INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines SOA-basierten Konzepts zur Unterstützung des ITIL Incident-Management-Prozesses

Sabine Lehner

Aufgabensteller:	Prof. Dr. Heinz-Gerd Hegering
Betreuer:	Michael Brenner Dr. Markus Garschhammer

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Entwicklung eines SOA-basierten Konzepts zur Unterstützung des ITIL Incident-Management-Prozesses

Sabine Lehner

Aufgabensteller: Prof. Dr. Heinz-Gerd Hegering

Betreuer: Michael Brenner
Dr. Markus Garschhammer

Abgabetermin: 30. Juni 2006

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 30. Juni 2006

. . . .

(Unterschrift der Kandidatin)

Zusammenfassung

Mit zunehmender Größe und Komplexität der zu managenden Infrastrukturen wird für alle IT Service Provider auch die Effizienz und Effektivität der betrieblichen Abläufe immer wichtiger. Die Gestaltung der relevanten Geschäftsprozesse gehört daher zu den dringlichsten Fragestellungen aller IT-Dienstleister, ob nun eigenständige Unternehmen oder konzerninterne IT-Abteilungen. Diesen Prozessgedanken aufgreifend hat das Office of Government Commerce ein „Best Practice“-Rahmenwerk unter dem Label *IT Infrastructure Library (ITIL)* veröffentlicht.

Neben der Komplexität der Infrastruktur und der Prozesse stellt die sich ständig ändernde Umgebung einen weiteren wichtigen Faktor bei der Prozessgestaltung dar. Unternehmen müssen daher sich und ihre Prozesse an den ständigen Wandel am Markt und die sich ändernden Kundenbedürfnisse anpassen. Andernfalls können sie dem immer stärker werdenden Konkurrenzkampf am Markt nicht standhalten. Aus diesem Grund benötigt man flexible Tools, welche verschiedene (Teil-)Prozesse unterstützen. Damit die bereits getätigten Investitionen jedoch geschützt werden können, liefert das Konzept der *Service Oriented Architecture (SOA)* durch den Einsatz und die Definition von lose gekoppelten Diensten eine Möglichkeit, Tools und ihre Funktionalitäten flexibel und sowohl programmiersprachen- als auch technologieunabhängig zu integrieren.

Im Rahmen dieser Arbeit wird ein SOA-basiertes Konzept entwickelt und für den ITIL Incident-Management-Prozess, welcher zuvor verfeinert und mittels EPKs modelliert wird, prototypisch umgesetzt. Dazu werden Dienstschnittstellen, Nachrichtenformate, funktionale Kapseln sowie Workflows in Zusammenhang mit der Orchestierung definiert. Des Weiteren wird deren Umsetzung in den entsprechenden Web Service-Technologien (WSDL, BPEL4WS) vorgestellt. Um eine Wiederverwendbarkeit des Konzepts zu gewährleisten, wird das Vorgehen bei der funktionalen Kapselung, der Orchestierung und ebenso bei der Umsetzung der Service Oriented Architecture möglichst generisch beschrieben und erst im Anschluss daran anhand des Incident-Management-Prozsses vorgestellt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1.

Einleitung

Das Ziel dieser Arbeit ist die Entwicklung eines SOA-basierten Konzepts zur Unterstützung des ITIL Incident-Management-Prozesses. Bevor das entworfene Konzept vorgestellt wird, sollen in diesem Kapitel erst einmal die Ideen, die sich hinter diesem Ziel verbergen, motiviert werden. Auch die Aufgabenstellung soll in diesem Kapitel genauer beleuchtet werden, insbesondere, welche Anforderungen sich an das Konzept richten, welche Arbeitsschritte zur Erreichung des Ziels notwendig sind und wie die verschiedenen (Teil-)Ergebnisse in Bezug auf den Gesamtzusammenhang einzuordnen sind.

1.1. Motivation und Aufgabenstellung

Den Prozessgedanken aufgreifend, der immer stärker in den Vordergrund gestellt wird, hat das britische *Office of Government Commerce* ein „Best Practice"-Rahmenwerk u.a. für die Geschäftsprozesse des IT Service Managements (ITSM) in Form der *IT Infrastructure Library* veröffentlicht. Die ausgesprochenen Empfehlungen zur Gestaltung der verschiedenen Managementbereiche sollen es Unternehmen ermöglichen, ihre Prozesse effizienter und effektiver zu gestalten. In Anbetracht der ständig zunehmenden Größe und Komplexität der zu managenden Infrastruktur spielt gerade diese Effizienz der Geschäftsprozesse eine zentrale Rolle in Bezug auf „das Überleben am Markt".

Einer der in der ITIL definierten Service-Support-Prozesse ist das Incident Management, welches sich mit der Störungsbehandlung und den Dienstanforderungen sowie deren Koordination befasst. Allerdings reicht ein gut strukturierter Incident-Prozess allein nicht aus. Denn für die effiziente Behandlung der Incidents genügt es im Allgemeinen nicht, diese irgendwie zu beheben. Vielmehr ist auch die Art der Behebung von Bedeutung.

Mit zunehmender Komplexität ergibt sich daraus die Forderung, die verschiedenen Prozesse durch Tools zu unterstützen. Wie diese Tools genau auszusehen haben, richtet sich nach deren Einsatzgebiet und Umfeld. Im einfachsten Fall kann man sich im Incident Management eine Karteikarte oder Datenbank zur Erfassung der Störungsdaten vorstellen. ITIL empfiehlt ebenfalls den Einsatz geeigneter Tools zur Prozessunterstützung. Allerdings sollte dabei unter keinen Umständen das unternehmerische Umfeld vergessen werden, denn ungeeignete Toolunterstützung kann den Prozess genauso negativ beeinflussen wie gar keine Unterstützung.

Wie jedoch sollten solche Tools aussehen? Neben der eigentlichen Aufgabe, der Prozessunterstützung, tauchen auch immer neue Anforderungen für die eingesetzten Tools auf. Zum Beispiel sollten die verschiedenen Tools miteinander kommunizieren können. Denn was bringt es, wenn man den Datenbestand des Tools, das in dem einen IT-Bereich eingesetzt wird, händisch mit dem eines anderen abgleichen muss? Natürlich lassen sich u.U. Schnittstellen definieren, oder es existieren bereits proprietäre Schnittstellen. Wenn man alles von einem Hersteller bezieht, können sich diese Probleme verringern (unter der Annahme, dass dieser alle benötigten Unterstützungstools anbietet), aber man begibt sich damit auch in eine starke Abhängigkeit von diesem Anbieter.

Eine Alternative zu den Tools stellt das Konzept einer *Service Oriented Architecture (SOA)* dar, welches zur Prozessunterstützung und -gestaltung verwendet werden kann. In diesem Zusammenhang werden Dienste innerhalb der zu unterstützenden Prozesslandschaft identifiziert und modelliert, welche einzelne wiederverwendbare Funktionalitäten bereitstellen und miteinander interagieren können. Durch das Verbergen dieser Dienste hinter standardisierten Schnittstellen, welche bspw. mittels der *Web Service Description Language (WSDL)* beschrieben werden, können die einzelnen Dienste davon unabhängig in einer beliebigen Programmiersprache implementiert werden. Außerdem kann die Dienstimplementierung unter Beibehaltung der Schnittstelle durch eine lose Kopplung innerhalb der SOA leicht ausgetauscht werden, ohne dass die Interaktion mit dem jeweiligen Dienst modifiziert werden muss.

Im Rahmen dieser Arbeit soll ein dienstorientiertes Konzept im Sinne einer SOA für die Unterstützung des Incident Managements erarbeitet werden. Dieses Konzept wird dabei für den modellierten Prozess Funktionalitäten identifizieren, welche anschließend den Diensten der SOA zugeordnet werden. Im Rahmen dieser Identifikation wird dazu der Begriff der *funktionalen Kapselung* eingeführt, welcher eine zentrale Rolle in dieser Arbeit einnehmen wird. Im Anschluß daran werden die modellierten Bestandteile der SOA auf eine konkrete Implementierung abgebildet und prototypisch für den Incident-Management-Prozess vorgestellt.

1.2. Vorgehensweise

Nachdem das zu entwickelnde Konzept ausreichend motiviert und die Aufgabenstellung genauer betrachtet wurden, wird im Folgenden das Vorgehen zur Erreichung des angestrebten Ziels beschrieben, welches in Abbildung 1.1 dargestellt ist. Dabei soll insbesondere der Zusammenhang zwischen den einzelnen, hier als Ellipsen dargestellten Phasen innerhalb der Entwicklung des Konzepts verdeutlicht werden. Die Ellipsen sind hierbei als Aktivitäten zu verstehen, wohingegen die rechteckigen Elemente zwischen diesen Aktivitätsphasen als Artefakte betrachtet werden, die den Ausgangspunkt bzw. das Ergebnis einer Phase darstellen.

Als gegeben wird die Incident-Management-Beschreibung innerhalb der ITIL sowie das Konzept der SOA in Kombination mit Web Services, WSDL und BPEL betrachtet. Aus der ITIL ergibt sich für den Incident-Management-Prozess eine relativ abstrakte Beschreibung der Prozessstruktur. In der ersten Aktivitätsphase wird diese Prozessstruktur im Rahmen der Prozessanalyse verfeinert. Dazu werden mögliche Subprozesse und Aktivitäten ermittelt, woraus die Prozesslogik und die Inputs und Outputs extrahiert und konkretisiert werden müssen. Da diese Phase fast ausschließlich von dem zu analysierenden Prozess abhängt, wird sie direkt am Beispiel des Incident-Management-Prozesses durchgeführt. Durch Vorwegnahme dieses Teils der prototypischen Implementierung kann während der konzeptionellen Beschreibung der beiden folgenden Phasen auf den Incident-Management-Prozess als Beispiel Bezug genommen werden, wodurch verschiedene Situationen leichter veranschaulicht werden können.

Der zweite Schritt beinhaltet die Umsetzung der SOA-Idee. Dazu werden die in der Prozessanalyse ermittelten Inputs und Outputs, die Aktivitäten, aber auch die Prozesslogik als Ausgangspunkt für die Entwicklung des geplanten SOA-basierten Konzepts verwendet, um daraus die Schnittstellen, die funktionalen Kapseln sowie Informationen für die spätere Orchestrierung abzuleiten. Diese Phase wird in Kapitel 5 weitgehend unabhängig vom gewählten Prozess beschrieben.

Im Anschluss daran erfolgt gemäß Abbildung 1.1 die Implementierung des entwickelten Konzepts, wobei die Spezifikation der Schnittstellen in *WSDL (Web Service Description Language)* erfolgt, da die funktionalen Kapseln später mittels Web Services realisiert werden sollen. Für die Umsetzung der Orchestrierung wird *BPEL (Business Process Execution Language)* verwendet. Mit einer überwiegend theoretischen Beschreibung der Konzeptumsetzung wird die Phase der SOA-Implementierung abgeschlossen. Anschließend erfolgt die prototypische Umsetzung der letzten beiden Konzeptphasen am Beispiel des Incident-Management-Prozesses.

Nachdem kurz die einzelnen Phasen dieser Arbeit betrachtet wurden, befasst sich der nächste Abschnitt mit einigen grundsätzlichen Anforderungen an das zu entwickelnde Konzept, bevor der letzte Abschnitt einen Überblick über die Gliederung der Arbeit liefert.

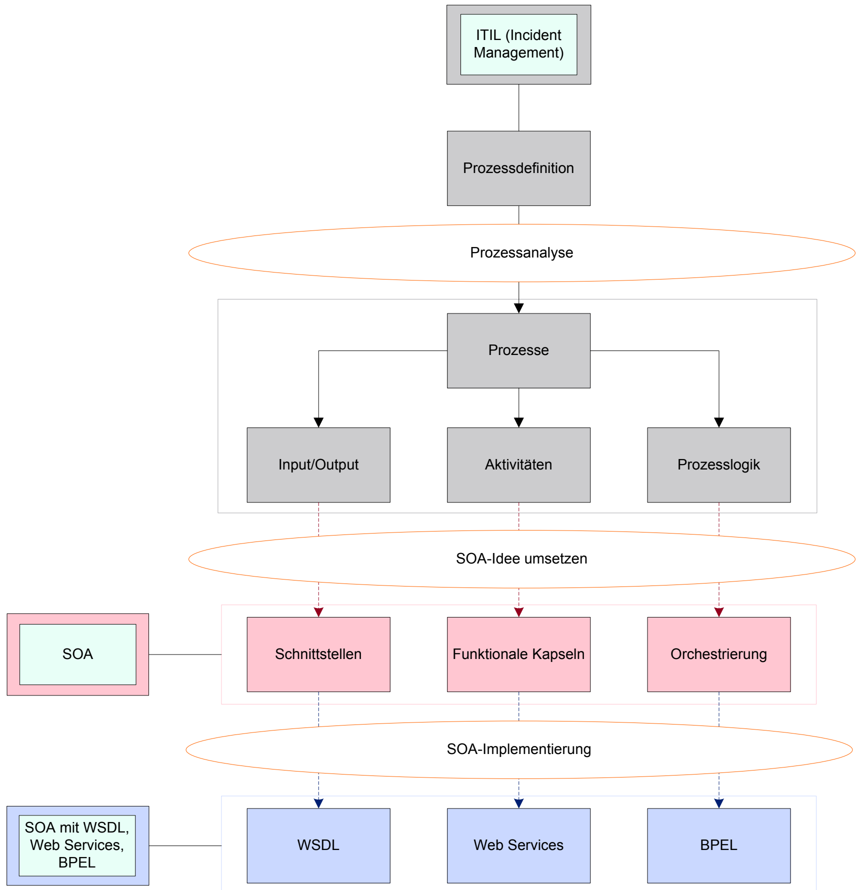


Abbildung 1.1.: Vorgehensweise

1.3. Anforderungen an das zu entwickelnde Konzept

In diesem Abschnitt werden einige Anforderungen beschrieben, welche sich an das zu entwickelnde SOA-basierte Konzept in dieser Arbeit richten. Auch wenn im Rahmen dieser Arbeit das Konzept anhand des ITIL Incident-Management-Prozesses vorgestellt wird, stehen dabei nicht die Anforderungen des Incident Managements im Vordergrund, sondern die an das Konzept an sich, da dieses und dessen Anwendung grundsätzlich nicht auf den Incident-Management-Prozess beschränkt ist.

Die Identifikation von Anforderungen, wie sie in diesem Abschnitt formuliert werden, ist für das Konzept bzw. das Vorgehen dabei insofern notwendig, als dieses zum einen möglichst allgemein und damit wiederverwendbar gestaltet und zum anderen jedoch für den Incident-Management-Prozess beispielhaft umgesetzt werden soll. Aus diesem Grund basiert die erste Anforderung auch auf der Gestaltung des Prozesses.

Kann ein Unternehmen einen wichtigen Prozess nicht an die Bedürfnisse des Kunden bzw. des Marktes anpassen, so wird es sich nicht lange gegen die ständig wachsende Konkurrenz zur Wehr setzen können. Aus diesem Grund wird das Unternehmen versuchen, den Incident-Management-Prozess wie alle seine Prozesse möglichst dynamisch zu gestalten. Diese Dynamik innerhalb des Prozesses darf allerdings auf das Konzept an sich keinen Einfluss haben.

Neben dem Prozess entwickeln sich auch die Umgebung des Prozesses, die eingesetzten Technologien, Plattformen und auch die verwendeten Programmiersprachen weiter. Ein Beispiel hierfür ist die zunehmende Verlagerung auf das Internet und die Datenmodellierung in XML-Formaten. Damit das Konzept auch in komplexen sich ständig ändernden Umgebungen anwendbar ist, muss es an die jeweilige vorliegende Situation anpassbar sein. Besonderes Augenmerk liegt dabei auf der Veränderbarkeit der Prozessbeschreibungen. Ändert sich bspw. die Beschreibung der Prozessmodellierung, welche in dieser Arbeit mittels EPKs erfolgte, dann muss das Konzept nicht verworfen werden, sondern nur die Abbildung der ersten Konzeptphase auf die SOA-Idee angepasst werden.

Ausgehend von der relativ abstrakten ITIL-Prozessbeschreibung kann der Incident-Management-Prozess auf unterschiedlichen Verfeinerungs- und Konkretisierungsstufen betrachtet werden. Da man sich im Rahmen der Konzeptanwendung jedoch nicht auf eine spezielle Modellierungstiefe festlegen müssen soll, ist als eine weitere Anforderung die weitgehende Abstraktion des Konzepts von der Granularität der Prozessmodellierung zu verlangen. Zwar kann nicht verhindert werden, dass die Granularität der Prozessgestaltung die Granulariät der funktionalen Kapselung und der Orchestrierung beeinflusst, dennoch darf dies nicht das allgemeine Vorgehen beeinflussen.

Die als Ausgangspunkt für die Prozessverfeinerung dienende ITIL liefert nicht nur eine Beschreibung für den Incident-Management-Prozess, sondern deckt mit ihren Empfehlungen weite Bereiche im IT-Management ab. Würde für jeden der dort beschriebenen Bereiche oder sogar für jeden Prozess ein eigenes Konzept für die angestrebte SOA-basierte Unterstützung notwendig sein, so wäre die resultierende Konzeptvielfalt kaum mehr zu überblicken. Selbst wenn man sich auf den Incident-Management-Prozess beschränkt, kann dieser in verschiedenen Unternehmen unterschiedlich modelliert werden. Aus diesem Grund soll das Konzept zwar mit Blick auf den Incident-Management-Prozess entwickelt werden, jedoch nicht so speziell auf diesen ausgerichtet werden, dass es nicht auf verschiedene Modellierungen dieses Prozesses oder auch auf andere verwandte Prozesse angewandt werden kann. Bei dieser angestrebten kanonischen Vorgehensweise sollen dabei sowohl der durch ITIL vermittelte Prozessgedanke als auch die zu etablierenden wieder verwendbaren Services einer SOA berücksichtigt werden. Dazu soll der Prozess ausgehend von einer vorhandenen Prozessbeschreibung in verschiedene wiederverwendbare Services aufgespalten werden, wobei die Prozessstruktur jedoch erhalten bleibt muss.

Nachdem bereits auf die Anpassbarkeit des Konzepts in Bezug auf eine Erweiterung der Prozessbeschreibung eingegangen wurde, ist in diesem Zusammenhang auch die weitgehende Unabhängigkeit von der Implementierung einzuordnen. Diese bezieht sich vornehmlich auf die Abbildung der zu entwickelnden SOA auf die entsprechende Dienstrealisierung, die im Rahmen dieser Arbeit mit Web Services, WSDL und BPEL beschrieben wird, da die gewählte Programmiersprache bei der Implementierung der Services keine Rolle spielt.

Die in diesem Abschnitt identifizierten Anforderungen an das zu entwickelnde Konzept bilden den gedanklichen Hintergrund für die Analyse des Incicent-Management-Prozesses, die anschließende Umsetzung der SOA-Idee und schließlich die Implementierung der SOA mittels Web Services, WSDL und BPEL erfolgt.

1.4. Gliederung der Arbeit

In den ersten Kapiteln werden dem Leser die Grundzüge des zu entwickelnden Konzepts vorgestellt. Angefangen wird in Kapitel 2 mit der Vorstellung der Ideen, welche hinter einer Service Orientiented Architecture stehen und inwieweit sie für das zu entwickelnde Konzept von Nutzen sind. Kapitel 3 bietet einen kurzen Einblick in die IT Infrastructure Library. Insbesondere werden die verschiedenen Service-Management-Bereiche mit Schwerpunkt auf den Support-Prozessen (Kap. 3.2.1) vorgestellt.

In Kapitel 4 wird das Incident Management genauer betrachtet. Dabei werden die einzelnen Teilaktivitäten und Interaktionen mit anderen Prozessen identifiziert und graphisch mittels *Ereignisgesteuerten Prozessketten (EPKs)* modelliert.

Aus der Prozesslogik, den Aktivitäten und den Input-/Outputdaten des Incident-Management-Prozesses werden im Kapitel 5 die Schnittstellen, die funktionalen Kapseln und die Orchestrierung abgeleitet. Da im Rahmen dieser Arbeit die Service Oriented Architecture in Form von Web Services realisiert werden soll, werden die Schnittstellen zu den funktionalen Kapseln in Kapitel 6 mittels Web Service Description Language (WSDL) beschrieben. Für die Orchestrierung wird die Business Process Execution Language(BPEL) verwendet.

Diese zwei letzten Phasen des Vorgehens werden in den Kapiteln 5 und 6 eher allgemein betrachtet. Deshalb wird in Kapitel 7 die mit der Prozessanalyse die in Kapitel 4 begonnene prototypische Implementierung des Konzepts für den Incident-Management-Prozess wieder aufgenommen und vervollständigt. Anschließend werden im Kapitel 8 die Erkenntnisse dieser Arbeit noch einmal zusammengefasst.

Kapitel 2.

Service Oriented Architecture

Wie die Aufgabenstellung bereits verrät, liegt der erste Schwerpunkt dieser Arbeit auf der Service Oriented Architecture (SOA). Bevor nun das zu entwickelnde Konzept vorgestellt werden kann, werden in diesem Kapitel zuerst die Grundbegriffe einer SOA charakterisiert. Anschließend werden die Konzeptideen und die Vorteile einer SOA betrachtet, bevor die im Rahmen dieser Arbeit mit einer Service Oriented Architecture in Verbindung stehenden Begrifflichkeiten vorgestellt werden.

2.1. SOA-Überblick

Eine *Service Oriented Architecture* (SOA) [KBS 05] ist ein System- und Anwendungsarchitekturkonzept, das die Bereitstellung fachlicher Dienste in Form von wieder verwendbaren, möglichst lose gekoppelten, so genannten *Services* unterstützt. Im Kontext von SOA ist ein *Service* eine mittels Software-Komponente(n) implementierte Funktionalität, die über eine definierte, veröffentlichte und standardisierte Schnittstelle durch andere Services oder Anwendungen in Anspruch genommen werden kann.

Damit solch ein Service in Anspruch genommen werden kann, wird von einem Dienst-Nehmer (*Service Requestor*) eine Anfrage (*Service Request*) an den Dienst-Geber (*Service Provider*) gestellt, welcher daraufhin eine Folge von Aktionen ausführt und u.U. eine Antwort (*Sevice Response*) zurücksendet. Die gerade vorgestellten Rollen (Service Provider und Service Requestor), die von Akteuren in einer SOA eingenommen werden können, sind zwar innerhalb dieser Interaktionen fest, können jedoch sowohl auf die Zeit als auch den Interaktionspartner bezogen variieren (siehe auch Abschnitt 2.3).

Eine SOA kann prinzipiell auf jeder dienstbasierten Technologie wie beispielsweise Enterprise Java Beans [Sun 04] oder CORBA [OMG 01b, OMG a] aufgebaut werden. Da Services in unterschiedlichen Programmiersprachen und auf unterschiedlichen Systemplattformen realisiert werden können, wird eine SOA häufig zur Anwendungsintegration genutzt. Im Rahmen dieser Arbeit sollen die Dienste mittels Web Services [ZTP 03, W3C 04a] realisiert werden. Die Schnittstellenbeschreibungen werden dementsprechend in Web Service Description Language (WSDL) [W3C 05b, NEW 02] formuliert. Die Kommunikation zwischen den einzelnen Services erfolgt mittels SOAP-Messages [W3C 03a] .

2.2. SOA-Konzeptidee

Im ersten Abschnitt wurde die Service Oriented Architecture als ein Achitekurkonzept definiert, in dessen Fokus die Bereitstellung fachlicher Dienste steht. Dieser Abschnitt wird sich mit der grundlegenden Idee des SOA-Konzepts befassen. Außerdem soll in Form eines kurzen Exkurses das OASIS-Refernzmodell für eine Service Oriented Architecture vorgestellt werden.

Vom Ansatz, Daten und ihre zugehörige Prozesslogik bzw. Verarbeitung in eigenständige Objekte zusammenzufassen, wird abgewichen (vgl. [Hao 03]). Vielmehr ist das Ziel einer SOA, die bereitgestellten Funktionalitäten aus den Objekten oder Anwendungen herauszuziehen und in Dienste zu kapseln. Dadurch müssen sie nur einmal implementiert werden, können aber mehreren Applikationen oder auch anderen

Services zur Verfügung gestellt werden. Redundante Implementierung wird vermieden. Außerdem können neue Anwendungen schneller entwickelt werden, da Funktionalitäten nicht neu programmiert werden müssen, falls sie bereits als Service zur Verfügung stehen. Dieser Ansatz des Neu-Zusammenbauens bereits existierender Dienste wird unter dem Begriff *Orchestrierung* in Verbindung mit der Business Process Execution Language (BPEL) in das zu entwickelnde Konzept einfließen. Der Ansatz ermöglicht es außerdem, Daten eigenständig zu halten, und dass alle Anwendungen auf dieselben aktuellen Daten zugreifen, wodurch redundante Datenhaltung und nächtliches Aktualisieren der einzelnen Datensilos zum Herstellen eines konsistenten Datenbestandes überflüssig werden.

Indem die Funktionalitäten gekapselt werden, ist beim Design und bei der Entwicklung von Anwendungen eine Modularisierung möglich, d.h. die einzelnen Funktionalitäten bzw. Services können unabhängig voneinander entwickelt, implementiert, getestet und auch modifiziert werden. Die Modifikation ist dabei nur an einer Stelle durchzuführen, und nicht in jedem Objekt oder in jeder Anwendung, die diese Funktionalität implementiert hat (vgl. [Soik 05]), wodurch solche Systeme bedeutend flexibler und auch einfacher an zukünftige Anforderungen anpassbar sind. Ebenfalls können die Vorteile existierender Systeme (z.B. Legacy-Systeme) adaptiert werden, ohne diese modifizieren zu müssen, was auch unter dem Schlagwort „Investitionsschutz“ in der IT-Organisation von Bedeutung ist.

Betrachtet man die geplante Einführung einer neuen Architekturlandschaft innerhalb der bestehenden IT-Landschaft eines Unternehmens, so stellt man fest, dass, auch wenn die neue Architektur enorme Vorteile bietet, diese Einführung trotzdem immer wieder hinausgeschoben wird, da sie auch enorme Risiken birgt. Ein altbekanntes Sprichwort, „Never change a running System!“, greift hier sehr gut. In dieser Situation bietet SOA neben der Flexibilität und langfristigen Kostenreduktion auch noch den Vorteil, dass, wenn als Projekt organisiert, die Etablierung einer Service Oriented Architecture schrittweise möglich ist. Dadurch wird das Risiko eines Scheiterns bei der Realisierung deutlich verringert.

Wie bereits erwähnt, ist das Konzept der SOA unabhängig von der Programmiersprache. Dies wird dadurch erreicht, dass die Implementierung der Services hinter wohldefinierten und standardisierten Schnittstellen verborgen wird. Betrachtet man wieder die Situation, dass ein Service Consumer einen Service des Providers nutzen will, so muss er nur die Schnittstelle für diesen Dienst kennen. Wie der Provider die Aufgabe erledigt, insbesondere in welcher Programmiersprache, spielt für den Consumer i.d.R. keine Rolle. Dieser Aspekt der „lose gekoppelten“ Systeme, der häufig als erstes genannt wird, wenn man von einer Service Oriented Architecture spricht, ist speziell in verteilten Systemen von großer Bedeutung. Insbesondere wird die Situation noch dadurch verschärft, wenn unterschiedliche Objektmodelle, Plattformen und Programmiersprachen zum Einsatz kommen.

Geht man von der gerade beschriebenen Situation in einem verteilten System aus, dann bietet eine SOA noch zwei weitere Merkmale, nämlich asynchrone Kommunikation und das „Publishing“ von Diensten. Neben der asynchronen Kommunikation ist zwar eine synchrone Kommunikation auch denkbar, allerdings erhöht sich der damit verbundene Aufwand zum Verwalten bestehender Sessions, speziell wenn diese lang sind. Außerdem kann die Skalierbarkeit abnehmen sowie sich die Kopplung verstärken. Realisisiert man die Dienste jedoch zustandslos, d.h. alle für die Bearbeitung des Service Request notwendigen Informationen sind in der Nachricht enthalten, so reduziert sich an dieser Stelle der Aufwand.

Das Publishing von Diensten greift u.a. dann, wenn Services über das Netzwerk zugänglich gemacht werden. Im Zuge dessen ist ein Register sinnvoll, bei dem Service Provider ihre angebotenen Services registrieren. Sucht ein Service Consumer einen Service, so kann er dessen Schnittstellenbeschreibung über das Register abfragen. Ebenso kann das Register genutzt werden, um Informationen über den Service oder auch den Service Provider abzurufen.

Kommen wir aber noch einmal auf die Services zurück. Betrachtet man eine SOA als eine Art Netzwerk, so stellen die Knoten innerhalb dieses Netzes die Dienste dar. Diese Dienste stehen jedoch nicht vollkommen isoliert im Raum. Vielmehr wird eine Service Oriented Architecture nicht nur durch Services, sondern auch durch die Beziehungen der Services untereinander und zur Umgebung charakterisiert (vgl. [Satt 05]). In Abbildung 2.1 sind Services dargestellt, die untereinander Daten austauschen.

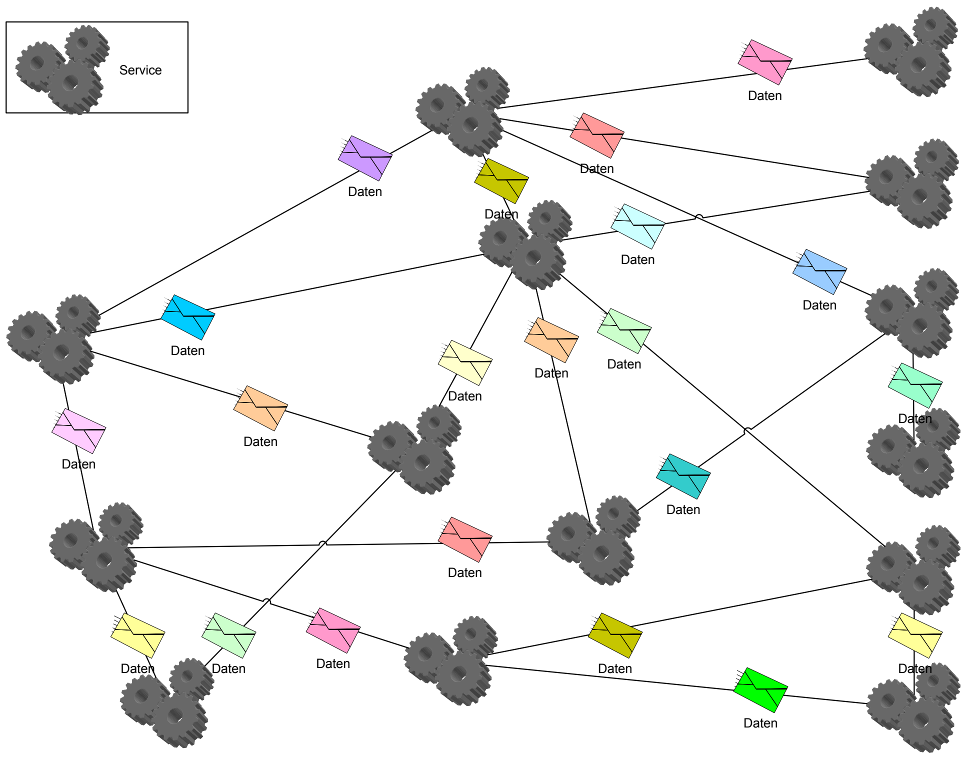


Abbildung 2.1.: Vermaschte Kommunikation innerhalb einer SOA

Die Kommunikation in dieser Abbildung ist bewusst nicht strukturiert oder in irgendeiner Weise koordiniert dargestellt, da dies nicht Aufgabe einer Service Oriented Architecture ist. Die SOA legt nur fest, wie sich die unterschiedlichen Services verstehen und miteinander interagieren können.

Dennoch ist eine koordinierende Komponente durchaus denkbar (siehe Abb. 2.2). Ein möglicher Grund für die Einführung einer zentralen Komponente ist das Weiterleiten der Anfragen eines Services an einen anderen. Dabei ist es jedoch nicht zwingend notwendig, dass diese Komponente auch steuernd in die Kommunikation zwischen den Diensten eingreift. Vielmehr erfährt sie vom anfragenden Service, an welchen Service der Service Request weitergeleitet werden soll. Das gleiche gilt auch für den Service Response. Die dargestellte zentrale Komponente „tunnelt" quasi die Nachrichten zwischen den einzelnen Diensten. Häufig wird dieser Aspekt mit dem Begriff eines „Service Bus" assoziiert (siehe [Fran 05]). Dies hat den Vorteil, dass, wenn sich der Ort eines Dienstes ändert, nur die Weiterleitung durch die zentrale Komponente angepasst werden muss, wodurch zusätzlich eine Ortstransparenz geschaffen wird. Des Weiteren kann man die Aufgaben Komponente noch mit einer Transformationsfunktion erweitern, welche die auszutauschenden Datenformate auf die jeweiligen Bedürfnisse der beteiligten Services anpasst.

Neben der Weiterleitung von Anfragen und Antworten sowie der Transformation der Daten kann diese Komponente auch noch mit Prozesswissen ausgestattet werden. Im Rahmen der Orchestrierung wird eine darauf aufbauende Interaktion der beteiligten Dienste z.B. mit BPEL beschrieben. Dabei wird spezifiziert, welche Services wann kontaktiert werden müssen, um das angestrebte Ziel zu erreichen (siehe auch Abb. 2.3 links). Da der Begriff der Orchestrierung im weiteren Verlauf der Arbeit immer wieder auftaucht und neben diesen Begriff auch noch der Begriff Choreographie existiert, sollen an dieser Stelle die beiden Begriffe unterschieden werden.

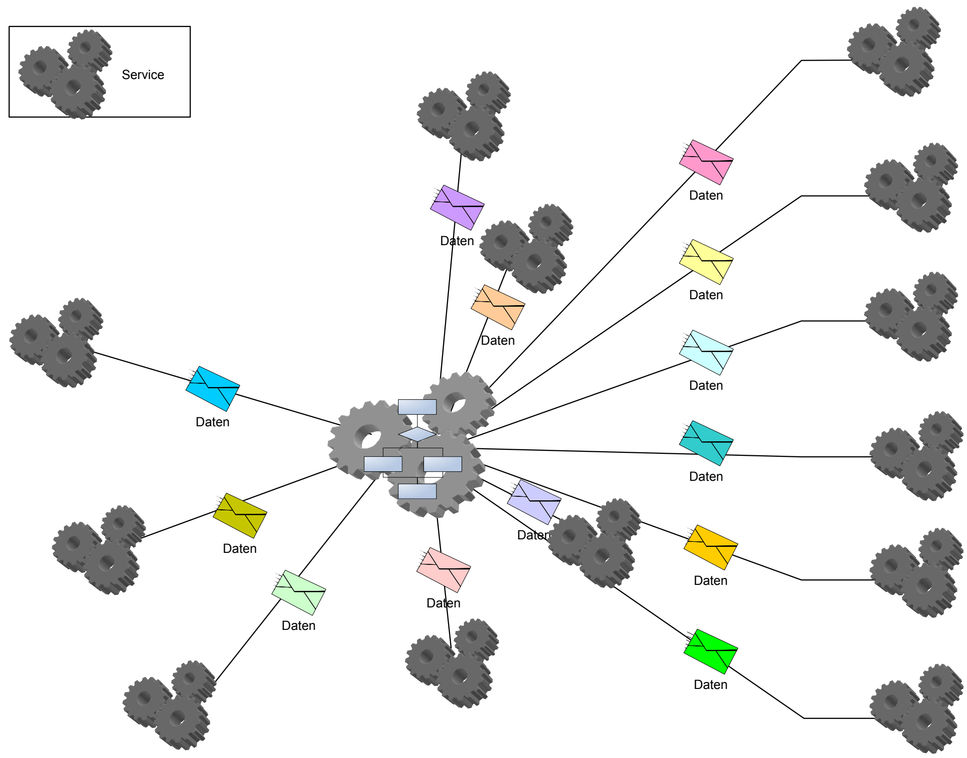


Abbildung 2.2.: Zentrale Komponente zur Steuerung

Bei der Orchestrierung existiert ein zentraler Prozess, der die Kontrolle über die beteiligten Web Services hat. Dieser Prozess, der seinerseits ebenfalls ein Web Service sein kann, koordiniert die Ausführung verschiedener Operationen der Web Services. Die involvierten Web Services wissen nicht, dass sie komponiert werden und Teil eines höheren Geschäftsprozesses sind und müssen es auch nicht wissen. Nur der zentrale Koordinator der Orchestrierung weiß dies. Orchestrierung wird normalerweise bei privaten Geschäftsprozessen verwendet und wird links in Abbildung 2.3 schematisch gezeigt.

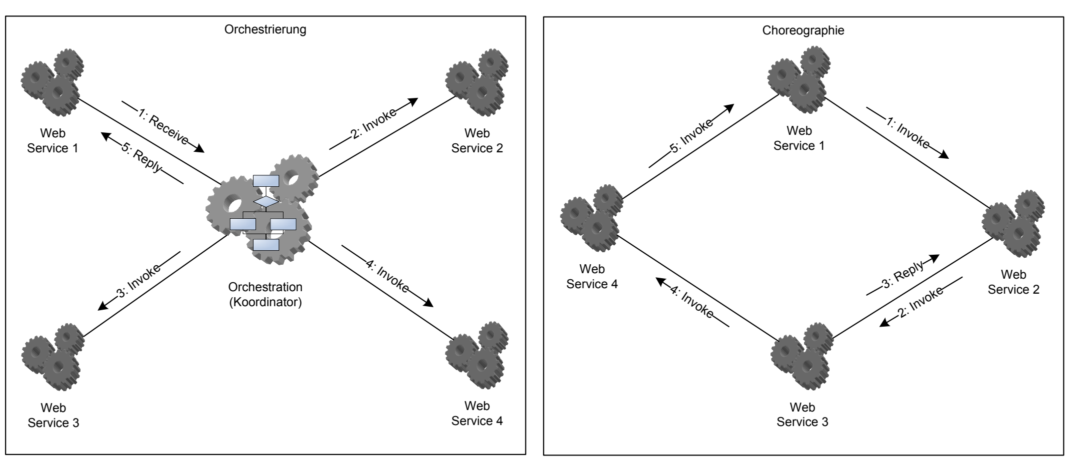


Abbildung 2.3.: Orchestrierung vs. Choreographie (vgl. [JMS 04])

Die Choreographie dagegen kennt keinen zentralen Koordinator. Vielmehr weiß jeder involvierte Web Service, wann seine Operationen auszuführen sind, und mit wem er zu interagieren hat (siehe Abb. 2.3 rechts). Choreographie ist als eine Kollaboration zu verstehen, deren Ziel der Austausch von Nachrichten in einem Geschäftsprozess ist. Alle Beteiligten müssen dazu den Geschäftsprozess, die auszuführenden Operationen und das Timing für den Nachrichtenaustausch kennen.

In diesem Abschnitt wurden die grundsätzlichen Ideen des SOA-Konzepts, die Verwendung standardisierter Schnittstellen, über welche auf in Services gekapselte Funktionalitäten zugegriffen werden kann, sowie das Neu-Zusammenbauen im Rahmen der Orchestrierung motiviert und vorgestellt. Ebenso wurden neben der Technologie- und Programmiersprachenunabhängigkeit weitere wichtige Aspekte hervorgehoben, welche hier nochmal kurz zusammengefasst werden sollen (vgl. [KBS 05, Kraf 05]):

- lose Kopplung:
  Die Kommunikation erfolgt ausschließlich über standardisierte Schnittstellen, welche die Implementierung der dahinterliegenden Services verbergen.
- asynchrone Kommunikation und zustandslose Dienste:
  Dadurch reduziert sich der Verwaltungsaufwand, und das System ist besser skalierbar.
- Programmiersprachenunabhängigkeit:
  Die Implementierung der Services ist nicht an eine spezielle Programmiersprache gebunden, da diese hinter einer Schnittstelle verborgen wird, über welche die Kommunikation erfolgt.
- Service Publishing:
  Service-Schnittstellen und Beschreibungen werden in einer Art Register verwaltet, welches auch verschiedene Suchfunktionen unterstützt.
- Verkürzung der Entwicklungszeiten (Flexibilität):
  Durch die Verwendung vorhandener Services reduziert sich sowohl die Zeit als auch der Aufwand für die Koordinierung, den Test und die Produktionseinführung.
- Risikominimierung:
  Die Einfachheit von SOA, Wiederverwendung und schrittweises Vorgehen reduziert das Projektrisiko, insbesondere, wenn bereits erprobte Funktionalitäten wieder verwendet werden.
- Aktualität der Unternehmensdaten:
  Die Services, welche durch von den Anwendungen gemeinsam genutzt werden, greifen auf dieselben Daten zu. Dadurch werden Inkonsistenzen und aufwändige Datenaktualisierung vermieden.
- Investitionsschutz (mittel- und langfristige Kostenreduktion):
  Vorhandene Services und Applikationen werden wieder verwendet, und damit amortisieren sich die getätigten Investitionen über die Zeit.
- Vereinfachung:
  Die Vereinfachung bezieht sich auf die bessere Strukturierung der Anwendungslandschaft, die Reduktion von doppelten Implementierungen gleicher Funktionalitäten sowie die vereinfachte Kommunikation zwischen der IT und den darauf aufsetzenden Anwendungen über standardisierte Schnittstellen.

Bevor die im Rahmen dieser Arbeit zur Realisiserung einer Service Oriented Architecture verfolgten Konzepte vorgestellt werden, soll zunächst in Form eines kurzen Exkurses noch das SOA-Referenzmodell von OASIS betrachtet werden, welches sich momentan noch im Status „Draft“ (Entwurf) befindet.

Exkurs: OASIS-Referenzmodell für SOA

Abbildung 2.4 zeigt die im Referenzmodell für eine Service Oriented Architecture von OASIS identifizierten Aspekte. Um die Services sind die Konzepte Visibility, Interaction und Real World Effect gruppiert, welche die dynamische Perspektive betrachten. Visibility bezieht sich auf das „Einander Sehen“ und setzt

Awareness, Willingness und Reachability voraus. Im Konzept der Interaction werden das Behavior Model und das Information Model betrachtet, wobei sich das Behavior Model mit den Aktionen und dem Prozess und das Information Model grundsätzlich mit der Struktur und Semantik der auszutauschenden Informationen befasst.

Kommt man noch einmal auf die grundsätzliche Idee einer SOA zurück, so ist doch eines der Ziele, dass Service Consumer versuchen, mittels Diensten, die von Service Providern zur Verfügung gestellt werden, etwas zu realisieren. „Trying to get the service to do something", nennt es OASIS und bezeichnet dies gleichzeitig als Real World Effect. Das Konzept des Real World Effect steht oft in Verbindung mit einem Shared State. In diesem Zusammenhang wird die Akkumulation der Änderungen des Shared States einer Service-Interaktion als Real World Effect bezeichnet.

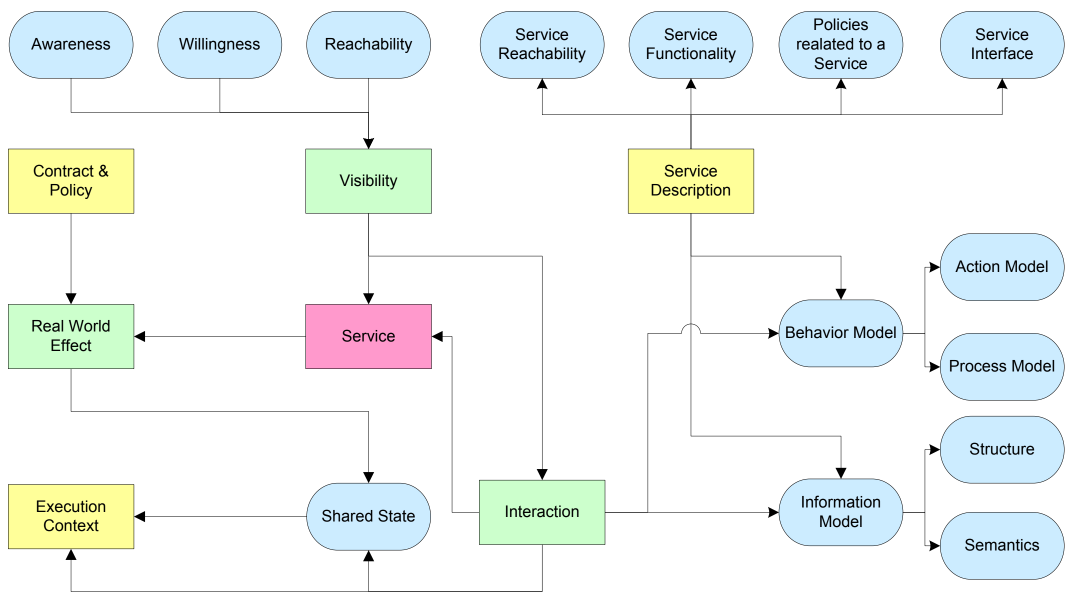


Abbildung 2.4.: OASIS-Referenzmodell für SOA (vgl. [OASI 05a])

Zur Unterstützung der Dynamik bei der Interaktion mit Diensten werden die Konzepte Service Description, Contract & Policy und Execution Context betrachtet. Die Servicebeschreibung repräsentiert jene Informationen, die notwendig sind, um den Service zu nutzen, wobei diese vom Kontext und den Bedürfnissen der beteiligten Parteien abhängen. Neben Informationen bzgl. der Erreichbarkeit und den angebotenen Funktionen ist besonders die Interfacebeschreibung von Bedeutung, durch welche die Serviceausführung und -implementierung vor dem Service Consumer verborgen wird. Contracts und Policies spielen besonders in grenzüberschreitender Interaktion eine wichtige Rolle. Policies stellen in diesem Zusammenhang Bedingungen und Voraussetzungen für die Nutzung, den Einsatz oder die Beschreibung eines Services, Contracts hingegen Vereinbarungen zwischen zwei oder mehreren Parteien dar. Der Ausführungskontext einer Service-Interaktion ermöglicht die Unterscheidung zwischen verschiedenen Diensten und verschiedenen Instanzen eines Dienstes. Er umfasst u.a. Infrastrukturelemente, Prozessentitäten, Policies und Vereinbarungen. Kurz gesagt beinhaltet der Execution Context alle notwendigen Informationen auf dem Pfad vom Service Consumer zum Service Provider zur Nutzung des Dienstes.

Nach der Vorstellung der Konzeptidee einer Service Oriented Architecture und diesem kurzen Exkurs zum OASIS-Referenzmodell für SOA werden im folgenden Abschnitt die für das zu entwickelnde SOA-basierte Konzept verwendeten Begriffe eingeführt und im Kontext einer Service Orientierten Architecture erklärt.

2.3. SOA mit Web Services, WSDL und BPEL

Dieser Abschnitt befasst sich mit der Realisierung einer Service Oriented Architecture mittels Web Services, WSDL (Web Service Description Language) und BPEL (Business Process Execution Language). Dabei sollen diese Begriffe als Grundlage für das zu entwickelnde Konzept erklärt, in Zusammenhang gebracht und in das Architekturkonzept von SOA eingeordnet werden.

Kern einer Service Oriented Architecture sind Services, welche ihre Funktionalitäten anderen Services und Anwendungen über standardisierte Schnittstellen zur Verfügung stellen (siehe Abschnitt 2.1), wobei sie aus Sicht des Service Requestor als unabhängige und eigenständige Funktionalitäten erscheinen, auch wenn sie selber wiederum andere Services nutzen. Da die Implementierungsdetails hinter der angebotenen Schnittstelle verborgen werden und die Kommunikation sich auf Nachrichteninhalte (Service Request und Response) beschränkt, spricht man auch von einer losen Kopplung zwischen den beteiligten Akteuren.

Obwohl eine SOA ein auf Services aufbauendes Architekturprinzip darstellt, ist der SOA-Servicebegriff nicht zwingend identisch mit dem des Web Service. Vielmehr ist das Prinzip der Web Services ein Technologiesatz zur Realisierung von Services im Sinne der SOA. Da im Rahmen dieser Arbeit Web Services zur Implementierung der funktionalen Kapseln des Incident-Management-Prozesses verwendet werden sollen, werden im Folgenden diese beiden Begriffe gleichbedeutend verwendet, auch wenn SOA-Services i.A. auch anderweitig implementiert werden können. Nachfolgend ist eine Definition für den Begriff Web Service abgebildet:

*„A **Web Service** implements an interface that describes a collection of network-accessible operations through standard XML messaging.“[ZTP 03]*

Folgende Rollen können innerhalb einer SOA identifiziert werden:

- Service Requestor/Consumer:
  Ein Service Requestor oder Service Consumer greift auf Web Services des Service Providers zu.
- Service Provider:
  Ein Service Provider stellt Web Services über ein Netzwerk einem Service Requestor zur Verfügung.
- Discovery Agent:
  Ein Discovery Agent leitet den Service Requestor an einen Service Provider weiter.

In Abbildung 2.5 ist die Interaktion zwischen Service Provider und Service Requestor nochmals graphisch dargestellt. Der Service Provider hat Dienste in Form von Web Services implementiert. Mittels WSDL wird für den jeweiligen Web Service die Schnittstelle (u.a. Input- und Outputdaten) spezifiziert. Allerdings wird durch WSDL nicht festgelegt (bzw. die WSDL-Beschreibung liefert keinen Aufschluss darüber), was der Web Service, der sich hinter der Schnittstelle verbirgt, leistet. Dies muss an anderer Stelle geschehen.
Innerhalb des WSDL-Files werden folgende Aspekte beschrieben (vgl. [Soik 05]):

- Wo der Service zu finden ist.
- Wie der Service angesprochen werden kann.
- Welche Funktionen aufgerufen werden können.
- Welche Datenformate unterstützt werden.

Grundsätzlich müssen Service Provider und Service Requestor neben dieser Schnittstellenbeschreibung nichts voneinander wissen, weswegen man auch von einer „losen Kopplung“ spricht. Allerdings ist für eine effektive Interaktion auch die Semantik der aufgerufenen Services wichtig, die in der „Binding and Implementation“-Beschreibung festgehalten wird. Die WSDL-Beschreibung sowie Informationen über den Service Provider und die Web Services werden i.d.R. in einem Service Register gespeichert. Häufig wird zur Implementierung des Dicovery Agents *UDDI (Universal Description, Discovery and Integration)* eingesetzt; auch *Web Service Inspector Language (WSIL)* ist möglich, soll aber im Rahmen dieser Arbeit nicht betrachtet werden. UDDI ist eine Art Register, über welches Informationen der Web Services abgefragt werden können. Die Art der Informationen unterteilt man in drei Klassen:

- Weiße Seiten, u.a. Name, Adresse etc. eines Unternehmens.
  Dies würde einem Telefonbuch entsprechen.
- Gelbe Seiten, u.a. Branche des Unternehmens, Produkte bzw. Dienstleistungen.
  Dies würde einem Branchenbuch entsprechen.
- Grüne Seiten, u.a. bereitgestellte Services, wie man mit dem Unternehmen in Verbindung treten kann, z.B. ein Verweis auf das zu verwendende WSDL-File.

Möchte nun ein Service Requestor einen Service in Anspruch nehmen, so muss er die Schnittstellenbeschreibung kennen. Hat er keine Informationen über die Schnittstelle, den Web Service bzw. den Service Provider, so kann er in einem UDDI-Register danach suchen. Hat er die WSDL-Beschreibung vom Register erfragt, so kann er einen Service Request an den Service Provider senden. Je nach Art des angesprochenen Services und dessen Spezifikation sendet der Service Provider u.U. einen Service Response zurück, allerdings kann die Anfrage auch abgelehnt werden. Die Art und Weise einer (möglichen) Antwort oder auch Ablehnung wird im WSDL-File spezifiziert.

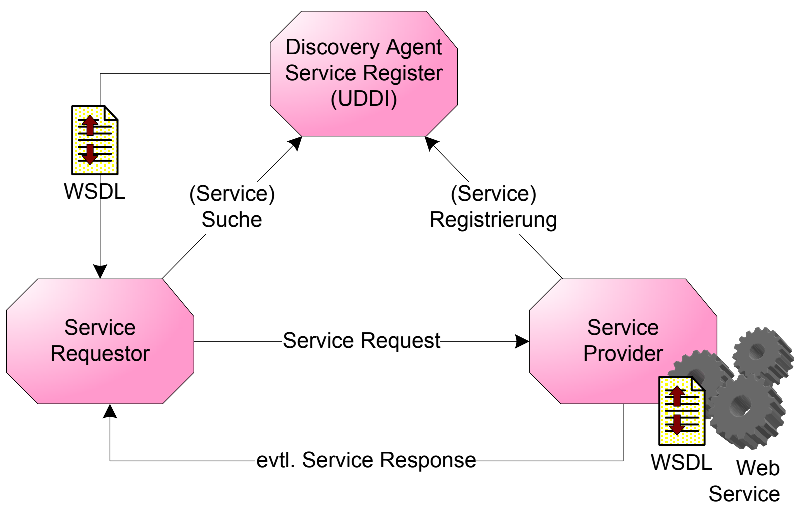


Abbildung 2.5.: Web Service Framework

Abbildung 2.6 zeigt einen häufig verwendeten Protokollstack für Web Services. Neben den bereits betrachteten Begriffen UDDI und WSDL sind in dieser Abbildung noch die Protokolle für den Datentransfer und Transport aufgezeigt[1]. Für den Datentransfer innerhalb der in Abbildung 2.5 beschriebenen Umgebung wird *SOAP*[2] verwendet. SOAP ist ein XML-basiertes Protokoll zum Austausch strukturierter Daten zwischen Applikationen, wobei es selber weder die Semantik der Applikationen noch ein Programmiermodell definiert oder fordert ([ZTP 03]). Da es der Kommunikation auch keine bestimmten Muster aufzwingt, kann eine SOAP-Message in verschiedenste Transportprotokolle wie z.B. HTTP eingebunden werden.

Kommen wir noch einmal auf WSDL zurück. WSDL bietet eine Möglichkeit den Austausch von Nachrichten zu beschreiben. Dennoch sind die modellierbaren Beziehungen inadäquat, um komplexe Kompositionen mehrerer Web Services auszudrücken, zu modellieren und zu beschreiben. Zur Kombination,

[1] Für einen Überblick über den Status der verschiedenen Referenzen siehe [OASI 05b].
[2] Vor der Version 1.2 wurde SOAP als Akronym für *Simple Object Access Protocol* verwendet.

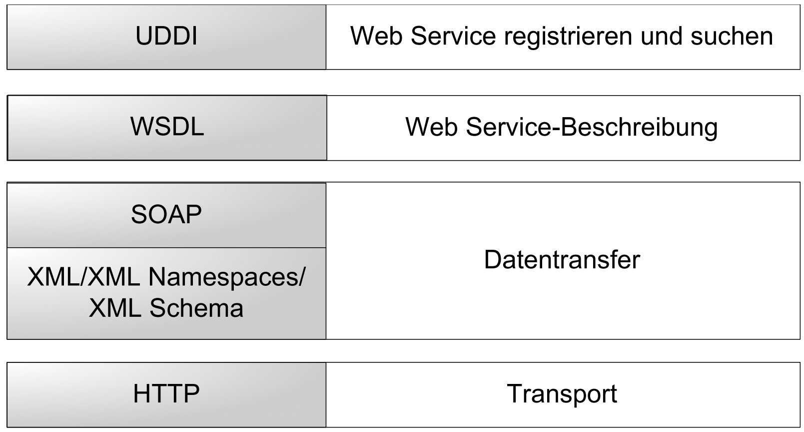


Abbildung 2.6.: Häufig verwendeter Protokollstack für Web Services

Orchestrierung und Koordination von Web Services in einer Service Oriented Architecture zu einem Geschäftsprozess wird u.a. BPEL verwendet, dessen Grundidee die Spezifikation einer verteilten serviceorientierten Anwendung auf hohem Abstraktionsniveau ist [Küst 05].

BPEL (Business Process Execution Language) koordiniert die Aktivitäten verschiedener miteinander interagierender Web Services. Es wird auf eingehende Nachrichten reagiert, Ergebnisse ausgewertet, Daten manipuliert (Transformationen und Übersetzung zum Zwecke der Interaktion mit Diensten) oder Nachrichten versendet. BPEL und Web Services ermöglichen es einem Unternehmen, private (unternehmensinterne) Prozesse zu implementieren. Aber es können auch Prozesse basierend auf Ad-hoc-Zusammenarbeit implementiert werden, wobei mit „ad hoc“ das dynamische Auswählen einer geeigneten Dienstinstanz verstanden wird und nicht das Verändern der Reihenfolge der Dienstaufrufe. BPEL ist ein offener Standard, und Partner können aufgrund der Portabilität direkt in den Workflow integriert werden. Die BPEL-Orchestrierung bezieht sich auf die Koordination von Prozessflüssen und definiert die Ausführungslogik der Komponenten. Zusammengefasst kann man sagen, dass BPEL eingesetzt wird, um die Prozessflusskoordination, aber auch die Verbindung zur Semantik, zu den parallelen Abläufen, der (komplexen) Ausnahmebehandlung und der Datentransformation zu definieren (siehe [SYS- 05a]). In diesem Zusammenhang liefert BPEL eine standardisierte und portable Sprache.

Nachdem im ersten und zweiten Abschnitt dieses Kapitels das abstrakte Konzept einer Service Oriented Architecture vorgestellt wurde, ist in diesem Abschnitt das Konzept der Web Services als konkrete Realisierungsmöglichkeit für die in der SOA beschriebenen Services vorgestellt worden. Dieses Web Service-Konzept verwendet bereits vorhandene Standards wie z.B. SOAP, WSDL und UDDI, welche ebenfalls kurz erklärt wurden. Der im Abschnitt 2.2 motivierte Einsatz einer zentralen Komponente (siehe Abbildungen 2.1 und 2.2) zur Steuerung der mittels Diensten realisierten Geschäftsprozesse spiegelt sich in der Verwendung von BPEL zur Orchestrierung wider.

Das folgende Kapitel wird sich mit der IT Infrastructure Library (ITIL) befassen. Dabei soll dem Leser erst ein kurzer Überblick über die verschiedenen in der ITIL betrachteten Managementbereiche verschafft werden, bevor die Prozesse des Service Managements genauer betrachtet werden, zu denen auch der Incident-Management-Prozess im Rahmen der Service-Support-Prozesse zählt. Nach dieser kurzen Einführung in das Service Management der IT Infrastructure Library wird anschließend in einem eigenen Kapitel der Incident-Management-Prozess nochmals aufgegriffen und genauer analysiert.

Kapitel 3.

IT Infrastructure Library

Neben der SOA ist der zweite Schwerpunkt dieser Arbeit das Incident Management als Teil der IT Infrastructure Library (ITIL). Aus diesem Grund wird dieses Kapitel einen kurzen Überblick über die verschiedenen Bereiche der ITIL geben, wobei im Speziellen auf die Support-Prozesse des Service Managements, zu denen auch der Incident-Management-Prozess gehört, eingegangen wird.

3.1. ITIL-Überblick

Bevor die einzelnen Bereiche des IT Service Managements vorgestellt werden, soll dieser Abschnitt dem Leser einen allgemeinen Überblick über die ITIL verschaffen. Dabei soll kurz ihre Grundstruktur vermittelt werden.

Die *IT Infrastructure Library (ITIL)* ist eine Sammlung von Büchern, welche Empfehlungen für die verschiedenen Managementbereiche innerhalb einer Organisation enthalten. Diese Managementbereiche wurden im Jahre 2000 vom Office of Government Commerce (OGC) in folgende fünf Bereiche aufgeteilt:

- Deliver IT Services (Service Delivery)
- Support IT Services (Service Support)
- Manage the Infrastructure (IT Infrastructure Management)
- The Business Perspective
- Managing Applications

Obwohl es auf den ersten Blick so aussieht, als ob diese fünf Bereiche nichts miteinander zu tun hätten, bestehen zwischen ihnen Abhängigkeiten, welche beim Studium der Bücher beachtet werden sollten.

Die ITIL-Prozessbeschreibungen sind eher als Orientierungshilfen denn als konkrete Beschreibungen zu verstehen, da die Prozesse auf einem hohen Abstraktionsniveau beschrieben werden und teilweise noch recht lückenhaft sind.

Abbildung 3.1 zeigt die verschiedenen Managementbereiche und ihre Einordnung in die Gesamtsituation. Als Kern der ITIL betrachtet man das Service Management[1], das sich in die Bereiche Service Delivery und Service Support untergliedert, welche nochmals unterteilt werden. Das so genannte „Business" oder auch die Geschäftsperspektive hat das Ziel, das Business Management mit den zugrunde liegenden Bestandteilen des Service Managements, des ICT Infrastructure Managements, des Application und Security Managements vertraut zu machen, da diese die Geschäftsprozesse unterstützen sollen. Eine Art Brücke zwischen der Technologie auf der einen und dem „Business" sowie dem Service Management auf der anderen Seite stellt das ICT Infrastructure Management (ICTIM)[2] dar, welches in vier Bereiche unterteilt wird. Verantwortlich für die Applikationen vom ersten Einsatz über Anpassungen im Betrieb bis hin zur Stilllegung ist das Application Management (siehe [OGC 02a]). Dies erfordert Interaktionen insbesondere

[1] [OGC 02c] befasst sich mit dem Planen, Implementieren und Verbessern des Service Managements innerhalb einer Organisation.

[2] „Information and Communications Technology Infrastructure Management" [OGC 02b] stellt u.a. die Zusammenhänge zum Service Management und Application Management dar. Siehe auch [Marc 04] und [Lehn 04].

mit dem ICT Infrastructure Management. Das Security Management befasst sich mit allen auftauchenden Sicherheitsaspekten und unterhält deswegen Schnittstellen mit allen Managementbereichen der ITIL (siehe [OGC 02d]).

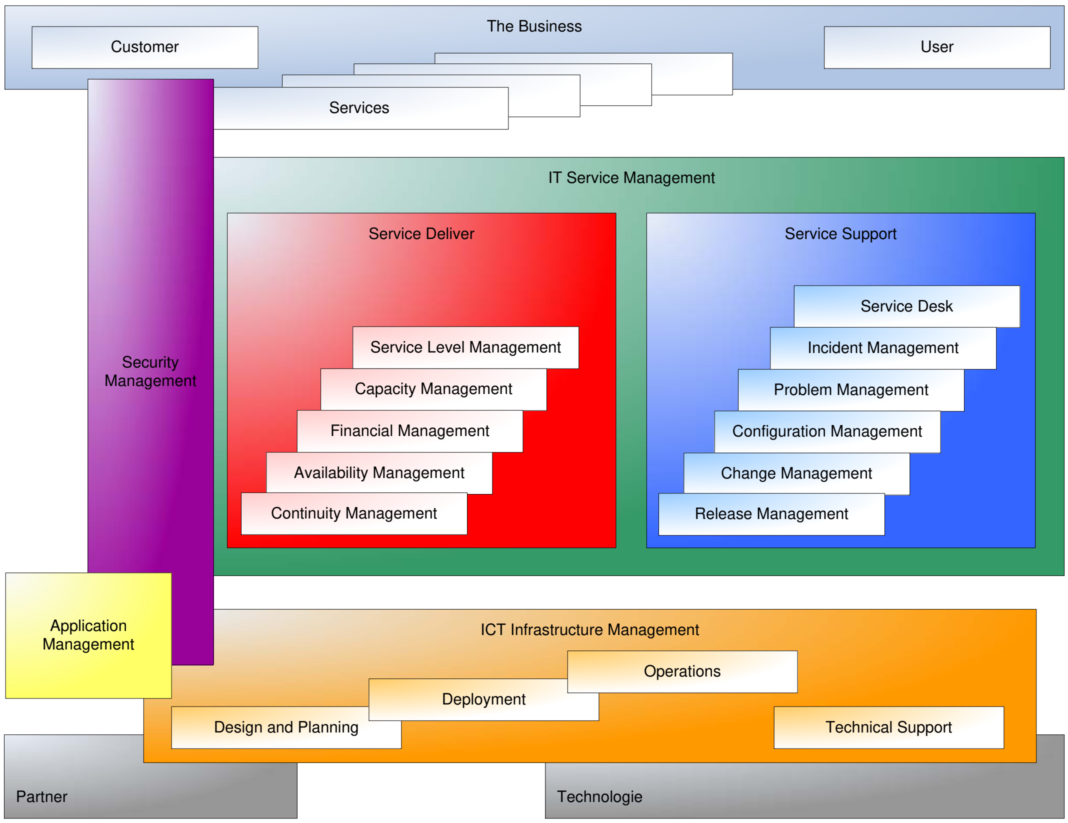


Abbildung 3.1.: ITIL-Übersicht (vgl. [OGC 02b])

Nach diesem kurzen Überblick über die Grundstruktur der ITIL soll im nächsten Abschnitt auf das Service Management eingegangen werden. Da im Rahmen dieser Arbeit ein Konzept zur Unterstützung des Incident-Management-Prozesses entwickelt werden soll, der ein Unterprozess des Service Supports innerhalb des Service Management ist, werden die anderen Managementbereiche nur so weit behandelt, wie es für die Aufgabenstellung notwendig ist.

3.2. IT Service Management

Dieser Abschnitt widmet sich dem IT Service Management, welches mit seinen Bereichen Service Support und Service Delivery den Hauptbestandteil der ITIL ausmacht. Bei der Betrachtung dieser „Core Titles“ wird der Schwerpunkt auf dem Service Support liegen. Dennoch werden auch die Service-Delivery-Prozesse nicht außer Acht gelassen.

Das IT Service Management (ITSM) befasst sich mit Prozessen und Vorgehensweisen, um IT-Dienstleistungen (Services) zielgerichtet, kundenorientiert und kostenoptimiert zu erbringen, zu planen, zu steuern und zu überwachen [OLB 02]. Momentan werden im Service Management zehn Prozesse beschrieben, die in die zwei Bereiche „Service Support“ (siehe Abschnitt 3.2.1) und „Service Delivery“ (siehe Abschnitt 3.2.2) aufgeteilt sind. Zusätzlich wird dem Service Support noch der „Service Desk“ (Abschnitt 3.2.1.2) zugeordnet, der keinen Prozess, sondern eher eine Funktion darstellt [OGC 00].

Zur Verdeutlichung der Zusammenarbeit der verschiedenen Service-Management-Prozesse betrachte man Abbildung 3.2. Der Service Desk nimmt eine Störungsmeldung entgegen und leitet sie an das Incident Management weiter, welches die Störung untersucht. Das Problem Management nimmt sich dieser Störung an, prüft, ob es sich um ein Problem handelt und analysiert dieses in Kooperation mit dem Capacity Management. Ist die Ursache gefunden, so wird dem Change Management ein RFC (Request for Change, Änderungswunsch) eingereicht. Um festzustellen, ob die gewünschten Änderungen durchführbar sind und auch durchgeführt werden sollten, ist u.a. eine Interaktion des Change Management mit dem Financial, dem Availability und dem Continuity Management notwendig. Wird der RFC autorisiert, implementiert das Release Management diese Änderung, wobei das Configuration Management dafür verantwortlich ist, dass die vorgenommenen Änderungen der Konfiguration korrekt und vollständig in der CMDB erfasst werden. Der in der Abbildung 3.2 dargestellte Ablauf sowie die Interaktionen sind beispielhaft und sollen nur zum besseren Verständnis beitragen.

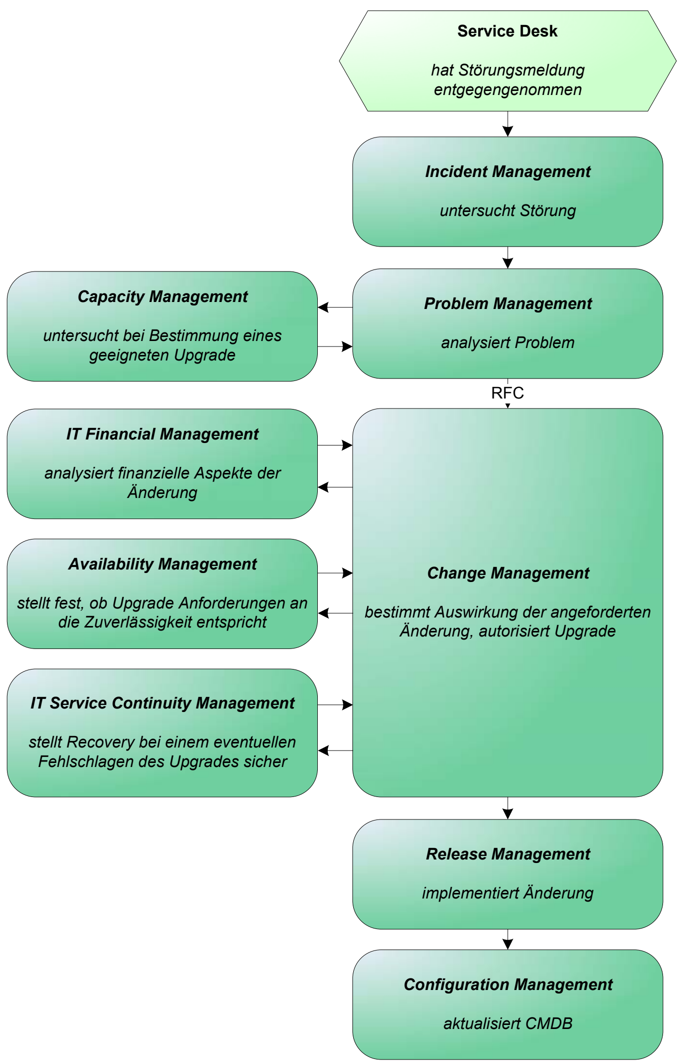


Abbildung 3.2.: Interaktion von Prozessen in ITIL (Beispiel) (Quelle: [Bren 02])

Aufgrund der Zusammenhänge zwischen den einzelnen Bereichen des Service Management werden in den nächsten beiden Abschnitten die Prozesse des Service Supports und des Service Delivery vorgestellt.

3.2.1. Service Support

In diesem Abschnitt werden kurz die Ziele und Aufgaben der Managementprozesse des Service Supports vorgestellt. Da im Rahmen dieser Diplomarbeit für eine ausführlichere Betrachtung kein Platz ist, empfiehlt sich [OGC 00] und [ITS 02] für ein vertiefendes Studium.

3.2.1.1. Incident Management

Der Prozess *Incident Management* hat das Ziel, aufgetretene *Störungen (Incidents)* möglichst schnell zu beheben, dadurch die Funktionalität eines Services wiederherzustellen und somit die negativen Auswirkungen einer Störung auf die Geschäftsprozesse zu minimieren.

*„Ein **Incident** (Störung) ist ein Ereignis, das nicht zum standardmäßigen Betrieb eines Service gehört und das tatsächlich oder potenziell eine Unterbrechung oder Minderung der Service-Qualität verursacht.“ [OGC 00]*

Neben diesem Verständnis für einen Incident werden in ITIL aber auch z.B. Service-Anfragen von Anwendern zu den Incidents gezählt, die ja nicht im eigentlichen Sinne Störungen sind. Um eine Trennung zwischen diesen verschiedenen Incident-Arten vorzunehmen, die sich auch in deren Behandlung innerhalb des Incident-Management-Prozesses widerspiegelt, wird im Rahmen dieser Arbeit ein *Incident* entsprechend der angegeben Definition verstanden. Anfragen von Anwendern werden zukünftig unter dem Begriff *User Service Request* geführt.

*„Ein **User Service Request** ist die Anfrage eines Anwenders zur Unterstützung, Service-Erweiterung, Lieferung, Information, zum Rat oder Dokumentation.“ [ITS 02]*

In der ITIL ist der gerade eingeführte Begriff eines „User Service Request“ unter der Bezeichnung „Service Request“ zu finden. Diese namentliche Anpassung wurde vorgenommen, um Verwechslungen mit den im Kontext einer SOA definierten Begriff eines „Service Requests“ zu vermeiden.

Im Rahmen des Incident Managements spielen *Workarounds (Behelfs- oder Übergangslösungen)* eine wichtige Rolle. Damit werden zwar das Symptom des Incidents beseitigt, aber nicht die Ursache der Störung. Dies ist Aufgabe des Problem Management (siehe Abschnitt 3.2.1.3). Da das zu entwickelnde Konzept den Incident-Management-Prozess unterstützen soll, wird dieser nochmals genauer im Kapitel 4 betrachtet.

3.2.1.2. Service Desk

Wie bereits erwähnt ist der Service Desk kein eigener Support-Prozess sondern eine „Function“ oder organisatorische Einheit, welche meist den First-Level-Support im Incident-Management-Prozess realisiert. Er fungiert als zentrale Anlaufstelle (Single Point of Contact) und stellt damit die Schnittstelle zwischen den Anwendern und dem IT Service Management dar.

Unter einem Anwender versteht man eine Person, die einen vereinbarten Service in Anspruch nimmt. Im Gegensatz dazu hat ein Kunde eine vertragliche Vereinbarung mit dem Service Provider. Kunden müssen nicht zwingend Personen sein, sondern können auch andere andere Organisationen sein. In diesem Fall wäre dann der Kunde die Firma und ein Anwender ein Mitarbeiter dieser Firma. Ebenso müssen Anwender und Kunde nicht zwingend identisch sein, wie dieses Beispiel zeigt. Die Unterscheidung zwischen der Rolle des Anwenders und der des Kunden wird häufig vorgenommen, da der Anwender immer über den Service Desk Kontakt mit dem Service Provider aufnehmen sollte. Der Ansprechpartner für den Kunden hingegen ist in der Regel das Customer Relationship Management bzw. in ITIL das Service Level Management.

Wenn man die Literatur studiert, findet man neben dem Begriff Service Desk häufig auch noch andere Bezeichnungen wie z.B. Help Desk. Einige betrachten auch ein Call Center als eine Art Service Desk. Dennoch existieren Unterschiede. In Anlehnung an die IT Infrastructure Library grenzen wir die Begriffe Call Center, Help Desk und Service Desk wie folgt voneinander ab [Bren 02]:

- Call Center
  Organisationseinheit, deren Aufgabe im Abarbeiten von telefonbasierten Routine-Transaktionen besteht. Im Umfeld des Anwender-Supports sprechen wir von einem Call Center als der Abteilung oder Einheit, die von Anwendern eingehende Störungsmeldungen aufnimmt und gegebenenfalls weiterleitet.

- Help Desk
  Zusätzlich zu den Aufgaben eines Call Centers übernimmt ein Help Desk auch selbst die Behandlung von Incidents.

- Service Desk
  Übernimmt die Aufgaben eines Help Desk, bietet aber darüber hinaus eine Schnittstelle für Dienstanforderungen und Routine-Änderungswünsche (z.B. Einrichtung oder Änderung einer Kennung) des Anwenders.

3.2.1.3. Problem Management

Das Problem Management befasst sich im Gegensatz zum Incident Management mit der Ursachenanalyse. Dabei spielen die gerade aktuellen Störungen eine wichtige Rolle. Allerdings befasst sich das Problem Management auch mit möglichen Schwachstellen in der IT-Infrastruktur, um das Eintreten von Störungen bereits im Vorfeld zu verhindern. Dies bedeutet, dass das Problem Management im Gegensatz zum Incident Management, das eher einen reaktiven Charakter hat, sowohl reaktiv als auch proaktiv ist.

Folgende Begriffe werden im Problem Management unterschieden:

*„Ein **Problem** beschreibt eine unerwünschte Situation, hinweisend auf die noch unbekannte Ursache einer oder mehrerer (potentieller) Störungen.“ [OGC 00]*

*„Ein **Known Error** (bekannter Fehler) beschreibt ein Problem, dessen Ursache erfolgreich festgestellt wurde und für den es einen Workaround gibt.“ [OGC 00]*

Workarounds zu den bekannten Fehlern werden unter anderem dem Incident Management in Form der Known-Error-Datenbank (KDB) zur Verfügung gestellt. Ist die Ursache für ein Problem festgestellt und eine mögliche Lösung bekannt, so kann ein Request for Change (RFC) gestellt werden. Auch das Verfolgen und die Überwachung bekannter Fehler in der Infrastruktur fällt in diesen Aufgabenbereich. Ergebnis eines guten Problem Managements ist eine bessere Qualität und Beherrschung der IT-Services und eine höhere Erfolgsquote im First-Level-Support.

3.2.1.4. Configuration Management

Das primäre Ziel des Configuration Management ist es, gesicherte, genaue und aktuelle Informationen über die IT-Infrastruktur jederzeit den anderen Managementbereichen zur Verfügung zu stellen. Ein wichtiges Hilfsmittel dafür ist die Configuration Management Database (CMDB), welche sämtliche IT-Betriebsmittel in Form so genannter Configuration Items (CIs) und deren Beziehungen untereinander beinhaltet.

Um sicherstellen zu können, dass die in der CMDB enthaltenen Informationen immer aktuell und korrekt sind, überwacht das Configuration Management die internen Zusammenhänge zwischen den CIs sowie deren Standardisierung, Zulassung und Änderungen. Durch die Bereitstellung dieser Informationen unterstützt das Configuration Management alle anderen Service-Management-Prozesse.

3.2.1.5. Change Management

Aufgabe des Change Management ist es, mittels standardisierter Methoden und Verfahren Änderungen (Changes) möglichst schnell durchzuführen, ohne dass dadurch die Qualität der IT-Services zu leiden hat. Dabei sollten Änderungen auch gleichzeitig mit Verbesserungen einhergehen, obwohl dies nicht zwingend der Fall sein muss.

*„Ein **Request for Change** (RFC) kann ein Erneuerungs- oder Verbesserungsvorschlag oder auch eine Korrektur sein.“ [ITS 02]*

Die erhaltenen RFCs müssen geprüft, klassifiziert und genehmigt werden, bevor ein Plan für die Änderung erstellt werden kann. Nach koordinierter Erstellung, Test und Implementierung der Änderung ist die Durchführung einer Evaluation in Form von Reviews zwingend notwendig. Je nach Art des RFCs sind einzelne Aspekte stärker ausgeprägt. Beispielsweise können Standard-Changes, die schon vollständig beschrieben und standardisiert sind, bereits im Vorfeld autorisiert werden. Im Gegensatz dazu kann es für einen Request for Change unter Umständen notwendig sein, das Change Advisory Board (CAB) einzuberufen. In welche Klasse ein RFC fällt, hängt von der Priorität und der Kategorisierung ab.

Prioritäten und Kategorien, die im Rahmen des Change Management vergeben werden, sollen an dieser Stelle nicht weiter vertieft werden, da dies nicht Aufgabe dieser Arbeit ist. Im Rahmen des Incident Managements tauchen diese Begriffe erneut auf, allerdings hat diese Einstufung nicht unmittelbar etwas mit jener innerhalb des Change-Management-Prozesses zu tun. Im Kapitel 4.3.2 werden die Priorität und die Kategorisierung bezogen auf das Incident Management vorgestellt.

3.2.1.6. Release Management

Oberstes Ziel des Release Managements ist der Schutz der produktiven Systeme. Mittels formaler Verfahren und Kontrollen soll bei der Implementierung und Zusammenstellung neuer Versionen die Service-Qualität nicht negativ beeinträchtigt werden. Im Vergleich zum Change Management konzentriert sich das Release Management eher auf die Durchführung als auf die Kontrolle.

*„Ein **Release** ist eine Reihe neuer oder geänderter Konfigurations-Elemente (Configuration Items), die zusammenhängend getestet und in die Produktionsumgebung überführt werden.“[ITS 02]*

Aufgaben des Release Managements sind unter anderem das Planen, der Entwurf und die Zusammenstellung eines Release. Daran schließt sich eine Testphase an, die je nach Umfang des Release variieren kann. Auch das Erstellen so genannter Rollout- und Rollback-Pläne, die Kommunikation der Veränderungen, ggf. die Durchführung von Schulungen sowie die Verteilung und Installation der Releases fallen in den Aufgabenbereich des Release Managements.

In diesem Abschnitt wurden die Service-Support-Prozesse vorgestellt, zu denen auch der Incident-Management-Prozess zählt. Daneben existieren wie bereits aus Abbildung 3.2 ersichtlich, durchaus Abhängigkeiten zwischen den Support- und den Delivery-Prozessen, weswegen im Folgenden die Prozesse des Service Delivery betrachtet werden.

3.2.2. Service Delivery

Dieser Abschnitt wird sich mit den Service-Delivery-Prozessen befassen. Da der Fokus dieser Arbeit auf dem Service Support, genauer dem Incident-Management-Prozess ruht, werden die verschiedenen Service-Delivery-Prozesse, die sich mit der langfristigen Planung und Optimierung der IT-Dienstleistungen befassen, nur kurz charakterisiert. Für ein ausführlicheres Studium sind [OGC 01] und [ITS 02] zu empfehlen.

3.2.2.1. Service Level Management

Primäre Aufgabe des Service Level Managements ist die Pflege und ständige Verbesserung der mit dem Kunden vereinbarten IT-Services. Zu diesem Zweck trifft das Service Level Management Vereinbarungen hinsichtlich der Leistungen der IT-Organisation, überwacht und dokumentiert diese.

„Ein ***Service Level Agreement*** *(SLA) ist ein schriftlicher Vertrag zwischen einem Service Provider und einem Kunden hinsichtlich der zu leistenden Services. In diesen werden die zentralen Serviceziele und Verantwortlichkeiten beider Parteien definiert, aber auch die Qualität und die Kosten.“ [OGC 01]*

Vorteil eines effektiven Service Level Management ist die Verbesserung der Servicequalität. Durch die ständigen Anpassungen der Service Level Agreements an die sich ändernden Kundenbedürfnisse und -wünsche kann die Zufriedenheit der Kunden und ihre Beziehung zum Service Provider nachhaltig verbessert werden. Neben den Service Level Agreements, welche Vereinbarungen mit externen Kunden betreffen, existieren auch so genannte Operational Level Agreements und Underpinning Contracts.

„Ein ***Operational Level Agreement*** *(OLA) ist eine Vereinbarung mit einer internen IT-Abteilung. Diese enthält Absprachen über die Erstellung von (Teil-)Services in diesem Bereich.“[ITS 02]*

„Ein ***Underpinning Contract*** *(UC) ist eine Vereinbarung mit einem externen Service Provider zur Erbringung von benötigten (Teil-)Services.“*

Zur Veranschaulichung der eben vorgestellten Vereinbarungen betrachte man Abbildung 3.3. Die Forderungen des Kunden in Bezug auf die zu erbringenden Services werden Service Level Requirements genannt und fließen neben den im Servicekatalog beschriebenen Services in die Service Level Agreements ein. Die Service Specsheets beinhalten die technische Umsetzung der in den SLAs getroffenen Vereinbarungen zwischen Kunde und IT-Organisation. Wird aus Sicht der IT-Organisation nicht mit dem Kunden, sondern mit einer internen oder externen IT-Organisation eine Vereinbarung über die Erbringung eines (Teil-)Services getroffen, so spricht man nicht von Service Level Agreements, sondern von Operational Level Agreements im Falle einer internen, und von Underpinning Contracts im Falle einer externen IT-Organisation. Wie in der Abbildung dargestellt, können auch interne IT-Organisationen ihrerseits Underpinning Contracts mit externen IT-Organisationen abschließen.

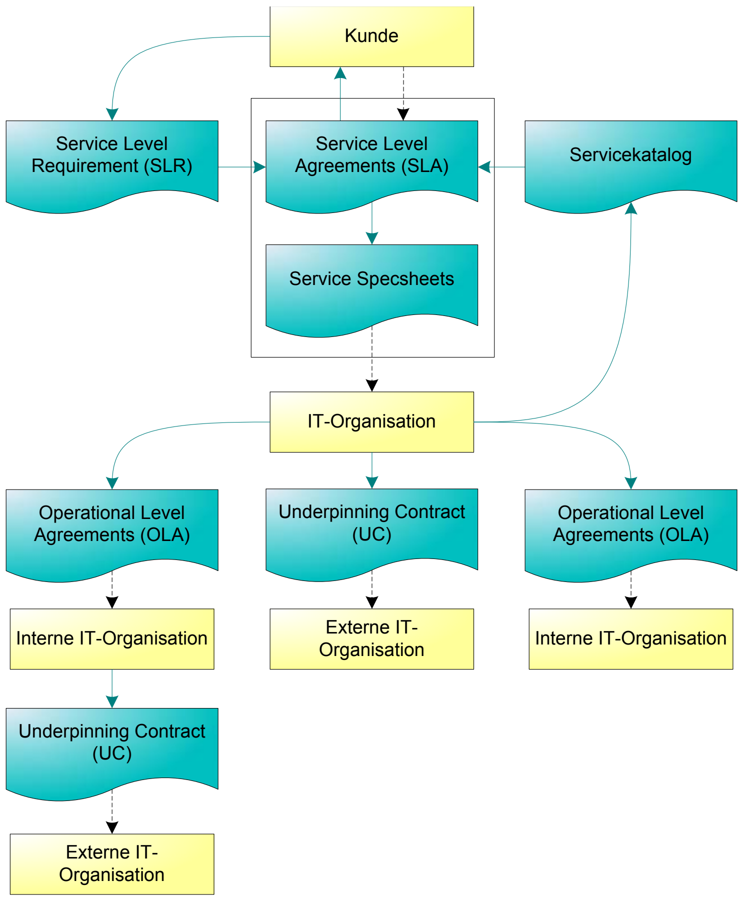


Abbildung 3.3.: Grundbegriffe des Service Level Management (Quelle: [Muni 05])

3.2.2.2. Financial Management for IT Services

Das Financial Management for IT Services umfasst die Bereiche Budgeting (Finanzplanung), Accounting (Kostenrechnung) und Charging (Leistungsverrechnung) von IT-Services. In diesem Rahmen stellt es Management Informationen zur Verfügung, womit eine effiziente, wirtschaftliche und kostenwirksame Erbringung der Services gewährleistet werden soll. Dies ermöglicht der Organisation, die Kosten zu ermitteln und den angebotenen Services zuzuordnen.

3.2.2.3. Capacity Management

Das Capacity Management beschäftigt sich mit der rechtzeitigen und kosteneffizienten Bereitstellung ausreichender IT-Kapazität, wobei sowohl die aktuellen als auch die zukünftigen Kapazitäts- und Performanceaspekte zu berücksichtigen sind. Wichtig ist hierbei auch, ständig die richtigen Kapazitäten an IT-Mitteln entsprechend den bestehenden und den zukünftigen Bedürfnissen des Kunden zu kennen.

Dabei ist zu beachten, dass verschiedene Kapazitäten eines Services gut aufeinander abgestimmt werden, damit teuere Investitionen in bestimmte Komponenten auch ihre volle Rendite bringen. Beispielsweise bedeutet ein Data Center mit tausenden von Servern, das aber nur zu 30 oder 40 % ausgelastet ist, eine Verschwendung enormer Geldmengen.

3.2.2.4. IT Service Continuity Management

Innerhalb einer Organisation existieren zwei ITIL-Arbeitsbereiche, die sich mit dem Continuity Management befassen, nämlich das Business Continuity Management (BCM) und das IT Service Continuity Management (ITSCM). Das ITSCM, welches ein Bestandteil des übergeordneten Business Continuity Management ist, befasst sich mit Plänen bzgl. eines Katastrophenfalls innerhalb der IT-Services. Bei einer Katastrophe, z.B. einem Brand, muss sichergestellt werden, dass die IT-Infrastruktur und damit die darauf aufbauenden IT-Services möglichst rasch wiederhergestellt werden. Damit unterstützt das ITSCM auch alle Business-Continuity-Management-Prozesse. Durch die erstellten Notfallpläne soll das Restrisiko auf ein akzeptables Maß reduziert werden.

3.2.2.5. Availability Management

Das Ziel des Availability Managements ist es, das Leistungsvermögen der IT-Infrastruktur, der Services und der unterstützenden Prozesse zu verbessern, um ein kosteneffizientes und festgelegtes Verfügbarkeitsniveau zu gewährleisten, mit dessen Hilfe das Unternehmen in der Lage ist, seine Ziele zu verwirklichen. Aspekte, die dabei betrachten werden müssen, sind die Verfügbarkeit (Availability), die Zuverlässigkeit (Reliability), die Wartbarkeit (Maintainability) und die Servicefähigkeit (Serviceability).

3.2.3. Zusammenfassung

Wie aus den Beschreibungen der Service-Delivery-Prozesse deutlich wird, sind die Abhängigkeiten zum Incident-Management-Prozess auf den ersten Blick nicht so ausgeprägt wie zu den anderen Service-Support-Prozessen. Dennoch sollten sie nicht vollständig außer Acht gelassen werden um ihre Funktion zu verstehen, weshalb im Abschnitt 3.2 (Service Management) auch die Service-Delivery-Prozesse kurz vorgestellt wurden.

Die im Service Level Management getroffenen Vereinbarungen stellen die Grundlage für die im Incident Management zur Bearbeitung der Störungen notwendigen Rahmeninformationen dar. Sie haben Einfluss auf die Zeitfenster für eine Behebung des Incidents oder auch dessen Priorität. Notfallpläne, welche im Katastrophenfall sicherstellen sollen, dass die entsprechenden IT-Services so schnell wie möglich wieder hergestellt werden, haben ebenfalls Einfluss auf die Bearbeitung innerhalb des Incident-Management-Prozesses, woraus sich eine Abhängigkeit zum Continuity Management ergibt. Das Financial, das Capacity und Availability Management beziehen Input in Form der im Incident-Management-Prozess gesammelten Daten z.B. über die Art und die Anzahl der bearbeiteten Störungen sowie die Bearbeitungszeiten in den einzelnen Support-Leveln. Diese werden verwendet, um den Incident-Management-Prozess kosteneffizienter zu gestalten, ihm ausreichend Kapazitäten zur Verfügung zu stellen oder die mit den Kunden getroffenen Vereinbarungen bezüglich Zuverlässigkeit oder Verfügbarkeit in Kooperation mit dem Service Level Management anzupassen.

Neben dem vorgestelltem Service Management existieren noch weitere Bereiche der ITIL, und zwar das Security Management, das ICT Infrastructure Management und das Application Management sowie das „Business“, auf die nicht weiter eingegangen wird.

In diesem Kapitel wurde dem Leser zuerst ein Überblick über die Struktur der ITIL vermittelt. Anschließend wurden deren „Core Titles“, die Prozesse des Service Supports und des Service Delivery vorgestellt. Der Schwerpunkt lag dabei auf den Service-Support-Prozessen, zu denen auch der Incident-Management-Prozess zählt. Da im Rahmen dieser Arbeit ein Konzept zur Unterstützung des Incident Managements entwickelt werden soll, wird dieser Prozess nochmals aufgegriffen und im nächsten Kapitel eingehender betrachtet. Dazu wird der Prozess in einzelne Subprozesse aufgeteilt, welche sich ihrerseits aus einer Reihe von Aktivitäten zusammensetzen. Anhand dieser Subprozesse und deren Aktivitäten sollen Schnittstellen identifiziert werden, die dann im Rahmen des zu entwickelnden SOA-basierten Konzepts mittels WSDL beschrieben werden.

Kapitel 4.

Incident Management

In den bisherigen Kapiteln wurde bereits kurz das Prinzip von SOA und die ITIL mit ihren Service-Management-Bereichen vorgestellt. Dieses Kapitel wird den Incident-Management-Prozess nochmals aufgreifen und vertiefen. Wie alle Prozessbeschreibungen der ITIL weist auch der Incident-Management-Prozess teilweise Lücken in seiner Beschreibung auf, da diese eher als Empfehlungen zur Prozessstrukturierung denn als strikte Norm zur Implementierung zu verstehen sind.

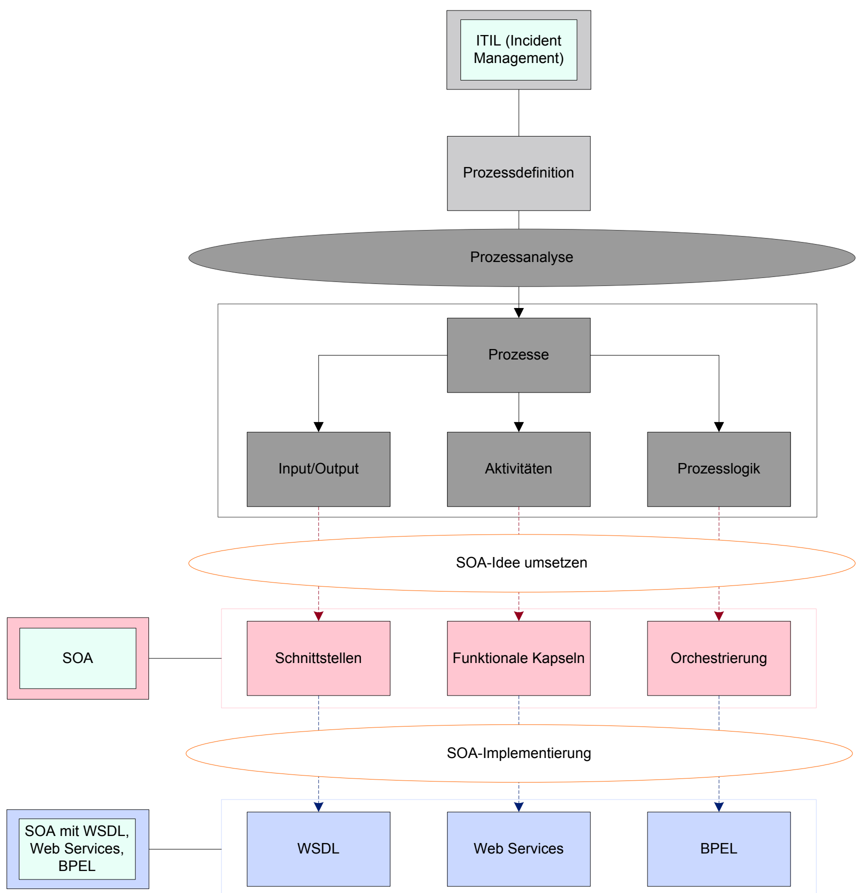


Abbildung 4.1.: Vorgehensweise zur Prozessanalyse im Incident Management

Ziel dieser Prozessanalyse, welche die erste Aktivitätsphase in Abbildung 4.1 darstellt, ist deshalb die Identifizierung und Beschreibung der Basis für die angestrebte Konzept-Implementierung. Diese Basis bilden die Input- und Outputdaten, die Aktivitäten sowie die Prozesslogik des Prozesses.

Der erste Abschnitt dieses Kapitels wird sich noch relativ allgemein mit dem Umfeld des Incident-Prozesses befassen. Im zweiten und dritten Abschnitt wird dann der Incident-Management-Prozess in seine Subprozesse und diese dann weiter in ihre Aktivitäten aufgespalten.

4.1. Prozessumfeld

Dieser Abschnitt befasst sich mit dem Umfeld des Incident-Management-Prozesses. Dabei soll vorrangig auf die einzelnen Inputs und Outputs sowie die Schnittstellen zu anderen Support-Prozessen eingegangen werden, weshalb der Prozess an sich erst einmal als Blackbox betrachtet wird. Im Abschnitt 4.2 werden dann die einzelnen Subprozesse vorgestellt.

Abbildung 4.2 zeigt das Prozessumfeld des Incident-Management-Prozesses. Dieser bezieht vom Service Desk, der Systemverwaltung, dem Netzwerk, den Verfahren (beispielsweise etablierte Vorgehensweise oder Bearbeitungsschritte) etc. Störungsmeldungen als Input, welche dann, wie in Abbildung 4.3 beschrieben, bearbeitet werden. Die Bearbeitung von User Service Requests, wie sie in Abschnitt 3.2.1.1 definiert werden, weicht von der Behandlung von Incidents ab, weshalb sie eine Art Sonderstellung einnehmen und separat aufgeführt sind. Weiteren Input bezieht das Incident Management vom Configuration Management, welches für die CMDB verantwortlich ist und benötigte Daten über CIs bereitstellt.

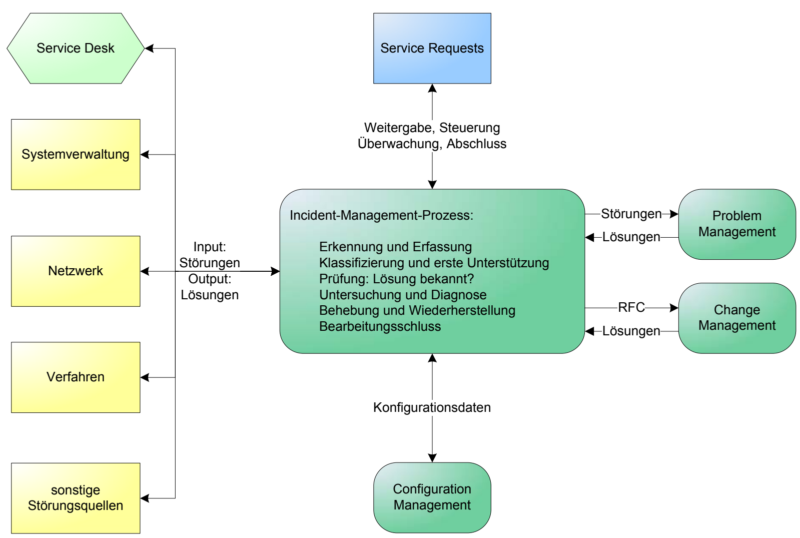


Abbildung 4.2.: Incident-Prozess-Umfeld (vgl. [OGC 00])

Ziel des Incident-Management-Prozesses ist primär die Behebung des Incidents. Eine genauere Ursachenanalyse, insbesondere bei häufiger auftretenden Fehlern obliegt dem Problem Management. Dieser Prozess, welcher eher parallel zum Incident-Management-Prozess arbeitet als diesem nachgeschaltet ist, stellt dem Incident Management u.a. Wissen über Known Errors und deren Workarounds in Form der KDB zur Verfügung. Ist eine Lösung für ein Incident gefunden und bedarf es z.B. einem Hardware-Austausch, so wird beim Change-Management-Prozess ein RFC eingereicht, der dann von diesem genehmigt werden muss und in Kooperation mit dem Release Management durchgeführt wird.

Nach dieser kurzen Einordnung des Incident-Management-Prozesses in die Support-Umgebung soll im nächsten Abschnitt die Prozessstruktur genauer betrachtet werden. Die in der Abbildung 4.2 dargestellten Inputs, Outputs und Schnittstellen werden im Rahmen der einzelnen Subprozesse nochmals aufgegriffen, weswegen sie an dieser Stelle nicht weiter vertieft werden.

4.2. Der Incident-Management-Prozess

Dieser Abschnitt betrachtet zuerst den allgemeinen Ablauf des Incident-Management-Prozesses befassen. Anschließend werden die Subprozesse jeweils in einem eigenen Abschnitt mitsamt ihren Aktivitäten, Inputs, Outputs und Schnittstellen vertiefend behandelt.

Die grobe Struktur des Incident-Management-Prozesses wird in Abbildung 4.3 gezeigt. Ausgangspunkt für die Bearbeitung einer Störungsmeldung ist deren Annahme und Erfassung. Anschließend erfolgt eine Klassifizierung. Falls es sich um einen User Service Request handelt, so werden spezielle Bearbeitungsschritte angestoßen, andernfalls wird das Störmuster analysiert, um eventuelle Ähnlichkeiten oder Zusammenhänge zu anderen Störungen festzustellen. Ist eine Lösung für die Störung bekannt, so kann die Analyse- und Diagnosephase übersprungen und die Störung gleich an die Wiederherstellung übergeben werden. Kann die Störung in dieser Phase behoben werden, so kann sie abgeschlossen werden. Andernfalls erfolgt eine erneute Bearbeitung.

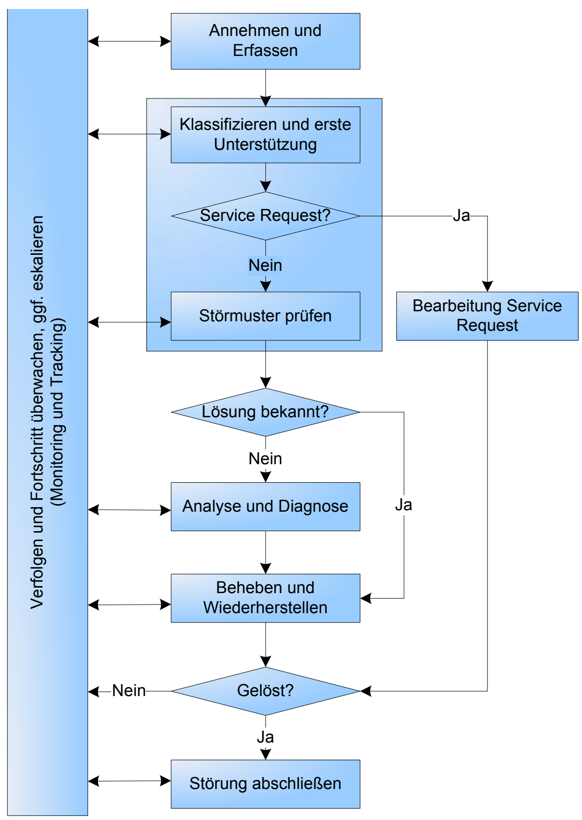


Abbildung 4.3.: Incident-Prozess (vgl. [OGC 00, ITS 02])

Der in Abbildung 4.3 dargestellte Incident-Prozess ist nicht, wie es vielleicht auf den ersten Blick scheint, ein rein sequentieller Prozess, sondern entspricht phasenweise eher einem Zyklus, da die Analyse und Diagnose ebenso wie die Behebung und Wiederherstellung auf jedem Support-Level wiederholt werden, wenn keine Lösung für den Incident auf dem vorherigen Level gefunden wurde. Dies ist u.a. in Abbildung 4.4 zu sehen. Im Rahmen der Abbildung 4.13 im Subprozess „Analyse und Diagnose“ (Abschnitt 4.3.3) wird diese Rekursion noch einmal beschrieben. Auch wenn eine Lösung des Incidents gefunden wurde, heißt dies noch lange nicht, dass die Störung auch tatsächlich behoben werden kann. Vielmehr spielen noch weitere Faktoren eine Rolle, wie die ggf. notwendige Genehmigung eines RFCs oder die Zustimmung des Anwenders.

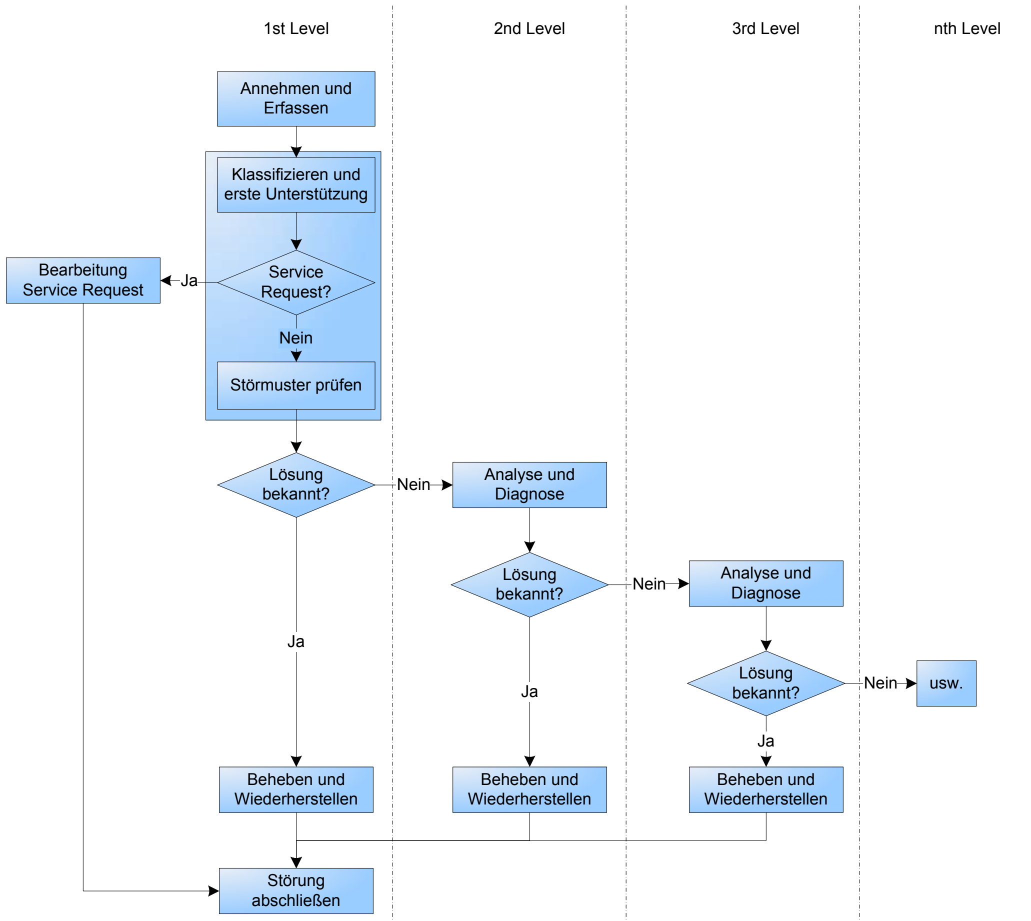


Abbildung 4.4.: Bearbeitung und Weiterleitung von Incidents (vgl. [OGC 00, ITS 02])

Der in der ITIL [OGC 00] aufgeführte Subprozess „Classification and Initial Support" umfasst die in itSMF [ITS 02] aufgeführten Aktivitäten „Klassifizierung und erste Unterstützung" und „Störmuster prüfen". Wie in Abbildung 4.3 und 4.4 gezeigt, wird zur Verdeutlichung, dass innerhalb dieses Subprozesses eine Verzweigung des Bearbeitungsablaufs stattfindet, der Subprozess in Anlehnung an itSMF [ITS 02] in diesen Abbildungen zweigeteilt dargestellt, obwohl natürlich eine Prüfung der Störmuster einer ersten Unterstützung vorausgeht.

Die in Abbildung 4.4 angedeutete Support-Hierarchie orientiert sich an den organisatorischen Erfordernissen. Für eine effiziente Bearbeitung der Incidents sollten das Know-how, die Erfahrungen und die Befugnisse in der jeweiligen Supportgruppe von Support-Level zu Support-Level ansteigen. Der First-Level-Support wird meist durch den Service Desk realisiert.

Nachdem das Prozessumfeld sowie die grobe Struktur des Incident-Management-Prozesses veranschaulicht wurden, widmet sich der folgende Abschnitt der Analyse der Subprozesse des Incident Managements und deren Aktivitäten.

4.3. Analyse der Subprozesse

In diesem Abschnitt werden die Subprozesse des Incident Management analysiert. Eine Analyse ist insofern notwendig, da die Beschreibung der Subprozesse nach ITIL [OGC 00] für eine weitere Betrachtung, wie es im Rahmen dieser Arbeit angestrebt wird, aufgrund der hohen Abstraktion nicht ausreicht. Dazu werden die im vorherigen Abschnitt aufgezeigten Subprozesse des ITIL Incident-Management-Prozesses aufgegriffen und einzeln analysiert. Ziel dieser Analyse ist die Herausarbeitung der Aktivitäten, der damit verbundenen Prozesslogik und die zur Bearbeitung einer Störungsmeldung notwendigen Input- und Outputdaten, welche den Ausgangspunkt für das zu entwickelnde SOA-basierte Konzept darstellen.

Folgende Subprozesse des Incident Managements werden betrachtet:

- Annehmen und Erfassen
- Klassifizieren und erste Unterstützung (inklusive Störmuster prüfen)
- Analyse und Diagnose
- Beheben und Wiederherstellen
- Störung abschließen
- Monitoring und Tracking

Jeder dieser Subprozesse besteht seinerseits wieder aus verschiedenen Aktivitäten, die sowohl einfacher als auch komplexer Natur sein können. Je nach Komplexität können sie auch durch ein Tool unterstützt werden. Besonderes Augenmerk soll dabei auf den Aktivitäten sowie den Input- und Outputdaten liegen, da diese später noch einmal aufgegriffen werden und in die WSDL-Schnittstellenbeschreibungen einfließen sollen.

Die Aktivitäten und die Prozesslogik werden mittels *eEPKs (erweiterte Ereignisgesteuerte Prozessketten)* modelliert. Die grundlegenden Elemente einer Ereignisgesteuerten Prozesskette sind Funktionen und Ereignisse sowie Kanten und Verknüpfungsoperatoren. Funktionen und Ereignisse werden abwechselnd über Kanten miteinander verknüpft. Erzeugt eine Funktion mehr als ein Ereignis oder lösen mehrere Ereignisse eine Funktion aus, so werden diese Sachverhalte mittels Verknüpfungsoperatoren dargestellt (siehe Anhang A und [Sine 03]).

Des Weiteren werden jedem Subprozess des Incident Managements in Tabellenform notwenige Input- und Outputdaten zugeordnet, welchen dann später aufgrund den innerhalb von ITIL und innerhalb des betrachteten Unternehmens definierten Verantwortlichkeiten Prozesse zugeordnet werden, die Schnittstellen zum Incident-Management-Prozess besitzen.

4.3.1. Annehmen und Erfassen

Alle Incidents und User Service Requests müssen registriert werden. Das Annehmen und Erfassen von Incidents und User Service Requests, die durch Anwender gemeldet werden, erfolgt in der Regel durch den Service Desk. Incidents können aber auch durch Managementsysteme, z.B. von Hard- und Softwarekomponenten, gemeldet werden. In diesem Fall wäre eine Schnittstelle sinnvoll, über die diese Systeme die zu erstellenden *Incident Records* (IRs) automatisch in eine Incident-Datenbank eintragen können. Eine weitere Möglichkeit, Incidents und User Service Requests zu erfassen, wäre ein Web-Portal, welches die Anwender nutzen können, um direkt Incident Records zu erzeugen.

Aktivitäten dieses Subprozesses sind (vgl. [Pleg 05, ITS 02]):

- Incident Record anlegen (Meldungsrahmen)
- Zuweisen einer eindeutigen Störnummer (Identifikationsnummer)
- Störungsaufnahme (Störungsbeschreibung, Zeitpunkt, Melder, Kontaktdaten, ...)

- Meldungsart festlegen (Incident oder User Service Request)
- Kontaktmedium festlegen
- Ergänzung der Störungsdaten (CIs, Services, ...)
- ggf. Warnung auslösen

Neben diesen aufgelisteten Aktivitäten zeigen die Abbildungen 4.5 bis 4.7 eine mögliche Verfeinerung der Logik dieses Subprozesses. Wie in Abbildung 4.5 dargestellt, kann dieser Subprozess durch verschiedene Ereignisse angestoßen werden. Beispielsweise kann ein Anwender telefonisch eine Störung melden oder ein Fax bzw. eine E-Mail an den Service Desk schicken. Diese möglichen auslösenden Ereignisse werden durch das Unternehmen bestimmt. Falls dem Anwender ein Web-Portal bereitgestellt wird oder Managementsyteme angebunden werden, so können auch diese eine Störungsbehandlung auslösen. Die durch eines dieser Ereignisse ausgelöste Funktion „Formular anfordern“ steht für die im Unternehmen für das jeweilige Ereignis erste auszuführende Aktion, welche zur Erfassung und Sicherung der Meldungsdaten notwendig ist. Im Fall des berühmten „Karteikartensystems“ stünde diese Funktion für das Hernehmen einer leeren Karteikarte. Wird bspw. ein Ticketsystem oder eine graphische Anwendung für eine Datenbank verwendet, so kann sich dahinter eventuell wirklich das Öffnen eines neuen Formulars verbergen. Das Formular, welches der Service Desk anfordert, könnte u.U. von dem abweichen, welches der Anwender über das Web-Portal auszufüllen hat.

Diese möglicherweise unterschiedliche Behandlung entsprechend dem auslösenden Ereignis soll durch das mehrfache Aufführen der Funktion „Formular anfordern“ in Abbildung 4.5 angedeutet werden. Damit das Formular und damit auch die Meldung sowie die dazugehörenden Daten jederzeit eindeutig identifiziert werden können, wird dem Meldungsrahmen eine Identifikationsnummer zugewiesen. Nach Abschluss der in Abbildung 4.5 aufgeführten Aktivitäten erhält man bezogen auf das „Karteikartensystem“ eine zwar noch leere, aber bereits eindeutig identifizierbare Karteikarte, welche im Laufe des weiteren Prozesses mit Daten gefüllt werden muss.

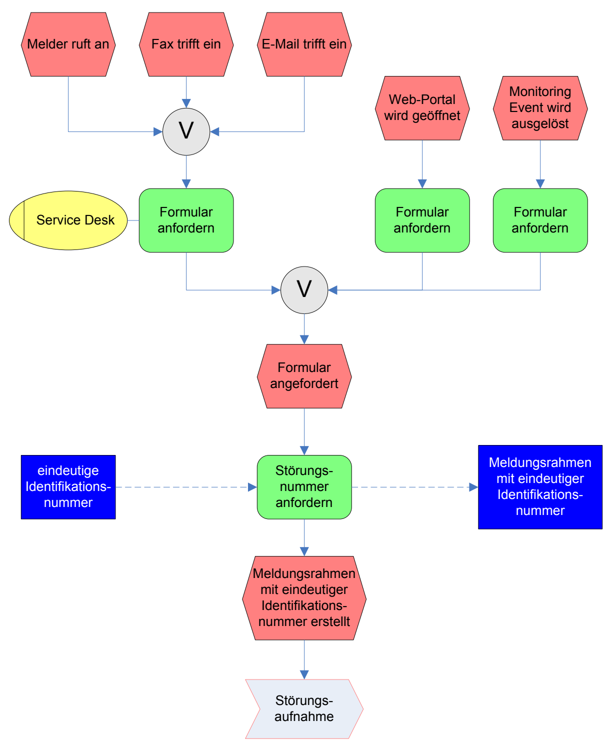


Abbildung 4.5.: Störungsrahmen anlegen

Abbildung 4.6 nimmt diesen noch leeren Rahmen und beginnt, ihn mit Daten zu füllen. Zuerst werden die Störungsbeschreibung sowie die Melderdaten ergänzt. Die grundsätzliche Meldungsart (Incident oder User Service Request) sowie das Kontaktmedium müssen ebenfalls angegeben werden, wobei die Meldungsart im Rahmen des zweiten Subprozesses noch einmal geprüft und ggf. konkretisiert oder korrigiert wird. Die Speicherung des Kontaktmediums ist für spätere Kontaktaufnahme mit dem Melder notwendig. Außerdem können fehlende Daten im Gespräch einfach und schnell erfragt werden, wohingegen bei einem Fax erst versucht werden muss, mit dem Melder (ggf. wiederum durch ein Fax) in Kontakt zu treten.

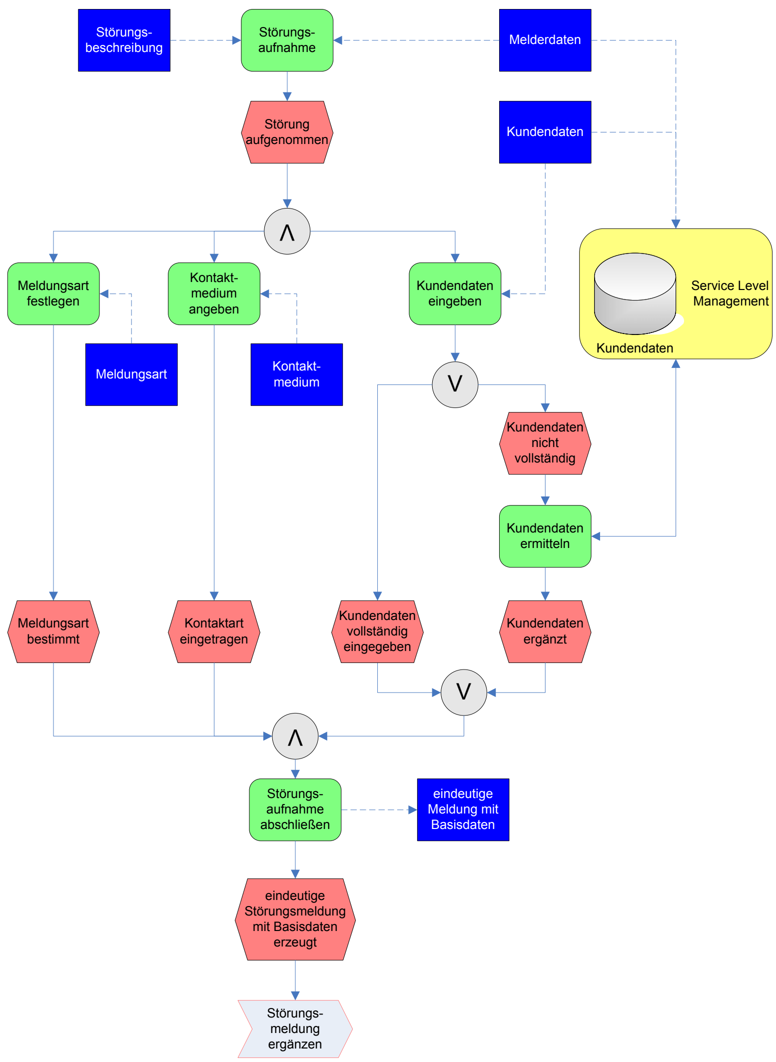


Abbildung 4.6.: Störungsaufnahme, Meldungsart und Kontaktmedium festlegen

Da im Incident Management zwischen dem Anwender und dem Kunden unterschieden wird (siehe Abschnitt 3.2.1.2), sind für die Bearbeitung neben den Daten des Melders auch die des Kunden zu erfassen. Kann der Melder nicht alle notwendigen Kundendaten (z.B. die Kundennummer) liefern, so müssen diese vom Service Level Management erfragt werden, bevor die erste Störungsaufnahme abgeschlossen werden kann. Ergebnis der in Abbildung 4.6 dargestellten Aktivitäten ist ein mit den Ausgangsdaten für die weitere Störungsbehandlung ausgefüllter Meldungsrahmen. Der Meldungsrahmen wird auch als Incident Record bezeichnet, auch wenn der bisherige Meldungsrahmen ebenso für einen User Service Request stehen kann, welcher in ITIL von einem Incident unterschieden wird[1]. Im Folgenden wird auch im Rahmen dieser Arbeit der Begriff „Incident Record“ verwendet, da die Behandlung der Incidents im Fokus dieser Analyse steht.

Der erzeugte Incident Record oder ein Teil davon steht innerhalb des Incident-Management-Prozesses jeder beteiligten Funktion zur Verfügung und wird aus diesem Grunde nicht immer explizit als Input oder Output angegeben. Vielmehr werden nur diejenigen Daten als Informationen innerhalb der EPKs modelliert, welche für die betreffende Funktion von Bedeutung sind.

Ausgehend von den bisher vorhandenen Basisdaten wird, wie in Abbildung 4.7 gezeigt, der Incident Record ergänzt. Zuerst werden dazu vom Configuration Management Daten aus der CMDB zu den Configuration Items und Diensten des Kunden bzw. Anwenders erfragt. Die so gewonnen Daten der CIs müssen nun mit den Angaben des Melders verglichen werden, da mögliche Abweichungen der Konfiguration sowie daraus abgeleitete, eventuell falsche Annahmen die Behebung der Störung beeinflussen können. Über die Kundendaten, den Meldezeitpunkt sowie die Services, welche mit dem Kunden vereinbart wurden oder die von der Störung betroffen sind, werden vom Service Level Management ausgehend von den betroffenen Service Level Agreements ermittelt, z.B. die einzuhaltenden Zeitfenster für die Bearbeitung der Meldung. Basierend auf den Zeitfenstern, der aktuellen Auslastung sowie den unternehmensspezifischen Vorgaben des Incident-Management-Prozesses kann dann geprüft werden, ob bereits eine Warnmeldung ausgelöst werden muss. Eine Warnmeldung müsste bspw. ausgelöst werden, falls die in den SLAs vereinbarten Bearbeitungszeiten bereits kurz vor dem Überschreiten stehen oder bereits überschritten wurden.

Tabelle 4.1 fasst die wichtigsten Input- und Outputdaten noch einmal zusammen. Basis für die Bearbeitung der Meldung sind die Details der Störung sowie die Daten des Melders und des Kunden. Daneben werden zur Ergänzung die Daten der CMDB sowie der Servicekatalog benötigt. Output sind ggf. aktualisierte Daten, das Feststellen möglicher Abweichungen in der CMDB (durch Abgleich mit dem Anwender), sowie das Auslösen einer Warnmeldung. Als weiteren Output dieses Subprozesses sieht ITIL das abschließende Benachrichtigen des Anwenders am Ende des Incident-Management-Prozesses vor. Im Idealfall sollte dies durch den Ansprechpartner auf Seiten des Service Provider erfolgen, der die Störungsmeldung angenommen hat.

Annehmen und Erfassen	
Input	Output
• Details der Störung • Melder- und Kundendaten • Konfigurationsdaten aus der CMDB • Servicekatalog	• aktualisierte Incident-Daten • festgestellte Unstimmigkeiten in der CMDB • Warnmeldung • Benachrichtigung des Melders über den Abschluss des Incidents nach dessen Behebung

Tabelle 4.1.: Annehmen und Erfassen: Input/Output

Im Rahmen von Annehmen und Erfassen schlägt ITIL [OGC 00] das automatische Generieren von Incident-Record-Rahmen in einer Datenbank durch ein Tool vor. Ein mögliches und häufig eingesetztes Tool in diesem Zusammenhang ist ein *Trouble-Ticket-System* (siehe Abschnitt 4.4).

Dieser erste Subprozess des Incident-Management-Prozesses wird in der Regel durch den Service Desk realisiert, da dieser die Schnittstelle zum Anwender darstellt und jedweder Kontakt zwischen Anwender und IT-Organisation über diesen erfolgen sollte. Wird dem Anwender neben dem Service Desk ein Web-Portal für die Meldung von Störungen bereitgestellt, so stellt der Service Desk zwar nicht mehr direkt den Single Point of Contact dar, dennoch obliegt es dem Service Desk, die durch den Anwender über das Web-Portal erstellten Incident Records oder *Trouble Tickets* zu überwachen. Dasselbe gilt auch für automatisch durch andere Managementwerkzeuge erzeugte Incident Records. Die bisherige Behandlung der Meldungen kann auch für User Service Request angewendet werden. Deswegen ist bislang keine Trennung nach Art der Meldungen notwendig. Diese Trennung wird erst zu Beginn des nächsten Subprozesses durchgeführt, in welchem die Störungsmeldungen klassifiziert werden.

[1] siehe Definitionen für „Incident“ und „User Service Request“ in Abschnitt 3.2.1.1

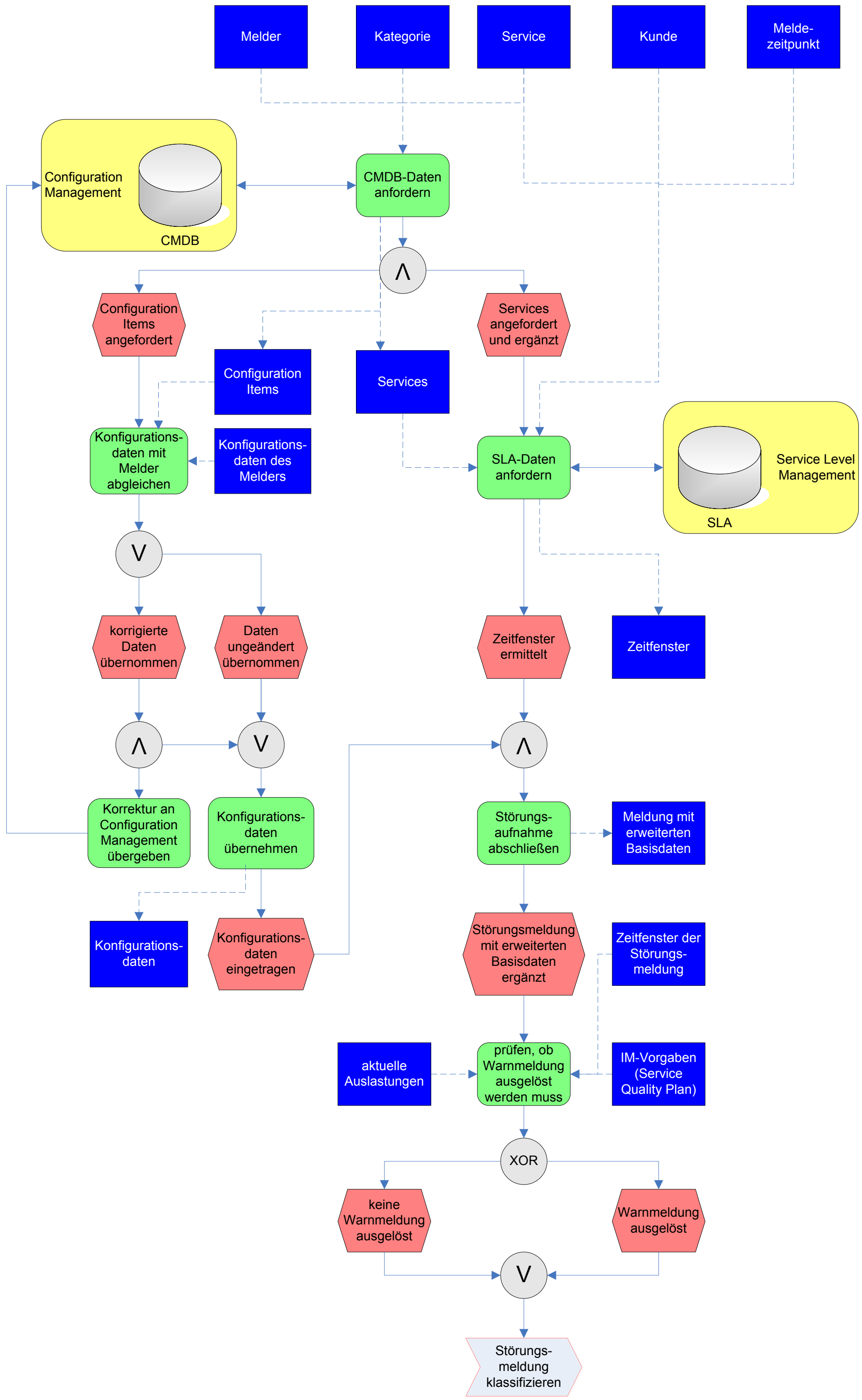


Abbildung 4.7.: Störungsmeldung ergänzen und ggf. Warnmeldung auslösen

4.3.2. Klassifizieren und erste Unterstützung

Die im ersten Subprozess erfassten Daten des Incident Records und die Konfigurationsdaten der CMDB werden in diesem Subprozess analysiert, um den Grund für die Störung festzustellen. Um eine bestmögliche Bearbeitung des Incident Records zu ermöglichen, wird dieser zuerst klassifiziert und in verschiedene Kategorien eingeteilt. Eine mögliche Kategorisierung könnte sich an den verschiedenen Supportbereichen orientieren. In diesem Fall kann man sich die Kategorien Netzwerk, Arbeitsplatz, Sicherheit, Organisation und Verfahren, User Service Requests oder auch Hard- bzw. Software vorstellen. Die Klassifizierung eines Incidents hat Auswirkung auf dessen weitere Bearbeitung. Je nach Incident-Klasse wird die Bearbeitung im Falle einer hierarchischen Eskalation[2] einer anderen Supporteinheit übertragen. Die erste Klassifizierung eines Incidents ist jedoch nicht bindend, sondern kann sich im Laufe der Bearbeitung ändern. Die am Incident Record oder Trouble Ticket vorgenommenen Änderungen werden in deren Aktions- und Zuständigkeitslisten erfasst und dokumentiert.

Aktivitäten dieses Subprozesses sind (vgl. [OGC 00, Pleg 05]):

- Störungsmeldung klassifizieren
- Feststellung von Dringlichkeit und Auswirkung und damit der Priorität
- Konfigurationsdetails und Services feststellen
- Identifikation der betroffen Supportvereinbarungen
- Störmuster prüfen
- Zuweisung einer Supportgruppe
- erste Unterstützung
- Schließen des Incidents oder Weiterbearbeitung

Die erste aufgelistete Aktivität dieses Subprozesses ist das Klassifizieren des Incident Records zu dessen besseren Bearbeitung, was in Abbildung 4.8 durch die Funktion „Störungsmeldung klassifizieren“ repräsentiert wird. Ausgehend vom Incident Record und dessen Störungsbeschreibung sowie eines Klassifizierungskataloges wird die Meldung einer bestimmten Klasse zugeordnet. Die Meldungsart, welche während der Annahme ermittelt wurde, kann dabei bestätigt, abgeändert oder konkretisiert werden. Neben der Klasse der „User Service Requests“ werden die anderen Störungen nochmals in „Incidents“ und „Standard-Incidents“ unterschieden. Eine häufig vorkommende Störung, wie das Zurücksetzen eines vergessenen Passworts kann bspw. als Standard-Incident eingestuft werden, da dessen Behebung meist fest geregelt und beschrieben ist. Die gesamte Behandlung eines User Service Requests wurde in Abbildung 4.8 in der Funktion „User Service Request bearbeiten“ zusammengefasst. Da deren Behandlung von der der in ITIL betrachteten Incidents abweicht, soll sie auch im Rahmen dieser Arbeit nicht ausführlicher betrachtet werden.

Nachdem Incidents als solche klassifiziert wurden, können sie einer Kategorie innerhalb der Supportstruktur des Unternehmens zugeordnet werden. Mögliche Kategorien in der Supportstruktur wurden bereits in der Einleitung dieses Abschnittes erwähnt. Falls eine Störung keiner Kategorie zugeordnet werden kann, sollte dafür eine Verhaltensregel innerhalb des Incident Management existieren, welche dann die Kategorie dieser Störung festlegt. Anschließend muss die Priorität der Incidents ermittelt werden, bevor die Standard-Incidents direkt bearbeitet werden (siehe Abb. 4.10) oder eine Störmusterprüfung durchgeführt wird, welche in Abbildung 4.11 näher beschrieben wird. Der in Abbildung 4.8 dargestellt Prozesswegweiser „Priorität ermitteln“ steht für die in Abbildung 4.9 übertragene Prozesskette. Neben der Priorität des Incidents kann der dargestellte Prozessstrang für die weitere Bearbeitung auch noch kundenspezifische Schwellenwerte liefern, sowie u.U. die Zeitfenster für die Incident-Behebung modifizieren.

[2] Definition siehe Abschnitt 4.3.3

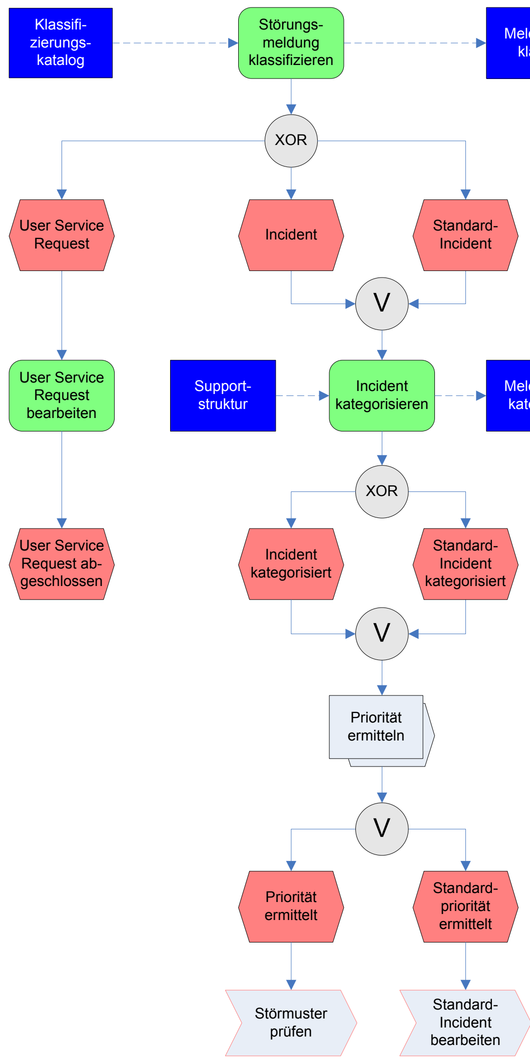


Abbildung 4.8.: Störungsmeldung klassifizieren

Wie an den vielen Inputs in Abbildung 4.9 zu sehen ist, spielt im Rahmen der Incident-Bearbeitung der Begriff Priorität eine wichtige Rolle. Die Priorität lässt sich primär aus zwei Aspekten, der Dringlichkeit (Urgency) und der Auswirkung (Impact), ermitteln. Die Dringlichkeit orientiert sich an den mit dem Kunden in den Service Level Agreements vereinbarten Ausfall- und Wiederherstellungszeiten. Da diese von Kunde zu Kunde in Bezug auf einen Service durchaus unterschiedlich sein können, ist es wichtig, die mit anderen Kunden vereinbarten Bearbeitungszeiten ebenfalls im Zusammenhang mit der Bestimmung der Dringlichkeit für diesen Incident einzubeziehen. Dies ist notwendig, da sich das Incident Management auch mit der Behebung potentieller Störungen, in diesem Fall mit der möglichen Nichteinhaltung eines SLA mit einem anderen Kunden, befasst.

Der zweite Aspekt, die Auswirkung, richtet sich nach den geschäftlichen Folgen, die diese Störung für den Kunden bedeutet. Mögliche weitere Einflusskriterien können die Größe, der Umfang und die Komplexität des Incidents sowie die Menge der zur Behebung benötigten Ressourcen sein.

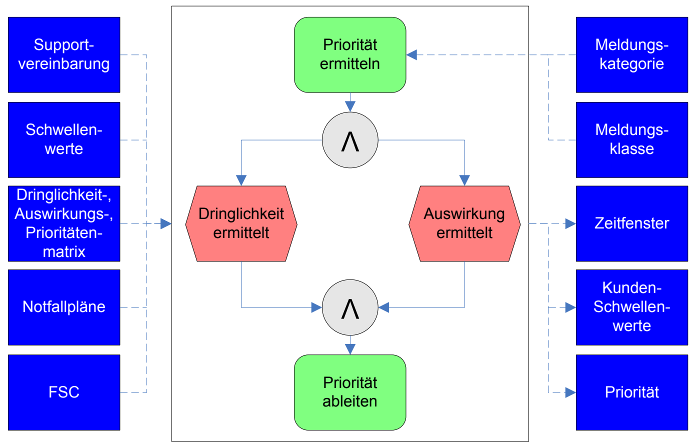


Abbildung 4.9.: Priorität ermitteln

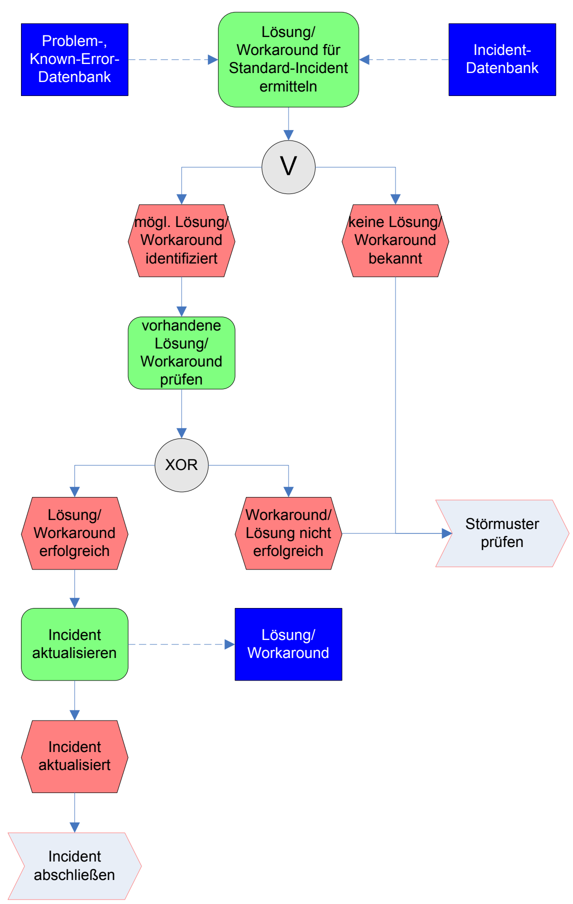


Abbildung 4.10.: Standard-Incident bearbeiten

Abbildung 4.10 zeigt die Bearbeitung eines Standard-Incidents. Wie im Beispiel mit dem vergessenen Passwort ist in der Regel dafür eine Lösung oder zumindest ein Workaround in der Incident-, Problem- oder Known-Error-Datenbank bekannt, welcher dann nur noch angewendet werden muss. Kann der Incident damit erfolgreich gelöst werden, z.B. indem das Passwort zurückgesetzt wurde, so kann der Incident nach dem Aktualisieren abgeschlossen werden. Kann der Standard-Incident nicht mit der vorhanden Lösung/Workaround behoben werden oder existiert keine bekannte oder anwendbare Vorgehensweise (z.B. weil das Passwort bei diesem System nicht einfach zurückgesetzt werden kann), so muss für diesen Incident, der ursprünglich als Standard-Incident eingestuft worden ist, ebenfalls eine Störmusterprüfung durchgeführt werden, die im Rahmen von Abbildung 4.11 beschrieben wird.

Die Aktivität „Störmuster prüfen“, die in itSMF [ITS 02] einen eigenen Subprozess darstellt, befasst sich im Anschluss an die Klassifizierung mit der Feststellung, ob eine ähnliche Störung bereits aufgetreten ist und ob dafür eine Lösung oder ein Workaround vorhanden ist. Weist der Incident ähnliche Symptome wie ein bereits bekanntes Problem oder ein bekannter Fehler auf, so wird er mit diesem verknüpft. Dadurch stehen dem Support-Mitarbeiter weitere Informationen zur Verfügung, die im Rahmen der Behebung zu Rate gezogen werden können. Auch Ähnlichkeiten mit anderen Incidents müssen abgeglichen werden.

Für die Störmusterprüfung ist der Einsatz eines Managementwerkzeugs zur *Event-Korrelation* ratsam. Mittels eines solchen Werkzeuges lassen sich zeitlich bzw. symptomatisch zusammenhängende Incidents erkennen und auf eine geringere Anzahl, im Idealfall genau auf eine Störungsmeldung zusammenfassen. Dadurch erfolgt dann keine separate Bearbeitung jeder einzelnen Störungsmeldung, sondern alle korrelierten Daten werden zusammen betrachtet. Wird eine Lösung für die korrelierte Störungsmeldung gefunden, so ist u.U. auch eine Lösung für alle Teil-Incidents gefunden.

Im Rahmen der in Abbildung 4.11 dargestellten Störmusterprüfung wird das Muster des Incidents bezogen auf die in den Incident-, Problem- und Known-Error-Datenbanken hin analysiert. In der Abbildung wird dabei im Falle des Vorhandenseins eines so genannten Störmusters zwischen einem Muster in den archivierten Daten und den momentan in Bearbeitung befindlichen unterschieden. Diese Unterscheidung wurde aus dem Grund vorgenommen, dass es im Falle eines archivierten Musters ausreichend ist, eine Datenkorrelation durchzuführen und eventuelle Lösungen oder Bearbeitungsschritte zu übernehmen. Hängt der betrachtete Incident jedoch zeitlich oder symptomatisch mit noch nicht abgeschlossenen Incidents zusammen, so können diese sich gegenseitig beeinflussen, weswegen neben der Datenkorrelation auch eine Verknüpfung der vermutlich zusammenhängenden Incidents erfolgen sollte. Die durch die Korrelation der Incident-Daten gewonnen neuen Informationen können die Kategorie und die Priorität des Incidents beeinflussen. Aus diesem Grunde ist es sinnvoll, nach der Korrelation eine Überprüfung bzw. Bestätigung der Meldungskategorie vorzunehmen. Eine Störung, die bspw. den Ausfall des Internets betrifft und in die Kategorie „Netzwerk“ eingeordnet worden ist, könnte nun der Kategorie „Hardware“ zugeteilt werden, falls die Ursache des Ausfalls nicht mehr am Router vermutet wird, sondern der berühmte Bagger ist, der alle Leitungen gekappt hat.

Die in Abbildung 4.11 aufgeführte Funktion „Lösung/Workaround ermitteln“ steht im Kontext dieses Subprozesses für die „erste Unterstützung“ und sollte nicht mit denen im Subprozess „Analyse und Diagnose“ durchgeführten Aktivitäten zum Identifizieren eines möglichen Behebungsansatzes verwechselt werden. Wird durch Korrelation oder durch die erste Untersuchung des Incidents eine (Übergangs-)Lösung ermittelt, so wird diese dem Subprozess „Beheben und Wiederherstellen“ übergeben (siehe Abschnitt 4.3.4). Kann in der Störmusterprüfung ausgehend von den bisherigen Daten dennoch keine Lösung oder Workaround innerhalb einer vorher festgelegten Zeitspanne ermittelt werden, so muss die Analyse des Incidents ausgeweitet werden. Zuvor sollte jedoch eventuell eine durch den Prozesswegweiser „Zeitfenster erweitern“ in Abbildung 4.11 angedeutete Aktualisierung des Zeitfensters für die Bearbeitung sowie der Priorität des Incidents erfolgen, um eventuelle Überschreitungen oder Engpässe im Prozess frühzeitig entdecken zu können. Dies ist in Abbildung 4.12 zu sehen. Im Rahmen des anschließenden Subprozesses „Analyse und Diagnose“ muss dann geprüft werden, ob eine Warnmeldung ausgegeben oder sogar eine Eskalation durchgeführt werden muss.

Bevor die erweiterte Analyse des Incidents im folgenden Abschnitt betrachtet wird, sind in Tabelle 4.2 noch einmal Inputs und Outputs dieses Subprozesses aufgelistet.

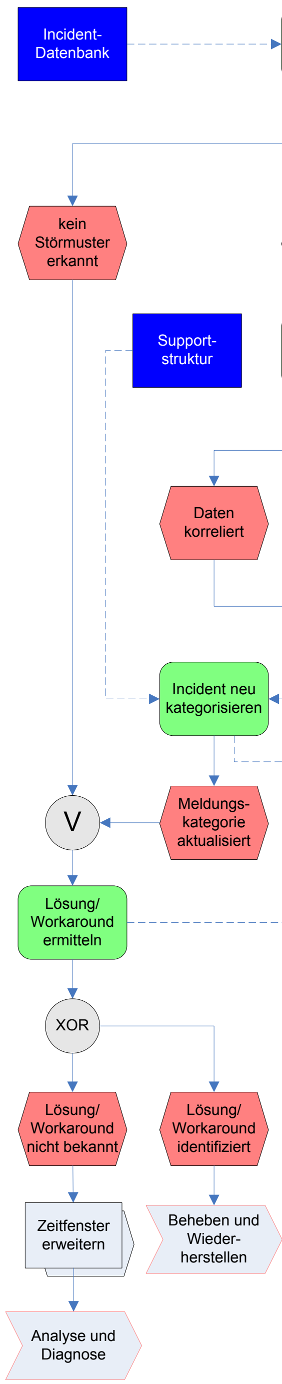


Abbildung 4.11.: Störmuster prüfen

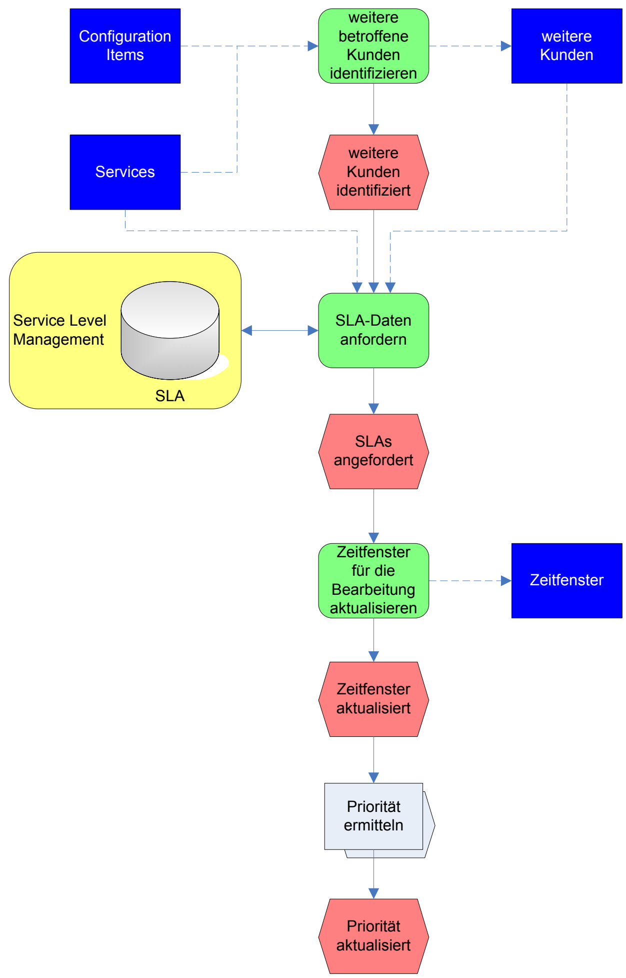


Abbildung 4.12.: Zeitfenster erweitern

Klassifizieren und erste Unterstützung	
Input	Output
• erfasste Störungsdaten (Incident Record) • Konfigurationsdaten aus der CMDB • Daten über bekannte Probleme/Fehler • Supportvereinbarungen (SLAs, OLAs, UCs) • Schwellenwerte/Zeitrahmen • Change-Plan (FSC) • Notfallpläne • Klassifizierungs- und Kategorisierungskatalog	• aktualisierte Incident-Daten • Workarounds/Lösungen für Incidents

Tabelle 4.2.: Klassifizieren und erste Unterstützung: Input/Output

Neben dem Incident Record an sich sind insbesondere für die Störmusterprüfung diesem Subprozess die verschiedensten Inputdaten zur Verfügung zu stellen, z.B. Konfigurationsdaten aus der CMDB, Daten über Probleme, bekannte Fehler aber auch über laufende Incidents, um nur einige zu nennen. Auch die Betrachtung des *FSC (Forward Schedule of Change)* kann für die Bearbeitung eines Incidents hilfreich sein. Wird in diesem bspw. der Austausch eines Routers angekündigt, so kann dies den Anwendern direkt mitgeteilt werden, oder die betroffenen Incidents können z.B. in ihrer Bearbeitung erst einmal zurückgestellt werden, bis der Austausch durchgeführt ist. Outputs sind neben den aktualisierten Daten der Meldung auch erste Lösungsansätze oder Workarounds für den Incident, die bspw. aus den Störmustern abgeleitet worden sind.

4.3.3. Analyse und Diagnose

Kann im Rahmen der ersten Unterstützung keine Lösung bzw. Workaround gefunden werden oder ist der Anwender mit dieser Lösung nicht einverstanden, so wird eine funktionale Eskalation auf das nächste Support-Level durchgeführt, was schematisch bereits in Abbildung 4.4 dargestellt ist. Eine funktionale Eskalation ist ebenfalls notwendig, wenn der Zeitrahmen für die Reaktions- oder Bearbeitungszeit auf dem jeweiligen Support-Level überschritten wird.

Da der Begriff der Eskalation eine wichtige Rolle spielt, soll er hier erst einmal definiert werden.

*„**Eskalation** ist ein Mechanismus, der eine schnelle Behebung von Störungen unterstützt.“[ITS 02]*

Eskalationsarten:

- Funktionale Eskalation:
  Funktional zu eskalieren bedeutet, weitere Spezialisten hinzuzuziehen, welche über mehr Know-how, Erfahrungen oder auch erweiterte Zugangsrechte verfügen. Das Weiterleiten eines Incident an das nächste Support-Level fällt in den Bereich einer funktionalen Eskalation.

- Hierarchische Eskalation:
  Eine hierarchische Eskalation wird eingeleitet, wenn die funktionale Eskalation nicht den gewünschten Erfolg bringt. Dies könnte z.B. an fehlenden Ressourcen oder Befugnissen liegen. In diesem Fall kann im Rahmen einer hierarchischen Eskalation dem Incident Management temporär z.B. weiteres Personal zur Verfügung gestellt oder dessen Befugnisse erweitert werden.

Aktivitäten dieses Subprozesses sind (vgl. [OGC 00, Pleg 05]):

- Beurteilen der Incident-Daten (überprüfen, verändern und korrigieren)
- Zusammentragen und Analyse aller mit dem Incident in Verbindung stehenden Informationen; dazu gehören u.a. Untersuchungsaktivitäten und -ergebnisse sowie die beteiligten Akteure.
- Incident-Daten aktualisieren

Wie in Abbildung 4.13 dargestellt, ist der Subprozess „Analyse und Diagnose“ iterativ. Ausgehend von den gegebenen Daten werden verschiedene mögliche Ursachen des Incidents eliminiert. Zuvor werden jedoch, wie durch den Prozesswegweiser „Warnung, Eskalation überprüfen“ signalisiert, die Rahmendaten des betrachteten Incidents überprüft. Diese durch den Prozesswegweiser zusammengefasste Überprüfung ist in Abbildung 4.14 zu sehen.

Bevor mit dem „Finden“ eines möglichen Lösungsansatzes für die Störung begonnen werden kann, werden noch einmal die bisherigen Daten des Incident z.B. auf ihre Korrektheit hin überprüft, ebenso die im Rahmen der „ersten Unterstützung“ erfasste Störungsbearbeitung. Nach den eben beschriebenen Aufgaben beginnt der eigentliche in Abbildung 4.13 dargestellte „iterative“ Teil dieses Subprozesses mit der Funktion „historische Daten überprüfen“. Im Rahmen der nun folgenden tiefer gehenden Analyse wird basierend auf den Datenbanken (Incident-, Problem- und Known-Error-Daten) sowie auch weiterführender Literatur und Benutzerhandbücher versucht, mögliche Störungsursachen zu eliminieren, um den Incident zu beheben.

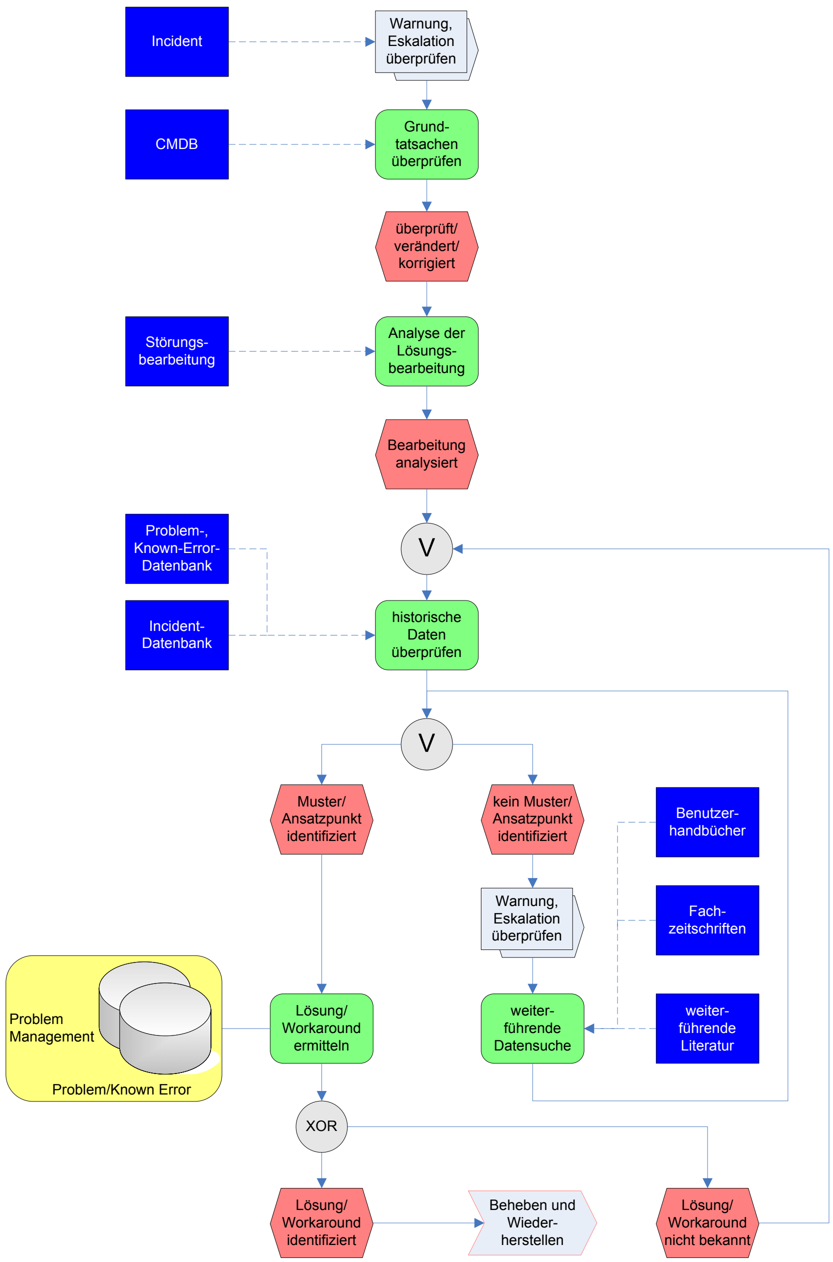


Abbildung 4.13.: Analyse und Diagnose des Incidents

Wird keine Lösung oder Workaround gefunden, so wird, falls möglich und falls die vorgegebenen Eskalationsschwellenwerte überschritten werden, zusätzliches Support-Personal hinzugezogen und auf das nächste Support-Level eskaliert. Auch eine hierarchische Eskalation ist möglich, wie zu Beginn dieses Abschnitts erläutert wurde.

Wird eine Lösung oder ein Workaround gefunden, so wird in Verbindung mit dem Problem Management ein Problem oder Known Error erstellt und in die entsprechende Datenbank eingetragen. Anschließend wird der Incident sowie der ermittelte Behebungsansatz dem Subprozess „Beheben und Wiederherstellen“ übergeben, der alle für die Behebung der Störung notwendigen Schritte einleitet bzw. durchführt.

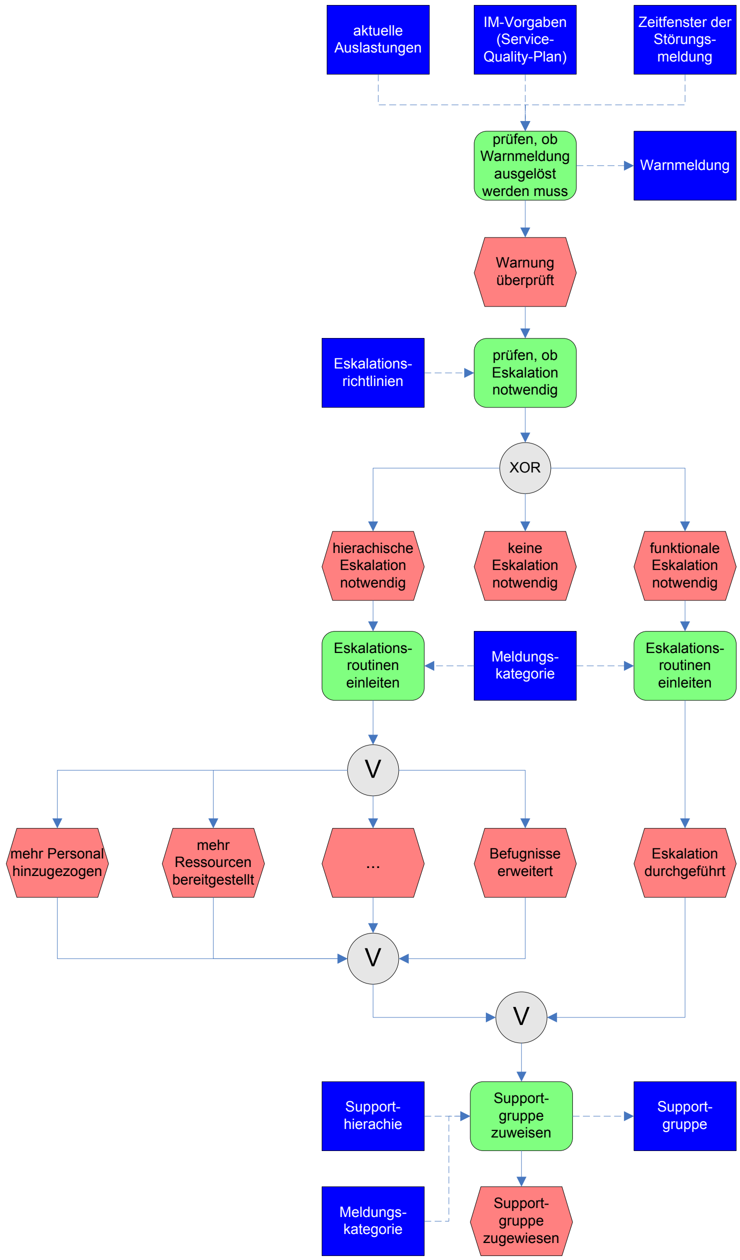


Abbildung 4.14.: Warnung und Eskalationsroutinen prüfen

Abbildung 4.14 zeigt die Verfeinerung des Prozesswegweisers „Warnung, Eskalation überprüfen". Die erste Funktion dieser Prozesskette befasst sich mit dem eventuellen Auslösen einer Warnmeldung, basierend auf der aktuellen Auslastung, den Vorgaben des Incident Management sowie dem Bearbeitungsfenster des Incidents. Wie bereits im Abschnitt 4.3.1 erwähnt, sollte eine Warnung ausgelöst werden, falls die in den SLAs vereinbarten Zeiten bereits kurz vor dem Überschreiten stehen oder bereits überschritten wurden. Ein weiterer Grund für eine Warnmeldung kann auch die Überlastung der Supportgruppe sein, welche für den Incident verantwortlich ist. Neben den Warnmeldungen wird wie in Abbildung 4.14 zu sehen auch die Notwendigkeit einer Eskalation geprüft. Ist eine hierarchische oder funktionale Eskalation durchzuführen, so wird der Incident in der Regel auch einer neuen Supportgruppe u.U. auf dem nächsten Support-Level zugewiesen. Das Weiterleiten an eine neue Supportgruppe wird meist mit zur Eskalation gezählt und soll durch die in der Abbildung 4.14 separat aufgeführte Funktion „Supportgruppe zuweisen“ nur noch einmal hervorgehoben werden. Im Falle, dass nicht auf das nächste Support-Level eskaliert wird, können der aktuellen Supportgruppe im Rahmen der hierarchischen Eskalation z.B. auch einfach mehr Personal, Ressourcen oder auch erweitere Befugnisse zugeordnet werden, um den Incident zu bearbeiten. Die Funktion „Supportgruppe zuweisen“ würde dann einfach die vorherige Supportgruppe bestätigen.

Wie bereits bei der Störmusterprüfung werden auch im Rahmen der Analyse und Diagnose neben dem Incident Record eine Vielzahl von Inputs benötigt. Einige wichtige sind in Tabelle 4.3 aufgeführt. Neben diesen können im Einzelfall weitere Inputs z.B. aus Fachzeitschriften, Handbüchern oder bekannte Bugs von Systemen und Anwendungen notwendig sein und für die erweitere Datensuche herangezogen werden. Die während der Analyse innerhalb dieses Subprozesses identifizierten Lösungen bzw. Workarounds sowie die verworfenen Behebungsansätze, welche im Incident Record vermerkt werden, aktualisieren die Daten der Störungsmeldung für deren weitere Bearbeitung.

Analyse und Diagnose	
Input	Output
• Incident-Daten • Konfigurationsdaten aus der CMDB • Problem- und Known-Error-Beschreibungen • Supportvereinbarungen (SLAs, OLAs, UCs) • Eskalationsschwellenwerte/Zeitrahmen • Change-Plan (FSC) • Release Policy/Plans/Notes • Klassifizierungs- und Kategorisierungskatalog	• aktualisierte Incident-Daten • Spezifikation über eine mögliche Lösung oder Workaround

Tabelle 4.3.: Analyse und Diagnose: Input/Output

Der in diesem Abschnitt beschriebenen Analyse und Diagnose schließt sich, falls eine Lösung oder ein Workaround für den betrachteten Incident identifiziert werden konnte, der Subprozess „Beheben und Wiederherstellen“ an, welcher im folgenden Abschnitt behandelt wird.

4.3.4. Beheben und Wiederherstellen

Wurde für den Incident eine Lösung oder ein Workaround ermittelt, so ist der Subprozess „Beheben und Wiederherstellen“ für deren Umsetzung zuständig. Unter Umständen muss für die Behebung ein RFC (Request for Change) beim Change Management gestellt werden. In diesem Fall wird die Bearbeitung des Incidents so lange zurückgestellt, bis ein Ergebnis für den gestellten RFC vorliegt. Wurde der Change genehmigt und erfolgreich durchgeführt, obliegt es wieder dem Incident-Management-Prozess, festzustellen, ob damit auch der Incident behoben wurde. Ist dies nicht der Fall, muss die Störungsmeldung neu klassifiziert und ein anderer Behebungsansatz gefunden werden.

Aktivitäten dieses Subprozesses sind (vgl. [OGC 00, Pleg 05]):

- Incident unter Verwendung der ermittelten Lösung bzw. Workarounds beheben
- ggf. RFC stellen (beinhaltet auch die Prüfung nach der Durchführung des RFCs)
- Service-Recovery-Maßnahmen ergreifen
- Incident-Daten aktualisieren (ggf. neue Klassifizierung und Kategorisierung)

Ausgangspunkt für die in Abbildung 4.15 dargestellte Bearbeitung bilden der Incident und die dafür identifizierte (Umgehungs-)Lösung. Daneben werden noch Informationen bzgl. der Notwendigkeit der Beantragung eines RFCs benötigt. Muss ein RFC beim Change Management eingereicht werden, so kann die

Behebung der Störung erst *nach* dessen Genehmigung durchgeführt werden. Wird der notwendige RFC abgelehnt, so geht der Incident zur erneuten Untersuchung an den Subprozess „Analyse und Diagnose“ zurück. Muss kein RFC gestellt werden oder ist dieser bereits vorautorisiert, so kann die Lösung direkt umgesetzt werden. Nach erfolgreicher Prüfung der Behebung kann der Incident geschlossen werden. Andernfalls muss er erneut untersucht werden.

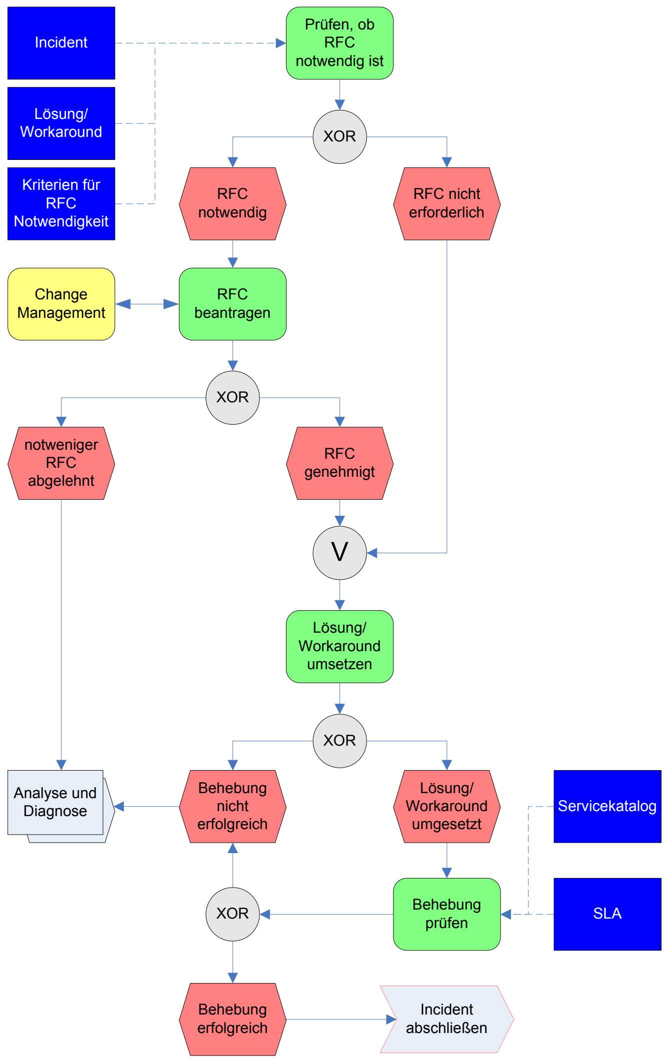


Abbildung 4.15.: Beheben und Wiederherstellen

In Tabelle 4.4 sind Inputs und Outputs dieses Subprozesses zusammengefasst. Neben den allgemeinen Inputdaten, wie dem Servicekatalog und den Supportvereinbarungen, spielen, insbesondere wenn ein RFC notwendig ist, auch Daten des Change Management z.B. in Form von Rollout-Informationen eine Rolle. Diese werden dazu benötigt, um feststellen zu können, ob die Lösung erfolgreich umgesetzt wurde, die Umsetzung das gewünschte Ergebnis gebracht hat, aber auch für eine spätere Durchführung eines eventuell notwendigen Rollbacks. Diese Informationen sind zwar in Abbildung 4.15 aufgrund der gewählten Verfeinerungstiefe nicht explizit als solche modelliert, stehen aber in Verbindung mit den dargestellten Funktionen „Lösung/Workaround umsetzen“ und „Behebung prüfen“.

Beheben und Wiederherstellen	
Input	Output
• Incident-Daten • Ergebnis eines gestellten RFC • Rollout-Informationen • abgeleitete Workarounds oder Lösungen • Servicekatalog • Supportvereinbarungen (SLAs, OLAs, UCs)	• aktualisierte Incident-Daten • RFC für zukünftige Incident-Behebung • beendeter Incident, inklusive Details zur Recovery

Tabelle 4.4.: Beheben und Wiederherstellen: Input/Output

Output dieses Subprozesses ist neben einem möglichen RFC im Idealfall eine aus Sicht des Incident Management behobene Störung, welche nur noch durch den Anwender verifiziert werden muss. Diese Verifikation sowie die weiteren mit dem Abschließen des Incidents verbundenen Aktivitäten werden im folgenden Abschnitt behandelt.

4.3.5. Störung abschließen

Konnte der Incident aus Sicht des Service Providers behoben werden, so muss der Anwender darüber informiert werden. Dies sollte im Idealfall durch die Person oder auch Abteilung erfolgen, welche die Störungsmeldung vom Anwender angenommen hat. Kann der Melder in einem vorher festgelegten Zeitrahmen über die Behebung der Störung informiert werden, so muss dessen Bestätigung eingeholt werden, bevor der Incident Record geschlossen werden kann. Ist der Anwender nicht mit der angebotenen Lösung einverstanden, so muss die Bearbeitung des Incidents an geeigneter Stelle wieder aufgenommen werden. Dies gilt besonders für Workarounds.

Aktivitäten dieses Subprozesses sind (vgl. [OGC 00, Pleg 05]):

- Anwender über Behebung des Incidents informieren
- Bestätigung einholen (Service Desk informieren)
- ggf. Incident erneut klassifizieren, falls Incident nicht geschlossen werden kann
- Incident Record abschließen und archivieren

Der in der Einleitung bereits beschriebene Ablauf ist in Abbildung 4.16 mittels EPK modelliert. Zu Beginn ist die Funktion „Melder über Behebung informieren“ zu sehen, die, wie der Name schon andeutet, versucht, mit dem Melder der Störung in Kontakt zu treten, um ihn über die Art der Behebung zu informieren. Neben dieser reinen Informationsaufgabe wird im Rahmen dieser Funktion auch überprüft, ob die Störung auf der Anwenderseite ebenfalls behoben wurde.

Ist die Störung auf Seite des Kunden nicht behoben oder der Anwender mit der Art der Behebung nicht einverstanden, so wird der Incident nach einer eventuellen Neukategorisierung für eine erneute Untersuchung wieder dem „Analyse und Diagnose“-Prozess übergeben. Ist die Behebung erfolgreich und wird sie vom Melder akzeptiert, so sollte die Dokumentation u.a. auf Vollständigkeit geprüft und die erfassten Incident-Daten gesichert werden, bevor der Incident endgültig abgeschlossen werden kann. Die archivierten Daten können dann z.B. in Zusammenhang mit späteren Störmusterprüfungen und Analysen zu Rate gezogen werden. Wurde im Rahmen der Behebung ein Rollout durchgeführt, so werden die im Incident Record gespeicherten Informationen für ein eventuelles Rollback benötigt. Für den Fall, dass der Melder der Störung nicht innerhalb eines vorher definierten Zeitfensters erreicht werden und somit deren Behebung nicht bestätigen kann, sollten im Incident-Management-Prozess Verhaltensweisen definiert sein, welche festlegen, wie mit dem noch nicht abgeschlossenen Incident weiter zu verfahren ist. Der Incident könnte bspw. nach Ablauf einer „Wartezeit“, wie in Abbildung 4.16 dargestellt, so behandelt werden, als ob der Melder die Behebung akzeptiert hat und somit abgeschlossen werden. Auch ein Aussetzen der weiteren Bearbeitung ist denkbar, bis der Anwender kontaktiert werden kann.

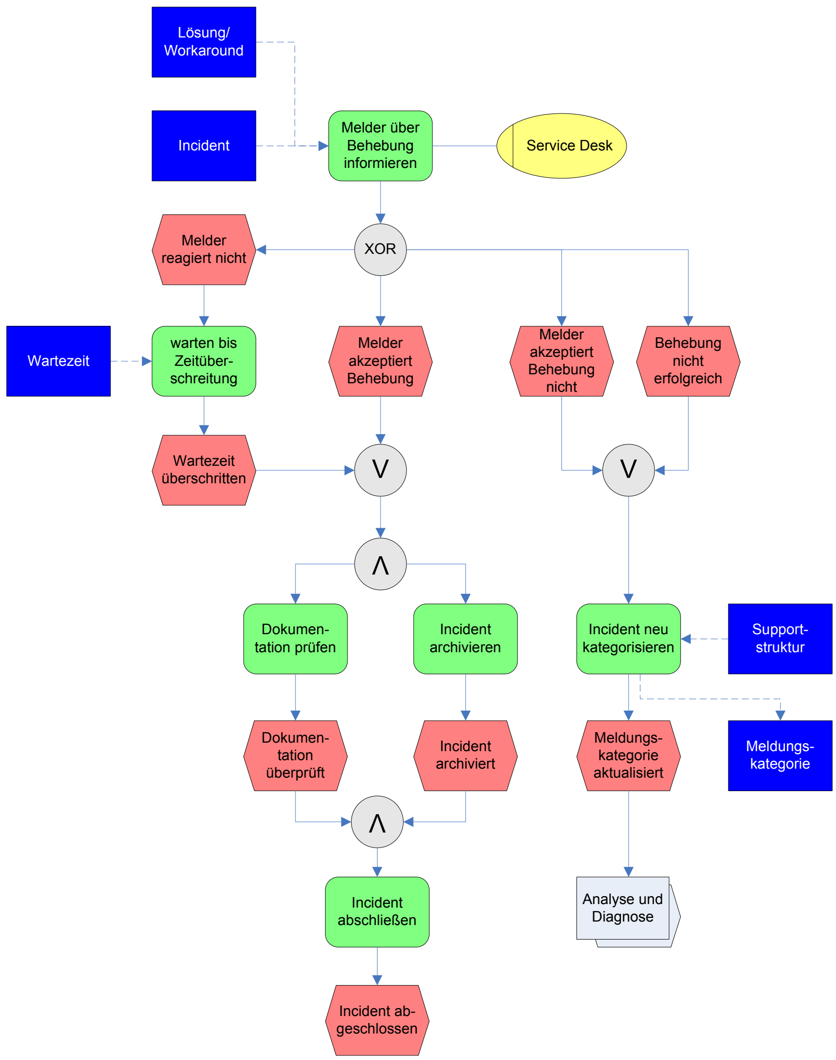


Abbildung 4.16.: Störung abschließen

Primärer Input für diesen Subprozess sind der Incident Record mit der umgesetzten (Umgehungs-)Lösung, wie Tabelle 4.5 entnommen werden kann. Ist nicht jeder Supportmitarbeiter berechtigt, einen Incident zu schließen, müssen noch zusätzliche Informationen diesbezüglich bereitgestellt werden. Kann der Incident nicht geschlossen werden, so sind für dessen eventuelle Neukategorisierung ebenso die entsprechenden Informationen z.B. in Form eines Klassifizierungs- und Kategorisierungskataloges als Input zu betrachten. Output stellen die aktualisierten Daten der Störungsmeldung dar, welche im Idealfall für einen archivierten Incident stehen, der erfolgreich abgeschlossen wurde.

Neben den bisher beschriebenen Subprozessen, welche in ihrer Gesamtheit gesehen (siehe Abb. 4.20) den Hauptteil des Incident-Management-Prozesses darstellen, existiert noch ein weiterer Subprozess „Monitoring und Tracking“, der gegenüber den bisher betrachteten eine Sonderstellung einnimmt.

Störung abschließen	
Input	Output
• Incident-Daten • behobener Incident • Informationen über Personen, die autorisiert sind, den Incident zu schließen • Klassifizierungs- und Kategorisierungskatalog	• aktualisierte Incident-Daten • geschlossener Incident Record

Tabelle 4.5.: Störung abschließen: Input/Output

4.3.6. Monitoring und Tracking

Der Subprozess „Monitoring und Tracking" ist parallel zu den anderen Subprozessen angeordnet, da er alle Aktivitäten zu überwachen und zu koordinieren hat. Für jeden Incident sind dabei dessen Bearbeitungsfortschritte und -zeiten zu verfolgen, um ggf. eine Eskalation durchzuführen oder den Anwender über Statusänderungen zu informieren. Prinzipiell ist eine Eskalation erforderlich, wenn ein Incident nicht innerhalb eines Support-Levels oder innerhalb eines vereinbarten Zeitrahmens behoben werden kann. Eine Definition von Eskalation sowie den zwei unterschiedenen Eskalationsarten wurden in Abschnitt 4.3.3 gegeben. Des Weiteren werden für andere Managementbereiche sowie für die Kunden Daten gesammelt und in Form von Reports zur Verfügung gestellt.

Aktivitäten dieses Subprozesses sind (vgl. [OGC 00, Pleg 05]):

- Protokollieren aller Bearbeitungsschritte
- Fortschritt der Incidentbehebung überwachen
- Support-Personal über die geltenden Zeitrahmen informieren, sofern diese nicht im Incident Record gespeichert sind
- Zeitrahmen überwachen, ggf. Eskalationen (funktional, hierarchisch) durchführen
- Anwender informieren
- Reports erstellen

Da dieser Subprozess für den reibungslosen Ablauf des gesamten Prozesses verantwortlich ist, indem er alle Aktivitäten zu überwachen und zu koordinieren hat, können ihm die in Tabelle 4.6 dargestellten Inputs und Outputs zugeordnet werden. Diese wurden bis auf die Reports und Kundenberichte, die in dieser Arbeit nicht weiter vertieft werden, bereits umfassend betrachtet.

Monitoring und Tracking	
Input	Output
• Incident Record • Supportvereinbarungen (SLAs, OLAs, UCs) • Change-Plan (FSC) • Klassifizierungs- und Kategorisierungskatalog • Eskalationsschwellenwerte/Zeitrahmen • Release Policy/Plans/Notes • Geschäftsentwicklung und -ziele	• Reports über den Incident-Fortschritt • eskalierte Incident-Daten • Kundenberichte • Capacity/Availability Events/Alerts

Tabelle 4.6.: Monitoring und Tracking: Input/Output

Fasst man den Subprozesses „Monitoring und Tracking" zusammen, so stellt man fest, dass er sich wie bereits erwähnt von den anderen Subprozessen des Incident Management dadurch unterscheidet, dass er parallel zu den anderen angeordnet ist (siehe Abb. 4.3). Doch nicht nur durch seine Lage innerhalb des Prozesses unterscheidet er sich, sondern auch in Bezug auf seine Aktivitäten, welche überwiegend passiver, begleitender, überwachender und kontrollierender Natur sind. Trotzdem greift er auch aktiv in den Bearbeitungsprozess z.B. in Form von Eskalationen ein, sammelt Daten für die Erstellung von Reports oder informiert die Anwender.

Nachdem der Incident-Management-Prozess sowohl als Ganzes als auch aufgespalten in seine Subprozesse und Aktivitäten betrachtet wurde, soll der nächste Abschnitt einen Blick auf bereits im Incident Management eingesetzte Tools werfen. Insbesondere soll kurz auf so genannte Trouble-Ticket-Systeme eingegangen werden.

4.4. Tool-Unterstützung

Wie bereits erwähnt schlägt ITIL [OGC 00] zur Unterstützung des Incident-Management-Prozesses die Verwendung von Management-Tools vor. Ein besonders im Rahmen des „Annehmen und Erfassen“ eingesetztes Tool ist ein Trouble-Ticket-System (TTS). Abbildung 4.17 zeigt den grundsätzlichen Aufbau eines solchen Systems. Je nach Umfang kann dieses einzelne Aktivitäten, Subprozesse oder sogar den gesamten Prozess des Incident Managements unterstützen. Wird ein solches Trouble-Ticket-System verwendet, so werden die Daten des bisher betrachteten Incident Records mit denen des Trouble Tickets gleichgesetzt, oder auf dieses abgebildet. Zur Vereinfachung wird in diesem Abschnitt der Begriff Trouble Ticket als Synonym für die Daten eines Incident Records verwendet.

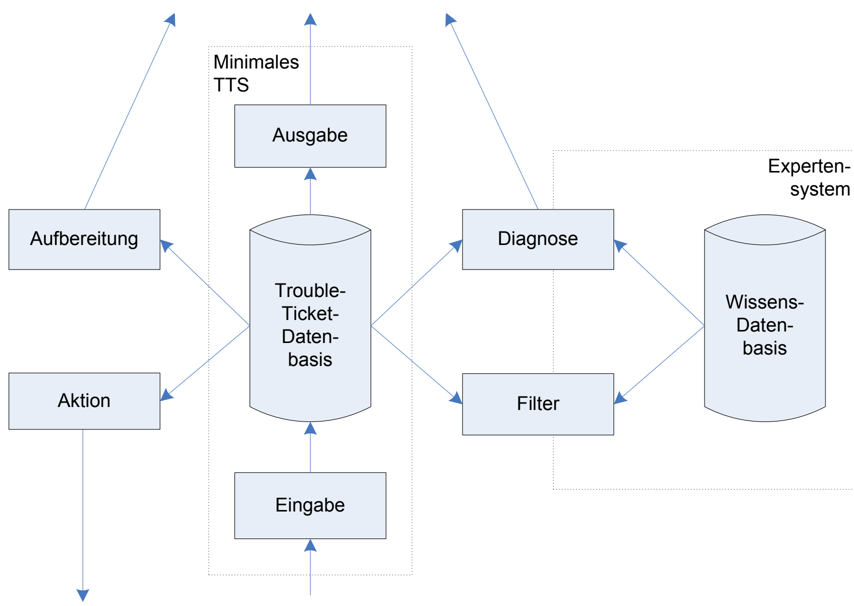


Abbildung 4.17.: Aufbau eines Trouble-Ticket-Systems (Quelle: [HAN 99])

Der Hauptbestandteil eines (minimalen) Trouble-Ticket-Systems ist gemäß Abbildung 4.17 eine Datenbank[3]. Diese enthält die Informationen der Incident Records in Form von so genannten Trouble Tickets (TT), die von einer Person oder einer Managementanwendung erzeugt und gesammelt werden. Neben dieser Datenbasis besteht ein minimales Trouble-Ticket-System zusätzlich noch aus einem Ein- und Ausgabemodul. Über das Eingabemodul werden aus den Störungsbeschreibungen Trouble Tickets erzeugt und in die Datenbasis eingetragen. Mittels des Ausgabemoduls können die Trouble Tickets wieder ausgelesen werden. Zusätzlich zu den bereits erwähnten Modulen kann ein Trouble-Ticket-System noch durch weitere Module erweitert werden.

[3] Mehr über Trouble-Ticket-Systeme, sowie die Begriffsverwendung in den Abbildungen 4.17 bis 4.19, ist [HAN 99] Kapitel 14.2 zu entnehmen.

Das Aktionsmodul führt Aktionen aus, von denen bereits im Voraus bekannt ist, dass sie nach dem Eintreffen einer Problemmeldung durchzuführen sind. Ein Beispiel hierfür ist Benachrichtigung des Anwenders, dass seine Störungsmeldung erfasst wurde. Ein anderes Modul ist das Aufbereitungsmodul, welches eher für nachgelagerte Instanzen von Bedeutung ist. Dieses ist, wie bereits aus seinem Namen ersichtlich, für die Aufbereitung der Daten eines Trouble Tickets zuständig. Das heißt, die Trouble-Ticket-Informationen werden entsprechend dem Zweck nachgelagerter Managementprozesse zusammengestellt, z.B. für Statistiken über die Auslastung des Service Desks oder die Bearbeitungszeit eines Trouble Tickets.

Im Filtermodul wird entschieden, ob für eine eingehende Störungsbeschreibung ein Trouble Ticket erstellt wird oder ob z.B. eine Korrelation bzgl. mehrerer eingehender Meldungen sinnvoll und notwendig ist. Betrachtet man den Incident-Prozess, so entsprechen die Aufgaben des Eingabe- und Filtermoduls grob den Aufgaben des Service Desks. Das Diagnosemodul liefert dann den Übergang von einem erweiterten Trouble-Ticket-System zu einem Expertensystem, welches auf Basis der vom Trouble Ticket gelieferten Symptome und einer vorhandenen Wissensbasis versucht, die Fehlerursache festzustellen und zu beseitigen. Abbildung 4.18 zeigt die grundsätzliche Informationsstruktur eines Trouble Tickets.

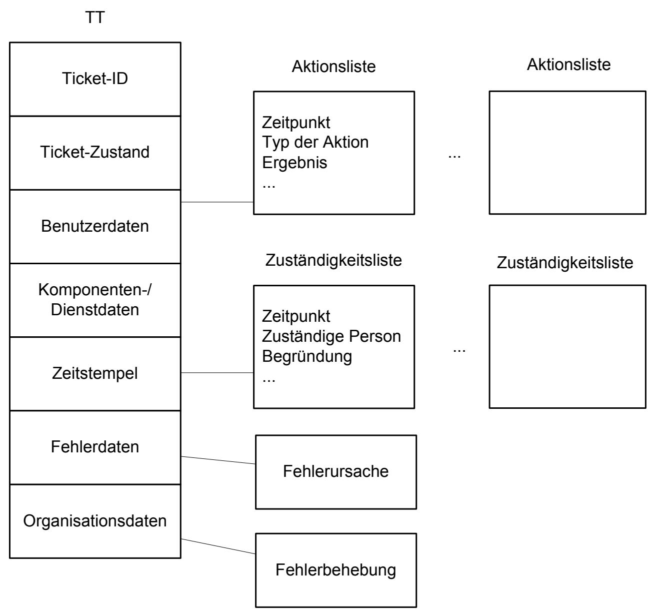


Abbildung 4.18.: Grundsätzliche Informationsstruktur eines Trouble Tickets (TT) (Quelle: [HAN 99])

Einzelnen Informationsteile eines Incident Record bzw. eines Trouble Tickets werden nachfolgend genauer betrachtet:

- Ticket-ID: Dies entspricht der Störnummer und dient der eindeutigen Identifizierung eines Trouble Tickets.
- Ticket-Zustand: Dieser Eintrag richtet sich nach der Position des Trouble Tickets im Bearbeitungszyklus des Incident-Management-Prozesses. Mögliche Unterscheidungen sind neu, angenommen, eingeplant, weitergeleitet, in Bearbeitung, zurückgestellt/Bearbeitung angehalten, gelöst und geschlossen.
- Benutzerdaten: Diese beinhalten u.a. Kunde, Anwender, Erreichbarkeit (für den weiteren Kontakt z.B. Telefonnummer und E-Mail-Adresse) und Standort.
- Komponenten-/Dienstdaten: Hier werden die betroffenen Configuration Items und Dienste erfasst.
- Zeitstempel: Dieser gibt an, wann die gemeldete Störung gemeldet bzw. aufgetreten ist.

- Fehlerdaten: Dieser Teil enthält eine textuelle Beschreibung der Störung.
- Organisationsdaten: Diese organisationsspezifischen Daten umfassen u.a. Daten über die erfassende Person, Priorität und zugewiesene Experten.

Aus den Komponenten-/Dienstdaten und dem Zeitstempel können weitere Zeitrahmen z.B. für die Reaktions- und Bearbeitungszeit abgeleitet und im Trouble Ticket gespeichert werden. Neben den Ausgangsdaten müssen weitere Daten erfasst werden, wobei ein Großteil der Organisationsdaten sowie die Daten der Aktions- und Zuständigkeitslisten, die Fehlerursache und die Fehlerbehebung erst im Laufe der Bearbeitung des Incidents ergänzt werden.

Angelehnt an [HAN 99] ist in Abbildung 4.19 ein in den Incident-Management-Prozess integriertes Trouble-Ticket-System dargestellt. Dieses System erhält primär Input vom Service Desk, welcher über Telefon, Fax oder E-Mail kontaktiert werden kann. Auch direkte Schnittstellen zwischen dem Trouble-Ticket-System und dem Anwender sind denkbar. Eine Möglichkeit ist in diesem Zusammenhang ein Web-Portal, in welches Anwender die notwendigen Störungs- und Anwenderdaten eintragen. Dieser direkte Zugang sollte jedoch nicht zu umfassend sein, d.h. es sollte nicht in jedem Fall automatisch ein Trouble Ticket erzeugt werden, da dem Anwender u.a. die providerinterne Klassifizierung und Priorisierung nicht bekannt ist bzw. sein sollte.

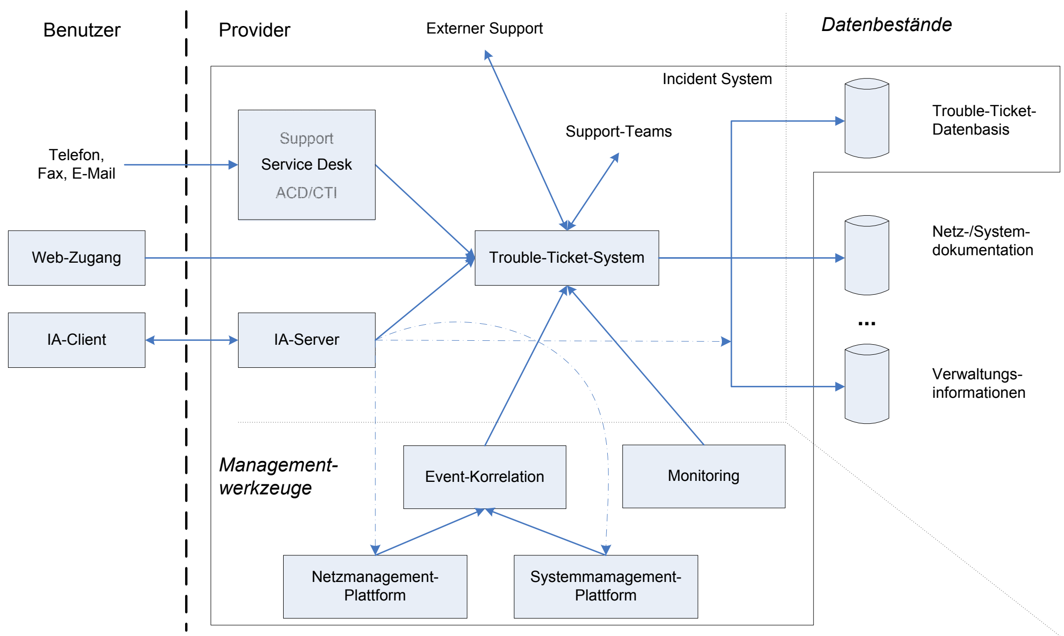


Abbildung 4.19.: Integration eines Trouble-Ticket-Systems (vgl. [HAN 99])

Eine andere direkte Schnittstelle kann zwischen Managementwerkzeugen, die z.B. Netzkomponenten oder auch Applikationen überwachen und im Fehlerfall automatisch eine Störungsmeldung erzeugen, und dem Trouble-Ticket-System bestehen. Für die so erzeugten Störungsmeldungen kann i.d.R. automatisch ein Trouble Ticket erzeugt werden, da ihnen die subjektive Betrachtung durch den Anwender fehlt.

Im Rahmen der Erfassung eines Incidents ist es in Kombination mit einem Trouble-Ticket-System u.U. nicht sinnvoll, für jeden Incident ein eigenes Trouble Ticket zu erzeugen. Hier setzt die Idee der Event-Korrelation an. Der Einsatz eines solchen Werkzeuges soll im Falle einer kausalen Kopplung die Menge der zu bearbeitenden Trouble Tickets reduzieren (siehe auch Abschnitt 4.3.2).

Neben der Event-Korrelation ist auch Monitoring ein wichtiger Aspekt im Incident Management, welcher im Subprozess „Monitoring und Tracking“ (Abschnitt 4.3.6) betrachtet wurde.

4.5. Zusammenfassung

In diesem Kapitel wurde der Incident-Management-Prozess betrachtet, welcher als Referenzprozess für das zu entwickelnde SOA-basierte Konzept dient. Dazu wurde zuerst das Umfeld des Prozesses vorgestellt, bevor die allgemeine Prozessstruktur nach ITIL vorgestellt wurde. Aufbauend auf den dort identifizierten Subprozessen des Incident Managements wurden diese im Rahmen einer Prozessanalyse verfeinert und mittels EPKs modelliert. Die identifizierten Inputs und Outputs wurden in Tabellenform einander gegenübergestellt. Da insbesondere im Rahmen des Incident-Management-Prozesses der Einsatz von Tools eine wichtige Rolle zur Prozessunterstützung spielt, wurde im letzten Abschnitt der grundsätzliche Aufbau und die Funktion eines Trouble-Ticket-Systems betrachtet.

Nachdem die Subprozesse des Incident-Management-Prozesses bereits durch EPKs dargestellt wurden, ist in Abbildung 4.20 noch einmal der Incident-Management-Prozess in seiner Gesamtheit als EPK abgebildet. Anhand dieser Abbildung ist zu sehen, dass der gewählte Ansatz, Ereignisgesteuerte Prozessketten zu verwenden, zur Modellierung des ausgewählten Prozesses geeignet ist.

Tabelle 4.7 liefert eine erweiterte Übersicht auf die verschiedenen aus den Aktivitäten des Incident-Management-Prozesses identifizierten Input- und Output-Datenklassen, welche an dieser Stelle aufgrund des hohen Abstraktionsniveaus der ITIL-Prozessbeschreibungen nur dem Namen nach genannt werden. Eine detailliertere Spezifikation der Daten der jeweiligen Klassen ist unternehmensspezifisch vorzunehmen.

Datenklasse	Input	Output
Capacity/Availability Events/Alerts		x
Change-Plan (FSC)	x	
Geschäftsentwicklung und -ziele	x	
Incident Records		x
Informationen über Konfigurationselemente	x	
Informationsbedarf bzgl. der CMDB		x
Klassifizierungs- und Kategorisierungskatalog	x	
Notfallpläne	x	
Probleme/Bekannte Fehler	x	
Release Policy/Plans/Notes	x	
Reports		x
Requests for Change (RFCs)		x
Rollout-Informationen	x	
Schwellenwerte/Checkpoints	x	
Servicekatalog	x	
User Service Request	x	x
Störungsinformationen	x	x
Supportvereinbarungen (SLAs OLAs, UCs)	x	
Workarounds/Lösungen	x	x

Tabelle 4.7.: Input- und Output-Datenklassen (vgl. [Sage 05])

Die in diesem Kapitel erstellte Modellierung des Incident-Management-Prozesses durch EPKs sowie die durch die Input- und Output-Datenklassen identifizierten Schnittstellen bilden die Ausgangssituation für die Entwicklung und Umsetzung der SOA-Idee in den Kapiteln 5 und 6.

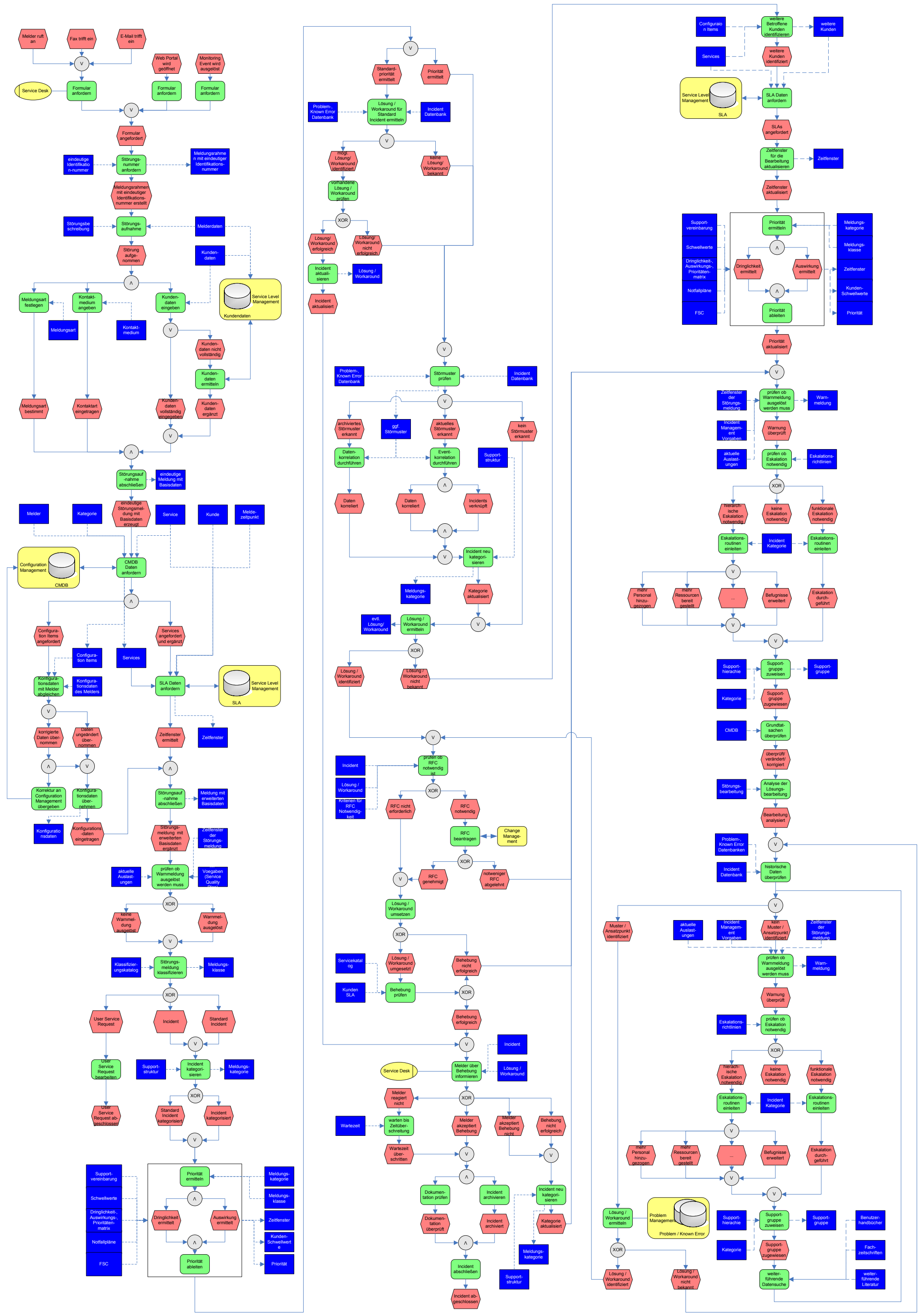


Abbildung 4.20.: EPK des ITIL Incident-Management-Prozesses

Kapitel 5.

Konzeptentwicklung

Dieses Kapitel befasst sich mit der eigentlichen Entwicklung des SOA-basierten Konzepts. Dazu werden die in der Prozessanalyse in Kapitel 4 identifizierten Inputs und Outputs, Aktivitäten und die Prozesslogik verwendet, um sie im Kontext von SOA zu betrachten und einzuordnen. Wie in Abbildung 5.1 dargestellt, bilden sie den Ausgangspunkt für die weitere Entwicklung des Konzepts im Rahmen der Phase „SOA-Idee umsetzen“. Aus ihnen werden in diesem Kapitel die Schnittstellen, die funktionalen Kapseln und die Orchestrierung abgeleitet, welche dann ihrerseits den Ausgangspunkt für die spätere Umsetzung des Konzepts bilden.

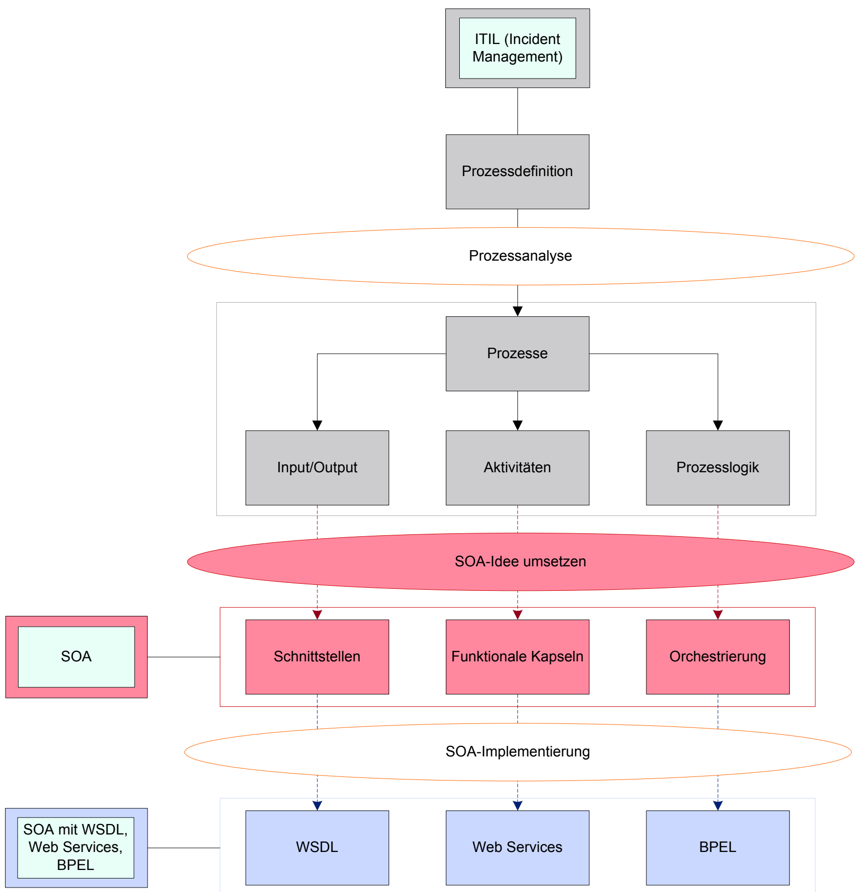


Abbildung 5.1.: Vorgehensweise bei der Konzeptentwicklung

Auch wenn in Abbildung 5.1 jedem Punkt im Incident Management (Input/Output, Aktivitäten, Prozesslogik) ein Aspekt des Konzepts innerhalb des SOA-Kontext (Schnittstellen, funktionale Kapselung, Orchestrierung) zugeordnet wird und jeder dieser Aspekte in einem eigenen Abschnitt in diesem Kapitel behandelt wird, stehen die betrachteten Punkte dennoch nicht völlig isoliert nebeneinander, sondern beeinflussen sich gegenseitig, wie in Abbildung 5.2 dargestellt ist. Für die Konzeptentwicklung wird an geeigneter Stelle der Datenaustausch betrachtet, um vorhandene Schnittstellen zu identifizieren. Anschließend fließen diese neben der Orchestrierung in die funktionale Kapselung des Incident-Management-Prozesses ein, wobei die aus der Prozesslogik abgeleitete Orchestrierung und die funktionale Kapselung voneinander abhängen. Dennoch soll die Orchestrierung hier prinzipiell als der funktionalen Kapselung nachgelagert verstanden werden, da sie auf diese aufbauend die einzelnen Kapseln wieder zum eigentlichen Prozess zusammenfügt, weshalb sie auch erst im Anschluss an den Abschnitt zur funktionalen Kapselung behandelt wird.

5.1. Schnittstellen

In diesem Kapitel wird der Begriff der Schnittstelle unter zwei unterschiedlichen Gesichtspunkten betrachtet. Zum einen wird in diesem Abschnitt der Prozess als Blackbox gesehen, welcher Schnittstellen zu anderen ITIL-Prozessen zugeordnet werden. Bei dieser Betrachtungsweise muss keine tiefer gehende Zuordnung der Schnittstellen zu einzelnen Bereichen des Prozesses erfolgen. Neben dieser ersten Sichtweise wird in Abschnitt 5.2.3 im Rahmen der funktionalen Kapselung die Identifikation von Schnittstellen zwischen Teilen des Prozesses betrachtet, welche erst durch diese entstehen.

Ausgangspunkt für die Identifizierung von Schnittstellen sind die in der Prozessanalyse ermittelten Inputs und Outputs. Entsprechend den Verantwortlichkeiten für Daten, die dem Prozess als Input zur Verfügung gestellt werden, und den Adressaten der Outputdaten lassen sich die Schnittstellen ermitteln. Die eben erwähnten Verantwortlichkeiten orientieren sich dabei an den von ITIL vorgegebenen Verantwortlichkeiten der einzelnen Prozesse sowie der unternehmensspezifischen Umsetzung dieser ITIL-Empfehlungen.

Da das Konzept möglichst unabhängig vom Prozess sein soll, auf den es angewendet wird, und die in diesem Abschnitt zu identifizierenden Schnittstellen und die auszutauschenden Input- und Outputdaten prozess- und unternehmensspezifisch sind, wird an dieser Stelle nicht näher darauf eingegangen. Aufbauend auf der Zuordnung der Schnittstellen werden diese im Kontext der funktionalen Kapselung vertiefend betrachtet. Für den ITIL Incident-Management-Prozess, welcher als Beispielprozess behandelt wird, sind denen in der Prozessanalyse ermittelten Inputs und Outputs im Abschnitt 7.1.1 Schnittstellen zu den anderen IITL-Prozessen zugeordnet.

5.2. Funktionale Kapselung

Der Begriff der funktionalen Kapselung spielt in diesem und in den nachfolgenden Kapiteln eine zentrale Rolle, weswegen er an dieser Stelle erst einmal definiert werden soll:

*Unter einer **funktionalen Kapselung** verstehen wir das Zusammenfassen einzelner Funktionalitäten zu einer logischen Einheit. Diese Einheit bezeichnen wir als **funktionale Kapsel**.*

Da die funktionale Kapselung und spätere Orchestrierung des Prozesses nicht trivial ist, werden in diesem Abschnitt zuerst einige theoretische Muster und Vorgehensweisen geschildert, bevor dann in Kapitel 7 ein Teilabschnitt des Incident-Management-Prozesses herausgegriffen und prototypisch umgesetzt wird. Auf einen möglichen Vollständigkeitsbeweis der in diesem Abschnitt besprochenen Muster wird verzichtet, da im Rahmen dieser Arbeit die Unterstützung des Incident-Management-Prozesses im Vordergrund steht.

Grundsätzlich kann die funktionale Kapselung der an der Störungsbearbeitung beteiligten ITIL-Prozesse und die funktionale Kapselung innerhalb des Incident-Management-Prozesses betrachten werden. Da im Rahmen dieser Arbeit der Incident-Management-Prozess im Vordergrund steht, wird schwerpunktmäßig die funktionale Kapselung innerhalb dieses Prozesses behandelt. Dennoch kann der vorgestellte Ansatz auch auf die anderen Prozesse übertragen werden.

Weil die funktionale Kapselung des Incident-Management-Prozesses verschiedenen z.T. prozessspezifischen Faktoren unterworfen ist, wird sich der erste Teil dieses Abschnitts mit den Aspekten befassen, welche prinzipiell die funktionale Kapselung des Incident-Management-Prozesses beeinflussen. Dazu gehören etwa die Eigenschaften von WSDL und BPEL, die aus der geplanten Umsetzung des Konzepts resultieren, aber erst im folgenden Kapitel ausführlich betrachtet werden. Anschließend wird ein Vorgehen entwickelt, das die Identifizierung und Bildung der funktionalen Kapseln beschreibt, bevor nochmals die Schnittstellen des vorangegangenen Abschnitts aufgegriffen und diesmal im Kontext der funktionalen Kapselung betrachtet werden.

5.2.1. Einflussfaktoren

Den Ausgangspunkt für die funktionale Kapselung stellt in erster Linie Prozess dar. Dieser Abschnitt wird sich mit den Aspekten befassen, welche Einfluss auf die mögliche funktionale Kapselung des Prozesses haben. In Abbildung 5.2 sind mögliche Einflussfaktoren dargestellt, welche im folgenden genauer betrachtet werden. Den Großteil der dargestellten Einflüsse bilden die prozessspezifischen Faktoren wie die Input- und Outputdaten, die Aktivitäten sowie die Prozesslogik. Daneben sind aber auch noch Faktoren der späteren Umsetzung und allgemeine Ansatzpunkte abgebildet.

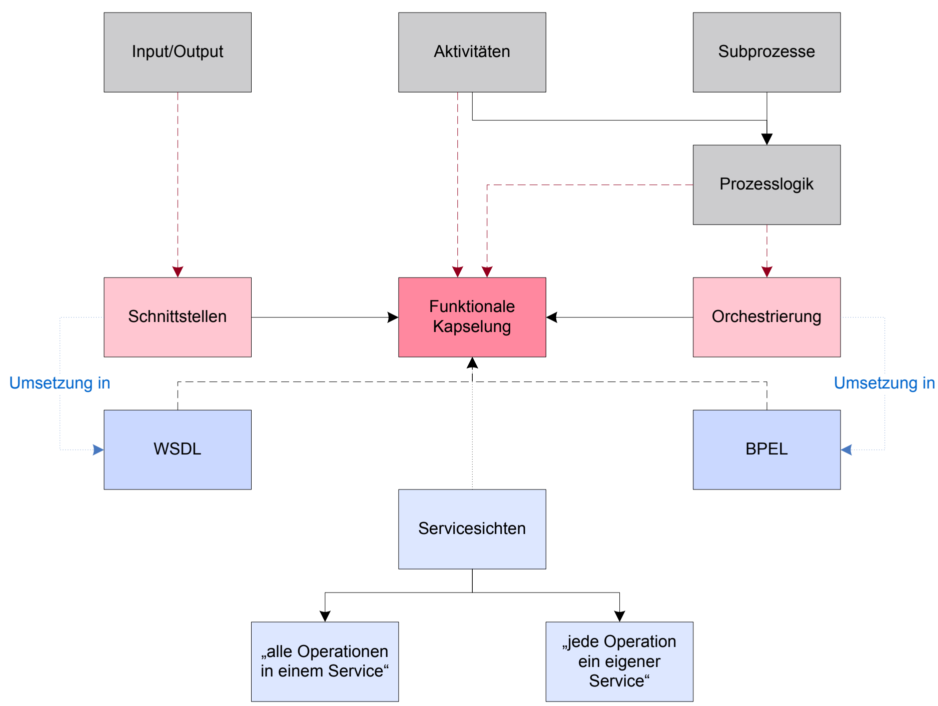


Abbildung 5.2.: Einflussfaktoren auf die funktionale Kapselung

Durch die in den Subprozessen des Incident-Management-Prozesses ermittelten Input- und Outputdaten können weitere an der Störungsbearbeitung beteiligte ITIL-Prozesse identifiziert werden (siehe Tabelle 7.1). Zur Vereinfachung der weiteren Betrachtung wird angenommen, dass jeder dieser Prozesse dem Incident Management über eine einzige Schnittstelle die benötigten Daten und Funktionalitäten bereitstellt.

In der Praxis wird man u.U. nicht alle Funktionalitäten in eine Kapsel bzw. einen Service packen, insbesondere wenn bereits etablierte Modellierungsmethoden (z.B. die objektorientierte Modellierung) zur Verfügung stehen.

Einen weiteren Einflussfaktor für die funktionale Kapselung des Incident Managements liefern Aktivitäten, Subprozesse und die daraus abgeleitete Prozesslogik, welche sich in der Orchestrierung widerspiegelt. Im Zusammenhang mit den Schnittstellen und der Orchestrierung zählen sprachspezifische Eigenheiten von WSDL und BPEL ebenfalls zu den Faktoren, die bei der funktionalen Kapselung beachtet werden müssen, da im Rahmen dieser Arbeit die Spezifikation der Schnittstellen mittels WSDL und die Umsetzung der Orchestrierung mittels BPEL erfolgen soll.

Des Weiteren sollen bei der Identifizierung einer geeigneten Kapselung auch noch die folgende beiden Standpunkte einfließen: Der erste beinhaltet die Vorstellung, jede benötigte Operation z.B. zum Abfragen oder Modifizieren von Daten innerhalb eines Services zu kapseln. Der zweite Standpunkt sieht die Kapselung aller Operationen in einem einzigen Service vor. Diese beiden Standpunkte stellen die Extrema dar, wobei sich das angestrebte Ziel für die funktionale Kapselung des Incident-Management-Prozesses wahrscheinlich zwischen diesen Punkten befinden wird.

5.2.2. Kapselung ableiten

Nachdem im vorangegangenen Abschnitt Einflussfaktoren auf die funktionale Kapselung vorgestellt wurden, soll in diesem Abschnitt ein allgemeines Vorgehen zur funktionalen Kapselung eines Prozesses beschrieben werden. Im Rahmen dieser funktionalen Kapselung werden alle Funktionen des Prozesses einer funktionalen Kapsel zugeordnet. Dazu wurde der zu modellierende Prozess bereits in der Analysephase in seine Subprozesse und Aktivitäten aufgespalten. Für den Incident-Management-Prozess wurde des Weiteren jeder dieser Subprozesse mittels einer oder mehrerer *eEPKs (erweiterte Ereignisgesteuerte Prozessketten)* graphisch dargestellt, welche an dieser Stelle als Grundlage für das Ableiten der funktionalen Kapseln dienen.

Bevor allerdings die funktionalen Kapseln des Incident-Management-Prozesses ermittelt werden, soll zuerst die allgemeine Struktur einer Ereignisgesteuerten Prozesskette betrachtet werden, welche als Ausgangspunkt für die funktionale Kapselung eines Prozesses dient. Die grundlegenden Elemente einer Ereignisgesteuerten Prozesskette sind Funktionen und Ereignisse, Kanten und Verknüpfungsoperatoren. Für die grundsätzliche Kapselung der Funktionalitäten sind vorrangig die Funktionen ausschlaggebend, wobei die Ereignisse mögliche Ergebnisse einer Funktion charakterisieren oder wichtige Teilaspekte einer Funktion hervorheben.

Neben den Funktionen und Ereignissen spielen auch noch die Kanten eine entscheidende Rolle, da diese Funktionen und Ereignisse abwechselnd miteinander verknüpfen. Erzeugt eine Funktion mehr als ein Ereignis oder lösen mehrere Ereignisse eine Funktion aus, so werden diese Sachverhalte mittels Verknüpfungsoperatoren modelliert (siehe Anhang A).

5.2.2.1. Identifikation identischer Muster

Bei der Analyse der eEPKs müssen zwei Dinge betrachtet werden. Zuerst werden die Ereignisgesteuerten Prozessketten des Prozesses dahingehend analysiert, ob sich Muster wiederholen. Dabei ist es wichtig, dass das identifizierte Muster sich nicht nur graphisch wiederholt, sondern auch vom gleichen Prozess bzw. von der Organisationseinheit ausgeführt wird. Ist dies nicht der Fall, so muss zuerst geprüft werden, ob Prozess- oder Organisationsspezifika berücksichtigt werden müssen.

Ebenso spielt der Kontext des Musters eine große Rolle. Bereits das Erfragen einer eindeutigen Identifikationsnummer an mehreren Stellen im EPK muss nicht zwingend den gleichen Kontext haben. Beispielsweise könnte an einer Stelle die Verantwortlichkeit für das Erteilen dieser Identifikationsnummer beim Incident Management liegen und an einer anderen Stelle in die Verantwortlichkeit des Problem Managements fallen. Beim Identifizieren gleicher Muster ist es deshalb enorm hilfreich, wenn bereits beim Erstellen der EPKs darauf geachtet wird, dass nur oberflächlich ähnliche Muster bereits im Vorfeld graphisch unterschiedlich dargestellt werden. Im einfachsten Fall lässt sich dies bereits durch eine unterschiedliche Formulierung bei der Benennung des Elements erreichen. Im Umkehrschluss sollten gleiche Funktionen oder Ereignisse dementsprechend auch die gleiche Benennung tragen.

Zur Verdeutlichung betrachte man die Funktionalitäten „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“ in Abbildung 4.11, bei denen durch die unterschiedlichen erzeugten Ereignisse mögliche Unterschiede bzw. erweiterte Aufgaben hervorgehoben sind. Im Falle dieser EPK-Modellierung wurde hervorgehoben, dass im Rahmen der „Eventkorrelation“ die betroffenen Incidents auch in gewisser Weise verknüpft werden müssen. Da dies bei der Funktion „Datenkorrelation“ nicht explizit als Ereignis angegeben wurde, kann dies ein Hinweis darauf sein, dass dies im Gegensatz dazu nicht notwendig oder auf eine andere Art zu realisieren ist als bei der „Eventkorrelation“.

Bei der Identifizierung von Mustern spielt auch die Entscheidung eine wichtige Rolle, wie bestimmte Sachverhalte, z.B. die Priorität eines Incidents zu ermitteln, modelliert wurden. Zum einen könnte man für die Ermittlung einer Priorität eines Standard-Incidents und eines Nicht-Standard-Incidents zwei unterschiedliche Funktionen modellieren, da für den Standard-Incident u.U. weniger Input benötigt wird oder Dringlichkeit und Auswirkung fest vorgegeben sind. Andererseits könnte man sich auch vorstellen, dass man die (evtl. unterschiedliche) Behandlung der verschiedenen Incidents in einer Funktionalität zusammenfasst. Im Falle des Incident Managements wurde das Ermitteln der Priorität eines (Standard-)Incidents in einem Teilprozess, und das unterschiedliche Verhalten auf diese ermittelte Priorität außerhalb dieses Prioritätsprozesses modelliert (siehe Abb. 4.9). Werden bereits bei der Entwicklung der EPKs wiederkehrende Muster in separaten Teilprozessen modelliert, so stellt dies bereits eine funktionale Kapselung im Sinne dieser Arbeit dar.

Wird ein Muster identifiziert, so kann es zu einer funktionalen Kapsel zusammengefasst werden. Grundsätzlich kann die Struktur dieses Musters von einer einzelnen Funktion bis hin zu einer komplexen Abfolge von Funktionen variieren. Im Falle einer komplexen Abfolge muss geprüft werden, ob innerhalb dieser Kapsel eine erneute Kapselung möglich ist. Abbildung 5.3 zeigt einen Beispiel-Prozessstrang mit den identischen Funktionen A und B, welche auch in einem separaten Teilprozess AB modelliert sein könnten, der an entsprechender Stelle über Prozesswegweiser referenziert wird. Unter der Annahme, dass für sie der gleiche Kontext gilt, können die Funktionen in ihrer Abfolge in einer funktionalen Kapsel zusammengefasst werden. Wie in Abbildung 5.4 dargestellt, beschränkt sich die Mustersuche jedoch nicht auf einen Prozessstrang, sondern bezieht sich auf den gesamten zu modellierenden Prozess.

Beispiel

Ein identisches Muster bezogen auf den Incident-Management-Prozess ist in Abbildung 5.5 zu sehen. Das Ermitteln der Priorität, das sowohl in Abbildung 4.8 als auch in 4.12 durch einen Prozesswegweiser dargestellt ist, kann funktional gekapselt werden, da der in Abbildung 4.9 dargestellte Prozessstrang bereits so modelliert wurde, dass er für beide Situationen verwendet werden kann, auch wenn unterschiedliche Ereignisse erzeugt werden. Ein möglicherweise weiteres Muster stellt die Funktion „prüfen, ob Warnmeldung ausgelöst werden muss“ dar (siehe Abb. 4.7 und 4.14). Diese Funktion könnte, da sie mehrfach im Prozess auftaucht, analog zu „Priorität ermitteln“ funktional gekapselt werden, auch wenn die Kapsel dann bezogen auf die gewählte Modellierungstiefe der EPKs nur eine einzige Funktion beinhalten würde.

Funktionen, welche den gleichen Bezeichner tragen, müssen nicht zwingend auch für die gleichen durchzuführenden Aufgaben stehen, da die Bezeichner einer Funktion nicht eindeutig sein müssen. Ein Beispiel, das diesen Sachverhalt verdeutlicht, ist die Funktion „Eskalationsroutinen einleiten“ in Abbildung 5.7. Im ersten Fall steht diese Funktion einfach für das Weiterleiten an das nächste Support-Level (funktionale Eskalation), wohingegen sie im zweiten Fall z.B. die Erweiterung von Befugnissen im Rahmen einer hierarchischen Eskalation beinhaltet und somit nicht dasselbe Muster darstellt.

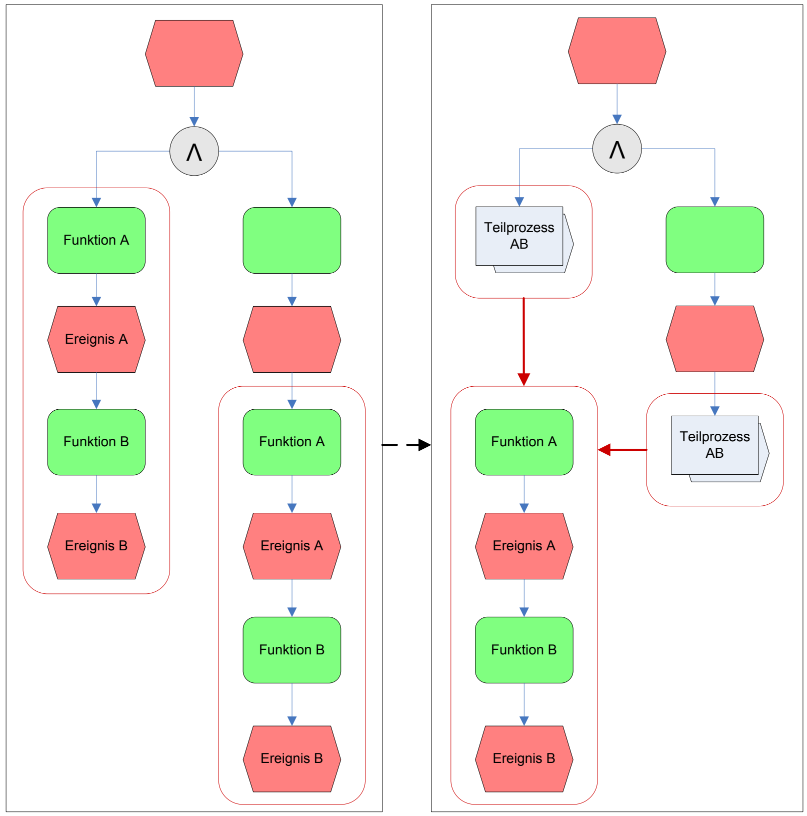


Abbildung 5.3.: Kapsel für identische Muster

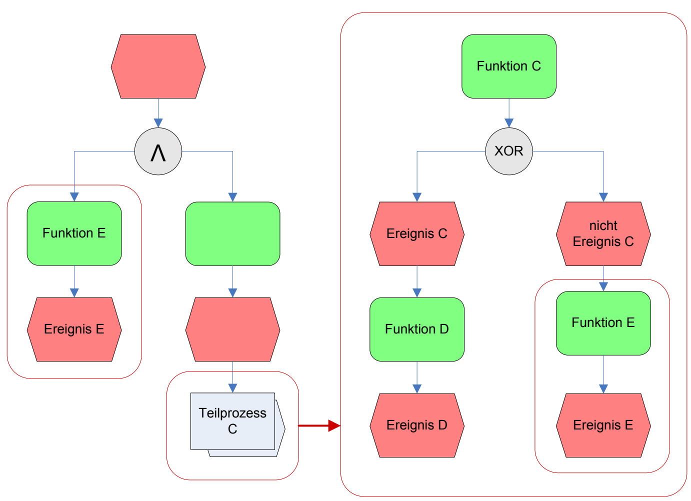


Abbildung 5.4.: Erneute Musterprüfung ggf. auch kapselübergreifend

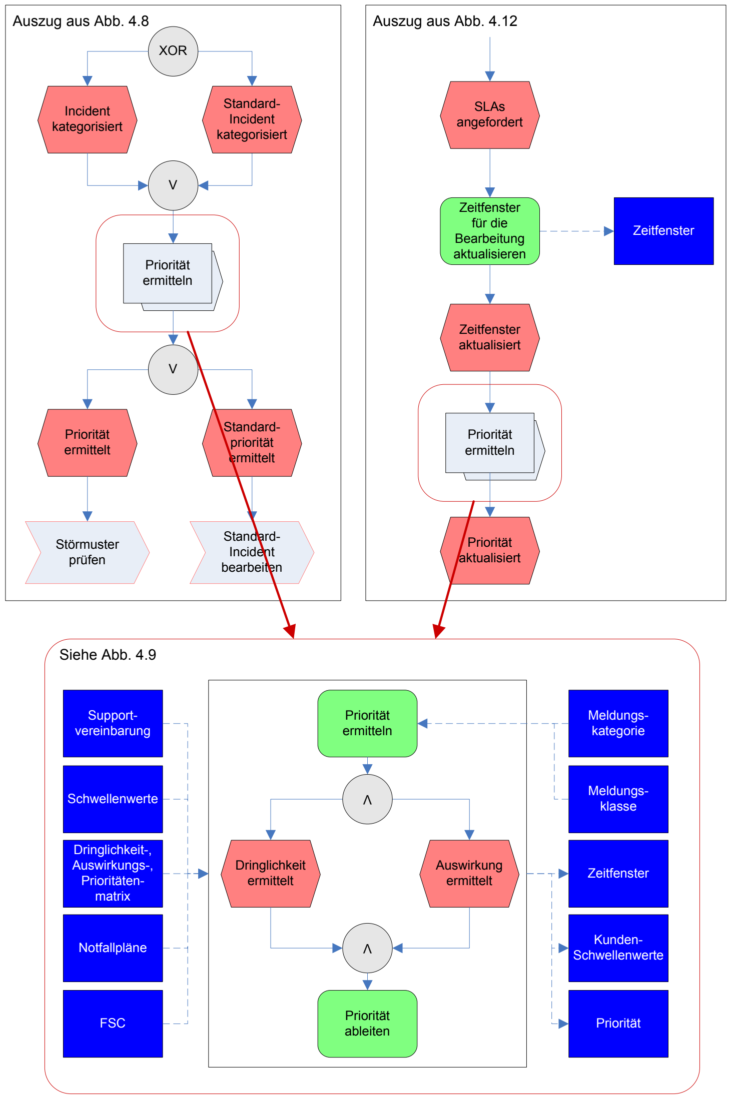


Abbildung 5.5.: Muster im Incident Management (siehe auch Abb. 4.8, 4.9 und 4.12)

5.2.2.2. Verknüpfungen analysieren

Neben dem Identifizieren von Mustern liefert die Art der Verknüpfung ein weiteres Indiz für die funktionale Kapselung. Wurde bei der Verknüpfung ein Verknüpfungsoperator verwendet, so bieten sich für die funktionale Kapselung zwei Möglichkeiten. Zum einen kann für den aufgespaltenen Prozessstrang eine große funktionale Kapsel gebildet werden, welche bis zur Wiedervereinigung oder Reduktion der Prozessstränge auf einen Strang reicht. Zum anderen kann für jeden Teilstrang eine eigene funktionale Kapsel gebildet werden. Wird jeder Teilstrang separat funktional gekapselt, so können die einzelnen Kapseln u.U. parallel ausgeführt werden, was bei der Abarbeitung zu Zeiteinsparungen führen kann und deswegen durchaus wünschenswert ist. Abbildung 5.6 zeigt einen Beispiel-Prozessstrang mit einer Funktion, welche über eine Oder-Verknüpfung drei Ereignisse erzeugen kann. Entsprechend dem zuvor beschriebenen Vorgehen kann die Prozesskette zum einen komplett einer funktionalen Kapsel zugeordnet (Abb. 5.6 links) oder für jeden Teilstrang eine eigene Kapsel angelegt werden (Abb. 5.6 rechts).

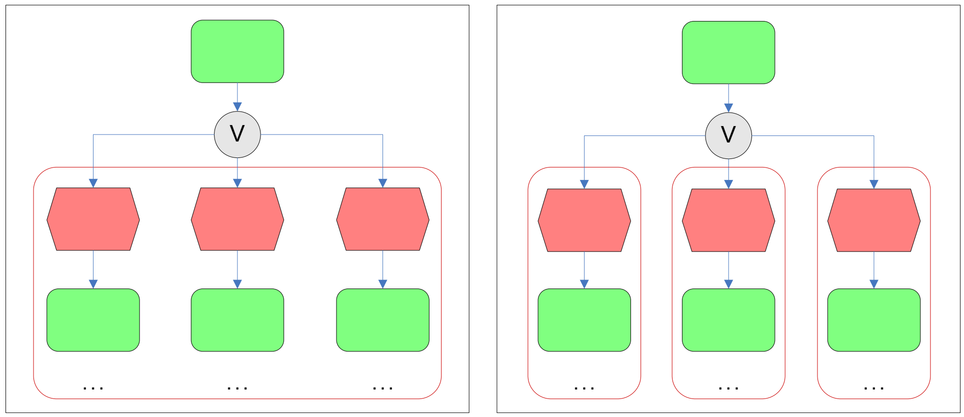


Abbildung 5.6.: Kapsel für gesamten Strang (links) oder pro Teilstrang (rechts)

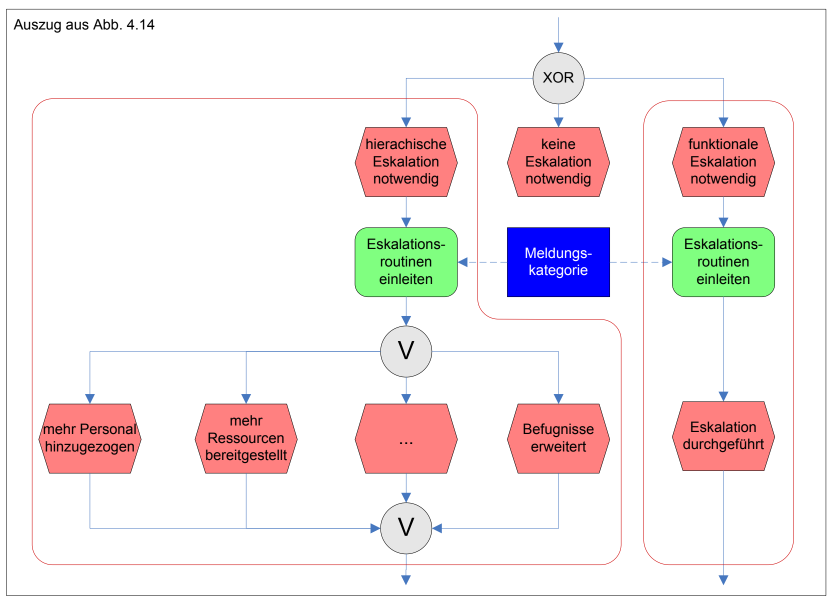


Abbildung 5.7.: Beispiel-Kapselung pro Teilstrang (siehe auch Abb. 4.14)

Beispiel

Die links in Abbildung 5.6 dargestellte Variante, alles einer funktionalen Kapsel zuzuordnen, bietet sich insbesondere bei solchen Prozesssträngen an, die sich nicht verzweigen. Ein Beispiel hierfür ist die Prozesskette in Abbildung 4.12. Die zweite Variante, in der jeder Teilstrang einer eigenen funktionalen Kapsel zugeordnet wird, ist in Abbildung 5.7 zu sehen. Die so gebildeten funktionalen Kapseln trennen die zwei Funktionalitäten, welche in dieser Abbildung zufällig die gleiche Bezeichnung tragen. Der dritte Teilstrang in der Mitte wird keiner funktionalen Kapsel zugeordnet, da eine Kapsel ausschließlich Funktionalitäten zusammenfasst und dieser Teilstrang jedoch nur aus einem Ereignis besteht.

In Abbildung 5.8 wurden beide Varianten gemischt. Dabei wurden zwei nach den Teilsträngen identifizierte funktionale Kapseln[1] zu einer funktionalen Kapsel zusammengefasst. Diese Zusammenfassung ist jedoch nur möglich, da diese beiden Teilstränge durch eine Und-Verknüfung verbunden sind.

[1] siehe funktionale Kapseln mit gestrichelten Linien

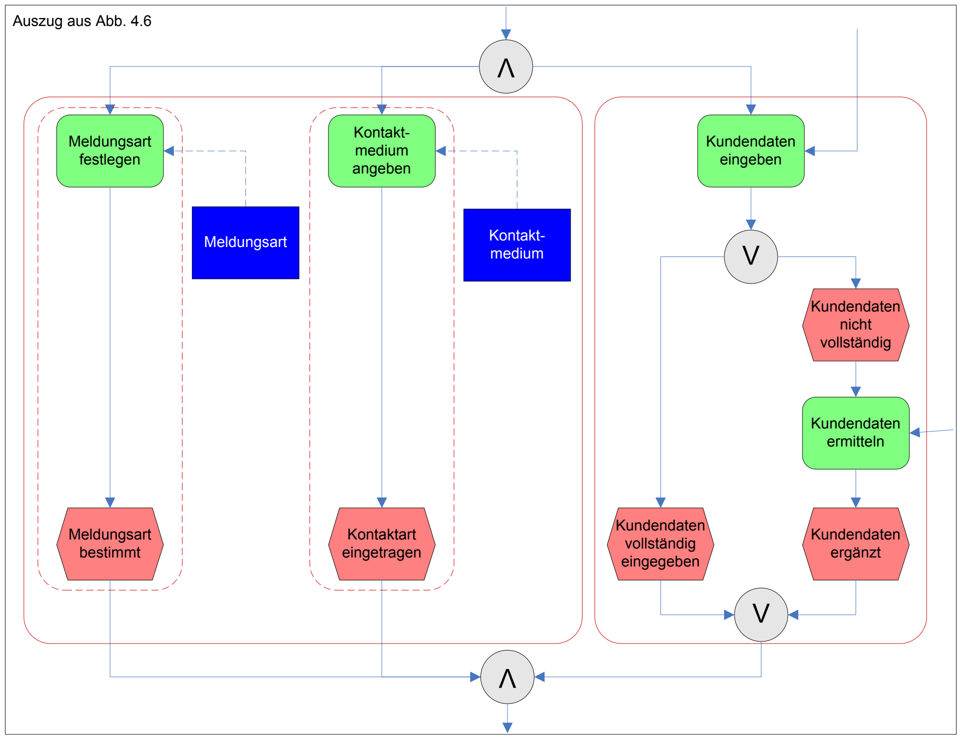


Abbildung 5.8.: Gemischte Beispiel-Kapselung (siehe auch Abb. 4.6)

5.2.2.3. Trennung gemeinsamer Teilabschnitte

Spaltet sich die Prozesskette in mehrere Teilstränge auf und soll für jeden dieser Teilstränge eine funktionale Kapsel angelegt werden, so ist dies u.U. nur schwer realisierbar, weil sich ein oder mehrere Teilstränge einzelne Abschnitte der Prozesskette teilen. Teilen sich alle (nicht abgeschlossenen) Teilstränge den gleichen Abschnitt, so kennzeichnet dies das Ende der ursprünglichen Aufspaltung und somit auch das Ende der funktionalen Kapsel. In Abbildung 5.9 ist eine Beispielprozesskette dargestellt. Die durch die Funktion A verursachten Ereignisse A1 und A2 können beide die Funktion B auslösen. Der Teilstrang, der mit dem Ereignis A3 beginnt, ist noch nicht abgeschlossen, wodurch die dargestellte Oder-Verknüpfung nicht das Ende der ursprünglichen Aufspaltung signalisiert.

Um jedem der Teilstränge eine separate funktionale Kapsel zuzuordnen, können die geteilten Abschnitte, welche zu einer funktionalen Kapsel zusammengefasst werden können, jedem beteiligten Teilstrang zugeordnet werden, sofern die Verknüpfungsart dies zulässt. Dies ist links in Abbildung 5.10 dargestellt. Im Falle einer Und-Verknüpfung ist dies nicht möglich (siehe Abschnitt 5.2.2.4). Beeinflusst das Ereignis der Funktion A die möglichen Ereignisse der Funktion B, so können in jedem Teilstran die jeweils unmöglichen Ereignisse gestrichen werden. Bestimmt ausschließlich das Ereignis dieser Funktion A die nachgelagerten Ereignisse der Funktion B, so kann, wie rechts in Abbildung 5.10 dargestellt ist, u.U. die Funktion B in die Funktionen B1 und B2 aufgespalten werden.

Beispiel

Innerhalb des Incident-Management-Prozesses wird im zweiten Subprozess („Klassifizieren und erste Unterstützung“) die Störungsmeldung klassifiziert, wobei die Meldungen, wie links in Abbildung 5.11 zu sehen ist, in drei Klassen aufgeteilt werden. Anschließend müssen die zwei unterschiedenen Incident-Arten kategorisiert und priorisiert werden.

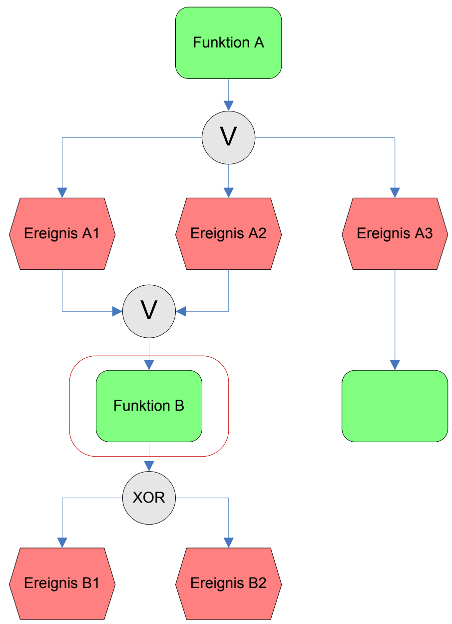


Abbildung 5.9.: Gemeinsamer Teilabschnitt

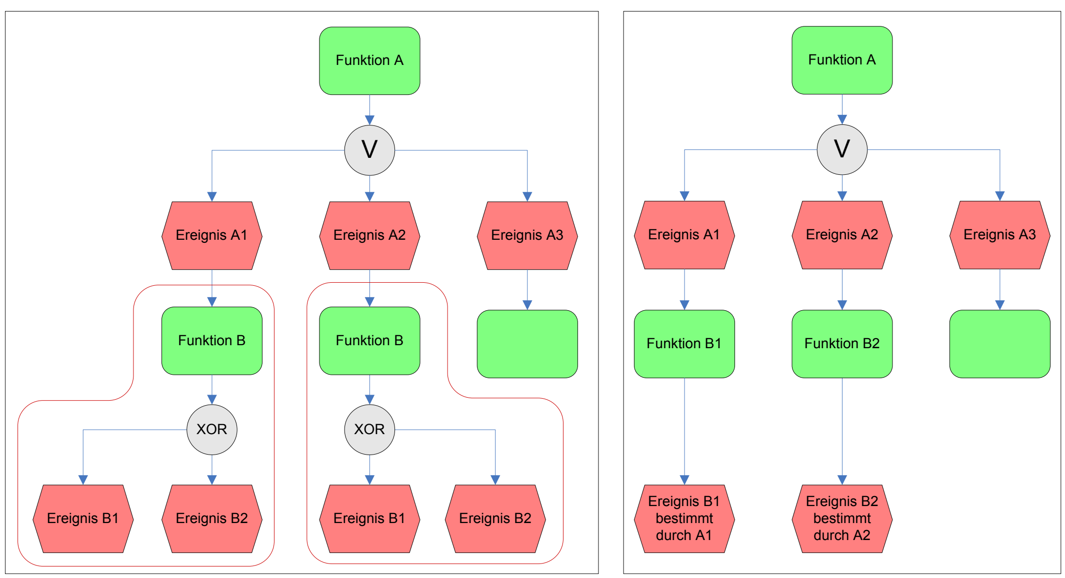


Abbildung 5.10.: Getrennte Teilstränge

Auch für einen User Service Request könnten diese Funktionen durchzuführen sein. Da deren Behandlung jedoch nicht im Vordergrund dieser Arbeit steht, wurden alle Aktivitäten, welche User Service Requests betreffen, in der Funktion „User Service Request bearbeiten“ zusammengefasst. Rechts in Abbildung 5.11 wurden die Funktionen „Incident kategorisieren“ und „Priorität ermitteln“ jedem betroffenen Teilstrang zugeordnet. Außerdem wurden die möglichen Ereignisse dieser Funktionen entsprechend dem Kontext reduziert, da sie zuvor nur der Unterscheidung der verschiedenen Incident-Arten dienten.

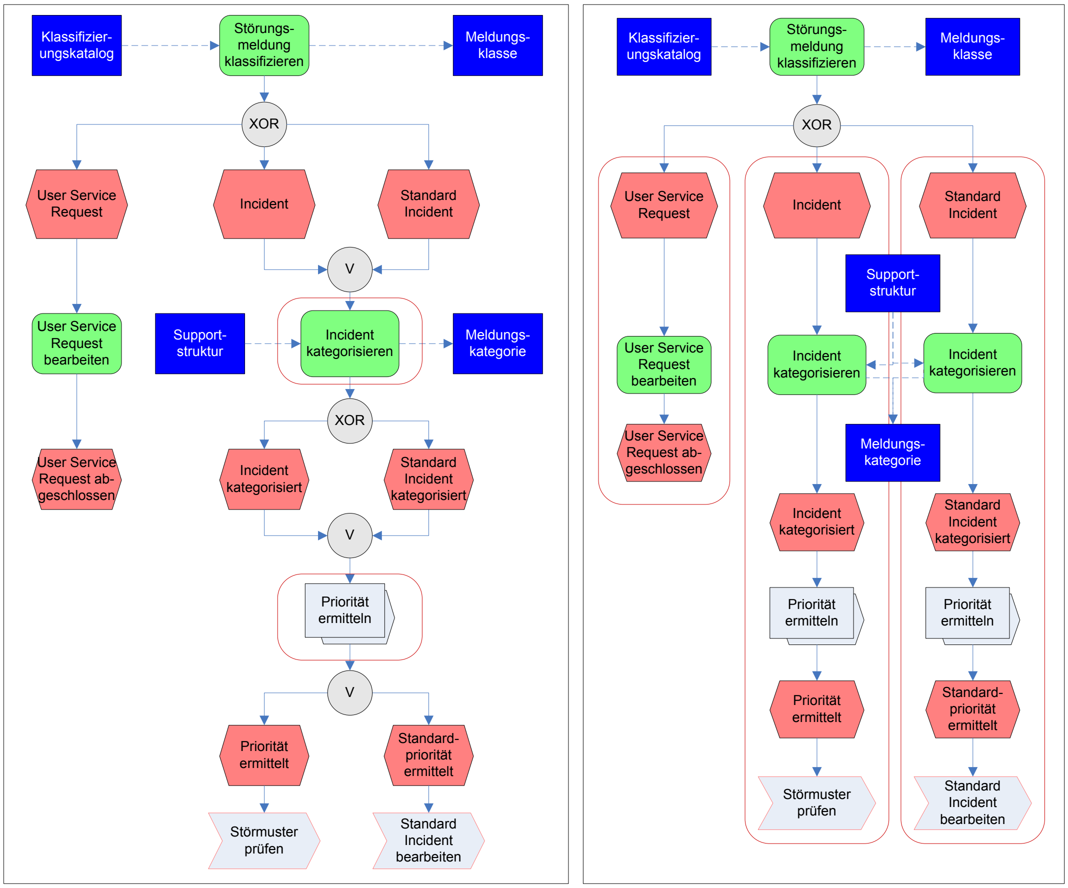


Abbildung 5.11.: Beispiel-Teilstränge trennen (siehe auch Abb. 4.8)

5.2.2.4. Nicht trennbare Teilstränge

Die eben vorgestellte Vorgehensweise anzuwenden, ist besonders in solche Fällen wie rechts in Abbildung 5.10 sinnvoll, oder wenn der geteilte Abschnitt nur eine oder sehr wenige Funktionen umfasst, ansonsten aber unabhängig vorangetrieben wird. Außerdem können die betroffenen Teilstränge nicht immer getrennt werden. Deshalb soll nun eine weitere Vorgehensweise zum Identifizieren von funktionalen Kapseln vorgestellt werden. Anstatt geteilte Abschnitte jedem beteiligten Prozessstrang zuzuordnen, wird hierbei nicht der gesamte Teilstrang einer funktionalen Kapsel zugeordnet, sondern nur der Abschnitt bis zur nächsten Reduktion der Ketten. Für die neu entstehende Prozesskette gelten dann wieder die bereits vorgestellten Möglichkeiten. Ein Beispiel hierfür ist die in Abbildung 5.12 dargestellte Und-Verknüpfung. Da zum Auslösen der dargestellten Funktion C beide Ereignisse A *und* B notwendig sind, kann der gemeinsame Abschnitt nicht jedem Teilstrang separat zugeordnet werden. Die so vereinigten Teilstränge müssen deswegen als ein Teilstrang betrachtet werden, wobei die bis zur Vereinigung separat verlaufenden Abschnitte im Rahmen einer angestrebten Parallelisierung durchaus eigene funktionale Kapseln darstellen können.

Beispiel

Die Funktionen „Dokumentation überprüfen“ und „Incident archivieren“ in Abbildung 4.16 müssen beide ausgeführt werden, bevor der Incident abgeschlossen werden kann. Diese können, wie links in Abbildung 5.13 dargestellt, ebenso wie die Funktion „Incident abschließen“, jeweils separaten funktionalen Kapseln zugeordnet werden.

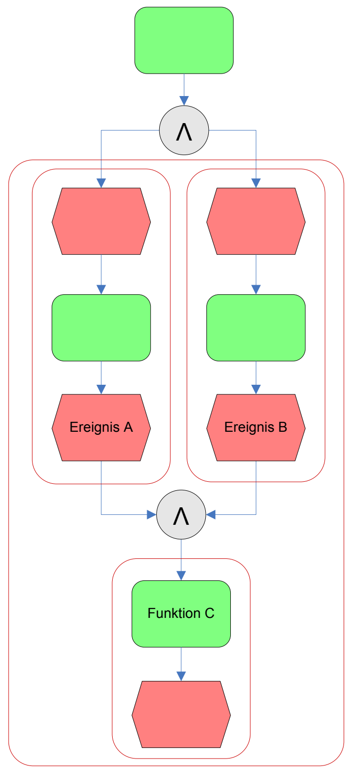


Abbildung 5.12.: Vereinigung durch eine Und-Verknüpfung

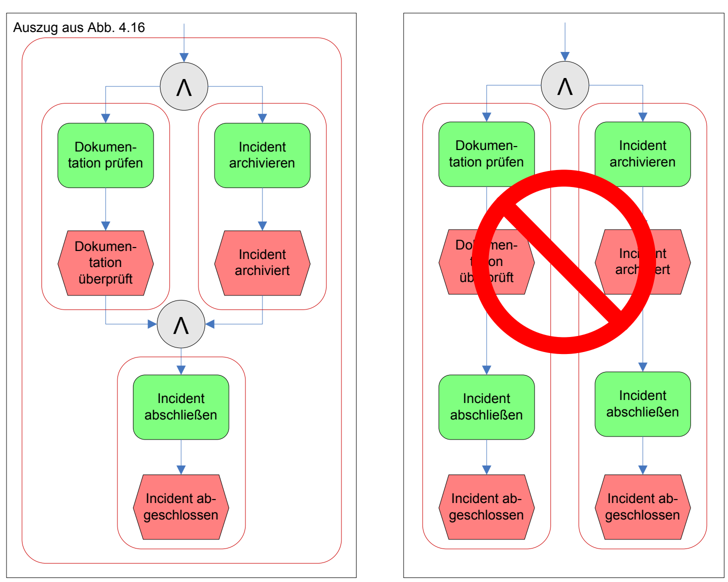


Abbildung 5.13.: Beispiel für nicht trennbare Teilstränge (siehe auch Abb. 4.16)

Auch ein Zusammenfassen des gesamten dargestellten Prozessstrangs sowie Kombinationen davon sind möglich. Nicht erlaubt ist jedoch das Auftrennen der Und-Vereinigung, wie es rechts in Abbildung 5.13 zu sehen ist.

Anhand des beschriebenen Vorgehens wird der gesamte Incident-Management-Prozess in funktionale Kapseln aufgespalten. Dabei erfolgt dies i.d.R. nicht in einem Durchgang. Auch die Schnittstellen und damit die Input- und Outputdaten, welche den funktionalen Kapseln übergeben werden, haben Einfluss auf die Kapselung (siehe auch Abb. 5.2) und werden deshalb im nächsten Abschnitt betrachtet.

5.2.3. Schnittstellenzuordnung

Nachdem in Abschnitt 5.1 bzw. für den Incident-Management-Prozess in Abschnitt 7.1.1 die Schnittstellen für den Prozess als Einheit basierend auf den benötigten Input- und Outputdaten identifiziert wurden, betrachtet dieser Abschnitt diejenigen Schnittstellen, die aus der funktionalen Kapselung des Prozesses resultieren. Ausgangspunkt für die Identifikation dieser Schnittstellen bildet der in seiner Gesamtheit in funktionale Kapseln aufgespaltene Prozess. Durch die Zuordnung der Inputs und Outputs zu den funktionalen Kapseln müssen diese u.U. nochmals angepasst werden.

In Abbildung 5.14 ist eine Prozesskette mit drei funktionalen Kapseln dargestellt. Der Funktion A ist ein Input A zugeordnet. Da sich die Funktion A innerhalb der Kapsel A befindet, wird durch die Kapselung der Input A der Kapsel A zugeordnet und steht dadurch allen Funktionen dieser Kapsel zur Verfügung, also auch der Funktion A2. Analog zur Inputbehandlung wird der Output B der Funktion B ebenfalls der gesamten Kapsel B zugeordnet. Wird der Output B von einer nachgelagerten Funktion derselben Kapsel als Input benötigt, so muss dieser nicht als Input der Kapsel modelliert werden. Die von der Funktion B erzeugten Daten müssen dann nur intern verwaltet bzw. zwischengespeichert werden.

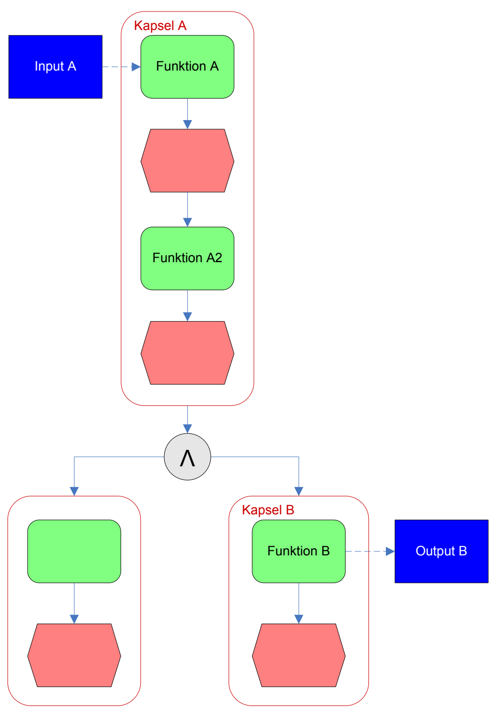


Abbildung 5.14.: Input/Output-Zuordnung (allgemein)

Werden die Kapseln später mittels Services realisiert und das Interface mittels WSDL beschrieben, so wird die Funktionalität der Kapsel hinter einer angebotenen Operation des Interfaces verborgen. Dieser Operation muss dann der gesamte Input für die Kapsel als Teil des *Service Requests* übergeben werden. Der Output einer Kapsel wird im *Service Response* zurückgegeben.

5.2.3.1. Inputbehandlung

Wie bereits erwähnt, wird der Input einer Funktion der gesamten Kapsel zugeordnet und steht deshalb allen Funktionen der Kapsel zur Verfügung. Werden allerdings innerhalb einer Kapsel C Funktionalitäten einer anderen Kapsel D in Anspruch genommen, so steht der Input C nicht automatisch auch der Kapsel D zur Verfügung. Benötigt aber die Funktion D den Input C oder eine Teilmenge C' dieses Inputs, so muss er, wie in Abbildung 5.15 dargestellt, der Kapsel D explizit übergeben werden, auch wenn dies u.U. ursprünglich nicht so modelliert wurde.

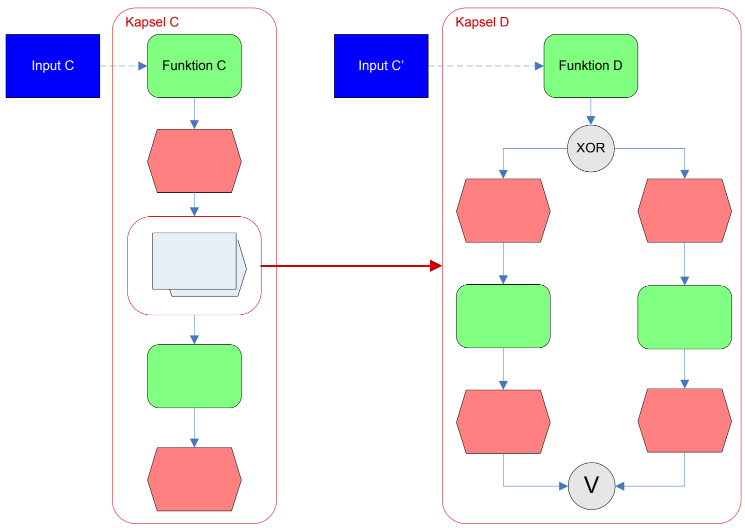


Abbildung 5.15.: Weiterreichen von Inputdaten

5.2.3.2. Outputbehandlung

Eine weitere zu überprüfende Situation ist in Abbildung 5.16 links dargestellt. Zu sehen ist eine Kapsel F, welche eine Funktionalität der Kapsel G verwendet. Die Funktion F liefert einen Output F, der wie eben beschrieben, der Kapsel F zugeordnet wird. Liefert die Funktion G bereits einen Teil G des Outputs F so kann dieser Teil G der Kapsel G zugeordnet werden, wobei der Kapsel F dann nur noch der verbleibende Output F' zugeordnet werden muss, wie rechts in der Abbildung 5.16 zu sehen ist.

Beeinflusst der von der Funktion G gelieferte Output G jedoch den Output der Funktion F, so muss er der umgebenden Kapsel F wieder als Input G zur Verfügung gestellt werden. Dies ist in der Abbildung 5.17 dargestellt. Da der Input G wie zuvor beschrieben jedoch der gesamten Kapsel F zugeordnet werden müsste, er jedoch zu Beginn der Prozesskette noch gar nicht bekannt ist, bestehen in dieser Situation zwei Möglichkeiten der Modellierung. Zum einen kann der Output G der Kapsel G als interner Input G modelliert werden (siehe Abb. 5.18 links). In diesem Fall würde der Aufruf der Kapsel G zwar den Output G liefern, diesen jedoch nicht über die Grenzen der Kapsel F hinaus propagieren. Die Funktion F müsste dann wie ursprünglich modelliert wieder den ursprünglichen Output F (Vereinigung der zuvor aufgespaltenen Outputs G und F') liefern.

Eine andere Möglichkeit der Modellierung ist rechts in Abbildung 5.18 dargestellt. Dabei wurde die ursprüngliche Kapsel F in die zwei Kapseln F1 und F2 aufgespalten. Dadurch kann der Input G der Funktion F und damit der Kapsel F2 zur Verfügung gestellt werden. Diese zweite Methode hat aber u.U. den Nachteil, dass die Gesamtstruktur auf drei Kapseln aufgeteilt ist und an anderer Stelle koordiniert werden muss. Ein etwaiger Makel der ersten Modellierungsvariante ist der, dass der Funktion F Input übergeben wird, welcher u.U. unmodifiziert im Output wiederzufinden ist. In diesem Fall tunnelt die Funktion F die Daten quasi nach außen.

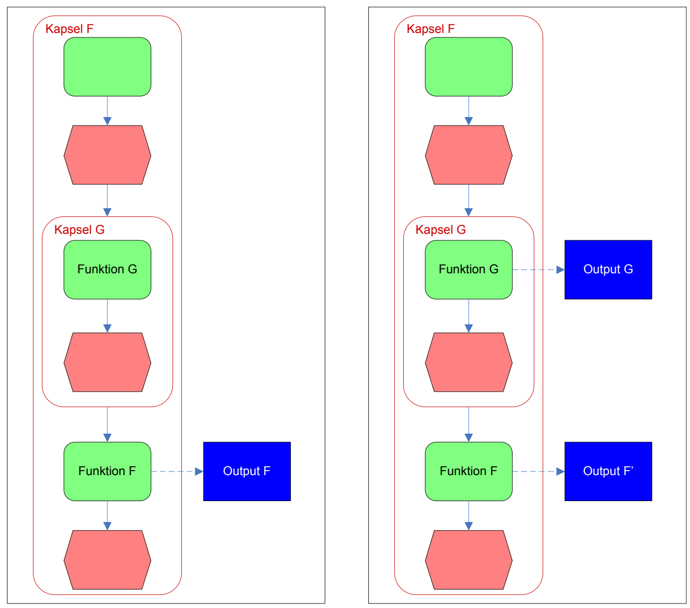


Abbildung 5.16.: Output aufspalten

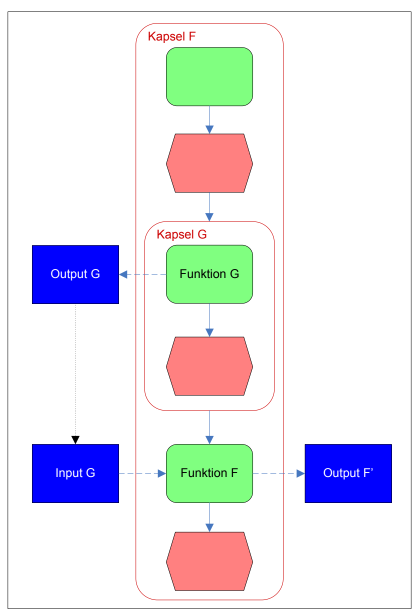


Abbildung 5.17.: Output einer inneren Kapsel verwenden

Wie dieser Abschnitt der Schnittstellenzuordnung verdeutlicht hat, kann der zu modellierende Prozess meist nicht in einem Durchgang in funktionale Kapseln aufgespalten werden. Vielmehr wird die Kapselung des Prozesses schrittweise entwickelt. Dabei wird mit dem Identifizieren vom Mustern begonnen, anschließend erfolgt eine Betrachtung der Prozessstränge, bei welcher die Verknüpfungen beim Aufspalten bzw. der Wiedervereinigung der Teilstränge analysiert werden. Im nächsten Schritt werden dann die bisher identifizierten funktionalen Kapseln im Zusammenhang mit den Schnittstellen nochmals überprüft.

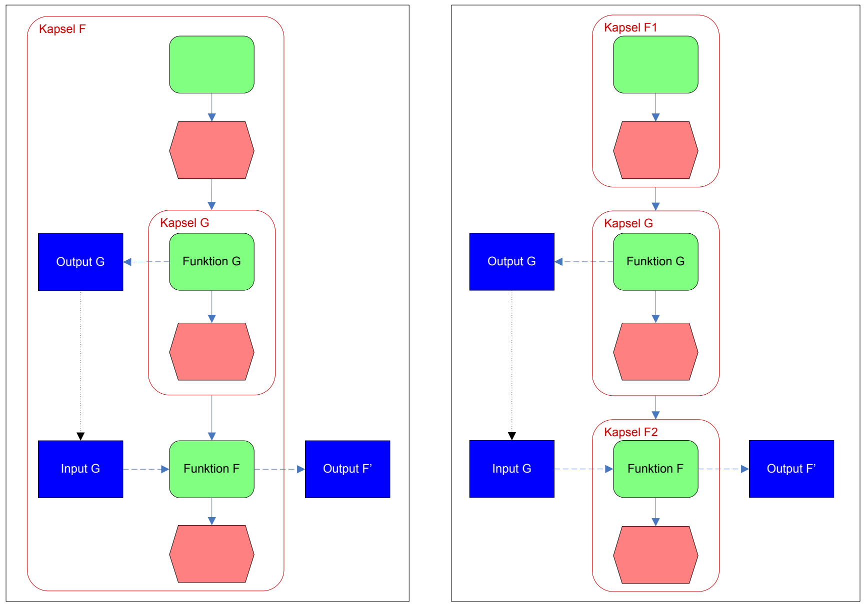


Abbildung 5.18.: Output einer inneren Kapsel als Input verwenden

Das Ergebnis kann bereits als Ausgangssituation für die realisierenden Dienste betrachtet werden. Da diese jedoch im Rahmen dieser Arbeit noch orchestriert werden sollen und die Orchestrierung, wie in Abbildung 5.2 dargestellt, ebenfalls Einfluss auf die funktionale Kapselung hat, befasst sich der folgende Abschnitt mit einer weiteren Anpassung der funktionalen Kapseln.

5.3. Orchestrierung

Durch die Kapselung wurde der Prozess in einzelne Funktionalitäten aufgespalten. Die eigentliche Prozessstruktur und die Prozesslogik sind dabei verloren gegangen, da die einzelnen Kapseln durch lose gekoppelte Dienste realisiert werden können, die in keinerlei Bezug zueinander stehen müssen. Mit Hilfe der Orchestrierung der funktionalen Kapseln soll der Ausgangsprozess wieder „zusammengesetzt“ werden, ohne dass der Aspekt der losen Kopplung, welcher eine zentrale Rolle in einer Service Oriented Architecture darstellt, verloren geht.

Ausgehend von den funktionalen Kapseln des Prozesses werden diese entsprechend der in den EPKs modellierten Logik wieder zusammengesetzt. Im Folgenden wird analog zur funktionalen Kapselung ihre Handhabung in verschiedenen Situationen betrachtet. Wie der Name bereits verdeutlicht, fassen die funktionalen Kapseln Funktionalitäten, also Funktionen zusammen. Neben diesen Funktionen existieren bekanntlich auch noch Ereignisse, welche in der Orchestrierung eine wesentlich wichtigere Rolle spielen als in der funktionalen Kapselung.

Auch wenn in der funktionalen Kapselung Ereignisse wie Funktionen jeweils einer Kapsel zugeordnet werden, gehören sie nicht zwingend zu dieser Kapsel, denn Ereignisse stellen vielmehr die Verbindungsglieder zwischen den gekapselten Funktionalitäten dar, wobei für die Orchestrierung nur diejenigen Ereignisse betrachtet werden müssen, welche Funktionalitäten verschiedener Kapseln miteinander verbinden. Ereignisse, welche innerhalb einer Kapsel zwei oder mehrere Funktionen miteinander verbinden, beschreiben die innere Logik der funktionalen Kapsel, welche durch die Implementierung des realisierenden Service umgesetzt werden muss.

Abbildung 5.19 zeigt zwei einfache Kapselungssituationen. Sowohl rechts als auch links in der Abbildung befinden sich die Funktionen A und B in zwei verschiedenen funktionalen Kapseln A und B. Die Behandlung des Ereignis A wurde bei der Betrachtung der funktionalen Kapselung vernachlässigt, weshalb das Ereignis wie abgebildet sowohl der Kapsel A als auch der Kapsel B zugeordnet werden könnte. Für die Orchestrierung wird das Ereignis A allerdings immer als Ergebnis der vorangehenden Funktion A betrachtet, wie in Abbildung 5.19 rechts zu sehen ist. Das dargestellte Ereignis stellt das Verbindungsglied zwischen Kapsel A und Kapsel B dar. Da sich die Prozesskette in der Abbildung weder aufspaltet noch vereinigt, werden im Rahmen der Orchestrierung die Aktivitäten der dargestellten funktionalen Kapseln einfach hintereinander ausgeführt.

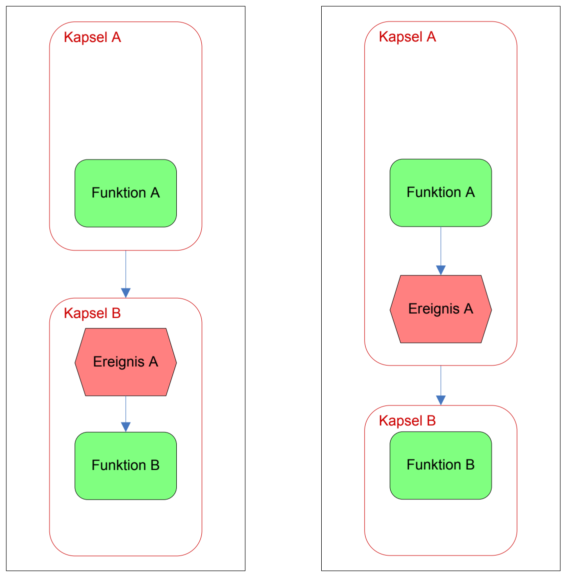


Abbildung 5.19.: Zuordnung eines Ereignisses zu einer funktionalen Kapsel

Neben der in Abbildung 5.19 dargestellten Verbindung zweier funktionaler Kapseln können mehr als zwei Kapseln entsprechend der EPK-Spezifikation (siehe Anhang A) nur über einen Verbindungsoperator miteinander verbunden sein. Dabei werden grundsätzlich vier Situationen unterschieden, welche in Abbildung 5.20 schematisch zusammengefasst sind.

Zum einen kann ein Ereignis mehrere Funktionen auslösen (Abb. 5.20a). Da entsprechend der EPK-Spezifikation in dieser Situation nur die Und-Verknüpfung erlaubt ist, muss sich im orchestrierten Prozess die Aufrufkette ebenfalls verzweigen, indem alle Kapseln der ausgelösten Funktionen, wenn möglich sogar parallel, ausgeführt werden.

Zum anderen können mehrere Funktionen ein gemeinsames Ereignis erzeugen (Abb. 5.20b). In dieser Situation muss prinzipiell jede der betroffenen Funktionen das Ereignis alleine erzeugen können. Entsprechend dem Verknüpfungsoperator muss der orchestrierte Prozess dann entscheiden, ob jede, mindestens eine oder genau eine Funktion dieses Ereignis erzeugen muss, damit die nächste funktionale Kapsel ausgeführt werden kann. Wird bei der Überprüfung der Ereignisverknüpfung festgestellt, dass die geforderte Konstellation nicht erfüllt ist, muss eine zuvor festgelegte Behandlungsroutine ausgeführt werden. Dies könnte im einfachsten Fall z.B. ein Beenden des Prozesses mit einer Fehlermeldung bedeuten.

Im dritten Fall (Abb. 5.20c) kann oder muss eine Funktion mehrere Ereignisse erzeugen, wobei die Funktion prinzipiell alle Ereignisse z.B. in Form spezieller Outputdaten der Kapsel liefern können muss, anhand derer der Prozess den weiteren Verlauf bestimmen kann. Analog zur vorherigen Situation muss dann entsprechend dem Verknüpfungsoperator die Konstellation der Ereignisse geprüft und dementsprechend gehandelt werden.

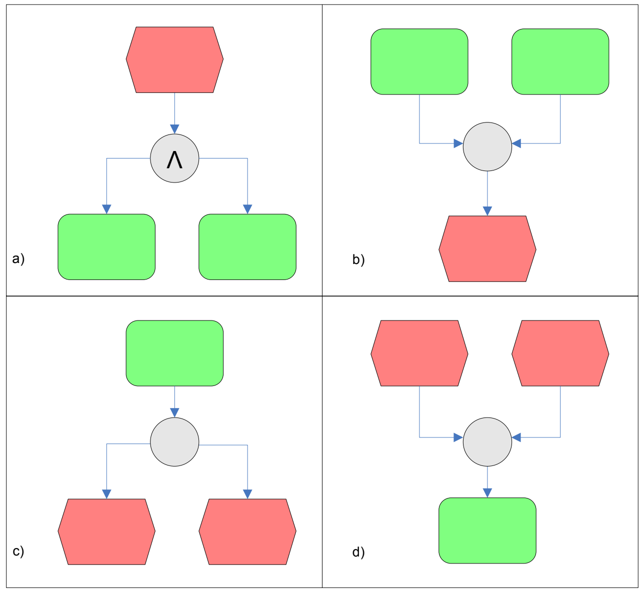


Abbildung 5.20.: Verknüpfungssituationen (schematisch)

Die vierte Situation (Abb. 5.20d) liegt vor, wenn mehrere Ereignisse eine Funktion auslösen, wobei hierbei jedoch nicht entschieden werden muss, welche Funktion und damit welche funktionale Kapsel als nächstes ausgeführt werden muss, sondern nur, ob die Ereigniskonstellation der Verknüpfungsoperation gerecht wird.

Der zweite Einflussfaktor bei der Orchestrierung ist die Behandlung der Input- und Outputdaten. In der Ausführung müssen dem Service, welcher eine funktionale Kapsel realisiert, die geforderten Inputdaten entsprechend seiner Interfacespezifikation übergeben werden. Abbildung 5.21 zeigt eine Konstellation, bei der die Kapsel B für ihre Bearbeitung den von der Kapsel A gelieferten Output oder einen Teil davon benötigt. Da in diesem Fall die benötigten Daten durch die Abfolge der Service-Aufrufe im orchestrierten Prozess zur Verfügung stehen, muss in diesem Fall nur geprüft werden, ob diese Datenformate der beteiligten Services zusammenpassen. Ist dies nicht der Fall, so muss eine entsprechende Transformation vorgenommen werden, bevor die Daten der nächsten funktionalen Kapsel übergeben werden können. Wie diese Datentransformationen genau auszusehen haben, wird vernachlässigt, da diese auf den innerhalb des Unternehmens verwendeten Datenstrukturen und -formaten aufbauen und somit nicht Teil der vorgestellten allgemeine Betrachtung sind.

Links in Abbildung 5.22 ist eine ähnliche Situation dargestellt, welche jedoch dadurch verkompliziert wird, dass die von der Kapsel B benötigten Inputdaten B in die Verantwortlichkeit des Prozesses P fallen und somit dem orchestrierten Prozess selbst nicht zur Verfügung stehen. Unter der bereits getroffenen Annahme, dass jeder weitere beteiligte Prozess benötigte Daten oder Funktionalitäten dem zu modellierenden Prozess über eine Schnittstelle bereitstellt, können diese an entsprechender Stelle erfragt werden. Im Prozess des ITIL Incident Managements entspricht dies z.B. den Konfigurationsdaten oder den Service Level Agreements des Kunden, welche durch das Configuration bzw. Service Level Management verwaltet werden. In dieser Situation muss der orchestrierte Prozess die benötigten Daten B vom Prozess P erfragen, bevor die Funktionalität der Kapsel B ausgeführt werden kann. Dies kann wieder über einen Service realisiert werden.

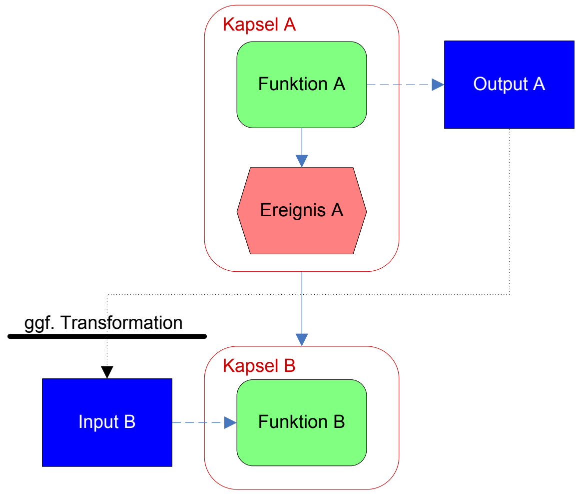


Abbildung 5.21.: Datenübergabe, ggf. mit Transformation

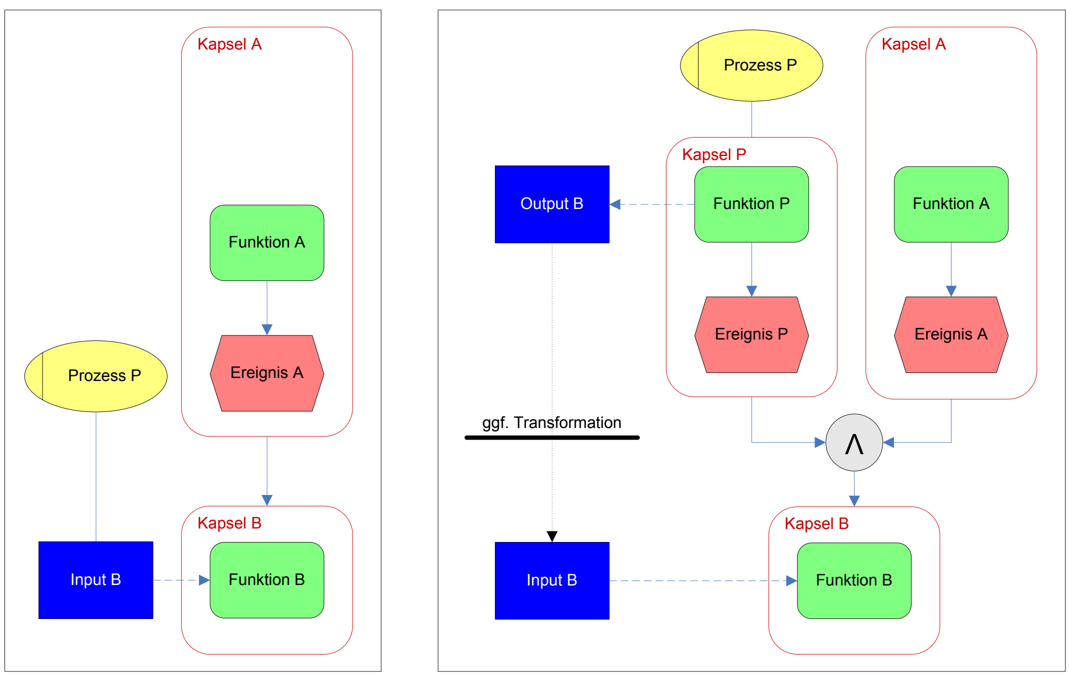


Abbildung 5.22.: Zusätzliche Interaktion zur Datenbeschaffung

Rechts in Abbildung 5.22 ist der soeben geschilderte Sachverhalt graphisch dargestellt, wobei die Kapsel P keine Daten der Kapsel A als Input benötigt, so dass beide parallel aufgerufen werden können. Erst wenn sowohl Kapsel A als auch die Kapsel P abgearbeitet sind, und ggf. die Daten P auf das geforderte Format der Inputdaten B transformiert worden sind, kann die Funktionalität der Kapsel B aufgerufen werden, was durch die logische Und-Verknüpfung der beiden Prozessketten verdeutlicht wird. Für den Fall, dass die Kapsel P Daten der Kapsel A als Input benötigt, wird dieser Aufruf sequenziell zwischen die Kapseln A und B mit eventuellen Datentransformationen geschaltet.

Beispiel

Zur Verdeutlichung der eben beschriebenen Vorgehensweise ist links in Abbildung 5.23 ein Auszug aus dem Subprozess „Analyse und Diagnose“ eines Incidents dargestellt. Der abgebildete Ausschnitt der Abbildung 4.13 zeigt die Funktion „historische Daten überprüfen“, die einer funktionalen Kapsel zugeordnet wurde. Als Input sind neben den Daten der Incident-Datenbank auch die Problem- und Known-Error-Daten aufgeführt. Die Incident-Daten obliegen der Verantwortlichkeit des Incident-Management-Prozesses. Die Problem- und Known Error-Daten fallen jedoch in den Verantwortlichkeitsbereich des Problem Managements und müssen dementsprechend von diesem erfragt werden. Entsprechend dem in Abbildung 5.22 dargestellten Vorgehen wird dieses „Erfragen“ durch eine zusätzlich in den Prozessablauf eingebaute Verknüpfung mit einer weiteren funktionalen Kapsel realisiert, wie rechts in Abbildung 5.23 zu sehen ist. Ähnliches muss auch für die in der Abbildung 4.13 abgebildeten CMDB-Daten vom Configuration Management durchgeführt werden.

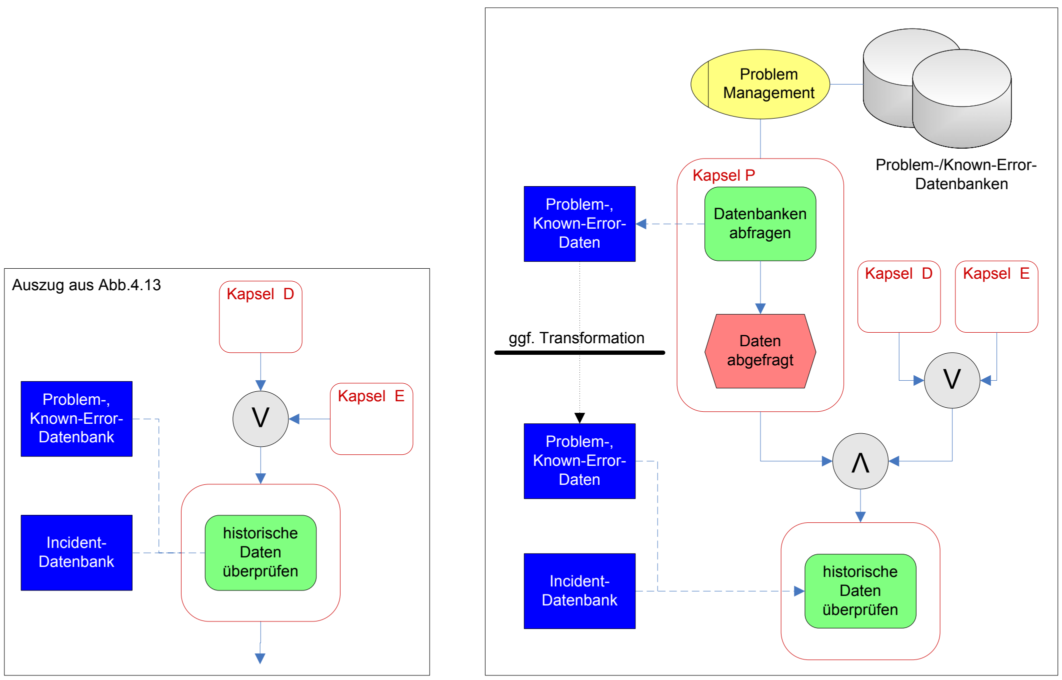


Abbildung 5.23.: Beispiel für zusätzliche Interaktion (siehe auch Abb. 4.13)

Ausgehend von den funktionalen Kapseln, in die ein Prozess entsprechend dem Abschnitt 5.2 aufgespalten wurde, befasste sich dieser Abschnitt mit den Aspekten, welche im Rahmen der Orchestrierung die Gestaltung der funktionalen Kapselung beeinflussen, wobei insbesondere auf die Input-/Outputbehandlung eingegangen wurde. Des Weiteren wurde in diesem Zusammenhang auch auf das eventuell notwendige Durchführen von Datentransformationen eingegangen.

5.4. Zusammenfassung

Nachdem in Kapitel 4 die Prozessanalyse direkt am Beispiel des Incident-Management-Prozesses durchgeführt wurde, hat sich dieses Kapitel mit der zweiten Enticklungsphase, der konzeptionellen Umsetzung der SOA-Idee, befasst. Für die funktionale Kapselung und die anschließende Orchestrierung wurde in diesen Kapitel das Vorgehen zuerst in Form von theoretischen Mustern beschrieben, wodurch das geschilderte Vorgehen allgemein und prozessunabhängig gehalten wurde. Auch zur gewählten Verfeinerungstiefe der EPK-Modellierung wurden keinerlei Annahmen gemacht. Zum besseren Verständnis der identifizierten Muster wurden diese durch Beispiele aus dem betrachteten Incident-Management-Prozess veranschaulicht. Die in den Beispielen aus dem Incident Management dargestellten Vorgehensweisen wurden dabei den gerade betrachteten Mustern des jeweiligen Abschnittes angepasst. Deshalb kann und soll an dieser Stelle nicht ausgeschlossen werden, das sie in Bezug auf den Gesamtprozess u.U. ungünstig sind. Die Ursache dafür liegt darin, dass die vorgestellten Muster sich immer nur auf einen kleinen Ausschnitt beziehen und nicht den Prozess in seiner Gesamtheit betrachten.

Nachfolgend sind noch einmal die betrachteten Aspekte aufgelistet:

- Identifikation identischer Muster (Abb. 5.3 und 5.4)
- Verknüpfungen analysieren (Abb. 5.6)
- Trennung gemeinsamer Teilabschnitte (Abb. 5.9 und 5.10)
- Vereinigung durch Und-Verknüpfung (Abb. 5.12)
- Inputdaten weiterreichen (Abb. 5.15)
- Outputdaten aufspalten (Abb. 5.16)
- Output als Input verwenden (Abb. 5.17 und Abb. 5.18)
- Datenübergabe, ggf. mit Transformation (Abb. 5.21)
- zusätzliche Prozessinteraktion zur Datenbeschaffung (Abb. 5.22)

Da sich das beschriebene Vorgehen insbesondere bei der Auflistung der Muster trotz der allgemeinen Behandlung stark an dem in der Prozessanalyse modellierten Incident-Management-Prozess orientiert hat, wird und soll im Rahmen dieser Arbeit kein Beweis zur Vollständigkeit dieser Muster angetreten werden. Die in [Ahkb 02] definierten „Workflow Patterns" bilden zwar nicht den Ausgangspunkt für die hier beschriebenen Muster, dennoch sind die angestrebten Ziele vergleichbar. Grundlage für die beschriebene Musteridentifikation sind im Rahmen dieser Arbeit die EPK-spezifischen Modellierungsmöglichkeiten in Hinblick auf die angestrebte SOA-basierte Unterstützung des Incident-Management-Prozesses und die spätere Umsetzung mit WSDL und BPEL, welche im folgenden Kapitel betrachtet wird.

Die in diesem Kapitel identifizierten funktionalen Kapseln stellen die späteren Services des Prozesses dar, wobei jedoch nicht zwingend jede Kapsel als eigenständiger Service realisiert werden muss. Vielmehr können kleine, im besten Fall verwandte Kapseln auch in einem einzigen Service zusammengefasst werden, wobei jede der funktionalen Kapseln dann in der WSDL-Interfacebeschreibung über eine eigene Operation angesprochen wird. Dieser Aspekt der Zuordnung einer oder mehrerer funktionaler Kapseln zu einem Dienst der SOA wird in Abschnitt 6.1 im Rahmen der allgemeinen Konzeptumsetzung betrachtet.

Neben den funktionalen Kapseln bilden die Schnittstellen sowie die im Rahmen der Orchestrierung betrachteten Aspekte den Ausgangspunkt für die im nächsten Kapitel beschriebene Umsetzung der SOA durch Web Services, WSDL und BPEL. Analog zu diesem Kapitel wird das in Kapitel 6 geschilderte allgemeine Vorgehen ebenfalls durch Beispiele aus dem Kontext des Incidents Managements veranschaulicht.

Kapitel 6.

Konzeptumsetzung

Nachdem im vorhergehenden Kapitel ausgehend von den während der Prozessanalyse identifizierten Input- und Outputdaten, den Aktivitäten sowie der Prozesslogik die Schnittstellen, die funktionalen Kapseln und die Orchestrierung im Kontext einer Service Oriented Architecture entwickelt wurden, befasst sich dieses Kapitel mit der Umsetzung dieser Aspekte im Rahmen der „SOA-Implementierung“.

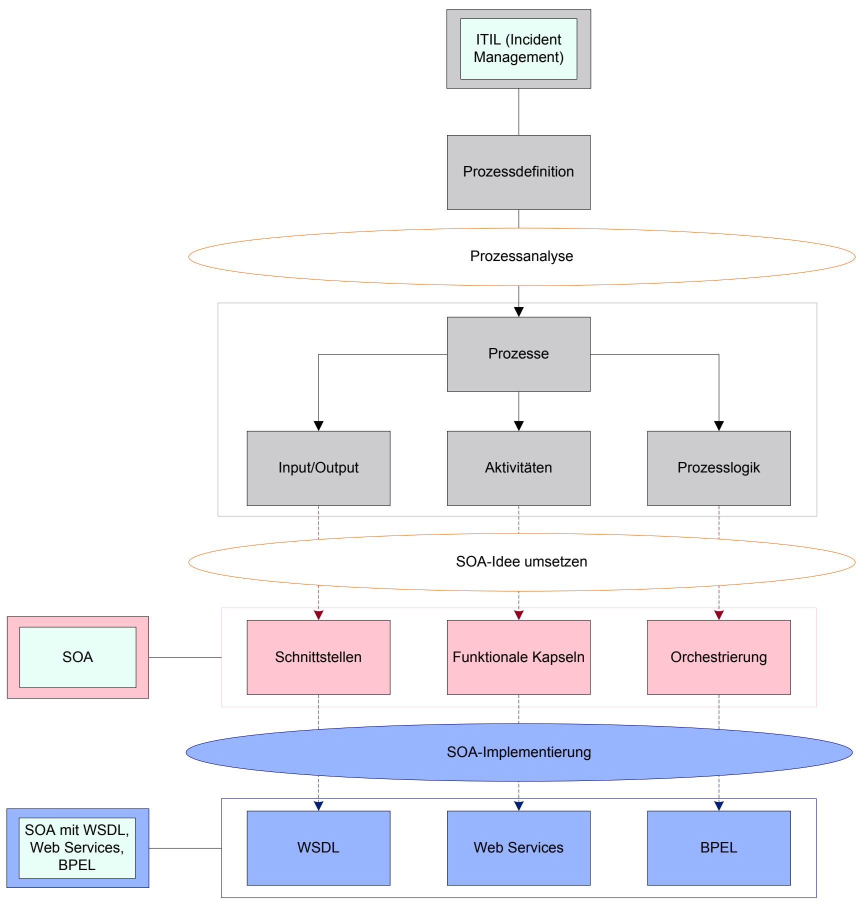


Abbildung 6.1.: Vorgehensweise bei der Konzeptumsetzung

Wie in Abbildung 6.1 dargestellt, ist für die Implementierung der funktionalen Kapseln das Konzept der Web Services vorgesehen. Dementsprechend werden die Schnittstellen zwischen den Diensten mit der Web Service Description Language (WSDL) beschrieben. Für die Umsetzung der Orchestrierung wird die Business Process Execution Language (BPEL) verwendet.

Auch wenn in der Abbildung diese drei Bereiche (WSDL, Web Service, BPEL) getrennt aufgeführt werden, so bilden sie jedoch im Grunde eine Einheit, was insbesondere durch die Abhängigkeit der jeweiligen Schnittstellenbeschreibung vom zugehörigen Web Service verdeutlicht wird. Dies wurde auch im vorangegangenen Kapitel bereits dadurch hervorgehoben, dass die Schnittstellen sowohl als Einflussfaktoren für die funktionale Kapselung (siehe Abb. 5.2) als auch für die Orchestrierung aufgeführt wurden. Aus diesem Grund wird die Umsetzung der Schnittstellenbeschreibung in WSDL innerhalb dieses Kapitels auch nicht in einem separaten Abschnitt behandelt, sondern in Bezug auf die Spezifikation der Web Services und der BPEL-Orchestrierung.

Da im Rahmen dieses Kapitels die Web Services sowie deren Orchestrierung spezifiziert werden sollen, bislang aber nur funktionale Kapseln, deren Schnittstellen sowie die Orchestrierung im Allgemeinen betrachtet wurden, müssen davon ausgehend zuerst die Services der angestrebten SOA identifiziert werden. Der folgende Abschnitt befasst sich aus diesem Grund mit der Gestaltung der SOA in Bezug auf die Zuordnung der funktionalen Kapseln des Prozesses zu den Web Services, welche dann im Anschluss daran spezifiziert werden. Im letzten Teil dieses Kapitels wird dann die BPEL-Orchestrierung der Web Services hergeleitet.

Analog zur Konzeptentwicklung im vorangegangenem Kapitel wird auch in diesem Kapitel aufgrund der Komplexität des Incident-Management-Prozesses das Vorgehen weitgehend unabhängig von diesem beschrieben. Die so vermittelten theoretischen Sachverhalte, welche durch einfache Prozessbeispiele unterstützt werden, bilden dann die Basis für die prototypische Implementierung, in deren Rahmen die vorgestellte Vorgehensweise dann für einen Teil des Incident-Management-Prozesses verwendet wird (siehe Kapitel 7).

6.1. Service-Hierarchie

Bevor die im vorherigen Kapitel identifizierten funktionalen Kapseln in Form von Web Services umgesetzt werden können, müssen die Kapseln erst einmal auf diese aufgeteilt werden. Diese Aufteilung sollte jedoch nicht völlig willkürlich geschehen, weswegen in diesem Abschnitt eine mögliche Vorgehensweise vorgestellt wird.

Im Rahmen der Konzeptumsetzung wird jede identifizierte funktionale Kapsel einem Web Service zugeordnet. Dabei ist es jedoch nicht notwendig, dass jeder Web Service immer nur eine funktionale Kapsel des Prozesses beinhaltet. Durch die Verteilung der in der Konzeptentwicklung identifizierten funktionalen Kapseln auf die Web Services kann eine Service-Hierarchie aufgebaut werden, die umso flacher ist, je mehr koordinatorische Arbeit derjenige Service leisten muss, welcher die Prozesslogik als Ganzes realisiert. Dieser orchestrierende Prozess wird später im Rahmen dieser Arbeit mittels BPEL beschrieben (siehe Abschnitt 6.4).

Abbildung 6.2 zeigt eine vereinfachte Darstellung einer möglichen Service-Hierarchie[1]. Die Web Services wurden entsprechend ihrer Funktionalitäten und Komplexität in drei Hierarchie-Stufen eingeteilt. Die unterste Stufe innerhalb einer SOA, die Basis der Hierarchie, bilden die Basisdienste, welche den übergeordneten Web Services grundlegende Funktionalitäten anbieten, die keine weitere Interaktion mit anderen Diensten erfordern. Im Idealfall erfolgen alle Datenzugriffe z.B. auf die CMDB oder die Incident-Datenbank über solche Services.

Die zweite Stufe der Service-Hierarchie bilden solche Web Services, welche bereits mit anderen Diensten aus der darunter liegenden Schicht interagieren müssen, um die angebotenen Funktionalitäten erfüllen zu können. Sie werden als Indermediarydienste bezeichnet. Prinzipiell können Dienste dieser Stufe natürlich auch direkt auf Daten zugreifen, dennoch ist es für eine „saubere“ Hierarchie sinnvoll, diesen Datenzugriff über die Basisdienste erfolgen zu lassen. Anhand der Interaktionen der Web Services dieser Stufe mit anderen Diensten könnten die Intermediarydienste erneut auf verschiedene Ebenen aufgeteilt werden, worauf an dieser Stelle jedoch verzichtet wird.

[1] siehe auch [KBS 05]

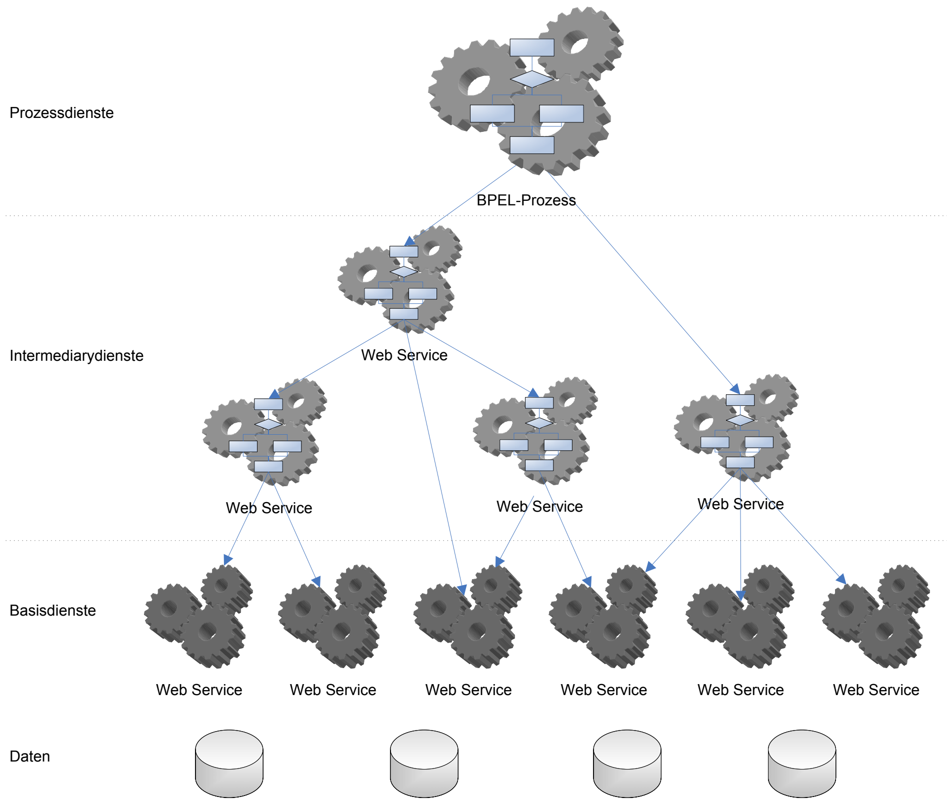


Abbildung 6.2.: Service-Hierarchie

Die oberste Hierarchie-Stufe bilden die sogenannten Prozessdienste, welche auf untergeordnete Dienste zugreifen, den eigentlichen Prozess und dessen Logik umsetzen und in seiner Gesamtheit nach außen repräsentieren. Im Rahmen des entwickelten SOA-Konzepts stehen diese Dienste für die orchestrierenden BPEL-Prozesse, welche natürlich ihrerseits auch Web Services darstellen.

Die eben beschriebene Aufteilung der Web Services orientiert sich an den Funktionalitäten und den Interaktionen mit anderen Web Services, wobei die Verantwortlichkeiten für die einzelnen funktionalen Kapseln bislang unbeachtet geblieben sind. Dieser Faktor darf bei der Zuordnung der funktionalen Kapseln zu den Diensten jedoch nicht außer Acht gelassen werden. Insbesondere sollten keine Funktionalitäten, welche in die Verantwortlichkeit verschiedener Prozesse fallen, durch einen gemeinsamen Web Service realisiert werden, was sich im Speziellen auf die Basisdienste bezieht. Im Beispiel bedeutet dies, dass eine funktionale Kapsel, die zum Abfragen von Daten aus der Incident-Datenbank dient, nicht zusammen mit einer anderen, welche Auskunft über die CIs der CMDB liefert, demselben Web Service zugeordnet werden sollte, da dafür unterschiedliche Prozesse verantwortlich sind. Demnach sollte also pro Prozess mindestens ein Web Service existieren, welcher die angebotenen Funktionalitäten bereitstellt.

Ein weiterer Faktor, der Einfluss auf die Zuordnung der funktionalen Kapseln haben kann, ist die Einordnung der Funktionalität in die verschiedenen Phasen des Gesamtprozesses. Neben der reinen Aufteilung der Funktionalitäten auf die Web Services kann die entstehende Service-Hierarchie dazu verwendet werden, die möglichen Interaktionen zwischen den einzelnen Web Services zu visualisieren, wodurch die spätere Zuweisung von Kommunikationsrollen, insbesondere in Bezug auf die Orchestrierung, vereinfacht wird.

Dieser kurze Abschnitt zum Thema Aufteilen der funktionalen Kapseln auf einzelne Web Services und Etablierung einer Service-Hierarchie sollte dem Leser ein Gefühl dafür vermitteln, dass diese Zuordnung nicht beliebig geschehen sollte, da sie Einfluss auf die Spezifikation der Web Services mittels WSDL und der daran anschließenden BPEL-Orchestrierung hat. Im Rahmen des nächsten Abschnittes wird jedoch zuvor das allgemeine Vorgehen bei der Beschreibung der zu betrachten Spezifikationen vorgestellt und für die Abbildungen eine (Farb-)Semantik zum leichteren Verständnis eingeführt.

6.2. Allgemeines Vorgehen

Bevor die Spezifikation der Schnittstellen, Web Services und der Orchestrierung mittels WSDL und BPEL betrachtet wird, soll dieser Abschnitt das allgemeine Vorgehen verdeutlichen. Dies ist notwendig, da neben der späteren Veranschaulichung an einem Beispiel auch die theoretischen Grundlagen sowie die Abhängigkeiten zwischen einzelnen Aspekten betrachtet werden müssen.

Im Rahmen dieser Arbeit wird WSDL zur Beschreibung einer standardisierten Schnittstelle für die Web Services der SOA eingesetzt. Diese Spezifikation wird um *partnerLinkType*-Elemente erweitert, welche zur Definition von Kommunikationsverbindungen im BPEL-Prozess benötigt werden. Mittels BPEL werden dann die Interaktionen zwischen den Web Services beschrieben. Die in diesem Kapitel betrachteten Spezifikationssprachen sind XML-basiert, so dass sich die Elemente durch eine Baumstruktur veranschaulichen lassen.

Zum besseren Verständnis der Abbildungen in diesem Kapitel werden für die Elemente der jeweiligen Spezifikation verschiedene Unterscheidungen eingeführt. Abbildung 6.3 zeigt eine farblich hervorgehobene semantische Unterscheidung. Elemente, die Teil der WSDL-Spezifikation sind, werden in Abschnitt 6.3.1 vorgestellt. Sie sind links eingezeichnet und können durch ihre gelbe Farbe leicht von den grünen BPEL-Elementen aus Abschnitt 6.4 unterschieden werden, die rechts abgebildet sind. Die dritte Klasse von Elementen, die mittig in Abbildung 6.3 zu sehen ist, umfasst diejenigen Elemente, welche innerhalb des WSDL-Files eines Web Service spezifiziert werden, aber nicht Teil der Web Service Description Language sind. Da diese Elemente als eine Art Bindeglied zwischen der WSDL- und der BPEL-Spezifikation fungieren, werden sie in einem Gelb-Grün dargestellt. Sie werden ebenfalls in Abschnitt 6.4 betrachtet.

	Spezifikation in WSDL	Spezifikation in WSDL (aber kein Teil von WSDL)	Spezifikation in BPEL
Tag	*WSDL operation*	*partnerLinkType*	*BPEL variables*
Element (keine Tag-bezeichnung)	*Datenelement*		*Aktivität*

Abbildung 6.3.: farbliche Unterscheidung und Semantik der Abbildungselemente

Neben der rein farblichen Unterscheidung ist in Abbildung 6.3 auch eine semantische Differenzierung basierend auf den Bezeichnern für die WSDL- und BPEL-Elemente dargestellt. Je nachdem, ob der Bezeichner „WSDL“ oder „BPEL“ enthält, steht das abgebildete Element für ein in der jeweiligen Spezifikation vorkommendes Tag. Der Bezeichner „WSDL operation“ steht also für das *operation*-Element der WSDL-Spezifikation, wohingegen das Element „Datenelement“ nur stellvertretend für ein Element steht, welches Daten wie z.B. einen String in XML Schema repräsentiert, ohne das die genaue Tagbezeichnung angegeben wird.

In Abbildung 6.4 ist anhand des *variables*-Element der BPEL-Spezifikation eine „Vater-Kind“-Beziehung abgebildet. Für das dargestellte *variable*-Element bedeutet dies, dass es innerhalb eines umgebenden *variables*-Element vorkommt. Eine andere Art der Beziehung ist die Referenzierung. Für das in der zweiten

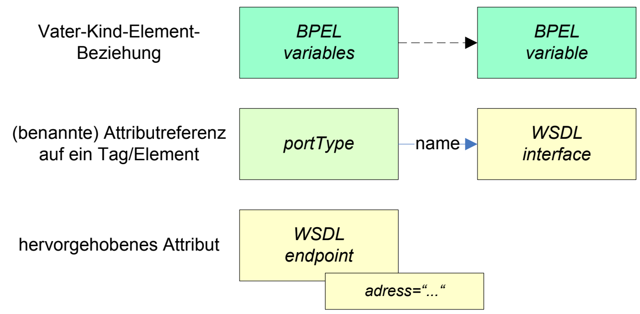


Abbildung 6.4.: Beziehungen zwischen Abbildungselementen

Zeile abgebildete Beispiel ist die Referenzierung folgendermaßen zu verstehen: Das *portType*-Element referenziert über das *name*-Attribut auf ein WSDL-*interface*-Element. Ist die Referenzierung eindeutig oder verbergen sich hinter der Referenz mehrere Instanzen dieser Beziehung, welche nicht weiter unterschieden werden sollen, so wird auf die Benennung der Referenz verzichtet. Sollen einzelne Attribute eines Elements besonders hervorgehoben werden, dann wird neben dem Element selbst auch das entsprechende Attribut gelistet wie im Beispiel mit dem *endpoint*-Element und dessen *adress*-Attribut dargestellt.

Im Rahmen der jeweiligen Beschreibung sieht das allgemeine Vorgehen wie folgt aus: Zuerst wird das jeweilige Element theoretisch anhand seiner Grammatik betrachtet und dazu einige Aspekte wie z.B. wichtige Attribute und ihre Funktion erklärt. Anschließend werden die Abhängigkeiten, die durch Referenzierung entstehen, graphisch dargestellt, bevor dann die Veranschaulichung an einem Beispiel erfolgt.

Nachdem die wichtigsten Elemente und Beziehungen erklärt und das allgemeine Vorgehen bei der Spezifikation vorgestellt wurden, folgt in den nächsten Abschnitten die Spezifikation der Web Services und der Orchestrierung. Abbildung 6.5 zeigt eine Übersicht über die dabei zu betrachtenden Elemente sowie ihre Einordnung innerhalb dieses Kapitels.

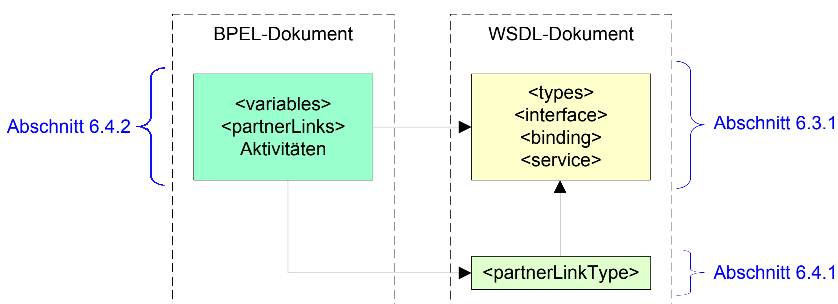


Abbildung 6.5.: Einordnung der Spezifikationen in die Kapitelgliederung

Wie Abbildung 6.5 zu entnehmen ist, werden drei Spezifikationsaspekte in diesem Kapitel unterschieden. Begonnen wird in Abschnitt 6.3 mit der Spezifikation der Web Services. Dabei wird auf die WSDL-Schnittstellenbeschreibung für die identifizierten Web Services eingegangen. Abschnitt 6.4 befasst sich mit der Umsetzung der Orchestrierung. In diesem Zusammenhang wird zunächst u.a. das *partnerLinkType*-Element betrachtet. Dieses wird, wie in Abbildung 6.5 dargestellt ist, dem WSDL-Dokument zugeordnet, aber nicht in WSDL spezifiziert. Im Anschluss daran werden die BPEL-Elemente betrachtet, welche zur Beschreibung der Orchestrierung benötigt werden.

6.3. Web Services

Nachdem die funktionalen Kapseln des Prozesses entsprechend den Kriterien aus Abschnitt 6.1 auf die Dienste der Service Oriented Architecture aufgeteilt wurden, können die identifizierten Web Services spezifiziert werden. Die Spezifikation eines Web Services umfasst die Aspekte Schnittstellenspezifikation und Implementierung. Im Rahmen dieses Konzepts steht die Spezifikation der Web Service-Schnittstelle im Vordergrund, weswegen diese auch ausführlicher betrachtet wird, auch wenn die Implementierung des Web Service für die Realisierung der angebotenen Funktionalitäten unerlässlich ist.

6.3.1. WSDL-Spezifikation

Da die auf diesem SOA-Konzept aufbauenden Dienste durch Web Services implementiert werden sollen, wird die Schnittstelle mittels WSDL beschrieben. Die Schnittstelle in Form eines WSDL-Files erlaubt es den Diensten, innerhalb einer SOA auf standardisierte Weise miteinander zu interagieren, ohne dass der anfragende Dienst die tatsächliche Implementierung des Web Service kennen muss.

Die Web Services Description Language (WSDL) definiert dabei eine plattform-, programmiersprachen- und protokollunabhängige XML-Spezifikation zur Beschreibung von Web Services. Mittels WSDL können die angebotenen Funktionen, Daten, Datentypen und Austauschprotokolle eines Web Service beschrieben werden. Dabei werden im Wesentlichen Operationen mit ihren Parametern und Rückgabewerten definiert, welche der Web Service nach außen hin anbietet.

Ausgangspunkt für die WSDL-Spezifikation in diesem Abschnitt sind die funktionalen Kapseln, die den Diensten zugeordnet wurden. Das in diesem Abschnitt beschriebene Vorgehen greift sich einen dieser Web Services mit seinen funktionalen Kapseln heraus (siehe auch Abb. 6.6) und leitet für diesen eine Schnittstellenbeschreibung in Form eines WSDL-Files ab. Dazu werden die dafür notwendigen Elemente eines WSDL-Dokuments und ihre Bedeutung betrachtet. Zur Verdeutlichung wird anhand eines Beispiels sukzessive ein WSDL-File aufgebaut und erklärt. Grundsätzlich ist ein WSDL-Dokument ein XML-Dokument, dessen konkrete Syntax bzw. dessen allgemeine Struktur auszugsweise in Anhang B dargestellt ist.

Ein WSDL-File besteht aus folgenden vier grundlegenden Elementen:

- Typbeschreibung (`<types>`-Element)
- Interfacebeschreibung (`<interface>`-Element)
- Bindingbeschreibung (`<binding>`-Element)
- Servicebeschreibung (`<service>`-Element)

Diese Elemente lassen sich in zwei Gruppen unterteilen, eine abstrakte, welche die Typ- und Interfacebeschreibung enthält und eine konkrete, die die beiden anderen Elemente der Auflistung umfasst und womit die konkrete Implementierung des Web Services an das Interface gebunden wird.

Neben den aufgeführten Elementen kann ein WSDL-File noch weitere Elemente wie z.B. `<import>` oder `<include>` beinhalten, welche es ermöglichen, das WSDL-Dokument modular aufzubauen. Auch wenn diese mögliche Modularisierung im Rahmen der angestrebten SOA durchaus eine Rolle spielt, wird im Rahmen dieser Arbeit nicht weiter darauf eingegangen. Für die Definition der WSDL-Elemente wird der Namensraum *„http://schemas.xmlsoap.org/wsdl“* verwendet. Zur Beschreibung der Datentypen wird häufig XML Schema und dementsprechend der Namensraum *„http://www.w3.org/2001/XMLSchema“* benutzt.

Bevor die Typ- und Interfacebeschreibung im nächsten Abschnitt betrachtet wird, zeigt Abbildung 6.6 eine graphische Darstellung des Konzepts, welches sich hinter der WSDL-Schnittstellenbeschreibung für einen Web Service verbirgt. Zu sehen ist ein Service Requestor, der mit einem Web Service kommunizieren möchte. Der Web Service bietet innerhalb der SOA Funktionalitäten an, die er z.B. in Form von Java-Methoden implementiert hat. Im Rahmen der Implementierung wurden dabei u.a. die Parameter und

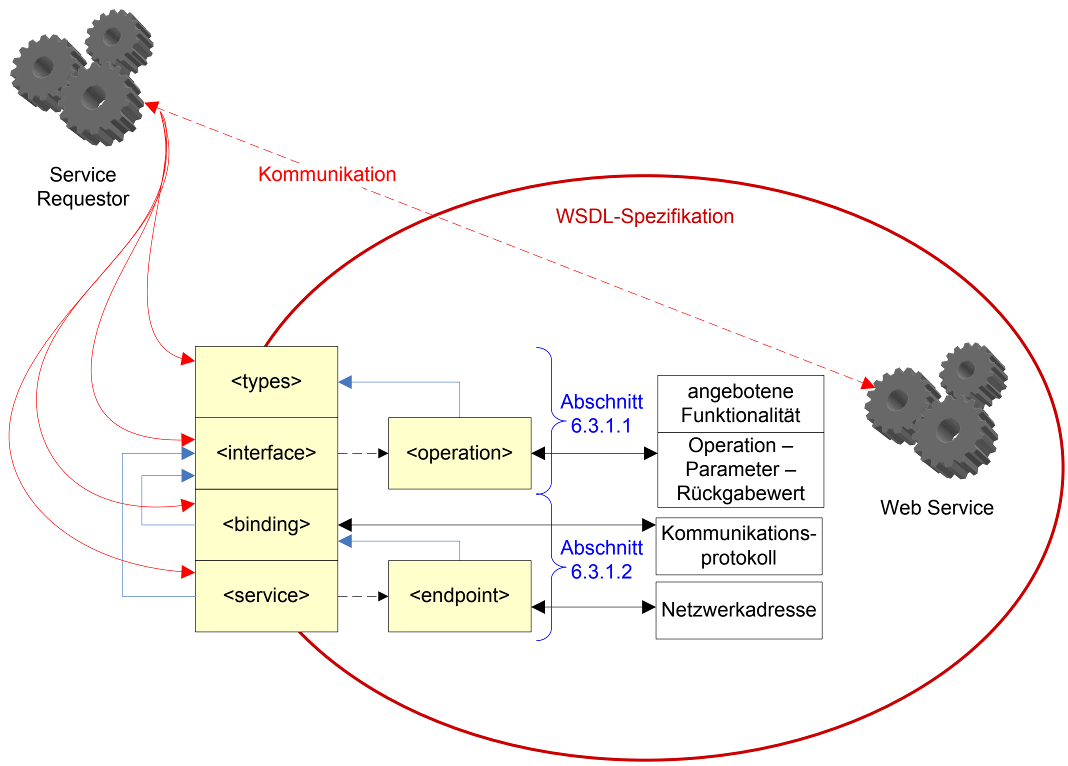


Abbildung 6.6.: Grundkonzept der WSDL-Spezifikation

Rückgabewerte dieser Methode festgelegt. Neben der Implementierung der angebotenen Funktionalitäten ist für eine WSDL-Spezifikation das Kommunikationsprotokoll und die Netzwerkadresse, über die der Web Service kontaktiert werden kann, notwendig, welche symbolisch in Abbildung 6.6 zu sehen sind.

Im Rahmen der WSDL-Spezifikation in diesem Abschnitt werden die Implementierungsdetails des Web Service hinter einer standardisierten Schnittstellenbeschreibung verborgen, welche eine Art Mauer[2] um die Web Service-Implementierung zieht und durch den roten Kreis in Abbildung 6.6 symbolisiert wird. Damit der Service Requestor dennoch die notwendigen Informationen erhält, die für eine Kommunikation mit dem Web Service notwendig sind, werden in der WSDL-Beschreibung verschiedene Elemente definiert.

Die *types*- und *interface*-Elemente sowie die darin enthaltenen *operation*-Elemente einer Interfacebeschreibung werden in Abschnitt 6.3.1.1 eingehender betrachtet. Das *binding*-Element und das *service*-Element mit seinen *endpoint*-Elementen werden in Abschnitt 6.3.1.2 erklärt.

In Zusammenhang mit der Schnittstellenbeschreibung werden die vom Web Service angebotenen Funktionalitäten, unterstützten Kommunikationsprotokolle und Netzwerkadressen auf WSDL-Elemente abgebildet. Diese Abbildung wird durch die schwarzen Pfeile in Abbildung 6.6 dargestellt. Bei der Spezifikation eines *operation*-Elements muss neben den Parametern und Rückgabewerten auch noch die Kommunikationsart und -richtung angegeben werden, obwohl dies i.d.R. durch die gewählte Service-Implementierung vorgegeben ist und nicht frei gewählt werden kann. Da WSDL die Kommunikationsart und -richtung jedoch von der tatsächlichen Implementierung abkoppelt, müssen diese explizit bei der Spezifikation eines *operation*-Elements angegeben werden.

[2]Passender wäre der Begriff „Kapsel“, da die WSDL-Spezifikation die Web Service-Implementierung von der restlichen SOA-Welt abkapselt. Dieser Begriff wird in dieser Arbeit jedoch bereits verwendet.

Zwischen den einzelnen WSDL-Elementen bestehen verschiedene Abhängigkeiten (Attributreferenzen), von denen einige durch die blau dargestellten Pfeile angedeutet sind. Innerhalb der Beschreibung der *operation*-Elemente wird bspw. auf Datentypen im *types*-Element referenziert. Das *binding*-Element referenziert ebenso wie das *service*-Element wiederum auf eine Interfacebeschreibung, und das *endpoint*-Element verweist auf ein *binding*-Element.

Anhand der dargestellten Verweise wird deutlich, dass durch den Wunsch, die Implementierung des Web Services protokollunabhängig zu verbergen, innerhalb der Beschreibung der WSDL-Schnittstelle „alles" (Datentypen, Operationen, Kommunikationsprotokolle und Netzwerkadressen) detailliert spezifiziert werden muss, wodurch jedoch die Implementierung unter Beibehaltung der Schnittstelle jederzeit ausgetauscht werden.

Möchte ein Service Requestor nun mit dem Web Service kommunizieren und eine von diesem angebotene Funktionalität in Anspruch nehmen, so kann er alle dazu notwendigen Informationen der WSDL-Spezifikation des Web Service entnehmen, was in Abbildung 6.6 durch die roten Pfeile angedeutet ist. In größeren SOAs wird zu dieser Zeit häufig ein Register eingesetzt (siehe Abb. 2.5), welches dem Service Requestor entweder direkt das WSDL-File des Web Service liefert oder ihm mitteilt, wo er dieses finden kann. Der Ort des WSDL-Files muss dabei nicht mit dem des Web Service identisch sein.

Nach dieser kurzen Einführung in das grundsätzliche Konzept einer WSDL-Spezifikation werden in den nächsten beiden Abschnitten die vier grundlegenden WSDL-Elemente vorgestellt. In diesem Zusammenhang wird zuerst jeweils deren allgemeine Grammatik vorgestellt sowie einzelne Attribute und Kindelemente bzgl. ihrer Funktionalität bzw. Verwendung genauer betrachtet. Anschließend werden die theoretischen Zusammenhänge an einem Beispiel veranschaulicht, welches innerhalb der nächsten beiden Abschnitte sukzessive aufgebaut wird.

6.3.1.1. Typ- und Interfacebeschreibung

Dieser Abschnitt befasst sich mit der allgemeinen, abstrakten Beschreibung der Web Service-Schnittstelle in Form der Typ- und Interfacebeschreibung. In der Typbeschreibung werden dabei zuerst die abstrakten Datentypen spezifiziert, welche für die angebotenen Operationen als Input bzw. Output verwendet werden. Anschließend werden im zweiten Teil dieses Abschnitts durch die Interfacebeschreibung dann die Signaturen für die Operationen spezifiziert.

Abbildung 6.7 zeigt u.a. die logische Struktur bzw. das Vorgehen, welches diesem Abschnitt zu Grunde liegt. Dargestellt ist ein Web Service, welchem eine Menge von funktionalen Kapseln zugeordnet ist. Basierend auf diesen Kapseln werden für diesen Web Service die Typ- und Interfacespezifikationen in WSDL abgeleitet. Dabei werden vier Kapselvarianten, welche im Zusammenhang mit den Message-Exchange-Pattern der Operationsspezifikation auf Seite 88 beschrieben werden, bei der Modellierung der Interface-Operationen unterschieden, wobei das Unterscheidungskriterium der benötigte Dateninput und -output ist.

Die folgenden Beschreibungen orientieren sich an den Datentypen bzw. funktionalen Kapseln, welche anderen Diensten innerhalb der Service Oriented Architecture angeboten werden und für die Interaktion vonnöten sind. Auf die Spezifikation weiterer Datentypen oder funktionalen Kapseln, die mit dem zu unterstützenden Prozess nicht direkt in Verbindung stehen, wird verzichtet.

Des Weiteren werden klassische Datenstrukturen, z.B. die Struktur eines Incident Records, die den Gesamtprozess betreffen, in diesem Abschnitt nicht explizit betrachtet, da an dieser Stelle die WSDL-Spezifikation eines Services innerhalb des umgebenden Prozesses im Vordergrund steht. Dies bedeutet, dass die Datenstrukturen einer WSDL-Schnittstellenbeschreibung theoretisch völlig unabhängig von anderen Web Services definiert werden können. In Bezug auf eine spätere Zusammenarbeit der Web Services im Rahmen der Orchestrierung sollte die schematische Beschreibung dieser Datenstrukturen jedoch konsistent erfolgen, um unnötige Datentransformationen zu vermeiden.

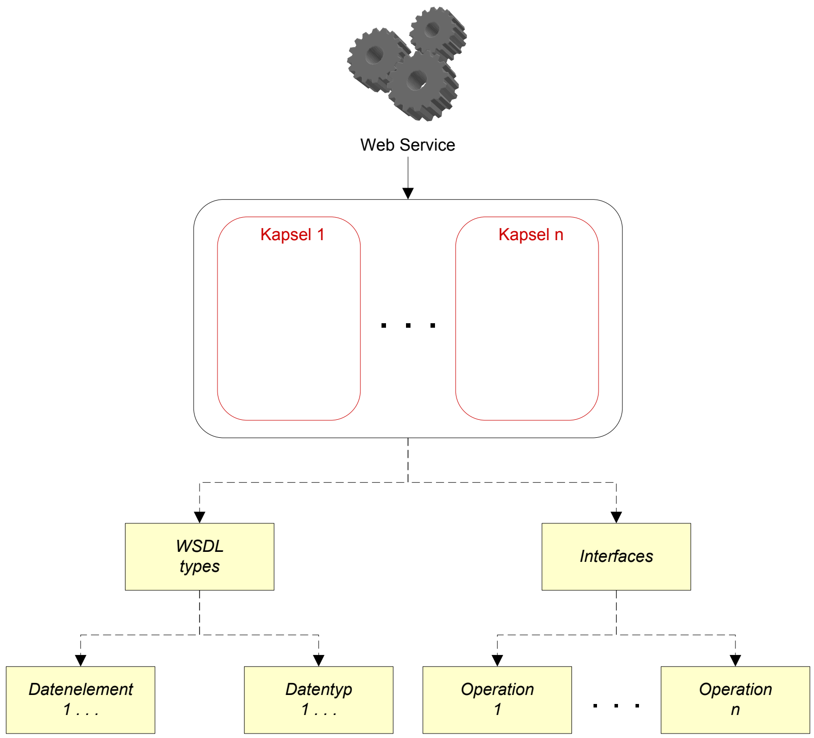


Abbildung 6.7.: Typ- und Interfacebeschreibung

Typbeschreibung

Begonnen wird mit der Spezifikation der Datentypen, welche den funktionalen Kapseln als Input dienen oder von diesen als Output geliefert werden. Nachfolgend ist die allgemeine Grammatik für die Typbeschreibung aufgezeigt:

```
<types>
  <documentation />*
  [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
    <xs:schema targetNamespace="xs:anyURI" /> |
    other extension elements ]*
</types>
```

Bezogen auf den Web Service müssen für jede funktionale Kapsel die Input- und Outputdaten passend zur gewünschten bzw. benötigten Struktur innerhalb des *types*-Elements modelliert werden. Umfasst der zu modellierende Web Service z.B. eine funktionale Kapsel, welche eine Incident-ID liefert, so muss für diese ein Datentyp definiert werden. Dieser könnte in XML Schema folgende Form haben[3]:

```
<types>
  <xsd:schema ...
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="IncidentID" type="Identifikationsnummer" />
    <xsd:simpleType name="Identifikationsnummer">
      <xsd:restriction base="xsd:long" />
    </xsd:simpleType>
  </xsd:schema>
</types>
```

[3]Die Spezifikation von XML Schema ist [W3C 04b] und [W3C 04c] zu entnehmen.

In diesem Beispiel wurde ein Datentyp-Element mit dem Namen „IncidentID“ definiert, welches vom Typ „Identifikationsnummer“ ist. Diese „Identifikationsnummer“ wurde durch einen *simpleType* spezifiziert, welcher auf dem XML Schema-Basisdatentyp „xsd:long“ basiert. Eine mögliche aus dem zu modellierenden Incident-Prozess abgeleitete Forderung, dass diese Identifikationsnummer eindeutig ist, kann an dieser Stelle nicht festgelegt werden. Dafür muss der realisierende Web Service sorgen, welcher diese Identifikationsnummer liefert. Innerhalb der Typdefinition wird ausschließlich das Format für den Datentyp spezifiziert und nicht die Semantik oder mögliche Forderungen. Dies bedeutet ebenso, dass die Benennung der Datentypen keinerlei Rolle spielt. Der gerade definierte Datentyp muss deshalb in der späteren Kommunikation nicht zwingend auch eine Identifikationsnummer darstellen, wie wir sie im Rahmen des Incident Managements vielleicht erwarten. Sie könnte auch die Anzahl der Blumenarten einer Gattung darstellen.

Bei der Modellierung der Datentypen eines Web Service ist es außerdem sinnvoll, diese möglichst so zu gestalten, dass Teilstrukturen, die in verschiedenen Datentypen vorkommen, nur einmal modelliert werden, um Redundanzen und Inkonsistenzen zu vermeiden. Nachdem alle Datentypen aller funktionalen Kapseln eines Web Service spezifiziert sind, können die Operationssignaturen des WSDL-Interface-Elements beschrieben werden.

Interfacebeschreibung

Mit dem *interface*-Element des WSDL-Dokuments wird die Schnittstelle des Web Service durch eine abstrakte Menge von Operationen definiert. Bevor jedoch die Operationen näher betrachtet werden, ist nachfolgend zuerst wiederum die allgemeine Grammatik für die Interfacebeschreibung abgebildet.

```
<interface name="xs:NCName" extends="list of xs:QName"?
  styleDefault="list of xs:anyURI"? >
  <documentation />*
  [ <fault /> | <operation /> | <feature /> | <property /> ]*
</interface>*
```

Wie an dem Stern(*) in der abgebildeten Interface-Grammatik zu sehen, kann jedes WSDL-Dokument beliebig viele *interface*-Elemente beinhalten. Obwohl dies bedeutet, dass ein WSDL-File theoretisch gesehen auch kein *interface*-Element haben kann, gehen wir im Weiteren davon aus, dass das zu spezifizierende WSDL-File mindestens ein *interface*-Element hat, da es für die weitere Untersuchung nur Sinn macht, Web Services zu betrachten, die mindestens eine Operation anbieten, welche im Rahmen der BPEL-Orchestrierung aufgerufen werden kann.

Zur eindeutigen Identifizierung eines Interfaces wird das *name*-Attribut des *interface*-Elements verwendet. Außerdem können die *binding*- und *service*-Elemente, welche im nächsten Abschnitt betrachtet werden, über dieses Attribut das gewünschte *interface*-Element referenzieren.

Die Signatur einer angebotenen Funktionalität wird mit dem *operation*-Element innerhalb des *interface*-Elements spezifiziert. Neben diesem *operation*-Element können noch weitere Elemente wie bspw. ein *fault*-Element zur Fehlerbehandlung spezifiziert werden. Die konkrete Syntax sowie die Bedeutung dieser Elemente und der anderen Attribute des *interface*-Elements sind der WSDL-Spezifikation [W3C 05b] zu entnehmen.

Bevor ausführlicher auf die Modellierung der *input*- und *output*-Elemente eingegangen wird, ist hier zuerst die allgemeine Grammatik für die Operationsspezifikation dargestellt.

```
<interface>
  <operation name="xs:NCName" pattern="xs:anyURI" style="list of xs:anyURI"? >
    <documentation />*
    [ <feature /> | <property /> |
      [ <input /> | <output />   | <infault /> | <outfault /> ]+
    ]*
  </operation>*
  ...
</interface>
```

Ein *operation*-Element wird über sein *name*-Attribut innerhalb seines umgebenden *interface*-Element eindeutig identifiziert. Zur Zuweisung des gewünschten Message-Exchange-Pattern[4] wird das *pattern*-Attribut verwendet, dessen mögliche Werte an dieser Stelle nur aufgelistet sind. Der Wert des *pattern*-Attributs kann z.B. *„http://www.w3.org/2005/08/wsdl/"* sein, gefolgt von:

- „in-only"
- „robust-in-only"
- „in-out"
- „in-opt-out"
- „out-only"
- „robust-out-only"
- „out-in"
- „out-opt-in"

Durch ein Message-Exchange-Pattern werden Art, Richtung und Abfolge des Nachrichtenaustausches zwischen den Web Services festgelegt. Außerdem wird spezifiziert, wer die Interaktion anstößt. Im Falle des „in-out"-Pattern leitet der anfragende Dienst durch eine Request-Nachricht die Interaktion ein, wohingegen beim „out-in"-Pattern der die Funktionalität anbietende Service zuerst aktiv wird. Im Zusammenhang mit Abbildung 6.8 und der Datenmodellierung werden diese Pattern nochmals genauer betrachtet.

Für die Datenmodellierung der Operationsaufrufe werden die *input*- und *output*-Elemente verwendet. Mittels dieser Elemente wird die Struktur der an die Operation zu übergebenden bzw. die zurückzugegebenden Datentypen spezifiziert. Die Fehlerbehandlung wird an dieser Stelle im Rahmen der Operationsspezifikation nicht betrachtet, sondern nur die Beschreibung der möglichen *input*- bzw. *output*-Elemente einer Operation.

Der betrachtete Prozess und damit auch die funktionalen Kapseln eines Service wurden in der Prozessanalyse mittels EPKs modelliert. Innerhalb dieser Prozessketten wird zwischen einem Kontroll- und einem Datenfluss unterschieden[5]. Der Kontrollfluss wird in WSDL durch das verwendete Message-Exchange-Pattern repräsentiert. Für jede funktionale Kapsel muss (mindestens) ein *operation*-Element spezifiziert werden, welches entsprechend dem gewünschten Message-Exchange-Pattern und dem Vorhandensein von Input- bzw. Outputdaten *input*- bzw. *output*-Elemente beinhaltet. Daneben können auch noch *infault*- und *outfault*-Elemente spezifiziert werden, die jeweils im Falle eines fehlerhaften Aufrufs greifen.

In Tabelle 6.1 sind jedem Message-Exchange-Pattern (WSDL 2.0) die notwendigerweise zu spezifizierenden *input*- bzw. *output*-Elemente zugeordnet, wobei zwischen den Initiatoren der jeweiligen Interaktion unterschieden wird. Beispielsweise startet der Service Requestor die Interaktion durch Senden einer Input-Nachricht, wenn das Pattern mit „in" beginnt. Im zweiten Fall (Pattern beginnen mit „out"), der im unteren Teil der Tabelle zu sehen ist, wird zuerst der Service Provider aktiv und sendet aus seiner Sicht eine Output-Nachricht, bevor er eventuell einen Input zurückerhält. Die in Tabelle 6.1 dargestellte Zuordnung muss bei der Spezifikation eines *operation*-Elements beachtet werden. Wird etwa für eine Operation das „in-out"-Pattern gewählt, so muss das *operation*-Element sowohl ein *input*- als auch ein *output*-Element beinhalten. Außerdem ist der anfragende Service der Initiator der Interaktion. Liefert die betrachtete funktionale Kapsel entsprechend ihrer Modellierung in der Prozessanalyse keinen Output, so muss entweder ein anderes Message-Exchange-Pattern für die Operation verwendet werden oder ein Output ergänzt werden.

Service Requestor startet Interaktion		
Message-Exchange-Pattern	Service Requestor sendet	Service Provider sendet
„in-out"	Input	Output
„in-opt-out"	Input	Output (optional)
„in-only", „robust-in-only"	Input	-
Service Provider startet Interaktion		
Message-Exchange-Pattern	Service Provider sendet	Service Requestor sendet
„out-in"	Output	Input
„out-opt-in"	Output	Input (optional)
„out-only", „robust-out-only"	Output	-

Tabelle 6.1.: Message-Exchange-Pattern und Datenzuordnung

[4] Message-Exchange-Pattern für WSDL 2.0 siehe [W3C 05c], für WSDL-Version 1.1 siehe [W3C 01].
[5] siehe Anhang A

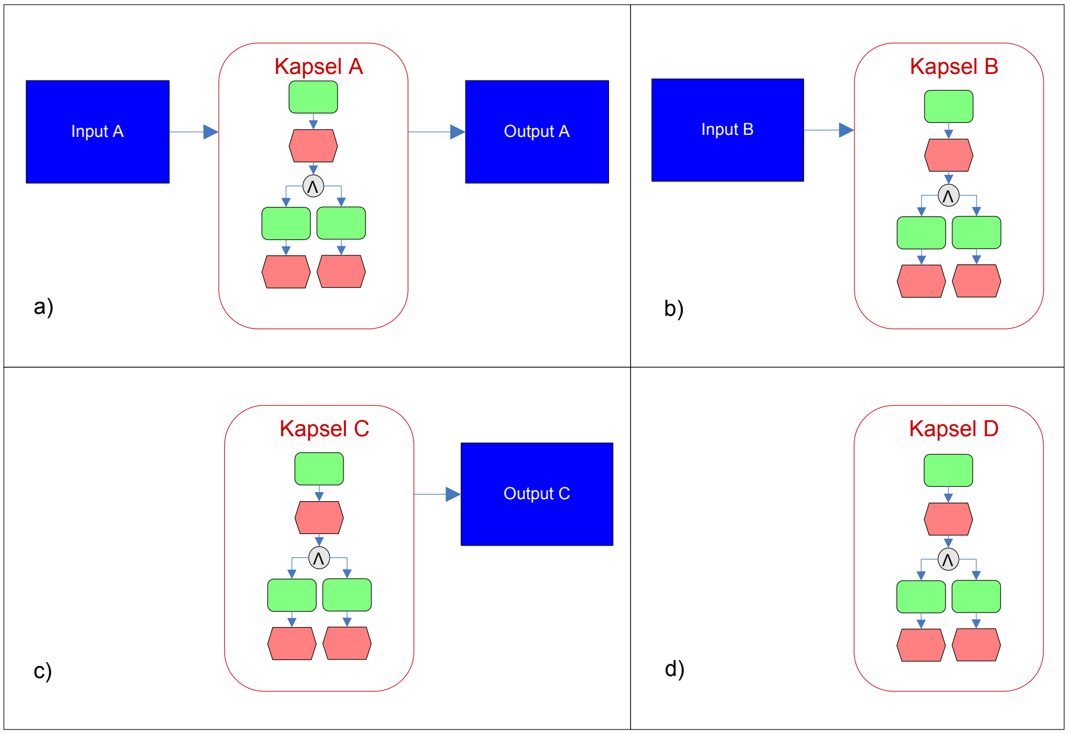


Abbildung 6.8.: Kapselarten bezogen auf Input-/Outputdaten

Abbildung 6.8 zeigt die vier datenbezogenen Kapselungssituationen, welche im Rahmen der Operationsspezifikation bezogen auf das zu wählende Message-Exchange-Pattern zu betrachten sind. Ein klassischer Fall ist in Abb. 6.8a) dargestellt. Der betrachteten funktionalen Kapsel A wird ein Input des Typs A übergeben, und anschließend wird von der Operation eine Output-Nachricht Output A zurückgeliefert. Der zugehörige Wert des *pattern*-Attributs des *operation*-Elements ist „in-out“, wenn die Interaktion vom anbietenden Service ausgeht, sonst „out-in“. Falls Input- oder Output-Nachricht optional sind, dann ist die Pattern-Variante mit dem entsprechenden optionalen Part („in-opt-out“ oder „out-opt-in“) zu wählen, wobei beim optionalen Input entsprechend der Pattern-Spezifikation der anbietende Service zuerst aktiv werden muss, da keines der bereitgestellten Pattern eine Anfrage mit optionalem Input vom anfragenden Dienst vorsieht. Diese Situation wird im Rahmen eines Exkurses auf siehe Seite 91 noch einmal aufgegriffen.

Die Fälle b) und c) in Abb. 6.8 zeigen Kapseln, für die das Nachrichtenformat jeweils nur für eine Kommunikationsrichtung zu spezifizieren ist. Für die Operation, welche die Signatur der funktionalen Kapsel B definiert, muss deshalb nur das *input*- und für den Fall c) nur das *output*-Element angegeben werden. Die Werte des *pattern*-Attributs sind entsprechend „in-only“ respektive „out-only“. Für diese beiden Patterns können jeweils auch die „robusten“ Varianten verwendet werden, auf die im Rahmen dieser Arbeit nicht weiter eingegangen wird.

Eine Sonderstellung nimmt die Kapsel D ein, welche in Abb. 6.8d) zu sehen ist. Die dargestellte Kapsel bedarf keinerlei Inputdaten, noch liefert sie dem anfragenden Dienst spezifizierte Rückgabedaten. Solch eine Kapselung kann in bestimmten Fällen aus der gewählten Verfeinerungstiefe bei der EPK-Modellierung des Prozesses resultieren. Um diese Situation dennoch auf eine WSDL-Operationsbeschreibung abbilden zu können, muss die Modellierung dieser Kapsel entweder verfeinert oder um künstliche Dateninputs bzw. -outputs ergänzt werden, damit sie einem Message-Exchange-Pattern zugeordnet werden kann. Da in diesem Abschnitt jedoch die Datenmodellierung im Vordergrund steht, wird an dieser Stelle auf Fall D nicht weiter eingegangen.

Tabelle 6.2 fasst noch einmal die eben beschriebene Zuordnung der Message-Exchange-Pattern zu den in Abbildung 6.8 dargestellten Kapselungsvarianten zusammen. Im Vergleich zu Tabelle 6.1 werden hier ausgehend von den abgebildeten Daten die möglichen Message-Exchange-Pattern zugeordnet.

Message-Exchange-Pattern-Zuordnung entsprechend den Daten in Abb. 6.8				
Kapsel A	Input A	Output A	„in-out", „out-in"	
	Input A	Output A (optional)	„in-opt-out"	(*)
	Input A (optional)	Output A	„out-opt-in"	(*)
Kapsel B	Input B	-	„in-only", „robust-in-only"	(*)
Kapsel C	-	Output C	„out-only", „robust-out-only"	(*)
Kapsel D	-	-	-	

Tabelle 6.2.: Daten-Kapselungsvariante und Message-Exchange-Pattern

Vergleicht man die beiden Tabellen 6.1 und 6.2, so stellt man fest, dass die mit Stern (*) in Tabelle 6.2 markierten Kapselungssituationen nicht immer dem gewünschten Kontrollfluss entsprechen, wenn man ausschließlich die zu modellierenden Daten betrachtet. Diese Abweichung resultiert daraus, dass der Kontrollfluss in WSDL an den Datenfluss gekoppelt ist. Der Datenfluss bestimmt zwar nicht zwingend den Kontrollfluss, dennoch muss der Datenfluss u.U. künstlich angepasst werden, um den gewünschten Kontrollfluss zu modellieren. Beispielsweise ist bei Kapsel C nur ein Output zu modellieren und deswegen das Pattern „out-only" oder „robust-out-only" zu verwenden (siehe Abb. 6.8c und Tabelle 6.2). Entsprechend der Übersicht aus Tabelle 6.1 startet demzufolge der Service Provider die Interaktion, was vielleicht nicht gewünscht ist. Anhand dieser Situation wird im Rahmen eines Exkurses noch einmal die künstliche Anpassung des Datenflusses betrachtet[6].

Die bisher dargestellte Spezifikation einer Interface-Operation ging von einem synchronen Aufruf dieser Operation aus. Grundsätzlich werden jedoch zwei Arten von Aufrufen einer Operation unterschieden (vgl. [JMS 04]):

synchone Web-Service-Operationen: Es wird ein Request gesendet und anschließend auf ein Response gewartet. Die so angesprochenen Operationen benötigen normalerweise für ihre Bearbeitung kaum Zeit, weswegen der Requestor auf die Antwort des Service Providers wartet und solange blockiert ist.

asynchone Web-Service-Operationen: Normalerweise erfordern solche Operationen eine längere Zeit für die Bearbeitung. Damit dennoch, ohne dass der Requestor für die Bearbeitungszeit blockiert wird, ein mögliches Ergebnis an den Requestor zurückgesendet werden kann, wird die Spezifikation geeignet angepasst, indem z.B. eine „Call-back"-Operation definiert wird.

Bevor also eine funktionale Kapsel auf eine WSDL-Operation abgebildet werden kann, muss entsprechend der Funktionalität dieser Kapsel entschieden werden, ob die Operation synchron oder asynchron spezifiziert werden soll. Dabei sind jedoch nur diejenigen funktionalen Kapseln zu untersuchen, für welche sowohl Input- als auch Output-Nachrichten zu spezifizieren sind. Ob diese Nachrichten künstlich erzeugt wurden oder nicht, spielt dabei keine Rolle. Abbildung 6.8b) und c) müssen nicht betrachtet werden, da hier nur Nachrichten in eine Richtung versendet werden. Die in Abbildung 6.8a) dargestellte funktionale Kapsel muss im Falle einer synchronen Kommunikation nicht modifiziert werden und kann entsprechend dem bisher beschriebenen Vorgehen behandelt werden.

Soll jedoch eine asynchrone Kommunikation modelliert werden, so muss die funktionale Kapsel, wie in Abbildung 6.9 dargestellt, in eine Hin- und eine Rückrichtung zerlegt werden. Für jede der zwei Richtungen ist dann ein *operation*-Element innerhalb eines *interface*-Elements zu spezifizieren, wobei die umgebenden *interface*-Elemente dabei jedoch i.d.R. nicht dieselben sind. Bei der in Abbildung 6.9 dargestellte Aufteilung in eine Hin- bzw. Rückrichtung wird davon ausgegangen, dass der anfragende Dienst entsprechend der synchronen Variante „in-out" zuerst aktiv wird. Im Fall, dass der Service Provider die Interaktion startet, müssten Hin- und Rückrichtung vertauscht werden.

[6] siehe Seite 91

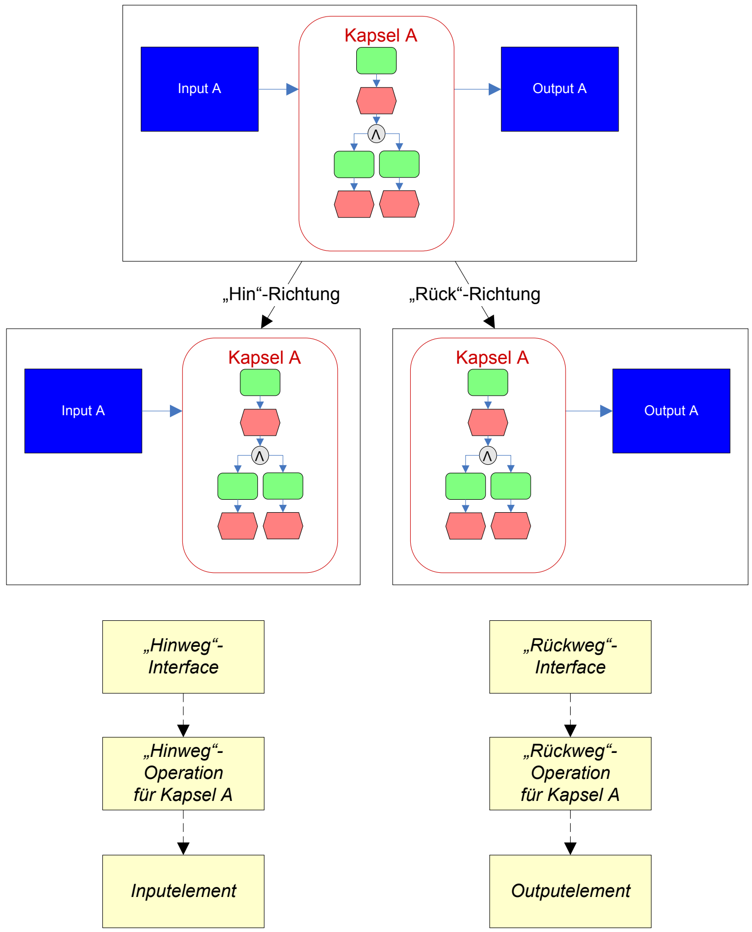


Abbildung 6.9.: Asynchrone Kommunikation mit einer funktionalen Kapsel

Nachdem bisher davon gesprochen wurde, dass innerhalb der Operationsbeschreibung entsprechend den Daten, welche von der betrachteten funktionalen Kapsel benötigt bzw. geliefert werden, sowie dem daraus resultierenden Message-Exchange-Pattern *input*- bzw. *output*-Elemente spezifiziert werden müssen, wird nun die allgemeine Grammatik dieser Elemente betrachtet:

```
<operation>
  ...
  <input messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? >
    <documentation />*
    [ <feature /> | <property /> ]*
  </input>
  <output messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? >
    <documentation />*
    [ <feature /> | <property /> ]*
  </output>
  ...
</operation>
```

Sowohl beide im Schema abgebildeten Elemente als auch die nicht weiter betrachteten Elemente *infault* und *outfault* besitzen ein Attribut *messageLabel*, welches die Richtung in Form der Werte „In“ oder „Out“ entsprechend dem *pattern*-Attribut des *operation*-Elements angibt. Falls es nur eine Richtung gibt, ist das *messageLabel*-Attribut optional (siehe auch [W3C 05c]). Neben dieser Richtungsangabe besitzen die abgebildeten Elemente noch ein *element*-Attribut, welches auf ein Element innerhalb der *types*-Datentypdefinition referenziert, das der gewünschten Datenstruktur der funktionalen Kapsel entspricht.

Nachdem das Schema der *input*- und *output*-Elemente einer Operationsbeschreibung vorgestellt wurde, kommen wir noch einmal zu den Message-Exchange-Pattern zurück. Lässt man die „robusten“ Varianten bei den Message-Exchange-Pattern unbeachtet, so bleiben sechs mögliche Varianten zur Modellierung der Nachrichtenkommunikation zwischen den beteiligten Diensten. Vergleicht man diese Möglichkeitender hier verwendeten Version 2.0 mit Version 1.1 der WSDL-Spezifikation, stellt man fest, dass in jener Version die Pattern mit den optionalen Inputs oder Outputs nicht zur Verfügung stehen. Im Rahmen des folgenden Exkurses soll in gewisser Weise eine weitere Kommunikationsvariante motiviert werden, die auf die vorhandenen Message-Exchange-Pattern abgebildet wird, womit auch die Einführung der WSDL-2.0-Pattern mit optionalem Bestandteil begründet wird. Die dabei betrachtete Beispielsituation entspricht der in Abbildung 6.8c) dargestellten Kapselungsvariante. Wie bereits angemerkt, kann ausgehend von den abgebildeten Daten (nur Output) theoretisch nur ein „out-...“-Pattern zur Modellierung des Kontrollflusses verwendet werden. Warum dies nicht immer erwünscht ist und wie der Datenfluss künstlich über einen „leeren Input“ dem gewünschten Kontrollfluss angepasst werden kann, soll in diesem Exkurs betrachtet werden.

Exkurs: Künstliche Datenflussanpassung

Für die Analyse der zu modellierenden Operation werden die Pattern von WSDL Version 1.1 verwendet, damit der Leser auch mit diesen Bezeichnern der Kommunikationspattern vertraut gemacht wird. Der betrachtete Aspekt betrifft jedoch auch WSDL 2.0. Analog zum bereits beschriebenen Vorgehen werden die während der Prozessanalyse ermittelten Inputs und Outputs der funktionalen Kapseln des zu spezifizierenden Web Service als Datentypen modelliert. Anschließend müssen die Operationen des *interface*-Elements spezifiziert werden, welches in WSDL 1.1 noch mit *portType* bezeichnet wird. Bei der Beschreibung von Kommunikation mittels WSDL 1.1 werden von der Spezifikation prinzipiell vier Möglichkeiten des Nachrichtenaustausches genannt, wobei in Klammern die Entsprechungen in WSDL 2.0 angegeben sind und an dieser Stelle unter einem Endpunkt ein angesprochener Service verstanden werden kann.

- One-Way (in-only): Endpunkt erhält eine Nachricht.
- Request-Response (in-out): Endpunkt erhält eine Nachricht und sendet eine korrelierte Nachricht.
- Solicit-Response (out-in): Endpunkt sendet eine Nachricht und erhält eine korrelierte Nachricht.
- Notification (out-only): Endpunkt sendet eine Nachricht.

Man betrachte folgende Situation: Man möchte von einem Dienst eine ID für eine neue Störungsmeldung erfragen, durch welche die Störungsmeldung später eindeutig identifiziert werden kann. Diese ID kann nicht willkürlich ausgewählt werden, sondern muss die bereits vergebenen IDs berücksichtigen. In der objektorientierten Modellierung kann diese Situation mit einer Methode ohne Parameter aber mit Rückgabewert charakterisiert und umgesetzt werden. In WSDL ist dies nur mit dem Pattern Request-Response realisierbar. Dennoch soll an dieser Stelle aufgezeigt werden warum Solicit-Response, Notification und One-Way für die Modellierung der beschriebenen Situation ungeeignet sind, obwohl erstere nicht durch die Bindings in WSDL unterstützt werden.

Da bei Solicit-Response der Service zuerst eine Nachricht sendet, kann man dies dazu verwenden, die ID für eine Störungsmeldung quasi unaufgefordert in vorher festgelegten Intervallen potenziellen Interessenten zu senden. Wird vom potenziellen Interessenten die empfange ID tatsächlich benötigt, so muss dies dem Service mitgeteilt werden, ebenso, falls diese nicht benötigt wird. Gibt es für die ID nur einen potentiellen Interessenten, so ließe sich diese Interaktion relativ einfach umsetzten, da der sendende Service nur seinen Interaktionspartner und dessen für die Interaktion notwendigen Spezifikationen kennen muss. Existiert mehr als ein potentieller Interessent, so müssen Strategien überlegt werden, wem welche ID angeboten wird. Wenn z.B. allen Interessenten die gleiche ID angeboten wird, könnte es passieren, dass mehrere von ihnen diese verwenden wollen, was ausgeschlossen werden muss, da in der beschriebenen Situation gefordert wird, dass diese genau einer Störungsmeldung zugeordnet wird. Wird im Gegenzug jede mögliche ID nur an einen potentiellen Interessenten gesendet, kann sich der Aufwand zur Verwaltung der IDs u.U. deutlich erhöhen. Außerdem wird das System durch viele überflüssig gesendete Nachrichten unnötig belastet.

Der Ansatz der Notification ist für die Situation ebenfalls ungeeignet, da der sendende Service nicht weiß, zu welchem Zeitpunkt eine ID benötigt wird. Auch wenn man die Idee des Intervall-Sendens aufgreift, so ist man mit ähnlichen Problemen wie bei Solicit-Response konfrontiert. Zudem müsste u.U. irgendwie überprüft werden, welche der angebotenen IDs tatsächlich verwendet werden und welche nicht.

One-Way ist in dieser Situation ungeeignet, da derjenige, der bezüglich der ID einen Request stellt, einen Response in Form einer eindeutigen ID für die Störungsmeldung erwartet. Betrachtet man das Prinzip des Request-Response, so ist es auf den ersten Blick für die Modellierung dieser Situation geeignet, dennoch wird bei der Spezifikation in WSDL die Frage aufgeworfen, wie der Request zu modellieren ist, wenn dessen Inhalt keinen Einfluss auf den Response hat.

Fassen wir diese Situation noch einmal zusammen: Ein Request ist in diesem Zusammenhang sinnvoll und auch notwendig, da zum einen, wie bereits erwähnt, Solicit-Response und Notification nicht unterstützt werden und zum anderen, damit der Service, welcher die eindeutige ID für die Störungsmeldung liefert, den Zeitpunkt kennt, wann diese benötigt wird. Die zu liefernde ID stellt den Inhalt der Response-Nachricht dar, wodurch One-Way ausscheidet. Einzige mögliche Modellierungsvariante stellt somit Request-Response dar.

Da für den Datenzugriff einer „leeren“ Anfrage kein Input benötigt wird, muss für die Realisierung des Request-Response-Pattern ein künstlicher Input geschaffen werden. Dieser leere Inputdatentyp wird bspw. in XML Schema spezifiziert. Je nach Art der Spezifikation der Datentypen und Elemente in XML Schema ist es u.U. ausreichend, diesen leeren Inputtyp nur einmal zu definieren. Drei mögliche Modellierungsvarianten sind hier dargestellt:

In allen drei Varianten wird ein Element „NullInput“ definiert, welches im ersten Fall einen String darstellt, aber auch „leer“ sein kann (*nillable = true*). Der mögliche „String“-Inhalt des Elements kann dann einfach ignoriert oder als optional betrachtet werden.

```
<description ...
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <types>
    <xsd:schema ... xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element
        name = "NullInput"
        nillable = true
        type = "xsd:string" />
    </xsd:schema>
  </types>
  ...
</description>
```

Soll überhaupt kein „Inhalt“ angegeben werden können, so kann das Element auch als *complexType* definiert werden, dessen Inhalt (*complexContent*) aber keine Elemente oder Text enthält.

```
<xsd:element name="NullInput">
  <xsd:complexType>
    <xsd:complexContent>
      <xsd:restriction base="xsd:anyType" />
    </xsd:complexContent>
  </xsd:complexType>
</xsd:element>
```

Die dritte Alternative wäre ein auf die Länge null begrenzter String.

```
<xsd:element name="NullInput">
  <xsd:simpleType>
    <xsd:restriction base="xsd:string" />
      <xsd:length value="0" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

Dieser Exkurs sollte die Einführung der Message-Exchange-Pattern mit optionalen Inputs oder Outputs motivieren. Außerdem wurde die künstliche Anpassung des Datenflusses zur Spezifikation des gewünschten Kontrollflusses vorgestellt. Dazu wurde gezeigt, dass auch Operationen, welche zwar grundsätzlich zuerst eine Nachricht des anfragenden Dienstes an den Web Service erwarten, wobei jedoch kein „Inhalt“ mittels der Input-Nachricht übermittelt werden muss, über einen Umweg (den „leeren“ Datentyp) entsprechend den zur Verfügung stehenden Möglichkeiten modelliert werden können. Wie dieser „leere“ Datentyp modelliert wird, spielt dabei keine Rolle.

Nachdem die für die Typ- und Interfacebeschreibungen notwendigen WSDL-Grammatiken vorgestellt und einige Bestandteile näher erklärt wurden, fasst Abbildung 6.10 die beschriebene Umsetzung noch einmal schematisch zusammen, bevor dann mit Abbildung 6.11 eine entsprechende Beispielcodierung behandelt wird.

Beispiel

In Abbildung 6.10 ist ein Web Service dargestellt, welcher eine funktionale Kapsel A umfasst, die der Variante A aus Abbildung 6.8 entspricht. Innerhalb des *types*-Elements werden die später für den Input A und den Output A verwendeten Datentypen in XML Schema beschrieben. Die funktionale Kapsel wird innerhalb des *interface*-Elements durch ein *operation*-Element repräsentiert. Da dieser Operation Daten in Form des zuvor definierten Datentyps (Input A) übergeben werden und sie anschließend Daten der Form Output A zurückliefert, erhält das *pattern*-Attribut dieser Operation den Wert *„.../in-out“*[7]. Entsprechend diesem Pattern muss für diese Operation sowohl ein *input*- als auch ein *output*-Element mit dem jeweils passenden *messageLabel*-Wert definiert werden. Die Werte der *element*-Elemente richtet sich dann nach den im *types*-Element definierten Datentypen für Input und Output.

Zur Verdeutlichung der vorgestellten Vorgehensweise zeigt Abbildung 6.11 eine beispielhafte Umsetzung der eben betrachteten Situation in WSDL. Der betrachtete Web Service trägt den Namen „IncidentStatus-Service“ gemäß dem *name*-Attribut des *description*-Elements, und der Namespace dieses Web Service ist *„http://example.org/ws/“*. Neben diesem Namensraum und dem WSDL-Namensraum sind weitere, z.B. für die Verwendung von XML Schema, angegeben. Anschließend werden für den Service innerhalb des *types*-Elements zwei Elemente („IncidentID“ und „IncidentStatus“) mittels XML Schema definiert. Die Datentypen dieser Elemente werden nur aus dem Grund separat statt innerhalb der Elemente definiert, damit sie auch noch für andere Elemente verwendet werden können. Der *xsd:simpleType* „Identifikationsnummer“ basiert auf dem *xsd:long*-Typ von XML Schema (*base=„xsd:long“*), während der „Status“ eine Aufzählung möglicher String-Werte („angenommen“, „inBearbeitung“, ...) ist.

Neben der Beschreibung der Datentypen ist unterhalb des *types*-Elements ein *interface*-Element mit dem Namen „KapselAInterface“ spezifiziert, welches eine Operationsbeschreibung enthält. Da der beschriebenen „opKA“-Operation durch das *pattern*-Attribut (*pattern=„http://www.w3.org/2005/08/wsdl/in-out“*) die Kommunikationsart „in-out“ zugewiesen ist, wurde sowohl ein *input*- als auch ein *output*-Element definiert, wobei die Input-Nachricht dem im *types*-Element spezifiziertem Datentyp „IncidentID“ (*element=„tnst:IncidentID“*) entspricht und die zurückgelieferte Output-Nachricht vom Typ „IncidentStatus“ (*element=„tns:IncidentStatus“*) ist.

[7] Der Wert hängt von der verwendeten WSDL-Version ab und kann z.B. für *“http://www.w3.org/2005/08/wsdl/in-out“* stehen.

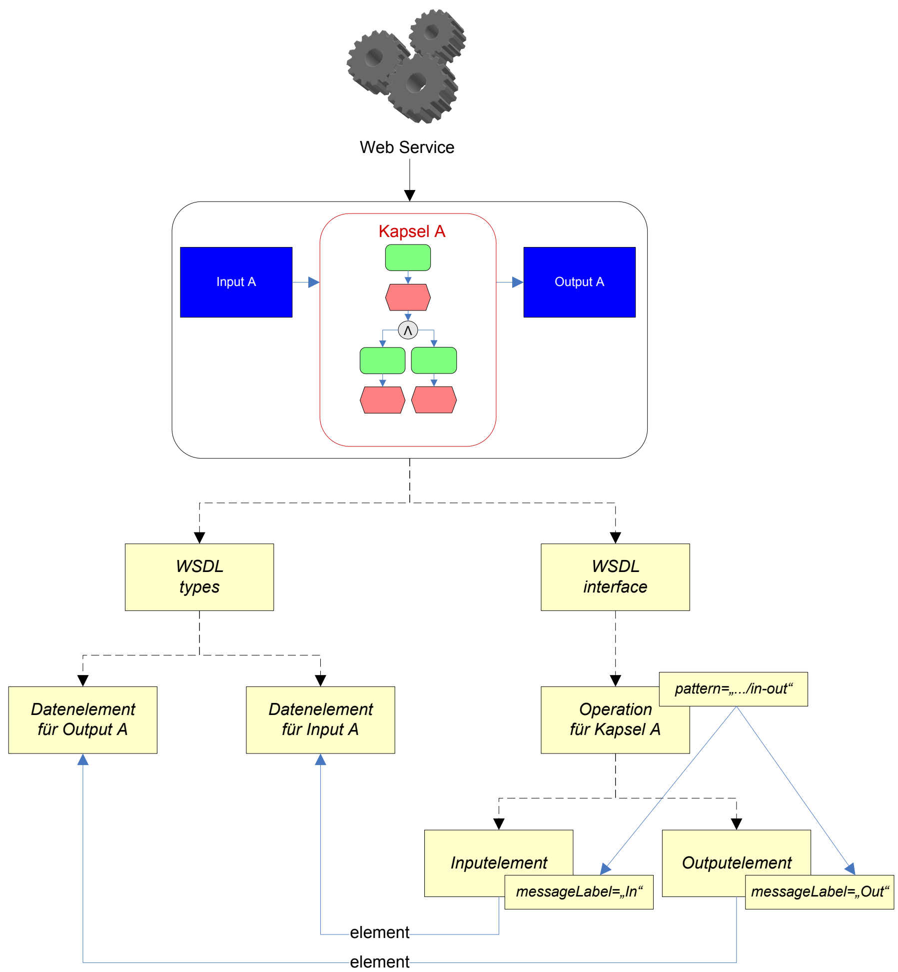


Abbildung 6.10.: Schematische Beispielumsetzung der Typ- und Interfacebeschreibung

Wie bereits erwähnt wurde, besteht ein WSDL-Dokument aus vier grundlegenden Elementen. In diesem Abschnitt wurden Typ- und Interfacebeschreibung, welche zusammen den abstrakten Teil eines WSDL-Files bilden, betrachtet. Die dargestellte WSDL-Beschreibung stellt, so wie sie in Abbildung 6.11 dargestellt ist, jedoch noch kein vollständiges WSDL-File im angestrebten Sinne dar, da die Spezifikationen des *binding*- und des *service*-Elements noch fehlen, welche im nachfolgenden Abschnitt 6.3.1.2 ergänzt werden.

6.3.1.2. Binding- und Servicebeschreibung

Dieser Abschnitt befasst sich mit der konkreten Beschreibung der Web Service-Schnittstelle in Form der Binding- und Servicebeschreibung. In der Bindingbeschreibung werden dabei zuerst die technischen Details wie bspw. Informationen zum verwendeten Nachrichtenprotokoll spezifiziert. Anschließend werden durch die Servicebeschreibung dann die konkreten „Endpunkte“ für die Kommunikation festgelegt.

Bisher reichte es aus, den abgebildeten Web Service als eine Art logische Einheit von funktionalen Kapseln zu betrachten. Diese Sichtweise ist nicht ganz korrekt. Vielmehr können sich hinter dieser logischen Einheit verschiedenste Implementierungen verbergen, die über unterschiedliche Service-Adressen angesprochen

```
<?xml version="1.0" encoding="UTF-8"?>

<description name="IncidentStatusService"
  targetNamespace="http://example.org/ws/"
  xmlns="http://www.w3.org/2005/08/wsdl"
  xmlns:tns="http://example.org/ws/"
  xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
      http://www.w3.org/2005/08/wsdl/wsdl20.xsd">

  <types>
    <xsd:schema
      targetNamespace="http://example.org/ws/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" />
      <xsd:element name="IncidentID" type="Identifikationsnummer" />
      <xsd:element name="IncidentStatus" type="Status">
      <xsd:simpleType name="Identifikationsnummer">
        <xsd:restriction base="xsd:long" />
      </xsd:simpleType>
      <xsd:simpleType name="Status">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="angenommen" />
          <xsd:enumeration value="inBearbeitung" />
          <xsd:enumeration value="zurückgestellt" />
          <xsd:enumeration value="weitergeleitet" />
          <xsd:enumeration value="gelöst" />
          <xsd:enumeration value="geschlossen" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:schema>
  </types>

  <interface name="KapselAInterface">
    <operation name="opKA" pattern="http://www.w3.org/2005/08/wsdl/in-out">
      <input messageLabel="In" element="tns:IncidentID" />
      <output messageLabel="Out" element="tns:IncidentStatus" />
    </operation>
  </interface>

</description>
```

Bezeichner des Web Services

Typ-beschreibung

Datentyp-beschreibung, die für das Beispiel definiert wurde.

Struktur-beschreibung für die Nachricht

Interface-beschreibung

Operations-beschreibung für Kapsel A

Abbildung 6.11.: Beispielcodierung für die Typ- und Interfacebeschreibung

werden. Auch die Verwendung unterschiedlicher Protokolle zur Kommunikation oder die Aufspaltung des logischen Web Services in mehrere Services, welche jeweils nur einen Teil der Schnittstellenbeschreibung realisieren, sind denkbar, was durch die „Wolke" in Abbildung 6.12 angedeutet werden soll. Damit sich der Leser nicht in der Vielfalt der daraus resultierenden Möglichkeiten zur Spezifikation der *binding*- und *service*-Elemente verliert, wird an dieser Stelle insbesondere für das Beispiel stark vereinfacht angenommen, dass ein Service alle spezifizierten Interface-Operationen realisiert und nur über ein Protokoll an einer Netzwerkadresse angesprochen werden kann.

Abbildung 6.12 zeigt die strukturellen Elemente, welche für die Spezifikation von Binding- und Service-beschreibung betrachtet werden müssen. Zu sehen ist derjenige Web Service, für welchen im vorangegangenen Abschnitt die benötigten Datentypen und Interface-Operationen beschrieben wurden.

Entsprechend der in Abschnitt 6.2 vorgestellten allgemeinen Abbildungssemantik steht das Element mit dem Bezeichner „binding 1..." in Abbildung 6.12 für eine Menge von *binding*-Elementen der WSDL-Spezifikation. Jedes kann über ein Attribut auf ein *interface*-Element der WSDL-Spezifikation referenzieren. Des Weiteren wird das zu verwendende Nachrichtenprotokoll über ein Attribut spezifiziert, welches Einfluss auf die Konkretisierung der Operationen der Interfacebeschreibung hat. Neben dem *binding*-Element ist noch ein WSDL-*service*-Element in Abbildung 6.12 dargestellt, welches auf ein vorhandenes Interface referenziert.

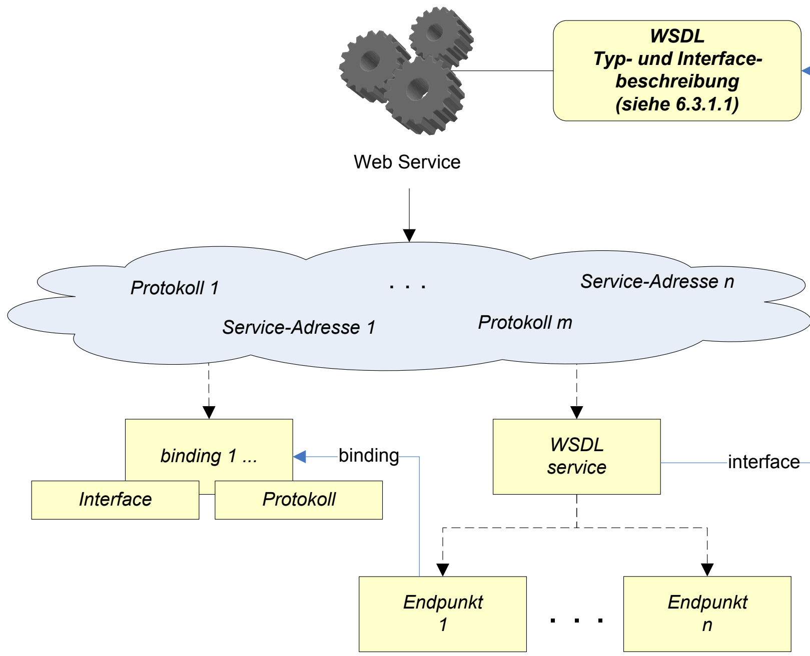


Abbildung 6.12.: Binding- und Servicebeschreibung

Die Beschreibung der *interface*-Elemente wurde zusammen mit der Typbeschreibung im vorherigen Abschnitt betrachtet und ist deshalb in der Abbildung nur durch ein symbolisches Element oben rechts dargestellt. Innerhalb eines *service*-Elements können mehrere sogenannte Endpunkte definiert werden, welche auf ein zuvor definiertes *binding*-Element verweisen.

Da die Spezifikation der Endpunkte eines *service*-Elements auf die Beschreibung der *binding*-Elemente aufbaut, werden diese zuerst betrachtet. Im Fokus dieser Betrachtung liegen dabei jedoch nicht die möglichen unterschiedlichen Protokolle, sondern die allgemeine Struktur dieses Elements sowie die Bedeutung wichtiger Attribute und Elemente.

Bindingbeschreibung

Mittels des *binding*-Elements wird die Interfacebeschreibung auf eine existierende Implementierung abgebildet, indem das konkrete Protokoll für die Kommunikation spezifiziert wird. Außerdem werden die tatsächlichen Datenformate der bisher nur abstrakt beschriebenen Nachrichten für ein spezielles Interface festgelegt und damit an eine konkrete Implementierung gebunden. Nachfolgend ist die allgemeine Grammatik für die Bindingbeschreibung aufgezeigt:

```
<binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI">
  <documentation />*
  [ <fault /> | <operation /> | <feature /> | <property /> ]*
</binding>
```

Jedem Binding kann über das *interface*-Attribut eine spezielle Interfacebeschreibung zugeordnet werden. Da jedoch mehrere *binding*-Elemente auf das gleiche *interface*-Element referenzieren können, welches über sein *name*-Attribut eindeutig bestimmbar ist, können die *binding*-Elemente nur durch ihr *name*-Attribut in Kombination mit ihrem *interface*-Attribut eindeutig bestimmt werden. Das *type*-Attribut liefert bindungstechnische Details des verwendeten Nachrichtenformats, wobei der Wert *„http://www.w3.org/2005/08/wsdl/soap“* beispielsweise für eine Bindung an SOAP steht. Durch Weglassen des Interface-Bezugs sowie spezieller Operationsangaben kann das Binding wieder verwendbar gehalten werden.

Das *operation*-Element innerhalb des Bindings, für welches hier die allgemeine Grammatik abgebildet ist, beschreibt die technischen Details einer speziellen Operation eines bestimmten Interfaces für einen Endpunkt. Mittels dessen *ref*-Attributs werden die Nachrichten einer abstrakten Operationsdefinition mit dem spezifischen Nachrichtenformat verknüpft[8].

```
<binding>
  <operation ref="xs:QName">
    <documentation />*
    [ <input /> | <output /> | <infault /> | <outfault /> |
      <feature /> | <property /> ]*
  </operation>
  ...
</binding>
```

Nachdem durch die Bindingbeschreibung die abstrakte Interfacespezifikation auf ein konkretes Protokoll und konkrete Nachrichtenformate abgebildet wurde, kann diese Abbildung einem Service-Endpunkt zugewiesen werden.

Servicebeschreibung

Das *service*-Element eines WSDL-Dokuments ist eine Kollektion von verwandten Endpunkten, wie der allgemeinen Grammatik zu entnehmen ist.

```
<service name="xs:NCName" interface="xs:QName" >
  <documentation />*
  <endpoint />+
  [ <feature /> | <property /> ]*
</service>
```

Durch das *name*-Attribut des *service*-Elements kann dieses von anderen Komponenten referenziert werden. Über das *interface*-Attribut wird die Schnittstelle angegeben, die der Service instanziert. Jedes *service*-Element muss mindestens ein *endpoint*-Element spezifizieren, dessen allgemeine Grammatik nachfolgend dargestellt ist.

```
<endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? >
  <documentation />*
  [ <feature /> | <property /> ]*
</endpoint>
```

Die *endpoint*-Elemente innerhalb eines *service*-Elements teilen sich das gleiche Interface, können aber unterschiedliche Bindings bzw. Netzwerkadressen verwenden. Durch das *address*-Attribut wird der absolute URI des Dienstes unter Einbeziehung des Bindings, welches im *binding*-Attribut referenziert wird, eindeutig festgelegt. Ein *endpoint*-Element beschreibt somit die konkrete Komponente, welche sich hinter einer bestimmten Netzwerkadresse verbirgt und den Dienst zur Verfügung stellt [DJMZ 05].

Beispiel

Abbildung 6.13 fasst die Beschreibung der eben vorgestellten WSDL-Grammatiken noch einmal zusammen. In der Mitte ist das *binding*-Element zu sehen, womit das zu verwendende Nachrichtenprotokoll (*type*=*"../wsdl/soap"*) für das im vorangegangenen Abschnitt beschriebene Interface festgelegt wird. Für das Interface müssen dann die Nachrichten der Operationen noch in das entsprechende Protokoll umgesetzt werden, wozu das *operation*-Element innerhalb des Bindings verwendet wird. Die konkreten Endpunkte der Dienste einer SOA, welche die angebotenen Operationen implementieren, werden für das entsprechende Interface in einem *service*-Element zusammengefasst. Jedem *endpoint*-Element wird dazu ein Binding zugewiesen sowie die eindeutige Adresse der realisierenden Komponente festgelegt.

[8]Eine Verknüpfung mit SOAP ist in Abb. 6.14 zu sehen.

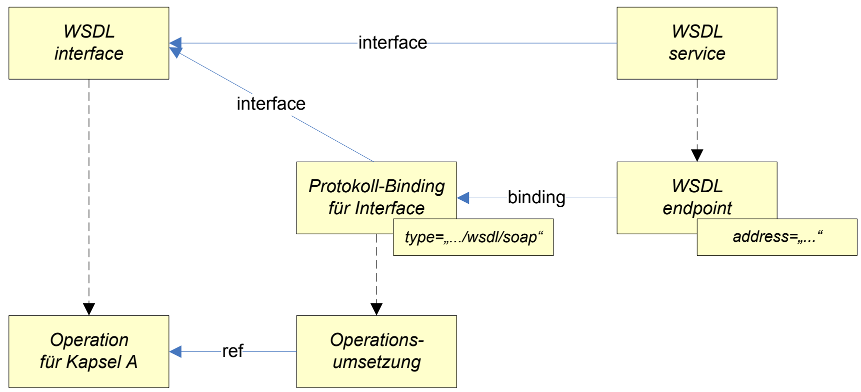


Abbildung 6.13.: Schematische Beispielumsetzung der Binding- und Servicebeschreibung

Die beispielhafte Typ- und Interfacebeschreibung von Abbildung 6.11 wird in Abbildung 6.14 aufgegriffen und um ein *binding*- und ein *service*-Element ergänzt. Durch das *binding*-Element „KapselASOAPBinding“ bzw. dessen *type*-Attribut (*type=„http://www.w3.org/2005/08/wsdl/soap“*) wird für das definierte Interface (*interface=„tns:KapselAInterface“*) eine Bindung an SOAP spezifiziert, da SOAP eines der gebräuchlichsten Protokolle für den Austausch von Nachrichten zwischen den Diensten einer SOA ist. Anschließend wird für die Operation „opKA“ die entsprechende SOAP-Umsetzung der Nachrichten festgelegt[9]. Die SOAP-Message-Exchange-Pattern, wie das im Beispiel verwendete *„http://www.w3.org/2003/05/soap/mep/request-response“* oder auch *„http://www.w3.org/2003/05/soap/mep/soap-response“* sowie deren Verwendung sind der SOAP-Spezifikation [W3C 03b] zu entnehmen.

Zur Vereinfachung des Beispiels wurde angenommen, dass die Operation „opKA“ durch genau einen Dienst realisiert wird, welcher nur einmal innerhalb der SOA zu finden ist und nur ein Nachrichtenprotokoll unterstützt. Im Beispiel wird dieser Sachverhalt durch das eine *endpoint*-Element „IncidentKAEndpoint“ innerhalb des *service*-Elements repräsentiert. Über das *binding*-Attribut (*binding="tns:KapselASOAPBinding"*) wird SOAP für diesen Endpunkt als zu verwendendes Nachrichtenprotokoll festgelegt. Die Komponente, welche den Dienst realisiert, verbirgt sich hinter der durch das *address*-Attribut angegebenen Netzwerkadresse. Würden mehrere Service-Endpunkte existieren bzw. unterschiedliche Nachrichtenprotokolle verwendet werden können, müssten entsprechend weitere *endpoint*- und *binding*-Elemente definiert werden, worauf in diesem Beispiel verzichtet wird.

Mit dem Beispiel, welches in Abbildung 6.14 dargestellt ist, wird die WSDL-Spezifikation abgeschlossen, nachdem zuerst der abstrakte Teil eines WSDL-Files, der die Typ- und Interfacebeschreibung beinhaltet, betrachtet wurde. Anschließend wurde die konkrete Abbildung auf die tatsächliche Implementierung beschrieben. Im Rahmen dieser Abbildung wurde dabei sowohl die Spezifikation der Nachrichtenprotokolle durch ein *binding*-Element als auch die Definition so genannter Endpunkte besprochen. Der nächste Abschnitt widmet sich kurz der Implementierung der Dienste, welche sich hinter den Endpunkten des WSDL-Files verbergen, bevor die Orchestrierung der einzelnen Dienste mittels BPEL im Abschnitt 6.4 betrachtet wird.

6.3.2. Web-Service-Implementierung

Neben der im vorigen Abschnitt beschriebenen Spezifikation der Web Service-Schnittstelle mit WSDL ist der zweite Punkt der Servicespezifikation die Implementierung der Funktionalitäten des Web Services, wobei die innere Logik der zugehörigen funktionalen Kapseln ausschlaggebend ist. In Kapitel 2 wurde neben der losen Kopplung der Komponenten bzw. der Dienste die Unabhängigkeit von der Programmiersprache

[9]Alternativen zur abgebildeten Spezifikation der Bindung an SOAP können u.a. in [ZTP 03] nachgelesen werden.

Abschnitt 6.3.1.1

```
<?xml version="1.0" encoding="UTF-8"?>

<description name="IncidentStatusService"
  targetNamespace="http://example.org/ws/"
  xmlns="http://www.w3.org/2005/08/wsdl"
  xmlns:tns="http://example.org/ws/"
  xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
      http://www.w3.org/2005/08/wsdl/wsdl20.xsd">

  <types>
    <xsd:schema
      targetNamespace="http://example.org/ws/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="IncidentID" type="Identifikationsnummer" />
      <xsd:element name="IncidentStatus" type="Status" />
      <xsd:simpleType name="Identifikationsnummer">
        <xsd:restriction base="xsd:long" />
      </xsd:simpleType>
      <xsd:simpleType name="Status">
        <xsd:restriction base="xsd:string">
          <xsd:enumeration value="angenommen" />
          <xsd:enumeration value="inBearbeitung" />
          <xsd:enumeration value="zurückgestellt" />
          <xsd:enumeration value="weitergeleitet" />
          <xsd:enumeration value="gelöst" />
          <xsd:enumeration value="geschlossen" />
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:schema>
  </types>

  <interface name="KapselAInterface">
    <operation name="opKA" pattern="http://www.w3.org/2005/08/wsdl/in-out">
      <input messageLabel="In" element="tns:IncidentID" />
      <output messageLabel="Out" element="tns:IncidentStatus" />
    </operation>
  </interface>
```

abstraktes Interface und Operation auf konkrete Implementierung abbilden

SOAP-Binding-Beschreibung für KapselAInterface

Abschnitt 6.3.1.2

```
  <binding name="KapselASOAPBinding"
    interface="tns:KapselAInterface"
    type="http://www.w3.org/2005/08/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
    <operation ref="tns:opKA"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response" />
  </binding>
```

Service-beschreibung für KapselAInterface

Endpunkt-beschreibung mit SOAP-Binding für Interface

```
  <service name="IncidentServiceKA" interface="tns:KapselAInterface">
    <endpoint name="IncidentKAEndpoint"
      binding="tns:KapselASOAPBinding"
      address="http://example.org/IncidentAgent" />
  </service>

<description>
```

Abbildung 6.14.: Beispielcodierung für die Binding- und Servicebeschreibung

als weiteres wichtiges Merkmal einer SOA vorgestellt. Da das vorgestellte Konzept basierend auf einer Service Oriented Architecture entwickelt wurde, spielt also die tatsächlich verwendete Programmiersprache bei der Implementierung der identifizierten und spezifizierten Web Services keine Rolle. Des Weiteren ist die tatsächliche Implementierung der Dienste nicht Teil dieser Arbeit, weswegen an dieser Stelle auch darauf verzichtet wird.

Ein weiterer Grund dafür, dass die Implementierung nicht weiter vertieft wird, ist der Aspekt, bereits vorhandene Dienste für die SOA-basierte Umsetzung des Prozesses zu verwenden, indem diese durch eine entsprechende Schnittstelle angebunden werden. Diese so genannten Legacy-Applications sollen und müssen in einer SOA nicht neu entwickelt werden. Ebenso wenig ist es nicht notwendig, dass alle genutzten Dienste in derselben Programmiersprache vorliegen. Die Betrachtung der dadurch entstehenden Vielfalt an möglichen zur Implementierung der Web Services verwendbaren Programmiersprachen fällt ebenfalls nicht in den Rahmen dieser Arbeit.

Die einzigen Aspekte, die im Rahmen dieses Konzeptes bei der Implementierung der Web Services beachtet werden müssen, sind die in der Schnittstelle des Dienstes angebotenen Funktionalitäten, sowie das Format der Input- und Outputdaten, welche bereits in den vorhergehenden Abschnitten betrachtet worden sind.

Nachdem mit dem Aspekt der Implementierung der Web Services sowie der WSDL-Schnittstellenbeschreibung die Betrachtung der Dienste im Rahmen der Konzeptumsetzung abgeschlossen ist, soll im folgenden Abschnitt die Umsetzung der Orchestrierung mittels der Business Process Execution Language (BPEL) vorgestellt werden. Dabei soll neben der eigentlichen BPEL-Umsetzung auch die Verbindung mit der WSDL-Beschreibung der Web Services hervorgehoben werden.

6.4. BPEL-Orchestrierung

BPEL wurde mit dem Ziel entwickelt, Geschäftsprozesse mit Hilfe von standardisierten und syntaktisch festgelegten Sprachelementen zu modellieren. Der „Programmcode“ wird dabei jedoch als XML-Dokument dargestellt und folgt einer vorgegebenen XML Schema-Definition. Das so erstellte BPEL-Dokument dient im Gegensatz zum WSDL-Dokument aus Abschnitt 6.3.1 als Eingabe für einen Prozessmanager oder eine Workflow-Engine, welche die Anweisungen ausführt. Wie diese Komponente dies letztlich umsetzt und in welcher Programmiersprache diese entwickelt wurde, ist unerheblich. Wichtig ist nur, dass die Komponente eine Schnittstelle zur Übernahme des BPEL-Dokuments besitzt und die darin enthaltenen Anweisungen gemäß der Spezifikation ausführen kann [DJMZ 05].

Die bereits vorgestellte Beschreibungssprache WSDL steht BPEL konzeptionell sehr nahe, was sich insbesondere dadurch ausdrückte, dass sich die Bezeichnungen der Kommunikationstypen in BPEL 1.0 eng an die Typen in WSDL anlehnten. Dies hat sich mit der BPEL-Version 1.1, die im Rahmen dieser Arbeit verwendet wird, auf der syntaktischen Ebene geändert, wobei das Grundkonzept jedoch bestehen blieb (siehe auch Tabelle C.1). Einer der Gründe für die syntaktische Umbenennung ist, dass das BPEL-Dokument die Kommunikation aus Sicht von Geschäftspartnern beschreibt, welche in einem Prozess verschiedene Rollen einnehmen können, wodurch unterschiedliche Services realisiert werden können. Ein WSDL-Dokument hingegen beschreibt genau einen Service, welcher durch einen Kommunikationspartner zur Verfügung gestellt wird [DJMZ 05]. Aus diesem Grund werden die PartnerLink-Definitionen, welche von der Syntax und Positionierung her zum jeweiligen WSDL-File eines Web Service denn zur BPEL-Spezifikation gehören, dennoch innerhalb dieses Abschnittes behandelt.

Da der Prozess, welcher durch das entstehende BPEL-Dokument repräsentiert wird, ebenfalls einen Service im Sinne der SOA darstellt (siehe Abb. 6.2 und 6.15), der durch einen Web Service realisiert werden kann, muss auch für diesen ein WSDL-File spezifiziert werden, was im Unterabschnitt 6.4.1 behandelt werden wird. Die Beschreibung im Rahmen der Orchestrierung umfasst somit eine WSDL-Spezifikation und eine BPEL-Spezifikation für den BPEL-Prozess.

Abbildung 6.15 zeigt zuvor jedoch eine graphische Darstellung des Konzepts, welches in diesem Abschnitt beschrieben wird. Dazu wurde die WSDL-Spezifikation nach Abbildung 6.6 um ein *partnerLinkType*-Element ergänzt. Der dargestellte Service Requestor kommuniziert mit dem Web Service, welcher hinter seiner WSDL-Schnittstellenbeschreibung verborgen ist. Ergänzend wird der Web Service in Abbildung 6.15 mittels BPEL spezifiziert. Diese BPEL-Beschreibung umfasst die Definition von *partnerLink*- und *variable*-Elementen sowie von Aktivitäten. Das BPEL-*partnerLink*-Element referenziert auf die soeben diskutierten *partnerLinkType*-Elemente der beteiligten Web Services und wird im Rahmen der Kommunikationsbeziehung in Abschnitt 6.4.2.2 betrachtet. Die *variable*-Elemente, welche in Abschnitt 6.4.2.1 vorgestellt werden, dienen dem Zwischenspeichern von Nachrichten, die den aufgerufenen Operationen als Input übergeben oder von diesen zurückgegeben werden. Aus diesem Grund beziehen sie sich auf die Datentypen, welche im WSDL-File des angesprochenen Web Service spezifiziert sind.

In Abschnitt 6.4.2.3 werden wichtige BPEL-Aktivitäten vorgestellt, welche bspw. für die Spezifikation der Interaktionen zwischen dem BPEL-Prozess und den Web Services verwendet werden. Diejenigen Aktivitäten, welche zur Beschreibung einer Interaktion zwischen Web Services dienen, verwenden die zuvor

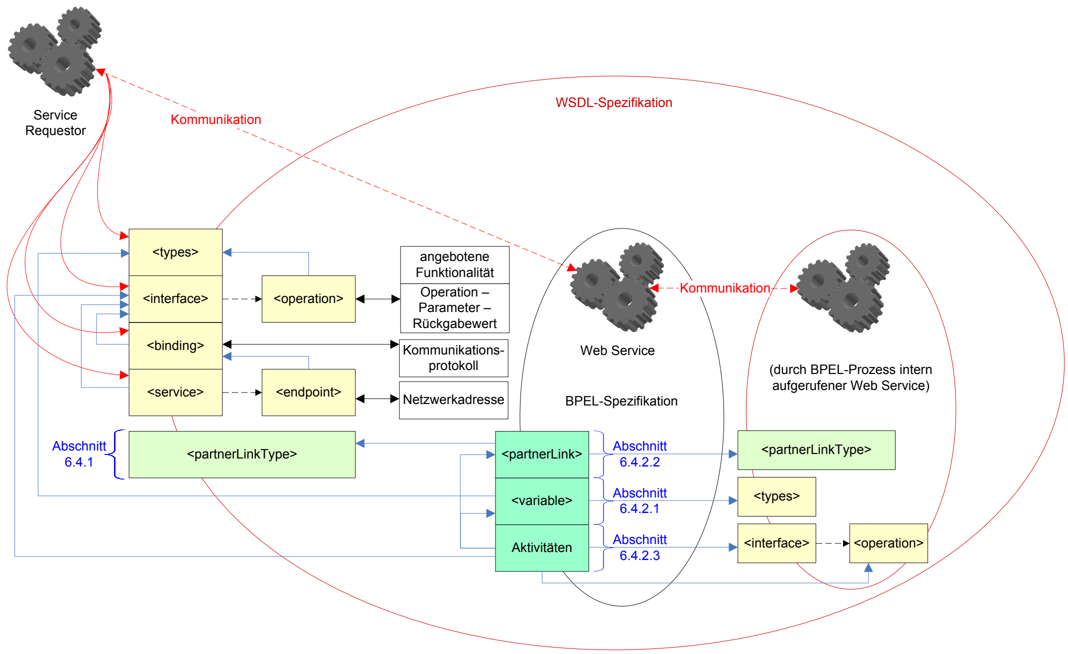


Abbildung 6.15.: Grundkonzept der BPEL-Spezifikation

definierten *partnerLink*- und *variable*-Elemente. Des Weiteren muss über Aktivitätsattribute das WSDL-Interface sowie die Operation angegeben werden, über die mit dem Web Service kommuniziert werden soll. Die in Abbildung 6.15 dargestellt Kommunikation zwischen dem Web Service, der mittels BPEL spezifiziert wurde, und dem intern aufgerufenen Web Service wird vor dem Service Requestor verborgen. Der BPEL-Service nimmt innerhalb der internen Kommunikation selbst die Rolle des Service Requestors ein, welcher die Funktionalität eines anderen Web Service nutzt. Für diese interne Kommunikation gelten deshalb die gleichen Bedingungen wie für die Kommunikation zwischen dem Service Requestor und dem BPEL-Service.

Wie bereits in Zusammenhang mit Abbildung 6.6 erklärt wurde, stellt die WSDL-Schnittstellenbeschreibung eine Art Mauer dar, welche die interne Realisierung der angebotenen Funktionalitäten verbirgt. Da aus diesem Grund auch die intern aufgerufenen Operationen anderer Web Services genauso wie sämtliche spezifizierten Datentypen vor dem Service Requestor verborgen werden, tauchen diese nur dann in der WSDL-Spezifikation des BPEL-Services auf, wenn er diese ebenfalls definiert hat. In diesem Fall müssen die Beschreibungen einander jedoch nicht entsprechen. Beispielsweise kann der intern aufgerufene Web Service einen Datentyp „ID" vom XML Schema-Typ *xsd:integer* definiert haben, wohingegen der Datentyp „ID" in der WSDL-Spezifikation des BPEL-Services vom Typ *xsd:string* ist.

Im Anschluss an Abschnitt 6.4.1, welcher sich mit dem *partnerLinkType*-Element befasst, werden im Abschnitt 6.4.2 zuerst wichtige BPEL-Elemente sowie das Zusammenspiel der BPEL- und WSDL-Spezifikationen betrachtet. Außerdem wird beschrieben, wie aus der konzeptionellen Beschreibung der Orchestrierung im vorigen Kapitel der tatsächliche BPEL-Code abgeleitet wird. In diesem Zusammenhang wird ein beispielhaftes BPEL-Dokument sukzessive aufgebaut und erklärt. Für dieses Beispiel wird die WSDL-Spezifikation aus Abbildung 6.14 verwendet, die um ein *partnerLinkType*-Element erweitert wird, sowie die WSDL-Spezifikation für den BPEL-Prozess, welche im nächsten Abschnitt vorgestellt wird (siehe Abb. 6.18).

6.4.1. WSDL-Spezifikation für den BPEL-Prozess

Der Ausgangspunkt für die Betrachtung der WSDL-Spezifikation in Abschnitt 6.3.1 waren die funktionalen Kapseln, die verschiedenen Diensten zugeordnet wurden. Durch die BPEL-Orchestrierung, welche im nachfolgenden Abschnitt beschrieben wird, sollen die Web Services und damit auch die in ihnen enthaltenen funktionalen Kapseln wieder zu einem Prozess vereinigt werden. Der BPEL-Prozess fasst also die Funktionalitäten wieder zu einer logischen Einheit zusammen und stellt gemäß der Definition auf Seite 56 ebenfalls eine funktionale Kapsel dar. Daraus ergibt sich auch für diesen realisierenden BPEL-Prozess wie für jeden anderen Web Service die Notwendigkeit einer Schnittstellenspezifikation in WSDL, die in diesem Abschnitt betrachtet werden soll.

Abbildung 6.16 zeigt einen solchen als Web Service spezifizierten BPEL-Prozess, der mit mehreren Web Services interagiert. Die Interaktion zwischen den Diensten erfolgt über in WSDL spezifizierte Schnittstellen. Die vier Grundelemente `<types>`, `<interface>`, `<binding>` und `<service>` einer WSDL-Spezifikation wurden schon in Abschnitt 6.3.1 ausführlich behandelt, deshalb befasst sich dieser Abschnitt mit dem *partnerLinkType*-Element, welches in der Abbildung dargestellt ist.

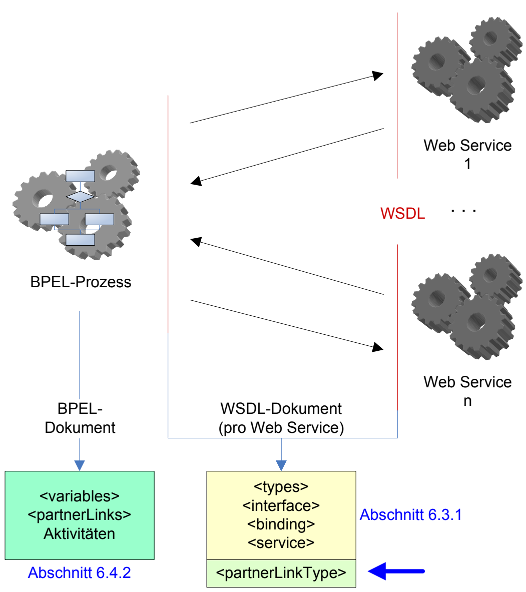


Abbildung 6.16.: WSDL-Spezifikation und BPEL

Auch wenn das *partnerLinkType*-Element nicht Teil der WSDL-Spezifikation ist, so wird es dennoch im WSDL-File spezifiziert. Der zu verwendende Namensraum ist dann allerdings *„http://schemas.xmlsoap.org/ws/2003/05/partner-link/"*.

```
<description ...
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">
  ...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname" />
    </plnk:role>
    <plnk:role name="ncname">
      <plnk:portType name="qname" />
    </plnk:role>?
  </plnk:partnerLinkType>*
</description>
```

Wie dem abgebildeten Schema zu entnehmen ist, müssen dem *partnerLinkType*-Element sowie den *role*-Elementen Bezeichner zugewiesen werden. Diese werden später im BPEL-Dokument für die Definition einer Kommunikationsverbindung zwischen den Diensten verwendet. Einem *partnerLinkType*-Element können je nach Kommunikationsart ein oder zwei Rollen zugewiesen werden. Die Kommunikationsart richtet sich nach dem im *portType*-Element referenzierten *interface*-Element.

Abbildung 6.17 zeigt die zwei unterschiedenen Kommunikationsarten, welche Einfluss auf die Spezifikation des *partnerLinkType*-Elements haben. Im Fall der synchronen Kommunikation wird ein *partnerLinkType*-Element mit einer Rolle definiert, wohingegen bei einer asynchronen Kommunikation zwei Rollen angegeben werden müssen, da die Kommunikation in eine Hin- und Rückrichtung zerlegt wurde (siehe Seite 89).

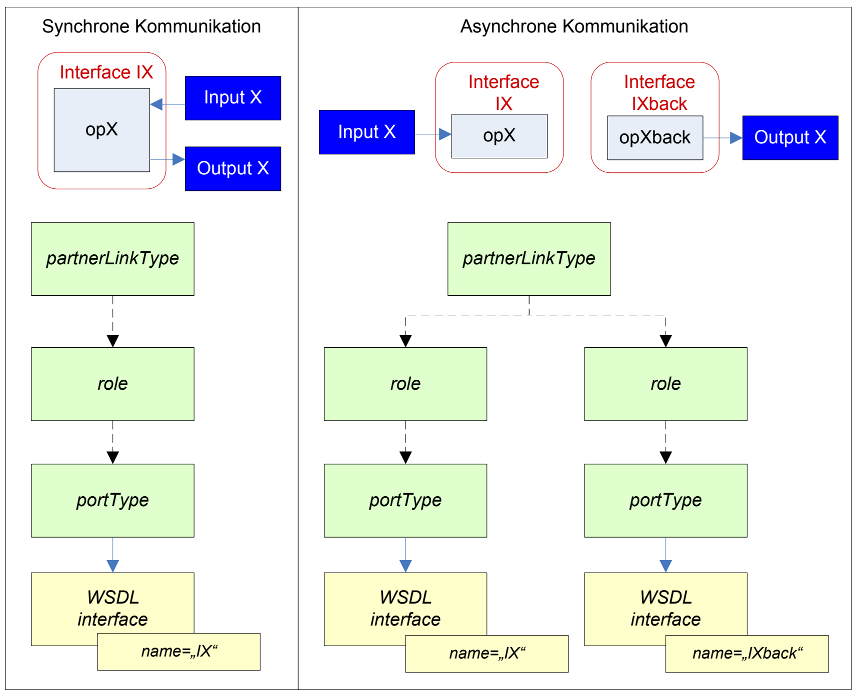


Abbildung 6.17.: Schematische Beispielumsetzung der *partnerLinkType*-Beschreibung

Beispiel

Da für die BPEL-Spezifikation im folgenden Abschnitt nur die Typ- und Interfacebeschreibung sowie die gerade vorgestellte *partnerLinkType*-Beschreibungen verwendet werden, soll im Rahmen der WSDL-Spezifikation des BPEL-Prozesses in 6.18 darauf verzichtet werden, das Binding und die Service-Endpunkt-Spezifikation zu beschreiben. Dies kann, dem Verfahren von Abschnitt 6.3.1.2 folgend, leicht ergänzt werden.

Abbildung 6.18 zeigt die vereinfachte WSDL-Spezifikation für einen möglichen BPEL-Prozess. Das abgebildete *description*-Element trägt den Namen „IncidentService“ und enthält eine Typbeschreibung, welche neben dem „IncidentStatus“ einen Typ „IncidentIDBearbeiter“ umfasst, der sich aus einer „IncidentID“ und einem „Bearbeiter“ zusammensetzt. Der Datentyp „Bearbeiter“ sowie der darauf aufbauende Datentyp „IncidentIDBearbeiter“ wurde eingeführt, um später die in BPEL mögliche Datentransformation zu veranschaulichen (siehe Abb. 6.30).

```
<?xml version="1.0" encoding="UTF-8"?>
```

Bezeichner des BPEL-Web Services

```
<description name="IncidentService"
 targetNamespace="http://example.org/bpel/"
 xmlns="http://www.w3.org/2005/08/wsdl"
 xmlns:tns="http://example.org/bpel/"
 xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
      http://www.w3.org/2005/08/wsdl/wsdl20.xsd">
```

Typ-beschreibung

Datentyp-beschreibung, die für das Beispiel definiert wurde.

Datenformate festlegen

```
 <types>
  <xsd:schema
    targetNamespace="http://example.org/bpel/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    <xsd:element name="IncidentStatus" type="Status" />
    <xsd:element name="IncidentIDBearbeiter" />
     <xsd:complexType>
       <xsd:complexContent>
         <xsd:restriction base="xsd:anyType">
           <xsd:sequence>
             <xsd:element name="IncidentID" type="Identifikationsnummer" />
             <xsd:element name="Bearbeiter" type="xsd:string" />
           </xsd:sequence>
         </xsd:restriction>
       </xsd:complexContent>
     </xsd:complexType>
    </xsd:element>
    <xsd:simpleType name="Identifikationsnummer">
     <xsd:restriction base="xsd:long"/>
    </xsd:simpleType>
    <xsd:simpleType name="Status">
     <xsd:restriction base="xsd:string">
       <xsd:enumeration value="angenommen" />
       <xsd:enumeration value="inBearbeitung" />
       <xsd:enumeration value="zurückgestellt" />
       <xsd:enumeration value="weitergeleitet" />
       <xsd:enumeration value="gelöst" />
       <xsd:enumeration value="geschlossen" />
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:schema>
 </types>
```

Interface-beschreibung

```
 <interface name="IncidentAnnahme">
   <operation name="opCall"
     pattern="http://www.w3.org/2005/08/wsdl/in-only">
     <input messageLabel="In" element="tns:IncidentIDBearbeiter" />
   </operation>
 </interface>
 <interface name="ClientCallback">
   <operation name="opCallback"
     pattern="http://www.w3.org/2005/08/wsdl/out-only">
     <output messageLabel="Out" element="tns:IncidentStatus" />
   </operation>
 </interface>
 ...
```

PartnerLink-Beschreibung

Rolle für Annahme der Anfrage

Rolle für Zurücksenden der Antwort

```
 <plnk:partnerLinkType name="incidentService">
   <plnk:role name="iService">
     <plnk:portType name="tns:IncidentAnnahme" />
   </plnk:role>
   <plnk:role name="iServiceCustomer">
     <plnk:portType name="tns:ClientCallback" />
   </plnk:role>
 </plnk:partnerLinkType>

<description>
```

Abbildung 6.18.: Auszug der WSDL-Spezifikation des BPEL-Prozesses

Neben der Typbeschreibung sind in Abbildung 6.18 auch zwei *interface*-Elemente („IncidentAnnahme“ und „ClientCallback“) definiert. Im Vergleich zum WSDL-File in Abbildung 6.14 spezifizieren die *operation*-Elemente hier jeweils nur eine Kommunikationsrichtung. Die Operation „opCall“ nimmt eine Anfrage im Format „IncidentIDBezeichner“ entgegen, und die Operation „opCallback“ sendet eine Nachricht vom Typ „IncidentStatus“.

Da im Rahmen dieser Arbeit der Incident-Management-Prozess als Referenz dient und die Kommunikation mit dem BPEL-Prozess deshalb asynchron erfolgen soll, wurde diese Trennung der Kommuniktionsrichtungen für das Beispiel gewählt.

Das dritte abgebildete Element ist das *partnerLinkType*-Element. Dieses Element umfasst aufgrund der gewählten asynchronen Kommunikation zwei Rollen. Das *role*-Element „iService“ wird durch sein *portType*-Element an das Interface „IncidentAnnahme“ gebunden. Die zweite Rolle „iServiceCustomer“ referenziert auf das Interface „ClientCallback“.

In diesem Abschnitt wurde die im Abschnitt 6.3.1 vorgestellte WSDL-Spezifikation und die daraus resultierende Beschreibung der Schnittstelle eines Web Service durch die Einführung eines *partnerLinkType*-Elements ergänzt. Dieses Element, das selbst weder Teil der WSDL- noch der BPEL-Spezifikation ist, stellt den WSDL-seitigen Teil der Kommunikationsbeziehung zwischen den beiden Spezifikation dar. Aus diesem Grund wird es im Rahmen der nun folgenden Beschreibung des BPEL-Prozesses nochmals betrachtet werden.

6.4.2. BPEL-Spezifikation

In diesem Abschnitt werden einige BPEL-Elemente vorgestellt. Das Augenmerk liegt dabei nicht auf der Syntax dieser Elemente, sondern auf ihrer Bedeutung bzw. Funktion, welche kurz erläutert wird. Die konkrete Syntax bzw. die allgemeine Struktur eines BPEL-Dokuments ist auszugsweise im Anhang C dargestellt. Der verwendete Namensraum für die abgebildeten Grammatiken ist *„http://schemas.xmlsoap.org/ws/2003/03/business-process/“*, falls nichts anderes angegeben wird. Neben den Funktionen der vorgestellten Elemente wird in diesem Abschnitt auch das Zusammenspiel zwischen den WSDL-Files der angesprochenen Web Services und dem koordinierenden BPEL-Dokument schematisch aufgezeigt.

Zwischen einer BPEL-Spezifikation und der WSDL-Schnittstellenbeschreibung für einen Web Service lassen sich drei Beziehungen identifizieren. Im Zuge der Datenbeziehung werden die WSDL-Datentypen mit den BPEL-Datenvariablen in Verbindung gebracht. Die Kommunikationsbeziehung beschreibt die Kommunikationsverbindung zwischen dem BPEL-Prozess und einem Web Service, welche bei einer Interaktion mit diesem wird. Als dritte Beziehung wird die Interaktionsbeziehung in diesem Abschnitt betrachtet, welche auf den anderen beiden Beziehungen aufbaut.

Abbildung 6.19 zeigt die drei Bereiche einer BPEL-Spezifikation. Zuerst werden die *variable*-Elemente für die Modellierung der Datenbeziehung betrachtet, bevor die Kommunikationsbeziehung unter Verwendung der *partnerLink*-Elemente definiert wird. Schließlich werden im Rahmen der Interaktionsbeziehung ausgewählte Aktivitäten sowohl aus dem Bereich „Basisaktivitäten“ als auch „steuernde“ Aktivitäten vorgestellt.

Entsprechend dem Vorgehen bei der WSDL-Spezifikation der funktionalen Kapseln eines Web Services sollen auch bei der nun folgenden Beschreibung zuerst die Daten der auszutauschenden Nachrichten betrachtet werden. Anschließend werden die Kommunikationsbeziehungen definiert, bevor die eigentliche Interaktionsabfolge mittels BPEL modelliert wird, da die spezifizierten Aktivitäten auf diesen und den zuvor definierten Daten aufbauen.

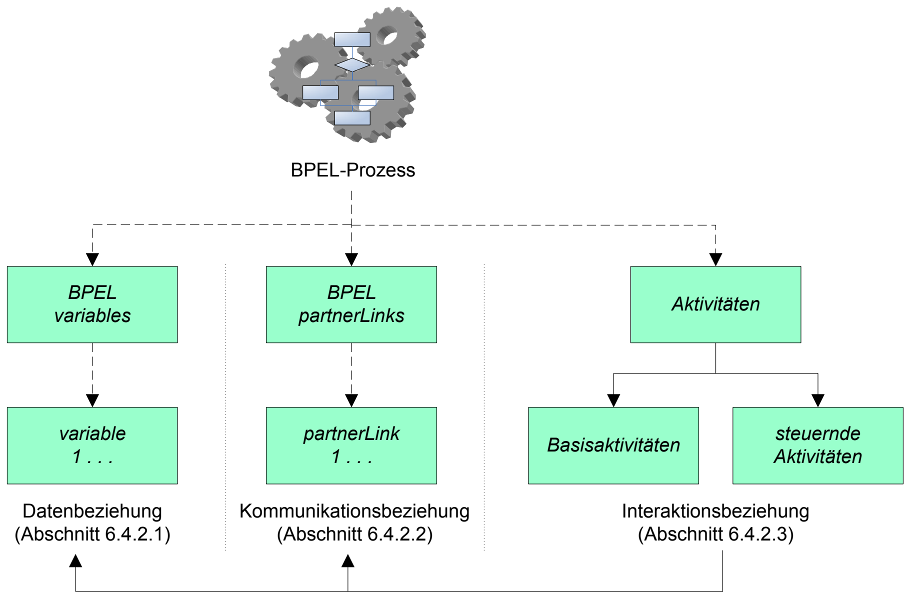


Abbildung 6.19.: Die drei Teile einer BPEL-Spezifikation und ihre Realisierung

6.4.2.1. Datenbeziehung

Ein zustandsbehafteter Prozess wie der Incident-Management-Prozess hängt von den ausgetauschten Nachrichten und den temporären Daten ab, die solche Zustände repräsentieren (vgl. [DJMZ 05]). Für die Modellierung dieser Informationen werden *variable*-Elemente verwendet, die in einem Block zusammengefasst werden:

```
<variables>
  <variable name="ncname"
   messageType="qname"? type="qname"? element="qname"? />+
</variables>?
```

Innerhalb des umgebenden Gültigkeitsbereichs[10] dürfen keine zwei Variablen mit demselben Bezeichner definiert sein. Der Bezeichner einer Variable wird durch dessen *name*-Attribut festgelegt. Über die drei optionalen Attribute kann als Datentyp des *variable*-Elements ein WSDL-*message-type*, ein XML Schema-*simpleType* oder ein XML Schema-*element* angegeben werden. Um einer Variable den Wert einer anderen Variable oder einen neu berechneten Wert zuweisen zu können, wird das *assign*-Element verwendet, dessen allgemeine Grammatik in Anhang C abgebildet ist[11]. Zum Auswerten von Variablenwerten, Eigenschaften sowie dem Status einer Verknüpfung spezifiziert BPEL basierend auf XPath 1.0 Funktionen, die über den Namensraum *„http://schemas.xmlsoap.org/ws/2003/03/business-process/“* verwendet werden können.

Grundsätzlich benötigt der BPEL-Prozess, wie in Abbildung 6.20 dargestellt, für jede später aufgerufene Operation eines jeden Web Service eine Variable für die benötigten bzw. gelieferten Daten. Wurden die Datentypen in WSDL bezogen auf den Gesamtprozess konsistent modelliert, so können eventuell übereinstimmende Inputs und Outputs von Operationen in einer Variable zusammengefasst werden. Neben den Variablen für die Interaktionen mit den Web Services müssen noch Variablen definiert werden, welche aus der WSDL-Spezifikation des BPEL-Prozesses resultieren.

[10] Dieser Gültigkeitsbereich kann mit einem <scope>-Element definiert werden. Siehe BPEL-Spezifikation [IBM 03].
[11] siehe Beschreibung der <!– activity –>-Elemente im Anhang

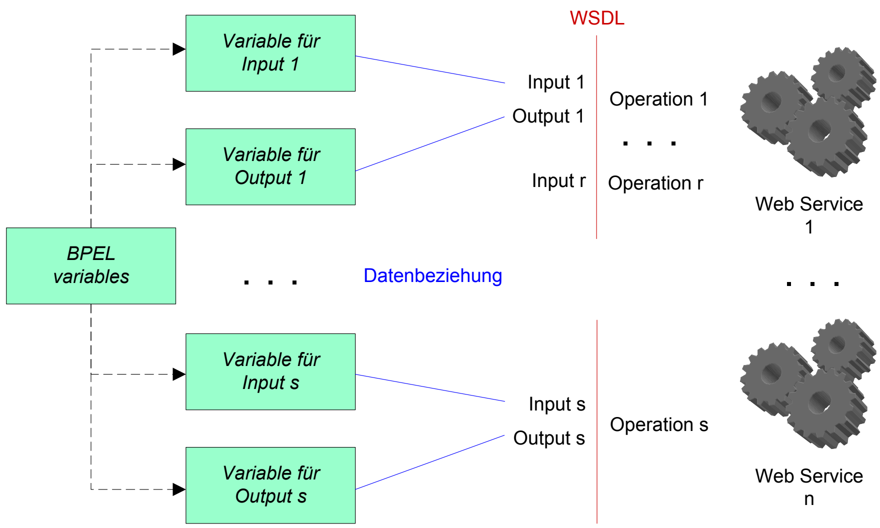


Abbildung 6.20.: Zusammenhang BPEL-*variable*-Elemente und WSDL-Beschreibung

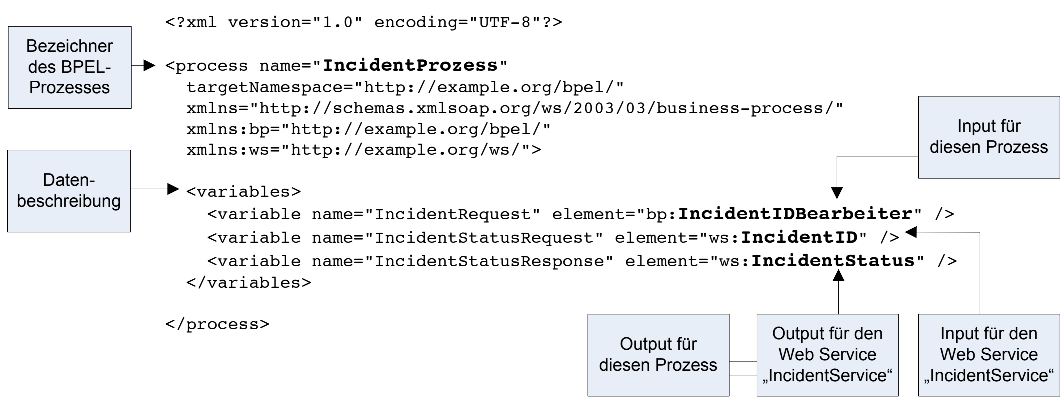


Abbildung 6.21.: Beispielcodierung der Datenvariablen

Im Falle des Incident-Management-Prozesses bildet der Incident Record die zentrale Datenstruktur des Prozesses und sollte deshalb nicht von Web Service zu Web Service verschieden modelliert werden, auch wenn z.B. mittels des *assign*-Elements Datenfransformationen möglich sind. Um den Transformationsaufwand gering zu halten, bietet es sich deshalb an, wiederkehrende Datenstrukturen wie den Incident Record oder andere Teilstrukturen serviceübergreifend konsistent zu gestalten.

Beispiel

Betrachten wir zur Verdeutlichung das im Abschnitt 6.3.1 schrittweise erstellte WSDL-Dokument (vgl. Abb. 6.14). Soll später eine Interaktion über die dort spezifizierte Operation „opKA“ mit dem dahinter stehenden Web Service erfolgen, so muss der BPEL-Prozess *variable*-Elemente für die benötigten Datenstrukturen bereitstellen. Abbildung 6.21 zeigt die ersten Elemente der BPEL-Spezifikation, welche in den folgenden Abschnitten sukzessive um den jeweils betrachteten Aspekt ergänzt wird.

Als Bestandteil des *process*-Elements sind neben dem Namen des Prozesses (*name="IncidentProzess"*) und dem BPEL-Namensraum *„http://schemas.xmlsoap.org/ws/2003/03/business-process/“* noch die Namensräume *„bp“* für die BPEL-Beschreibung und *„ws“* als Ort für das WSDL-File aus Abbildung 6.14 zu sehen.

Die drei *variable*-Elemente repräsentieren für das Beispiel die in diesem Abschnitt betrachtete Datenbeschreibung. Die Operation „opKA“ des Web Service „IncidentStatusService“, die im Rahmen der Interaktionsbeschreibung im Abschnitt 6.4.2.3 aufgerufen wird, benötigt einen Input vom Typ „IncidentID“ und liefert einen Output der Form „IncidentStatus“. Für diese beiden Datentypen wurden die *variable*-Elemente „IncidentStatusRequest“ und „IncidentStatusResponse“ definiert. Die Datenvariable „IncidentStatusResponse“ soll dabei gleichzeitig auch noch als Output für den BPEL-Prozess verwendet werden. Als Input für den BPEL-Prozess dient ein *variable*-Element „IncidentRequest“ vom Typ „IncidentIDBearbeiter“, der im WSDL-File des BPEL-Prozesses definiert wurde (siehe Abb. 6.18).

Das Spezifizieren von Datenvariablen reicht jedoch nicht aus, um mit dem Web Service zu interagieren. Damit die Operation „opKA“ aufgerufen werden kann, muss zuerst eine Kommunikationsbeziehung mit dem entsprechendem Web Service definiert werden. Der nächste Abschnitt befasst sich sowohl mit den BPEL-seitigen als auch den WSDL-seitigen Elementen dieser Definition.

6.4.2.2. Kommunikationsbeziehung

Damit ein BPEL-Prozess innerhalb eines WSDL-Files spezifizierte Interface-Operationen aufrufen kann, muss zuerst eine Kommunikationsbeziehung zwischen diesen beiden definiert werden. Dazu wird das im WSDL-File definierte *partnerLinkType*-Element und in der BPEL-Spezifikation das *partnerLink*-Element benötigt. Abbildung 6.22 zeigt die notwendigen Kommunikationselemente eines BPEL-Dokuments und ihre Beziehung zu einer WSDL-Interfacebeschreibung eines Web Service. In der Mitte der Abbildung ist das BPEL-*partnerLink*-Element mit seinem *partnerLinkType*-Attribut dargestellt, welche Bezug auf die WSDL-Interface-Spezifikation eines Web Service nehmen, die ihrerseits über ein Binding an einen Service-Endpoint gebunden ist. Die dargestellten Aktivitäten „receive“, „reply“, „pick“ und „invoke“ werden im nächsten Abschnitt betrachtet.

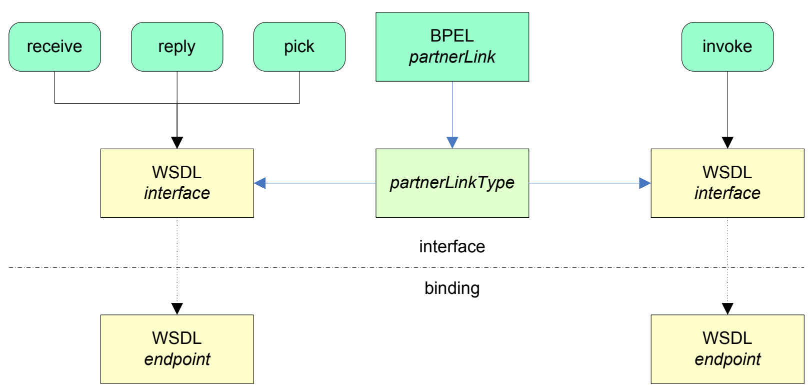


Abbildung 6.22.: Kommunikationselemente eines BPEL Dokuments (vgl. [DJMZ 05])

Im BPEL-Dokument wird auf die abstrakte Schnittstellenbeschreibung eines Web Services in einem WSDL-Dokument referenziert. Da WSDL keine Beschreibung von Prozessaspekten vorsieht, wird mittels des in Abschnitt 6.4.1 ergänzten *partnerLinkType*-Elements spezifiziert, wie die Verbindung vom BPEL-Dokument zum abstrakten Teil eines WSDL-Dokuments aussehen soll. Wie ein Prozess mit seinem Partner kommuniziert, wird dann mit Hilfe eines *partnerLink*-Elements in der BPEL-Prozessbeschreibung festgelegt. Die allgemeine Grammatik eines BPEL-*partnerLink*-Elements, welches innerhalb eines *partnerLinks*-Elements zusammengefasst wird, ist nachfolgend dargestellt:

```
<partnerLinks>
  <partnerLink name="ncname" partnerLinkType="qname"
    myRole="ncname"? partnerRole="ncname"? >
  </partnerLink>+
</partnerLinks>?
```

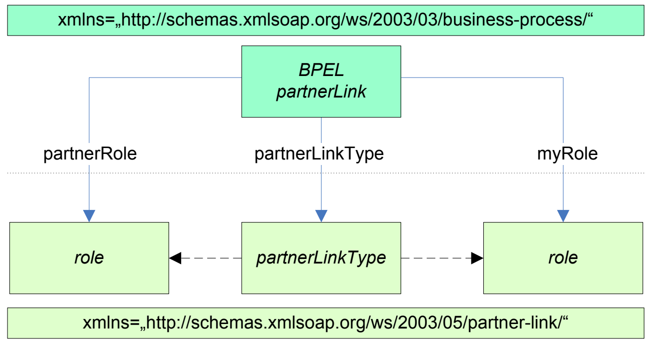


Abbildung 6.23.: Zusammenhang: BPEL-*partnerLink*-Element und *partnerLinkType*-Element

Für jede mögliche Kommunikationsverbindung, die zwischen den Web Services genutzt werden könnte, muss ein „PartnerLink“ definiert werden. Anhand eines eindeutigen Namens können dann die verschiedenen Kommunikationsverbindungen unterschieden werden. Außerdem können einem *partnerLink*-Element über die nachfolgenden Attribute bis zu zwei Rollen innerhalb der Kommunikationsverbindung zugewiesen werden.

myRole: Dieses Attribut enthält den eindeutigen Bezeichner für die Rolle des eigenen Prozesses, in welcher er mit dem Partner kommuniziert.

partnerRole: Dieses Attribut spezifiziert durch einen eindeutigen Bezeichner die Rolle des Partners in der Kommunikationsverbindung.

Die Verbindung zum Web Service wird, wie in Abbildung 6.23 dargestellt, mittels des BPEL-*partnerLinkType*-Attributs hergestellt, das auf ein *partnerLinkType*-Element in WSDL (siehe Abb. 6.23) referenziert und eine Art Container für alle Kommunikationsbeziehungen zwischen den beiden Partnern darstellt. Durch Definition dieser WSDL-*partnerLinkType*-Elements soll das Prinzip der losen Kopplung nicht verletzt werden, denn auch eine direkte Referenzierung auf bestimmte WSDL-Elemente wäre möglich, wodurch dann aber die BPEL-Spezifikation vom WSDL-Spezifizierungsprozess abhängig wäre.

Neben der Beziehung zwischen dem *partnerLinkType*-Attribut in BPEL und dem *partnerLinkType*-Element in WSDL zeigt die Abbildung 6.23 auch die Verbindung der Kommunikationsrollen (BPEL-Attribute *myRole* und *partnerRole*) mit den im WSDL-Dokument innerhalb des *partnerLinkType*-Elements spezifizierten *role*-Elementen.

Bei der Betrachtung von Abbildung 6.23 ist es wichtig, die unterschiedlichen Namensräume der Elemente zu beachten. Für die BPEL-Spezifikation des *partnerLink*-Elements ist der Namensraum *„http://schemas.xmlsoap.org/ws/2003/03/business-process/“* zu verwenden, wohingegen die Spezifikation des *partnerLinkType*-Elements sowie seiner Kindelemente im Namensraum *„http://schemas.xmlsoap.org/ws/2003/05/partner-link/“* innerhalb des WSDL-Files erfolgt.

Beispiel

Für das im Abschnitt 6.3.1 erstellte WSDL-File (siehe Abb. 6.14) zeigt Abbildung 6.24 links die Ergänzung des *partnerLinkType*-Elements. Innerhalb dieses Elements mit dem Namen „iStatus“ wurde eine Rolle „iStatusService“ definiert, die auf das „KapselAInterface“-Element referenziert. Rechts in Abbildung 6.24 ist ein Auszug aus dem WSDL-File des BPEL-Prozesses (Abb. 6.18) zu sehen. Darin ist innerhalb des *partnerLinkType*-Elements („iService“) für jedes Interface eine Kommunikationsrolle definiert.

Abbildung 6.25 nimmt den im vorherigen Abschnitt definierten Code des BPEL-Beispiels (Abb. 6.21) und erweitert ihn um die Spezifikation der *partnerLink*-Elemente. Für den Aufruf der Operation „opKA“ der WSDL-Spezifikation, die auch im Auszug links in Abbildung 6.24 zu sehen ist, wird das *partnerLink*-Element „incidentStatusService“ definiert. Dieses referenziert auf das *partnerLinkType*-Element „iStatus“

Spezifikation innerhalb des WSDL-File des jeweiligen Web Service	
Ergänzung zur WSDL-Beschreibung in Abb. 6.14	Siehe WSDL-Spezifikation des BPEL-Prozesses in Abb. 6.18

```
<?xml version="1.0" encoding="UTF-8"?>

<description name=“IncidentStatusService“
 targetNamespace="http://example.org/ws/"
 xmlns="http://www.w3.org/2005/08/wsdl"
 xmlns:tns="http://example.org/ws/"
 xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:plnk=
 "http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
     http://www.w3.org/2005/08/wsdl/wsdl20.xsd">
 ...
 <interface name="KapselAInterface">
   <operation name="opKA"
     pattern="http://www.w3.org/2005/08/wsdl/in-out">
     <input messageLabel="In"
      element="tns:IncidentID" />
     <output messageLabel="Out"
      element="tns:IncidentStatus“ />
   </operation>
 </interface>

 <plnk:partnerLinkType name="iStatus">
   <plnk:role name="iStatusService">
     <plnk:portType name="tns:KapselAInterface" />
   </plnk:role>
 </plnk:partnerLinkType>
 ...
</description>
```

```
<?xml version="1.0" encoding="UTF-8"?>

<description name=“IncidentService“
 targetNamespace="http://example.org/bpel/"
 xmlns="http://www.w3.org/2005/08/wsdl"
 xmlns:tns="http://example.org/bpel/"
 xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:plnk=
 "http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
 xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
     http://www.w3.org/2005/08/wsdl/wsdl20.xsd">
 ...
 <interface name="IncidentAnnahme">
   <operation name="opCall"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In"
     element="tns:IncidentIDStatus" />
   </operation>
 </interface>
 <interface name="ClientCallback">
   <operation name="opCallback"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out"
     element="tns:IncidentStatus“ />
   </operation>
 </interface>

 <plnk:partnerLinkType name="incidentService">
   <plnk:role name="iService">
     <plnk:portType name="tns:IncidentAnnahme" />
   </plnk:role>
   <plnk:role name="iServiceCustomer">
     <plnk:portType name="tns:ClientCallback" />
   </plnk:role>
 </plnk:partnerLinkType>
 ...
<description >
```

Abbildung 6.24.: WSDL-seitige Beispielcodierung der Kommunikationsbeziehung

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="IncidentProzess"
  targetNamespace="http://example.org/bpel/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bp="http://example.org/bpel/"
  xmlns:ws="http://example.org/ws/">

  <partnerLinks>
    <partnerLink name="client" partnerLinkType="bp:incidentService"
      myRole="iService" partnerRole="iServiceCustomer"/>
    <partnerLink name="incidentStatusService"
      partnerLinkType="ws:iStatus" partnerRole="iStatusService" />
  </partnerLinks>

  <variables>
    <variable name="IncidentRequest" element="bp:IncidentIDBearbeiter" />
    <variable name="IncidentStatusRequest" element="ws:IncidentID" />
    <variable name="IncidentStatusResponse" element="ws:IncidentStatus" />
  </variables>

</process>
```

Kommunikationsbeschreibung

Abschnitt 6.4.2.1

“partnerLinkType“ im „IncidentService“ -WSDL

Rollen im „IncidentService“ -WSDL

“partnerLinkType“ im „IncidentStatusService“ -WSDL

Abbildung 6.25.: Beispielcodierung der Kommunikationsbeziehung (siehe auch Abb. 6.24) mit der Rolle „iStatusService“. Da in der PartnerLinkType-Ergänzung links in Abbildung 6.24 nur eine Rolle definiert wurde, kann auch bei der Spezifikation in BPEL nur eine Rolle angegeben werden.

Die zweite Kommunikationsverbindung (*partnerLink name=„client“*), die in Abbildung 6.25 definiert ist, dient der Kommunikation mit dem BPEL-Prozess. Diese Kommunikationsbeziehung ist notwendig, da die anfragenden Dienste über das WSDL-File des BPEL-Prozess mit diesem kommunizieren. Ist jedoch keine Verbindung zwischen der WSDL-Spezifikation und dem BPEL-Prozess durch ein *partnerLink*-Element definiert, kann auch keine Interaktion erfolgen, wie bereits zu Beginn dieses Abschnitts erwähnt wurde.

Dieser Abschnitt hat sich mit der Definition von Kommunikationverbindungen zwischen Web Services befasst. Dabei wurden nach einem Überblick über die notwendigen Kommunikationselemente die allgemeine Grammatik der *partnerLink*-Elemente vorgestellt. Im Rahmen des Beispiels wurden anschließend ausgehend von den ergänzten *partnerLinkType*-Definitionen der WSDL-Files in Abbildung 6.24 die Spezifikation der *partnerLink*-Elemente im BPEL-Dokument abgeleitet, die für die im folgenden Abschnitt betrachtete Interaktionsbeziehung notwendig sind.

6.4.2.3. Interaktionsbeziehung

Der dritte grundlegende Bestandteil eines BPEL-Dokuments sind neben den Variablen und PartnerLinks die Aktivitäten, welche auf den in den beiden vorherigen Abschnitten beschriebenen Aspekten aufbauen. Unter einer Aktivität kann dabei z.B. eine Interaktion mit einem Web Service durch den Aufruf einer Operation, das Belegen einer Datenvariable oder auch das Festlegen der Reihenfolge von Interaktionen verstanden werden. Bevor wichtige Aktivitäten aufgelistet werden, sollen diejenigen Attribute und Elemente betrachtet werden, welche bei allen Aktivitäten spezifiziert werden können. Anschließend wird die unterschiedliche Modellierung von synchronen und asynchronen Operationsaufrufen betrachtet, bevor die Orchestrierung auf die BPEL-Aktivitäten abgebildet wird.

Nachfolgend sind drei Attribute, die bei jeder Aktivität spezifiziert werden, dargestellt:

```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?
```

Über das *name*-Attribut kann einer Aktivität ein eindeutiger Bezeichner zugewiesen werden, der es nach einmaliger Definition erlaubt, diese Aktivität an mehreren Stellen zu verwenden. Mit Hilfe des optionalen *joinCondition*-Attibuts kann die Ausführung der Aktivität an mehrere Bedingungen geknüpft werden. Soll ein Fehler beim Zusammenführen zweier z.B. parallel ausgeführter Interaktionspfade unterdrückt werden, muss der Wert des *suppressJoinFailure*-Attributs auf true gesetzt werden.

Neben den zuvor angegeben Attributen können innerhalb einer jeden Aktivität folgende Elemente spezifiziert werden:

```
<source linkName="ncname" transitionCondition="bool-expr"? />*
<target linkName="ncname" />*
```

Die Kindelemente *source* und *target* einer Aktivität dienen der Zuweisung einer Verknüpfung/eines Links. Durch die so definierten Links können Aktivitäten miteinander verknüpft werden, wobei der Fluss der Aktivitäten von der „source“- zur „target“-Aktivität verläuft. Da es keine Rolle spielt, auf welcher Verschachtelungsebene sich die zwei Aktivitäten befinden, sie aber dennoch in eine zeitliche Abfolge gebracht werden und somit die geordnete-hierachische Struktur des Dokuments sprengen können, sollte man von der Verwendung dieser Link-Elemente absehen (siehe [DJMZ 05]).

Neben den eben beschriebenen optionalen Attributen und Kindelementen hat jede Aktivität entsprechend ihrer Aufgaben weitere spezielle Attribute und Elemente, die Anhang C oder direkt der BPEL-Spezifikation zu entnehmen sind. Im Rahmen dieser Arbeit werden nur die Aktivitäten sowie deren Kindelemente und Attribute betrachtet, die bezogen auf die Abbildung der Orchestrierung auf BPEL benötigt werden.

Basisaktivitäten

Ein BPEL-Dokument beschreibt nun das Zusammenspiel einzelner Aktivitäten im Rahmen eines Prozesses. In komplexen Prozessen werden in der Regel vorhandene Dienste (Web Services) als Aktivitäten in den Prozess eingebunden. Nachfolgend sind die Grammatiken einiger Basisaktivitäten abgebildet[12]:

invoke: Durch die *invoke*-Aktivität ruft der BPEL-Prozess Operationen anderer Web Services auf. Die Art der Kommunikation wird durch Sezifikation der Attribute *inputVariable* und *outputVariable* festgelegt. Bei einer synchronen Request-Response-Operation („in-out") wird z.B. die Response-Nachricht in der *outputVariable* gespeichert.

```
<invoke
  partnerLink="ncname"
  portType="qname" operation="ncname"
  inputvariable="ncname"? outputvariable="ncname"? >
</invoke>
```

receive: Mittels der *receive*-Aktivität wartet der BPEL-Prozess auf eingehende Nachrichten. Diese können u.a. zum Starten des BPEL-Prozesses genutzt werden oder allgemein als „Call-Back" verstanden werden.

```
<receive
  partnerLink="ncname"
  portType="qname" operation="ncname"
  variable="ncname"? createInstance="yes|no"? >
</receive>
```

reply: Die *reply*-Aktivität wird zum Senden eines Response bei einem synchronen BPEL-Prozess genutzt, wobei sie sich immer auf das initiale *receive* bezieht, durch das der BPEL-Prozess gestartet wurde.

```
<reply
  partnerLink="ncname"
  portType="qname" operation="ncname"
  variable="ncname"? faultName="qname"? >
</reply>
```

pick: Die *pick*-Aktivität wird dazu verwendet, um auf das Auftreten eines Ereignisses aus einer Menge von Ereignissen zu warten und dann eine mit diesem Ereignis verbundene Aktivität auszuführen (Grammatik siehe Anhang C).

Für die vorgestellten BPEL-Aktivitäten *invoke*, *receive* und *reply* werden nun zuerst die Funktionen einzelner Attribute erklärt, bevor weitere Aktivitäten betrachtet werden. Das *partnerLink*-Attribut verweist auf eine zuvor festgelegte Kommunikationsbeziehung und steht in direkter Verbindung mit derjenigen Operation (*operation*-Attribut), die aufgerufen wird. Über das *portType*-Attribut wird das Interface dieser Operation angegeben. Entsprechend der Operationsspezifikation werden dann die *variable*-Attribute der Aktivitäten mit den Datentypvariablen verbunden.

Steuernde Aktivitäten

Zur Steuerung bzw. Strukturierung der gerade vorgestellten Aktivitäten eines BPEL-Prozesses können folgende Elemente verwendet werden:

sequence: Die innerhalb eines *sequence*-Elements definierten Aktivitäten werden sequenziell in der Reihenfolge der Aufrufe ausgeführt.

```
<sequence>
  <!-- standard-elements -->
  <!-- activity -->+
</sequence>
```

[12] Bei den ausgewählten Grammatiken wurde darauf verzichtet, optionale Kindelemente darzustellen. Siehe dazu Anhang C.

flow: Die innerhalb eines *flow*-Elements definierten Aktivitäten werden parallel ausgeführt.

```
<flow>
  <!-- activity -->+
</flow>
```

switch: Dieses Element ermöglicht, anhand von booleschen Ausdrücken zu entscheiden, welche Aktivität(en) ausgeführt werden.

```
<switch>
  <case condition="bool-expr">
    <!-- activity -->+
  </case>+
  <otherwise>
    <!-- activity -->+
  </otherwise>?
</switch>
```

while: Das *while*-Element dient der Definition von Schleifen, wobei eingeschlossene Aktivitäten so lange ausgeführt werden, bis die Schleifenbedingung nicht mehr erfüllt ist (Grammatik siehe Anhang C).

Die in diesem Abschnitt bisher beschriebenen Elemente sind rechts oben in Abbildung 6.26 noch einmal schematisch dargestellt. Schwerpunkt bilden die Basisaktivitäten, welche unter Verwendung der zuvor definierten Daten- und Kommunikationsbeziehungen die Interaktionsbeziehung beschreiben und innerhalb so genannter steuernde Aktivitäten zusammengefasst werden können, die der Ablaufsteuerung dienen. Für die Basisaktivitäten ist eine Referenz auf die in Abschnitt 6.4.2.1 definierten *variable*-Elemente abgebildet, die je nach Aktivität ein oder zwei Datenvariablen umfasst. Über ihr *partnerLink*-Attribut referenzieren Basisaktivitäten wie z.B. <invoke> auf eine durch ein *partnerLink*-Element definierte Kommunikationsverbindung (siehe Abschnitt 6.4.2.2). Die Attribute *portType* und *operation* dienen der eigentlichen Spezifikation der Interaktion mit dem Web Service, indem die aufzurufende WSDL-Operation und ihr Interface angegeben werden.

Auch wenn die Beziehung der durch eine Basisaktivität referenzierten Elemente in Abbildung 6.26 nur angedeutet werden, so stehen die verwendeten *variable*-, *partnerLink*-, *interface*- und *operation*-Elemente dennoch in einem festen Kontext und können nicht willkürlich gewählt werden. Die Datenvariablen richten sich bspw. nach den verwendeten Datentypen, welche über das *operation*-Element referenziert werden. Ebenso muss die referenzierte Kommunikationsbeziehung zu der aufgerufenen Operation und dem Interface des Web Services passen. Dieser Zusammenhang zwischen den Datenvariablen, Kommunikationselementen und den Aktivitäten eines BPEL-Prozesses und der WSDL-Spezifikation eines Web Services wurde bereits in Abbildung 6.15 graphisch dargestellt.

Nachdem die zur Beschreibung der Interaktionsbeziehung notwendigen Elemente vorgestellt wurden, soll nachfolgend die Abbildung der Orchestrierung auf die BPEL-Aktivitäten betrachtet werden, da über mittels der Aktivitäten die Reihenfolge der Interaktionen mit den Web Services und somit die zu modellierende Prozesslogik festgelegt wird. Dabei wird zuerst auf die unterschiedliche Modellierung synchroner und asynchroner Kommunikation eingegangen, welche bereits wegen der Zerlegung der Kommunikationsrichtungen unterschieden wurden.

Abbildung 6.27 zeigt links die schematische Darstellung einer synchronen und rechts einer asynchronen Interaktion eines BPEL-Prozesses mit einem Web Service. Gemäß der zu modellierenden Kommunikationsart trägt der Bezeichner der Kommunikationsverbindung (*partnerLink*-Element) im synchronen Fall den Bezeichner „SyncService“ und im asynchronen den Bezeichner „AsyncService“. Für die dargestellten Beispiele wird angenommen, dass jeder Datentyp eine Entsprechung in Form eines BPEL-*variable*-Elements mit demselben Bezeichner hat. Beispielsweise existiert für den Datentyp „syncIn“ ein *variable*-Element „syncIn“ mit dem entsprechenden Datenetyp. Im synchronen Fall enthält die WSDL-Spezifikation des Web Services „SyncService“ ein Interface „Isync“. Der darin definierten Operation „opSync“ wird ein Input „syncIn“ übergeben und anschließend ein Output „syncOut“ zurückgegeben. Die synchrone Interaktion wird durch das links abgebildete *invoke*-Element in BPEL beschrieben.

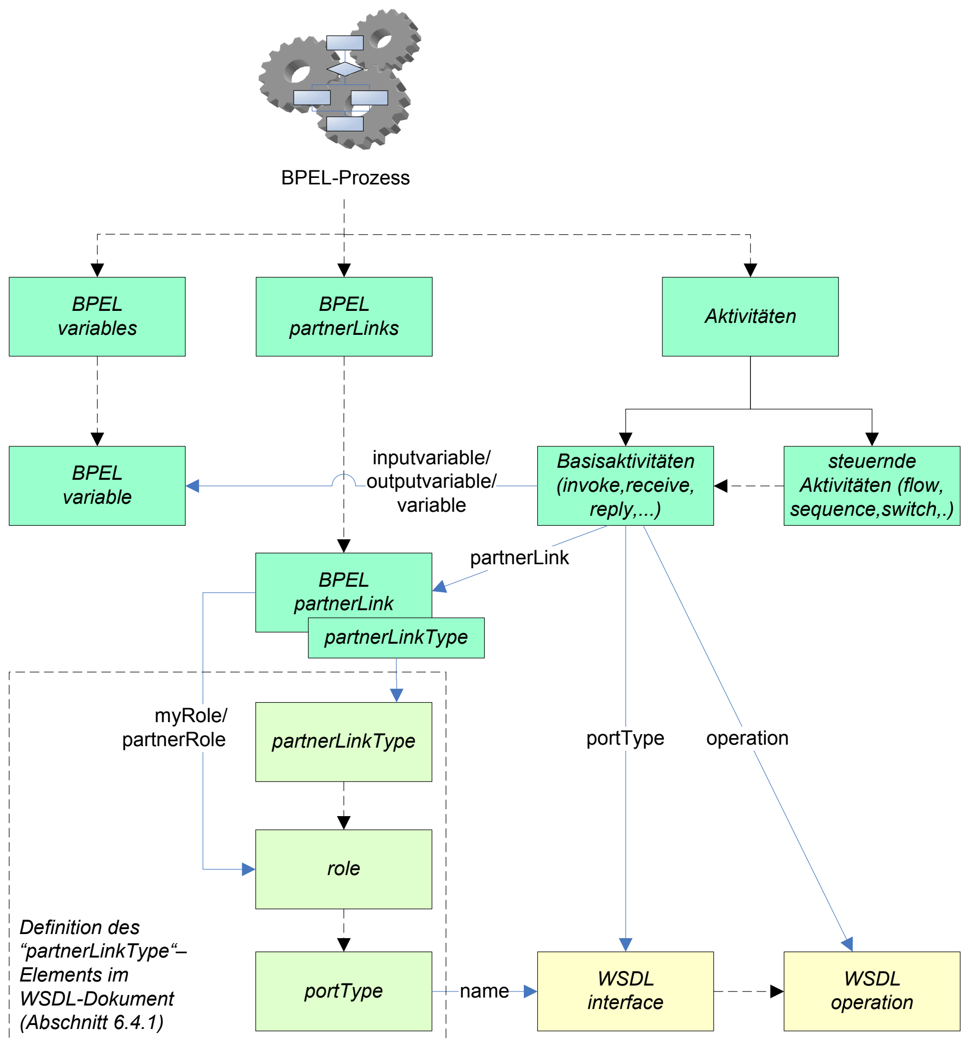


Abbildung 6.26.: Schematische Zusammenfassung der Interaktionsbeschreibung

Für die Beschreibung des asynchronen Falls rechts in Abbildung 6.27 sind zwei BPEL-Aktivitäten zu spezifizieren, da mit Operationen verschiedener Interfaces interagiert wird. Das rechts dargestellte *invoke*-Element sendet eine Anfrage an den Web Service „AsyncService“, indem es die Operation „opCall“ mit dem Inhalt der Datenvariable „asyncIn“ aufruft. Anschließend wird durch das *receive*-Element der Output der Operation „opBack“ in der Datenvariable „asyncOut“ gespeichert. Wird die Interaktion im asynchronen Fall nicht durch den anfragenden Service sondern vom BPEL-Prozess angestoßen, so muss die Reihenfolge des *invoke*- und *receive*-Elements vertauscht werden.

Nachfolgend wird die Abbildung der Orchestrierung auf die BPEL-Aktivitäten betrachtet. Die zu modellierende Prozesslogik besteht dabei aus einer Reihe von Aktivitäten, welche Interaktionen mit den Web Services spezifizieren. Zur Vereinfachung werden nun die Aufrufe von Web-Service-Operationen rein schematisch und unabhängig von der verwendeten Kommunikationsart betrachtet und entsprechend in den Abbildungen durch „Aktivität-Kapsel-A“ usw. dargestellt. Im konkreten Beispiel müssen dann diese Aufrufe durch die in Abbildung 6.27 dargestellten Aktivitäten ersetzt werden. Des Weiteren werden keine eventuell notwendigen Datentransformationen betrachtet.

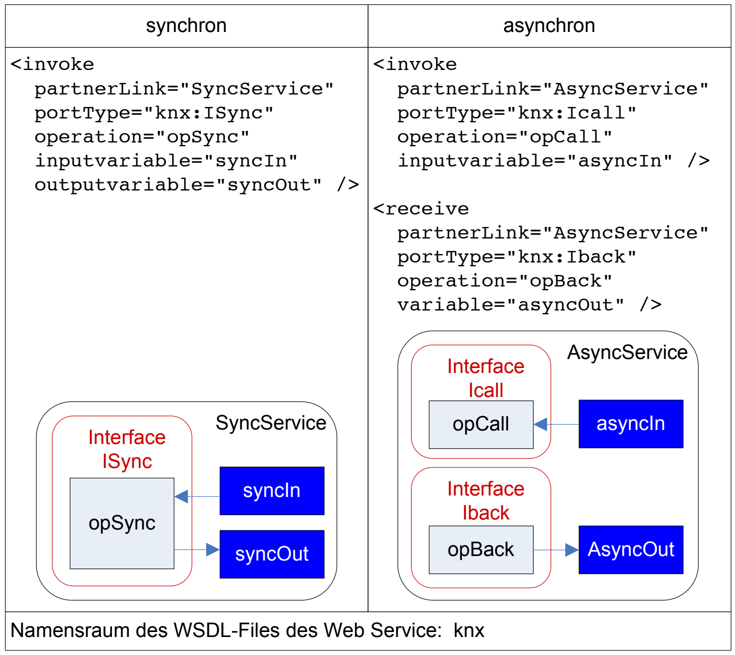


Abbildung 6.27.: Synchroner vs. asynchroner Interaktionsaufruf

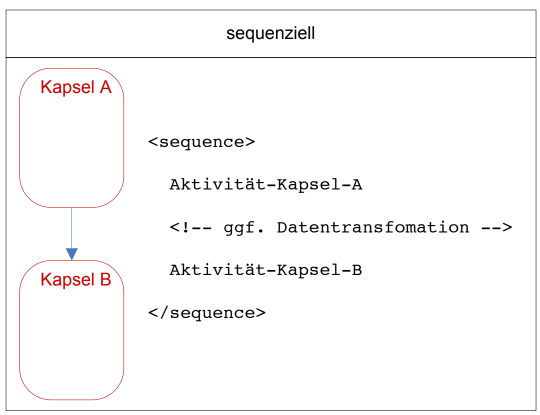


Abbildung 6.28.: sequenzieller Interaktionsaufruf

Abbildung 6.28 zeigt die Umsetzung einer sequenziellen Abarbeitung unter Verwendung des *sequence*-Elements. Innerhalb diesem wird zuerst die in Kapsel A spezifizierte Funktionalität ausgeführt, bevor mit dem Web Service interagiert wird, welcher Kapsel B realisiert.

Verzweigt sich der Prozessstrang, nachdem eine Kapsel A ausgeführt wurde, so sind prinzipiell zwei Situationen vorstellbar. Sollen und können wie links in Abbildung 6.29 abgebildet die Kapseln B und C parallel ausgeführt werden, so müssen deren Interaktionsaufrufe durch ein *flow*-Element eingeschlossen werden. Können die Kapseln B und C nicht parallel ausgeführt werden, so werden sie entsprechend ihrer Abhängigkeiten sequenziell hintereinander ausgeführt, wie am Beispiel der Kapseln A und B in Abbildung 6.28 beschrieben wurde.

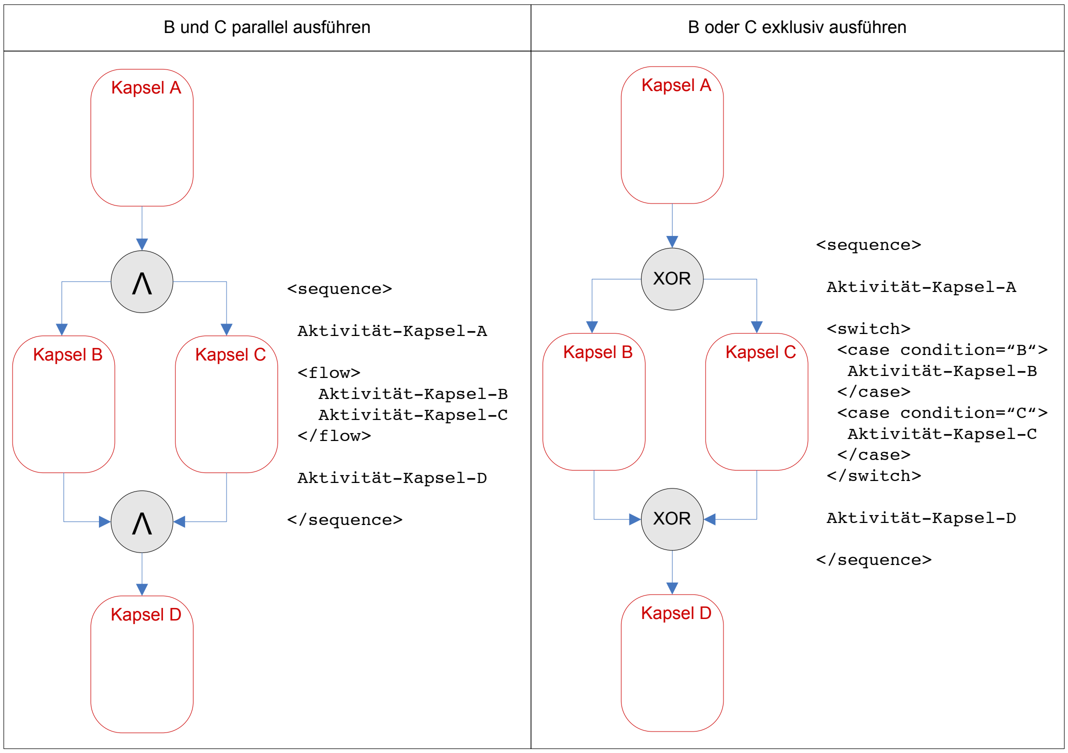


Abbildung 6.29.: paralleler und exklusiver Interaktionsaufruf

Sind die getrennten Teilstränge nicht durch eine Und-Verknüpfung sondern durch eine XOR-Verknüpfung miteinander verbunden, so ist ein *switch*-Element zu verwenden, wie rechts in Abbildung 6.29 dargestellt ist. Für jeden Teilstrang ist dann zur Festlegung der Ausführungsbedingung ein boolescher Ausdruck zu formulieren. In der Regel wird die Entscheidung, welche der Aktivitäten ausgeführt werden muss, dann anhand des Ergebnisses der Kapsel A getroffen.

Eine Oder-Verknüpfung, in der ein oder mehrere Teilsträng ausgeführt werden müssen, ist in BPEL nicht direkt auf ein Element abbildbar. In diesem Fall kann es z.B. künstlich auf ein XOR abgebildet werden. Schleifen können ebenso z.B. mit dem *while*-Element realisiert werden, was im Rahmen dieser Arbeit jedoch nicht betrachtet wird.

Wie bei den Web Service-Operationen kann auch der BPEL-Prozess selber synchron oder asynchron sein. Weil der betrachtete Incident-Management-Prozess der Charakterisierung nach eher ein asynchrones Verhalten beschreibt, nachdem die Bearbeitung eines Incidents in der Regel eine gewisse Zeitspanne beansprucht, wird auch im Beispiel die Interaktion mit dem BPEL-Prozess asynchron beschrieben.

Beispiel

Abbildung 6.30 nimmt das um die Definition der Kommunikationsverbindungen ergänzte BPEL-Dokument aus Abb. 6.25 und fügt für das Beispiel Interaktionsaufrufe hinzu. Der dargestellte BPEL-Prozess „IncidentProzess“ ist asynchron definiert. Die Kommunikationsverbindung „client“, welche durch das entsprechende *partnerLink*-Element definiert wird, repräsentiert die Verbindung des BPEL-Prozesses zu seiner WSDL-Spezifikation, das auszugsweise in Abbildung 6.18 dargestellt ist.

Innerhalb des *sequence*-Elements beginnt der BPEL-Prozess seine Tätigkeit, falls er über die Kommunikationsverbindung „client“ eine Anfrage über die Operation „opCall“ des Interfaces „IncidentAnnahme“ erhält. Dies wird durch das *receive*-Element mit dem Bezeichner „ClientCall“ spezifiziert. Die in der Daten-

Abschnitt 6.4.2.2

Abschnitt 6.4.2.1

Interaktions-
beschreibung
(Anfang)

initialer
Request vom
Client

Daten-
transformation

synchrone
Interaktion mit
„IncidentSatus
Service“

Ergebnis
an Client
übermitteln

Interaktions-
beschreibung
(Ende)

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="IncidentProzess"
  targetNamespace="http://example.org/bpel/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bp="http://example.org/bpel/"
  xmlns:ws="http://example.org/ws/" >

  <partnerLinks>
    <partnerLink name="client" partnerLinkType="bp:incidentService"
      myRole="iService" partnerRole="iServiceCustomer"/>
    <partnerLink name="incidentStatusService"
      partnerLinkType="ws:iStatus" partnerRole="iStatusService"/>
  </partnerLinks>

  <variables>
    <variable name="IncidentRequest" element="bp:IncidentIDBearbeiter"/>
    <variable name="IncidentStatusRequest" element="ws:IncidentID"/>
    <variable name="IncidentStatusResponse" element="ws:IncidentStatus"/>
  </variables>

  <sequence>

   <receive name=“ClientCall“ partnerLink="client"
    portType="bp:IncidentAnnahme" operation="opCall"
    variable="IncidentRequest" createInstance="yes"/>

   <assign>
    <copy>
     <from variable=“IncidentRequest“ part=“IncidentID“/>
     <to variable=“IncidentStatusRequest“ />
    </copy>
   </assign>

   <invoke partnerLink="incidentStatusService"
    portType="ws:KapselAInterface" operation="opKA"
    inputvariable="IncidentStatusRequest"
    outputvariable="IncidentStatusResponse"/>

   <invoke name=“ClientCallback“ partnerLink="client"
    portType="bp:ClientCallback" operation="opCallback"
    inputvariable="IncidentStatusResponse"/>

  </sequence>

</process>
```

Abbildung 6.30.: Beispielcodierung der Interaktion

variable „IncidentRequest“ gespeicherte Nachricht des Service Requestors wird, bevor die Anfrage an den Web Service „IncidentStatusService“ weitergegeben werden kann, durch das abgebildete *assign*-Element auf das geforderte Datenformat gebracht. Im Rahmen dieser Datentransformation wird der „IncidentID“-Teil der Nachricht in das *variable*-Element mit dem Bezeichner „IncidentStatusRequest“ umkopiert. Anschließend wird synchron durch ein *invoke*-Element über den PartnerLink „incidentStatusService“ mit dem Web Service „IncidentStatusService“ interagiert. Dessen WSDL-Spezifikation stammt aus Abbildung 6.14 und wurde wie rechts in Abbildung 6.24 gezeigt, um ein *partnerLinkType*-Element erweitert.

Die von diesem Service zurückgelieferte Nachricht, welche in der Datenvariable „IncidentStatusResponse“ gespeichert wird, entspricht bereits dem Datenformat der Operation „opCallback“ des Interfaces „Client-Callback“. Aus diesem Grund kann die Nachricht ohne etwaige Transformation durch ein *invoke*-Element über die Kommunikationsverbindung „client“ an den anfragenden Service zurückgegeben werden.

Dieser Abschnitt hat sich mit der Beschreibung der Interaktionsbeziehung in BPEL befasst. Dazu wurden zuerst die allgemeinen Grammatiken einiger BPEL-Elemente vorgestellt, bevor die Abbildung der Orchestrierung auf diese Elemente betrachtet wurde. Im Fokus dieser Abbildung stand dabei vorrangig die Modellierung des zeitlichen Ablaufs der Interaktionen mit den Web Services, die mit Vervollständigung des BPEL-Prozess-Beispiels abgeschlossen wurde.

Im Rahmen der BPEL-Spezifikation als Teil der BPEL-Orchestrierung wurden dazu drei verschiedene Beziehungen (Daten-, Kommunikations- und Interaktionsbeziehung) identifiziert und betrachtet. Neben diesen wurde zu Beginn des Abschnitts über BPEL-Orchestrierung auf die WSDL-Spezifikation für den BPEL-Prozess eingegangen. Im Kontext dieser Betrachtung wurde die WSDL-Spezifikation aus Abschnitt 6.3.1 um die Definition des *partnerLinkType*-Elements ergänzt, welches zwar kein Teil von WSDL ist, dennoch innerhalb des WSDL-Files spezifiziert wird. Dieses Element wurde dann nochmals in Zusammenhang mit der Beschreibung der Kommunikationsbeziehung in BPEL betrachtet.

6.5. Zusammenfassung

Dieses Kapitel hat sich mit der Implementierung der in Kapitel 5 identifizierten funktionalen Kapseln, Schnittstellen sowie deren Orchestriertung befasst. Da im Rahmen dieser Arbeit die Dienste der Service Oriented Architecture durch Web Services realisiert werden sollen, bis dahin aber nur von funktionalen Kapseln gesprochen wurde, ist in Abschnitt 6.1 die Aufteilung der funktionalen Kapseln auf Web Services beschrieben worden.

Anschließend wurden entsprechende Schnittstellenbeschreibungen in WSDL abgeleitet. Zur besseren Veranschaulichung der WSDL-Spezifikation wurde ein WSDL-File für einen Web Service sukzessive aufgebaut und erklärt. Die tatsächliche Implementierung der Web Services wurde in diesem Kapitel vernachlässigt, da auch im Kontext einer SOA die Wahl der verwendeten Programmiersprache keine Bedeutung hat[13].

Abschnitt 6.4 hat sich schließlich mit der Beschreibung der Orchestrierung in BPEL befasst. Dazu wurde die WSDL-Beschreibung zuerst um das *partnerLinkType*-Element ergänzt. Dieses wurde nach der Spezifikation der Datenbeziehung für die Definition einer Kommunikationsverbindung zwischen dem Web Service und dem BPEL-Prozess benötigt. Im Zuge der Beschreibung der Interaktionsbeziehung als dritten Bereich der BPEL-Spezifikation wurden dann die Interaktionen zwischen den Web Services im Rahmen der Orchestrierung auf BPEL abgebildet.

Nachfolgend sind noch einmal wichtige durchzuführende Schritte aufgelistet:

- Verteilung der funktionalen Kapseln auf die Web Services.
- Definition einer Service-Hierarchie und der grundsätzlichen Kommunikation. (vgl. Abb. 6.2)
- Identifikation serviceübergreifender Datenstrukturen, welche einheitlich modelliert werden sollten, um überflüssige Datentransformationen zu vermeiden.
- WSDL-Schnittstellenbeschreibung für die Web Services formulieren (siehe Abschnitt 6.3.1).
- Implementierung der Web Services, falls nicht vorhanden.
- Ergänzung der WSDL-Spezifikation um *partnerLinkType*-Elemente (siehe Abschnitt 6.4.1), zur späteren Definition einer Kommunikationsverbindung mit einem BPEL-Prozess.
- Orchestrierung mittels BPEL beschreiben (siehe Abschnitt 6.4.2).

Nachdem die konzeptionelle Umsetzung des in Kapitel 5 entwickelten SOA-basierten Konzepts auf die gewählten Spezifikationen abgeschlossen ist, wird im folgenden Kapitel die prototypische Implementierung des Incident-Management-Prozesses beschrieben.

[13] siehe „Programmiersprachenunabhängigkeit“ als Eigenschaft einer SOA

Kapitel 7.

Prototypische Implementierung

In diesem Kapitel steht die prototypische SOA-Modellierung des Incident-Management-Prozesses im Vordergrund. Nachdem die Phase der „Prozessanalyse“ für den Incident-Management-Prozess bereits durch das Kapitel 4 vorweggenommen wurde, widmet sich dieses Kapitel der prototypischen Realisierung der zwei weiteren Phasen „SOA-Idee umsetzen“ und „SOA-Implementierung“ in der in Abbildung 7.1 dargestellten Vorgehensweise.

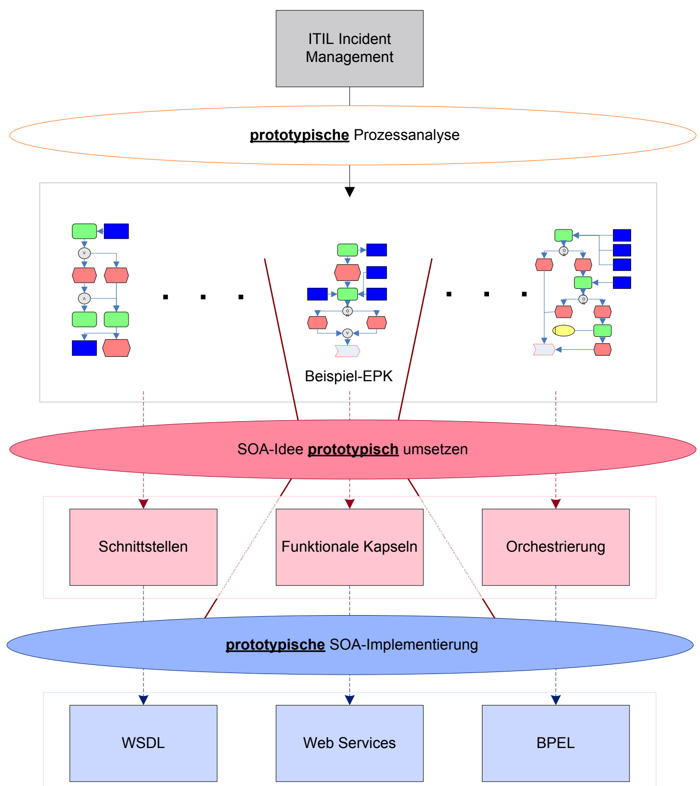


Abbildung 7.1.: Vorgehensweise bei der prototypischen Implementierung

Ausgehend von der verfeinerten Prozessstruktur aus Kapitel 4 werden zunächst gemäß Kapitel 5 die Schnittstellen, funktionalen Kapseln und die Orchestrierung für den Incident-Management-Prozess bestimmt. Dazu werden zuerst die identifizierten Prozess-Schnittstellen entsprechend den in ITIL definierten Verantwortlichkeiten den beteiligen ITIL-Prozessen zugeordnet. Anschließend wird die funktionale Kapselung für den mittels EPK modellierten Prozess abgeleitet und um Aspekte der Orchestrierung wie z.B. zusätzliche Interaktionen zur Datenbeschaffung ergänzt.

Im zweiten Abschnitt, welcher sich an der in Kapitel 6 beschriebenen Vorgehensweise orientiert, werden dann entsprechend der für diese Arbeit gewählten SOA-Implementierung die funktionalen Kapseln auf Web Services abgebildet und deren Schnittstellen mit WSDL beschrieben. Für die Spezifikation der Orchestrierung wird BPEL verwendet. Da der betrachtete Incident-Management-Prozess selbst auf der gewählten Abstraktionsebene bereits sehr komplex ist (siehe Abb. 4.20), wird die Vorgehensweise, welche in Kapitel 5 und 6 beschrieben wurde, in diesem Kapitel nur anhand eines Teilstücks prototypisch umgesetzt.

7.1. Schnittstellen, funktionale Kapselung und Orchestrierung

Dieser Abschnitt befasst sich mit der Phase „SOA-Idee umsetzen“ (siehe Abb. 7.1), welche in Kapitel 5 theoretisch betrachtet wurde. Entsprechend dem dort beschriebenen Vorgehen werden gemäß der in Abschnitt 5.1 definierten „Blackbox“-Betrachtung zuerst die Schnittstellen des Incident Managements zu anderen Prozessen identifiziert. Dann werden ausgehend von den in der Prozessanalyse erstellten EPKs die funktionalen Kapseln abgeleitet und in Bezug auf die Orchestrierung ergänzt.

7.1.1. Schnittstellen

Ausgangspunkt für die „Blackbox“-Identifizierung der Schnittstellen sind die in der Analyse des Incident-Management-Prozesses ermittelten Inputs und Outputs (siehe Kap. 4.3), welche in Tabelle 4.7 nochmals zusammengefasst aufgeführt sind. Entsprechend den Verantwortlichkeiten für die Daten, die dem Incident-Management-Prozess als Input zur Verfügung gestellt werden, und den Adressaten der Outputdaten lassen sich nun Schnittstellen ermitteln, welche in Tabelle 7.1 dargestellt sind. In der Tabelle sind mit „I“ Daten gekennzeichnet, die das Incident Management aus dem angegeben Prozess als Input bezieht. Daten die mit „O“ markiert sind, werden als Output an den jeweiligen Prozess geliefert.

Neben den in dieser Tabelle identifizierten Schnittstellen sind noch weitere zu anderen ITIL-Managementbereichen wie z.B. dem Security Management möglich. Sie werden ebenso wie diejenigen zum Financial Management for IT Services nicht betrachtet, obwohl im Kapitel 3 darauf hingewiesen wurde, dass Schnittstellen zu diesem Service-Delivery-Prozess bestehen.

Nachdem ausgehend von den in der Analyse des Incident-Management-Prozesses ermittelten Inputs und Outputs sowie den von ITIL vorgegebenen Verantwortlichkeiten der einzelnen Prozesse Schnittstellen des Incident-Management-Prozesses zu anderen ITIL-Prozessen identifiziert wurden, sollen im Rahmen der funktionalen Kapselung dieses Beispiels zwei Annahmen getroffen werden:

Jeder in der Tabelle 7.1 identifizierte Prozess, der Daten dem Incident-Management-Prozess bereitstellt oder Daten von diesem erhält,

1. stellt dafür eine funktionale Kapsel und damit dem Incident Management eine aufrufbare Funktionalität eines Web Service zur Verfügung.
2. liefert die Daten im vom Incident Management benötigtem Format oder erhält sie von diesem im geforderten Format, so dass diesbezüglich zwecks Vereinfachung keine Datentransformationen notwendig sind.

Diese zwei Annahmen zur Interprozessinteraktion werden in die BPEL-Orchestrierung in Abschnitt 7.2.3 einfließen. Bevor jedoch die Orchestrierung im Rahmen der Konzeptentwicklung am Beispiel betrachtet wird, befasst sich der nächste Abschnitt mit der Identifikation der funktionalen Kapseln eines Teilstücks des Incident-Management-Prozesses.

		ITIL-Prozesse									
		Service Support					Service Delivery				
		Incident Management	Problem Management	Configuration Management	Change Management	Release Management	Service Level Management	Capacity Management	Availability Management	Continuity Management	The Business/IT-Organisation
Daten	Capacity/Availability Events/Alerts								O	O	
	Change-Plan (FSC)				I						
	Geschäftsentwicklung und -ziele										I
	Incident Records	I O	O								
	Informationen über Konfigurationselemente			I							
	Informationsbedarf bzgl. der CMDB			O							
	Klassifizierungs- und Kategorisierungskatalog						I				
	Notfallpläne									I	
	Probleme/Bekannte Fehler		I								
	Release Policy/Plans/Notes					I					
	Reports	alle Interessengruppen									
	Requests for Change (RFCs)				O						
	Rollout-Informationen				I						
	Schwellenwerte/Checkpoints						I				
	Servicekatalog										I
	User Service Request	I O					I O				
	Störungsinformationen	I O	I O								
	Supportvereinbarungen (SLAs OLAs, UCs)						I				
	Workarounds/Lösungen	I O	I O								

Tabelle 7.1.: Identifizierte Schnittstellen (I - Input, O - Output)

7.1.2. Funktionale Kapselung

In diesem Abschnitt wird für den Incident-Management-Prozess die funktionale Kapselung abgeleitet. Die Grundlage dafür bilden die mittels EPK modellierten Aktivitäten der Prozessanalyse aus Kapitel 4, welche in Abbildung 4.20 zusammenfassend dargestellt sind. Da jedoch aufgrund der Komplexität des Incident-Management-Prozesses im Rahmen dieser Arbeit nicht der gesamte Prozess prototypisch implementiert werden kann, wird nur das in Abbildung 7.2 dargestellte Teilstück betrachtet. Dieses Teilstück wurde dem zweiten Subprozess „Klassifizieren und erste Unterstützung“ des ITIL Incident Managements entnommen, der in Abschnitt 4.3.2 beschrieben wurde.

Für die beispielhafte Implementierung in diesem Kapitel wird noch eine weitere Einschränkung/Vereinfachung bzgl. des in Abbildung 7.2 dargestellten Teilstücks des Incident-Management-Prozesses vorgenommen. Die durch die Prozesswegweiser „Standard-Incident bearbeiten“, „Zeitfenster erweitern“, „Analyse und Diagnose“ und „Beheben und Wiederherstellen“ repräsentierten EPKs werden im Rahmen der prototypischen Konzeptumsetzung nicht betrachtet. Gemäß dem in Kapitel 5 beschriebenen Vorgehen werden die Funktionalitäten wie folgt gekapselt:

Abbildung 7.2.: Auszug aus dem Incident-Management-Prozess (vgl. Abb. 4.8 und 4.11)

Identifikation identischer Muster:

- **Funktion „Incident (neu) kategorisieren"**: Die zwei in Abbildung 7.3 dargestellten Varianten unterscheiden sich nur in Art und Anzahl der Ereignisse, wobei jedes Mal eine Meldungskategorie geliefert wird. Aus diesem Grund kann die Funktion innerhalb einer funktionalen Kapsel zusammengefasst werden.
- **Funktion „Priorität ermitteln"**: In dem betrachteten Teilstück kommt dieser durch einen Prozesswegweiser dargestellte Teilabschnitt zwar nicht mehrmals vor, dennoch stellt er innerhalb des Incident-Management-Prozesses, wie in Abbildung 5.5 gezeigt, ein wiederkehrendes Muster dar und wird deshalb funktional gekapselt. Die Prozesskette, welche sich hinter diesem Prozesswegweiser verbirgt, ist in Abbildung 7.4 nochmals dargestellt.

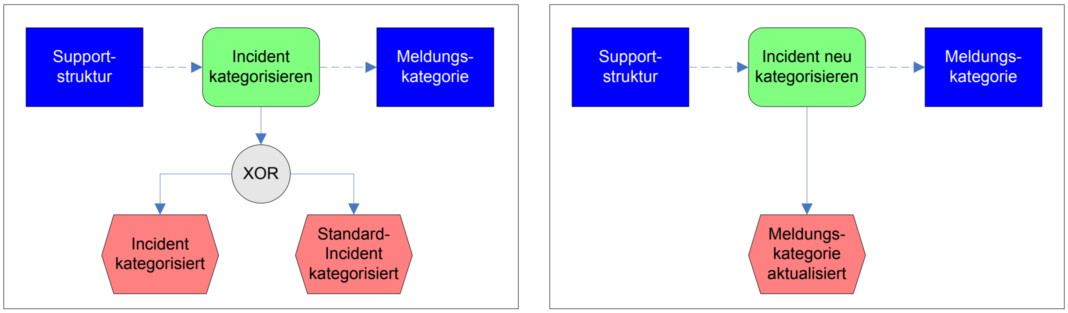


Abbildung 7.3.: Muster funktional zusammenfassen

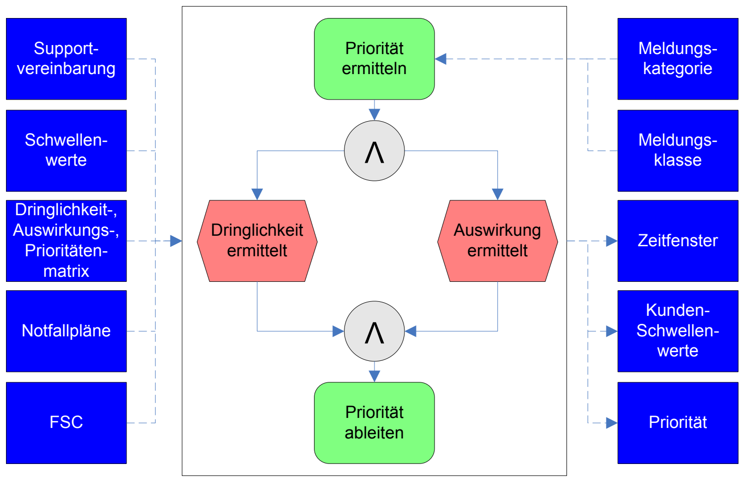


Abbildung 7.4.: Priorität ermitteln (siehe auch Abb. 4.9)

Verknüpfungen analysieren:
Grundsätzlich empfiehlt sich in dieser Situation die Kapselung pro Teilstrang, wie sie rechts in Abbildung 5.6 dargestellt ist. Im Beispielstrang gemäß Abbildung 7.2 beinhaltet beinahe jeder Teilabschnitt zwischen zwei Verknüpfungen nur eine Funktion, weswegen jede dieser Funktionen eine funktionale Kapsel repräsentiert. Der einzige Teilabschnitt, welcher zwei Funktionen umfasst, wird in Hinblick auf die Orchestrierung nicht zu einer einzigen funktionalen Kapsel zusammengefasst, wie links in Abbildung 7.5 dargestellt ist; da die Funktion „Lösung/Workaround ermitteln" bereits einer funktionalen Kapsel zugeordnet ist, werden beide Funktionen separat gekapselt. Dies ist rechts in Abbildung 7.5 zu sehen.

Trennung gemeinsamer Teilabschnitte:
Im Gegensatz zu den beispielhaften Erläuterungen in Abschnitt 5.2.2.3 sollen die in Abbildung 7.2 links zu sehenden Teilstränge nicht getrennt werden. Die Information darüber, ob ein Incident oder ein Standard-Incident bearbeitet werden soll, wird später im BPEL-Prozess innerhalb einer Datenvariable zwischengespeichert. Deshalb muss diese Information nicht durch die Funktionen „Incident kategorisieren" und „Priorität ermitteln" durchgeschleift werden, um anschließend eine Entscheidung bzgl. der weiteren Bearbeitung treffen zu können.

Vereinigung durch Und-Verknüpfung:
Die einzigen zwei Und-Verknüpfungen im betrachteten Teilstück (siehe Abb. 7.6) umschließen jeweils ausschließlich Ereignisse und spielen aus diesem Grund für die funktionale Kapselung keine Rolle.

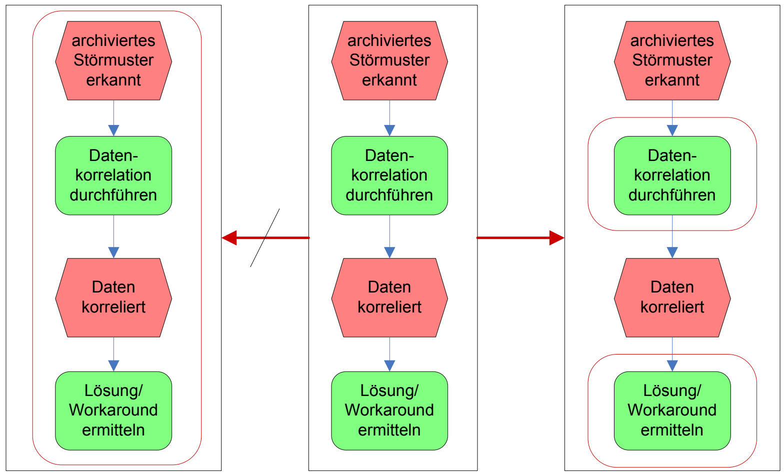


Abbildung 7.5.: Verknüpfungsanalyse

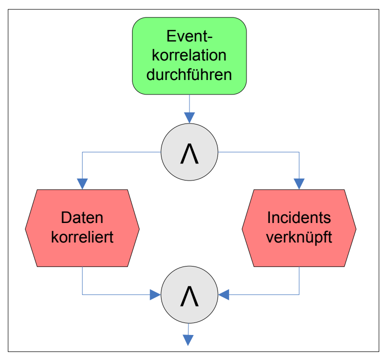


Abbildung 7.6.: Analyse von Und-Verknüpfungen

Input- und Outputbehandlung:
Diese Aspekte der funktionalen Kapselung spielen in dem betrachteten Beispiel keine Rolle, da die Kapselung so gestaltet wurde, dass keine funktionale Kapsel eine andere Kapsel enthält.

Abbildung 7.7 zeigt die funktionale Kapselung des Beispiels, welche in diesem Abschnitt entwickelt wurde. Bei der Darstellung der funktionalen Kapseln in Abbildung 7.7 wurde dabei zur besseren Übersicht darauf verzichtet, die Ereignisse mit in die Kapseln einzufassen. Dies stellt jedoch keine Abweichung von der in Kapitel 5 beschriebenen funktionalen Kapselung dar, da entsprechend der Definition von Seite 56 nur die Funktionen zu logischen Einheiten zusammengefasst werden. Die gestrichelt blau eingefassten Prozesswegweiser werden bezogen auf das Beispiel in diesem Kapitel nicht betrachtet.

7.1.3. Orchestrierung

Die im vorherigen Abschnitt identifizierten funktionalen Kapseln werden im Rahmen der Orchestrierung nochmals betrachtet. Dabei wird insbesondere auf die Datenübergabe sowie eine zusätzliche Interaktion mit anderen Prozessen der ITIL zwecks Datenbeschaffung eingegangen. Jede funktionale Kapsel in dem betrachteten Beispiel enthält, wie Abbildung 7.7 zu entnehmen ist, nur eine Funktion. Aus diesem Grund werden die funktionalen Kapseln zukünftig entweder über den Bezeichner der darin enthaltenen Funktion oder durch den numerischen Bezeichner aus Abbildung 7.7 referenziert.

Abbildung 7.7.: Funktionale Kapselung für das Beispiel

Datenübergabe, ggf. mit Transformation:
Betrachtet man die funktionalen Kapseln für das Beispiel in Abbildung 7.7, so stellt man fest, dass nicht alle notwendigen Daten dargestellt sind. Neben dem *Incident Record (IR)*, welcher alle Daten der Störungsmeldung enthält, sind des Weiteren auch keine Datenübergaben zwischen den funktionalen Kapseln modelliert. Ebenso fehlt teilweise die Darstellung von Daten, welche jedoch durch die von einer Funktion ausgelösten Ereignisse angedeutet werden. Aus diesem Grund muss für jede funktionale Kapsel die Datenmodellierung und -übergabe geprüft werden.

Im Rahmen dieser Überprüfung ist festzustellen, dass, wenn man die bisher nicht abgebildete globale Datenstrukur des Incident Records einführt, alle Funktionen einen Teil der darin enthaltenen Informationen benötigen, liefern oder modifizieren. Zur Veranschaulichung betrachte man die Kapsel „Störungsmeldung klassifizieren“. Diese benötigt neben dem Klassifizierungkatalog des Incident Managements die Angaben der Störungsmeldung, um diese zu klassifizieren. Wird die Meldungsklasse innerhalb des Incident Records gespeichert, so kann darauf verzichtet werden, diese Meldungsklasse der Kapsel „Incident katagorisieren“ zu übergeben, wenn stattdessen der Incident Record übergeben wird. In diesem Fall muss u.a. auch die zur Bestimmung der Meldungskategorie notwendige Störungsbeschreibung nicht explizit übergeben werden, da auch sie Teil des Incident Records ist. Nachfolgend wird deshalb angenommen, dass jeder funktionalen Kapsel der Incident Record als Ganzes übergeben wird.

Zusätzliche Prozessinteraktion zur Datenbeschaffung:

- **Kapsel „Störmuster prüfen“**: Der von dieser Kapsel benötigte Input aus der Problem- und Known-Error-Datenbank obliegt der Verantwortlichkeit des Problem Managements. Aus diesem Grund muss eine zusätzliche Interaktion zur Datenbeschaffung im Rahmen der Orchestrierung modelliert werden, welche in Abbildung 7.8 dargestellt ist.

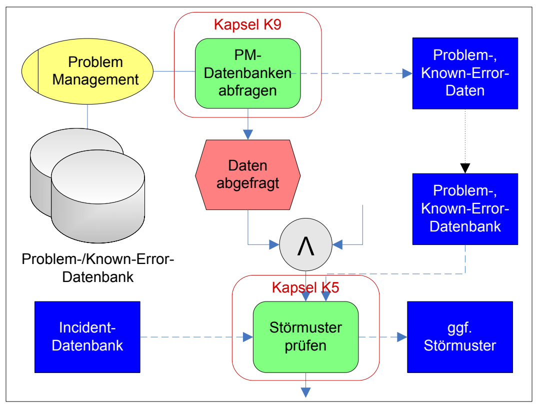


Abbildung 7.8.: Interaktion mit dem Problem Management zur Datenbeschaffung

- **Kapseln „Störungsmeldung klassifizieren“, „Incident kategorisieren“, „Störmuster prüfen“**: Diese drei Kapseln benötigen Input, der zwar in den Verantwortlichkeitsbereich des Incident-Management-Prozesses fällt, dennoch später über so genannte Basisdienste der SOA erst noch erfragt werden soll. Aus diesem Grund wird für jede dieser Kapseln eine Interaktion mit dem Incident Management zur Datenbeschaffung modelliert (siehe Abb. 7.9).

- **Kapsel „Priorität ermitteln“**: Aufgrund der Vielzahl der Daten, die dieser Kapsel zur Verfügung gestellt werden müssen, soll an dieser Stelle diesbezüglich eine Abstrahierung vorgenommen werden. Im Folgenden wird davon ausgegangen, dass dieser funktionalen Kapsel nur der Incident Record übergeben wird und sie für diesen eine Priorität liefert. Außerdem wird angenommen, dass der Web Service, welcher diese Kapsel später realisiert, die weiteren in Abbildung 7.4 dargestellten Daten selbst erfragt und den Incident Record entsprechend aktualisiert. Dadurch sind für diese funktionale Kapsel keine weiteren Prozessinteraktionen notwendig.

Abbildung 7.10 zeigt die im Rahmen der Orchestrierung hinzugekommenen Interaktionen zur Datenbeschaffung für das vorliegende Beispiel. Auf die Darstellung des Incident Records als Input jeder funktionalen Kapsel wurde zur Erhaltung der Übersichtlichkeit verzichtet.

Damit ist die funktionale Kapselung, welche entsprechend dem in Kapitel 5 beschriebenen Vorgehen für das Beispiel angewandt wurde, abgeschlossen. Dazu wurden zuerst die Schnittstellen für den Incident-Management-Prozess zu anderen ITIL-Prozessen abgeleitet. Anschließend wurde in Abschnitt 7.1.2 die funktionale Kapselung für den ausgewählten Beispiel-Prozessstrang aus dem Incident Management entwickelt, welche schließlich um Aspekte der Orchestrierung erweitert wurde. Dabei wurden die Daten, deren Übergabe sowie zusätzliche Interaktionen zur Datenbeschaffung betrachtet. Die so ermittelten funktionalen Kapseln werden im Rahmen des nächsten Abschnitts auf die konkrete Implementierung der SOA mittels Web Services, WSDL und BPEL abgebildet.

Abbildung 7.9.: Interaktion mit dem Incident Management zur Datenbeschaffung

7.2. Web Services, WSDL und BPEL

Dieser Abschnitt orientiert sich an dem in Kapitel 6 beschriebenen Vorgehen zur Implementierung der zuvor entwickelten SOA mittels Web Services, WSDL und BPEL. Dazu werden zuerst die identifizierten funktionalen Kapseln den Web Services der SOA zugeordnet. Anschließend werden in Abschnitt 7.2.2 für ausgewählte Web Services die Schnittstellen mit WSDL beschrieben und um eine *partnerLinkType*-Spezifikation für die spätere Orchestrierung ergänzt. Schließlich wird dann für das Beispiel die Orchestrierung mittels BPEL spezifiziert.

7.2.1. Web Service-Zuordnung

Ausgehend von den im vorherigen Abschnitt identifizierten funktionalen Kapseln müssen diese auf die Web Services verteilt werden, bevor sie mittels WSDL beschrieben werden können. Neben dieser Zuordnung wird gleichzeitig auch noch eine Service-Hierarchie innerhalb der entstehenden SOA für das Beispiel definiert.

Wie bereits in Abschnitt 6.1 angedeutet wurde, spielen bei der Verteilung der funktionalen Kapseln auf die Web Services verschiedene Aspekte eine Rolle, von denen im Rahmen der prototypischen Implementierung einige nachfolgend aufgelistet sind:

- Stellt die funktionale Kapsel eine Basisaktivität dar?
- Welcher Prozess ist für die Realisierung der Funktionalität, z.B. die Bereitstellung der Daten verantwortlich?

Abbildung 7.10.: Orchestrierung für das Beispiel

- Greift die Funktionalität auf andere funktionale Kapseln zu?
- Gehören einzelne funktionale Kapseln vom Kontext her zusammen? Bauen sie bspw. aufeinander auf?

Entsprechend der Beantwortung der eben aufgeführten Fragestellungen wurden die funktionalen Kapseln aus Abbildung 7.10 auf die Web Services verteilt. Die Verteilung der Kapseln[1] auf die zu implementierenden Web Services ist in Tabelle 7.2 zusammenfassend dargestellt. In der Tabelle sind die Web Services mit kurzen Bezeichnern versehen, die später auch in den Abbildungen und den WSDL-Spezifikationen verwendet werden. Im Zusammenhang mit dieser Verteilung wurden die funktionalen Kapseln, welche im Rahmen der Orchestrierung in Abschnitt 7.1.3 zur Datenbeschaffung modelliert wurden, als Basisaktivitäten eingestuft und dementsprechend einem Basisdienst zugeordnet. Dabei wurden die funktionalen Kapseln, die eine Interaktion mit dem Problem Management darstellen, von denjenigen getrennt, welche innerhalb des Incident Managements aufgeführt werden.

Web Service	funktionale Kapsel	Hierarchie-Stufe
„BPEL-Service“ (WSH)	*(gesamtes Beispiel)*	Prozessdienst
„Klassifikation“ (WSA)	„Störungsmeldung klassifizieren“ (K1)	Intermediarydienst
	„Incident kategorisieren“ (K3)	
	„Priorität ermitteln“ (K4)	
„Korrelation“ (WSB)	„Störmuster prüfen“(K5)	Intermediarydienst
	„Eventkorrelation durchführen“ (K6)	
	„Datenkorrelation durchführen“ (K7)	
„Lösung/Workaround“ (WSC)	„Lösung/Workaround ermitteln“ (K8)	Intermediarydienst
„User Service Request“ (WSD)	„User Service Request bearbeiten“ (K2)	Intermediarydienst
„IM-Strukturen“ (WSE)	„Klassifizierungskatalog anfordern“ (K10)	Basisdienst
	„Supportstruktur anfordern“ (K11)	
„IM-Daten“ (WSF)	„IM-Datenbanken abfragen“ (K12)	Basisdienst
	„IR-Daten abfragen“	
„PM-Daten“ (WSG)	„PM-Datenbanken abfragen“ (K9)	Basisdienst
	„Problem-Daten abfragen“	
	„Known-Error-Daten abfragen“	

Tabelle 7.2.: Web Services mit den enthaltenden funktionalen Kapseln

Der zusätzlich definierte Web Service „BPEL-Service“ repräsentiert die Gesamtheit dieses Beispiels und wird auf der Ebene der Prozessdienste eingeordnet. Dieser soll die Interaktion der Web Services steuern und wird mittels BPEL spezifiziert werden.

Funktionale Kapseln wie z.B. „Störungsmeldung klassifizieren“ verlangen zur Realisierung ihrer Funktionalität eine Interaktion mit einer anderen funktionalen Kapsel zur Datenbeschaffung. Da sie entsprechend dem Kommunikationsmodell für dieses Beispiel selbstständig mit diesen kommunizieren, werden diese Web Services auf der Stufe der Intermediarydiensten zugeordnet. Die drei funktionalen Kapseln „Störungsmeldung klassifizieren“, „Incident kategorisieren“ und „Priorität ermitteln“ werden dabei demselben Web Service zugeordnet, da sie alle Schlüsselgrößen für die weitere Incidentbearbeitung ermitteln. Auch die funktionalen Kapseln „Störmuster prüfen“, „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“ werden einem Web Service zugewiesen, da die letzten beiden Kapseln auf dem aufbauen, was die Kapsel „Störmuster prüfen“ liefert. Die funktionalen Kapsel „Lösung/Workaround ermitteln“ und „User Service Request bearbeiten“ werden jeweils einem eigenen Web Service auf der Stufe der Intermediarydienste zugeordnet, da sie u.a. umfangreichere Aufgaben umfassen.

Neben den zusätzlichen Prozessinteraktionen aus Abschnitt 7.1.3 sollen an dieser Stelle noch weitere Funktionalitäten definiert werden, welche dem Abfragen einzelner Daten aus den Problem-, Known-Error- und Incident-Datenbanken dienen. So ermöglicht etwa die Funktionalität „IR-Daten abfragen“ das Auslesen

[1] Nummerierung in Klammern gemäß Abbildung 7.10

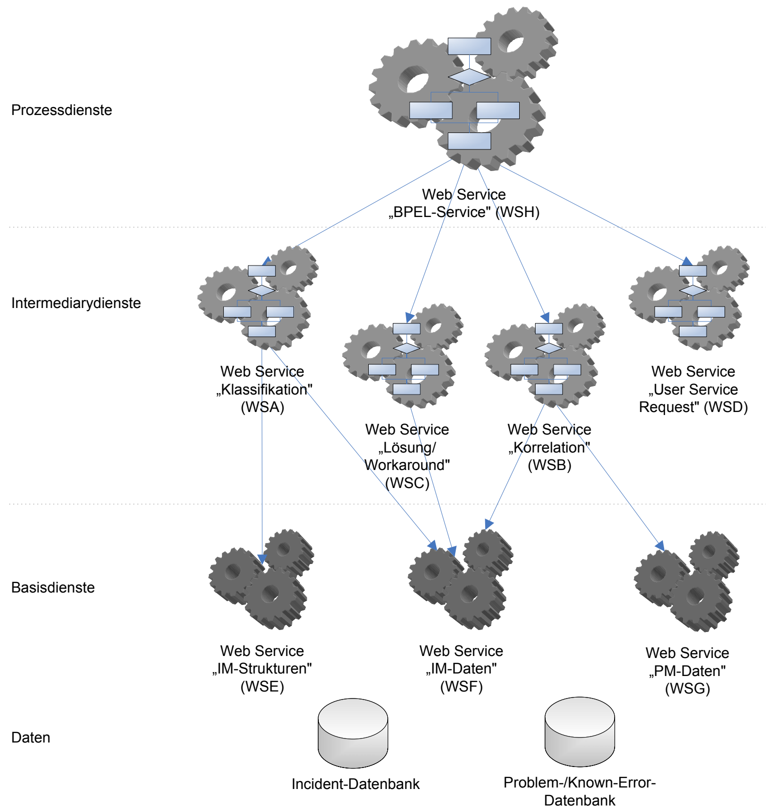


Abbildung 7.11.: Service-Hierarchie für die Web Services aus Tabelle 7.2

der Daten, die im Incident Record gespeichert sind, und wird dem Web Service „IM-Daten" zugeordnet. Dem Web Service „PM-Daten" werden in diesem Zusammenhang die Funktionalitäten „Problem-Daten abfragen" und „Known-Error-Daten abfragen" zugeordnet.

Für die in Tabelle 7.2 identifizierten Web Services zeigt Abbildung 7.11 eine Service-Hierarchie in Anlehnung an Abbildung 6.2. Im Rahmen der Spezifikation der Kommunikation innerhalb der Service-Hierarchie wird für das Beispiel angenommen, dass die Web Services auf der Hierarchie-Stufe der Intermediarydienste direkt mit den Basisdiensten kommunizieren, um sich die notwendigen Grunddaten zu beschaffen. Für den Web Service „BPEL-Service" auf der Stufe der Prozessdienste bedeutet dies, dass dieser Dienst im gewählten Beispiel nur noch die Interaktion mit den Intermediarydiensten steuern muss. Außerdem kann dadurch vermieden werden, dass große Datenmengen, die z.B. zur Störmusterprüfung notwendig sind, erst zum Web Service „BPEL-Service" transportiert werden müssen und von dort innerhalb der Hierarchie wieder nach unten weitergereicht werden. Im betrachteten Beispiel reduziert sich dadurch die übertragene Datenmenge zwar für nur einen Schritt, dies liegt jedoch an der hier nur drei Ebenen umfassenden Service-Hierarchie.

Abbildung 7.12 zeigt graphisch die Zuordnung der funktionalen Kapseln zu den Web Services der SOA. Umschließt ein Web Service (schwarz umrandet) einen anderen, so bedeutet dies für das Beispiel, dass gemäß der zuvor definierten Kommunikation in der Service-Hierarchie der direkt umgebende Web Service mit dem inneren Service selbstständig interagiert. Dadurch müssen diejenigen Daten, welche vom eingeschlossenen Web Service geliefert werden, nicht in die Schnittstellenbeschreibung für den umgebenden Web Service aufgenommenen werden. Beispielsweise umfasst der Web Service „Klassifikation" mit der funktionalen Kapsel „Störungsmeldung klassifizieren" den Web Service „IM-Strukturen", welcher den Klassifizierungskatalog liefert. Dieser ist laut Tabelle 7.2 ein Basisdienst und wird entsprechend der An-

Abbildung 7.12.: Web Services für das Beispiel

nahme, die bzgl. der Kommunikation in der Service-Hierarchie getroffen wurde, vom Web Service „Klassifikation“ zur Datenbeschaffung aufgerufen. Der Web Service „BPEL-Service“, welcher das gesamte Beispiel umfasst, muss deshalb zum Aufruf der funktionalen Kapsel „Störungsmeldung klassifizieren“ nur mit dem Web Service „Klassifikation“ kommunizieren, ohne zuvor über den Web Service „IM-Strukturen“ den Klassifizierungskatalog zu erfragen.

Neben den in Abbildung 7.11 dargestellten (Basis-)Diensten müssten noch weitere Dienste definiert werden, welche z.B. die von der funktionalen Kapsel „Priorität ermitteln“ benötigten Daten liefern. Auf die Spezifikation dieser Dienste wird im Rahmen dieser Arbeit jedoch verzichtet.

In diesem Abschnitt wurde jede funktionale Kapsel aus Abschnitt 7.1.2 einem Web Service zugeordnet und eine Service-Hierarchie für das Beispiel etabliert. Für die Intermediarydienste und den Prozessdienst wird im nächsten Abschnitt die Schnittstellenbeschreibung mittels WSDL spezifiziert, bevor die BPEL-Orchestrierung der Web Services in Abschnitt 7.2.3 beschrieben wird.

7.2.2. WSDL-Spezifikation

Dieser Abschnitt befasst sich mit der WSDL-Spezifikation der identifizierten Prozess- und Intermediarydienste. Das Vorgehen wird sich dabei an der Beschreibung in Abschnitt 6.3.1 orientieren. Dort wurden im Rahmen der WSDL-Schnittstellenbeschreibung die *types*-, *interface*-, *binding*- und *service*-Elemente besprochen. Neben diesen WSDL-Elementen wurde in Zusammenhang mit der BPEL-Orchestrierung in Abschnitt 6.4.1 das *partnerLinkType*-Element betrachtet, welches zwar nicht Teil der WSDL-Spezifikation ist, aber für die spätere Kommunikation mit dem BPEL-Prozess benötigt wird. Da es im WSDL-File des Web Service definiert werden muss, wird es bereits in diesem Abschnitt und nicht wie in Kapitel 6 erst im Kontext der BPEL-Beschreibung betrachtet.

Die Spezifikationen der *binding*- und *service*-Elemente entsprechend Abschnitt 6.3.1.2 unterscheidet sich nicht wesentlich voneinander. Außerdem wird sie in Hinblick auf die spätere Beschreibung der Orchestrierung mittels BPEL nicht direkt benötigt. Deshalb wird in diesem Abschnitt darauf verzichtet, die *binding*- und *service*-Elemente für jeden Web Service einzeln zu beschreiben. Wie die Elemente definiert werden müssen, wird im ersten Abschnitt für den Web Service „Klassifikation“ beispielhaft gezeigt.

Entsprechend den in den Abschnitten 7.1.3 und 7.2.1 getroffenen Entscheidungen, dass jeder Funktion der gesamte Incident Record übergeben wird, die Kapsel „Priorität ermitteln“ nur die Priorität an den aufrufenden Service zurückliefert und jeder Intermediarydienst weitere benötigte Daten selbstständig über die Basisdienste erfragt, werden die Input- und Outputdaten der betrachteten Web Service abgewandelt.

Innerhalb des Incident-Management-Prozesses spielt der Incident Record mit seinen Bestandteilen eine wichtige Rolle. Da in der Realität in Zusammenhang mit der Bearbeitung eines Incident Records jedoch nicht jede beteiligte Partei alle Informationen lesen darf oder lesen sollte, wird auch im betrachteten Beispiel nicht der gesamte Incident Record den Web Services übergeben, sondern nur dessen ID. Über diese können die Web Services dann die notwendigen und die für sie vorgesehenen Daten über Basisdienste erfragen.

Die im Rahmen des Beispiels notwendigen Bestandteile eines Incident Records wie die Incident-ID, die Meldungsklasse, die Meldungskategorie und die Priorität werden serviceübergreifend und in einem separatem Dokument mit dem Namen „IR.xsd“ wie folgt modelliert. Dabei steht „xsd“ für den Namensraum *xmlns:xsd=„http://www.w3.org/2001/XMLSchema“*.

```
<xsd:element name="IncidentID" type="xsd:long" />
<xsd:element name="Meldungsklasse" type="MKlasse" />
<xsd:simpleType name="MKlasse">
  <xsd:restriction base="xsd:string">
    <xsd:enumeration value="User Service Request" />
    <xsd:enumeration value="Incident" />
    <xsd:enumeration value="Standard-Incident" />
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:element name="Meldungskategorie" type="xsd:string" />
<xsd:element name="Priorität" type="xsd:int" />
```

Die ID eines Incident Records wird vom Basisdatentyp „xsd:long" abgeleitet. Da die Meldungsklasse später im BPEL-Prozess als Entscheidungskriterium verwendet werden soll, werden die drei möglichen Werte fest definiert. Die Meldungskategorie wird vom Basisdatentyp „xsd:string" abgeleitet, und das ohne irgendwelche Einschränkungen, da für das Beispiel keine Differenzierung anhand der Meldungsklasse vorgenommen wird. Dasselbe gilt für die Priorität, welche vom Typ „xsd:int" ist.

Die verwendeten Namensräume sind in Tabelle 7.3 aufgelistet. Die Prozess- und Intermediarydienste haben je einen eigenen Namensraum, der nur die Dateien des betreffenden Web Service enthält. Den Basisdiensten werden keine Namensräume zugewiesen, da sie in diesem Beispiel nicht spezifiziert werden. Der Incident Record, welcher die serviceübergreifenden Datentypen für die beteiligten Web Services enthält, wird eine Ebene oberhalb der Servicespezifikationen in der Datei „IR.xsd" abgelegt.

targetNamespace/xmlns:tns	Inhalt
http://example.org/prototyp/	IR.xsd
http://example.org/prototyp/bpel	Spezifikation für Web Service „BPEL-Service"
http://example.org/prototyp/wsA	Spezifikation für Web Service „Klassifikation"
http://example.org/prototyp/wsB	Spezifikation für Web Service „Korrelation"
http://example.org/prototyp/wsC	Spezifikation für Web Service „Lösung/Workaround"
http://example.org/prototyp/wsD	Spezifikation für Web Service „User Service Request"

Tabelle 7.3.: Struktur der Namensräume der Beispielbeschreibung

Das nachfolgende *description*-Element ist allen WSDL-Spezifikationen der Web Services in den folgenden Abschnitten gemein. Dessen *name*-Attribut, der *targetNamespace* sowie der Namensraum *xmlns:tns* sind dabei entsprechend der Tabelle 7.3 zu vervollständigen.

```
<description name=" "
  targetNamespace=" "
  xmlns:tns=" "
  xmlns="http://www.w3.org/2005/08/wsdl"
  xmlns:wsoap="http://www.w3.org/2005/08/wsdl/soap"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2005/08/wsdl
    http://www.w3.org/2005/08/wsdl/wsdl20.xsd">
  ...
</description>
```

7.2.2.1. Web Service „Klassifikation"

Der Web Service „Klassifikation" umfasst die drei funktionalen Kapseln „Störungsmeldung klassifizieren", „Incident kategorisieren" und „Priorität ermitteln". Diese sind zusammen mit der im Rahmen des Beispiels modifizierten Datenmodellierung in Abbildung 7.13 dargestellt. Ausgehend von dieser Abbildung wird die WSDL-Schnittstellenbeschreibung für diesen Web Service abgeleitet.

Typbeschreibung

Für die Modellierung der Datentypen müssen bekanntlich die Inputs und Outputs der funktionalen Kapseln des Web Services betrachtet werden. Jeder funktionalen Kapsel des Web Services wird die ID des zu betrachtenden Incident Record als Input übergeben. Da diese Datentypen serviceübergreifend definiert wurden, müssen sie nur noch importiert werden. Die ursprünglichen Inputs „Klassifizierungskatalog" und

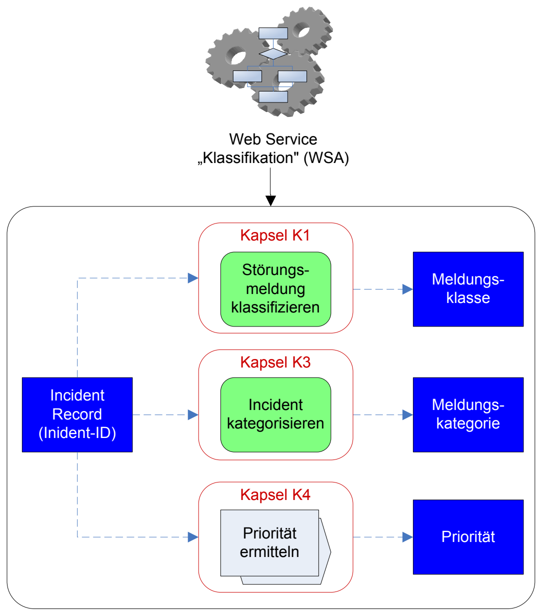


Abbildung 7.13.: Web Service „Klassifikation“

„Supportstruktur“ müssen nicht mehr als solche modelliert werden, da sie intern vom Web Service „Klassifikation“ über einen Basisdienst (Web Service „IM-Daten“) erfragt werden und somit nicht vom anfragenden Service als Parameter der Anfrage übergeben werden müssen. Die Outputs „Meldungsklasse“, „Meldungskategorie“ und „Priorität“ sind Teil der vorherigen IR-Spezifikation und müssen nicht lokal definiert werden. Da alle benötigten Datentypen bereits extern definiert wurden, enthält die Typbeschreibung nur eine *import*-Anweisung für diese Definitionen.

```
<types>
  <xsd:import namespace="http://example.org/prototyp/IR.xsd"
    schemaLocation="IR.xsd" />
</types>
```

Interfacebeschreibung

Der Web Service umfasst drei funktionale Kapseln, welche Funktionalitäten bereitstellen, die jeweils synchron aufgerufen werden können. Für jede dieser Kapseln wird deshalb nur ein *operation*-Element spezifiziert. Als Message-Exchange-Pattern ist für alle Operationen das Pattern „in-out“ zu wählen, da jeder Kapsel Input übergeben werden muss und sie anschließend einen Output zurückliefert. Außerdem wird jeweils ein *input*- und *output*-Element mit Referenz auf das entsprechende Nachrichtenformat definiert. Die so spezifizierten synchronen Operationen werden durch ein *interface*-Element zusammengefasst.

```
<interface name="WSAInterface">
  <!-- Operation für die Kapsel "Störungsmeldung klassifizieren"  -->
  <operation name="klassifizieren"
    pattern="http://www.w3.org/2005/08/wsdl/in-out">
    <input messageLabel="In" element="tns:IncidentID" />
    <output messageLabel="Out" element="tns:Meldungsklasse" />
  </operation>
  <!-- Operation für die Kapsel "Incident kategorisieren"  -->
  <operation name="kategorisieren"
    pattern="http://www.w3.org/2005/08/wsdl/in-out">
    <input messageLabel="In" element="tns:IncidentID" />
    <output messageLabel="Out" element="tns:Meldungskategorie" />
  </operation>
```

```
  <!-- Operation für die Kapsel "Priorität ermitteln"  -->
  <operation name="priorisieren"
    pattern="http://www.w3.org/2005/08/wsdl/in-out">
    <input messageLabel="In" element="tns:IncidentID" />
    <output messageLabel="Out" element="tns:Priorität" />
  </operation>
</interface>
```

Bindingbeschreibung

Durch das *binding*-Element in der WSDL-Schnittstellenbeschreibung wird für das zuvor definierte *interface*-Element das konkrete Kommunikationsprotokoll spezifiziert. Im Rahmen des Beispiels wurde dafür wie in Abschnitt 6.3.1.2 SOAP ausgewählt. Für die Operationen „klassifizieren“, „kategorisieren“ und „priorisieren“ des Interfaces „WSAInterface“ wird dann die bisher nur abstrakte Beschreibung auf eine tatsächliche Implementierung abgebildet. Dazu wird für alle drei Operationen das SOAP Request-Response-Message-Exchange-Pattern verwendet.

```
<binding name="WSASOAPBinding" interface="tns:WSAInterface"
 type="http://www.w3.org/2005/08/wsdl/soap"
 wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP">
  <operation ref="tns:klassifizieren"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response" />
  <operation ref="tns:kategorisieren"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response" />
  <operation ref="tns:priorisieren"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response" />
</binding>
```

Servicebeschreibung

Nachdem das Binding für das Interface spezifiziert ist, wird für diese Kombination ein *endpoint*-Element innerhalb des *service*-Elements definiert. Als Adresse für den hinter der WSDL-Schnittstellenbeschreibung verborgenen Web Service „Klassifikation“ wird dessen Namensraum aus Tabelle 7.3 verwendet.

```
<service name="WSA" interface="tns:WSAInterface">
  <endpoint name="WSAEndpoint" binding="tns:WSASOAPBinding"
   address="http://example.org/prototyp/wsA/" />
</service>
```

PartnerLinkType-Spezifikation

Für die spätere Definition der Kommunikationsverbindung mit dem BPEL-Prozess wird nun noch folgendes *partnerLinkType*-Element spezifiziert. Sein *role*-Element verweist auf das zuvor definierte *interface*-Element und ermöglicht so den Aufruf der darin enthaltenden Operationen.

```
<plnk:partnerLinkType name="WSALinkType">
  <plnk:role name="WSARolle">
    <plnk:portType name="WSAInterface" />
  </plnk:role>
</plnk:partnerLinkType>
```

7.2.2.2. Web Service „Korrelation“

Der zweite Web Service, „Korrelation“, umfasst die drei funktionalen Kapseln „Störmuster prüfen“, „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“ und ist zusammen mit der im Rahmen des Beispiels angepassten Datenmodellierung in Abbildung 7.14 zu sehen. Ausgehend von dieser Abbildung wird die WSDL-Schnittstellenbeschreibung für diesen Web Service abgeleitet.

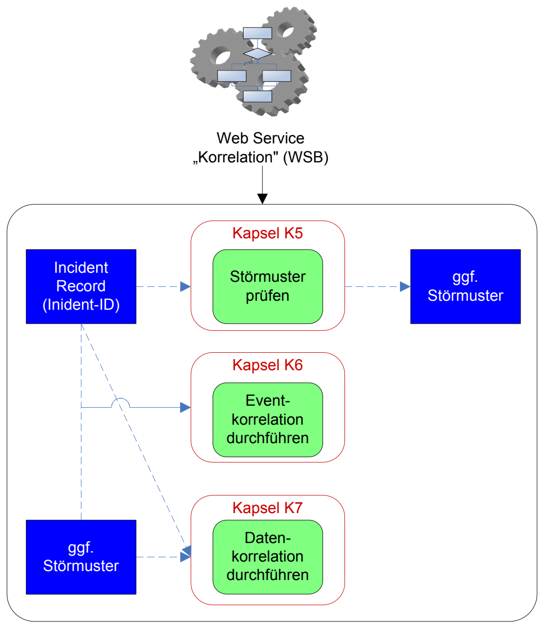


Abbildung 7.14.: Web Service „Korrelation“

Typbeschreibung

Da der Input global in der Datei „IR.xsd“ definiert ist, muss er wiederum importiert werden. Für die Störmuster wird ein *complexType* definiert, welcher zwei optionale Kindelemente „Aktuell“ und „Archiviert“ umfassen kann. Sowohl das Element „Aktuell“ als auch das Element „Archiviert“ können selber wiederum drei optionale Kindelemente („PMuster“, „KEMuster“ und „IMuster“) haben. Jedes dieser Elemente enthält seinerseits mindestens ein Kindelement, das die Identifikationsnummer eines Problems, Known Errors oder Incidents beinhaltet, welches ein identisches Störmuster wie der aktuell betrachtete Incident vorweist. Des Weiteren wird für die funktionalen Kapseln „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“ noch ein aus der ID des Incidents und den ermittelten Störmustern zusammengesetztes Datenelement namens „IDStörmuster“ definiert. Die eben beschriebene Beziehung zwischen den spezifizierten Typen ist in Abbildung 7.15 graphisch dargestellt.

```
<types>
  <xsd:import namespace="http://example.org/Prototyp/IR.xsd"
    schemaLocation="IR.xsd" />
  <xsd:schema targetNamespace="http://example.org/prototyp/wsB/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="IDStörmuster">
      <xsd:complexType>
        <xsd:complexContent>
          <xsd:restriction base="xsd:anyType">
            <xsd:sequence>
              <xsd:element name="Incident" type="IncidentID"
                minOccurs="1" maxOccurs="1" />
              <xsd:element name="Störmuster" type="Muster"
                minOccurs="1" maxOccurs="1" />
            </xsd:sequence>
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>
    </xsd:element>
```

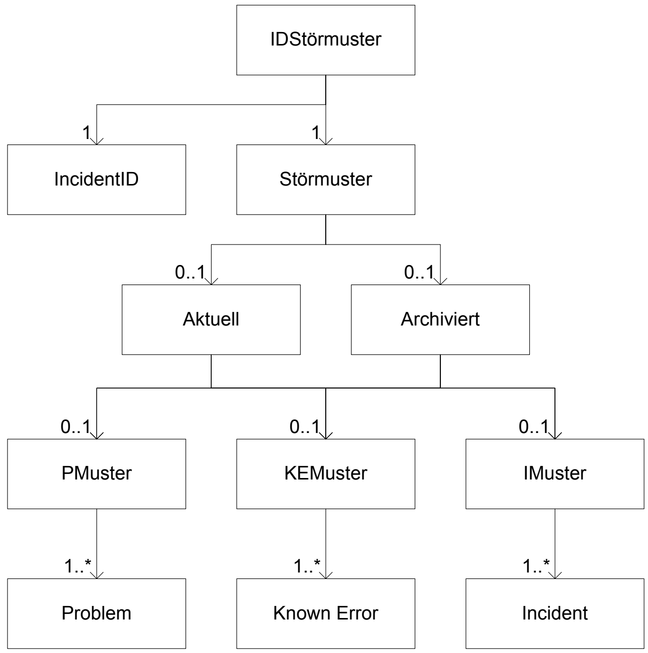


Abbildung 7.15.: Datentypstruktur für den Web Service „Korrelation“

```
<xsd:element name="Störmuster" type ="Muster" />
<xsd:complexType name="Muster">
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
      <xsd:sequence>
        <xsd:element name="Aktuell" type="TeilMuster"
          minOccurs="1" maxOccurs="1" />
        <xsd:element name="Archiviert" type="TeilMuster"
          minOccurs="1" maxOccurs="1" />
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TeilMuster">
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
      <xsd:sequence>
        <xsd:element name="PMuster" type="PNummern"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="KEMuster" type="KENummern"
          minOccurs="0" maxOccurs="1" />
        <xsd:element name="IMuster" type="INummern"
          minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PNummern">
  <xsd:complexContent>
    <xsd:restriction base="xsd:anyType">
```

```
          <xsd:sequence>
           <xsd:element name="Problem" type="xsd:long"
             minOccurs="1" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="KENummern">
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
           <xsd:element name="KnownError" type="xsd:long"
             minOccurs="1" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
    <xsd:complexType name="INummern">
      <xsd:complexContent>
        <xsd:restriction base="xsd:anyType">
          <xsd:sequence>
           <xsd:element name="Incident" type="xsd:long"
             minOccurs="1" maxOccurs="unbounded" />
          </xsd:sequence>
        </xsd:restriction>
      </xsd:complexContent>
    </xsd:complexType>
  </xsd:schema>
</types>
```

Interfacebeschreibung

Der Web Service „Korrelation“ beinhaltet drei funktionale Kapseln. Da die dadurch repräsentierten Funktionalitäten der Störmusterprüfung, Event- und Datenkorrelation i.d.R. längere Zeit in Anspruch nehmen, wird eine asynchrone Kommunikation spezifiziert. Dazu wird jede funktionale Kapsel, wie in Abschnitt 6.3.1.1 dargestellt, in ihre zwei Kommunikationsrichtungen aufgespalten. Für jede Richtung wird dann ein *operation*-Element definiert. Die Operationen der „Hin“-Richtung werden dabei dem Interface „WSBHin“ und die der asynchronen „Rück“-Richtung dem Interface „WSBRück“ zugeordnet.

```
<interface name="WSBHin">
  <!-- Anfrage an Kapsel "Störmuster prüfen"  -->
  <operation name="prüfen"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IncidentID" />
  </operation>
  <!-- Anfrage an Kapsel "Eventkorrelation durchführen"  -->
  <operation name="eventkorrelieren"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IDStörmuster" />
  </operation>
  <!-- Anfrage an Kapsel "Datenkorrelation durchführen"  -->
  <operation name="datenkorrelieren"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IDStörmuster" />
  </operation>
</interface>
```

```
<interface name="WSBRück">
  <!-- Antwort zur Kapsel "Störmuster prüfen"  -->
  <operation name="übermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:Störmuster" />
  </operation>
  <!-- Antwort zur Kapsel "Eventkorrelation durchführen"  -->
  <operation name="ekorrelübermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:IncidentID" />
  </operation>
  <!-- Antwort zur Kapsel "Datenkorrelation durchführen"  -->
  <operation name="dkorrelübermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:IncidentID" />
  </operation>
</interface>
```

PartnerLinkType-Spezifikation

Für die Kommunikationsverbindung mit dem BPEL-Prozess wird ein *partnerLinkType*-Element „WSBLinkType“ mit zwei *role*-Elementen definiert. Jede dieser Rollen wird dann über ihr *portType*-Element mit einem Interface verbunden.

```
<plnk:partnerLinkType name="WSBLinkType">
  <plnk:role name="WSBRolle1">
    <plnk:portType name="tns:WSBHin" />
  </plnk:role>
  <plnk:role name="WSBRolle2">
    <plnk:portType name="tns:WSBRück" />
  </plnk:role>
</plnk:partnerLinkType>
```

7.2.2.3. Web Service „Lösung/Workaround“

Diesem Web Service wurde die funktionale Kapsel „Lösung/Workaround ermitteln“ zugeordnet. Ausgehend von der im Rahmen des Beispiels modifizierten Datenmodellierung, die in Abbildung 7.16 dargestellt ist, wird die WSDL-Schnittstellenbeschreibung für diesen Web Service abgeleitet.

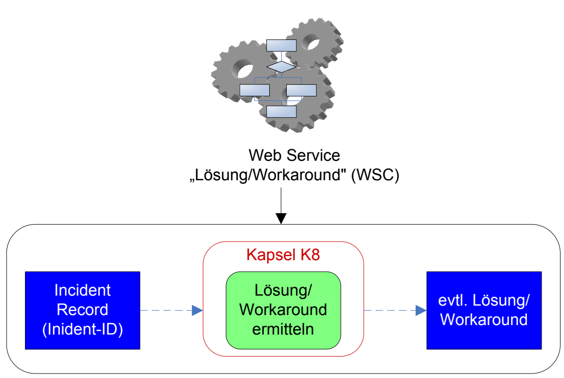


Abbildung 7.16.: Web Service „Lösung/Workaround“

Typbeschreibung

Wie bereits bei den vorangegangenen Spezifikationen der jeweiligen Typbeschreibung wird auch diesem Web Service die ID des Incident Records übergeben, so dass sie entsprechend importiert werden muss. Wird durch die funktionale Kapsel „Lösung/Workaround ermitteln“ eine (Übergangs-)Lösung ermittelt, so wird nicht diese als Output geliefert, sondern nur, ob eine gefunden wurde oder nicht. Da die (Übergangs-) Lösung innerhalb des Incident Records ausformuliert abgespeichert wird, kann der hier zurückgelieferte Outputtyp lokal als Typ „xsd:boolean“ spezifiziert werden.

```
<types>
  <xsd:import namespace="http://example.org/Prototyp/IR.xsd"
    schemaLocation="IR.xsd" />
  <xsd:schema targetNamespace="http://example.org/prototyp/wsC/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="LWgefunden" type="xsd:boolean" />
</types>
```

Interfacebeschreibung

Das Ermitteln einer Lösung oder eines Workarounds im Rahmen der betrachteten „ersten Unterstützung“ für die Behebung eines Incidents benötigt Zeit. Aus diesem Grund wird die Kommunikation mit dem Web Service wiederum asynchron über zwei Operationen („ermitteln“ und „übermitteln“) und zwei Interfaces („WSCHin“ und „WSCRück“) modelliert.

```
<interface name="WSCHin">
  <operation name="ermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IncidentID" />
  </operation>
</interface>
<interface name="WSCRück">
  <operation name="übermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:LWgefunden" />
  </operation>
</interface>
```

PartnerLinkType-Spezifikation

Entsprechend der asynchronen Kommunikation werden über die zwei Rollen innerhalb eines *partnerLinkType*-Elements „WSCLinkType“ die Interfaces „WSCHin“ und „WSCRück“ angebunden.

```
<plnk:partnerLinkType name="WSCLinkType">
  <plnk:role name="WSCRolle1">
    <plnk:portType name="tns:WSCHin" />
  </plnk:role>
  <plnk:role name="WSCRolle2">
    <plnk:portType name="tns:WSCRück" />
  </plnk:role>
</plnk:partnerLinkType>
```

7.2.2.4. Web Service „User Service Request“

Dem Web Service „User Service Request“ schließlich wurde die funktionale Kapsel „User Service Request bearbeiten“ zugeordnet. Ausgehend von der im Rahmen des Beispiels modifizierten Datenmodellierung nach Abbildung 7.17 wird die WSDL-Schnittstellenbeschreibung für diesen Web Service abgeleitet.

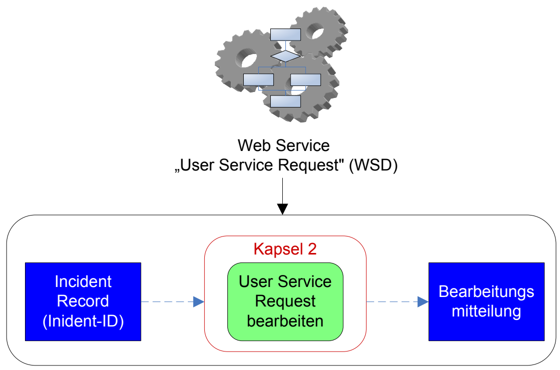


Abbildung 7.17.: Web Service „User Service Request“

Typbeschreibung

Der funktionalen Kapsel „User Service Request bearbeiten“ wird die ID des Incident Records übergeben, auch wenn durch den realisierenden Web Service tatsächlich ein User Service Request bearbeitet wird. Da jedoch bisher keine Differenzierung bzgl. der Datenerfassung vorgenommen wurde, kann die Bearbeitung der Meldung als User Service Request auf den bisher gesammelten Daten aufbauen. Diese müssen dann nur im jeweils angepassten Kontext betrachtet werden. Auch wenn im Rahmen dieser Arbeit die Bearbeitung von Incidents im Vordergrund steht, wird für die in Abbildung 7.17 dargestellte funktionale Kapsel ein „string“-Output modelliert. Dieser wird vom später definierten BPEL-Prozess nach Beendigung der Bearbeitung an den Client zurückgeliefert. Außerdem muss die Definition der „IncidentID“ importiert werden.

```
<types>
  <xsd:import namespace="http://example.org/prototyp/IR.xsd"
    schemaLocation="IR.xsd" />
  <xsd:schema targetNamespace="http://example.org/prototyp/wsD/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ErgebnisUSRBearbeitung" type="xsd:string" />
  </xsd:schema>
</types>
```

Interfacebeschreibung

Für die Schnittstellenbeschreibung des Web Services „4“ werden zwei *interface*-Elemente mit jeweils einem *operation*-Element spezifiziert. Die Operation „USRbearbeiten“ des Interface „WSDHin“ repräsentiert dabei das Senden der Anfrage an den realisierenden Web Service zur Bearbeitung des User Service Requests, wohingegen die Operation „übermitteln“ zum asynchronen Zurücksenden einer Nachricht dient, wenn die Bearbeitung abgeschlossen ist.

```
<interface name="WSDHin">
  <operation name="USRbearbeiten"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IncidentID" />
  </operation>
</interface>
<interface name="WSDRück">
  <operation name="übermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:ErgebnisUSRBearbeitung" />
  </operation>
</interface>
```

PartnerLinkType-Spezifikation

Die Definition des *partnerLinkType*-Elements erfolgt analog zum Web Service „Lösung/Workaround“. Für jedes der zwei Interfaces wird ein darauf referenzierendes *portType*-Element innerhalb je einer Rolle des *partnerLinkType*-Elements „WSDLinkType“ definiert.

```
<plnk:partnerLinkType name="WSDLinkType">
  <plnk:role name="WSDRolle1">
    <plnk:portType name="tns:WSDHin" />
  </plnk:role>
  <plnk:role name="WSDRolle2">
    <plnk:portType name="tns:WSDRück" />
  </plnk:role>
</plnk:partnerLinkType>
```

7.2.2.5. Web Service „BPEL-Service“

Dieser Web Service umfasst die gesamte in Abbildung 7.2 dargestellte Prozesskette. Da diese einen Teil des Incident-Management-Prozesses aus Abbildung 4.20 darstellt und somit Input bzw. Output für diesen Web Service nicht explizit spezifiziert ist, werden diese künstlich definiert. Dadurch kann später ein anfragender Service über die WSDL-Schnittstellenbeschreibung mit dem dahinter verborgenen BPEL-Prozess kommunizieren.

Typbeschreibung

Da zu Beginn des Beispiel bereits eine Beschreibung der Störungsmeldung in Form eines Incident Records existiert und dieser eine eindeutige Identifikationsnummer besitzt, wird als Input für den Web Service „BPEL-Service“ diese ID vom Service Requestor übergeben. Der Datentyp für die ID des Incidents wird wiederum der Definition auf Seite 132 entnommen. Ist die in diesem Beispiel nur auszugsweise dargestellte Klassifikation und erste Unterstützung abgeschlossen, liefert der Web Service einen Output „ErgebnisBearbeitung“ der Form „xsd:string“.

```
<types>
  <xsd:import namespace="http://example.org/prototyp/IR.xsd"
    schemaLocation="IR.xsd" />
  <xsd:schema targetNamespace="http://example.org/prototyp/bpel/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:element name="ErgebnisBearbeitung" type="xsd:string" />
  </xsd:schema>
</types>
```

Interfacebeschreibung

Für den Zugriff auf die Funktionalität des Web Services „BPEL-Service“ wird eine asynchrone Kommunikation über zwei Operationen definiert, da die dargestellte Klassifikation und erste Unterstützung den angesprochenen Web Services zufolge eine längere Zeit beansprucht und der anfragende Service in dieser Zeit nicht blockiert werden soll. Über die Operation „bearbeiten“ kann unter Angabe der Incident-ID eine Anfrage an den Web Service gesendet werden. Ist die Bearbeitung der Störungsmeldung abgeschlossen, so erhält der anfragende Service über die Operation „übermitteln“ asynchron eine Antwort auf seine Anfrage.

```
<interface name="WSHHin">
  <operation name="bearbeiten"
    pattern="http://www.w3.org/2005/08/wsdl/in-only">
    <input messageLabel="In" element="tns:IncidentID" />
  </operation>
</interface>
```

```
<interface name="WSHRück">
  <operation name="übermitteln"
    pattern="http://www.w3.org/2005/08/wsdl/out-only">
    <output messageLabel="Out" element="tns:ErgebnisBearbeitung" />
  </operation>
</interface>
```

PartnerLinkType-Spezifikation

Auch wenn sich hinter dem hier definierten WSDL-File ein BPEL-Prozess verbergen wird, muss auch für die Kommunikationsverbindung zwischen dem anfragenden Web Service und diesem BPEL-Prozess ein *partnerLinkType*-Element definiert werden. Dieses enthält aufgrund der gewählten asynchronen Kommunikation zwei *role*-Elemente „WSHRolle1“ und „WSHRolle2“, die über ihre *portType*-Elemente die beiden definierten *interface*-Elemente referenzieren.

```
<plnk:partnerLinkType name="WSHLinkType">
  <plnk:role name="WSHRolle1">
    <plnk:portType name="tns:WSHHin" />
  </plnk:role>
  <plnk:role name="WSHRolle2">
    <plnk:portType name="tns:WSHRück" />
  </plnk:role>
</plnk:partnerLinkType>
```

Mit der Spezifikation des Web Services „BPEL-Service“ wird die WSDL-Schnittstellenbeschreibung der Prozess- und Intermediarydienste im Rahmen der prototypischen Implementierung für das Beispiel abgeschlossen. Im nächsten Abschnitt erfolgt dann die Spezifikation der Orchestrierung mittels BPEL. Für den daraus resultierenden BPEL-Prozess wurde bereits die WSDL-Schnittstellenbeschreibung geliefert. Über diese kann ein anfragender Service mit dem BPEL-Prozess des betrachteten Teilabschnitts des Incident-Management-Prozesses interagieren.

7.2.3. BPEL-Spezifikation

Dieser Abschnitt befasst sich mit der BPEL-Spezifikation der Orchestrierung des Beispiels. Ausgangspunkt dafür ist die Zuordnung der funktionalen Kapseln zu den Web Services der SOA gemäß Tabelle 7.2. Außerdem werden die Service-Hierarchie und die WSDL-Schnittstellenbeschreibungen aus 7.2.1 und 7.2.2 benötigt. Aufbauend auf diesen Informationen wird für den Web Service „BPEL-Service“ die BPEL-Spezifikation gemäß der in Abschnitt 6.4.2 vorgestellten Vorgehensweise abgeleitet. Im Rahmen dieser Beschreibung werden zuerst die Daten- und Kommunikationsbeziehungen definiert, bevor im Anschluss daran die Interaktionen zwischen den Web Services modelliert werden.

Bevor die Spezifikation des BPEL-Prozesses entwickelt wird, zeigt Abbildung 7.18 die zu beschreibenden Interaktionen des BPEL-Prozesses mit den zuvor definierten Web Services. In dieser Abbildung wurden alle Daten, welche vom BPEL-Prozess gemäß der WSDL-Spezifikation im vorherigen Abschnitt an die Web Services übergeben und von ihnen zurückgegeben werden, explizit dargestellt.

Die Verknüpfungsoperatoren wurden für die spätere Beschreibung der Interaktionsfolge durchnummeriert, da sie Hinweise für die Modellierung der Interaktionsbeziehungen geben. Des Weiteren wurde ausgehend von Abbildung 7.12 auf die Darstellung folgender Elemente in Abbildung 7.18 verzichtet:

- Basisdienste, die service-intern sind und nicht vom BPEL-Prozess aufgerufen werden.
- Ereignisse, welche nicht zur Auswahl der nächsten auszuführenden Interaktion dienen.
- Daten, die nicht vom BPEL-Prozess an den Web Service übergeben werden müssen.

Abbildung 7.18.: Graphische Darstellung der zu beschreibenden Interaktionen

Für die Spezifikation des BPEL-Prozesses werden folgende Namensräume verwendet:

```
<process name="WSH BPEL"
  targetNamespace="http://example.org/prototyp/bpel/"
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:bp="http://example.org/prototyp/bpel/"
  xmlns:wsA="http://example.org/prototyp/wsA/"
```

```
  xmlns:wsB="http://example.org/prototyp/wsB/"
  xmlns:wsC="http://example.org/prototyp/wsC/"
  xmlns:wsD="http://example.org/prototyp/wsD/">
  ...
</process>
```

Datenbeziehung

Im Rahmen der Beschreibung der Datenbeziehungen zwischen den Web Services müssen für alle Datentypen, die als Input oder Output verwendet werden, *variable*-Elemente definiert werden. Nachfolgend ist eben diese Spezifikation für den BPEL-Prozess dargestellt. Für die ID des Incidents Records ist es dabei ausreichend, eine einzige Datenvariable zu definieren, da alle Web Services dieselbe globale Definition für den Datentyp „InicdentID“ verwenden.

```
<variables>
  <!-- Input von BPEL-Prozess bzw. globale Variable-->
  <variable name="IncidentID" element="bp:IncidentID" />
  <!-- Output von BPEL-Prozess -->
  <variable name="Ergebnis" element="bp:ErgebnisBearbeitung" />
  <!-- Output von Web Service A -->
  <variable name="MKlasse" element="wsA:Meldungsklasse" />
  <variable name="MKategorie" element="wsA:Meldungskategorie" />
  <variable name="Priorität" element="wsA:Priorität" />
  <!-- Input von Web Service B -->
  <variable name="IDSMuster" element="wsB:IDStörmuster" />
  <!-- Output von Web Service B -->
  <variable name="SMuster" element="wsB:Störmuster" />
  <!-- Output von Web Service C -->
  <variable name="LWgefunden" element="wsC:LWgefunden" />
  <!-- Output von Web Service D -->
  <variable name="ErgebnisUSR" element="wsD:ErgebnisUSRBearbeitung" />
</variables>
```

Kommunikationsbeziehung

Nach der Definition der Datenvariablen muss für jede mögliche Kommunikationsverbindung, die vom BPEL-Prozess benötigt wird, ein *partnerLink*-Element definiert werden. Da der anfragende Service mit dem BPEL-Prozess über die WSDL-Schnittstellenbeschreibung des Web Service „BPEL-Service“ kommuniziert, wird als erstes ein *partnerLink*-Element mit dem Namen „Client“ zwischen diesen beiden definiert. Die weiteren *partnerLink*-Elemente definieren die Verbindungen zu den Intermediarydiensten „Klassifikation“, „Korrelation“, „Lösung/Workaround“ und „User Service Request“, mit denen der BPEL-Prozess interagieren soll.

```
<partnerLinks>
  <!-- PartnerLink Web Service H und BPEL-Prozess. -->
  <partnerLink name="Client"
    partnerLinkType="bp:WSHLinkType"
    myRole="WSHRolle1" partnerRole="WSHRolle2" />
  <!-- PartnerLink Web Service A und BPEL-Prozess. -->
  <partnerLink name="WSAPartnerLink"
    partnerLinkType="wsA:WSALinkType"
    partnerRole="WSARolle" />
  <!-- PartnerLink Web Service B und BPEL-Prozess. -->
  <partnerLink name="WSBPartnerLink"
    partnerLinkType="wsB:WSBLinkType"
    myRole="WSBRolle1" partnerRole="WSBRolle2" />
  <!-- PartnerLink Web Service C und BPEL-Prozess. -->
  <partnerLink name="WSCPartnerLink"
    partnerLinkType="wsC:WSCLinkType"
    myRole="WSCRolle1" partnerRole="WSCRolle2" />
```

```
  <!-- PartnerLink Web Service D und BPEL-Prozess. -->
  <partnerLink name="WSDPartnerLink"
    partnerLinkType="wsD:WSDLinkType"
    myRole="WSDRolle1" partnerRole="WSDRolle2" />
</partnerLinks>
```

Interaktionsbeziehung

Für die Beschreibung der Interaktionsbeziehung müssen die Funktionalitäten der aufzurufenden Web Services, die Verbindungen zwischen diesen Aufrufen und die Reihenfolge betrachtet werden[2]. Der BPEL-Prozess wird asynchron definiert. Die Bearbeitung einer Störungsmeldung wird durch einen Request vom anfragenden Service ausgelöst, der die ID des zu untersuchenden Incidents an den BPEL-Prozess sendet. Dazu ruft der Service Requestor die Operation „bearbeiten“ des Interfaces „WSHHin“ auf, welche in der Schnittstellenbeschreibung des Web Service „BPEL-Service“ definiert ist. Dies wird durch ein *receive*-Element über den PartnerLink „Client“ realisiert. Die erhaltene ID wird in der Variable „IncidentID“ zwischengespeichert.

```
<sequence name="interaktionsbeispiel">
  <!-- Anfrage annehmen -->
  <receive partnerLink="Client"
   portType="bp:WSHHin" operation="bearbeiten"
   variable="IncidentID" createInstance="yes" />
  ...
</sequence>
```

Nachdem die ID vom Client übergeben wurde, muss als erstes die funktionale Kapsel „Störungsmeldung klassifizieren“ des Web Service „Klassifikation“ aufgerufen werden. Da die zugehörige Operation synchron definiert ist, erfolgt der Aufruf durch ein synchron definiertes *invoke*-Element. Als Input wird dem Web Service die in der Variable „IncidentID“ zwischengespeicherte ID der Störungsmeldung übergeben. Der Output der aufgerufenen Operation wird der Variable „MKlasse“ zugewiesen.

```
<sequence name="interaktionsbeispiel">
  ...
  <!-- Web Service A Kapsel "Störungsmeldung klassifizieren" -->
  <invoke partnerLink="WSAPartnerLink"
    portType="wsA:WSAInterface" operation="klassifizieren"
    inputVariable="IncidentID" outputVariable="MKlasse" />
  ...
</sequence>
```

Im Anschluss an die Klassifizierung der Störungsmeldung verzweigt sich der in Abbildung 7.18 dargestellte Prozessstrang. Der Verknüpfungsoperator, welcher durch die Nummer „1“ gekennzeichnet ist, wird entsprechend Abbildung 6.29 rechts in BPEL durch ein *switch*-Element spezifiziert. Die Entscheidung, wie die weitere Bearbeitung aussieht, wird dabei anhand der Variable „MKlasse“ getroffen. Der Datentyp dieser Variable ist „Meldungsklasse“, welcher global in der „IR.xsd“ definiert wurde und die drei möglichen Werte „User Service Request“, „Incident“ und „Standard-Incident“ annehmen kann, welche die drei Klassen von Störungsmeldungen repräsentieren, anhand derer die weitere Bearbeitung unterschieden wird.

Da die Behandlung der Incidents und Standard-Incidents an dieser Stelle noch nicht unterschieden wird, ist es ausreichend, die User Service Requests zu identifizieren und separat zu bearbeiten.

```
<sequence name="interaktionsbeispiel">
  ...
  <!-- zwischen Incident und User Service Request unterscheiden -->
  <switch name="verknüpfung1">
```

[2] Eine Zusammenfassung der BPEL-Codierung der Interaktionen ist in Anhang D dargestellt.

```
    <case condition="bpws:getVariableData('MKlasse') = 'User Service Request'">
      <!-- User Service Request bearbeiten -->
      <sequence name="usrbearbeiten">
      ...
      </sequence>
    </case>
    <otherwise>
      <!-- Incident/Standard-Incident bearbeiten -->
      <sequence name="bearbeiten">
      ...
      </sequence>
    </otherwise>
  </switch>
  ...
</sequence>
```

Für die Bearbeitung eines User Service Requests muss die funktionale Kapsel „User Service Request bearbeiten" des Web Service „User Service Request" aufgerufen werden. Da die Kommunikation asynchron über die Operationen „USRbearbeiten" und „übermitteln" spezifiziert ist, wird innerhalb des BPEL-Prozesses zuerst mittels *invoke* ein Request an den Web Service „User Service Request" gesendet. Anschließend wartet der BPEL-Prozess auf dessen Antwort. Dies wird durch das anschließende *receive*-Element ausgedrückt. Das Ergebnis der Bearbeitung des User Service Requests wird in der Variable „ErgebnisUSR" gespeichert. Weil das Ergebnis des Web Service „User Service Request" nicht gleich in der Variable „Ergebnis" abgelegt wird, könnte diese Antwort noch modifiziert werden, bevor sie an den anfragenden Service zurückgesendet wird. Dies ist durch das *assign*-Element angedeutet. Da jedoch angenommen wird, dass keine Datentransformation notwendig ist, wird der Inhalt der Variable „ErgebnisUSR" einfach in die Variable „Ergebnis" kopiert. Anschließend wird über den PartnerLink „Client" dieses Ergebnis an den Service Requestor zurückgesandt, indem die Operation „übermitteln" des Web Service „BPEL-Service" aufgerufen wird.

```
<!-- User Service Request bearbeiten -->
<sequence name="usrbearbeiten">
  <!-- Web Service D Kapsel "User Request bearbeiten" -->
  <invoke partnerLink="WSDPartnerLink"
   portType="wsD:WSDHin" operation="USRbearbeiten"
   inputVariable="IncidentID" />
  <receive partnerLink="WSDPartnerLink"
   portType="wsD:WSDRück" operation="übermitteln"
   variable="ErgebnisUSR" />

  <assign>
    <copy>
      <from variable="ErgebnisUSR" />
      <to variable="Ergebnis" />
    </copy>
  </assign>

  <!-- Meldung zurücksenden -->
  <invoke partnerLink="Client"
   portType="bp:WSHRück" operation="übermitteln"
   inputVariable="Ergebnis" />
</sequence>
```

Der zweite Ast des durch die Verknüpfung Nummer „1" aufgespaltenen Prozessstrangs behandelt die Bearbeitung von Incidents und Standard-Incidents. Diese müssen kategorisiert und priorisiert werden. Dazu werden die funktionalen Kapseln „Incident kategorisieren" und „Priorität ermitteln" des Web Service „Klassifikation" sequenziell nacheinander und jeweils synchron über die entsprechende WSDL-Operation der Schnittstellenbeschreibung aufgerufen. Jeder dieser Operationen wird dazu die ID des Incident Records übergeben. Die durch die Kategorisierung des Incidents ermittelte Meldungskategorie wird der Variable „MKategorie“, und die Priorität der Variable „Priorität“ zugewiesen.

```
<sequence name="bearbeiten">
  <!-- Web Service A Kapsel "Incident kategorisieren" -->
  <invoke partnerLink="WSAPartnerLink"
   portType="wsA:WSAInterface" operation="kategorisieren"
   inputVariable="IncidentID" outputVariable="MKategorie" />

  <!-- Web Service A Kapsel "Priorität ermitteln" -->
  <invoke partnerLink="WSAPartnerLink"
   portType="wsA:WSAInterface" operation="priorisieren"
   inputVariable="IncidentID" outputVariable="Priorität" />
  ...
</sequence>
```

Nachdem die Kategorie und die Priorität der (Standard-)Incidents bestimmt wurde, verzweigt sich jetzt auch deren weitere Bearbeitung. Die mit der Nummer „2“ bezeichnete Verzweigung wird, wie in Abschnitt 6.4.2.3 beschrieben, ebenfalls durch ein *switch*-Element in BPEL umgesetzt. Als Entscheidungsgrundlage dient erneut der Inhalt der Variable „MKlasse“. Für die Standard-Incidents müsste nun die durch den Prozesswegweiser „Standard-Incident bearbeiten“ dargestellte funktionale Kapsel ausgeführt werden. Da diese jedoch im Rahmen des Beispiels nicht betrachtet wird[3], ist die Implementierung der Bearbeitung von Standard-Incidents hiermit abgeschlossen.

```
<sequence name="bearbeiten">
  ...
  <!-- zwischen Standard-Incident und Incident unterscheiden -->
  <switch name="verknüpfung2">
    <case condition="bpws:getVariableData('MKlasse') = 'Standard-Incident'">
      <!-- Standard-Incident bearbeiten -->
    </case>
    <otherwise>
      <!-- Incident bearbeiten -->
      <sequence name="incidentbearbeiten">
      ...
      </sequence>
    </otherwise>
  </switch>
  ...
</sequence>
```

Der weitere Prozessstrang beschreibt die Bearbeitung eines Incidents, der nicht als Standard-Incident eingestuft wurde. Für diesen wird zuerst eine Störmusterprüfung durchgeführt. Dazu wird über die Operation „prüfen“ des Web Service „Korrelation“ dessen funktionale Kapsel „Störmuster prüfen“ asynchron aufgerufen. Anhand der ID des Incident Records kann diese feststellen, ob das Muster der betrachteten Störung mit bereits archivierten bzw. momentan in Bearbeitung befindlichen Datensätzen übereinstimmt. Das Ergebnis dieser Analyse wird in der Variable „SMuster“ gespeichert, welche dem Datentyp „Störmuster“ entspricht.

```
<sequence name="incidentbearbeiten">
  <!-- Web Service B Kapsel "Störmuster prüfen" -->
  <invoke partnerLink="WSBPartnerLink"
   portType="wsB:WSBHin" operation="prüfen"
   inputVariable="IncidentID" />
```

[3]Erklärt wird dies in Abschnitt 7.1.2, zu dem die Abbildungen 7.2 und 7.7 gehören.

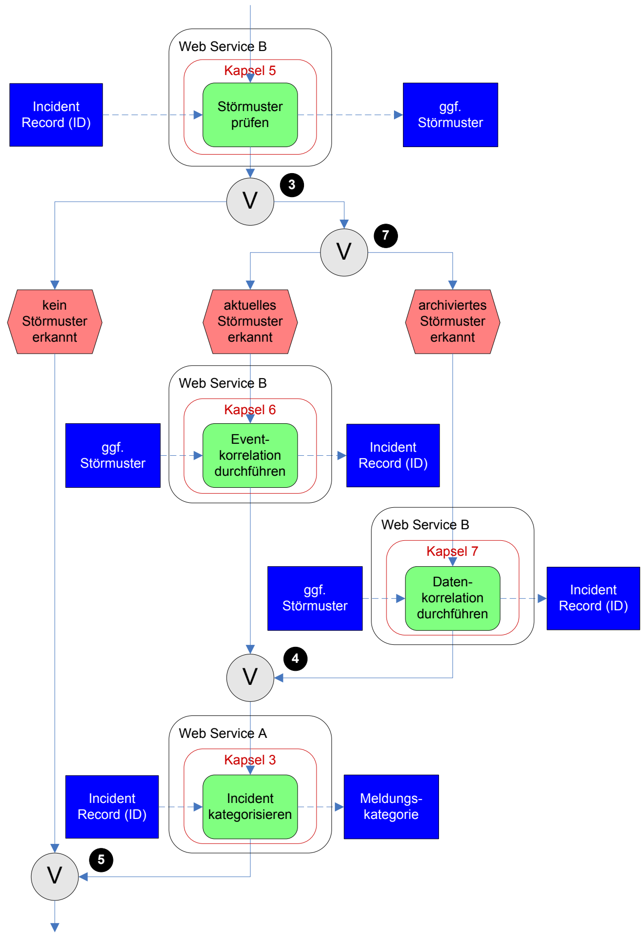


Abbildung 7.19.: modifizierte Verzweigung zur Redundanzvermeidung

```
  <receive partnerLink="WSBPartnerLink"
   portType="wsB:WSBRück" operation="übermitteln"
   variable="SMuster" />
  ...
</sequence>
```

Anhand der Daten, welche im Rahmen der Störmusterprüfung zurückgeliefert werden, muss in Verbindung mit Verzweigung „3“ ein *switch*-Element definiert werden, welches zwischen den drei durch die Ereignisse („kein/aktuelles/archiviertes Störmuster erkannt“) in Abbildung 7.18 angegebenen Situationen differenziert. Da sich zwei der Prozessstränge nach dem Aufruf einer funktionalen Kapsel wieder vereinigen (siehe Verknüpfung „4“), in BPEL jedoch ein *switch*-Element nur vollständig abgeschlossen werden kann, wird die Verzweigung „3“ so abgewandelt, dass sie nur unterscheidet, ob ein Störmuster identifiziert wurde oder nicht. Außerdem wird eine zusätzliche Verzweigung „7“ eingeführt, welche anschließend die Unterscheidung zwischen aktuellen und archivierten Störmustern vornimmt. Wird Verzweigung „3“ nicht wie beschrieben abgewandelt, so muss das Füllen der Datenvariable „IDSMuster“ entweder für alle drei Prozessstränge ausgeführt oder an mehreren Stellen definiert werden. Außerdem müsste in beiden Fällen, in denen ein Störmuster erkannt wird, die Interaktion mit der funktionalen Kapsel „Incident kategorisieren“ spezifiziert werden. Diese Aktionen müssen nun durch die in Abbildung 7.19 dargestellte modifizierte Verzweigung des Prozessstranges nicht redundant definiert oder unnötig ausführt werden.

Das nachfolgende *switch*-Element zeigt die Bedingung für das Vorhandensein von Störmustern gemäß der modifizierten Darstellung in Abbildung 7.19. Auf den Fall, dass kein Muster für den Incident identifiziert werden kann, wird innerhalb dieses *switch*-Elements verzichtet: Die nächste Aktion („Lösung/Workaround ermitteln“), die in diesem Fall durchgeführt werden muss, muss nämlich auch dann ausgeführt werden, wenn zuvor ein Störmuster erkannt wurde (siehe Verknüpfung „5“). Deshalb wird sie im Anschluss an das dargestellte *switch*-Element spezifiziert. Kann ein Störmuster, welches in der Variable „SMuster“ zwischengespeichert wird, für den aktuell zu bearbeitenden Incident identifiziert werden, muss dieses in den „Störmuster“-Teil, und die ID des Incidents in den „IncidentID“-Teil der Variable „IDSMuster“ kopiert werden. Dies ist notwendig, da die funktionalen Kapseln „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“, die im Falle einer positiven Störmusterprüfung ausgeführt werden müssen, eine Nachricht vom Datentyp „IDStörmuster“ fordern. Die Variable „IDSMuster“ entspricht diesem Datentyp und wird über das abgebildete *assign*-Element mit den vorhandenen Werten belegt.

```
<sequence name="incidentbearbeiten">
  ...
  <switch name="verknüpfung3">
    <case condition=
      "count(bpws:getVariableData('SMuster','/*')) >= 1">
      <!-- Störungsmuster identifiziert -->
      <!-- Datenvariable füllen -->
      <assign>
        <copy>
          <from variable="SMuster" />
          <to variable="IDSMuster" part="Störmuster" />
        </copy>
        <copy>
          <from variable="IncidnetID" />
          <to variable="IDSMuster" part="IncidentID" />
        </copy>
      </assign>
      <!-- Störungsmuster identifiziert -->
      ...
    </case>
  </switch>
  <!-- Lösung/Workaround ermitteln -->
  ...
</sequence>
```

Nachfolgend wird das zusätzlich eingeführte und die funktionalen Kapseln „Eventkorrelation durchführen“ und „Datenkorrelation durchführen“ umfassende *switch*-Element vorgestellt, welches im Anschluss an das Füllen der Datenvariable „IDSMuster“ ausgeführt wird. Dieses *switch*-Element steht für die Verzweigung Nummer „7“ (siehe Abb. 7.19) und dient der Unterscheidung zwischen aktuellen und archivierten Störmustern. Steht der betrachtete Incident z.B. mit anderen aktuellen Incidents, Problems oder Known Error in Verbindung, so muss die funktionale Kapsel „Eventkorrelation durchführen“ vom Web Service „Korrelation“ ausgeführt werden. Im Gegensatz dazu wird die funktionale Kapsel „Datenkorrelation durchführen“ ausgeführt, wenn für den Incident nur Störmuster mit bereits archivierten Datensätzen identifiziert wurden. Bei der ID, die von den Operationen zurückgegeben wird, beachte man, dass diese nicht zwingend mit der zuvor übergebenen übereinstimmen muss. Dies kann bspw. der Fall sein, wenn mehrere aktuelle Incidents verknüpft werden und für diese Verknüpfung ein neuer Datensatz mit einer neuen gemeinsamen Identifikationsnummer angelegt wird. Auf sich daraus eventuell ergebende Abwandlungen des dargestellten Prozessstrangs wird im Rahmen der hier beschrieben Modellierung verzichtet.

```
<switch name="verknüpfung3">
  ...
  <!-- Störungsmuster identifiziert -->
  <switch name="verknüpfung7">
```

```
    <case condition="count(bpws:getVariableData('SMuster','/Aktuell')) = 1">
      <!-- es wurde mind. ein aktuelles Störmuster erkannt -->
      <!-- Web Service B Kapsel "Eventkorrelation durchführen" -->
      <invoke partnerLink="WSBPartnerLink"
       portType="wsB:WSBHin" operation="eventkorrelieren"
       inputVariable="IDSMuster" />
      <receive partnerLink="WSBPartnerLink"
       portType="wsB:WSBRück" operation="ekorrelübermitteln"
       variable="IncidentID" />
    </case>

    <case condition="count(bpws:getVariableData('SMuster','/Archiviert')) = 1">
      <!-- es wurde kein aktuelles Störmuster erkannt -->
      <!-- Web Service B Kapsel "Datenkorrelation durchführen" -->
      <invoke partnerLink="WSBPartnerLink"
       portType="wsB:WSBHin" operation="datenkorrelieren"
       inputVariable="IDSMuster" />
      <receive partnerLink="WSBPartnerLink"
       portType="wsB:WSBRück" operation="dkorrelübermitteln"
       variable="IncidentID" />
    </case>
  </switch>
  ...
</switch>
```

Da nach einer Event- oder Datenkorrelation u.U. die Kategorie des Incidents nicht mehr zutreffend ist, muss der Incident neu kategorisiert werden. Dazu wird im Anschluss an das zuvor beschriebene *switch*-Element die Operation „kategorisieren“ der WSDL-Schnittstellenbeschreibung des Web Services „Klassifikation“ über das nachfolgende *invoke*-Element aufgerufen.

```
<switch name="verknüpfung3">
  ...
  <!-- Web Service A Kapsel "Incident kategorisieren" -->
  <invoke partnerLink="WSAPartnerLink"
   portType="wsA:WSAInterface" operation="kategorisieren"
   inputVariable="IncidentID" outputVariable="MKategorie" />
  ...
</switch>
```

Nachdem durch das Kategorisieren des Incidents das *switch*-Element, welches durch die Verzweigungen „3“ und „5“ in Abbildung 7.18 repräsentiert wird, abgeschlossen ist, wird die funktionale Kapsel „Lösung/Workaround ermitteln“ des Web Service „Lösung/Workaround“ ausgeführt. Hierzu wird zuerst durch Aufruf der Operation „ermitteln“ die ID des Incidents an diesen Web Service übermittelt. Anschließend wird asynchron in Form eines *receive*-Elements auf das von dieser Funktionalität gelieferte Ergebnis gewartet, das in der Variable „LWgefunden“ vom Typ *xsd:boolean* abgelegt wird.

```
<sequence name="incidentbearbeiten">
  ...
  <!-- Lösung/Workaround ermitteln -->
  <!-- Web Service C Kapsel "Lösung/Workaround ermitteln" -->
  <invoke partnerLink="WSCPartnerLink"
   portType="wsC:WSCHin" operation="ermitteln"
   inputVariable="IncidentID" />
  <receive partnerLink="WSCPartnerLink"
   portType="wsC:WSCRück" operation="übermitteln"
   variable="LWgefunden" />
  ...
</sequence>
```

Für die XOR-Verknüpfung Nummer „6“ in Abbildung 7.18 wird wiederum ein *switch*-Element spezifiziert. Der boolesche Wert der Variable „LWgefunden“ wird zur Fallunterscheidung innerhalb des *switch*-Elements verwendet. Wurde eine Lösung oder ein Workaround für den Incident gefunden, so enthält die Variable „LWgefunden“ den Wert „true“. Der Incident kann daraufhin dem Subprozess „Beheben und Wiederherstellen“ übergeben werden. Wird keine (Übergangs-)Lösung gefunden (Wert „false“), so müssen die durch die Prozesswegweiser „Zeitfenster erweitern“ und „Analyse und Diagnose“ in Abbildung 7.18 angedeuteten gestrichelt blau umrandeten funktionalen Kapseln ausgeführt werden. Diese und die funktionale Kapsel „Beheben und Wiederherstellen“ werden im Rahmen des hier betrachteten Beispiels jedoch nicht behandelt. Somit ist für uns die Bearbeitung des Incidents abgeschlossen. Da der BPEL-Prozess asynchron über die Operation „übermitteln“ der WSDL-Schnittstellenbeschreibung des Web Services „BPEL-Service“ eine Nachricht vom Typ „xsd:string“ an den anfragenden Service zurücksendet, wird für beide Fälle des dargestellten *switch*-Elements künstlich eine Rückgabenachricht erstellt. Dazu wird über ein *assign*-Element ein Text in die Variable „Ergebnis“ kopiert, welcher anschließend über ein *invoke*-Element an den Client gesendet wird.

```
<sequence name="incidentbearbeiten">
  ...
  <switch name="verknüpfung6">
    <case condition="bpws:getVariableData('LWgefunden')">
      <!-- es wurde eine (Übergangs-)Lösung gefunden -->
      <!-- "Beheben und Wiederherstellen" -->
      <!-- Ergebnisvariable füllen -->
      <assign>
        <copy>
          <from expression="string('Beheben und Wiederherstellen einleiten.')" />
          <to variable="Ergebnis" />
        </copy>
      </assign>
    </case>
    <otherwise>
      <!-- es wurde keine (Übergangs-)Lösung gefunden  -->
      <!-- "Zeitfenster erweitern" und dann "Analyse und Diagnose" -->
      <!-- Ergebnisvariable füllen -->
      <assign>
        <copy>
          <from expression="string('Analyse und Diagnose einleiten.')" />
          <to variable="Ergebnis" />
        </copy>
      </assign>
    </otherwise>
  </switch>

  <!-- Meldung zurücksenden -->
  <invoke partnerLink="Client"
   portType="bp:WSHRück" operation="übermitteln"
   inputVariable="Ergebnis" />
</sequence>
```

Durch die Rückgabe des Ergebnisses der in diesem Beispiel dargestellten Bearbeitung einer Störungsmeldung ist das *switch*-Element, welches aus der Verknüpfung „1“ in Abbildung 7.18 resultierte, abgeschlossen. Da anschließend keine weiteren Interaktionen mit Web Services mehr modelliert werden müssen, ist somit die Beschreibung der Interaktionsbeziehung für das Beispiel abgeschlossen.

Ein Teil der mit BPEL spezifizierten Interaktionen zwischen dem anfragenden Service Requestor und dem BPEL-Prozess sowie die zwischen dem BPEL-Prozess und den ausführenden Web Services werden in Abbildung 7.20 durch ein leicht abgewandeltes UML-Sequenzdiagramm visualisiert[4]. Dabei bezieht sich

[4]Die UML-Spezifikation ist [OMG 01a] zu entnehmen.

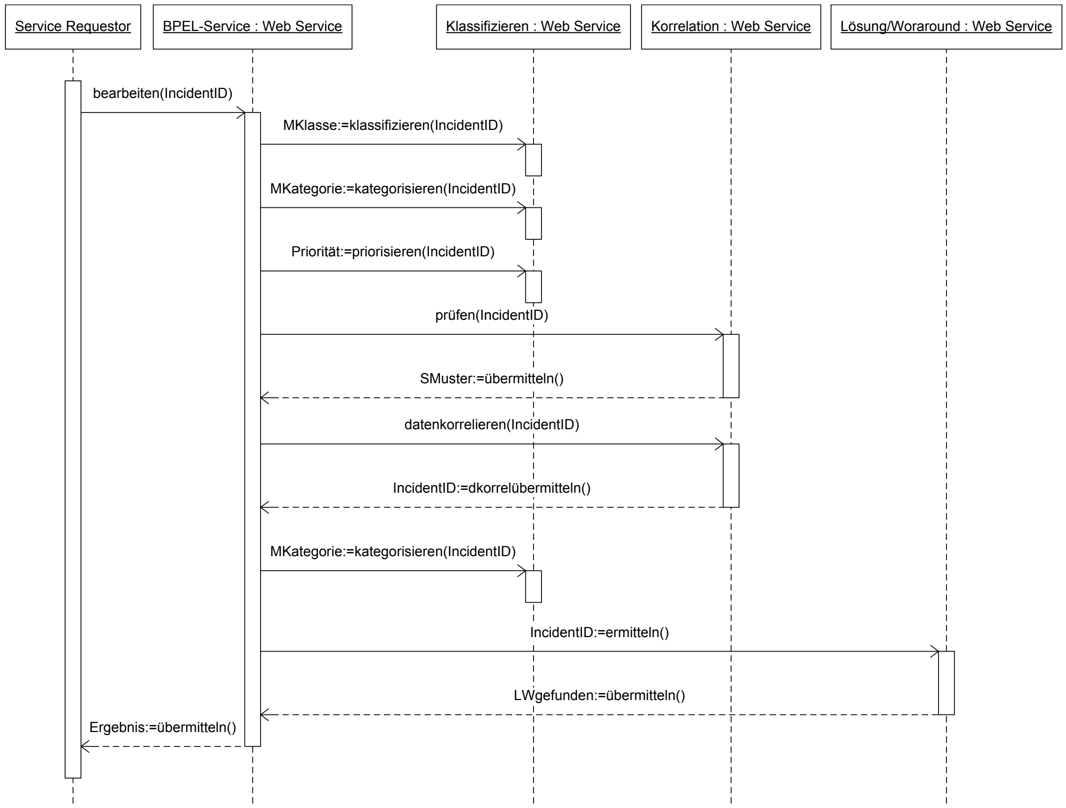


Abbildung 7.20.: Visualisierung der Interaktion

die Abwandlung des Sequenzdiagramms auf die Unterscheidung zwischen synchronen und asynchronen Aufrufen, auf welche in der Abbildung verzichtet wurde. Für die Aufrufe wurden die Nachrichtenpfeile der UML-Spezifikation verwendet.

Die zwischen den Web Services ausgetauschten Nachrichten entsprechen im synchronen BPEL-Fall semantisch der synchronen UML-Modellierung. Im asynchronen Fall sendet ein Web Service eine Nachricht an einen anderen Web Service und wartet dann, dass dieser eine Nachricht zurücksendet (gestrichelte Pfeile), wobei dafür eine weitere Operation angegeben ist.

Da Verzweigungen, die in der BPEL-Beschreibung durch *switch*-Elemente umgesetzt werden, ein Sequenzdiagramm schnell unübersichtlich werden lassen, stellt Abbildung 7.20 eine Interaktionsfolge dar, welche für eine als Incident eingestufte Störungsmeldung durchlaufen wird, für die, nachdem ein archiviertes Störmuster identifiziert wurde, eine Datenkorrelation durchgeführt werden muss, bevor eine (Übergangs-)Lösung ermittelt werden kann. Zu sehen ist ein Service Requestor, der über die Operation „bearbeiten“, der wiederum ein Input vom Typ „IncidentID“ übergeben wird, mit dem BPEL-Service in Kontakt tritt. Der BPEL-Service tritt dann in Interaktion mit dem Web Service „Klassifizieren“, indem er über die drei angebotenen Operationen nacheinander die entsprechenden Funktionalitäten des Web Service nutzt. Der Wert der Variable „MKlasse“ wird innerhalb der BPEL-Spezifikation zur Unterscheidung der weiteren Bearbeitung verwendet. Für die in Abbildung 7.20 dargestellte Interaktion wurde auf die Darstellung dieser Verzweigung sowie der daraus resultierenden Bedingungsangaben im Sequenzdiagramm zwecks Erhalt der Übersichtlichkeit verzichtet.

Nachdem der Incident klassifiziert, kategorisiert und priorisiert ist, wird über die Operation „prüfen“ die Störmusterprüfung durch den Web Service „Korrelation“ eingeleitet. Gemäß der BPEL-Spezifikation wird diesbezüglich asynchron mit dem Web Service kommuniziert. Da der BPEL-Prozess jedoch auf die Antwort wartet, welche ihm über die Operation „übermitteln“ mitgeteilt wird und damit im Grunde einem synchronen Aufruf nach UML entspricht, ist der Aufruf in Abbildung 7.20 nicht als asynchroner UML-Aufruf modelliert worden. Für die visuelle Darstellung der Interaktion wurde angenommen, dass zwar ein Störmuster identifiziert wurde, jedoch nur mit archivierten Datensätzen. Deshalb wird als nächstes eine Datenkorrelation durchgeführt, welche über die Operation „datenkorrelieren“ des Web Services „Korrelation“ in Anspruch genommen wird. Anschließend wird der Incident erneut kategorisiert bevor über den Web Service „Lösung/Workaround“ versucht wird, eine (Übergangs-)Lösung zu ermitteln. Die letzte Kommunikation stellt das Senden einer Nachricht an den Service Requestor über die Operation „übermitteln“ des BPEL-Service dar, welche als Antwort zur asynchron gestellten Anfrage zu Beginn der Interaktionsfolge zu verstehen ist.

In diesem Abschnitt wurde ein BPEL-Prozess definiert, welcher die Orchestrierung der funktionalen Kapseln bzw. der zugehörigen Web Services gemäß der gewünschten Prozesslogik des Beispiels, die in Abbildung 7.18 dargestellt ist, realisiert. Dazu wurden entsprechend Abschnitt 6.4.2 zuerst die Daten- und Kommunikationsbeziehung zu den identifizierten Web Services der SOA definiert, bevor anschließend die Interaktionsbeziehung zwischen diesen modelliert wurde.

7.3. Zusammenfassung

Dieses Kapitel hat sich mit der prototypischen Umsetzung des entwickelten SOA-basierten Konzepts für das Incident Management befasst. Ausgehend von der Prozessanalyse, welche durch Kapitel 4 vorweggenommen wurde, sind die zwei verbleibenden Phasen „SOA-Idee umsetzen“ und „SOA-Implementierung“ für einen Teil des modellierten Incident-Management-Prozesses umgesetzt worden.

Entsprechend der in Kapitel 5 entwickelten Vorgehensweise wurden im ersten Abschnitt Schnittstellen zwischen dem Incident Management und denjenigen ITIL-Prozessen identifiziert, welche dem Incident-Management-Prozess zur Bearbeitung einer Störungsmeldung Daten bereitstellen oder von diesem erhalten. Im Anschluss daran wurde die funktionale Kapselung ausgehend von der EPK-Modellierung des Beispiels abgeleitet und um Aspekte der Orchestrierung, wie bspw. die zusätzliche Interaktion mit anderen Prozessen zur Datenbeschaffung, ergänzt. Die so identifizierten funktionalen Kapseln dienten als Ausgangspunkt für die in Abschnitt 7.2 betrachtete prototypische „SOA-Implementierung“.

Das Vorgehen in Abschnitt 7.2 orientierte sich an der in Kapitel 6 beschriebenen Vorgehensweise. Dementsprechend wurden zuerst die funktionalen Kapseln aus Abschnitt 7.1 den Web Services der SOA zugeordnet und eine Service-Hierarchie für das Beispiel etabliert. Anschließend wurden die Schnittstellen zu den identifizierten Prozess- und Intermediarydiensten mittels WSDL beschrieben und um eine *partnerLinkType*-Definition erweitert. Im letzten Teilabschnitt wurde ein BPEL-Prozess spezifiziert, welcher die Interaktion zwischen den Services beschreibt. Die Interaktionsfolge ist in Anhang D noch einmal zusammengefasst dargestellt.

Mit diesem Kapitel wurde die prototypische Implementierung des im Rahmen dieser Arbeit entwickelten SOA-basierten Konzepts abgeschlossen. Auch wenn das Konzept aufgrund der Komplexität des ITIL Incident-Management-Prozesses nur für einen Teil dieses Prozesses umgesetzt wurde, zeigt dieses Kapitel dennoch, dass die angestrebte Implementierung entsprechend den zuvor in Kapitel 5 und 6 theoretisch beschriebenen Vorgehensweisen möglich ist und für den Rest des modellierten Prozesses leicht ergänzt werden kann.

Das nächste Kapitel wird zum Abschluss der Arbeit das entwickelte Konzept sowie die gewonnenen Erkenntnisse nochmals zusammenfassen. Außerdem soll dem Leser ein Ausblick auf weitere Aspekte gegeben werden, welche ergänzend zu dieser Arbeit betrachtet werden können.

Kapitel 8.

Zusammenfassung und Ausblick

Zum Abschluss der Arbeit wird in diesem Kapitel die Bearbeitung der gestellten Aufgabe zusammenfassend dargestellt. Dabei wird insbesondere nochmals auf die drei Phasen bei der Entwicklung des SOA-basierten Konzepts sowie die dabei gewonnenen Erkenntnisse eingegangen. Anschließend werden verschiedene Aspekte vorgestellt, welche zusätzlich zu dieser Arbeit oder in Bezug auf das Umfeld des Arbeitsthemas von Interesse sein könnten.

8.1. Zusammenfassung

Im Rahmen dieser Arbeit wurde ein SOA-basiertes Konzept zur Unterstützung des ITIL Incident-Management-Prozesses entwickelt und prototypisch implementiert. Bevor jedoch das Konzept vorgestellt wurde, lieferte Kapitel 1 einen einleitenden Überblick über die Aufgabenstellung, das Vorgehen bei der Erarbeitung des Konzepts sowie die allgemeinen Anforderungen und die Gliederung der Arbeit.

Nach dieser Übersicht wurde der Leser in Kapitel 2 in das Konzept einer Service Oriented Architecture (SOA) eingeführt sowie die Merkmale einer SOA betrachtet, zu denen u.a. die lose Kopplung der Dienste, die Programmiersprachen- und Technologieunabhängigkeit, die asynchrone Kommunikation und das Service Publishing gezählt werden. Im Rahmen dieser Einführung wurde außerdem die Idee einer Orchestrierung der Services motiviert und deren Konzept dem der Choreographie gegenübergestellt (siehe Abb. 2.3). Schließlich wurde das Prinzip der Web Services vorgestellt, welches im Rahmen dieser Arbeit für die Implementierung der SOA ausgewählt wurde. Neben den Web Services wurden in diesem Abschnitt noch die Web Service Description Language (WSDL) und die Business Process Execution Language (BPEL) eingeführt, da sie im Verlauf dieser Arbeit für die Beschreibung der Schnittstellen der Web Services bzw. zur Beschreibung der Orchestrierung zum Einsatz kamen. ´

In Kapitel 3 erhielt der Leser einen Einblick in die IT Infrastructure Library (ITIL), deren grundlegende Struktur und Ziele sowie verschiedene Managementbereiche vorgestellt wurden. Da das SOA-basierte Konzept, welches im Rahmen dieser Arbeit entwickelt wurde, den ITIL Incident-Managemet-Prozess unterstützen soll und dieser dem Service Support des ITIL Service-Managements zugeordnet wird, sind insbesondere dessen Teilbereiche vorgestellt worden. Die Service-Delivery-Prozesse wurden nur insoweit betrachtet, als das sie Schnittstellem zum ITIL Incident-Management-Prozess ausweisen.

Nachdem die Prozessanalyse die erste Phase zur Entwicklung des Konzepts darstellt, wurde ausgehend von der ITIL-Prozessbeschreibung für das Incident Management in Kapitel 4 dieser Prozess genauer analysiert. Diese „Aufbereitung“ des Incident-Management-Prozesses war insofern notwendig, da die in der ITIL beschriebene Modellierung des Prozesses noch relativ abstrakt ist, teilweise Lücken aufweist und deshalb für die weitere Bearbeitung erst verfeinert werden musste. Ziel dieser Prozessanalyse war, wie in Abbildung 8.1 dargestellt ist, die Identifikation von Prozessinputs und -outputs, Aktivitäten und der Prozesslogik. Für die in Abschnitt 4.3 herausgearbeiteten Subprozesse des Incident Management wurden Ereignisgesteuerte Prozessketten (EPKs) zur Beschreibung der Aktivitäten und der Prozesslogik verwendet. Durch die in dieser Phase eingesetzte allgemeine Modellierung mittels EPKs, welche den Ausgangspunkt für die weitere Betrachtung bildet, kann gewährleistet werden, dass das Konzept auf unterschiedliche Prozesse angewendet werden kann. Außerdem können Prozesse mittels EPKs auf unterschiedlichen Granularitätsstufen modelliert werden.

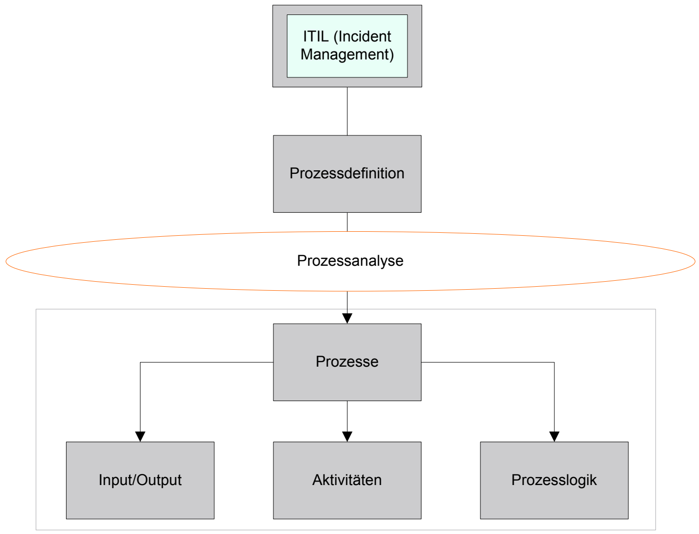


Abbildung 8.1.: Prozessanalyse des Incident Managements (Kapitel 4)

Die zweite Phase in der Entwicklung des Konzepts umfasst die Umsetzung der SOA-Idee, welche in Kapitel 5 betrachtet wurde. In diesem Zusammenhang wurde der Begriff der funktionalen Kapselung eingeführt, welcher innerhalb dieser Arbeit eine wichtige Rolle einnimmt und das Zusammenfassen einzelner Funktionalitäten zu einer logischen Einheit beschreibt, welche dann im Rahmen der dritten Konzeptphase serviceorientiert umgesetzt werden. Wie in Abbildung 8.2 dargestellt ist, bilden die Ergebnisse der Prozessanalyse den Ausgangspunkt für die Ableitung von Schnittstellen, funktionalen Kapseln und Orchesterierung. Diese drei Aspekte beeinflussen sich gegenseitig. Aufgrund der Komplexität des Incident-Management-Prozesses wurden in dieser Phase zunächst nur theoretisch die Muster und Vorgehensweisen für die funktionale Kapselung des Prozesses vorgestellt, welche anschließend durch Beispiele aus dem Incident-Management-Prozess veranschaulicht wurden. Ausgangspunkt für die Identifikation der funktionalen Kapseln war dabei die mittels EPK modellierte Prozessstrukur. Da diese Modellierung unterschiedlich granular und unternehmensspezifisch erfolgen kann und diesbezüglich jedoch in Kapitel 5 keine Annahmen getroffen wurden, ist das Konzeptvorgehen allgemein und auf verschiedenen Ebenen anwendbar und erfüllt somit die zuvor gestellten Anforderungen.

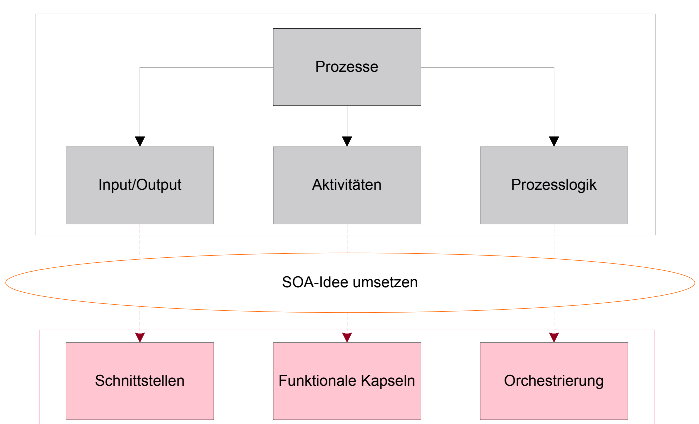


Abbildung 8.2.: Ableitung von Schnittstellen, funktionalen Kapseln und der Orchestrierung (Kapitel 5)

In Kapitel 6 wurde die für diese Arbeit gewählte Implementierung der SOA vorgestellt, was die dritte Phase des Konzepts darstellt. Abbildung 8.3 zeigt graphisch die Abbildung von Schnittstellen, funktionalen Kapseln und der Orchestrierung auf die WSDL-Spezifikation, die Web Services und die BPEL-Beschreibung. Vor der Zuordnung dieser Beschreibung wurde zuerst eine allgemeine Service-Hierarchie etabliert. Anschließend wurden analog zum Kapitel „Konzeptentwicklung“ theoretische Muster und Vorgehensweisen zur WSDL- und BPEL-Spezifikation vorgestellt. Innerhalb der Spezifikation der Web Services wurde zwischen der Beschreibung der Web Service-Schnittstellen in WSDL und der eigentlichen Implementierung unterschieden.

In Abschnitt 6.3.1, welcher sich mit der WSDL-Spezifikation befasste, wurden die vier grundlegenden Bestandteile eines WSDL-Files vorgestellt. Da durch die Spezifikation einer standardisierten Schnittstelle, über welche die Kommunikation mit dem dahinter liegenden Web Service erfolgt, die gewählte Programmiersprache zur Realisierung des Web Services innerhalb der SOA keine Rolle spielt, konnte auch die Anforderung bzgl. der Unabhängigkeit von der Implementierung der Dienste erfüllt werden.

Im Anschluss an die Spezifikation der Web Services wurde in Abschnitt 6.4 die Abbildung der Orchestrierung auf BPEL betrachtet. Dazu wurde zuerst die WSDL-Schnittstellenbeschreibung um ein *partnerLinkType*-Element ergänzt, welches zur Definition von Kommunikationsrollen bezogen auf die Interaktion zwischen dem Web Service dient und später in der BPEL-Prozessbeschreibung in Verbindung mit dem BPEL-*partnerLink*-Element referenziert wird. Anschließend wurde die Abbildung der Orchestrierung auf eine BPEL-Prozessbeschreibung erörtert. Dazu wurde die BPEL-Spezifikation in drei Blöcke aufgespalten, welche die Daten-, Kommunikations- und Interaktionsbeziehung zwischen einem BPEL-Prozess und der WSDL-Schnittstellenbeschreibung eines Web Services betrachteten.

Da die im Rahmen dieser Arbeit gewählte Umsetzung der SOA auf den zuvor identifizierten funktionalen Kapseln und der Orchestrierung aufbaute und diese unabhängig von der WSDL- und BPEL-Spezifikation formuliert wurden, können diese Beschreibungen im Zuge der SOA-Implementierung leicht ausgetauscht werden, ohne dass das Konzept seine Gültigkeit verliert. Des Weiteren wurden die Anforderung, welche sich auf die service- und prozessorientierte Umsetzung des Konzepts bezog, erfüllt, da wiederverwendbare Services innerhalb des Prozesses identifiziert und beschrieben wurden. Dabei wurde jedoch stets der Zusammenhang zwischen den funktionalen Kapseln der realisierenden Services mit beschrieben, so dass die Struktur des Ausgangsprozesses erhalten und durch die BPEL-Modellierung des Prozesses umgesetzt werden konnte.

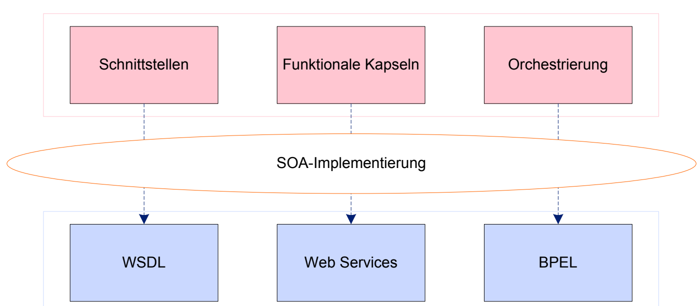


Abbildung 8.3.: Abbildung des Konzepts auf WSDL, Web Services und BPEL (Kapitel 6)

Kapitel 7 widmete sich der prototypischen Umsetzung des entwickelten Konzepts für den ITIL Incident-Management-Prozess. Dabei konnte auf die Umsetzung der ersten Phase verzichtet werden, da die Prozessanalyse für den Incident-Management-Prozess bereits in Kapitel 4 durchgeführt wurde. Entsprechend der Gliederung des Kapitels 5 wurden in Abschnitt 7.1 ausgehend von den Inputs und Outputs zuerst die Schnittstellen zwischen dem Incident Management und anderen ITIL-Managementbereichen identifiziert. Anschließend wurde ein Teil des Incident-Management-Prozesses für die weitere Betrachtung ausgewählt, da die prototypische Implementierung des gesamten Incident-Management-Prozesses aufgrund seiner Komplexität den Rahmen dieser Arbeit gesprengt hätte. Für den implementierten Teil des Incident

Managements wurde dessen funktionale Kapselung identifiziert, welche anschließend in Bezug auf die spätere BPEL-Orchestrierung abgewandelt und ergänzt wurde. Beispielsweise wurden notwendige zusätzliche Interaktionen zur Datenbeschaffung modelliert.

In Abschnitt 7.2 wurden dann die identifizierten funktionalen Kapseln auf Web Services verteilt, in einer Hierarchie angeordnet und einige grundsätzliche Entscheidungen bzgl. der Kommunikation innerhalb dieser Hierarchie getroffen. Anschließend wurden für die Intermediary- und Prozessdienste der Service-Hierarchie WSDL-Schnittstellenbeschreibungen erstellt. Auf Basis dieser WSDL-Spezifikationen wurde dann systematisch ein BPEL-Prozess entwickelt, welcher die Orchestrierung für das ausgewählte Beispiel beschreibt. Durch diese Spezifikation der Orchestrierung in BPEL wurde die prototypische Umsetzung im Rahmen dieser Arbeit abgeschlossen.

8.2. Ausblick

Nachdem im vorherigen Abschnitt Grundlagen, Entwicklung und Umsetzung des Konzepts für den Incident-Management-Prozess nochmals zusammengefasst wurden, befasst sich dieser Abschnitt mit einigen Aspekten, welche ausgehend von dieser Diplomarbeit noch betrachtet werden können. In diesem Zusammenhang soll der Leser einen Ausblick über mögliche weitere Gesichtspunkte erhalten, welche zum thematischen Umfeld dieser Arbeit passen.

In der Prozessanalyse, welche in Kapitel 4 für den Incident Management Prozess durchgeführt wurde, sind die Aktivitäten und die Prozesslogik der Subprozesse mittels EPKs modelliert worden. Alternativ wäre auch der Einsatz der *Business Process Modeling Notation (BPMN)* möglich gewesen (siehe [Bpmn 06] und [Orp 06]). Vergleicht man die beiden Modellierungsarten, so stellt man fest, dass die drei grundlegenden Elemente, Event, Activitiy und Gateway der BPMN sich leicht auf die EPK-Elemente Ereignis, Funktion und Verknüpfung abbilden lassen. Zur Veranschaulichung betrachte man die in Abbildung 8.4 dargestellte EPK, welche einen Teil des Incident-Management-Subprozesses „Annehmen und Erfassen“ zeigt (siehe Abschnitt 4.3.1). Dreht man diese Abbildung um 90° und modelliert sie mittels BPMN, so würde dies etwa Abbildung 8.5 entsprechen. Obwohl diese beiden Abbildungen sich graphisch sehr ähneln, müsste man prüfen, ob die theoretischen Muster, welche im Rahmen der Konzeptentwicklung und -umsetzung in dieser Arbeit betrachtet wurden, auch für einen in BPMN modellierten (Incident-Management-)Prozess anwendbar sind. Außerdem muss überprüft werden, ob alle notwendigen Situationen abgedeckt wurden. Das Konzept müsste dann noch um weitere Muster ergänzt werden, da in BPMN keine Einschränkungen bzgl. der Anzahl der eingehenden bzw. ausgehenden Kanten (*Sequence-Flow-Kanten*) getroffen werden, wie es bei der EPK-Modellierung der Fall ist: Bei der Beschreibung einer Prozesskette mittels EPK werden zur Vervielfältigung bzw. Reduktion der eingehenden bzw. ausgehenden Kanten Verknüpfungsoperatoren verwendet. Im Falle von BPMN müssen die Kanten nicht durch einen Verknüpfungsoperator in einen Zusammenhang gebracht werden. Die Abbildung von BPMN auf einen BPEL-Prozess wird in [Whit 05] beispielhaft betrachtet und müsste ggf. nur ergänzt werden.

Neben dem Austausch der zur Modellierung des Prozesses eingesetzten Notation könnte auch die Implementierung der Service Oriented Architecture ausgetauscht werden. Wie bereits in Kapitel 2 erklärt wurde, stellen Web Services nicht die einzige Möglichkeit dar, SOA-Dienste zu implementieren. Vielmehr kann eine SOA prinzipiell auf jeder dienstbasierten Technologie aufbauen, so etwa auch auf Enterprise Java Beans [Sun 04] oder CORBA [OMG 01b].

Entsprechend den ITIL-Empfehlungen sollen an geeigneten Stellen Tools zur Prozessunterstützung zum Einsatz kommen. Für den Incident-Management-Prozess wurde in Abschnitt 4.4 als Unterstützungstool das Trouble-Ticket-System vorgestellt. Im weiteren Verlauf der Arbeit wurde dieses jedoch nicht explizit in das entwickelte Konzept eingebunden. In diesem Zusammenhang könnte die Integration von vorhandenen Tools in das Konzept betrachtet werden. Innerhalb der Prozesslandschaft stellen die Tools Funktionalitäten zur Verfügung, welche von anderen genutzt werden. Dementsprechend können diese auch funktional gekapselt und durch einen Web Service zur Verfügung gestellt werden. Da ein Tool jedoch andererseits auch eine Art Kapselung darstellt, kann es demnach neben der ersten auch der zweiten Phase des Konzepts zugeordnet werden.

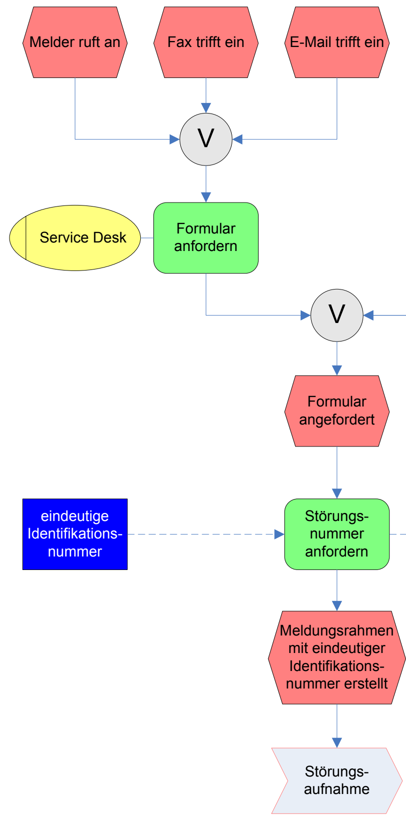


Abbildung 8.4.: Störungsrahmen anlegen, Modellierung als EPK (vgl. Abb. 4.5)

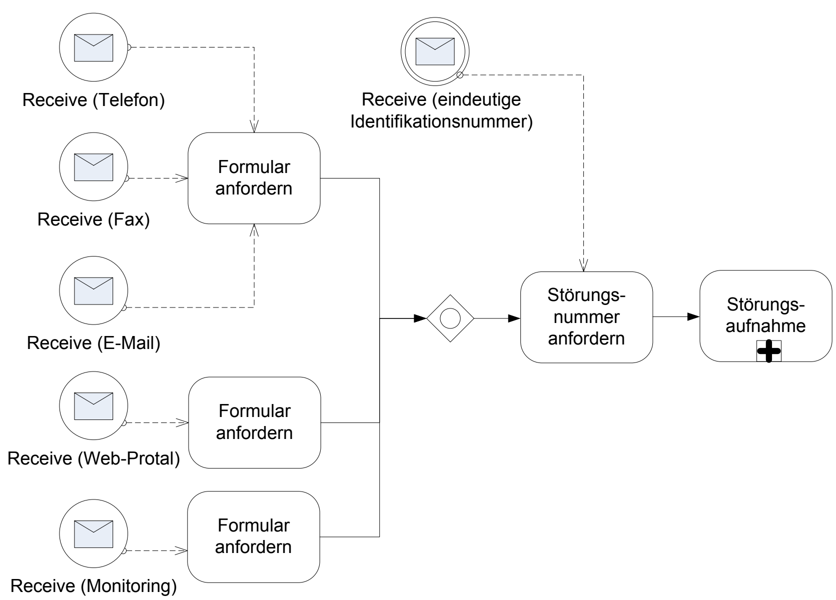


Abbildung 8.5.: Störungsrahmen anlegen, Modellierung in BPMN

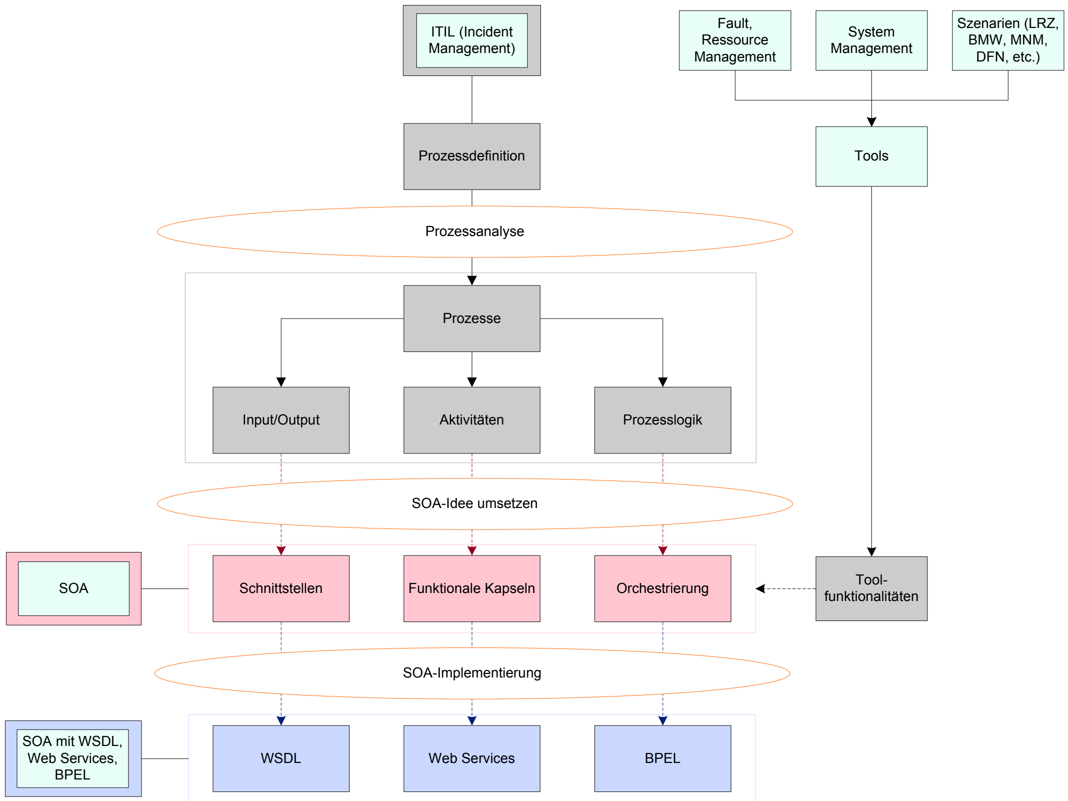


Abbildung 8.6.: Toolintegration

In Abbildung 8.6 ist neben der bisherigen Vorgehensweise die Einordnung der Tools graphisch dargestellt. Als Ausgangspunkt für eine Analyse könnten Tools aus den Bereichen Fault, Ressource bzw. System Management betrachtet werden, welche bspw. im Umfeld des Leibniz-Rechenzentrums (LRZ), der BMW Group, des Munich Network Management Teams (MNM Team) oder des Deutsche Forschungsnetzes (DFN) zu finden sind. Betrachtet man das Tool als funktionale Kapsel, so stellen sich bspw. folgende Fragen: Wie sieht die Schnittstelle dafür aus? Hat das Tool eine einzige Schnittstelle, oder für jede identifizierte Funktionalität eine eigene, entsprechend den einzelnen Web Services? Kann man das Tool als eine Art BPEL-Prozess verstehen oder realisieren, der bereits vorhandene Web Services nutzt, oder implementiert er sie neu? Wenn er die vorhandenen Web Services verwendet, aber nur eine Schnittstelle besitzt, muss dann ein Mapping auf die Schnittstellen der Web Services erfolgen, und wie müsste man dieses realisieren? Wie man anhand dieser wenigen Fragen bereits feststellt, ist das Thema dieser Arbeit in Bezug auf den Einsatz und die Integration vorhandener Tools beliebig erweiterbar.

Als letzten Aspekt wird das automatische Discovery von Diensten im Rahmen dieses Ausblicks kurz vorgestellt. Unter diesem automatischen Discovery verbirgt sich die Idee, zur Laufzeit denjenigen Dienst innerhalb der SOA zu finden, welcher die gewünschte Funktionalität in der gewünschten Qualität bereitstellt. Stellen mehrere Dienste diese Funktionalität zur Verfügung, so könnte z.B. dynamisch vom anfragenden Dienst derjenige ausgewählt werden, welcher momentan am geringsten ausgelastet ist. Andere Kriterien sind aber ebenso denkbar. Im Gegensatz dazu wurde in Abschnitt 6.4.2 fest spezifiziert, welcher Dienst zur Erbringung der gewünschten Funktionalität aufgerufen wird. Allerdings könnte das Konzept dahingehend erweitert werden, dass die jeweiligen *binding*- und *service*-Elemente in der WSDL-Spezifikation erst zur Laufzeit ergänzt werden. Dann stellt sich allerdings die Frage, wie die Bindung an die zuvor definierten *interface*- und deren *operation*-Elementen erfolgt, falls diese nicht der Signatur des Web Service entsprechen, der ausgewählt wurde.

Dieser Abschnitt sollte dem interessierten Leser verdeutlichen, dass sich das behandelte Themengebiet in verschiedenste Richtungen vertiefen lässt und das in dieser Arbeit entwickelte SOA-basierte Konzept eine gute Ausgangsbasis für eine vollständige Umsetzung für den ITIL Incident-Management-Prozess darstellt. Wie man im Verlauf dieser Arbeit feststellt, so ist die prototypische Implementierung eines Teils des modellierten Incident Managements sehr komplex, so dass diesbezüglich einige vereinfachende Annahmen getroffen werden musten. Die getroffenen Annahmen schränken jedoch nicht die allgemeinen Möglichkeiten des Konzepts ein, sondern beziehen sich nur auf das gewählte Maß der beispielhaften Umsetzung im Rahmen dieser Arbeit.

Das im Rahmen dieser Arbeit entwickelte Konzept vereinigt die Bereiche SOA, Web Services und ITIL. Da diesen Bereichen in den nächsten Jahren zunehmend mehr Bedeutung beigemessen werden wird, kann diese Arbeit als Basis für weitere Entwicklung angesehen werden.

Anhang A.

Ereignisgesteuerte Prozessketten

Tabelle A.1 zeigt die Elemente einer (erweiterten) Ereignisgesteuerte Prozesskette[1].

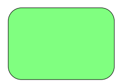
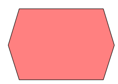
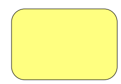
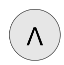
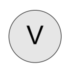
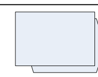
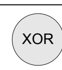
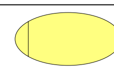
EPK - Elemente				
Grundelement		Darstellung	(erweitertes) Element	Darstellung
Knoten	Funktion		Information	
	Ereignis		Hauptprozess	
Verknüpfungen	Und (AND)		Prozesswegweiser	
	Und/Oder (OR)			
	Entweder/Oder (XOR)		Organisationseinheit	
Kanten	Kontrollfluss	→	Datenfluss	⇢

Tabelle A.1.: EPK - Elemente

Bei den Grundelemente werden in drei Elementtypen unterschieden:

- **Knoten**
  - Funktionen stellen Aktivitäten dar, die durch Ereignisse ausgelöst werden und wieder in Ereignissen resultieren.
  - Ereignisse sind Vorbedingung von Funktionen und können das Resultat von Funktionen sein.
- **Verknüpfungen**
  - Konjunktion (AND): alle Fälle müssen eintreten
  - Disjunktion (OR): ein oder mehrere Fälle sind denkbar
  - Antivalenz (XOR): nur exakt ein Fall darf eintreten
- **Kanten** verbinden die abgebildeten Elemente, wobei zwischen einem Kontroll- und einem Datenfluss unterschieden wird (vgl. Abb. A.1).

[1] siehe auch [Sine 03]

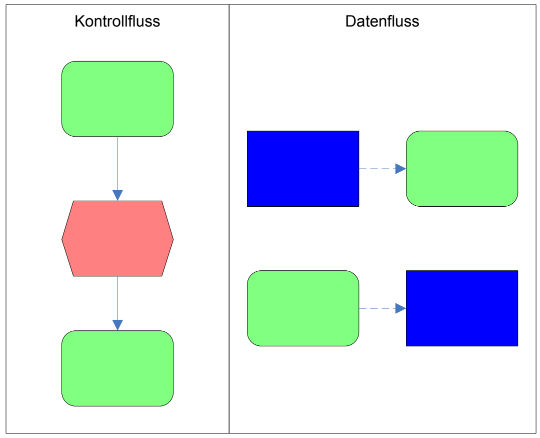


Abbildung A.1.: Beispiele für Kontroll- und Datenfluss

Weitere in Tabelle A.1 dargestellte Elemente eines erweiterten EPKs sind Informationselemente, welche zur Modellierung von Datenein- und ausgaben in Verbindung mit den Datenfluss-Kanten verwendet werden sowie die Hauptprozesselement und Prozesswegweiser, die dazu dienen, ein EPK in Module zu gliedern. Außerdem können die mittels EPK modellierten logischen Abläufe eines Geschäftsprozesses um die Elemente der Organisations- und Leistungsmodellierung erweitert werden.

Kanten verknüpfen abwechselnd Funktionen und Ereignisse miteinander. In der EPK-Spezifikation werden zwei Arten von Kanten zur Modellierung verschiedener Sachverhalte unterschieden, welche in Abbildung A.1 dargestellt sind. Links in Abbildung A.1 ist der Kontrollfluss dargestellt, welche Funktionen und Ereignisse, aber auch Verknüpfungen, Prozesswegweiser oder Hauptprozesse verbindet. Die Organisationseinheiten werden dabei in der Regel über eine leicht modifizierte „Kontrollfluss"-Kante angebunden, wobei dann das Pfeilende weggelassen wird. Neben den gerade erwähnten Elementen existiert noch das Element, welches für Informationen steht, die den Funktionen oder Prozessen als Ein-/Ausgabe zur Verfügung gestellt werden. Dieser Datenfluss ist in Abbildung A.1 rechts abgebildet.

Bei der Modellierung eines so komplexen Prozesses wie des ITIL Incident-Management-Prozesses spielen Verknüpfungen eine zentrale Rolle. Erzeugt eine Funktion mehr als ein Ereignis oder lösen mehrere Ereignisse eine Funktion aus, so werden diese Sachverhalte mittels Verknüpfungsoperatoren modelliert. Verzweigt sich der Prozessstrang oder vereinigen sich einzelne Teilstänge wieder zu einem Strang, so ist ein Verknüpfungsoperator zu verwenden. Je nachdem, ob der Verknüpfungsoperator dazu verwendet wird, den Prozessstrang in mehrere Ereignisse oder Funktionen aufzuspalten oder diesen wieder zu vereinigen, spricht man im ersten Fall von einer Ereignis- und im zweiten von einer Funktionsverknüpfung. Für die drei verschiedenen Verknüpfungen (AND, OR, XOR) sind die erlaubten Aufspaltungen oder Vereinigungen in den Abbildungen A.2 und A.3 aufgezeigt, wobei jeweils beispielhaft zwei Ereignisse bzw. Funktionen vereinigt und aufgespalten werden.

Bei den dargestellten Verknüpfungen sind jedoch gewisse Einschränkungen zu beachten. Da Ereignisse keine „Intelligenz" besitzen, welche es ihnen erlauben würde, aus einer Menge möglicher Funktionen die passende auszuwählen, kann ein Ereignis nicht durch eine OR-/XOR-Verknüpfung mit mehreren Funktionen verbunden werden (siehe erlaubte Funktionsverknüpfungen in Abb. A.3).

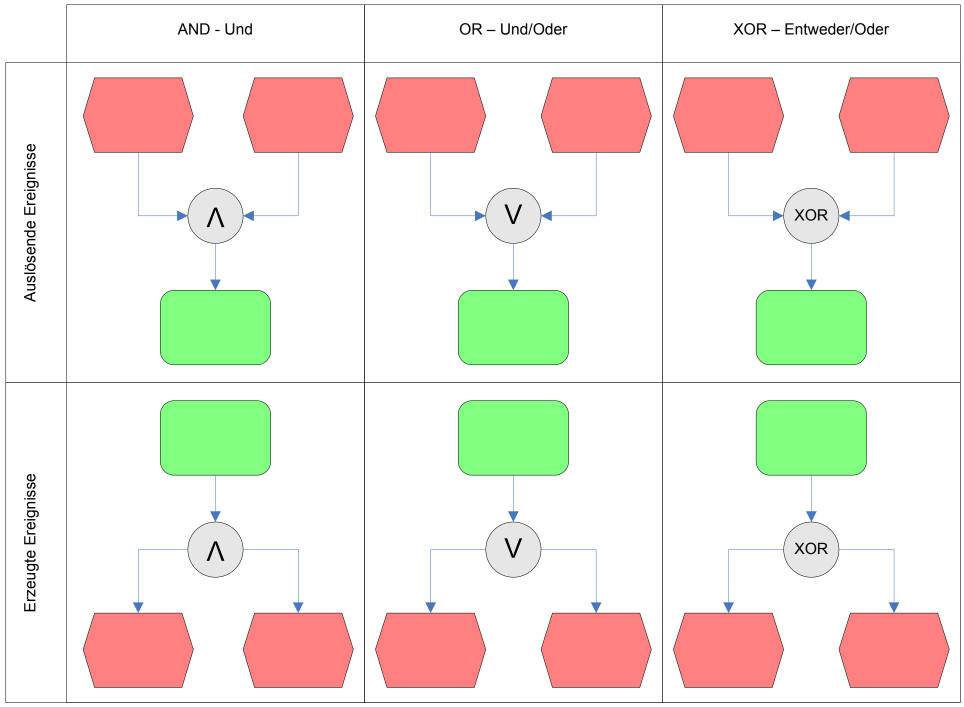


Abbildung A.2.: Ereignisverknüpfung

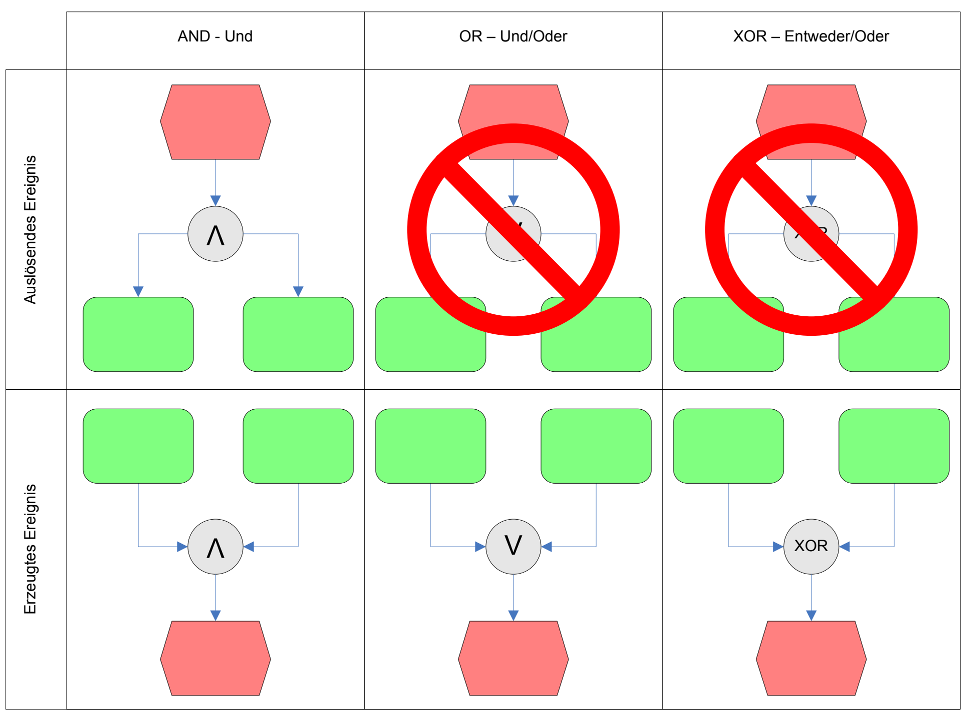


Abbildung A.3.: Funktionsverknüpfung

Anhang B.

Web Service Description Language

Die Web Service Description Language (WSDL) wird zur Beschreibung der Schnittstellen von Web Services verwendet. Dieses Kapitel enthält Auszüge der WSDL-2.0-Spezifikation. Für eine detailliertere Betrachtung siehe u.a. [W3C 05b]. Da in vielen vorhandenen Quellen, z.B. in [ZTP 03], noch mit der WSDL-Version 1.1 gearbeitet wird, soll Tabelle B.1 zunächst einen kurzen Überblick über wichtige Versionsunterschiede geben:

Änderungen von Bezeichnern	
WSDL Version 1.1	WSDL Version 2.0
definitions	description
message	(*nicht mehr vorhanden*)
portType	interface
port	endpoint
Attribute des operation-input-Tags:	
message	element
name	messageLabel
Attribute des binding-Tags:	
type	interface

Tabelle B.1.: WSDL - Unterschiede zwischen den Versionen

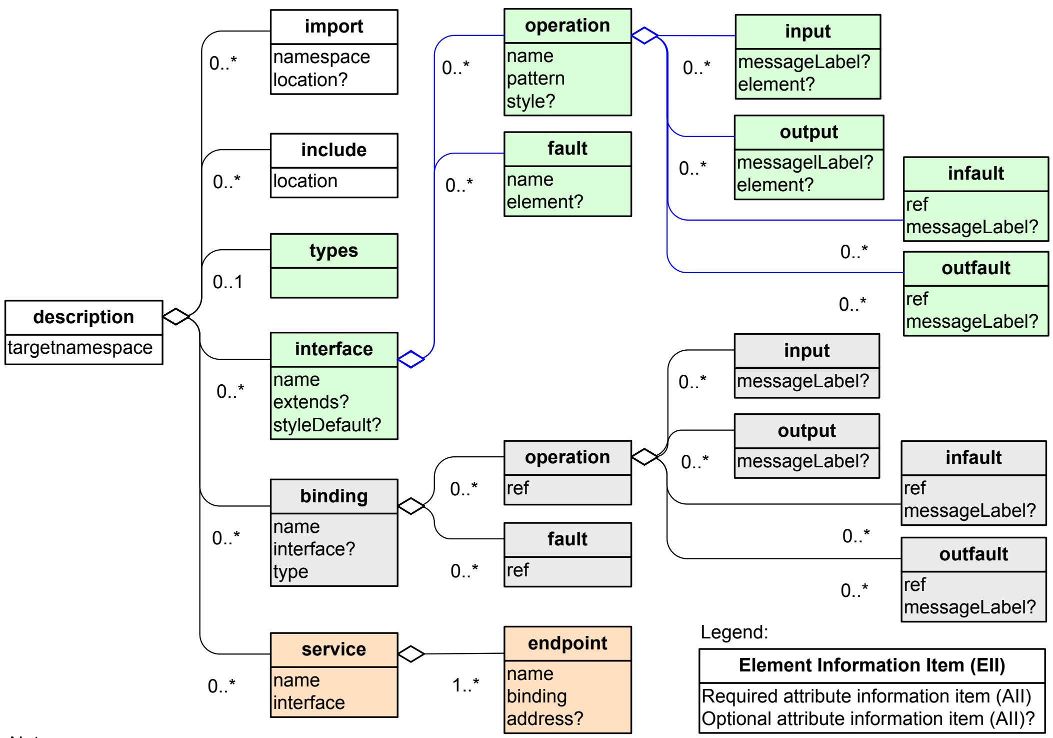
Die allgemeine Grammatik eines WSDL-Dokuments nach Version 2.0 sieht folgendermaßen aus:

```
<description
  ...
  targetNamespace="xs:anyURI">
  <documentation />?

  <import namespace="xs:anyURI" location="xs:anyURI"? >
    <documentation />*
  </import>*

  <include location="xs:anyURI">
    <documentation />*
  </include>*

  <types>
    <documentation />*
      [ <xs:import namespace="xs:anyURI" schemaLocation="xs:anyURI"? /> |
        <xs:schema targetNamespace="xs:anyURI" /> |
        other extension elements ]*
  </types>?

  <interface name="xs:NCName" extends="list of xs:QName"?
        styleDefault="list of xs:anyURI"? >
    <documentation />*
```

```
    <fault name="xs:NCName" element="xs:QName"? >
      <documentation />*
      <feature ... />*
      <property ... />*
    </fault>*

    <operation name="xs:NCName" pattern="xs:anyURI" style="list of xs:anyURI"? >
      <documentation />*

      <input messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </input>*

      <output messageLabel="xs:NCName"? element="union of xs:QName, xs:token"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </output>*

      <infault ref="xs:QName" messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </infault>*

      <outfault ref="xs:QName" messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </outfault>*

      <feature ... />*
      <property ... />*
    </operation>*

    <feature ref="xs:anyURI" required="xs:boolean"? >
      <documentation />*
    </feature>*

    <property ref="xs:anyURI">
      <documentation />*
      <value> xs:anyType </value>?
      <constraint> xs:QName </constraint>?
    </property>*
  </interface>*

  <binding name="xs:NCName" interface="xs:QName"? type="xs:anyURI">
    <documentation />*

    <fault ref="xs:QName">
      <documentation />*
      <feature ... />*
      <property ... />*
    </fault>*
```

```
    <operation ref="xs:QName">
      <documentation />*

      <input messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </input>*

      <output messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </output>*

      <infault ref="xs:QName" messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </infault>*

      <outfault ref="xs:QName" messageLabel="xs:NCName"? >
        <documentation />*
        <feature ... />*
        <property ... />*
      </outfault>*

      <feature ... />*
      <property ... />*
    </operation>*

    <feature ... />*
    <property ... />*
  </binding>*

  <service name="xs:NCName" interface="xs:QName">
    <documentation />*

    <endpoint name="xs:NCName" binding="xs:QName" address="xs:anyURI"? >
      <documentation />*
      <feature ... />*
      <property ... />*
    </endpoint>+

    <feature ... />*
    <property ... />*
  </service>*
</description>
```

Das gerade vorgestellte allgemeine Schema eines WSDL-Dokuments ist in Abbildung B.1 in Form eines Klassendiagramms graphisch dargestellt, welches der WSDL-2.0-Primer-Spezifikation [W3C 05a] entnommen ist und deswegen nicht näher erklärt wird.



Note:
- All EIIs except <description>, <import>, <include> and <types> may have <feature> and/or <property> as children.
- All EIIs may have <documentation> as first child

Abbildung B.1.: WSDL-2.0-Infoset-Diagramm (Quelle: [W3C 05a])

Anhang C.

Business Process Execution Language

Die Business Process Execution Language (BPEL) oder auch Business Process Execution Language for Web Services (BPEL4WS) wird zur Orchestrierung der mittels WSDL beschriebenen Web Services verwendet. Dieses Kapitel enthält Auszüge der BPEL-1.1-Spezifikation. Für eine detailliertere Betrachtung siehe u.a. [IBM 03]. Bevor einige wichtige Elemente schematisch vorgestellt werden, zeigt Tabelle C.1 einige Unterschiede zwischen den Bezeichnern in den BPEL-Versionen 1.0 und 1.1.

Änderungen von Bezeichnern	
BPEL4WS Version 1.0	BPEL4WS Version 1.1
serviceLinks	partnerLinks
serviceLinkTypes	partnerLinkTypes
serviceReferences	endpointReferences
containers	variables

Tabelle C.1.: BPEL - Unterschiede zwischen den Versionen

Die Beschreibung eines BPEL-Prozesses erfolgt innerhalb eines umgebenden *process*-Elements, dessen allgemeine Grammatik gemäß BPEL-Version 1.1 folgendermaßen aussieht:

```
<process
  name="ncname"
  targetNamespace="uri"
  queryLanguage="anyURI"?
  expressionLanguage="anyURI"?
  suppressJoinFailure="yes|no"?
  enableInstanceCompensation="yes|no"?
  abstractProcess="yes|no"?
  xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/">

  <partnerLinks>
  <!-- Note: At least one role must be specified. -->
    <partnerLink name="ncname" partnerLinkType="qname"
              myRole="ncname"? partnerRole="ncname"?>
    </partnerLink>+
  </partnerLinks>?

  <partners>
    <partner name="ncname">
      <partnerLink name="ncname" />+
    </partner>+
  </partners>?

  <variables>
    <variable name="ncname"
      messageType="qname"? type="qname"? element="qname"?/>+
  </variables>?
```

```
  <correlationSets>
    <correlationSet name="ncname" properties="qname-list" />+
  </correlationSets>?

  <faultHandlers>
  <!-- Note: There must be at least one fault handler or default. -->
    <catch faultName="qname"? faultVariable="ncname"?>
      <!-- activity -->
    </catch>*
    <catchAll>
      <!-- activity -->
    </catchAll>?
  </faultHandlers>?

  <compensationHandler>
    <!-- activity -->
  </compensationHandler>?

  <eventHandlers>
  <!-- Note: There must be at least one onMessage or onAlarm handler. -->
    <onMessage partnerLink="ncname" portType="qname"
        operation="ncname" variable="ncname"?>
      <correlations>
        <correlation set="ncname" initiate="yes|no"?>+
      <correlations>?
      <!-- activity -->
    </onMessage>
    <onAlarm for="duration-expr"? until="deadline-expr"?>
      <!-- activity -->
    </onAlarm>*
  </eventHandlers>?

  <!-- activity -->
</process>
```

`<!-- activity -->` kann für jedes der folgenden Tags stehen:

- `<receive>` (*)
- `<reply>` (*)
- `<invoke>` (*)
- `<sequence>` (*)
- `<switch>` (*)
- `<while>` (*)
- `<pick>` (*)
- `<flow>` (*)
- `<assign>` (*)
- `<throw>`
- `<terminate>`
- `<wait>`
- `<empty>`
- `<scope>`
- `<compensate>`

Die Syntax der mit (*) gekennzeichneten Aktivitäten wird im Anschluss an die Definition der Standardattribute und -elemente genauer betrachtet.

Standardattribute (`standard-attributes`) für jede Aktivität:

```
name="ncname"?
joinCondition="bool-expr"?
suppressJoinFailure="yes|no"?
```

Standardelemente (`<!-- standard-elements -->`) für jede Aktivität:

```
<source linkName="ncname" transitionCondition="bool-expr"?/>*
<target linkName="ncname" />*
```

Nachfolgend sind die allgemeinen Grammatiken wichtiger Aktivitäten aufgelistet:

Receive:

```
<receive
  partnerLink="ncname"
  portType="qname"
  operation="ncname"
  variable="ncname"?
  createInstance="yes|no"?
  standard-attributes>
  <!-- standard-elements -->
  <correlations>
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>?
</receive>
```

Reply:

```
<reply
  partnerLink="ncname"
  portType="qname"
  operation="ncname"
  variable="ncname"?
  faultName="qname"?
  standard-attributes>
  <!-- standard-elements -->
  <correlations>
    <correlation set="ncname" initiate="yes|no"?>+
  </correlations>?
</reply>
```

Invoke:

```
<invoke
  partnerLink="ncname"
  portType="qname"
  operation="ncname"
  inputvariable="ncname"?
  outputvariable="ncname"?
  standard-attributes>
  <!-- standard-elements -->
  <correlations>
    <correlation set="ncname" initiate="yes|no"? pattern="in|out|out-in" />+
  </correlations>?
  <catch faultName="qname" faultvariable="ncname"?>
    <!-- activity -->
  </catch>*
  <catchAll>
    <!-- activity -->
  </catchAll>?
  <compensationHandler>
    <!-- activity -->
  </compensationHandler>?
</invoke>
```

Sequence:

```
<sequence standard-attributes>
  <!-- standard-elements -->
  <!-- activity -->+
</sequence>
```

Switch:

```
<switch standard-attributes>
  <!-- standard-elements -->

  <case condition="bool-expr">
    <!-- activity -->
  </case>+
  <otherwise>
    <!-- activity -->
  </otherwise>?
</switch>
```

While:

```
<while
  condition="bool-expr"
  standard-attributes>
  <!-- standard-elements -->
  <!-- activity -->
</while>
```

Pick:

```
<pick
  createInstance="yes|no"?
  standard-attributes>

  <!-- standard-elements -->
  <onMessage
    partnerLink="ncname"
    portType="qname"
    operation="ncname"
    variable= "ncname"?>+
    <correlations>
      <correlation set="ncname" initiate="yes|no"?>+
    </correlations>?
    <!-- activity -->
  </onMessage>+
  <onAlarm (for="duration-expr" | until="deadline-expr")>
    <!-- activity -->
  </onAlarm>*
</pick>
```

Flow:

```
<flow standard-attributes>
  <!-- standard-elements -->
  <links>
    <link name="ncname">+
  </links>?
  <!-- activity -->+
</flow>
```

Assign:

```
<assign standard-attributes>
  <!-- standard-elements -->
  <copy>
    <!-- from -->
    <!-- to -->
  </copy>+
</assign>
```

`<!-- from -->` ist eines der folgenden Konstrukte:

```
<from variable="ncname" part="ncname"?/>
<from partnerLink="ncname" endpointReference="myRole|partnerRole" />
<from variable="ncname" property="qname" />
<from expression="general-expr" />
<from> ... literal value ... </from>
```

`<!-- to -->` ist eines der folgenden Konstrukte:

```
<to variable="ncname" part="ncname"?/>
<to partnerLink="ncname" />
<to variable="ncname" property="qname" />
```

Da mittels der *PartnerLink*-Elemente die Verbindung mit den beteiligten Web Services hergestellt wird, soll deren Syntax nochmal genauer betrachtet werden.

PartnerLinks: (Definition im BPEL-File)

```
<partnerLinks>
  <partnerLink
    name="ncname"
    partnerLinkType="qname"
    myRole="ncname"?
    partnerRole="ncname"?>
  </partnerLink>+
</partnerLinks>
```

PartnerLinkType: (Definition im WSDL-File)

```
<description
  name="ncname"
  targetNamespace="uri"
  xmlns="http://schemas.xmlsoap.org/wsdl/"
  ...
  xmlns:plnk="http://schemas.xmlsoap.org/ws/2003/05/partner-link/">

  ...
  <plnk:partnerLinkType name="ncname">
    <plnk:role name="ncname">
      <plnk:portType name="qname" />
    </plnk:role>
    <plnk:role name="ncname">
      <plnk:portType name="qname" />
    </plnk:role>?
  </plnk:partnerLinkType>
  ...
</description>
```

Alternativ zum *faultHandlers*- bzw. *compensationHandler*-Element kann die Behandlung auch direkt in Umgebung der Aktivität eingeschlossen werden, wie hier am Beispiel der *invoke*-Aktivität dargestellt.

```
<invoke
  partnerLink="ncname"
  portType="qname"
  operation="ncname"
  inputVariable="ncname"?
  outputVariable="ncname"?
  standard-attributes>
  <!-- standard-elements -->
  <correlations>
    <correlation set="ncname" initiate="yes|no"? pattern="in|out|out-in" />+
  </correlations>?
  <catch faultName="qname" faultVariable="ncname"?>
    <!-- activity -->
  </catch>*
  <catchAll>
    <!-- activity -->
  </catchAll>?
  <compensationHandler>
    <!-- activity -->
  </compensationHandler>?
</invoke>
```

Anhang D.

Interaktion des BPEL-Prozesses

In diesem Kapitel ist die Interaktionsfolge, welche im Rahmen der prototypischen Implementierung in Abschnitt 7.2.3 gemäß Abbildung 7.18 beschrieben wurde, zusammenfassend dargestellt.

```
<sequence name="interaktionsbeispiel">

 <!-- Anfrage annehmen -->
 <receive partnerLink="Client"
  portType="bp:WSHHin" operation="bearbeiten"
  variable="IncidentID" createInstance="yes" />

 <!-- Web Service A Kapsel "Störungsmeldung klassifizieren" -->
 <invoke partnerLink="WSAPartnerLink"
  portType="wsA:WSAInterface" operation="klassifizieren"
  inputVariable="IncidentID" outputVariable="MKlasse" />

 <!-- zwischen Incident und User Service Request unterscheiden -->
 <switch name="verknüpfung1">
  <case condition="bpws:getVariableData('MKlasse') = 'User Service Request'">
   <!-- User Service Request bearbeiten -->
   <sequence name="usrbearbeiten">
    <!-- Web Service D Kapsel "User Request bearbeiten" -->
    <invoke partnerLink="WSDPartnerLink"
     portType="wsD:WSDHin" operation="USRbearbeiten"
     inputVariable="IncidentID" />
    <receive partnerLink="WSDPartnerLink"
     portType="wsD:WSDRück" operation="übermitteln"
     variable="ErgebnisUSR" />

    <assign>
     <copy>
      <from variable="ErgebnisUSR" />
      <to variable="Ergebnis" />
     </copy>
    </assign>

    <!-- Meldung zurücksenden -->
    <invoke partnerLink="Client"
     portType="bp:WSHRück" operation="übermitteln"
     inputVariable="Ergebnis" />
   </sequence>
  </case>

  <otherwise>
   <!-- Incident/Standard-Incident bearbeiten -->
   <sequence name="bearbeiten">
```

```
<!-- Web Service A Kapsel "Incident kategorisieren" -->
<invoke partnerLink="WSAPartnerLink"
 portType="wsA:WSAInterface" operation="kategorisieren"
 inputVariable="IncidentID" outputVariable="MKategorie" />

<!-- Web Service A Kapsel "Priorität ermitteln" -->
<invoke partnerLink="WSAPartnerLink"
 portType="wsA:WSAInterface" operation="priorisieren"
 inputVariable="IncidentID" outputVariable="Priorität" />

<!-- zwischen Standard-Incident und Incident unterscheiden -->
<switch name="verknüpfung2">
 <case condition="bpws:getVariableData('MKlasse') = 'Standard-Incident'">
  <!-- Standard-Incident bearbeiten -->
 </case>

 <otherwise>
  <!-- Incident bearbeiten -->
  <sequence name="incidentbearbeiten">
   <!-- Web Service B Kapsel "Störmuster prüfen" -->
   <invoke partnerLink="WSBPartnerLink"
    portType="wsB:WSBHin" operation="prüfen"
    inputVariable="IncidentID" />
   <receive partnerLink="WSBPartnerLink"
    portType="wsB:WSBRück" operation="übermitteln"
    variable="SMuster" />

   <switch name="verknüpfung3">
    <case condition=
     "count(bpws:getVariableData('SMuster','/*')) >= 1">
     <!-- Störungsmuster identifiziert -->
     <!-- Datenvariable füllen -->
     <assign>
      <copy>
       <from variable="SMuster" />
       <to variable="IDSMuster" part="Störmuster" />
      </copy>
      <copy>
       <from variable="IncidnetID" />
       <to variable="IDSMuster" part="IncidentID" />
      </copy>
     </assign>

     <!-- Störungsmuster identifiziert -->
     <switch name="verknüpfung7">
      <case condition=
       "count(bpws:getVariableData('SMuster','/Aktuell')) = 1">
       <!-- es wurde mind. ein aktuelles Störmuster erkannt -->
       <!-- Web Service B Kapsel "Eventkorrelation durchführen" -->
       <invoke partnerLink="WSBPartnerLink"
        portType="wsB:WSBHin" operation="eventkorrelieren"
        inputVariable="IDSMuster" />
       <receive partnerLink="WSBPartnerLink"
        portType="wsB:WSBRück" operation="ekorrelübermitteln"
        variable="IncidentID" />
      </case>
```

```
      <case condition=
       "count(bpws:getVariableData('SMuster','/Archiviert')) = 1">
       <!-- es wurde kein aktuelles Störmuster erkannt -->
       <!-- Web Service B Kapsel "Datenkorrelation durchführen" -->
       <invoke partnerLink="WSBPartnerLink"
        portType="wsB:WSBHin" operation="datenkorrelieren"
        inputVariable="IDSMuster" />
       <receive partnerLink="WSBPartnerLink"
        portType="wsB:WSBRück" operation="dkorrelübermitteln"
        variable="IncidentID" />
      </case>
     </switch>

     <!-- Web Service A Kapsel "Incident kategorisieren" -->
     <invoke partnerLink="WSAPartnerLink"
      portType="wsA:WSAInterface" operation="kategorisieren"
      inputVariable="IncidentID" outputVariable="MKategorie" />
    </case>
   </switch>

   <!-- Lösung/Workaround ermitteln -->
   <!-- Web Service C Kapsel "Lösung/Workaround ermitteln" -->
   <invoke partnerLink="WSCPartnerLink"
    portType="wsC:WSCHin" operation="ermitteln"
    inputVariable="IncidentID" />
   <receive partnerLink="WSCPartnerLink"
    portType="wsC:WSCRück" operation="übermitteln"
    variable="LWgefunden" />

  </sequence>
 </otherwise>
</switch>

<switch name="verknüpfung6">
 <case condition="bpws:getVariableData('LWgefunden')">
  <!-- es wurde eine (Übergangs-)Lösung gefunden -->
  <!-- "Beheben und Wiederherstellen" -->
  <!-- Ergebnisvariable füllen -->
  <assign>
   <copy>
    <from expression="string('Beheben und Wiederherstellen einleiten.')" />
    <to variable="Ergebnis" />
   </copy>
  </assign>
 </case>

 <otherwise>
  <!-- es wurde keine (Übergangs-)Lösung gefunden  -->
  <!-- "Zeitfenster erweitern" und dann "Analyse und Diagnose" -->
  <!-- Ergebnisvariable füllen -->
  <assign>
   <copy>
    <from expression="string('Analyse und Diagnose einleiten.')" />
    <to variable="Ergebnis" />
   </copy>
  </assign>
 </otherwise>
</switch>
```

```
    <!-- Meldung zurücksenden -->
    <invoke partnerLink="Client"
     portType="bp:WSHRück" operation="übermitteln"
     inputVariable="Ergebnis" />

   </sequence>
  </otherwise>
 </switch>
</sequence>
```

Anhang E.

Abkürzungsverzeichnis

ACD	Automatic Call Distribution
BPEL	Business Process Execution Language
BPEL4WS	Business Process Execution Language for Web Services
BPMN	Business Process Modeling Notation
CAB	Change Advisory Board
CI	Configuration Item
CMDB	Configuration Management Database
CORBA	Common Object Request Broker Architecture
CTI	Computer Telephony Integration
EPK	Ereignisgesteuerte Prozesskette
eEPK	erweiterte Ereignisgesteuerte Prozesskette
FSC	Forward Schedule of Change
HTTP	Hypertext Transfer Protocol
IA	Intelligent Assistant
ICT	Information and Communications Technology
ICTIM	ICT Infrastructure Management
IR	Incident Record
IT	Information Technology
ITIL	IT Infrastructure Library (Definition durch OGC)
ITSCM	IT Service Continuity Management
ITSM	IT Service Management
ITSMF	IT Service Management Forum
KDB	Known-Error-Datenbank
OGC	Office of Government Commerce
OMG	Object Management Group
RFC	Request for Change
SLM	Service Level Management
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol (Definition durch W3C)
SPOC	Single Point Of Contact (siehe Service Desk)
TT	Trouble Ticket
TTS	Trouble-Ticket-System
UDDI	Univeral Description, Discovery and Integration (Definition durch OASIS)
WSIL	Web Service Inspector Language
WSDL	Web Service Description Language
XML	Extensible Markup Language

Literaturverzeichnis

[Ahkb 02] VAN DER AALST, W.M.P., A.H.M. TER HOFSTEDE, B. KIEPUSZEWSKI und A.P. BARROS: *Workflow Patterns*. 2002, http://www.workflowpatterns.com .

[Bpmn 06] OMG: *BPMN 1.0: OMG Final Adopted Specification*. February 2006, http://www.bpmn.org/Documents/OMG Final Adopted BPMN 1-0 Spec 06-02-01.pdf .

[Bren 02] BRENNER, M.: *Erstellung eines Kriterienkatalogs zur Beurteilung des Anwender Supports in der BMW Group*. Diplomarbeit, 2002.

[COY 02] COYLE, F. P.: *XML, Web Services, and the Data Revolution*. Addison Wesley, ISBN 0-201-77641-3, 2002.

[DJMZ 05] DOSTAL, W., M. JECKLE, I. MELZER und B. ZENGLER: *Service-orientierte Architecturen mit Web Services: Konzepte - Standards - Praxis*. Elsevier-Spektrum Akademischer Verlag, ISBN 3-8274-1457-1, 2005.

[ERL 04] ERL, T.: *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*. Pearson Education, ISBN 0-13-142898-5, 2004, http://www.soabooks.com .

[Fran 05] FRANKE, S.: *Web Services: Dienste für die Allgemeinheit*. Windows IT Pro Nr.10, 2005.

[HAN 99] HEGERING, H.-G., S. ABECK und B. NEUMAIR: *Integriertes Management vernetzter Systeme: Konzepte, Architekturen und deren betrieblicher Einsatz*. dpunkt-Verlag, ISBN 3-932588-16-9, 1999, http://www.dpunkt.de/produkte/management.html .

[Hao 03] HAO HE: *What Is Service-Oriented Architecture*. 2003, http://www.xml.com/pub/a/ws/2003/09/30/soa.html .

[IBM 03] IBM: *Business Process Excecution Language for Web Services: Version 1.1*. 5 May 2003, http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/ .

[ITS 02] IT SERVICE MANAGEMENT FORUM GERMANY (ITSMF): *IT Service Management, eine Einführung*. Van Haren Publishing, ISBN 90-806713-5-5, 2002.

[JMS 04] JURIC, M. B., MATHEW B. und SARANG P.: *Business Process Execution Language for Web Services*. Packt Publishing, ISBN 1-904811-18-3, 2004.

[KAY 03] KAYE, D.: *Loosley Coupled, The Missing Pieces of Web Services*. RDS Press, ISBN 1-881378-24-1, Marin Country, California, 2003.

[KBS 05] KRAFZIG, D., K. BANKE und D. SLAMA: *Enterprise SOA – Service-Oriented Architecture Best Practices*. Pearson, ISBN 0-13-146575-9, 2005.

[Kraf 05] KRAFZIG, D.: *Benefits of SOA*. 6.2.2005, http://www.enterprise-soa.com .

[Küst 05] KÜSTER, U.: *WSBPEL - Web Service Business Execution Language*. 22.11.2005, http://hnsp.inf-bb.uni-jena.de/professur/files_WS2005/oberseminar/20051122WSBPEL.pdf .

[Lehn 04] LEHNER, S.: *ICT Infrastruktur Management: Deployment und Technical Support*. 02.2004, http://www.nm.ifi.lmu.de/teaching/LMU/Seminare/2003ws/itiletom/ausarbeitungen/itil_4_ausarbeitung.pdf . Seminararbeit.

[Marc 04] MARCU, P.: *ICT Infrastructure Management: Design und Planning*. 02.2004, http://www.nm.ifi.lmu.de/teaching/LMU/Seminare/2003ws/itiletom/ausarbeitungen/itil_3_ausarbeitung.pdf . Seminararbeit.

[Muni 05] MUNICH INSTITUT FOR IT SERVICE MANAGEMENT: *ITIL Foundation Training*. 2005, http://www.mitsm.de .

[NEW 02] NEWCOMER, E.: *Understanding Web Services: XML, WSDL, SOAP, and UDDI*. Addison Wesley, ISBN 0-201-75081-3, 2002.

[OASI 05a] OASIS: *Reference Model for Service Oriented Architectures - Woking Draft 11*. 15. December 2005, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=soa-rm .

[OASI 05b] OASIS: *Web Services References Status*. 2005, http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsqm .

[OGC 00] OFFICE OF GOVERNMENT COMMERCE (OGC): *Service Support: ITIL - The key to Managing IT Services*. The Stationery Office, ISBN 0-11-330015-8, London, 2000.

[OGC 01] OFFICE OF GOVERNMENT COMMERCE (OGC): *Service Delivery: ITIL - The key to Managing IT Services*. The Stationery Office, ISBN 0-11-330017-4, London, 2001.

[OGC 02a] OFFICE OF GOVERNMENT COMMERCE (OGC): *Application Management*. The Stationery Office, ISBN 0-11-330866-3, London, 2002.

[OGC 02b] OFFICE OF GOVERNMENT COMMERCE (OGC): *ICT Infrastructure Management: ITIL - The key to Managing IT Services*. The Stationery Office, ISBN 0-11-330865-5, London, 2002.

[OGC 02c] OFFICE OF GOVERNMENT COMMERCE (OGC): *Planning to Implement Service Management*. The Stationery Office, ISBN 0-11-330877-9, London, 2002.

[OGC 02d] OFFICE OF GOVERNMENT COMMERCE (OGC): *Security Management*. The Stationery Office, ISBN 0-11-330014-X, London, 2002.

[OLB 02] OLBRICH, A.: *ITIL kompakt und verständlich*. Vieweg, ISBN 3-528-15892-1, 2002.

[OMG a] OMG: *Catalog of OMG CORBA®/IIOP® Specifications*. , http://www.omg.org/technology/documents/corba_spec_catalog.htm .

[OMG b] OMG: *CORBA® BASICS*. , http://www.omg.org/gettingstarted/corbafaq.htm .

[OMG 01a] OMG: *Unified Modeling Language Specification, Version 1.4*. 2001, http://www.omg.org/cgi-bin/doc?formal/01-09-67.pdf .

[OMG 01b] OMG: *CORBA - The Common Object Request Broker: Architecture and Specification v2.4.2*. Febuary 2001.

[Orp 06] OWEN, M., J. RAJ und POPKIN SOFTWARE: *BPMN and Business Process Management*. September 2003, http://www.bpmn.org/Documents/6AD5D16960.BPMN and BPM.pdf .

[Pleg 05] PLEGER, B.: *Evaluation von Werkzeugen zur Unterstützung der ITIL Service Management Prozesse*. Diplomarbeit, 2005.

[Quin 05] QUINN, A.: *JavaBeans^{TM} Tutorial - Trail: JavaBeans*. 2005, http://java.sun.com/docs/books/tutorial/javabeans/index.html .

[Sage 05] SAGER, F.: *Konzeptentwicklung für Configuration Management in einem Rechenzentrum nach ITIL und MOF*. Diplomarbeit, 2005.

[Satt 05] SATTLER, H. (FUJITSU SIEMENS COMPUTER): *Web Services und Service Orientierte Architekturen (SOA)*. 12.12.2005, http://www13.informatik.tu-muenchen.de/gi/files/05-12-12.pdf .

[Scha 05] SCHAUMLÖFFEL, E.: *SOA: Mehr als nur Web-Services?* OBJEKTspektrum Nr.5, 2005.

[Sine 03] SINEM, K.: *Ereignisgesteuerte Prozessketten*. 2003, http://www.informaktik.uni-unlm.de/dbis/01/lehre/ss03/gs/p_ablauf/EPK.pdf .

[Soik 05] SOIKA, R. (IMIXS SOFTWARE SOLUTIONS GMBH): *BPEL und SOA Architekturen*. 2005, http://bpm-guide/articles/25 .

[Sun 04] SUN MICROSYSTEMS, INC.: *Enterprise JavaBeans^{TM} Components and CORBA Clients: A Developer Guide*. 2004, http://java.sun.com/j2se/1.5.0/docs/guide/rmi-iiop/interop.html .

[SYS- 05a] SYS-CON MEDIA, INC.: *BPEL To The Rescue: A Real-World Account on SOA Web Services*. 2005, http://webservices.sys-son.com/read/46870_p.htm .

[SYS- 05b] SYS-CON MEDIA, INC.: *Process-Centric Realization of SOA*. 2005, http://webservices.sys-con.com/read/46870_p.htm .

[W3C 01] W3C: *Web Services Description Language (WSDL) Version 1.1 (W3C Note)*. 15 March 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315 .

[W3C 03a] W3C: *SOAP Version 1.2 Part 1 (W3C Recommendation): Messaging Framework*. 24 June 2003, http://www.w3.org/TR/soap .

[W3C 03b] W3C: *SOAP Version 1.2 Part 2 (W3C Recommendation): Adjuncts*. 24 June 2003, http://www.w3.org/TR/soap12-part2/ .

[W3C 04a] W3C: *Web Services Architecture (W3C Working Group Note)*. 11 February 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/ .

[W3C 04b] W3C: *XML Schema Part 1: Structures Second Edition (W3C Recommendation)*. 28 October 2004, http://www.w3.org/TR/xmlschema-1/ .

[W3C 04c] W3C: *XML Schema Part 2: Datatypes Second Edition (W3C Recommendation)*. 28 October 2004, http://www.w3.org/TR/xmlschema-2/ .

[W3C 05a] W3C: *Web Services Description Language (WSDL) Version 2.0 Part 0 (W3C Working Draft): Primer*. 3 August 2005, http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803 .

[W3C 05b] W3C: *Web Services Description Language (WSDL) Version 2.0 Part 1 (W3C Working Draft): Core Language*. 3 August 2005, http://www.w3.org/TR/2005/WD-wsdl20-20050803 .

[W3C 05c] W3C: *Web Services Description Language (WSDL) Version 2.0 Part 2 (W3C Working Draft): Adjuncts*. 3 August 2005, http://www.w3.org/TR/2005/WD-wsdl20-adjuncts-20050803 .

[Wadh 02] WOHED, P., W.M.P. VAN DER AALST, M. DUMAS und A.H.M. TER HOFSTEDE: *Pattern Based Analysis of BPEL4WS*. Technical Report FIT-TR-2002-04, QUT, 2002.

[Whit 05] WHITE, S.A. IBM CORP. UNITED STATES: *Using BPMN to Model a BPEL Process*. February 2005, http://www.bpmn.org/Documents/Mapping BPMN to BPEL Example.pdf .

[ZTP 03] ZIMMERMANN, O., M. TOMLINSON und S. PEUSER: *Perspectives on Web Services: Applying SOAP, WSDL and UDDI to Real-World Projects*. Springer Verlag Berlin, ISBN 3-540-00914-0, 2003.

Index