Vereinfachung der Ontologieerstellung durch Designmuster

Henriette Röger, Tobias Freudenreich

September 2014

Technische Universität Darmstadt
Fachbereich Informatik
Fachgebiet Datenbanken und Verteilte Systeme

1 Einleitung

Ein Unternehmen ist durch die Vielzahl der zugehörigen Personen, Prozesse, und Objekte ein komplexes Konstrukt. Diese Komplexität erfordert klare Vorgaben in Form von Regeln und definierte Abläufen, um gewinnbringend zu wirtschaften sowie rechtliche und unternehmensinterne Vorgaben einzuhalten. Beispielsweise muss sichergestellt sein, dass kritische Unternehmensbereiche nur von Personen mit einer Zugangsberechtigung betreten werden und dass erwirtschaftete Einkünfte nachvollziehbar dokumentiert sind.

Informationstechnische Systeme unterstützen bei der Einhaltung dieser Vorgaben. So ermöglicht es beispielsweise ein Lagerbestandssystem, bei Unterschreitung eines Mindestlagerbestands automatische Neubestellungen zu generieren. Dies verhindert Produktionsausfälle aufgrund von Mangel an Rohmaterialien.

Um ein Informationssytem für die Einhaltung von Vorgaben einzusetzen, benötigt dieses Informationen über das zu überprüfende Unternehmen. Diese Informationen stellen die digitale Abbildung des Unternehmens dar - sie kann das gesamte Unternehmen umfassen oder, wie im oben genannten Beispiel, nur einen Ausschnitt wie den aktuellen Lagerbestand.

Freudenreich et al. stellen ein Informationssytem vor, das komplexe Unternehmensabbildung als Kombination von Sensor- und herkömmlichen Datenquellen handhabbar macht, so dass es damit der Durchführung von Unternehmensprozessen und der Einhaltung von Vorgaben dienlich ist [2]. Die oben beschriebenen Vorgaben lassen sich dabei in Form von Policies deklarativ formulieren. Der deklarative Ansatz ermöglicht es Domänenexperten, ihr Wissen dem Informationssystem zur Verfügung zu stellen, ohne über ausgeprägte Expertise im Technologiebereich zu verfügen.

Policies vereinen eine Sammlung an feingranularen Regeln, die ein gemeinsames Ziel verfolgen. Beispielsweise liegen der automatische Nachbestellung von Rohstoffen (Ziel) die Regeln Lagerbestand des Materials unter Mindestgrenze, Material wird in den kommenden Wochen zur Produktion genutzt und Material ist beim Lieferanten verfügbar zu Grunde. Zur Formulierung wird die Declarative Policy Language genutzt (DPL) [1].

Diese so formulierten Vorgaben werden an eine Middleware weitergeleitet, die durch die Generierung und Aufrechterhaltung eines ereignisbasierten Systems deren Einhaltung unterstützt. Als Beispielpoliciy soll ein Alarm ausgelöst werden, wenn ein Serverraum aktiv ist und die Temperatur einen Maximalwert überschreitet. Die Middleware wertet die Sensordaten (Temperatursensoren) gemeinsam mit den Bedingungen (Serverraum aktiv? Temperatur über Maximalwert?) aus und iniitiert bei Bedarf den gewünschten Alarm.

Die Middleware benötigt ein Domänenmodell (Domain Model), das die digitale Abbildung des Unternehmens bildet. Dieses Domain Model wird als Hintergrundwissen genutzt, um aus den formulierten Policies ausführbaren Code zu generieren (Abbildung 1). Das System erfordert dabei eine Basisstruktur des Domain Models, die wir in Kapitel 2.5 detaillierter erläutern. Jedes Modellierungssystem, das zur Erstellung des Domain Models genutzt wird, muss diese Struktur abbilden können, damit das Model zur Nutzung mit der vorgestellten Middleware geeignet ist.

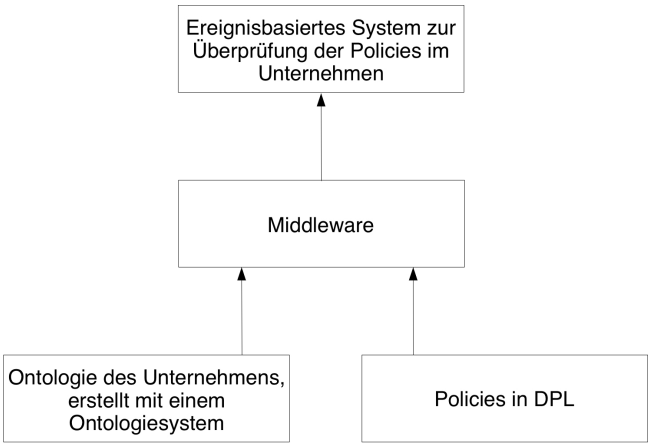

Abbildung 1: Zusammenspiel der Komponenten bei Einsatz der DPL

Da das Domain Model als Wissensgrundlage des Systems dient, unterstützt eine präzise, fehlerfreie und aussagekräftige Darstellung die Performanz des Gesamtsystems. Ein Möglichkeit, ein Domain Model zu realisieren, ist die Formulierung der Unternehmensdomäne in Form einer Ontologie [18].

Erstellt ein Modellierer die Ontologie eines Unternehmens, so ist es seine Aufgabe, die wesentlichen Aspekte des Unternehmens zu identifizieren und deren Beziehungen zueinander zu definieren [17]. Wesentlich sind dabei die Aspekte, die für die effektive Nutzung des vorgestellten Systems notwendig sind. Der Modellierer muss die geforderte Präzision, Fehlerfreiheit und Aussagekraft des Unternehmensmodels, dargestellt durch die Ontologie, stets berücksichtigen, um den bestmöglichen Einsatz der Middleware zu ermöglichen.

In dieser Arbeit entwickeln wir Ontolgie Designmuster, die die hochwertige Modellierung einer Ontologie vereinfachen, indem sie Modellierungslösungen für häufige Konstellationen von Unternehmenselementen bieten. Diese Muster repräsentieren Anordnungen der Subjekte, Objekte und Beziehungen des Unternehmens. Sie sind Muster in Form von Templates für die Modellierung des Unternehmens. Wir bezeichnen sie als Design Muster, da sie die Erstellung, also das Designen, der Ontologie unterstüzen und eine Parallele zu den Design Mustern des Software Engineerings sichtbar ist: Sie bieten einen Lösungskern für wiederkehrende Probleme, der immer wieder genutzt werden kann und gleichzeitig individuelle Lösungen ermöglicht. ([15] Seite 3). Der Modellierer der Ontologie erhält damit vorgefertigte Design Muster, die er für die Erstellung der Ontologie nutzt und somit eine hochwertige Ontologie erstellt, die die oben genannten Bedingungen erfüllt.

Ontologien werden mittels Ontologiessytemen erstellt [17]. Die von uns ermittelten Design Muster stellen somit auch Anforderungen an ein Ontologiesystem dar: Möchte man ein Unternehmen unter Verwendung der Design Muster mit den damit einhergehenden Vorteilen modellieren, so muss das Ontologiesystem diese darstellen können.

Wir haben somit eine Sammlung an Design Mustern erarbeitet, die die Ontologiemodellierung vereinfachen. Weiterhin dienen sie als Anforderungen an Ontologiesysteme: Ein Ontologiesystem wird dahingehend überprüft, ob es für die Verwendung im Rahmen des Systems von Freudenreich et al geeignet ist, siehe auch Abbildung 2.

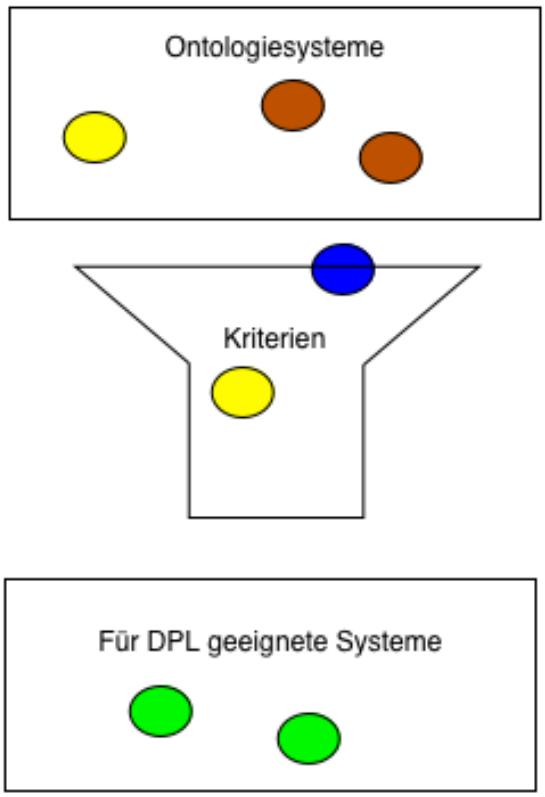

Abbildung 2: Filtern von Ontologiesystemen mit erarbeiteten Kriterien

Unser Vorgehen ist schematisch in Abbildung 3 dargestellt: Wir möchten Design Muster für die Modellierung von Unternehmen erarbeiten. Unser Einstiegspunkt war somit die Betrachtung von Unternehmen im Hinblick auf deren Bestandteile. So konnten wir häufig vorkommende Konstellationen von Unternehmensbestandteilen, also beispielsweise Mitarbeitern, Ressourcen, Produkten und anderen Objekten identifizieren, die wir schließlich zu generelleren Mustern überführt haben. Diese Muster wiederrum die Kriterien für die Anwendbarkeit eines Ontologiesystems, da diese durch das Ontologiesystem abbildbar sein müssen.

Zur Betrachtung von Unternehmen haben wir modellierte Unternehmensprozesse genutzt, die uns mit der Industry Performance Ready Datenbank in großem Umfang zur Verfügung standen [10].

Die erste Unterteilung der Unternehmensprozesse in ihre Bestandteile führten wir unter Verwendung der Ausgangsstruktur, die Freudenreich et al. als Anforderungen an das

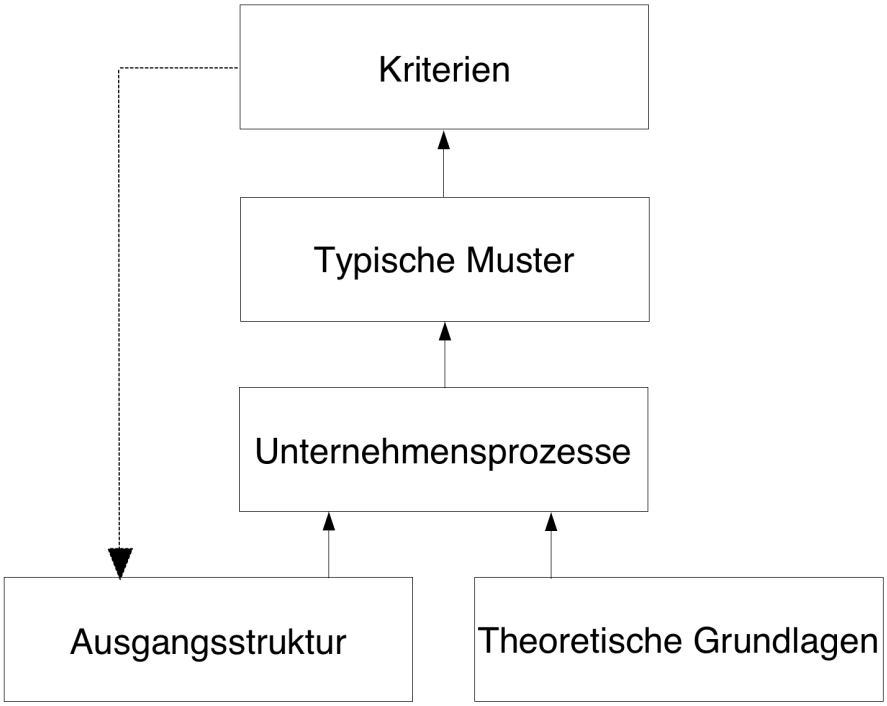

Abbildung 3: Das Vorgehen im Laufe dieser Arbeit - schematische Darstellung

Domain Model vorstellen [2], durch. So erhielten wir eine Sammlung an Prozessbeteiligten, deren Beziehungen untereinander sowie deren Eigenschaften. In dieser Sammlung haben wir Gemeinsamkeiten im Sinne der Modellierung identifiziert, um daraus allgemeineree Modellierungsprobleme zu formulieren. Diese haben wir dann durch die Entwicklung passender Muster gelöst. Dabei verwendeten wir unterstützend theoretische Grundlagen von Software Engineering Design Mustern und Ontologiemustern. Diese Muster bilden schließlich die finalen Kriteren: Ein Ontologiesystem muss sie darstellen können, um einen erfolgreichen Einsatz von DPL und ereignisbasierten Systemen zu unterstützen.

Schließlich haben wir ausgewertet, inwieweit die erarbeiteten Design Muster auf die Ausgangsstruktur zurückzuführen sind. Darauf aufbauend haben wir diskutiert, ob die Überprüfung eines Ontologiesystems nur Anhand der Ausgangsstruktur die gleichen Ergebnisse hervorbringt wie eine Überprüfung auf sämtliche von uns identifizierten Kriterien.

Der Rest dieser Arbeit ist wie folgt aufgebaut: Kapitel 2 beinhaltet den technischen und theoretischen Hintergrund. In Kapitel 3 stellen wir die theoretische Ausgangslage zu Designmustern in Ontologiesystemen dar. In Kapitel 4 beschreiben wir die identifizierten Design Muster, die in Kapitel 5 mit der Ausgangsstruktur verglichen und diskutiert werden. Kapitel 6 enthält ein Fazit und einen Ausblick auf weitere mögliche Arbeitsschritte.

2 Hintergrund

Dieses Kapitel erläutert den technischen Hintergrund und Konzepte der Informatik, die wir im Rahmen dieser Arbeit eingesetzt haben sowie die verwendete Ausgangsstruktur.

2.1 Ereignisgesteuerte Systeme, Policies und die Declarative Policy Language

Ein ergeignisgesteuertes System ermöglicht anonyme und asynchrone Kommunikation zwischen dessen Komponenten in einer verteilten Umgebung [12] S. 227 ff. Ein Ereignis enthält die Information über das Eintreten eines bestimmten Zustandes und wird von einem Senderobjekt (Publisher) generiert. Das Eregnis wird über ein BUS-System an alle interessierten Empfänger (Subscriber) versendet, die eine entsprechende Schnittstelle für den Empfang von Ereignissen zur Verfügung stellen. Dem Absender des Ereignisses ist die Empfängergruppe unbekannt, da eine Middleware die Kommunikation steuert.

Ein Beispiel ist ein System mit Sensoren: Generiert ein Temperatursensor das Ereignis Tempereratur: 60 Grad, wird diese durch das System an alle Systeme weitergeleitet, die sich für Temperaturwerte allgemein, Temperaturwerte über einen Schwellwert oder ähnliches interessieren. Diese Interesse wird der koordinierenden Middleware mitgeteilt, die generierenden Temperatursensoren benötigen diese Information nicht.

Das Eintreten eines Zustands und der Versand eines Ereignisses werden in dieser Arbeit, wenn nicht anders vermerkt, synonym verwendet.

Eine Policy ist eine Regel, die eine Situation und darauf folgende Aktionen beschreibt. Eine Situation kann dabei aus mehreren Teilzuständen zusammengesetzt sein. So soll beispielsweise beim Eintreten eines Mitarbeiters in einen sicherheitskritischen Bereich sichergestellt sein, dass er die Berechtigung hierfür besitzt. Ist dies nicht der Fall, wird ein Alarm ausgelöst. Die Situation ist in diesem Beispiel, dass der Mitarbeiter in einen sicherheitskritischen Bereich eingetreten ist (Teilzustand 1) sowie der Status seiner Sicherheitsberechtigung (Teilzustand 2). Die Aktion ist der Alarm, der bei Eintreten beider Teilzustände ausgelöst wird. Mit Policies lassen sich Regelsysteme für Unternehmen erstellen.

Ein ereignisbasiertes System kann genutzt werden, um zu überprüfen, ob formulierte Situationen eingetreten sind und Folgeaktionen initiieren. Eine Sprache, die eine Formulierung von Policies ermöglicht, ist die Declarative Policy Language (DPL) [1]. Durch ihren deklarativen Ansatz setzt diese Sprache den Schwerpunkt auf die Formulierung der Policies mit ihren Bedingungen und Folgeaktionen. Wie die Umsetzung gelöst wird, also wie und womit beispielsweise das Eintreten von Zuständen überprüft wird, muss bei der Formulierung nicht beantwortet werden. Eine Middleware verarbeitet in der DPL formulierte Policies und befähigt ein ereignisbasiertes System, die notwendigen Überprüfungen durchzuführen. So kann das Regelsystem eines Unternehmens zielorientiert implementiert und verwendet werden.

2.2 Ereignisgesteuerte Prozesseketten

Die Unternehmensprozesse, die wir im Rahmen dieser Arbeit analyisert haben, sind als ereignisgesteuerte Prozessketten (EPK) dargestellt. EPK ist eine Modellierungssprache mit der man Prozesse grafisch Abbilden kann. Dies befähigt Firmen, mögliche Schwachstellen in ihren Abläufen zu identifizieren und häufig wiederkehrende Vorgänge zu formalisieren. EPK wird hierfür in führenden Produkten wie SAP R/3 ERP und ARIS Architect verwendet [3].

EPK besteht aus fünf Komponenten, die zur Modellierung genutzt werden können (sh. Abbildung 4):

1. Funktion
2. Ereignis
3. ODER-Konnektor
4. UND-Konnektor
5. XOR-Konnektor

Abbildung 4: Die fünf Bausteine ereignisgesteuerter Prozessketten

Ein Beispiel für einen Prozessausschnitt findet sich in Abbildung 5. Dort wird dargestellt, dass sowohl das Ereignis Lagerbestand unter Schwellwert als auch das Ereignis Großauftrag zu einem Bestellvorgang führen, um das Lager entsprechend aufzufüllen.

2.3 Aris Architect und Industry Performance Ready

Die Prozesse, die im Zuge dieser Arbeit analysiert wurden, befinden sich in der Industry Performance Ready (IPR) Datenbank. Das Tool ARIS Architect liest sie von dort aus und stellt sie als EPKs dar. Beide Produkte entstammen der Software AG [4].

Mit ARIS Architect modelliert und analysiert man Geschäftsprozesse. Modelliert wird mit einer grafischen Oberfläche mit verbreiteten Prozessmodellierungssprachen wie EPK und Business Process Model and Notation (BPMN). Auf letztere gehen wir nicht weiter ein, da in dieser Arbeit ausschließlich EPK-Prozesse relevant sind. Man kann mit ARIS Architect die modellierten Prozesse um Zusammenhänge zu notwendigen Ressourcen und Umgebungseigenschaften ergänzen. Schließlich bietet ARIS Architect die Möglichkeit,

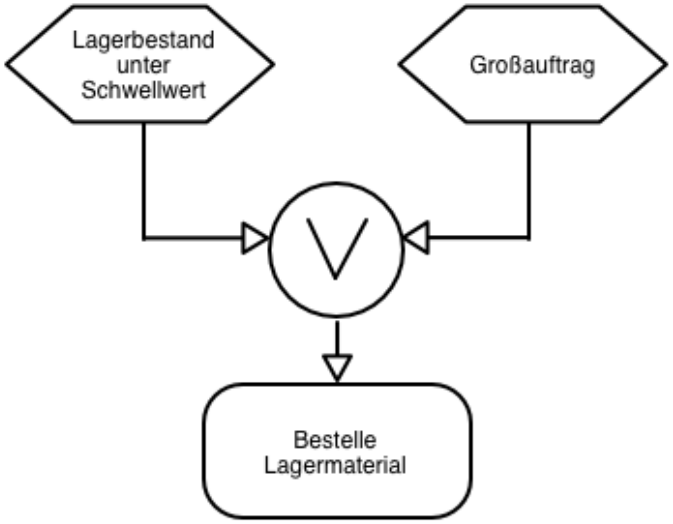

Abbildung 5: Beispielhafter Prozessausschnitt modelliert mit EPK

Prozesse unter Einsatz von Analysefunktionen auf Abhängigkeiten und Beziehungen hin zu überprüfen.

Neben der Möglichkeit, eigene Geschäftsprozesse zu modellieren, stellt ARIS Architect eine umfangreiche Anzahl an bereits modellierten Geschäftsprozessen dar, die in der IPR Datenbank hinterlegt sind. Diese sind nach Geschäftsarten unterteilt. So gibt es z.B. Prozesse in den Bereichen Verteidigung, Gesundheitswesen, Logistik und Finanzdienstleistungen. Wir haben eine Auswahl dieser mitgelieferten Prozesse für die Analyse genutzt und die Untergliederung nach Geschäftsarten für eine erste Strukturierung verwendet.

2.4 Ontologien

Der Begriff der Ontologie entstammt dem Griechischen und setzt sich aus den Wörtern on - seiend und logos - Wort zusammen. Allgemein ist eine Ontologie einer Domäne eine Beschreibung ihrer Terminologie, der westenlichen Konzepte der Domäne, deren Klassifizierung, hierarchische Anordnung und Beziehungen zueinander [17] S. 46.

Das Ziel einer Ontologie ist es, Wissen über eine Domäne zu teilen und nutzbar zu machen [17] S. 53. Ontologien in der Informationstechnologie beschreiben formal und explizit eine Konzeptualisierung [14] S. 4. Eine Konzeptualisierung bedeutet dabei eine abstrakte, vereinfachte Sicht auf die Welt [17] S. 46. Sie enthält die in der Welt beteiligten Elemente und die Zusammenhänge zwischen ihnen. Die beschriebene Welt kann real oder fiktiv sein [11].

Die Entwicklung einer Ontologie besteht aus drei Schritten [13].

- Spezifizierung von Terminologie, Ziel, Granularitätslevel und Umfang

- Dem Erstellen eines groben Konzepts
- Der Umsetzung der Ontologie unter Verwendung eines Ontologiesystems

Wir konzentrieren uns in dieser Arbeit auf den letzten Schritt, der Umsetzung der Ontologie mittels eines Ontologiesystems.

Ein Ontologiesystem umfasst sowohl die verwendete Ontologiesprache als auch die zur Modellierung genutzte Arbeitsumgebung. Eine weit verbreitete Ontologiesprache ist OWL [9], die besonders im Schwerpunktbereich von Ontologien, dem Semantic Web, verwendet wird.

In dieser Arbeit wird der Begriff Ontologie wenn nicht anders vermerkt synonym für Ontologieinstanzen, also konkreten Modellierungen einer Umgebung, genutzt.

2.5 Ausgangsstruktur

Als Basis für unsere Arbeit diente die von Freudenreich et al. entwickelte Struktur [1]. Sie stellt eine minimale Anforderungen an die sprachliche Mächtigkeit durch Ontologiesystem oder ein Domänenmodell dar, damit dieses sich für den Einsatz der definierten DPL und verwendeten Middleware zur Erzeugung ereignisbasierter Systeme eignet. In unserer Arbeit nutzen wir diese Struktur daher als Ausgangspunkt. Sie bietet den Vorteil einer stabilen Basis, die die weitere Analyse untersützt und sicherstellt, dass sämtliche Designmuster auf ihr aufbauen.

Die Struktur beinhaltet definierte Bausteine, die zur Modellierung der Welt in einer Ontologie eingesetzt werden. Diese sind

- Konzepte
- Attribute
- Beziehungen

Als Beispielszenario nehmen wir an, dass Peter M, ein Manager, im Konferenzraum Nummer 20 ist (sh. Abbildung 6). Dieses Szenario beinhaltet die Objekte Peter M. sowie den Konferenzraum Nummer 20. Diese sind Instanzen der abstrakten Objekte `Person` bzw. `Raum`. Sowohl `Person` als auch `Raum` wiederrum sind Instanzen des Bausteins `Konzept`. Weiterhin soll Peter M. den Status Manager erhalten, der Raum die Nummer 20. Beides sind Eigenschaften, die Instanzen der abstrakten Eigenschaften `Status` bzw. `Nummer` sind und den Konzeptinstanzen zugeordnet werden. `Status` und `Nummer` wiederum sind Instanzen des Bausteins Attribute. Schließlich besteht eine Beziehung zwischen Peter M. und dem Konferenzraum: Peter M. ist in dem Raum. Diese Beziehung ist ebenfalls Instanz einer abstrakten `ist in` - Beziehung, die wiederum eine Instanz des Bausteins Beziehung ist.
Kann man alle Elemente einer Ontologie auf diese drei Bausteine zurückführen, so erfüllt diese die minimalen Anforderungen für die Verwendung mit DPL. Der interessierte Leser sei für eine ausführlichere Erläuterung auf [1] verwiesen.

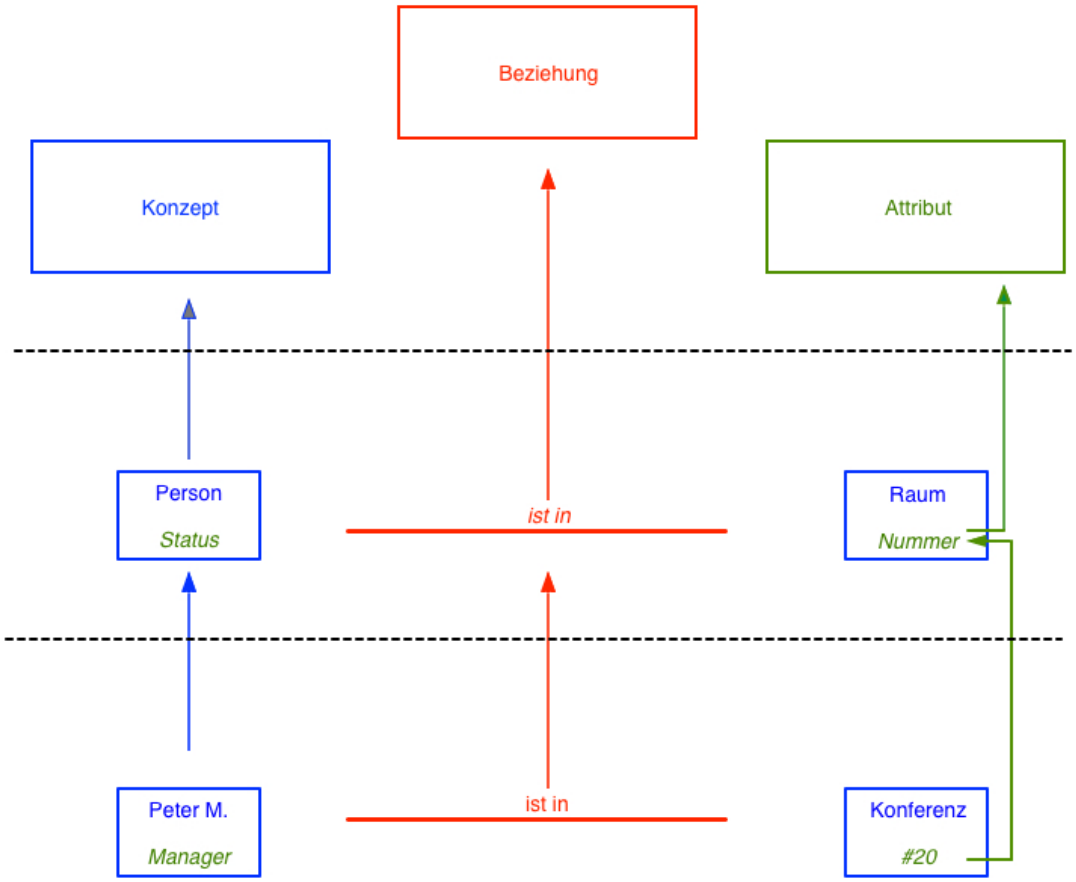

Abbildung 6: Beziehung zwischen dem Ausgangskonzept und der Modellierung eines Szenarios. Zur besseren Lesbarkeit werden nicht alle Instanziierungspfeile dargstellt.

3 Ausgangslage

In diesem Kapitel beschreiben wir weitere Ansätze zur Modellierung von Weltausschnitten, die in der Wissenschaftsliteratur aufgeführt sind. Zunächst gehen wir auf Domänenmodels ein. Weiterhin betrachten Ontologiemuster, die vorgefertigte Ontologiebausteine bilden. Anschließend stellen wir ausgewählte Designmuster des Software Engineering Modellierung vor und wie wir diese für unserer Design Muster nutzen konnten.

Nach gründlicher Recherche hat sich herausgestellt, dass weitere Quellen, die Unternehmensprozesse darstellen, in keiner Weise so für die vorgenommenen Analysen geeignet sind wie die im ARIS Architect modellierten Prozesse. Daher haben wir von weiteren Recherchen im Bereich der Unternehmensprozesse Abstand genommen.

3.1 Domänenmodelle

Domänenmodelle, auch Domain Models, sind formale Konzeptualisierungen einer zuvor informell beschriebene Domäne und werden häufig im Software Engineering eingesetzt [17] S. 167, sie können aber auch, wie bei Freudenreich et al., als generelle Wissensrepräsentation über eine Domäne dienen, ähnlich einer Ontologie. Domänenmodelle ermöglichen es, eine zu entwickelnde Software auf die Anwendungsdomäne auszurichten und bestehen dabei aus verschiedenen Facetten. [16] Der Fokus liegt auf der Darstellung der Domäne, noch nicht auf der technischen Realisierung. So sind die Elemente eines Domänenmodells im Unternehmensbereich beispielsweise Mitarbeiter, Kunden und Produkte sowie bestehende Beziehungen zwischen diesen, noch nicht aber die weiteren Implementierungsdetails wie Klassen, Methoden und Felder.

Kuzniarz und Staron stellen eine automatisierte Domänenmodellerstellung anhand gegebener Ontologien vor [18]. Hierbei sollen bereits bestehende, modellierte Ontologien genutzt werden um daraus Domänenmodelle zur Softwareentwicklung zu generieren. Die Ontologie ist dabei die Representation des gesamten Domänenwissens, wohingegen das Domänenmodell die abstrakte Beschreibung eines konkreten Real-World-Phänomens ist.

Eine häufige Darstellungsform sind UML-Diagramme [5].

3.2 Ontologiemuster

Bestehende Ontologiemuster, die sich in der Wissenschaftsliteratur finden lassen, sind vorgefertigte Ontologiebausteine, die für die Modellierung einer vollständingen Ontologie zusammengefügt werden. A. Gangemi und V. Presutti präsentieren verschiedene Typen von Ontologiemustern sowie Schritte zur Herleitung eines solchen [11]. Ihr Ziel ist es, mittels der Verbindung kleiner, vorgefertigter Bausteine komplexe Ontologien einfach und erweiterbar zu gestalten. Als Ontologiesprache fokussieren sie sich auf OWL und nutzen ausschließlich die dadurch verfügbaren Sprachelemente.

Eine besondere Gruppe von Ontologiemustern sind die Content Ontology Design Patterns (CPs), die sich nicht mehr auf die Logik der Ontologie (wie n-äre Relationen) konzentrieren, sondern inhaltliche Probleme abbilden [8]. Beispielsweise beschreiben die

Autoren das Participation-Pattern: Es modelliert, dass ein Objekt an einem Ereignis teilgenommen hat.

Die Herleitung eines CPs kann auf vier Wegen geschehen:

1. Ein anderes Modell oder Muster wird umgestellt und den Bedürfnissen angepasst.
2. Ein bestehendes Muster wird für den gewünschten Anwendungsbereich spezialisiert.
3. Mehrere bestehende Muster werden aneinandergefügt.
4. Muster werden aus einer bereits bestehenden Ontologie extrahiert.

Die Ergebnisse ihrer Forschung stellen die Autoren auf einer Internetplattform zur Verfügung [7].

Neben der Arbeit von Gangemi und Presutti sind nach unserer Kenntnis keine ausgeprägten Ansätze für die die Unterstützung von Ontologiemodellierung durch Ontologiemuster verbreitet.

3.3 Andere wiederverwendbare Strukturen

Da die Forschung im Bereich der Ontologiemuster noch nicht weit fortgeschritten ist, haben wir uns auch mit anderen in der Informationstechnologie gängigen Mustern beschäftigt. Die wohl am bekanntesten sind die Designmuster im Software Engineering, die die Zusammenhänge von Klassen, deren Methoden und Attribute in einer Software als Betrachtungsgegenstand haben. Sie bilden formalisierte Lösungen für gängige Probleme in der Software Entwicklung [15].

Zwei Muster, die wir im Verlauf dieser Arbeit berücksichtigt haben, sind das Stragety- und das Composite-Pattern. Wir haben jeweils einige Grundgedanken dieser Modellierungsarten extrahiert und für die Entwicklung unserer Designmuster verwendet.

Das Strategy-Pattern wird verwendet, wenn Klassen im Grundgerüst gleich sind, sich aber in einigen Verhaltensweisen unterscheiden. Diese werden dann als Strategie ausgelagert. Ein Beispiel ist in Abbilung 7 dargestellt: Je nach Rolle des Mitarbeiters unterscheidet sich die Verhaltensweise der Methode doJob():void, die durch die Implementierungen des Interfaces *IAktivitäten* bestimmt wird. Jeder Mitarbeiter wird mit einer implementierten Aktivität instanziiert und erhält somit sein spezielles Verhalten der Methode doJob():void. Gleichzeitig ist sichergestellt, dass die Methode für alle Mitarbeiterinstanzen zur Verfügung steht und so ohne zusätzliche Information über ihre Implementierung aufgerufen werden kann. Schließlich lässt sich das Verhalten der Methode performant anpassen: Ändert man die Implementierung einer Aktivität, beispielsweise weil ein Einkäufer zusätzliche Tätigkeiten ausführen soll, so ändert sich das Verhalten aller Mitarbeiterinstanzen, die diese Einkäuferaktivität als Strategie verwenden. Andere Mitarbeiterinstanzen sind nicht betroffen.

Aus dem Strategie-Pattern haben wir folgende Gedanken genutzt, um unsere Designmuster zu entwerfen:

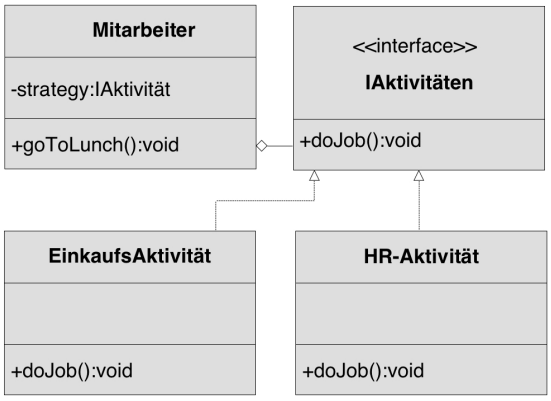

Abbildung 7: Beispiel Strategy-Pattern

- Abstrakte Verwendung von Komponenten: Die Methoden des Interfaces können unabhängig von ihrer konkreten Implementierung in anderen Methoden verwendet werden. So ist der Aufruf EINMITARBEITER.STRATEGY.DOJOB() immer gültig und die aufrufende Methode muss keine zusätzlichen Unterscheidungen nach Mitarbeiterrolle durchführen. Es wird sichergestellt, dass eine Implementierung der Methode exisitiert.

- Bündelung zusammengehöriger Verhaltensweisen: Die Implementierung einer Strategie enthält alle dazu gehörigen Funktionalitäten. So werden beispielsweise sämtliche Einkaufsaktivitäten an einer Stelle gebündelt. Instanziiert man einen Mitarbeiter mit EINKAUFSAKTIVITÄT, so ist sichergestellt, das alle notwendigen Funktionalitäten für einen Einkäufer vorhanden sind.

- Austauschbarkeit von Komponenten: Die Strategie einer Instanz kann ausgetauscht werden. Durch die Bündelungseigenschaft wird sichergestellt, dass alles Notwendige beim Austausch übernommen wird. Die sonstige Implementierung oder Verhaltensweise der Klasse wird dadurch nicht beeinflusst.

- Änderbarkeit an einer Stelle: Eine Strategie kann an einer Stelle, der Implementierung des Interfaces, geändert werden. Die Änderungen betreffen ausschließlich und vollständig die Instanzen, die diese Strategie verwenden. Dies ermöglicht einfache Erweiterbarkeit mit geringem Fehlerrisiko.

Das Composite-Pattern ermöglicht es, darüber zu abstrahieren, ob es sich bei einem Objekt um ein atomares Objekt oder einen Container, der mehrere Objekte enthält, handelt. Es ermöglicht die Modellierung sogenannter *Part-Whole-Hierarchies*. Dies sind Hierarchien, die sowohl aus Objekten als auch aus Containern, die diese Objekte oder weitere Container mit Objekten enthalten, gebildet werden. Inhalte eines Containers werden als dessen Kinder bezeichnet.

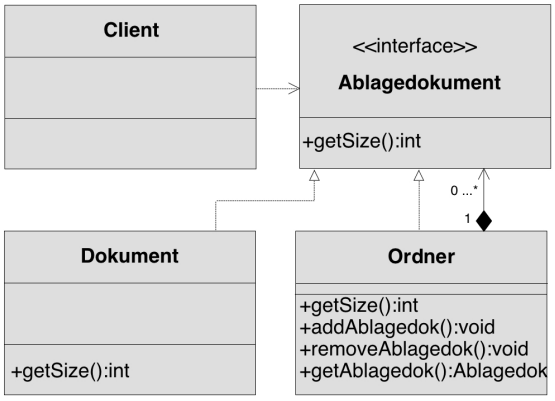

Abbildung 8: Beispiel Composite-Pattern

Ein Beispiel, nach [6], stellen wir in Abbildung 3.3 vor: Ein atomares Objekt ist dabei ein ein Objekt der Klasse Dokument, ein Container ein Objekt der Klasse Ordner, der wiederrum mehrere Instanzen des Typs Ablagedokumente, das abstrakte Composite-Objekt, enthalten kann. Beide implementieren die Methode getSize(), wobei in der Klasse Dokument die tatsächliche Dokumentengröße zurückgegeben wird, in der Container-Klasse hingegen auf die getSize() Methode der enthaltenen Ablagedokumente verwiesen wird und deren Ergebnisse beispielsweise addiert werden. Weiterhin verfügt die Klasse Ordner über Methoden zur Verwaltung ihrer Kinder: diese können hinzugefügt, entfernt und zurückgegeben werden. Handelt es sich bei einem entfernen oder zurückgeben-Aufruf auf ein Kind um ein Dokument, das im aktuellen Ordner-Objekt enthalten ist, so wird dieses zurückgegeben oder entfernt. Alternativ wird die Anfrage an alle enthaltenen Ordner-Objekte weitergegeben, die ebenso verfahren.

Eine Client-Methode ruft die Methode Ablagedokument.getSize() auf einer Instanz von Ablagedokument auf und muss nicht berücksichtigen, ob es sich um eine Instanz von Dokument oder von Ordner handelt. Nach außen hin wird also eine einheitliche Perspektive gewahrt und über die konkrete Implementierung des Ablagedokuments abstrahiert. Jedes Composite-Objekt dient als Einstieg in die Hierarchie, die es mit seinen Kindern bildet. Diese wird von Clients als Einheit betrachtet.

Aus dem Composite-Pattern haben wir folgende Gedanken für unsere Designmuster verwendet:

- Container-Konzept: Elemente können in Containern "geschachtelt" werden und so klare Hierarchien bilden. Zusammengehörende Elemente werden einzeln modelliert und in die Hierarchie eingeordnet, sind nach außen hin aber als Einheit mit einem festen Einstiegspunkt sichtbar. Für die Verwendung dieser Einheit wird lediglich eine Beziehung zum Einstiegspunkt, zB einem Ordner-Objekt, benötigt. Alle darin enthaltenen Elemente werden mit erfasst und können (im Beispiel über getAblagedok() angesprochen werden. Es ist unerheblich, wie tief in der Hierarchie das

gewünschte Ablagedokument liegt, für den Nutzer ist nur der Zusammenhang zwischen dem Einstiegspunkt und Zieldokument relevant.

- Abstraktion über konkrete Implementierung: Es ist für den Aufrufer unerheblich, ob es sich bei einem Ablagedokument-Objekt um eine Instanz von DOKUMENT oder von ORDNER handelt.

Im folgenden Kapitel erläutern wir, welche eigenen Designmuster wir aus den Inhalten dieses Kapitels und der Analyse der Unternehmensprozesse abgeleitet haben.

4 Ermitteln der Designmuster

Dieses Kapitel beschreibt, wie wir Designmuster herausgearbeitet haben, die für die Ontologieerstellung unterstützend sind. Weiterhin erläutert es diese und bespricht die Vorteile, die sie für die Ontologiemodellierung bringen.

4.1 Vorgehen bei der Musterentwicklung

Ein Designmuster ist eine Lösung auf immer wiederkehrende Modellierungsherausforderungen. (sh. Abbildung 3). Modellierungsherausforderungen sind dabei Konstellationen von Domänenelementen, die in der Domänenmodellierung berücksichtigt werden sollen. Beispielsweise können dies Personen sein, die miteinander in Beziehung stehen, oder Personen und Objekte, die sich gegenseitig beeinflussen. Diese Konstelltionen müssen bestimmte Eigenschaften aufweisen, damit sie relevant genug sind, in Designmuster überführt zu werden, statt sie beispielsweise durch Vereinfachungen nur näherungsweise zu lösen. Wir bezeichnen diese Eigenschaften als Anforderungen an die Konstellationen. Diese Anforderungen trennen relevante Konstellationen von solchen, die zu speziell, einmalig oder unausgeprägt sind.

Anforderungen an Konstellationen sind:

- Übertragbarkeit
- Verstärkung der Aussagekraft des Modells
- Vereinfachung der Modellierung und Fehlervermeidung

Übertragbarkeit bedeutet, dass sich eine solche Konstellation nicht nur in einer, sondern in verschiedenen Domänen wiederfinden lässt. Dies ist die Bedingung für die Wiederkehrung der Herausforderung und somit die Begründung für eine generelle Designlösung: Nur Probleme, die häufig und in verschiedenen Bereichen wiederkehren, bedürfen eines allgemeinen Designmusters.

Die Aussagekraft des Modells ist stärker, wenn ich durch eine genaue Abbildung der Konstellation Zusammenhänge klar, Abhängigkeiten eindeutig und scharfe Abgrenzungen zwischen semantischen Einheiten möglich sind und diese Eigenschaften ohne klare Abbildung weniger ausgeprägt wären. Dies schließt es aus, komplexere Konstellationen zu vereinfachen und mit herkömmlichen Methoden zu lösen, da es gerade durch die Komplexität zur erhöhten Aussagekraft führt.

Schließlich sollte die Modellierung durch die genaue Abbildung der Konstellation vereinfacht werden, beispielsweise durch Vermeidung von Redundanzen.

Erfüllt eine Konstellation diese Eigenschaften, wird sie als relevant genug betrachtet, um als Grundlage für ein Desingmuster zu dienen. Das Designmuster ist dabei die Modellierungslösung für die gegebene Konstellation.

Zum Ermitteln der möglichen Designmuster haben wir eine Auswahl an Unternehmensprozessketten betrachtet, die sich in der Industry Performance Ready Datenbank des ARIS Tools befinden. Diese nutzt die Software AG zur Unternehmensberatung [10] Die Auswahl der analysierten Prozesse basierte auf folgenden Kriterien:

- Der Prozess ist verständlich
- Der Prozess ist allgemeingültig oder charakterisierend für eine Branche
- Der Prozess ist aussagekräftig und vollständig modelliert

Verständlichkeit bedeutet, dass der Zweck des Prozesses für uns erkennbar war und die verwendeten Elemente im Gesamtzusammenhang verstanden wurden. Nur so konnte eine sinnvolle Analyse des Prozesses garaniert werden. Ein sehr allgemeingültiger Prozess ist beispielsweise ein Einkaufsprozess. Ein sehr branchentypischer beispielsweise die Planung von Routen und Fahrzeugen in einem Logistikunternehmen. Dieses Kriterium ermöglicht eine Fokussierung auf tatsächlich relevante Prozesse und somit weiterführend relevanten Problemen. Die Anforderung nach Aussagekraft und Vollständigkeit entstammt der Tatsache, dass einige der Prozesse in der Industry Performance Ready Datenbank Blankofelder enthalten oder nicht im Detail dargestellt sind.

Zusammenfassend gilt, dass ein Prozess repräsentativ und korrekt sein und in angemessenen Granularität vorliegen muss, um relevante Konstellationen ermitteln zu können.

Die Industry Performance Ready Datenbank wird in 13 Geschäftsfelder unterteilt, die jeweils aus Management-, Kern- und unterstützenden Prozessen bestehen. Von diesen 13 Geschäftsfeldern haben wir sechs näher betrachtet und bei diesen jeweils drei Prozesse oder Prozesseinheiten aus dem Management-, Kern- und unterstützenden Bereich angesehen.

Die ausgewählten Unternehmensprozessketten (UPKs) wurden in semantischen Einheiten betrachtet. Hat ein Prozess mehrere Teilprozesse, so wurden diese gemeinsam als Prozesseinheit analysiert, auch wenn sie aus mehreren EPK-Diagrammen bestanden, auch wenn diese jeweils einzeln als vollständige Prozesse in der Datenbank zu finden waren. Beispielsweise besteht ein Einkaufsprozess aus der Bedarfsermittlung, dem Bestellvorgang und der Qualitätsprüfung nach Eingang der Ware, was drei EPKs sind, aber einen semantischen Block bildet. Diese Betrachtungsweise erhöht das Verständnis, verhindert, dass Prozesse nach Erfüllung der oben genannten Kriterien wahllos und bruchstückhaft betrachtet werden und führt zu einer vollständigeren Analyse.

Die Analyse der Prozesse erfolgte nun in mehreren Schritten: Zunächst erfolge unter Verwendung des Ausgangskonzepts die Untergliederung der Prozesse in Konzepte, deren Attribute und Beziehungen. Im nächsten Schritt haben wir das Basisgerüst eines jeden betrachteten Prozesses hervorgehoben. Dies waren beispielsweise Konzepte, deren Attribute durch die Durchführung des Prozesses geändert werden. Wir wählten die Elemente, also Konzepte, Relationen und Attribute, die die Antwort auf die Frage „Was geschieht in diesem Prozess und wie?“bildeten. Konzepte, die Informationstechnologie repräsentieren, wurden von der weiteren Betrachtung ausgenommen, da sie nur unterstützende Funktionen erfüllen.

Schließlich haben wir die identifizierten Elemente auf eine höhere Abstraktionsstufe gehoben, um allgemeine Modellierungsprobleme zu erhalten. Hierfür haben wir sie zunächst auf häufige gemeinsame Konstellationen hin überprüft und so allgemeinere Muster gebildet. Beispiele sind Einkäufer und Bestellung, die als zwei Konzepte oder

auch Fahrer und aktualisiert, die als Konzept und Beziehung häufig gemeinsam auftreten. Wo sind Gemeinsamkeiten, wo Unterschiede? Mit diesen Fragen extrahierten wir allgemeinere Muster.

Als zweite Methode zur Erhöhung der Abstraktionsstufe haben mögliche Verallgemeinerungen herausgearbeitet. So können beispielsweise Einkäufer und Krankenschwester zur semantischen Obergruppe Mitarbeiter zusammengeführt und Beziehungen in schaffende, überprüfende und modizifizerende Aktionen eingeteilt werden.

Dieser Abstraktionsprozess erfolgte unter Berücksichtigung der in Kapitel 3 genannten Strukturen und Muster. Diese wurden aber lediglich als Leitfaden genutzt, um die Entwicklung neuer Designmuster nicht künstlich einzuschränken.

Die so ermittelten allgemeineren Modellierungsprobleme haben wir anschließend durch Designmuster gelöst. Diese erläutern wir im kommenden Abschnitt genauer.

4.2 Erläuterung und Diskussion von Ontologiedesignmustern

In diesem Abschnitt stellen wir die Designmsuter im Detail vor, die wir als Lösungen für die identifizierten Modellierungsprobleme entwickelt haben und gehen darauf ein, wie diese zu einer aussagekräftigen und fehlerfreihen Ontologie beitragen. Im Kontext des von Freudenreich et al. vorgestellten Systems werden diese Muster von Domänenexperten bei der Erstellung der Ontologie genutzt. Dies kann, muss aber nicht die gleiche Person sein, die die Policies in DPL formuliert.

4.2.1 Verallgemeinerung und Spezialisierung

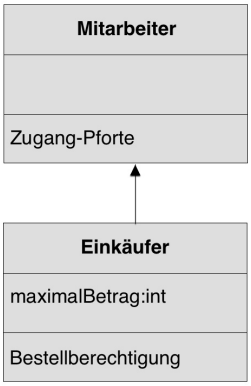

Abbildung 9: Beispiel Verallgemeinerung

Dieses Muster übernimmt das in der Softwareentwicklung häufig eingesetzte Konstrukt der Vererbung. Die Vererbung wird auf die Konzepte der Ontologie übertragen. So entstehen Konzepthierarchien, bei denen die Konzepte immer weiter spezialisiert sind. Ein Konzept erbt von einem anderen, wenn es dessen Eigenschaften übernimmt und um

spezialisiertere ergänzt. Eigenschaften können Attribute oder Beziehungen sein, die es eingehen kann. Ein Beispiel ist das Konzept Einkäufer, das vom Konzept Mitarbeiter erbt (Abbildung 9). So kann beispielsweise jeder Mitarbeit morgens an der Pforte seine Zutrittskarte scannen lassen, aber nur der Einkäufer kann zusätzlich Bestellungen bis zu einem festgesetzten Wert durchführen. Die Beziehung Mitarbeiter scannt Karte gilt für alle Instanzen des Konzepts Mitarbeiter, ebenso wie alle diese Instanzen ein Attribut Mitarbeiter-ID besitzen. Erbt das Konzept Einkäufer in der Ontologie vom Konzept Mitarbeiter, so kann er seine für ihn spezifischen Beziehungen eingehen (Einkäufer führt Bestellung durch), die durch das für den Einkäufer modellierte Attribut Maximale Bestellsumme beeinflusst wird. Gleichzeitig kann auch er seine Karte scannen und sich über seiner Mitarbeiter-ID identifizieren lassen. Ein Konzept Marketingmanager, das ebenfalls vom Konzept Mitarbeiter erbt, darf nun keine Bestellungen durchführen, kann aber dafür beispielsweise über ein Attribut Region verfügen, dass die geografischen Bereiche, für die er zuständig ist, repräsentiert.

Wird die Modellierung von Vererbung in einem Ontologiesystem ermöglicht, so entstehen mehrere Vorteile: Zum einen können Beziehungen und Attribute, die für mehrere Konzepte gelten, an einer Stelle modelliert werden. So werden redundante Modellierungen, die zu Fehlern führen können sowie unnötige Mehrarbeit reduziert. Gleichzeitig wird durch die Spezialisierung sichergestellt, dass Beziehungen und Attribute nur für solche Konzepte modelliert werden, die diese tatsächlich auch haben dürfen. Hier wird unter anderem die Anforderung an die scharfen Grenzen zwischen semantischen Einheiten erfüllt. Das Muster "Verallgemeinerung und Spezialisierung" wurde in unseren Analysen in jedem der betrachteten Prozesse erkannt und ist weiterhin eine Grundlage für die folgenden Muster.

4.2.2 Gruppierungen

Mit dem Muster Gruppierungen werden Konstellationen von Konzepten zu einer nach außen hin geschlossenen Einheit zusammengeführt. Diese Einheit, die wir als Gruppierung bezeichnen, wird dann für die Modellierung weiterer Beziehungen genutzt. Ein Beispiel ist die Verbindung von Einkäufer und Bestellung 10. Man modelliert mit dieser Gruppierung sämtliche Aktivitäten, die sich auf eine Bestellung beziehen und von einem Einkäufer durchgeführt werden, zum Beispiel eine Qualitätskontrolle der zu der Bestellung gehörigen Ware durch den Einkäufer.

Das Msuter Gruppierung ermöglicht es, Konzepte, die semantisch zusammengehören oder in vielen Aktionen gemeinsam auftreten, zusammenzufügen. Häufig auftretende Konstellationen müssen nur einmal modelliert werden. So wird das Fehlerpotenzial und Unklarheiten bei der Modellierung reduziert. Gleichzeitig können die Konzepte nach wie vor als einzelne Ontologiebausteine verwendet werden. Ein Einkäufer kann, muss aber nicht immer gemeinsam mit einer Bestellung modelliert werden. Die Flexibilität bleibt damit erhalten. Weiterhin ist es eindeutig, welche Konzepte zu einer Gruppierung gehören, da diese, einmal gebildet, konstant bleibt. Dies sorgt für Klarheit der Ontologie. Schließlich ist das Modellieren einfacher: Gruppierungen, die mit anderen Konzepten in Beziehung treten, werden nicht als n-äre Beziehungen mehrerer Konzepte modelliert,

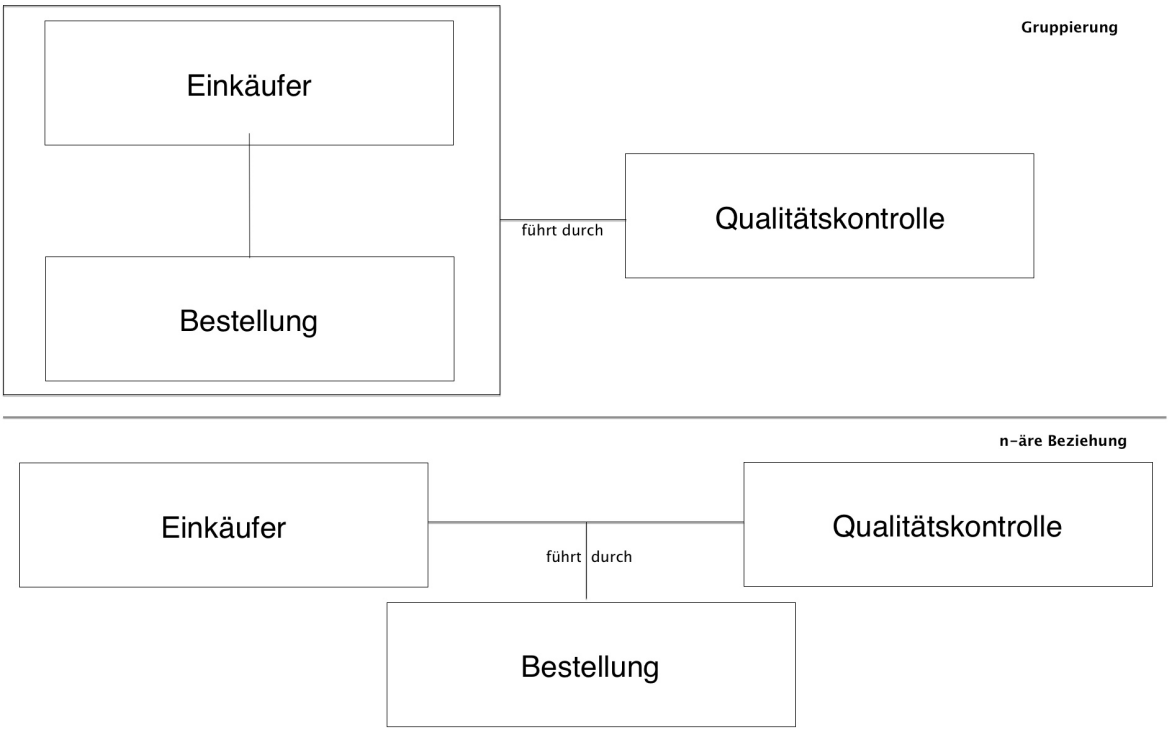

Abbildung 10: Beispiel Gruppierung

sondern auf binäre Beziehungen zurückgeführt.
Attribute der Konzepte einer Gruppierung werden zu einer Menge zusammengeführt, die die Attributmenge der Gruppierung bildet. Sämtliche Attribute der Gruppierungselemente werden dabei in die gemeinsame Menge übernommen. Ebenso können zusätzliche Attribute, die für die Konzepte in dieser Konstellation relevant sind, hinzugefügt werden. Ein Beispiel ist das Attribut Frequenz, das sich auf eine Konstellation aus Bestellung und Einkäufer bezieht und den Turnus darstellt, mit dem ein Einkäufer eine bestimmte, regelmäßig auftretende Bestellung durchführt.

Konzepte, die zu einer Gruppierung zusammengeführt werden, können, müssen aber nicht vom gleichen semantischen Typ sein. So kann es beispielsweie eine Gruppierung Liefernachrichten geben, die die Konzepte Lieferanweisung und Zustellbestätigung enthält. In diesem Fall handelt es sich bei beiden Konzepten um Nachrichentypen, die z.B. gemeinsam bearbeitet werden. Eine ebenfalls denkbare Gruppierung aus unterschiedlichen semantischen Typen ist die oben bereits aufgeführte Konstellation von Einkäufer und Bestellung.

Der Vorteil der Gruppierung gegenüber der Anwendung Vererbung liegt darin, dass für spezielle Zusammenstellung keine zusätzliche Konzeptebene eingeführt werden muss. Dies ist vor allem bei Gruppierungen von Konzepten stark unterschiedlichen Typs wie Einkäufer und Bestellung relevant: Es besteht durch die Gruppierung nicht die Gefahr, ein künstliches Hilfskonzept mit den gemeinsamen Attributen einzuführen, von dem beide erben können, das aber keine eigene semantische Bedeutung hat. Schließlich ist

es möglich, sicherzustellen, dass nur zuvor festgelegte Konzepte zu einer Gruppierung zusammengefügt werden und diese nicht unbeschränkt erweitert wird, wie es bei der Vererbung möglich ist.

4.2.3 Beziehungen mit variablen Komponenten

Um das Muster Beziehungen mit variablen Komponenten zu realisieren, benötigen wir eines Hilfskonstrukts:

Mengen sind die Basis für die im folgenden erläuterten Beziehungen mit variablen Komponenten. Eine Menge fügt ebenso wie Gruppierungen zusammengehörige Elemente aneinander. In der Modellierung dient eine Menge als Platzhalter für *ein* in ihr enthaltenes Element, ähnlich der Strategie im Strategy-Pattern: Dort dient die Referenz auf das Strategy-Interface als Platzhalter für eine konkrete Implementierung. Beispielsweise kann eine Menge aus den Konzepten Einkäufer, Marketingmanager und Logistikleiter unter dem Titel Leitende Angestellte oder aus den Beziehungsarten bestellt, bezahlt, überprüft unter dem Titel Einkaufsaktivitäten bestehen. So lässt sich modellieren, dass ein Mitarbeiter Zugang zu einem Gebäude hat, statt das dies für die Konzepte Einkäufer, Marketingmanager und Logistikleiter im Einzelnen dargestellt werden muss. Elemente einer Menge gehören semantisch einem gemeinsamen Typ an, benötigen sonst aber keine Gemeinsamkeiten wie beispielsweise ein Konzept, von dem alle erben und damit die Attribute dessen übernehmen. Sie werden gesondert zusammengestellt. So ist eine stärkere Differenzierung möglich als bei der Vererbung, bei der beispielsweise alle Konzepte, die von einem Konzept Leitende Angestellte erben, Zutritt zu einem Raum erhalten.

Mengen dienen dem Zusammenfügen einer größeren Anzahl Elemente und verfolgen das Ziel, dynamisch anpassbar zu sein, worin sie sich von Gruppierungen unterscheiden.

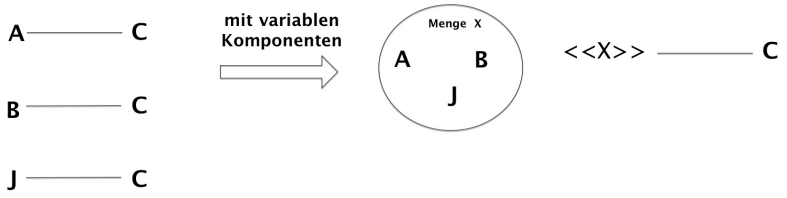

Abbildung 11: Beispiel Beziehung mit variablen Komponenten

Betrachtet man das Ausgangskonzept, so wird eine reguläre Beziehung zwischen zwei Konzepten in der Form Konzept - Beziehungsart - Konzept gebildet. Möchte man nun modellieren, dass sowohl Konzept A als auch Konzept B eine gleichwertige Beziehung mit Konzept C eingehen können, müssten zwei neue Beziehungen modelliert werden: A zu C und B zu C. (Abbildung11) Dies vereinfacht das Designmuster Beziehungen mit variablen Komponenten: A und B werden zu einer gemeinsamen Menge zusammengefügt zusammengefügt und die abstraktere Beziehung Menge AB - Beziehungsart - Konzept C modelliert.

Als Beispielfall sei angenommen, dass nicht nur der Einkäufer, sondern auch der Marketingmanager und der Logistikleiter Bestellungen erstellen dürfen. Statt drei Beziehungen zu modellieren, fügt man die drei Konzepte zu der Menge "Leitende Angestellte" zusammen (sh. oben), die jeweils alle über die Attribute verfügen müssen, mit denen sie die Beziehung eingehen. Aschließend modelliert man wie in der Abbildung 11 schematisch dargestellt. Ebenso verfährt man, wenn man in der Ontologie festhält, dass beispielsweise ein Einkäufer eine Bestellung bestellt, bezahlt und überprüft. Vereinfachend modelliert man die Beziehung: Einkäufer - "Einkaufsaktivitäten" - Bestellung. Legal sind dann alle Beziehungen, die *ein* Element aus der Menge "Einkaufsaktivitäten" als Beziehungsart zwischen Einkäufer und Bestellung nutzen.

Statt selbstdefinierter Mengen kann auch das Vererbungsmuster verwendet werden, die automatisch eine Menge aus allen Konzepten definiert, die von einem gemeinsamen Konzept erben.

Der Vorteil dieses Musters liegt zunächst darin, dass man Modellierungsarbeit reduziert und somit Fehler vermeiden kann, da mehrere Beziehungen in einer abstrakten Aussage dargestellt werden. Die in der Aussage genutzte Menge wird bei Bedarf vergrößert oder reduziert. Dies ist beispielsweise der Fall, wenn plötzlich durch Änderungen in der Unternehmenspolitik neu Mitarbeitergruppen zu bestimmten Aktivitäten berechtigt werden und somit in die Gruppe der Leitenden Angestellten aufgenommen werden sollen. Besonders bei mehrfacher Verwendung der gleichen Menge vereinfacht dies die Wartung und reduziert Fehlerpotenzial, was zu einer qualitativ hochwertigen Ontologie führt. Auch erhöht die eindeutige Zuordnung von Konzepten oder Beziehungsarten zu Gruppen die Aussagekraft der Ontologie, da logische Zusammenhänge und Hierarchieebenen innerhalb eines Unternehmens klar ersichtlich sind (Beispielsweise durch eine Gruppe Abteilungsleiter). Ebenso ermöglicht dieses Muster die Darstellung von oder-Beziehungen: Ein Konzept C kann mit Konzept A oder B in Beziehung treten. Für die gültige Abbildung des Unternehmens in der Ontologie ist es dabei irrelevant, welches der Konzepte es ist. Ziel ist es nur, darzustellen, *dass* Konzept C eine Beziehung mit ausgewählten Konzepten eingehen kann.

4.2.4 Containermodell

Das abschließende Muster Containermodell hat das Ziel, Zusammengehörigkeit von Konzepten zu modellieren und orientiert sich dabei an den Eigenschaften des Composite-Patterns. Die Zusammengehörigkeit entsteht, wenn Konzepte logisch Teil eines anderen Konzepts sind oder sich direkt oder indirekt auf ein anderes Konzept beziehen.

Beispielsweise kann eine Bestellung Prüflose mit sich führen, die für die Qualitätssicherung relevant sind (Sh. Abbildung 12). Die Liste dieser Lose gehört logisch zur Bestellung, jedes einzelne Los wiederum zur Liste der Prüflose. Kann man diese Zusammengehörigkeit nun mithilfe einer containerartigen Struktur darstellen, so ist jedes Los, dass auf einer Liste steht, mit dieser verknüpft, da es sich logisch im Container Losliste befindet. Gleichzeitig ist es mit der dazugehörigen Bestellung verknüpft, da der Container Losliste im Container Bestellung eingebettet ist. Diese Modellierungsmöglichkeit verhindert ein langes Verketten von Konzepten und gleichzeitig den Verlust von Zugehörigkeits-

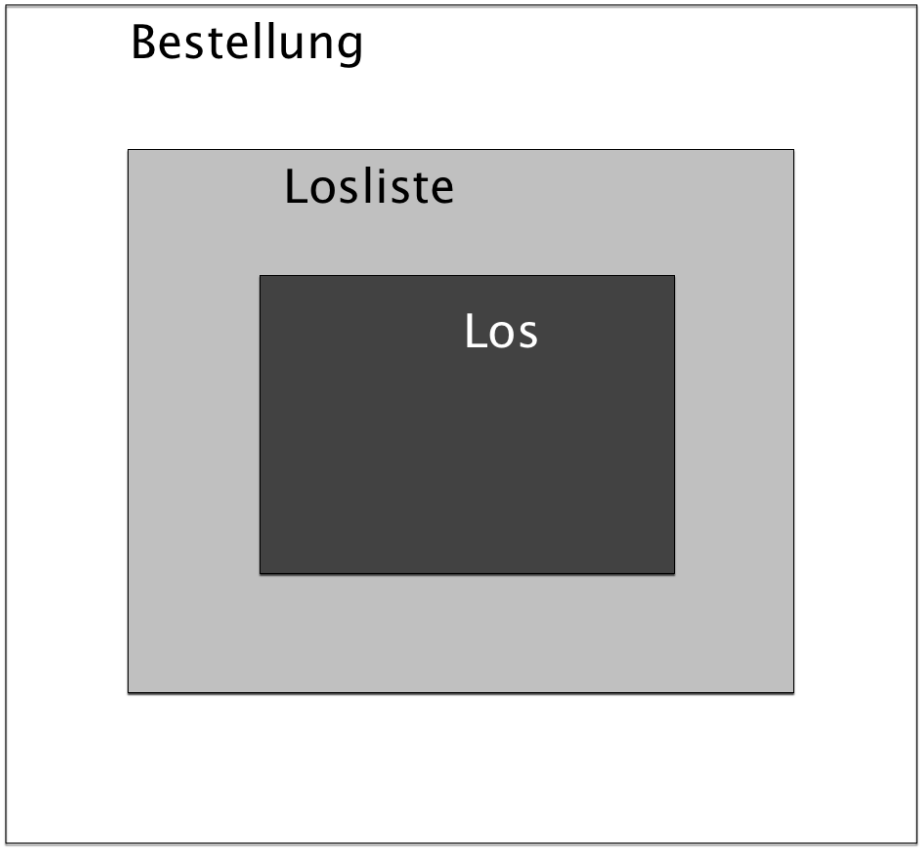

Abbildung 12: Beispiel Containermodell

informationen. Ein unmittelbarer Zusammenhang zwischen Los und Bestellung ist nun ersichtlich, ähnlich dem erläuterten Einstiegspunkt im Composite-Pattern.Los "weiß" nun unmittelbar, dass es mit Bestellung in Beziehung steht.

5 Vergleich der hergeleiteten Designmuster mit der Ausgangsstruktur

In diesem Abschnitt diskutieren wir, in wie weit die in Kapitel 4.2 ermittelten Designmuster mit der in Kapitel 2.5 vorgestellten Ausgangsstruktur darstellbar sind. Diese Darstellung wird anschließend bewertet. Für jedes der vorgestellten Muster erfolgt eine eigene Diskussion.
Lässt sich ein Muster angemessen in die Ausgangsstruktur übertragen, so bietet ein Ontologiesystem, das die Komponenten der Ausgangsstruktur abbilden kann, entsprechende Möglichkeiten zur Verwendung des von uns ermittelten Musters.

5.1 Verallgemeinerung und Spezialisierung

Die Ausgangsstruktur kann Verallgemeinerung und Spezialisierung nicht unmittelbar darstellen. Sie lässt sich jedoch durch den Einsatz von Rollenattributen annähern. Wir erläutern dies anhand des in Kapitel 4.2.1 verwendeten Beispiels:

Gehören sowohl der Einkäufer als auch der Marketingmanager zur Oberklasse Mitarbeiter, so werden Instanzen des Konzepts Mitarbeiter modelliert und mit Rollenattributen spezialisiert. Der Einkäufer ist somit ein Mitarbeiter mit der Rolle `Einkäufer`. Er verfügt ausschließlich über die allgemeinen Attribute, über die jeder Mitarbeiter verfügt. Um sein Budget, also das spezialisierte Attribut, darzustellen, gestaltet man das Rollenattribut komplexer, so dass es die Form {`Einkäufer, Budget`} erhält.

Wir können einfache Vererbung mit dieser Methode aussagekräftig darstellen. Wollen wir hingegen eine weitere Spezialisierung des Einkäufers, beispielsweise Materialeinkäufer, einführen, gibt es zwei Möglichkeiten: 1. Das Konzept erhält sowohl das Rollenattribut Einkäufer als auch das zusätzliche Rollenattribut Materialeinkäufer. Letzteres beinhaltet die für den Materialeinkäufer speziellen Attribute in Form eines komplexen Rollenattributs (sh. oben). 2. Man verzichtet auf das doppelte Rollenattribut und wählt das speziellste Konzept, in diesem Fall Materialeinkäufer, dass um alle in Einkäufer und Materialeinkäufer verwendeten Attribute ergänzt wird.

Die komplexen Attribute machen diesen Modellierungsweg fehleranfälliger und schwieriger zu warten als die klare Darstellung mittels Vererbungshierarchien. Dennoch lässt sich eine gewünschte Spezialisierung ohne Informationsverlust in die Ausgangsstruktur übertragen. Somit ist diese geeignet, als Kriterium für die Überprüfung auf Verallgemeinerung und Spezialisierung eines Ontologiesystems verwendet zu werden.

5.2 Gruppierung

Auch dieses Muster ist nicht unmittelbar durch die Ausgangsstruktur darstellbar. Eine mögliche Umgehungslösung ist es, ein zusätzliches Gruppierungsattribut einzuführen. Dieses führt für Konzepte einer Gruppierung die gleiche Identifikationsnummer (ID). Gehört ein Konzept zu mehreren Gruppierungen, so führt es statt einer ID eine ID-Liste der Form {`ID1, ID2,...`}. Dabei entsteht die Anforderungen, dass die Identifikations-

nummern über die gesamte Ontologie eindeutig sind.
Der Vorteil, dass eindeutig erkennbar ist, welche Elemente zu einer Gruppierung gehören und Gruppierungen wie Konstanten zu betrachten sind, geht bei dieser Darstellungsform verloren. Weiterhin modelliert man Beziehungen zu einer Gruppierung nun als n-äre Beziehung zwischen den einzelnen Gruppierungselementen und der dritten Partei, da eine Gruppierung keine nach außen hin geschlossene Einheit bildet, sondern aus IDs rekonstruiert wird. Dennoch lässt sich auch hier die gewünschte Information verlustfrei in die Ausgangsstruktur übertragten.

5.3 Beziehungen mit variablen Komponenten

Mit der Ausgangsstruktur kann man keine Beziehungen mit variablen Komponenten darstellen. Die einzige Lösung ist vollständiges Ausformulieren aller gewünschten Beziehungen. Somit entfällt der Bedarf für Mengen, die sich grundsätzlich wie Gruppierungen unter Verwendung von IDs zusammenfügen lassen.
Die Lösung führt dazu, dass die Vorteile dieses Designmusters entfallen: Die Wartbarkeit der Ontologie ist komplexer, da man ggf. jede Beziehung einzeln anpassen muss. Unübersichtlichtkeit ist ein weiterer Nachteil, sowie erhöhtes Fehlerpotenzial durch Redundanzen. Auch für diess Muster gilt jedoch, dass die gewünschte Information verlustfrei übertragen werden kann, da jede der Beziehungen einzeln ausformuliert wird.

5.4 Containermodell

Die Ausgangsstruktur bietet zwei Möglichkeiten, das Containermodell darzustellen: Komplexe Attribute und die Verkettung von Attributen.

Komplexe Attribute ermöglichen es, Attribute von Attributen zu modellieren. So ordnet man einer Bestellung das Attribut `Losliste` zu, dass wiederum die gesamten Prüflose enthält. Dies kann die Form haben `Losliste: {Prüflos 1, Prüflos 2, ... }, Loslisten Nummer,...` In diesem Fall sind Losliste und Prüflos keine Instanz von Konzept mehr, sondern fallen in die Gruppe der Attribute.

Der zweite Weg ist die Verwendung von verketteten Attributen: Eine Bestellung erhält eine ID, die wiederum bei der Losliste als Attribut `Bestellungs-ID` geführt wird. Ebenso erhält die Losliste eine ID, die analog bei jedem Prüflos der Liste steht. Über diese Verkettung kann vollständig rekonstruiert werden, welches Prüflos zu welcher Bestellung gehört.

Beide Wege ermöglichen eine verlustfreie Informationsübertragung in die Ausgangsstruktur, doch auch hier verliert man die Klarheit und Fehlerminimierung, die das Containermodell mit sich bringt.

5.5 Zusammenfassung

Zusammenfassend kann man sagen, dass wir alle erarbeiteten Designmuster mittels der ursprüngliche Minimalstruktur darstellen können, ohne Informationen über die Ontologie zu verlieren. Das am häufigsten verwendete Mittel sind zusätzliche Attribute, die

auch komplex sein können. Diese verlieren dabei ihre atomare Eigenschaft. Gleichzeitig verliert man bei jeder Übertragung die Vorteile, die jedes Designmuster mit sich führt und die zu einer präzisen, fehlerfreien und aussagekräftigen Ontologie beitragen. Kann ein Ontologiesystem also die Minimalstruktur abbilden, so lässt sich eine angemessene Ontolgie modellieren. Gleichzeitig wird sie nicht die Qualität erreichen, die ein Ontologiesystem ermöglicht, dass die komplexeren Designmuster abbilden kann. Daher ist es empfehlenswert, für die Modellierung ein Ontologiesystem zu wählen, dass mindestens einige, im Idealfall alle der vorgestellten Designmuster umsetzen kann. Wie genau die Implementierung der Muster aussieht, ist dabei irrelevant, solange sämtliche Eigenschaften der Muster realisiert werden.

6 Fazit und Ausblick

Es ist uns gelungen, fünf Designmuster zu ermitteln, die eine präzise und fehlerfreie Ontologieerstellung unterstützen. Diese konnten wir aus Unternehmensprozessen extrahieren, in dem wir allgemeingültigere Modellierungsprobleme identifizerten und Muster als Lösung für diese entwickelten. Ebenso konnten wir feststellen, dass sich alle Designmuster auf die Ausgangsstruktur zurückzuführen lassen, damit aber ein Verlust vieler der positiven Eigenschaften der Designmuster einhergeht.

Wir konnten feststellen, dass es in Unternehmensprozessen Gemeinsamkeiten gibt, auch wenn die Prozesse aus verschiedenen Geschäftsbereichen entstammen. Daher ist es lohnenswert, allgemeine Lösungen für die Modellierung von Unternehmen zu entwickeln. Allgemeine Lösungen führen weiterhin zu konsequenteren und vergleichbaren Modellen, was die Weiterverwendung derer fördert. Wir konnten erkennen, dass Ontologiemuster in Analogie zu Software Engineering Design Muster als solche allgemeinen Lösungen dienen können.

Für weitere Forschung könnte nun eine Anwendung der Designmuster als Kriterien auf ein konkrete Ontologiesysteme erfolgen. Erfüllt ein System nicht alle Kriterien, so könnten hierfür Modifikationen an den Mustern oder Umgehungslösungen entwickelt werden.

Ein zweiter Bereich ist die Betrachtung anderer Teilwelten als das Unternehmensumfeld. Beispielsweise könnten Prozesse aus dem privaten Umfeld analysiert werden um herauszufinden, ob für die Modellierung derer weitere Designmuster notwendig sind.

Literatur

[1] Freudenreich, T; Appel, S.; Frischbier, S.; Buchmann, A.: *DPL - Declarative Policy Language for Cyber-Physical Systems.* ICSE Hyderabad, India, 2014

[2] Freudenreich, T; Appel, S.; Frischbier, S.; Buchmann, A.: *Using Policies for Handling Complexity of Event-Driven Architectures.* ECSA 2014, pp. 114-129, Springer International Publishing Schweiz 2004

[3] van der Aalst, W.M.P: *Formalization and verification of event-driven process chains* Information and Software Technology 41 (1999) 639–650

[4] *www.softwareAG.com* (August 2014)

[5] *www.UML.org* (September 2014)

[6] *http://www.oodesign.com/composite-pattern.html* (August 2014)

[7] Daga,E., Presutti, V., Gangemi, A., Salvati, A.: *http://ontologydesignpatterns.org [ODP]*

[8] Presutti, V.; Gangemi, A. *Content Ontology Design Patterns as Practical Building Blocks for Web Ontologies* ISTC-CNR, Semantic Technology Lab, Italy Q. Li et al. (Eds.): ER 2008, LNCS 5231, pp. 128–141, 2008.

[9] *http://www.w3.org/TR/2012/REC-owl2-overview-20121211/* (August 2014)

[10] *http://www.softwareag.com/corporate/service/sap_ consulting/ipr/default.asp* (September 2014)

[11] Gangemi, A.; Presutti, V. *Ontology Design Patterns* Handbook on Ontologies, Springer Berlin Heidelberg 2009

[12] Coulouris, G., Dollimore, J., Kindberg, T. *Verteilte Systeme, Konzepte und Design* 3. Auflage, Pearson Studium München 2002

[13] Fernandéz-López, M., Gómez-Péres, A., Sierrea, J.P. & Sierra A.P. *Building a chemical ontology using Methonology and Ontology Design Environment* IEEE Intelligent Systems, Vol. 14 no. 1 pp. 37-46, 1999

[14] Calero, C., Ruiz, F., Piattini, M. *Ontologies for Software Engineering and Software Technology* Springer-Verlag Berlin Heidelberg 2006

[15] Gamma, E., Helm, R., Johnson, R., Vlissides, J. *Design Patterns - Elements of Reusable Object-Oriented Software* 20. Auflage, Person Education Upper Saddle River, USA 2000

[16] Bjorner, D. *Software Engineering 3 - Domains, Requirements and Software Design* Springer-Verlag Berlin Heidelberg 2006

[17] Gasevic, D., Djuric, D., Devedzic, V. *Model Driven Engineering and Ontology Development* 2. Auflage, Springer-Verlag Berlin Heidelberg 2009

[18] Kuzniarz, L., Staron, M. *Generating Domain Models from Ontologies* OOIS 2002, pp. 160-166 Springer-Verlag Berlin Heidelberg 2002