1 Einleitung

Geschäftsprozessmanagement (GPM, engl.: Business Process Management), Geschäftsprozessmodellierung, Geschäftsprozesse und serviceorientierte Architektur (SOA) – all dies sind Themen, die seit einigen Jahren in der IT in aller Munde sind. Die Aufmerksamkeit, die diese Themen dabei auf sich ziehen, ist im Wesentlichen darin begründet, dass sie versprechen, das eigentliche Geschäft eines Unternehmens besser durch IT unterstützen zu können. Dazu setzen sowohl Business Process Management (BPM) als auch SOA nicht erst auf der technischen Ebene an: Sie bieten Konzepte, um auf fachlicher Ebene die Prozess- und Servicelandschaft zu diskutieren, und zudem Mechanismen, die es ermöglichen, diese fachlichen Modelle einfach auf die technische Infrastruktur abzubilden. Daher wollen wir genauer betrachten, was sich hinter den Begriffen BPM und SOA verbirgt.

Der erste Schritt in BPM-Projekten ist meist die Dokumentation der Geschäftsabläufe eines Unternehmens in sogenannten *fachlichen Prozessmodellen*. Das mag banal klingen, ist jedoch häufig mit sehr viel Aufwand verbunden, da sich die in einem Unternehmen gelebten Prozesse über Jahre entwickelt haben. Sie sind oft über mehrere Systeme verteilt, die jeweils einen Teil des Geschäftsprozesses unterstützen, und beinhalten neben den in obigen Systemen implementierten Teilprozessen auch Teile, die (noch) nicht durch die IT unterstützt werden. Hinzu kommt, dass es oftmals niemanden gibt, der den Prozess als Ganzes kennt oder versteht. Ein gemeinsames Verständnis bezüglich der Prozessabläufe in einem Unternehmen zu erreichen, sollte also niemals unterschätzt werden und ist Grundlage für alle weiteren Schritte. Ein fachliches Prozessmodell beschreibt, welche Schritte bzw. Aktivitäten in welcher Reihenfolge ausgeführt werden müssen, um das Geschäftsziel des Prozesses zu erreichen. Außerdem müssen aus einem Prozessmodell Verantwortlichkeiten und benötigte bzw. erzeugte Daten hervorgehen. Meist ist das Ziel eines BPM-Projekts jedoch nicht die reine Dokumentation der vorhandenen Geschäftsprozesse, sondern beinhaltet auch deren Optimierung und deren optimale Unterstützung durch IT-Systeme. Dabei können die fachlichen Prozessmodelle zum einen als Softwareanforderung für herkömmliche IT-Systeme verstanden werden, zum anderen aber auch als Vorlage für *ausführbare Geschäftsprozesse*, die von einer Workflow-Engine interpretiert werden. Dies setzt jedoch voraus, dass für jede spezifizierte Aktivität des ausführbaren Geschäftsprozesses ein Stück Software vorhanden ist, das die von der Aktivität geforderte Funktionalität implementiert bzw. einem Benutzer die Möglichkeit bietet, die geforderte Funktionalität zu erbringen.

Mit dieser Anforderung der ausführbaren Geschäftsprozesse im Hinterkopf können wir betrachten, was sich hinter dem Begriff SOA und vor allem hinter dem Begriff *Service* verbirgt. Zunächst einmal lässt sich festhalten, dass sich die Frage: »Was ist SOA?« bzw. »Was ist ein Service?« nicht eindeutig beantworten lässt. Wenn man zehn verschiedene Experten fragt, bekommt man höchstwahrscheinlich zehn unterschiedliche, wenn nicht sogar teilweise widersprüchliche Antworten. So ist es nicht verwunderlich, dass sich die Definitionen von SOA in der Literatur mitunter unterscheiden. Wenn Sie sich selbst einen Überblick verschaffen möchten, gängige Definitionen von SOA finden Sie z. B. in [Josuttis 2008], [Erl 2005] und [Weerawarana et al. 2005].

Zur Verwendung in unserem Buch unterscheiden wir zwischen fachlichem Service und technischem Service. Unter einem fachlichen Service verstehen wir ganz allgemein die Bündelung von fachlich zusammenhängender Funktionalität. Diese fachlichen Services lassen sich anhand eines sogenannten Domänenmodells in unterschiedliche funktionale Bereiche kategorisieren (z. B. Entwicklung, Produktion und Kundenbetreuung). Im Kontext des Enterprise Architecture Management (EAM) sind das Domänenmodell (d. h. die fachlichen Services) und die fachlichen Prozessmodelle eines Unternehmens Bestandteile der sogenannten *Geschäftsarchitektur* und bieten somit die Grundlage für die *IT-Bebauung*. Analog zu den fachlichen Prozessmodellen können auch fachliche Services als Softwareanforderung für herkömmliche Systeme betrachtet werden. Sie können aber auch als Grundlage für eine SOA auf technischer Ebene dienen. Auf technischer Ebene definieren wir einen Service als ein Stück Software, das eine bestimmte Funktionalität implementiert und zudem folgende Eigenschaften hat bzw. haben sollte:

- Ein Service kapselt *Geschäftsfunktionalität*, d. h., er ist entsprechend grobgranular.
- Ein Service ist *in sich abgeschlossen* und unabhängig vom Kontext eines potenziellen *Servicenutzers*.
- Service und Servicenutzer sind *lose gekoppelt*.
- Ein Service verfügt über eine *wohldefinierte Schnittstelle* und ist *auffindbar*.
- Ein Service folgt der *Always-on*-Semantik, d. h., er ist *immer verfügbar* und muss vor der Nutzung nicht erst initialisiert werden.

Der *Serviceanbieter*, der den eben beschriebenen Service zur Verfügung stellt, und der ebenfalls bereits erwähnte *Servicenutzer*, der diesen Service nutzt, bilden zusammen mit dem *Verzeichnisdienst* die SOA. Die Beziehungen der drei Bausteine der SOA ist in Abbildung 1-1 dargestellt. Der Serviceanbieter veröffentlicht den von ihm angebotenen Service beim Verzeichnisdienst. Möchte der Servicenutzer einen Service nutzen, so kann er diesen über den Verzeichnisdienst finden, ihn anschließend binden und letztendlich den Service aufrufen.

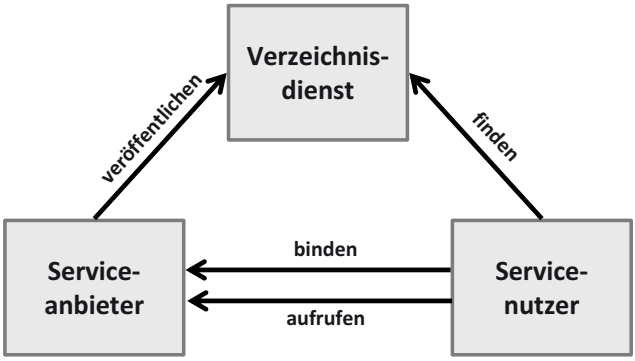

Abb. 1-1: Die Beziehungen der Bausteine einer serviceorientierten Architektur (SOA)

Aus einem anderen Blickwinkel betrachtet kann man auch sagen: Werden technische Services über eine Prozesslogik zusammengesteckt, so ergibt sich daraus die Implementierung eines Geschäftsprozesses in Form einer *Serviceorchestrierung*.

Was in der Theorie angenehm einfach und einleuchtend klingt, ist in der Praxis nicht immer ohne Weiteres umsetzbar. Außerdem gibt es mittlerweile für fast alle benötigten Technologien offizielle Standards, jedoch erschwert diese Vielzahl die Auswahl.

Dieses Buch zeigt auf, wie mit der Business Process Execution Language (BPEL), die zur Familie der Webservice-Spezifikationen gehört, Serviceorchestrierungen technisch sinnvoll umgesetzt werden können. Warum wir bei der technischen Umsetzung der SOA und der Prozessausführung gerade auf Webservices und BPEL setzen, wird in den folgenden beiden Abschnitten erläutert. In den Kapiteln 2, 4, 5 und 7 werden die Grundlagen beschrieben: die Modellierung einer fachlichen Geschäftsarchitektur inklusive der Modellierung von fachlichen Geschäftsprozessen, die wichtigsten Webservice-Spezifikationen bzw. Standards, die Sprache BPEL und das Testen von BPEL-Prozessen. Diese Abschnitte sind so angelegt, dass das Buch als Nachschlagewerk benutzt werden kann. Damit dieses Buch nicht nur aus theoretischen Inhalten besteht, werden nach und nach ausgewählte Prozesse in einer fiktiven Firma, die uns als Fallbeispiel dienen soll, identifiziert, modelliert, implementiert und schließlich getestet. Das Fallbeispiel wird in Kapitel 3 vorgestellt und die technische Umsetzung in Kapitel 6 präsentiert.

Zur Geschäftsprozessmodellierung und für BPEL gibt es eine Reihe von Werkzeugen zur Entwicklung und Ausführung. Der Markt ist groß und jeder größere (und auch viele kleinere) Hersteller bieten Produkte, Angebote und Dienstleistungen rund um BPEL an, u.a. IBM, Oracle, Microsoft, Active Endpoints, Intalio. Dieses Buch vermittelt die Konzepte der Sprache und ihre Einsatzmöglichkeiten und ist dabei weitestgehend herstellerunabhängig. Damit ist sichergestellt, dass dieses Wissen nicht nur für eine aktuelle Version einer bestimmten Software gül-

tig ist, sondern die erworbenen Kenntnisse auch langfristig erfolgreich eingesetzt werden können.

Alle Beispiele sind so konzipiert, dass sie nicht nur mit kommerziellen Produkten, sondern auch mit frei verfügbaren Werkzeugen nachvollzogen werden können. Das bedeutet, dass alle Konzepte praktisch angewendet werden können, auch wenn am Arbeitsplatz oder privat keine Lizenzen für kommerzielle BPEL-Produkte zur Verfügung stehen. Als Software verwenden wir dafür den BPEL-Designer für Eclipse, Apache ODE zur Ausführung und BPELUnit zum Testen. Im Anhang sind Kurzeinführungen und Installationsanleitungen für diese Tools zu finden, um den Start zu erleichtern.

1.1 Evolution der Anwendungsintegration

Bis zum heutigen Tage werden Anwendungen oft isoliert für einen bestimmten Verwendungszweck entwickelt. Als Folge dieses Vorgehens gibt es vor allem in großen Unternehmen viele, voneinander unabhängige Anwendungen, z. B. für den Einkauf, für die Produktion, für den Verkauf und für die Kundenbetreuung. Die unterschiedlichen Geschäftsfelder, die durch die Anwendungen unterstützt werden, sind jedoch alle Teil der gleichen Wertschöpfungkette, d. h., Daten, die ein System erzeugt, werden oft auch in anderen Systemen benötigt. Wird beispielsweise in der Produktion dokumentiert, aus welchen Bestandteilen bzw. Bauteilen ein Produkt besteht, so ist es unter Umständen sehr hilfreich und manchmal auch unbedingt notwendig, aus anderen Systemen auf diese Daten zugreifen zu können, um das Produkt z. B. im Falle einer Reklamation des Kunden kostengünstig reparieren zu können. Das hat zur Folge, dass unterschiedliche Systeme, die möglicherweise auch auf unterschiedlichen Plattformen basieren, miteinander integriert werden müssen.

Diese Probleme versuchte die IT zu unterschiedlichen Zeitpunkten der Geschichte der Anwendungsintegration mit unterschiedlichen Mitteln – und mit unterschiedlichem Erfolg – zu lösen. Neben Ansätzen, die einen entfernten Prozedur- oder Methodenaufruf ermöglichen wie RPC, RMI, DCOM oder die Common Object Request Broker Architecture (CORBA) existieren Integrationsansätze, die auf der Übertragung von Informationen beruhen wie beispielsweise Dateitransfer oder Messaging.

Letztgenanntes Messaging kam seit den 90er-Jahren im Zuge der Enterprise Application Integration (EAI) [Hohpe & Woolf 2003] vermehrt zur Integration verschiedener Anwendungen zum Einsatz. Im Gegensatz zu den bis dato existerenden Ansätzen zum Aufruf einer entfernten Prozedur oder Methode bietet Messaging den Vorteil, dass das aufrufende System und das aufgerufene System nicht direkt miteinander kommunizieren. Sie werden über sogenannte Adapter gekapselt, die über einen Kanal bzw. eine Queue Nachrichten austauschen. Somit sind sie in mehrerer Hinsicht entkoppelt: zum einen fachlich, da das sendende System immer

noch mit dem aufgerufenen System kommunizieren kann, wenn dieses das Nachrichtenformat der zu sendenden Nachricht ändert, weil die Nachricht innerhalb des Kanals transformiert werden kann. Zum anderen zeitlich, da durch den Einsatz einer Queue nicht beide Systeme gleichzeitig verfügbar sein müssen, um miteinander kommunizieren zu können. Die auf Messaging basierenden EAI-Produkte verfügen über Adapter zu vielen Technologien und ermöglichen so die Integration heterogener Systeme, die auf unterschiedlichen Plattformen beruhen. Allerdings folgen sie keinen offenen Standards. Die Verwendung eines EAI-Produkts resultiert daher auf kurz oder lang in einem Vendor Lock-in.

Ein zunächst Erfolg versprechender Ansatz, die Kommunikation von verteilten Systemen über ein *offenes* plattformunabhängiges Framework zu realisieren, war das bereits erwähnte CORBA [Vinoski 1997]. CORBA ist eine Spezifikation der Object Management Group (OMG), die das Paradigma der Objektorientierung direkt auf verteilte Systeme anwendet und definiert, wie verteilte Objekte sich gegenseitig aufrufen können. Allerdings birgt der von CORBA verfolgte Ansatz einige Schwachstellen. So führt die direkte Umsetzung der objektorientierten Programmierung auf verteilte Systeme zu sehr feingranularen Aufrufen – ein Umstand, der sich z. B. in Performanzproblemen niederschlägt – und zu einer gewissen Fragilität des Gesamtsystems: Will ein System die Methode einer durch ein entferntes Objekt implementierten Klasse aufrufen, so muss dem aufrufenden System nicht nur die Zielklasse bekannt sein, sondern die gesamte Klassenhierarchie. Wird nun innerhalb dieser Hierarchie eine Änderung vorgenommen, so müssen die aufrufenden Systeme ebenfalls modifiziert werden [Weerawarana et al. 2005]. Diese enge Kopplung führt im besten Fall zu einer hohen Komplexität und hohen Wartungskosten, im schlechtesten Fall werden ganze Systeme unwartbar. Die Komplexität wird dadurch verstärkt, dass bei CORBA die entfernten Objekte ebenso wie lokale Objekte vor ihrer Verwendung instanziiert werden müssen und nach der Instanziierung des Objekts die Referenz auf das Objekt vom System verwaltet werden muss. All dies spiegelt sich u.a. auch in der Komplexität der API von CORBA wider. Eine weitere Schwäche ist die nur bedingt erreichbare Interoperabilität von CORBA. Dies liegt zum einen am eingesetzten Datenmodell und den unzureichenden Datenmappings, die zur Inkompatibilität unterschiedlicher Programmiersprachen führen, und zum anderen an den Problemen der CORBA-Protokolle mit gängigen Firewalls. Letztendlich ist jedoch die mangelnde Akzeptanz seitens Microsoft als Hauptgrund für die schwindende Popularität von CORBA zu sehen. Nichtsdestotrotz ist mit CORBA ein wichtiger Schritt in der Evolution der Integration von Anwendungssystemen gelungen. Im Laufe der Zeit hat man in CORBA-Projekten erkannt, dass entgegen des objektorientierten Ansatzes grobgranulare Aufrufe verwendet werden sollten und dass vernünftige Richtlinien und Best Practices zur Vernetzung großer Applikationslandschaften erforderlich sind. Im Laufe der Zeit ist es so gelungen, mithilfe dieser Technologie verschiedene Anwendungen, die auf unterschiedlichen Plattformen aufsetzen, erfolgreich zu integrieren.

Der auf CORBA folgende Ansatz der plattformübergreifenden und technologieunabhängigen Integration durch ein *offenes* Framework war die *Webservice-Technologie*. Die Entwicklung der Webservice-Technologie begann um die Jahrtausendwende mit der Veröffentlichung von SOAP durch Microsoft als erstem Schritt eines Gegenentwurfs zu CORBA. Die Webservice-Technologie ist im Gegensatz zu CORBA keine Umsetzung des objektorientierten Paradigmas, da sich dieses als ungeeignet zur Integration von verteilten Systemen herausgestellt hatte. Stattdessen basiert sie auf der Idee des Austauschs von in XML serialisierten Nachrichten zwischen grobgranularen Services, die über eine wohldefinierte Schnittstelle verfügen. Änderungen der Implementierung eines Service haben keine Auswirkung auf die aufrufenden Systeme, solange der verwendete Teil der Schnittstelle stabil bleibt. Ein Service folgt der *Always-on*-Semantik, d. h., er ist immer bereit, eine der Schnittstellenbeschreibung entsprechende Nachricht zu empfangen. Eine Instanziierung ist nicht notwendig. Webservices unterstützen sowohl asynchrone als auch synchrone Kommunikation – vereinen also alle gängigen Interaktionsstile – und ermöglichen die Verwendung aller gängigen Transportprotokolle. In der Praxis wird jedoch meist das Hyper Text Transfer Protocol (HTTP) verwendet. Zum einen aus Interoperabilitätsgründen, weil nahezu alle Plattformen HTTP implementieren, und zum anderen, weil bei der Verwendung von HTTP alle Nachrichten über Port 80 der Firewalls getunnelt werden können.

Neben der vorteilhaften technischen Eigenschaften der Webservice-Technologie ist nicht zu unterschätzen, dass alle Schwergewichte der Softwarebranche – unter anderem Microsoft und IBM – an der Standardisierung teilnehmen und die Standards in ihren Produkten implementieren. Die Webservice-Technologie hat somit die Chance, sich *nachhaltig* zu *dem* Integrationswerkzeug zur Integration verteilter, heterogener Systeme zu etablieren.

1.2 Evolution der Prozessausführung

So wie Services sich aus vorhergehenden Ansätzen zur Integration heterogener, verteilter Systeme entwickelt haben, so hat sich auch die Architektur von Anwendungssystemen im Laufe der Zeit geändert. In den Anfängen der Anwendungsentwicklung wurden monolithische Anwendungen erstellt, in denen alle Aspekte – unter anderem Datenpersistenz und Prozesslogik – in einem System realisiert wurden. Ein großer Schritt in der Evolution der Anwendungsgeschichte (siehe Abb. 1-2) war in den 60er-Jahren die Einführung von sogenannten Datenbankmanagementsystem (DBMS) und die damit einhergehende Trennung der Datenhaltung vom Rest einer Anwendung. Die reine Anwendungsentwicklung konnte sich seit diesem Zeitpunkt auf die Implementierung der Daten*verarbeitung* konzentrieren und konnte zur Persistierung der Daten auf existierende DBMS zurückgreifen. Einen weiteren evolutionären Schritt stellt die Einführung der Workflow-Technologie zur Prozessausführung in den 90er-Jahren dar. Sie ist zwar bei Weitem noch nicht so

etabliert wie DBMS, da es lange Zeit keinen allgemein anerkannten Standard zur Definition von Workflows gab, hat aber in den letzten zehn Jahren zunehmend an Bedeutung gewonnen. Die Workflow-Technologie unterteilt Anwendungen in zwei Bestandteile: *Geschäftslogik* und *Geschäftsfunktionen*. Die Geschäftslogik wird in sogenannten Workflow- bzw. Prozessmodellen spezifiziert, die u.a. die Reihenfolge definieren, in der Geschäftsfunktionen ausgeführt werden, um das hinter dem Workflow- bzw. Prozessmodell stehende Geschäftsziel zu erreichen. Es wird also zwischen Programmieren »im Großen« (das Spezifizieren der Geschäftslogik) und Programmieren »im Kleinen« (die Implementierung der Geschäftsfunktionen) unterschieden und somit das »two-level programming« [Leymann & Roller 1999] als Paradigma eingeführt.

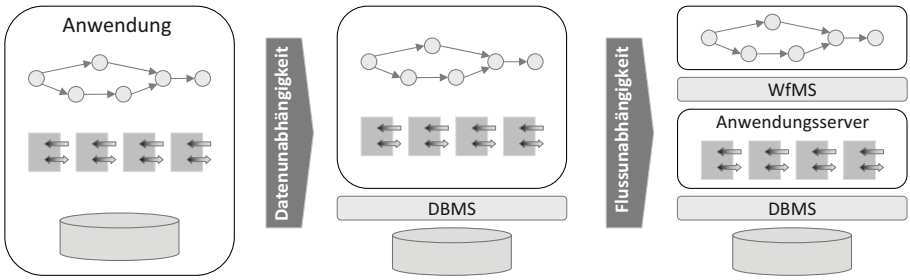

Abb. 1-2: Evolution der Anwendungsentwicklung

Zur Implementierung der Geschäftsfunktionen werden weiterhin die traditionellen Programmiersprachen verwendet. Zur Definition von Workflow- bzw. Prozessmodellen wurden eigens Sprachen entwickelt, die alle für einen Workflow relevanten Dimensionen [Leymann & Roller 1999] unterstützen. Diese sind in Abbildung 1-3 dargestellt. Die *Wie*-Dimension beschreibt, in welcher Reihenfolge Geschäftsfunktionen verwendet werden, um das Geschäftsziel des Prozesses zu erreichen; die *Wer*-Dimension gibt an, welche Personen oder Personengruppen an der Ausführung einer Geschäftsfunktion beteiligt sind, und die *Womit*-Dimension beschreibt, welche Hilfsmittel zur Ausführung einer Geschäftsfunktion verwendet werden.

Die Prozessmodelle werden durch sogenannte Workflow-Management-Systeme (WfMS) interpretiert. Dazu werden die Prozesse nach ihrer Modellierung auf ein Workflow-Management-System »deployed«, das dann Instanzen dieser Modelle erstellt und ausführt. Während der Ausführung werden alle durchgeführten Prozessschritte einer Prozessinstanz vom WfMS protokolliert, sodass sowohl Monitoring von Prozessinstanzen während der Ausführung als auch eine spätere Analyse der abgelaufenen Prozessinstanzen möglich ist. Durch die Workflow-Technologie verspricht man sich mehr Flexibilität bei der Anwendungsentwicklung und dem Betrieb von Anwendungen. Änderungen an Geschäftsprozessen, die beispielsweise durch veränderte Anforderungen des Markts oder durch sich ändernde gesetzliche Vorgaben bedingt sind, erfordern meist nur eine Änderung der Prozessmodelle

und sind daher sehr schnell umsetzbar, da die Prozessmodelle nach der Änderung lediglich »redeployed« werden müssen.

Anfang der 1990er-Jahre gab es noch keine Workflow-Management-Produkte, sodass manche große Unternehmen ihr eigenes WfMS implementierten. Mitte der 1990er-Jahre kamen die ersten Produkte auf den Markt, unter ihnen IBM MQSeries Workflow, Oracle Workflow, HP ChangEngine und SAP Business Workflow. Die Produkte basierten jedoch nicht auf einer einheitlichen Sprache, sondern implementierten jeweils eine eigens vom Hersteller für das WfMS definierte Prozessausführungssprache. Diese Sprachen unterschieden sich in der Repräsentation der Workflow-Dimensionen. Die »Wie«-Dimension wurde entweder über eine graphbasierte Repräsentation des (Daten- und) Kontrollflusses im Metamodell oder blockbasiert beschrieben. Die »Womit«-Dimension basierte auf unterschiedlichen Komponentenmodellen, und auch die Definition der »Wer«-Dimension geschah nicht auf einheitlicher Basis. Mitte der 1990er wurden erste Versuche im Rahmen der Workflow Management Coalition (WfMC) unternommen, eine Workflow-Sprache zu standardisieren, die jedoch mangels Akzeptanz der Hersteller erfolglos blieben. Mit dem Aufkommen der Webservice-Technologie wurde Anfang des

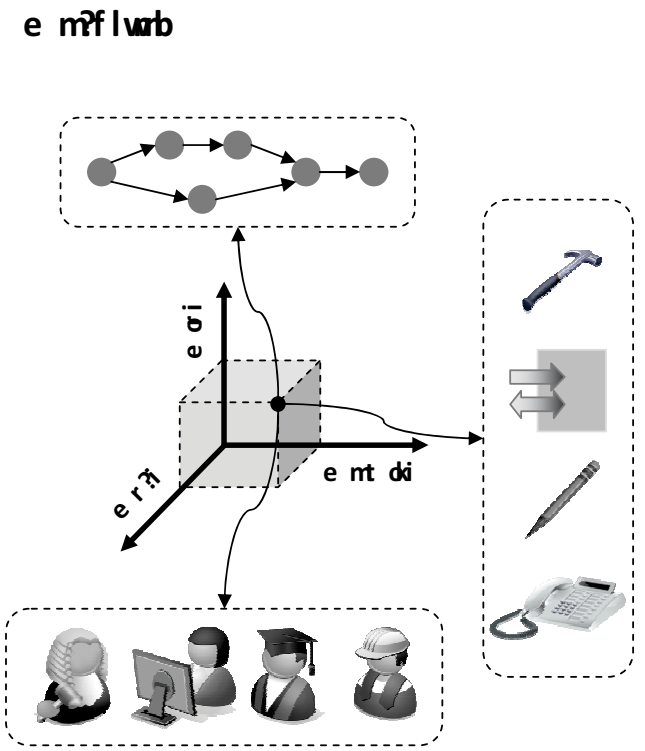

Abb. 1-3: Dimensionen eines Workflows

Jahrtausends ein weiterer Anlauf zur Standardisierung unternommen. Nachdem zunächst IBM und Microsoft mit der Web Services Flow Language (WSFL) [Leymann 2001] bzw. XLANG [Thatte 2001] jeweils eine eigene Sprache definierten, die zwar zur Beschreibung der »Wie«-Dimension zwei unterschiedliche Ansätze verfolgten (WSFL mit einem graphbasierten und XLANG mit einem blockbasierten Metamodell), zumindest aber zur Beschreibung der »Womit«-Dimension Webservices verwendeten, wurde erneut der Versuch gestartet, sich auf eine gemeinsame Sprache zur Definition von Workflows zu einigen. Das Ergebnis dieser Bemühungen war die im Jahr 2002 von IBM, Microsoft und BEA veröffentlichte Spezifikation BPEL 1.0 [Curbera et al. 2002].

BPEL4WS oder kurz BPEL verwendet Webservices zur Spezifikation der »Womit«-Dimension und verfügt über sprachliche Mittel, um die »Wie«-Dimension sowohl graphbasiert als auch blockbasiert zu spezifizieren. Die Spezifikation der »Wer«-Dimension wird von BPEL nicht direkt unterstützt, weil es konzipiert wurde, um vollständig automatisierte Geschäftsprozesse zu implementieren. Es handelt sich bei BPEL also um eine Sprache zur Workflow-basierten Komposition von Webservices. BPEL bietet ein rekursives Aggregationsmodell für Webservices: Ein BPEL-Prozess komponiert nicht nur Webservices, sondern bietet auch die eigene Funktionalität als Webservices an, d. h., die Sprache BPEL ermöglicht die Umsetzung des Composite-Patterns [Gamma et al. 1995]. Ein BPEL-Prozess kann also auch andere BPEL-Prozesse aufrufen oder von anderen BPEL-Prozessen aufgerufen werden. BPEL ist nicht beschränkt auf die Verwendung und Bereitstellung zustandsloser Services, sondern ermöglicht die Definition einer lang andauernden Konversation zwischen Service und Servicenutzer, die mehrere Nachrichten umfasst.

Der Version 1.0 folgte im Mai 2003 die Version 1.1, an der neben den bereits genannten drei Unternehmen weitere Industriepartner beteiligt waren. Im Jahr 2007 wurde BPEL schließlich in der Version 2.0 [Alves et al. 2007] unter dem Namen WS-BPEL von OASIS standardisiert und hat – gemessen an der Akzeptanz in der Industrie – das Potenzial, zu *dem* Workflow-Standard zu werden, so wie SQL [Date & Darwen 1998] der Standard für Datenbanken ist. Zusätzlich werden für BPEL diverse Erweiterungen standardisiert, wie z. B. BPEL4People, das Benutzerinteraktionen mit Serviceorchestrierungen erlaubt und somit die Spezifikation von BPEL um die »Wer«-Dimension erweitert.

Parallel zur Standardisierung von BPEL wurde mit BPMN (Business Process Model and Notation, damals stand die Abkürzung jedoch noch für Business Process Modeling Notation) eine Spezifikation einer grafischen Darstellung (Notation) für Geschäftsprozesse veröffentlicht, einem Aspekt, der bei der Entwicklung von BPEL aus Zeitgründen offengelassen wurde. Während sich BPEL zum De-facto-Standard für technische und ausführbare Prozesse entwickelt hat, wurde die BPMN im Bereich der fachlichen Modellierung ein großer Erfolg. Nicht-IT-Experten können mit BPMN ihre Geschäftsprozesse modellieren und kommunizieren, ohne tief in technische Details einsteigen zu müssen.

Zur Drucklegung dieses Buches zwar fertiggestellt, aber noch nicht offiziell verabschiedet, ist die Version 2.0 der BPMN. Sie definiert ein XML- und XMI-basiertes Austauschformat für Modellierungswerkzeuge und – ähnlich wie BPEL – eine wohldefinierte Ausführungssemantik. Das ist nicht nur für die Automatisierung der Prozesse wichtig, sondern garantiert auch, dass Modelle von Experten auf identische Weise interpretiert werden. Anders als bei BPEL wird eine Ausführbarkeit der Modelle jedoch nicht garantiert. Unklar ist zur Zeit, ob BPMN 2.0 langfristig BPEL als Prozessautomatisierungsstandard ablösen wird und wie schnell der momentane Hype um BPMN abflaut, u.a. weil Werkzeuge im frühen Entwicklungsstadium den produktiven Einsatz von BPMN erschweren. Nicht zuletzt aufgrund des höheren Reifegrads der verfügbaren Werkzeuge für BPEL wird die Sprache in den kommenden Jahren einen festen Platz im Bereich der Prozessautomatisierung einnehmen.