

Den Software-Prozess beschleunigen – mit gezielter Innovation

Die Software-Entwicklung dauert nach wie vor zu lang und ist zu teuer. Diese Zustandsbeschreibung wird nun schon jahrzehntelang wiederholt und trifft trotz mancher Verbesserungen immer noch zu.

Die Zahl der Methoden und Werkzeuge, mit deren Hilfe sich dieser Zustand abmildern lässt, sind in den letzten Jahren deutlich gewachsen. Oberflächlich betrachtet müsste nun eine kritische Masse erreicht sein, um das Übel ein für alle mal zu beseitigen. Jedoch wird immer noch oft vergessen, dass Software nicht vollautomatisch entsteht, sondern immer noch zu einem bedeutenden Teil von Menschen erzeugt wird. Genau an diesem Punkt liegt nach wie vor die Schwäche.

Noch immer scheitern viel zu viele Projekte und Millionen werden buchstäblich in den Sand gesetzt, weil an den Anforderungen des Auftraggebers vorbei entwickelt wurde. Viel zu schnell wird Code produziert, weil sichtbare Ergebnisse gefordert werden. Nur, was nützt das Ganze, wenn am Ende nicht das gewünschte Ergebnis steht?

Trotz bemerkenswerter Effizienzsteigerungen im Software-Entwicklungsprozess, die in Teilbereichen eine 100%ige Code-Generierung ermöglichen, bleiben immer noch eine ganze Reihe von Problembereichen. Einer dieser Problembereiche ist die Schnittstelle zwischen Fachbereich und IT, die Umsetzung von fachlichen Anforderungen in die Systemspezifikation. An dieser Stelle können noch bedeutende Produktivitätsreserven erschlossen werden.

Wie würden unsere Häuser aussehen, wenn Bauherren nicht gezwungen wären, einen Bauplan einzureichen und die Statik prüfen zu lassen? Vermutlich würden sehr viele Bauherren auf Basis eines Minimalplans sehr schnell mit dem Bauen anfangen. Man weiss ja schliesslich wie ein Haus gebaut wird. Was würde am Ende aber tatsächlich herauskommen? Natürlich sind Hausbau und Software-Entwicklung völlig unterschiedliche Disziplinen. Es ist auch wesentlich einfacher, sich ein fertiges Haus bildlich vorzustellen als ein Anwendungssystem. Weil dies so ist, muss in der Software-Entwicklung wesentlich mehr getan werden, damit Auftraggeber am Ende das bekommen was sie in Auftrag gegeben haben.

Das Hauptproblem liegt darin, die Verständnislücke zwischen Fachbereichen, die das Anwendungssystem später nutzen, und dem Projektteam, das das Anwendungssystem entwickelt, möglichst klein zu halten. Auch das ist seit Jahrzehnten bekannt. Nur, wie löst man das Problem?

1 Geschäftsprozesszentriertes Vorgehen

In vielen Software-Projekten der jüngeren Zeit, ab Ende der neunziger Jahre, hat es sich als sinnvoll erwiesen, von der realen Welt der Fachbereiche auszugehen und in Geschäftsprozessen zu denken. Am Anfang steht somit die Geschäftsprozessanalyse, -modellierung und -optimierung. Diese Aufgaben verlangen noch keine Entwicklung von Programm-Code. Geschäftsprozesse bestehen aus ein oder mehreren Aktivitäten, die in einer zur Ausführungszeit dynamisch bestimmten Folge ausgeführt werden.

Die grosse Herausforderung besteht darin, dass die im Software-Prozess beteiligten Personen in der Lage sein müssen, Geschäftsprozesse in strukturierter Weise mit reichhaltiger Semantik zu beschreiben. Dafür werden jedoch geeignete Werkzeuge und Beschreibungsmethoden benötigt. Beschreibungsmethoden für objektorientierte Systeme, wie z.B. UML,

sind nur sehr bedingt geeignet. Deshalb bleiben auch die mit den heutigen Werkzeugen erzielten Arbeitsergebnisse hinter den Erwartungen zurück.

In den meisten Software-Projekten wird auf eine strukturierte Beschreibung des Problemereichs viel zu wenig Wert gelegt. Die wichtigen Begriffe werden zwar anfänglich in einem Glossar beschrieben, jedoch wird die Pflege des Glossars vernachlässigt und somit veraltet es sehr schnell. Davon abgesehen ist die Beschreibung der einzelnen Begriffe meist nicht präzise genug und wenig strukturiert. So kann es sehr schnell vorkommen, dass mögliche Synonyme (z.B. Kunde, Klient) und Homonyme (z.B. Bank) entstehen und nicht sofort entdeckt werden.

Häufig wird zu schnell zum Werkzeug gegriffen, um Modelle zu erstellen und damit die Anforderungen an das System in eine bestimmte Syntax zu übersetzen. Dass Syntax nicht einfach Semantik gleichzusetzen ist, wird oft vergessen. Die meisten Modelle können Semantik nur ungenügend ausdrücken und viel Semantik geht gerade durch die Fokussierung auf die Modellierung mit geringem Semantikgehalt verloren. Dabei ist Semantik eine wesentliche Voraussetzung für die Generierung sinnvoller Ergebnisse.

Die Konsequenz kann nur lauten: Modelle müssen semantisch so reichhaltig wie möglich sein. Je früher und besser dies erreicht wird, desto „intelligenter“ kann Quell-Code aus einer Spezifikation erzeugt werden. Was am Anfang versäumt wird, lässt sich später nicht mehr kompensieren. Gerade in Grossprojekten mit einer Vielzahl von Software-Ingenieuren entsteht oft sehr schnell Wildwuchs. Im Endeffekt entstehen vermeidbare Reibungsverluste und „die Leute arbeiten aneinander vorbei“.

Je besser die Code-Generierung aus der Spezifikation gelingt, desto besser können auch Produktivitätssteigerungspotenziale erschlossen werden. Die Arbeitsproduktivität wird schliesslich am wirkungsvollsten über Innovationen gesteigert, wobei der intensivere Computereinsatz daran wesentlichen Anteil hat.

Darüber hinaus dürfen auch andere Möglichkeiten zur Steigerung der Arbeitsproduktivität nicht unerwähnt bleiben. Ein nachgewiesenermassen wirksames Mittel besteht in der Reduzierung von Komplexität durch Vereinfachung von Prozessen. Auch die engere Kooperation mit anderen Unternehmen durch vollständige oder teilweise Auslagerung von Geschäftsprozessen trägt zur Produktivitätssteigerung bei.

Die an dieser Stelle kurz skizzierten Massnahmen repräsentieren nur einen Ausschnitt des gesamten Arsenal. Unternehmen müssen künftig sehr viel schneller auf Ereignisse reagieren können und Produktivitätssteigerungen sehr viel schneller wirksam machen. Im Endeffekt muss ein Unternehmen in der Lage sein, das operative Geschäft immer wieder und in kurzer Zeit umzubauen.

Wenn ein Unternehmen immer schneller reagieren können muss, hat dies auch Auswirkungen auf die Software-Entwicklung. Auch in diesem Bereich muss die Produktivität deutlich gesteigert werden, damit Anforderungen der Fachbereiche zügig umgesetzt werden können. Projekte mit einer Laufzeit von mehr als zwei Jahren sind vor diesem Hintergrund nicht mehr darstellbar. Darüber hinaus muss auch das Risiko, dass die Anforderungen des Fachbereichs nicht erfüllt werden, minimiert werden.

Auf die eigentliche Implementierung von Geschäftsprozessen entfällt, gemessen am Gesamtaufwand, nur ein relativ geringer Anteil. Am zeit- und ressourcenaufwendigsten sind Analyse, Modellierung und Simulation mit einem typischen Anteil von etwa 80% am Gesamtaufwand. Produktivitätssteigerungen wirken sich in diesem Bereich deshalb besonders stark aus.

Analyse, Modellierung und Simulation erfordern noch nicht die Erzeugung von üblichen Design-Artefakten wie z.B. UML-Diagrammen. Der Schwerpunkt liegt auf der fachlichen Kenntnis der Materie, wobei allerdings auch ein fundiertes Vorstellungsvermögen darüber, wie Geschäftsprozesse systemgestützt ausgeführt werden können, wichtig ist. In der Rolle des „Business Analyst“, die dieses Aufgaben- und Verantwortlichkeitsprofil ausfüllt, kommt auch eine verbindende Rolle zwischen Fachbereich und IT zum Ausdruck.

Ein Business Analyst ist in der Regel Mitarbeiter eines Fachbereichs, zumindest wenn der Fachbereich eine gewisse Personalstärke aufweist. Er verfügt über einen starken betriebswirtschaftlichen Hintergrund, ist aber auch mit Analyse- und Design-Techniken vertraut und kann auf diese Weise als Bindeglied zwischen Fachbereich und IT fungieren. Der Business Analyst nimmt, gemessen am Risiko und an den Auswirkungen von Fehlentscheidungen, die insgesamt wohl kritischste Rolle im Software-Entwicklungsprozess ein.

2 Interne und „öffentliche“ Geschäftsprozesse

Die wohl grösste Hebelwirkung wird erzielt, wenn es einem Unternehmen gelingt, seine Geschäftsprozesse wirkungsvoll zu automatisieren. Dabei kann jedoch nicht nur die Automatisierung interner Geschäftsprozesse das Ziel sein. Eine ganz wesentliche Herausforderung besteht darin, externe Geschäftspartner in die Wertschöpfungsprozesse mit einzubeziehen. Dazu wird eine „normierte Kupplung“ benötigt, die die internen (privaten) Geschäftsprozesse der beteiligten Geschäftspartner miteinander verknüpfen kann.

Angewandt auf die Kopplung von privaten Geschäftsprozessen beschreibt die „normierte Kupplung“ einen gesteuerten Nachrichtenaustausch zwischen Geschäftspartnern. Dieser Nachrichtenaustausch ist dem bisherigen papiergestützten Verfahren nachempfunden und auf die Möglichkeiten des automatisierten elektronischen Zusammenwirkens abgebildet. Sendet beispielsweise ein Besteller einen Auftrag in Form eines elektronischen Geschäftsdokuments (elektronisches Bestellformular), antwortet der Lieferant mit einer Auftragsbestätigung oder mit einer Ablehnung. Die Art und Weise der Kollaboration zwischen Geschäftspartnern wird auch als „öffentlicher Prozess“ bezeichnet, da die Art und Weise des Zusammenwirkens allen beteiligten Partnern offen liegt.

Um Geschäftsdokumente (Content) zwischen Geschäftspartnern, gesteuert durch Kollaborationsdefinitionen, austauschen zu können, ist ein Bindeglied erforderlich, das gewissermaßen den Briefumschlag (Envelope) zum Versand zur Verfügung stellt. Dieses Bindeglied wird durch einen Nachrichtendienst (Message Service) bereit gestellt.

Ein Geschäftsprozessmanagementsystem (Business Process Management System, BPMS) stellt die Verbindung zwischen „öffentlichen“ und „privaten“ Geschäftsprozessen her. Jeder Geschäftspartner ist in der Wahl der BPMS-Software frei.

Der Zusammenhang wird aus der folgenden Abbildung deutlich.

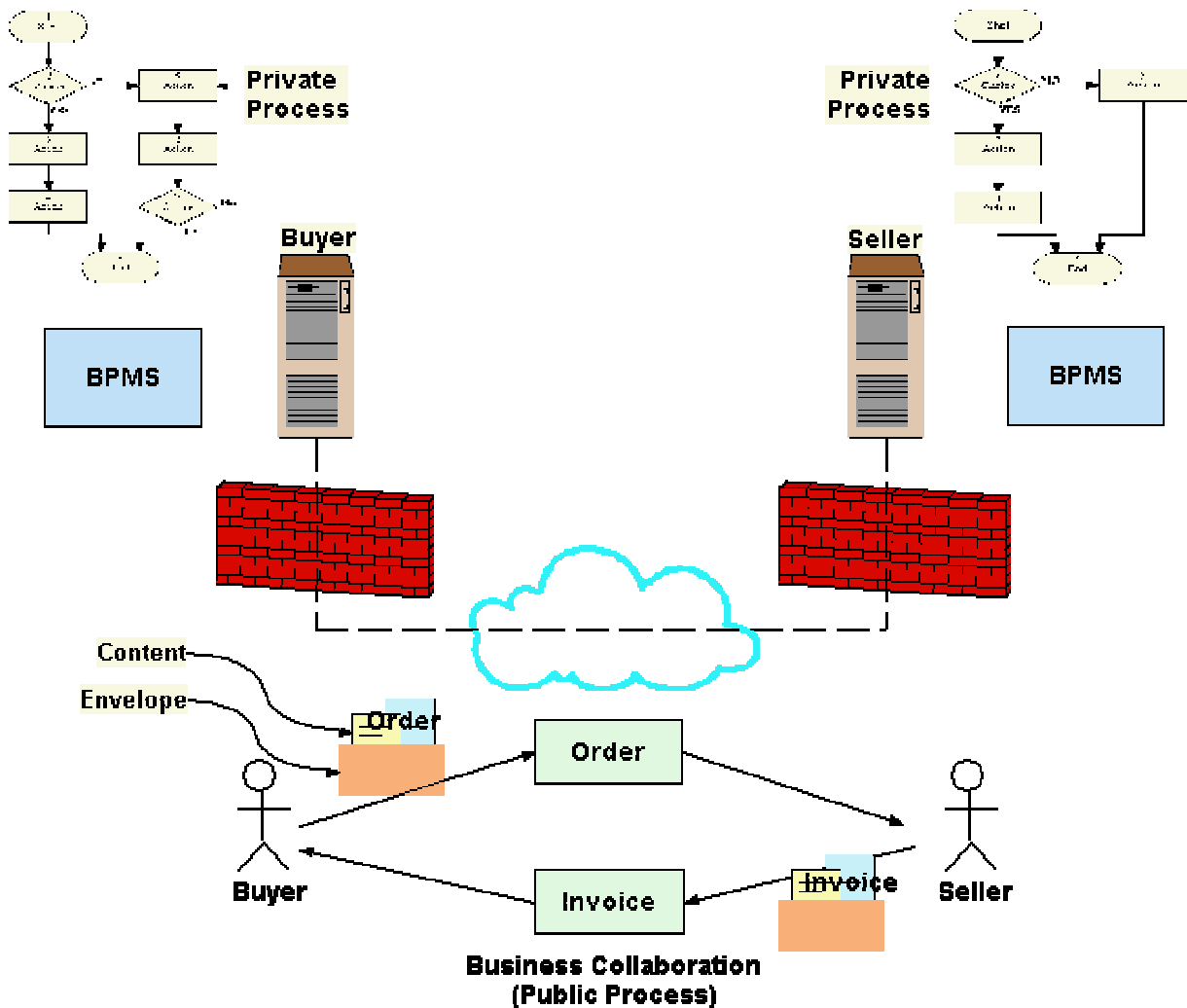


Abbildung 1: Verknüpfung interner Prozesse über einen "öffentlichen" Prozess

2.1 Modellierung privater Geschäftsprozesse

Die Geschäftsprozessmodellierung ist das geeignete Mittel, die Sprachbarriere zwischen IT und Fachbereichen zu überbrücken. Geschäftsprozessmodelle verkörpern die fachlichen und funktionalen Anforderungen, sind für die Fachbereiche leicht zu verstehen und eignen sich gleichzeitig als Ankerpunkt für die Software-Entwicklung. Geschäftsprozessmodelle in grafischer Form legen betriebliche Abläufe offen und bilden eine gemeinsame Verständigungsbasis für Fachbereiche und IT. Optimierungspotenzial wird frühzeitig erkennbar. Durch Verbesserung und Optimierung von Geschäftsprozessen können sehr schnell Schwachstellen beseitigt werden und ein konkreter Nutzen ist nachweisbar.

Semantisch reichhaltige Geschäftsprozessmodellierung umfasst die folgenden Aspekte:

- Geschäftsprozessmodelle erfordern eine durchdachte, leicht verständliche Syntax und Semantik. Es muss möglich sein, ein Modell auf Vollständigkeit und Konsistenz zu prüfen;
- Aktivitäten (Prozessschritte, Aufgabe, Funktion), neben Ereignissen und logischen Konnektoren eine der drei Basiselemente der Geschäftsprozessmodellierung, entsprechen Use Cases und müssen als solche in der üblichen Form (Vor- und Nachbe-

dingungen, Invarianten, erwarteter Ausführungsfluss usw.) beschrieben werden können;

- Aktivitäten müssen jeweils mit einer Rolle (z.B. Einkaufs-Sachbearbeiter) assoziiert werden können, die die Aktivität ausführt;
- Aktivitäten müssen jeweils mit ein oder mehreren Geschäftsobjekten assoziiert werden können, die von der Aktivität konsumiert und/oder manipuliert werden;
- Ereignisse (Events) beschreiben Zustände vor und/oder nach Ausführung von Aktivitäten;
- Logische Konnektoren verbinden Aktivitäten und Ereignisse miteinander.

Darüber hinaus müssen Ausnahmebedingungen (exceptions) berücksichtigt werden, die zu jeder Zeit auftreten können. Eine solche Ausnahmebedingung wäre beispielsweise die abnormale Beendigung einer Geschäftsaktivität. Als Folge einer Ausnahmebedingung müssen möglicherweise auch bereits korrekt durchgeführte Transaktionen wieder in ihrer Wirkung kompensiert werden. Wenn beispielsweise eine Reise gebucht werden soll, jedoch kein Hotelzimmer verfügbar ist, muss der zuvor gebuchte Flug wieder storniert werden.

2.2 Modellierung öffentlicher Geschäftsprozesse (Kollaborationen)

Die Kopplung der internen Geschäftsprozesse der miteinander zusammenwirkenden Geschäftspartner geschieht mit Hilfe von Kollaborationsdefinitionen (business collaborations). Dabei handelt es sich um einen definierten Zusammenhang von einzelnen Geschäftstransaktionen, die wiederum einen bestimmten Nachrichtenfluss definieren. Die beteiligten Geschäftspartner verständigen sich auf gemeinsame Kollaborationsdefinitionen, die die erwarteten Aktionen und Reaktionen der beteiligten Geschäftspartner festlegen. Es mag beispielsweise zwischen zwei Geschäftspartnern vereinbart sein, dass der Auftragnehmer innerhalb eines bestimmten Zeitraums nach Bestelleingang eine Auftragsbestätigung schickt.

Kollaborationsdefinitionen wirken als normierte Kupplung zwischen internen Geschäftsprozessen. Sie formalisieren Vereinbarungen des Geschäftsverkehrs in elektronischer Form. Kollaborationsdefinitionen werden auch als „öffentliche“ Prozesse (public processes) bezeichnet, da sie im Gegensatz zu den internen („privaten“) Geschäftsprozessen für alle beteiligten Geschäftspartner sichtbar sind. Allerdings unterscheiden sich Kollaborationsdefinitionen semantisch deutlich von „privaten“ Prozessen. Während Kollaborationsdefinitionen einen gesteuerten Nachrichtenaustausch zwischen Geschäftspartnern im Kontext von Geschäftstransaktionen definieren, beschreiben „private“ Geschäftsprozesse eine gesteuerte Abfolge von Aktivitäten. Kollaborationsdefinitio

aktuellen Entwicklungen lassen jedoch darauf hoffen, dem Ziel wieder etwas näher zu kommen.

Damit die Verständnisbrücke zwischen Fachbereich und IT geschlagen werden kann, müssen Beschreibungsmittel verfügbar sein, die den Ansprüchen sowohl des Fachbereichs als auch des IT-Bereichs genügen. Fachbereiche denken in Aufgaben und Prozessen, während IT-Fachleute in Diensten, Komponenten und Objekten denken.

Zur Modellierung von Geschäftsprozessen werden beispielsweise ereignisgesteuerte Prozessketten (EPKs) verwendet, die bestimmte Syntax und Semantik verkörpern. Bei der Modellierung entstehen Ergebnisse unterschiedlichen Typs, so z.B. Anwendungsfallbeschreibungen (Use Cases) als Repräsentanten von Geschäftsaktivitäten und Geschäftsobjektkandidaten. Es erweist sich als schwierig, relevante Arbeitsergebnisse (z.B. Geschäftsobjektkandidaten) in ein CASE-Werkzeug zu überführen. In der Konsequenz führt dies zu Doppelerfassung mit allen Konsequenzen wie z.B. der Gefahr, dass die Modelle im Zeitverlauf auseinander driften.

Die fehlende Brücke kann mit Hilfe von Ontologien geschlagen werden. Eine Ontologie beschreibt einen Wissensbereich (knowledge domain) mit Hilfe einer standardisierenden Terminologie sowie Beziehungen und ggf. Ableitungsregeln zwischen den dort definierten Begriffen. Die Assoziation von Ontologien mit künstlicher Intelligenz hat durchaus für viele IT-Fachleute eine höchst abschreckende Wirkung. Allerdings ist es nur von Vorteil, diese Abneigung zu überwinden.

Ontologien sind nicht wirklich etwas neues. Mit Berechtigung können auch Dictionaries, die schon seit den achtziger Jahren genutzt werden, als Vorläufer von Ontologien bezeichnet werden. Dictionaries dienen dazu, Datenelemente zu beschreiben und diese mit Datenbehältern (Dateien, Datenbanken) zu assoziieren. Bedingt durch mangelnden Formalismus und mangelnde Disziplin der Entwickler war es jedoch schwierig, den konkreten Nutzen von Dictionaries nachzuweisen. Die fehlende Integration von Dictionary und Design- und Entwicklungswerkzeugen führte auch hier schon zu Synchronisationsproblemen.

Im Vergleich zu Dictionaries, aber auch Glossaren, weisen Ontologien einen höheren Detaillierungsgrad auf und sind konsequenter strukturiert. Damit sind die Voraussetzungen geschaffen, um aus Ontologien IT-gestützt eine Vielzahl von „Produkten“ automatisiert ableiten zu können. Zu diesen „Produkten“ zählen beispielsweise Geschäftsobjektdefinitionen in UML-Syntax.

Eine Ontologie hat auch mit Datenbankschemata und UML-Klassendiagrammen viele Ähnlichkeiten. Es geht jedoch nicht darum, Objekttypen und deren Beziehungen zu modellieren und zu optimieren (z.B. Datennormalisierung). Vielmehr steht im Mittelpunkt, ein gemeinsames Verständnis von existierenden Dingen und deren Beziehungen zu erzielen, wobei die Nähe zur natürlichen Sprache ein besonders wichtiges Kriterium darstellt.

Die Notwendigkeit, die Produktivität der Software-Entwicklung stetig steigern zu müssen, lässt eine erneute Beschäftigung mit Ontologien als sinnvoll erscheinen. Der Nutzen einer Ontologie besteht schliesslich darin, dass eine Brücke zwischen der Welt der natürlichen Sprache und der Welt der Maschine geschlagen werden kann, oder, anders ausgedrückt, eine Brücke zwischen Fachbereich und IT-Bereich geschlagen werden kann.

In vielen Unternehmen werden Geschäftsprozesse und Geschäftsaktivitäten in Organisationshandbüchern beschrieben. Jedes Unternehmen, das über ein Qualitäts-Management entsprechend der ISO-Norm (ISO 900x) verfügt, muss seine Geschäftsprozesse beschreiben. Es liegt nahe, die Beschreibung so zu strukturieren und zu formalisieren, dass daraus

die weitgehende Generierung von ausführbaren Geschäftsprozessen möglich wird. Eine Ontologie ist ein geeignetes Mittel, dieses Vorhaben wirkungsvoll zu unterstützen.

Eine Ontologie ist ein formales, deklaratives semantisches Modell, das mit unterschiedlicher Syntax ausgedrückt werden kann. So eignet sich beispielsweise auch die Unified Modeling Language (UML) zur Erstellung von Ontologien. UML-Klassendiagramme sind ein geeignetes Mittel, um Entitäten zu beschreiben und deren Beziehungen zu definieren.

Wenn zu den in Ontologien definierten Klassen Instanzen erzeugt werden, entsteht eine Wissensbasis (Knowledge Base). Auf diese Weise lässt sich beispielsweise die gesamte Aufbauorganisation eines Unternehmens abbilden. Für den Fachbereich stellt eine Wissensbasis eine geeignete Schnittstelle zum IT-Bereich dar, vergleichbar in etwa mit dem Bauplan des Architekten.

Ontologien können in ganz praktischer Weise den Software-Entwicklungsprozess unterstützen, die Arbeitsproduktivität steigern und das Risiko von Fehlentwicklungen deutlich reduzieren. Dies zeigt sich insbesondere bei der Geschäftsprozessmodellierung.

Um eine durchgängige Wertschöpfungskette im Software-Entwicklungsprozess realisieren zu können, müssen Geschäftsprozesse mit reichhaltiger Semantik beschrieben werden. Aus dieser für den Fachbereich gut interpretierbaren Beschreibung können dann Ergebnisse unterschiedlichen Typs (auch implementierungsbezogene Ergebnistypen) generiert werden.

Das folgende UML-Klassendiagramm zeigt den sachlichen Zusammenhang zwischen Geschäftsprozessmodell und Implementierung. Ob Geschäftsprozesse durch ein Business Process Management System (BPMS) bzw. ein Workflow Management System (WfMS) ausgeführt werden, ist zu diesem Zeitpunkt noch unerheblich.

Eine Geschäftsaktivität (Business Activity) wird durch einen Dienst (Service) implementiert. Ein Dienst kann mehrere Schnittstellen anbieten und durchaus mehrere Geschäftsaktivitäten implementieren.

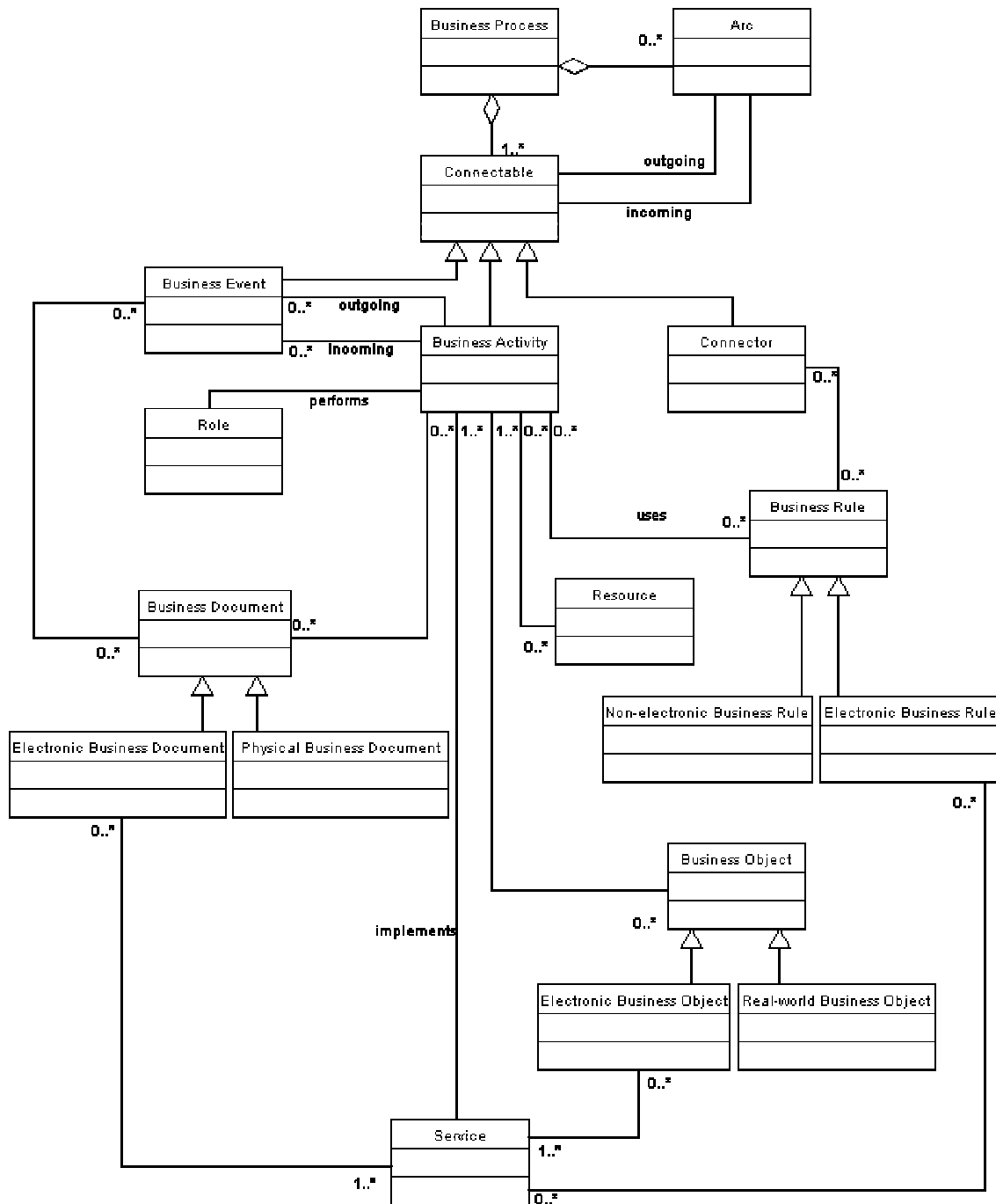


Abbildung 2: Sachlicher Zusammenhang zwischen Geschäftsprozessmodell und Implementierung

Die Anforderung besteht darin, das „Prozesswissen“ in einer Weise festzuhalten, die einen hohen Generierungsgrad ermöglicht. Zu diesem Zweck müssen bereits bei der Geschäftsprozessmodellierung Konzepte beschrieben werden, die eigentlich erst bei der objektorientierten Analyse relevant werden.

Die klassische Geschäftsprozessmodellierung fokussiert auf den Ausführungsfluss und die Beziehungen zwischen Geschäftsaktivitäten. Geschäftsobjekte wie z.B. Konto und Auftrag werden nicht explizit modelliert, geschweige denn strukturiert beschrieben. Dies wird den IT-

Fachleuten, den Modellierern, überlassen. Viel zu früh entstehen dann Klassendiagramme in UML-Notation, mit denen Nicht-IT-Experten ohne vorherige Schulung nichts mehr anfangen können. Die Abstraktionslücke ist bereits zu gross.

Um eine gemeinsame Verständigungsbasis zwischen Fachbereichen und dem IT-Bereich herzustellen, müssen beide einander entgegen kommen. Der Fachbereich tut gut daran, ein Bindeglied zum IT-Bereich in Form eines Business Analyst zu benennen (ist in vielen Unternehmen bereits geschehen) und der IT-Bereich tut gut daran, geeignetere Beschreibungsmittel anzubieten, die die Abstraktionslücke minimieren helfen, gleichzeitig aber maschineninterpretierbar sind. Ein solches Beschreibungsmittel sind Ontologien.

4 Integration mit dem Software-Entwicklungsprozess

Wenn Ontologien als ein gangbarer Weg gesehen werden, muss auch die Frage beantwortet werden, wie die Einbettung in den Software-Entwicklungsprozess geschehen kann. Primär stellt sich die Frage, mit welcher Sprache Ontologien erstellt und gepflegt werden sollen. Prinzipiell bieten sich dafür zum einen speziell für diesen Zweck entwickelte Ontologie-Sprachen an, zum anderen ist aber genauso denkbar, eine vorhandene OO-Modellierungssprache zu nutzen.

Spezifische Ontologie-Sprachen werden z.B. durch DARPA DAML (DARPA Agent Markup Language) und die Web Ontology Language (OWL) repräsentiert. Die vorherrschende OO-Modellierungssprache ist zweifellos UML (Unified Modeling Language). Da UML im Software-Entwicklungsprozess tief verankert ist und UML-Klassendiagramme eine zentrale Rolle spielen, stellt sich die Frage, ob überhaupt eine spezifische Ontologie-Sprache in Erwägung gezogen werden soll. Beide Ansätze haben Vor- und Nachteile.

4.1 Ontologie-Sprache

Ontologie-Sprachen sind von vornherein nicht für die Modellierung objektorientierter Systeme vorgesehen. Zwar können Klassen und deren Beziehungen dargestellt werden, jedoch beschränkt sich die Beschreibung einer Klasse auf die Eigenschaften (Attribute) eines Objekttyps. Das Verhalten von Objekttypen (Methoden) wird nicht beschrieben. Auch die Unterscheidung zwischen Objektklasse und Schnittstelle (Interface) ist Ontologie-Sprachen unbekannt.

Beim Mapping in UML werden Instanzen nicht in das UML-Klassendiagramm übernommen. Diese Einschränkung kann ohne weiteres hingenommen werden, da die Ontologie als Verständigungsbrücke zwischen Fachbereich(en) und IT dient und in Klassendiagrammen lediglich der strukturelle Aspekt im Vordergrund steht.

Als schwerwiegender erweist sich jedoch die fehlende Möglichkeit zur Beschreibung von Verhalten (behavior). Ontologien sind rein deklarativ und beschreiben Strukturen. Auf den Vorteil der Objektorientierung, die Beschreibung von Eigenschaften und Verhalten, wird verzichtet. Auf der anderen Seite sind Ontologien weitaus freier und mächtiger, weil noch keinerlei Rücksicht auf Funktionen genommen werden muss.

In der Konsequenz bedeutet dies, dass zwar UML-Klassendiagramme aus Ontologien erzeugt werden können, die einzelnen Klassenbeschreibungen jedoch unvollständig sind. Klassen und deren Beziehungen untereinander werden dargestellt, ebenso auch Attribute, jedoch fehlen die Methoden. Der Wert einer Ontologie geht dadurch zwar nicht verloren, sie kann jedoch nicht als Ausgangsbasis für die Software-Entwicklung fungieren.

4.2 UML

Es kann es nur sinnvoll sein, die Anzahl der Modellierungssprachen auf das absolute Minimum zu begrenzen, um den Ausbildungsaufwand in sinnvollen Grenzen zu halten. Andererseits muss jedoch eine Modellierungssprache auch die erforderliche Ausdruckskraft bereitstellen und „skalierbar“ sein. Im Optimalfall ist sie sowohl für den Business Analyst verwendbar, der seine Modelle mit der Fachabteilung abstimmen muss, als auch für den OO-Designer. UML ist zwar aufgrund der breiten Verankerung der aussichtsreichste Kandidat, hat jedoch einige Schwächen. Die Eigenschaften von Systemen basierend auf der dienstorientierten Architektur (SOA) können mit UML 1.x nur unzureichend dargestellt werden. Dennoch sprechen einige Gründe wie etwa die Erweiterungsfähigkeit dafür, UML auch für die Entwicklung von Ontologien vorzusehen. Dabei helfen insbesondere Stereotypen, benutzerdefinierte Eigenschaften (tagged values) sowie Einschränkungen/Zwangsbedingungen (Constraints).

Stereotypen sind hilfreich, um die Semantik von Modellen zu präzisieren. Ein Stereotyp spezialisiert eine modellierte Klasse (Beispiel: *Persistent Class* als Spezialisierung von *Class*) Mit Stereotypen kann jedoch nicht die Struktur von Klassen des UML-Metamodells erweitert werden.

Vorhandene Klassen können um benutzerdefinierte Eigenschaften (tagged values) erweitert werden. Auch für selbst definierte Stereotypen kann man beliebig viele benutzerdefinierte Eigenschaften definieren.

Mit Stereotypen und benutzerdefinierten Eigenschaften können statische und dynamische Modelle mit Semantik angereichert werden. Es ist möglich, sich Ausdrucksmittel zu schaffen, die UML im Kernumfang nicht kennt. Sowohl Stereotypen als auch benutzerdefinierte Eigenschaften sind für die Code-Generierung wichtige und mächtige Instrumente um die Produktivität ganz erheblich zu steigern. Mit Hilfe der Klassifizierung von Modellelementen kann die Code-Generierung wirkungsvoll gesteuert werden. Für jeden Stereotyp kann die Code-Generierung nach spezifischen Regeln erfolgen. Selbstverständlich kann ein Generator aus einem Stereotyp auch mehrere Ergebnisse generieren, um z.B. gleichzeitig plattformspezifischen Code für mehrere unterschiedliche Plattformen zu erzeugen.

Einschränkungen/Zwangsbedingungen (Constraints) schränken die Verwendung von stereotypisierten Metaklassen (UML-Metamodell) und benutzerdefinierten Eigenschaften (tagged values) ein. Sie werden mit einer in die UML integrierten Sprache, der Object Constraint Language (OCL), beschrieben. Constraints sind für jedes UML-Modell überprüfbar. Auf Basis eines definierten UML-Profiles können somit UML-Modelle automatisiert auf unerlaubte Konstrukte geprüft werden.

Mit Hilfe von UML-Profilen, einem bislang wenig beachteten und genutzten Entwurfselement der UML, können „Schablonen“ für den Entwurf von Modellen in bestimmten, abgegrenzten Anwendungsbereichen definiert werden. Das Ziel von UML-Profilen besteht darin, die durch die UML beschriebenen Möglichkeiten bewusst zu beschränken und auf bestimmte Anwendungsbereiche anzupassen. Kernelemente von UML-Profilen sind somit Stereotypen, benutzerdefinierte Eigenschaften (tagged values) und Einschränkungen/Zwangsbedingungen (constraints).

Um UML für bestimmte Aufgaben anpassen zu können, werden UML-Profile definiert, so z.B. für die Geschäftsprozessmodellierung. In gleicher Weise kann natürlich auch ein UML-Profil für Ontologien definiert werden.

Damit aus UML-Modellen tatsächlich ein Grossteil des Anwendungs-Code generiert werden kann, muss die Semantik von UML-Modellierungselementen reichhaltiger und präziser werden. Dann wäre es sogar theoretisch möglich, zur Ausführungszeit aus UML-Diagrammen ausführbaren Code zu generieren. Dies dürfte jedoch in der Tat eine theoretische Möglichkeit bleiben.

Wenn sich UML zu einer mehr formalen Sprache entwickelt, sind auch die Voraussetzungen geschaffen, UML für die Entwicklung von Ontologien nutzen zu können. Schliesslich hat die Semantik von Modellen umso mehr Gehalt, je geringer die Anzahl möglicher Interpretationen ist. Mit UML 2.0 wird jedenfalls ein bedeutender Schritt in diese Richtung vollzogen. Als besonderer Vorteil erweist sich des weiteren, dass UML als offener Standard bereits breit verankert ist und unter der Ägide der Object Management Group (OMG) weiter entwickelt wird. UML wird heute von nahezu allen CASE-Werkzeugen unterstützt, wenn auch in unterschiedlichem Umfang. Es würde sich als Akzeptanzhürde erweisen, wenn Designer zunächst den Umgang mit einem zusätzlichen, spezialisierten Werkzeug wie z.B. Protégé erlernen müssten.

Wie bereits erwähnt, sind spezialisierte Sprachen wie z.B. DARPA DAML (DARPA Agent Markup Language) und die Web Ontology Language (OWL) von vornherein besser für die Erstellung von Ontologien geeignet. Allerdings muss dann in Kauf genommen werden, dass die Überführung der Ergebnisse in UML nur mit Einschränkungen möglich ist und ein Reibungsverlust auftritt. Als Alternative bietet sich deshalb an, sich von vorn herein auf UML zu beschränken und die Sprachmittel von UML voll auszunutzen. Eine OWL-konforme Ontologie kann dann mit Hilfe eines Generators erzeugt werden. Voraussichtlich wird es Werkzeuge geben, mit denen zumindest der Standard-Anwendungsfall, die Erzeugung von OWL aus UML, abgedeckt werden kann.

4.3 Konvergenz – aber wie?

Es kann nur sinnvoll sein, beide Ansätze sinnvoll miteinander zu verbinden. Dann kann das Ziel, Anwendungs-Software aus einer Wissensbasis heraus zu generieren, tatsächlich erreicht werden. Dieser Ansatz ist vom Ziel her deckungsgleich mit dem der Model-driven Architecture (MDA). Auf natürliche Weise entsteht so ein plattformneutrales Modell, das dann über Generierungs-Skripts in ein oder mehrere plattformspezifische Modelle überführt werden kann. Der Wissensbasis-getriebene Ansatz ist jedoch noch umfassender, da die inhärente Fokussierung der MDA auf UML keine Rolle spielt.

Um eine durchgängige Wertschöpfungskette möglich zu machen, können Knowledge Base-Werkzeuge funktional erweitert werden. Zu einer Klasse können dann nicht nur Slots (Attribute) definiert werden, sondern auch Methoden. Natürlich muss dazu auch das zugrunde liegende Metamodell erweitert werden. Dies ist problemlos möglich. Demonstrieren lässt sich dies am Beispiel des Werkzeugs Protégé, das von der Stanford University School of Medicine entwickelt wurde. Protégé ist ein Ontologie- und Wissensbasis-Editor, dessen Entwicklung Mitte der neunziger Jahre begonnen wurde und der mittlerweile in der Version 1.8 (Beta) vorliegt.

Mit Protégé ist es möglich, UML-Artefakte, die mit einem CASE-Werkzeug (z.B. Rational Rose, Poseidon) erzeugt wurden, in Protégé zu importieren. Als Austauschformat dient XMI (XML Metadata Interchange), das von den meisten CASE-Werkzeugen entweder in den Versionen 1.1 oder 1.2 unterstützt wird. Ebenso können mit Protégé UML-Artefakte (Klassendiagramme) über ein Plug-in erzeugt und anschliessend in ein CASE-Werkzeug zur weiteren Bearbeitung importiert werden. Somit ist ein primitives, jedoch durchaus wirkungsvolles Roundtrip-Engineering möglich.

Das folgende Beispiel zeigt einen kleinen Ausschnitt einer Geschäftsobjektontologie. Der Export in UML ist problemlos möglich. Anschliessend können die Klassendefinitionen in ein CASE-Werkzeug zur weiteren Bearbeitung importiert werden.

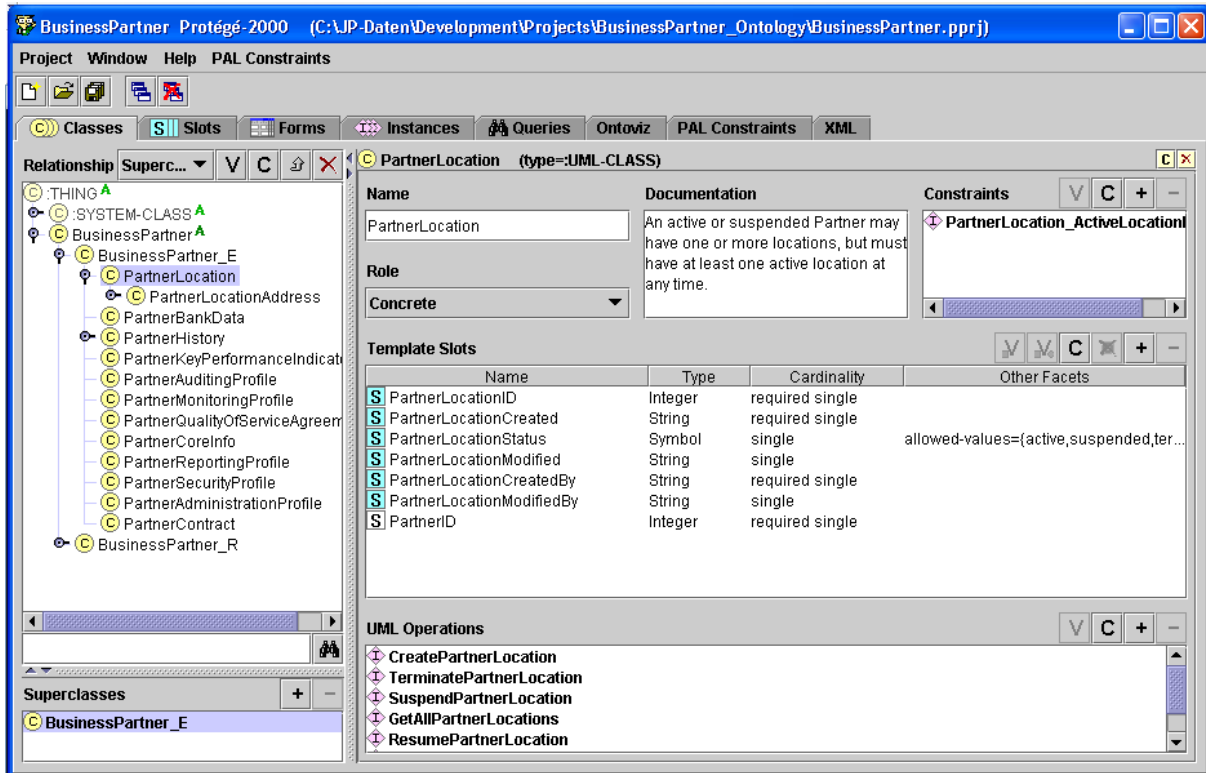


Abbildung 3: Bearbeitung einer Ontologie mit dem Werkzeug Protégé

Da die Metamodelle von Wissensbasis-Werkzeugen und UML nicht deckungsgleich sind, muss ein Informationsverlust bei Import und Export hingenommen werden. Der Informationsverlust kann jedoch minimiert werden, indem von vornherein einige Konventionen beachtet werden.

Ein anderer Ansatz verzichtet auf die Generierung von UML-Klassendiagrammen, um direkt aus Ontologien Programm-Code (z.B. in C# oder Java) zu generieren. Für jede Ontologie-Klasse könnte beispielsweise eine abstrakte Java-Klasse generiert werden, die alle Attribute und, soweit möglich und sinnvoll, Semantik enthält (z.B. `public abstract class AbstractPerson`). Ausserdem wird eine leere zweite Klasse erzeugt (z.B. `public class Person extends AbstractPerson`), sofern diese nicht bereits existiert. Der Software-Entwickler kann dann für die nicht-abstrakte (implementierende) Klasse Methoden definieren. Diese implementierende Klasse wird dann bei der nächsten Generierung nicht überschrieben. Bei diesem Ansatz wäre auch ein Roundtrip-Engineering möglich. Als Vorteil dieser Methode erweist sich, dass auf ein CASE-Werkzeug verzichtet werden kann.

Wie bereits erwähnt, erweist sich bei Ontologien die fehlende Möglichkeit zur Beschreibung von Verhalten als das grösste Defizit. Ein Blick auf das Spektrum möglicher Anwendungsfälle im Bereich Geschäftsprozessmodellierung zeigt jedoch, dass eine Verhaltensbeschreibung nicht immer notwendig ist. Ein Beispiel ist das Geschäftsprozessmodell selbst. Hier stehen der Kontrollfluss und die Beziehungen zwischen Geschäftsaktivitäten, Rollen, Geschäftsdokumenten und Geschäftsobjekten im Mittelpunkt. Das Verhalten spielt hier keine explizite Rolle.

5 Ontologien auf unterschiedlichen Ebenen

Ontologien können auf unterschiedliche Perspektiven bezogen sein. Eine fachbereichsbezogene Ontologie würde Begriffe, Entitäten und Konzepte definieren, die für einen bestimmten Fachbereich relevant sind. Die konzeptionelle Nähe zum Organisationshandbuch ist evident.

Eine Ontologie auf Branchenebene würde einen Beitrag dazu leisten, das Auslagern bzw. (Wieder)einlagern von Geschäftsprozessen zu erleichtern, indem semantische Inkompatibilitäten vermieden werden. Eine Branchen-Ontologie würde Begriffe und Entitäten sowie deren Beziehungen untereinander definieren.

Die oberste Ontologieebene würde durch eine branchenneutrale Ontologie repräsentiert. Hier würden z.B. Personen und ihre unterschiedlichen Rollen definiert. Ein Anwendungsbeispiel wäre die Entität „natürliche Person“ und ihre Beziehung zu „Rolle“. Eine natürliche Person kann beispielsweise die Rolle „Gast“ in einem Hotel übernehmen, die Rolle „Passagier“ bei einem Transportdienstleister und die Rolle „Klient“ bei einem Anwalt. Weitreichende Ansätze für branchenneutrale Ontologien existieren bereits. So verkörpern beispielsweise ebXML und RosettaNet Ontologien.

Branchenspezifische Ontologien existieren bereits in Teilbereichen, auch wenn diese nicht als solche bezeichnet werden. Ein Beispiel wäre etwa eine branchenspezifische Produktklassifikation. Der Bedarf, diese Ansätze zu umfassenden Ontologien zu erweitern, wird derzeit abgesehen von Einzelfällen nicht gesehen. Derartige Ontologien müssten von Branchenverbänden bzw. von internationalen Gremien wie z.B. ISO oder IEEE vorangetrieben werden. Solche Aktivitäten sind derzeit bereits erkennbar (siehe IEEE Standard Upper Ontology, SUO, zur Definition von Grundkonzepten, auf denen andere Ontologien gründen).

Für das Anwenderunternehmen haben natürlich Ontologien, die das eigene Geschäft betreffen, die grösste Bedeutung. Auch hier lässt sich die perspektivbezogene Betrachtungsweise anwenden. Die folgenden Perspektiven sind besonders wichtig, wobei eine Ontologie sowohl Terminologie als auch Konzepte definiert:

- Rollenontologie: Geschäftsaktivitäten können von Personen ausgeführt werden, die ein oder mehrere Rollen übernehmen können. Im Kontext einer bestimmten Geschäftsaktivität übernimmt eine bestimmte Rolle (z.B. Einkaufssachbearbeiter) deren Ausführung;
- Prozessontologie: beschreibt die Eingaben und Ausgaben eines oder mehrerer Geschäftsprozesse, die Geschäftsaktivitäten und deren Beziehungen (einschl. der Ausführungsfolge) und Abhängigkeiten, sowie Einschränkungen und Zwangsbedingungen;
- Geschäftsaktivitätontologie: beschreibt die Eingaben und Ausgaben, die Beziehung zur Rolle (Person oder System), die die Aktivität ausführt, die Beziehungen zu Geschäftsobjekten, sowie Einschränkungen und Zwangsbedingungen;
- Geschäftsobjektontologie: beschreibt die Eingaben und Ausgaben sowie Einschränkungen und Zwangsbedingungen.

Geschäftsregeln können mit einem Geschäftsprozess, einem Geschäftsobjekt oder einem Dienst assoziiert sein. Sie werden im jeweiligen Kontext auf Basis eines einheitlichen Pattern modelliert.

Eine Ontologie berücksichtigt generell neben den fachlichen Anforderungen auch Anforderungen im Hinblick auf Sicherheit, Management, sowie Quality of Service und Quality of Protection.

6 Nutzenaspekte von Ontologien

Der besondere Vorteil von Ontologien liegt in der plattform- und weitestgehend sprachenneutralen Beschreibung von Sachverhalten. Die einzige Bindung wird entweder an UML oder an eine spezifische Ontologiesprache vollzogen.

Aus einer Ontologie können die verschiedensten Artefakte durch Generierung erzeugt werden. Eine entsprechend mächtige Ontologie vorausgesetzt, ist es beispielsweise möglich, ausführbare Geschäftsprozessdefinitionen zu generieren. Aus einer Ontologie können mit unterschiedlichen Generierungsskripten durchaus Geschäftsprozessdefinitionen in unterschiedlichen Repräsentationen (z.B. XPDL, BPEL4WS, WSCI) erzeugt werden. Herstellerbindungen verlieren dadurch einen Grossteil ihrer Wirkung.

Für Artefakte wie z.B. Geschäftsobjekte und Dienste (als Aktivitätenimplementierung) können zumindest Rahmen generiert werden. Dabei ist nicht an die Code-Generierung gedacht, sondern an die Modellgenerierung. Aus einer Ontologie würde ein UML-Klassendiagramm in XMI-Repräsentation generiert werden, das dann mit einem UML-Werkzeug importiert und weiterbearbeitet werden kann. In einer Ontologie können ein oder mehrere Entwurfsmuster (Patterns) definiert sein, die bei der Generierung entsprechend umgesetzt werden können. Damit wäre der Anschluss geschaffen, um mit bereits vorhandenen Generatoren im nächsten Schritt dann plattformspezifischen Code erzeugen zu können. Die mit der Model-driven Architecture (MDA) propagierte Trennung von plattformneutralem und plattformspezifischem Modell ist eingehalten.

Ein wesentlicher Nutzenaspekt besteht darin, dass unternehmensspezifische Namenskonventionen bereits zu einem sehr frühen Zeitpunkt berücksichtigt werden können. Darüber hinaus ist es einfacher, Redundanzen und Mehrdeutigkeiten frühzeitig zu erkennen. Je früher Geschäftsobjekte und Attribute überprüft werden können, desto besser.

Der wohl grösste Nutzen liegt jedoch in der geschaffenen Verständigungsbrücke zwischen Fachbereich und IT-Bereich. Zwar handelt es sich bei einer Ontologie nicht um eine Beschreibung in natürlicher Sprache, jedoch stellt die Abstraktion andererseits auch kein wirkliches Problem dar. Der Abstraktionsgrad ist in etwa mit dem eines Bauplans für ein Gebäude vergleichbar.

Mächtige Werkzeuge vorausgesetzt wird es möglich sein, die Produktivität der Software-Entwicklung deutlich zu steigern. Auch wenn noch keine Erfahrungswerte aus abgeschlossenen komplexen Projekten vorliegen, kann dennoch ein Produktivitätsschub in einem Korridor zwischen 20 und 30% prognostiziert werden.

7 Werkzeuge

Wie so häufig stellt man fest, dass gerade das Werkzeug das man dringend braucht, noch nicht verfügbar ist. Es ist heute schlichtweg unmöglich, ein Werkzeug zu finden, das sämtliche wichtigen Anforderungen erfüllt. Eine grafische Repräsentation ist als Verständigungsbasis mit Fachbereichen unabdingbar. Einige Werkzeuge im akademischen Bereich unterstützen keine grafische Repräsentation, sondern machen die Interpretation von Code not-

wendig. Dieser ist für Mitarbeiter von Fachbereichen nicht leicht lesbar und deshalb in Review-Sitzungen auch nicht verwendbar.

Fast alle Werkzeuge zur Erstellung und Pflege von Ontologien und Wissensbasen haben ihre Wurzeln in Künstliche Intelligenz-Projekten. Die weitaus meisten der durchaus zahlreichen Werkzeuge haben experimentellen Charakter und sind für den Einsatz für die Software-Entwicklung im betrieblichen Umfeld nicht ausgereift genug. Viele Produkte wird im Rahmen von Open Source-Projekten entwickelt. Deshalb ist auch die Support-Frage bei den meisten Produkten ungeklärt.

Zu den wenigen Werkzeugen, die schon eine hinreichende Stabilität erreicht haben, zählt das bereits erwähnte Werkzeug Protégé. Es ist als freie Software unter der „Mozilla Public License“ verfügbar und kann als das wohl grösste Open-Source System zur Ontologie/Wissensmodellierung mit ueber 7000 registrierten Usern bezeichnet werden. Mittlerweile sind über 40 Plug-ins verfügbar, so unter anderem auch zur Erzeugung von UML und XMI (siehe oben). Zur Jahresmitte soll die Web Ontology Language (OWL) über ein Plug-in unterstützt werden, um auch Kompatibilität mit dieser Ontologie-Sprache zu erreichen.

Neben den Werkzeugen, die ihre Wurzeln im Bereich Künstliche Intelligenz haben, gibt es auch vereinzelt Software-Hersteller, die einen UML-orientierten Ansatz verfolgen. Diese Pioniere arbeiten an Werkzeugen, die auf bereits etablierte UML-Werkzeuge wie z.B. Rational Rose aufsetzen und diese erweitern. Dieser Ansatz hat den grossen Vorteil, dass die technische Infrastruktur des Basiswerkzeugs genutzt werden kann. Funktionalität wie z.B. Konfigurations-Management, muss nicht separat entwickelt werden.

Zwischen den spezifischen Ontologiesprachen wie z.B. DAML und OWL und der UML bestehen einige semantische Unterschiede. So erfordern z.B. Eigenschaften mit Beziehungscharakter in DAML bzw. OWL nicht notwendigerweise eine Beziehung zu einer anderen Klasse. Wenn beispielsweise ausgedrückt werden soll, dass eine Rechnung eine Rechnungsanschrift enthält und dass ein Auftrag eine Lieferanschrift enthält, so würde es sich in UML um zwei Assoziationen handeln. In einer Ontologie oder auch einem RDF-Schema würde nur die Eigenschaft (Property) „enthält“ zu definieren sein.

Spezifische Ontologiesprachen lassen auch die Vermischung von Klassen und Instanzen zu. Die Möglichkeit, dass eine Klasse somit Instanz einer anderen Klasse sein kann, entspricht nicht der Philosophie objektorientierter Systeme. Hier wird strikt zwischen Klassen und Instanzen unterschieden.

Auch wenn UML an sich die für die Definition von Ontologien erforderlichen Sprachkonstrukte unterstützt, bedeutet dies nicht, dass auch alle UML-Werkzeuge sämtliche UML-Konstrukte unterstützen. In der Tat unterstützen derzeit einige UML-Werkzeuge einige der im Ontologiekontext wichtigen UML-Konstrukte nicht. Dazu zählen z.B. n-ary associations und association inheritance.

Trotz mancher Defizite ist jedoch zu erwarten, dass sich UML auch für die Definition von Ontologien stärker etablieren wird. Der bereits ansehnlichen Basis an IT-Fachleuten, die mit UML gut umgehen können steht eine relativ kleine Gruppe von Fachleuten gegenüber, die mit spezifischen Ontologiesprachen vertraut sind. Darüber hinaus kann es nur vorteilhaft sein, wenn keine neuen Werkzeuge benötigt werden. Software-Hersteller, die entsprechende Generatoren zur Komplettierung der Wertschöpfungskette im Software-Prozess verfügbar machen, sind in wirtschaftlich schwierigen Zeiten eher zur Unterstützung von Mainstream-Technologien geneigt. Aus diesem Grund spricht manches dafür, UML zu präferieren. Die Generierung von DAML oder OWL aus UML ist schliesslich immer möglich.

8 Zusammenfassung

Nahezu lautlos vollzieht sich der nächste Evolutionsschritt: der Übergang zu einer höheren, mächtigeren Beschreibungsebene. Nicht mehr das objektorientierte System steht im Zentrum, sondern der Geschäftsprozess. Business Analysts werden in der Lage sein, Ontologien zu entwickeln, die dann als Quelle für die Code-Generierung dienen. Damit wird ein wichtiger Schritt auf dem Weg zur Beseitigung der „semantischen Lücke“ zwischen fachlicher Sicht und technischer Sicht unternommen.

Natürlich birgt der beschriebene Übergang keinen Automatismus in sich. Die Qualität einer Ontologie ist nach wie vor das Ergebnis sorgfältiger und zielgerichteter Arbeit. Der Grundsatz: „A fool with a tool is still a fool“ gilt nach wie vor. Im Zeitverlauf werden aber immer mehr generische Ontologien verfügbar werden, wobei Branchenverbände aber auch Beratungsunternehmen dafür geeignete Quellen sind. Diese Ontologien müssen dann nur noch um unternehmensspezifische Eigenheiten und Belange erweitert werden.

Diverse Ergebnisse wie z.B. Geschäftsprozessdefinitionen können direkt in unterschiedliche Zielsprachen generiert werden. Es ist beispielsweise möglich, ausführbare Geschäftsprozessdefinitionen in BPEL4WS, in WSCI oder in einer herstellereigenen Sprache zu generieren. Für Geschäftsobjekte und Dienste können zumindest Rahmen generiert werden, die dann von Entwicklern weiter bearbeitet werden können. In jedem Fall aber wird die Produktivität deutlich gesteigert werden können. Obwohl noch keine Erfahrungswerte aus der Praxis vorliegen, scheint die Erwartung in einem Korridor von 20-30% plausibel zu sein.