter
* über neu eingetragene Servicefloats und übergibt dabei die id.
*/
public Manageable getFloat (String id) throws UnknownServicefloatException;

/**
* Liefert die eindeutige Id des Servicepoints. Diese Id kann von einem
* Transportmechanismus als Adresse des Servicepoints verwendet werden.
*/
public String getSPID ();
```

FloatSender

```
/**
*  Übernimmt das versandfertige Servicefloat und benachrichtigt daraufhin seine
*  Beobachter darüber, dass ein neues Servicefloat zur Versendung bereitsteht.
*  Hierbei wird eine eindeutige id mitgeliefert.
*/
public void send (Servicefloat theFloat);

/**
* Liefert die eindeutige Id des Servicepoints. Diese Id kann von einem
* Transportmechanismus als Adresse des Servicepoints verwendet werden.
*/
public String getSPID ();

/**
* Liefert das im Receiver vorhandene Float, dass über den Parameter id
* identifiziert wird.
*/
public Manageable getFloat (String id) throws UnknownServicefloatException;
```

Manageable

```
/**
* Liefert den Identifikator, der den Besitzer des Materials eindeutig
* indentifiziert. Diese ID wird zur internen Verwaltung verwendet, so
* dass alle Materialien mit der gleichen OwnerID als zusammengehörig
* erkannt werden können.
*/
public String getIDOfOwner ();

/** Setzt den Identifikator des Besitzers des Materials; z.B. Kundennummer */
public void setIDOfOwner (String id);

/** Der eindeutige Identifikator, der auch im Verwaltbaren steht */
public void setKey (String id);

/** Liefert den eindeutigen Identifikator des Verwaltbaren */
public String getKey();

/**
* Liefert eine tiefe Kopie des Verwaltbaren.
*/
public Object getCopy();

/**
* Liefert den Haupttyp des Verwaltbaren. Wird benötigt, um beim speichern den
```

```
* richtigen Provider zu identifizieren
*/
public String getType ();
```

ManageableServicefloat

```
/**
* Übernimmt die Postconditions und trägt sie in dieses Float ein. Wenn eine
* Postcondition schon im Float im Sinne einer Vorlage vorhanden ist, so wird
* diese überschrieben. Existiert die Postcondition noch nicht (wurde also am
* Servicepoint neu erzeugt), so wird sie an die Liste der Postconditions angehängt.
* Annahme: Es handelt sich bei dem Übergabeobjekt um eine Liste von Elementen
* vom Typ "Node"
*/
public void appendPostconditions (List thePostconditions);

/**
* Hängt die Liste der übergebenen Dokumente an die vorhandene Dokumenten-Liste an.
* Annahme: Es wird der Parent-Knoten der Dokumente im Servicepointscript übergeben.
*/
public void appendDocuments(Object docList);

/** Setzt den Kunden-Namen */
public void setCustomerName(String vorname, String nachname);

/** Liefert den Nachnamen des Kunden */
public String getLastNameOfCustomer () throws NoCustomerException;

/** Liefert den Vornamen des Kunden */
public String getFirstNameOfCustomer () throws NoCustomerException;

/** Liefert die Kunden-Nummer */
public String getIDOfCustomer () throws NoCustomerException;

/** Setzt die Kunden-Nummer */
public void setIDOfCustomer (String id) throws NoCustomerException;

/**
* Ermöglicht das testen, ob schon eine Postcondition mit der angegebenen ID
* im Float geführt wird.
*/
public boolean hasPostcondition (String id);

/**
*  Macht das Servicefloat versandfertig, indem diese Methode den nächsten
*  scheduled Servicepoint bestimmt und diesen zum current_servicepoint
*  macht. Zusätzlich wird der aktuelle servicepoint in die Historie geschrieben.
*  Wenn der letzte Servicepoint erreicht ist (d.h. es ist keiner mehr in der
*  Scheduled-Liste
```

```
*  eingetragen), dann wird der current_servicepoint gelöscht. In der Historie
*  wird das aktuelle Datum im Format: JJJJ-MM-DDTHH-MM-SS
*  eingetragen um zu kennzeichnen, wann weitergeschaltet wurde.
*/
public void prepareForDispatch() throws NotPreparableException;

/**
*  Liefert den Untertypen des Verwaltbaren;
*  hierbei handelt es sich um den Untertyp von Servicefloat
*  (z.B. Briefwahlantrag_hh.de)
*/
public String getSubtype ();
```

ManageableServicepointscript

```
/**
* Ermöglicht das setzen der Id, die die eindeutige Zuordnung von Script und Float
* ermöglicht
*/
public void setFloatId (String id);

/** Liefert die id des zugehörigen Floats   */
public String getFloatId ();

/** Liefert die Dokumentenliste für das Servicefloat */
public Object getDocumentList();

/** Liefert eine Liste der erreichten Servicepointscript Nachbedingungen */
public List getListOfAchievedSpPostconditions();

/** Setzt die Kunden-Nummer */
public void setIDOfCustomer (String id) throws NoCustomerException;

/** Liefert die Kunden-Nummer */
public String getIDOfCustomer () throws NoCustomerException;

/** Setzt den Kunden Namen */
public void setCustomerName(String vorname, String nachname);

/**
*  Liefert den Untertypen des Verwaltbaren;
*  hierbei handelt es sich um den Untertyp von Servicefloat oder Servicepointscript
*  (z.B. Briefwahlantrag_hh.de)
*/
public String getSubtype ();
```

MaterialManager

```
/**
* Ermöglicht das hinzufügen eines Providers, der zum einlesen von Materialien
* aus dem Persistenzmedium verwendet wird.
*/
public void addProvider (MaterialProvider provider);

/**
* Lädt alle vorhandenen Materialien aus dem Zwischenspeicher
*/
public void loadMaterials ();

/**
* Liefert alle Materialien, die unter der angegebenen ID gehalten werden.
* Falls keine Materialien unter der angegebenen id gehalten werden,
* wird null zurückgegeben.
*/
public MaterialCollection getMaterials (String id);

/**
* Löscht die durch die id identifizierte Material-Sammlung.
*/
public void deleteMaterials (String id);

/**
* Löscht das durch idMaterial identifizierte Material der Sammlung, die durch
* den Identifikator id bestimmt wird.
*/
public void deleteMaterial (String id, String idMaterial);

/**
* Speichert alle Materialsammlungen im aktuellen Bearbeitungszustand
* im Zwischenspeicher ab, die durch die angegebenen Ids identifiziert werden.
*/
public void saveMaterials (List ids);

/**
* Fügt eine neue MaterialSammlung dem MaterialManager unter der angegebenen id
* hinzu.
*/
public void addMaterial(String id, MaterialCollection mcol);

/**
* Liefert die Liste aller ids, unter denen die Materialien gehalten werden.
*/
public List getIds ();
```

MaterialProvider

```
/**
* Liefert den Typ des vom Versorger angebotenen Materials
* (z.B. Servicepointscript).
*/
public String typeOfMaterials ();

/**
* Liefert alle am Servicepoint vorhandenen Materialien des jeweils angebotenen Typs
* aus dem jeweiligen Persistenzmedium.
* Hierbei werden die später zu verwendenden Objektdarstellungen erzeugt.
* Konkrete MaterialProvider erhält man über die ProviderFactory.
*/
public MaterialCollection getAllMaterials ();

/**
* Speichert die übergebenen Materialien auf dem Persistenzmedium ab, auf dem
* der Provider arbeitet.
*/
public void saveMaterials (MaterialCollection theCollection);

/** Löscht die über die angegebenen ids identifizierten Materialien */
public void deleteMaterials (List ids);
```

MaterialStore

```
/** Initialisiert das Magazin neu */
public void initialise ();

/**
* Fügt eine Material-Sammlung hinzu. Falls eine unter dem Schlüssel schon
* existiert, wird sie überschrieben.
*/
public void addMaterialCollection (Object key, MaterialCollection materialCol);

/**
* Hängt Materialien der übergebenen Material-Sammlung an eine vorhandene Sammlung
* an.
* Vorraussetzung: es gibt schon eine Materialsammlung mit der entsprechenden id.
*/
public void appendMaterialCollection (Object id, MaterialCollection materialCol);

/** Löscht die Materialien für die angegebenen ids, falls vorhanden*/
public void deleteMaterials (List ids);

/** Löscht das Material für die angegebenen id, falls vorhanden */
public void deleteMaterials (String id, String mId);
```

```
/** Liefert die Liste aller verwendeten Schlüssel */
public List getKeys();

/** Liefert die gewünschte Material-Sammlung, falls vorhanden, sonst null */
public MaterialCollection getMaterials (Object id);

/** Prüft, ob unter der id Materialien geführt werden */
public boolean hasMaterials (Object id);

/** Prüft, ob unter den ids Materialien geführt werden */
public boolean hasMaterials (List ids);
```

ServicepointManager

```
/**
* Erzeugt ein neues Servicefloat und ein dazugehöriges Servicepointscript
* und liefert eine ID, unter der das Float bzw. Script zugegriffen
* werden kann. Die ID wird als Kunden-ID interpretiert.
*/
public String createNewServicefloat (String type) throws UncreatableException;

/**
* Erzeugt ein neues Servicefloat und ein dazugehöriges Servicepointscript für
* die angegebene Kunden-ID.
*/
public Servicefloat createNewServicefloat (String customerId, String type)
                    throws UncreatableException;

/**
* Liefert das Servicepointscript des angegebenen Kunden. Vorraussetzung ist,
* dass der Kunde dem ServicepointManager bekannt ist und das Float unter der
* angegebenen id existiert.
* Ein Script kann nur dann existieren, wenn ein Float vorhanden ist. Deswegen
* wird hier die FloatId angegeben, um an das zugehörige Script zu gelangen.
*/
public Servicepointscript getServicepointscript(String customerId, String floatId)
       throws UnknownCustomerException, UnknownServicepointscriptException;

/**
* Liefert das Servicefloat des angegebenen Kunden. Vorraussetzung ist,
* dass der Kunde dem ServicepointManager bekannt ist und das Float unter der
* angegebenen id existiert.
*/
public Servicefloat getServicefloat(String customerId, String floatId)
       throws UnknownCustomerException, UnknownServicefloatException;
```

```
/**
* Dient der Überprüfung, ob überhaupt ein Servicefloat für den Kunden am
* Servicepoint vorhanden ist.
*/
public boolean hasServicefloat(String customerId);

/**
* Dient der Überprüfung, ob ein Servicefloat für den Kunden unter der
* angegebenen ID vorhanden ist.
*/
public boolean hasServicefloat(String customerId, String floatId);

/** Prüft, ob ein Servicefloat vom angegebenen Typ erzeugt werden kann. */
public boolean isServicefloatOfTypeCreatable (String type);

/** Liefert eine Liste mit Namen der verfügbaren Servicefloat-Typen. */
public List getListOfAvailableServicefloatTypes ();

/** Liefert Liste mit Namen der verfügbaren Servicepointscript-Typen. */
public List getListOfAvailableServicepointscriptTypes ();

/**
* Initiiert die Vorbereitung für die Versendung des Servicefloats, das durch die
* angegebene floatId gekennzeichnet wird. Hierbei werden alle erreichten
* Nachbedingungen und erzeugten Dokumente aus dem Servicepointscript in das
* Servicefloat und der aktuelle Servicepoint in die Historie geschrieben.
* Außerdem wird der nächste Scheduled-Servicepoint zum aktuellen gemacht.
* Anschließend kann die Methode sendServicefloat zum Versand aufgerufen werden.
*/
public void prepareForDispatch (String customerId, String floatId)
       throws UnknownCustomerException, NotPreparableException,
              UnknownServicefloatException;
/**
* Liefert eine Liste der IDs der Servicefloats, die sich an einem SP für einen
* Kunden befinden.
*/
public List getServicefloatIds (String customerId) throws UnknownCustomerException;

/** Liefert eine Liste der KundenIDs, die ein Servicefloat am Servicepoint haben.*/
public List getCustomerIds ();

/** Liefert Liste der verfügbaren nächsten Servicepoints.*/
public List getListOfAvailableNextServicepoints ();

/** Liefert die Liste der Namen der möglichen nächsten Servicepoints. */
public List getNamesOfPossibleNextServicepoints ();
```

```
/**
* Speichert alle geladenen Servicefloats und -scripts im momentanen Zustand
* zwischen, so dass der Zustand persistent gehalten werden kann.
*/
public void saveState();

/**
* Initiiert die Versendung des Servicefloats (legt es in den Ausgangskorb) und
* archiviert das zugehörige Servicepointscript. Zusätzlich werden die
* persistenten Kopien von Float und Script aus dem Zwischenspeicher gelöscht.
*/
public void sendServicefloat(String customerId, String floatId)
        throws UnknownCustomerException, UnknownServicefloatException;
```

Servicefloat

```
// Auf das Servicefloat bezogene Methoden:

/** Liefert den Namen des Serviceflow-Exemplars. */
public String getName();

/** Liefert den eindeutigen Identifikator des Servicefloats */
public String getKey();

//-------------------------------------------------

// Auf Servicepoints bezogene Methoden
/**
*  Liefert eine Liste von Strings, welche die Namen der bisher
*  durchlaufenen Servicepoints darstellen.
*  Falls noch keine Servicepoints durchlaufen wurden, ist die Liste leer.
*/
public List getListOfNamesOfPassedServicepoints();

/**
* Liefert das Datum, an dem ein bereits abgeschlossener Servicepoint geschlossen
* wurde. Falls nicht gesetzt, wird ein leerer String geliefert.
*/
public String getDateOfPassedServicepoint(String nameOfPassedServicepoint)
             throws UnknownServicepointException;

/**
* Liefert den Namen des aktuellen Servicepoints. Falls das Float am Ende ist, gibt
* es keinen current servicepoint.
*/
public String getNameOfCurrentServicepoint() throws NoCurrentServicepointException;
```

```
/**
* Liefert das Datum, an dem die Leistung des aktuellen Servicepoints erbracht
* werden soll.
* Falls nicht gesetzt, wird ein leerer String geliefert.
* Falls das float am Ende ist, gibt es keinen current servicepoint.
* Format: Format: JJJJ-MM-DDTHH-MM-SS
*/
public String getDateOfCurrentServicepoint() throws NoCurrentServicepointException;

/**
* Liefert das Datum, bis zu dem der aktuelle Servicepoint abgeschlossen sein muss.
* Format: JJJJ-MM-DDTHH-MM-SS
*/
public String getValidUntilDateOfCurrentServicepoint()
       throws NoCurrentServicepointException;

/**
* Liefert eine Liste der Namen aller scheduled Servicepoints, die im Service-
* float eingetragen sind. Wenn keine eingetragen sind, wird eine leere
* Liste zurückgegeben.
*/
public List getListOfNamesOfScheduledServicepoints();

/**
* Liefert das Datum, an dem das Serivcefloat am durch den Namen identifizierten
* Servicepoint sein muss.
* Entspricht einem Termin für die Leistungserbringung. Format: JJJJ-MM-DDTHH-MM-SS
*/
public String getDateOfScheduledServicepoint(String nameOfScheduledServicepoint)
       throws UnknownServicepointException;

/**
* Liefert das Datum bis zu dem ein scheduled Servicepoint gültig ist. Der
* Servicepoint wird eindeutig durch Angabe seines Namens identifiziert.
* Format: JJJJ-MM-DDTHH-MM-SS
*/
public String getValidUntilDateOfScheduledServicepoint(String
        nameOfScheduledServicepoint) throws UnknownServicepointException;

/**
* Liefert den Typ eines Servicepoints, der eindeutig durch Angabe seines Namens
* identifiziert wurde.
*/
public String getTypeOfServicepoint(String nameOfServicepoint)
       throws UnknownServicepointException;
```

```
/**
* Liefert die Adresse eines Servicepoints, der eindeutig durch Angabe seines Namens
* identifiziert wurde.
*/
public String getAddressOfServicepoint(String nameOfServicepoint)
       throws UnknownServicepointException;

/**
* Liefert den Namen des Providers, der am angegebenen Servicepoint den Service
* erbringt.
*/
public String getNameOfServicepointProvider(String nameOfServicepoint)
       throws UnknownServicepointException;

/**
* Liefert den Namen des Providers, der am aktuellen Servicepoint den Service
* erbringt.
*/
public String getNameOfCurrentServicepointProvider ()
       throws NoCurrentServicepointException;

/**
* Fügt einen neuen Servicepoint durch Angabe des Namens in die Scheduled-Liste ein.
* Die Position wird durch Angabe des Namens des Vorgänger-Servicepoints bestimmt.
* Der Name des neuen Servicepoints stammt aus einer Liste (XML-File) möglicher
* Servicepoints, wodurch die Eindeutigkeit sichergestellt ist. Falls es keinen
* Vorgänger gibt, ist ein leerer String zu übergeben.
*/
public void addScheduledServicepoint(String nameOfPredecessorServicepoint,
       String nameOfNewServicepoint)
       throws UnknownServicepointException, NotAddableException;

/**
* Löscht den durch den Namen eindeutig identifizierten Servicepoint aus der
* scheduled-Liste.
*/
public void deleteScheduledServicepoint(String nameOfScheduledServicepoint)
       throws UnknownServicepointException, NoScheduledServicepointsException;

/**
* Erlaubt die Veränderung des Datums, an dem die Leistung des scheduled
* Servicepoints erbracht wird. Format: JJJJ-MM-DDTHH-MM-SS
*/
public void modifyDateOfScheduledServicepoint(String nameOfScheduledServicepoint,
       String date) throws UnknownServicepointException, WrongDateFormatException;

/**
* Verändert das Valid-Until-Datum des identifizierten Servicepoints auf das
* angegebene Datum. Format: JJJJ-MM-DDTHH-MM-SS
*/
```

```
public void modifyValidUntilDateOfScheduledServicepoint(String
       nameOfScheduledServicepoint, String date)
              throws UnknownServicepointException, WrongDateFormatException;

/** Verändert den Typ eines Scheduled-Servicepoints auf den angegebenen. */
public void modifyTypeOfScheduledServicepoint(String nameOfScheduledServicepoint,
       String type) throws UnknownServicepointException;

/** Erlaubt die Veränderung des Namens eines scheduled Servicepoints. */
public void modifyNameOfScheduledServicepoint(String nameOfScheduledServicepoint,
       String name) throws UnknownServicepointException;

/** Erlaubt das Setzen eines Serviceanbieters am Servicepoint*/
public String modifyNameOfServicepointProvider(String nameOfServicepoint)
       throws UnknownServicepointException;
//-------------------------------------------------------------

// Auf Kunden bezogene Methoden

/**
* Liefert die Kunden-Nummer, dem das jeweilige Servicefloat-Exemplar zugeordnet ist
*/
public String getIDOfCustomer () throws NoCustomerException;

/**
* Stellt den Namen des Kunden zur Verfügung, dem das Servicefloat-Exemplar
* zugeordnet ist.
*/
public String getNameOfCustomer() throws NoCustomerException;

/**
* Liefert den Nachnamen des Kunden
*/
public String getLastNameOfCustomer () throws NoCustomerException;

/**
* Liefert den Vornamen des Kunden
*/
public String getFirstNameOfCustomer () throws NoCustomerException;

/**
* Setzt die Kunden-Namen, dem das jeweilige Servicefloat zugeordnet ist
*/
public void setCustomerName(String vorname, String nachname);

/**
* Setzt die Kunden-Nummer, dem das jeweilige Servicefloat zugeordnet ist
*/
public void setIDOfCustomer (String id) throws NoCustomerException;
//-------------------------------------------------------------
```

```
// Auf Bedingungen bezogene Methoden

/**
* Liefert die Namen aller im Servicefloat gesammelten Nachbedingungen
* der bisher durchlaufenen Servicepoints. Falls keine Postconditions vorhanden
* sind, wird eine leere Liste geliefert.
*/
public List getListOfNamesOfAccumulatedPostconditions();

//-------------------------------------------------------------

// Auf Dokumente bezogene Methoden

/**
* Liefert die Namen der im Servicefloat mitgeführten Dokumente. Falls keine
* Dokumente vorhanden sind, wird eine leere Liste geliefert.
*/
public List getListOfNamesOfDocuments();

/**
* Liefert den Inhalt eines Dokuments, das im Servicepointscript vorhanden ist.
* nameOfDocument identifiziert das Dokument
* contentName    identifiziert das Tag, das den gesuchten Inhalt enthält
*/
public String getContentOfDocument(String nameOfDocument, String contentName)
       throws UnknownDocumentException;

/**
* Liefert den Inhalt eines Dokuments, das im Servicepointscript vorhanden ist.
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
* contentName    identifiziert das Tag, das den gesuchten Inhalt enthält
*/
public String getContentOfDocument(String nameOfDocument, String documentPartName,
       String contentName) throws UnknownDocumentException;


/**
* Modifiziert den Inhalts eines Tags in einem Dokument
* nameOfDocument identifiziert das Dokument
* contentName    identifiziert das Tag
* newContent     der neue Wert des Tags
*/
public void modifyContentOfDocument(String nameOfDocument, String newContent,
       String contentName) throws UnknownDocumentException;

/**
* Modifiziert den Inhalts eines Tags in einem Teildokument
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
```

```
* contentName    identifiziert das Tag
* newContent     der neue Wert des Tags
*/
public void modifyContentOfDocument(String nameOfDocument, String documentPartName,
       String newContent, String contentName) throws UnknownDocumentException;
```

Servicepointscript

```
// Auf das Servicepointscript bezogene Methoden:

/** Liefert den Namen des Servicepointscripts */
public String getName();

/** Liefert den eindeutigen Identifikator des Servicepointscripts */
public String getKey();


/** Liefert Werte der Attribute des Servicepointscripts */
public List getAttributesOfServicepoint();

//-------------------------------------------------------

// Auf die Dienstleister bezogene Methoden

/** Liefert den Namen des Diensterbringers */
public String getNameOfProvider();

/**
* Liefert die Liste der Namen der Serviceerbringer an einem Servicepoint,
* die zur Bearbeitung der Aktivitäten vorgesehen sind
*/
public List getListOfNamesOfEmployee();

//-------------------------------------------------------

// Auf den Kunden bezogene Methoden

/** Setzt den Kunden-Namen */
public void setCustomerName(String vorname, String nachname);

/** Liefert die Kundennummer */
public String getIDOfCustomer () throws NoCustomerException;

/** Liefert den Nachnamen des Kunden, dem das Servicepointscript zugeordnet ist */
public String getLastNameOfCustomer () throws NoCustomerException;

/** Liefert den Vornamen des Kunden, dem das Servicepointscript zugeordnet ist */
public String getFirstNameOfCustomer () throws NoCustomerException;
```

```
/** Setzt die Kunden-Nummer */
public void setIDOfCustomer (String id) throws NoCustomerException;

//-------------------------------------------------------

// Auf Aktivitäten bezogene Methoden

/** Liefert den Namen der aktuellen Aktivität */
String getNameOfCurrentActivity();

/** Liefert Attribute der durch den Parameter identifizierten Aktivität */
public List getAttributesOfActivity(String activityName);

/** Liefert den Typen der durch den Parameter identifizierten Aktivität */
public String getTypeOfActivity(String nameOfActivity);

/** Liefert die Beschreibung der durch den Parameter identifizierten Aktivität */
public String getDescriptionOfActivity(String nameOfActivity);

/**
* Liefert den Namen des Diensterbringers der zur Durchführung der
* durch den Parameter identifizierten Aktivität vorgesehen ist
*/
public String getNameOfEmployeeOfActivity(String nameOfActivity);

/** Liefert die Namen der bereits durchgelaufenen Aktivitäten (Historie) */
public List getListOfNamesOfPassedActivities();

/** Liefert die Namen der noch zu durchlaufenen Aktivitäten (Schedule) */
public List getListOfNamesOfScheduledActivities();

/**
* Modifiziert die Beschreibung der als aktuelle Aktivität identifizierten Aktivität
*/
public void modifyDescriptionOfCurrentActivity(String nameOfCurrentActivity,
       String description);

/**
* Fügt eine neue Aktivität in die Liste der noch zu durchlaufenen Aktivitäten nach
* der durch nameOfPredecessorActivity identifizierten Aktivität ein
*/
public void addScheduledActivity(String nameOfPredecessorActivity,
       String nameOfNewActivity);

/**
* Löscht die durch nameOfScheduledActivity idnetifizierte Aktivität aus
* Liste der noch zu durchlaufenen Aktivitäten
*/
public void deleteScheduledActivity(String nameOfScheduledActivity);
```

```
/**
* Verändert den Typ der durch nameOfScheduledActivity identifizierten Aktivität
* aus der Liste der noch zu durchlaufenen Aktivitäten
*/
public void modifyTypeOfScheduledActivity(String nameOfScheduledActivity,
      String type);

/**
* Verändert den Namen der durch nameOfScheduledActivity identifizierten Aktivität
* aus der Liste der noch zu durchlaufenen Aktivitäten
*/
public void modifyNameOfScheduledActivity(String nameOfScheduledActivity,
      String name);

/**
* Verändert die Beschreibung der durch nameOfScheduledActivity identifizierten
* Aktivität aus der Liste der noch zu durchlaufenen Aktivitäten
*/
public void modifyDescriptionOfScheduledActivity(String nameOfScheduledActivity,
      String description);

/**
* Kennzeichnet die aktuelle Aktivität als erledigt, indem sie der Lister der
* bereits durchlaufenen Aktivitäten hinzugefügt wird
*/
public void endActivity();

/**
* Entfernt die letzte bearbeitete Aktität aus der Liste der bereits durchlaufenen
* Aktivitäten und macht sie zur aktuell auszuführenden Aktivität. Die bis dato
* aktuelle Aktivität wird wieder der Liste der noch zu erledigenden Aktivitäten
* hinzugefügt.
*/
public void reopenLastActivity();

//-------------------------------------------------------

// Auf Dokumente bezogene Methoden

/**
* Liefert den Inhalt eines Dokuments, das im Servicepointscript vorhanden ist.
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
* contentName    identifiziert das Tag, das den gesuchten Inhalt enthält
*/
public String getContentOfDocument(String nameOfDocument, String docPartName,
      String contentName);

/**
* Liefert den Inhalt eines Dokuments, das im Servicepointscript vorhanden ist.
```

```
* nameOfDocument identifiziert das Dokument
* contentName    identifiziert das Tag, das den gesuchten Inhalt enthält
*/
public String getContentOfDocument(String nameOfDocument, String contentName);

/**
* Liefert eine Liste der Dokumentennamen aller im Servicepointscript
* vorhandenen Dokumente
*/
public List getListOfNamesOfDocuments();

/**
* Liefert eine Liste in der Form Attributname, Attributwert.
* Bei den Attributen handelt es sich um Attribute eines Dokuments, das
* im Servicepointscript vorhanden ist.
* nameOfDocument identifiziert das Dokument
* documentType   identifiziert den Dokumenttyp (z.B. Briefwahlantrag)
*/
public List getListOfAttributesOfDocument(String nameOfDocument,
      String documentType);

/**
* Liefert eine Liste in der Form Attributname, Attributwert.
* Bei den Attributen handelt es sich um Attribute eines Tags im Dokument
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
* documentType   identifiziert das Tag
*/
public List getListOfAttributesOfDocumentContent(String nameOfDocument,
      String docPartName, String contentName);

/**
* Liefert eine Liste in der Form Attributname, Attributwert.
* Bei den Attributen handelt es sich um Attribute eines Teildokuments
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
*/
public List getListOfAttributesOfDocumentContent(String nameOfDocument,
      String contentName);

/** Verändert den Namen des durch nameOfDocument identifizierten Dokuments */
public void modifyNameOfDocument(String nameOfDocument, String newNameOfDocument);


/**
* Modifiziert den Inhalts eines Tags in einem Teildokument
* nameOfDocument identifiziert das Dokument
* docPartName    identifiziert ein Teildokument
* contentName    identifiziert das Tag
* newContent     der neue Wert des Tags
```

```
*/
public void modifyContentOfDocument(String nameOfDocument, String docPartName,
      String contentName, String newContent);
/**
* Modifiziert den Inhalts eines Tags in einem Dokument
* nameOfDocument identifiziert das Dokument
* contentName    identifiziert das Tag
* newContent     der neue Wert des Tags
*/
public void modifyContentOfDocument(String nameOfDocument, String contentName,
      String newContent);

//-----------------------------------------------

// Auf Bedingungen bezogene Methoden

/** Liefert eine Liste der Namen der im Servicepointscript eingetragenen
Vorbedingungen */
public List getListOfSpPreconditions();

/** Liefert eine Liste der der Namen der im Servicepointscript eingetragenen
Nachbedingungen */
public List getListOfSpPostconditions();

/** Liefert den Status der durch den Parameter identifizierten Vorbedingung */
public String getStateOfSpPrecondition(String preconditionName);

/** Liefert den Status der durch den Parameter identifizierten Nachbedingung */
public String getStateOfSpPostcondition(String postconditionName);

/**
* Setzt den Status der durch prconditionName identifizierten Vorbedingug
* mögliche Werte sind: "expected", "available", "not_available"
*/
public void setStateOfSpPrecondition(String preconditionName, String state);

/**
* Setzt den Status der durch prconditionName identifizierten Vorbedingug
* mögliche Werte sind: "expected", "achieved", "not_achieved"
*/
public void setStateOfSpPostcondition(String postconditionName, String state);
```

ProviderFactory

```
/**
* Liefert einen Provider für den angegebenen Ort. Ein Ort kann z.B. das temporäre
* Verzeichnis oder das Verzeichnis für die Vorlagen sein.
* Mögliche Werte für location sind: Master, Running, In, Out
* Der Parameter type identifiziert den Typ des Materials, der vom Provider
```

```
* angeboten werden soll (z.B. Servicefloat).
* In der ProviderFactory steckt das Wissen darüber, welcher konkrete Provider für
* welchen Ort und welchen Typ zu erzeugen ist (z.B. ein Provider für XML-Files, die
* im temporären Verzeichnis des File-Systems für Servicepointscripts abgespeichert
* sind)
*/
public MaterialProvider createProvider (String location, String type);
```

Repertoire

```
/**
* Fügt einen Provider hinzu, der die Prozessrepräsentationen aus dem Persistenz-
* medium ausliest und konkrete Vorlagen für Prozessrepräsentationen erzeugt.
*/
public void addProvider (MaterialProvider provider);

/**
* Liefert eine Kopie des Masters, der durch die Parameter type und subtype
* identifiziert wird. type repräsentiert hierbei den Haupttyp, z.B. Serivcefloat
* oder Servicepointscript, subtype den jeweiligen Untertyp, z.B.
* "Briefwahlantrag 2001", also die konkrete Prozessvorlage.
*/
public Object getProcessrepresentation (String mainType, String subType);

/**
* Liefert Liste der Prozesstypen-Bezeichnungen, für die das Repertoire Vorlagen
* anbietet
*/
public List getListOfNamesOfTypes ();

/**
* Liefert Liste der Bezeichnungen der Untertypen, die für den Angegebenen Haupttyp
* angeboten werden
*/
public List getListOfNamesOfSubtypes (String mainType);

/** Prüft, ob eine Vorlage des gewünschten Typs vorhanden ist */
public boolean hasType (String mainType, String subType);

/**
* Initialisiert den Provider. Dies führt dazu, dass ein Repertoire alle am
* Servicepoint vorhandenen Prozessvorlagen mit Hilfe der im Vorwege gesetzten
* Provider aus dem Persistenzmedium einliest und Prozessexemplare erzeugt,
* die die Master repräsentieren.
*/
public void initalise ();
```

XMLTransformer

```
/** Konstruiert aus dem angegebenen File ein DOM */
public Document createDocumentFromXMLFile (File theFile);

/** Schreibt aus dem DOM ein File unter dem angegebenen File-Name */
public void writeDocumentToFile(String fileName, Document doc);
```