Verknüpfung von Domänenwissen für ein Ontologie-basiertes IT-Management

Dissertation

zur Erlangung des akademischen Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)
im Fachbereich 16 — Elektrotechnik / Informatik
der Universität Kassel

vorgelegt von
Andreas Textor
Tag der Disputation: 20.12.2017

Gutachter: Prof. Dr. Kurt Geihs
Prof. Dr. Reinhold Kröger

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbständig, ohne unerlaubte Hilfe Dritter angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Dritte waren an der inhaltlichen Erstellung der Dissertation nicht beteiligt; insbesondere habe ich nicht die Hilfe eines kommerziellen Promotionsberaters in Anspruch genommen. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren durch mich verwendet worden.

Weinstadt, 19.08.2017

Andreas Textor

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Quellcodes

1 Einleitung

1.1 Motivation

Mit der steigenden Komplexität von IT-Systemen wird auch die Verwaltung dieser Systeme komplexer. Die Disziplin, die sich mit der Überwachung und Verwaltung aller IT-bezogener Ressourcen in einer Organisation beschäftigt, wird als *Information Technology Management* (IT- Management) bezeichnet. Hierbei werden nicht nur der Betrieb technischer Komponenten (Hardware und Software) betrachtet sondern auch die Planung zur Weiterentwicklung der IT-Infrastruktur.

Den Fragestellungen, die sich durch die Komplexität auf der Seite des Betriebs und der Überwachung von IT-Systemen stellen (also Hardware und Software), wird auf zweierlei Weise begegnet: Einerseits durch das *integrierte IT-Management*, bei dem versucht wird, unterschiedliche, existierende Sichten auf die Systeme nicht in Isolation zu betrachten, sondern die Zusammenhänge zu berücksichtigen, die zwischen Aspekten teilweise auch nur implizit bestehen. Es wird also das Zusammenspiel zwischen Anwendungen, Informationen, Systemen und Netzen betrachtet. Andererseits wird mit dem *automatisierten IT-Management* angestrebt, den manuellen Anteil an Überwachung und Pflege komplexer Systeme soweit wie möglich zu reduzieren und im Idealfall komplett zu eliminieren. Das Ziel ist dabei nicht nur die Senkung von Kosten durch geringeren Personaleinsatz und weniger Ausfallzeiten, sondern auch eine Steigerung der Resilienz der Systeme. Ist das System allein durch die Vorgabe abstrakter Regelsätze dazu in der Lage, auf sich ändernde Gegebenheiten zu reagieren und Fehlerzustände ohne äußere Eingriffe zu beheben, so spricht man auch von *autonomic computing*. Ein bekannter Vertreter ist hier insbesondere der durch IBM initiierte und vorangetriebene Ansatz des MAPE-K [IBM06]: In einer Regelschleife liest eine Monitoring-Komponente Informationen vom zu verwaltenden System, die in einer Analyse-Phase aufbereitet werden, anschließend in der Planungs-Phase in Handlungsvorschriften übersetzt werden, die schließlich in einer Ausführungs-Phase (Execute) zu einer Steuerung des Systems führen. Eine gemeinsam genutzte Wissensbasis (Knowledge) wird dabei in allen Phasen verwendet und beinhaltet dabei sowohl das Wissen über das Regelwerk, aus dem die Handlungsvorschriften abgeleitet werden, als auch die zur Laufzeit anfallenden Informationen aus dem zu verwaltenden System.

Ideen aus dem integrierten und dem automatisierten IT-Management wurden bereits erfolgreich kombiniert. Dabei wurden Management-Systeme umgesetzt, die sowohl unterschiedliche

technische Sichten integrieren als auch die daraus resultierenden kombinierten Modelle als Grundlage der gemeinsam genutzten Wissensbasis einer MAPE-K-Regelschleife nutzen. Bei dem Entwurf einer solchen Lösung müssen zahlreiche Einzelprobleme gelöst werden: Modelle der zu integrierenden Sichten müssen in eine gemeinsame Struktur gebracht und Konzepte entsprechend ihrer Bedeutung integriert werden; Adapter zur Interaktion mit den Komponenten der zu verwaltenden Systeme müssen entwickelt werden; Regelsätze müssen definiert werden und schließlich muss auch das Managementsystem implementiert werden.

Parallel zur Komplexität der IT steigt in Unternehmen auch der Bedarf, IT-Strukturen besser als zuvor in die restlichen Unternehmensstrukturen einzubinden, mit anderen Worten, besser nachvollziehbar zu machen, welche Teile der IT wie genau und in welchem Umfang zu den eigentlichen Geschäftszielen beitragen und beitragen können. Es findet also eine Entwicklung statt von der Betrachtung der IT als reiner Bereitsteller von Technologie oder Dienstleistungen hin zu einer Betrachtung als Schlüssel zur Umsetzung von Fragestellungen der Geschäftsseite. Das Vorantreiben dieser Entwicklung ist Aufgabe des Unternehmensmanagements.

Während bis in die 2000er Jahre bei dem Abgleich zwischen der Geschäftsseite und der IT, die sie befähigen soll, häufig vom *Business-IT-Alignment* gesprochen wurde, hat sich mit der fortschreitenden Konkretisierung seiner Aufgaben hierzu eine klarere, besser definierte Disziplin herausgebildet. Die *IT-Governance* hat die Sicherstellung der aus Sicht der Geschäftsführung gewünschten Funktionsweise der Unternehmens-IT zur Aufgabe und ist dazu mit der Erstellung und Verwaltung von geeigneten Prozessen und Strukturen betraut. Hierbei werden neben den ursprünglich im Business-IT-Alignment angesiedelten Aufgaben auch Compliance, Ressourcenmanagement, Erfolgsmessung und Risikomanagement als Teil der IT-Governance verstanden. Die IT-Governance befasst sich also damit, was IT-Systeme leisten sollen und wie sie damit das Geschäft unterstützen, weniger damit, wie dies im Detail erreicht werden soll.

Für die Umsetzung der IT-Governance im Unternehmen hat sich international das Rahmenwerk COBIT (Control Objectives for Information and Related Technology) etabliert. Dabei wird der Bogen von Geschäftszielen über Prozesse bin zu konkreten Handlungsanweisungen geschlagen, der auch Verantwortlichkeiten und Zwischenergebnisse berücksichtigt. COBIT wird in Form einer Sammlung von Publikationen herausgegeben, die detailliert die Zusammenhänge zwischen den Fragestellungen der Geschäftsseite und der IT beschreiben. Ein formales Modell, das die Zusammenhänge in maschinenlesbarer Form beschreibt, ist nicht Teil der Spezifikation.

Auf der einen Seite findet also eine starke Entwicklung in Richtung der Automatisierung der Verwaltung von IT-Systemen auf rein technischer Ebene statt, auf der anderen Seite gewinnt IT-Governance an Bedeutung, um die Integration unterschiedlicher Sichten im Unternehmen mit den IT-Systemen zu verbessern. Es wird daher eine Konvergenz beider Seiten angestrebt, um existierende Vorteile besser nutzbar zu machen und bestehende Lücken zu schließen. Verfahren zur Umsetzung von automatisiertem IT-Management müssen sich in einer Richtung

weiterentwickeln, die es erlaubt, den Fokus von der rein technischen Sicht und der automatischen Regelung von Systemen auch auf andere Unternehmenssichten zu erstrecken. Auch diejenigen Sichten, die als Teil der IT-Governance verstanden werden, müssen im automatisierten IT-Management berücksichtigt werden. Standards und Rahmenwerke aus diesen anderen Sichten müssen dagegen stärker formalisiert werden, nicht nur um eine weitgehende Automatisierbarkeit erreichen zu können, sondern auch um die Entstehung von Datensilos durch herstellerspezifische Implementierungen zu vermeiden. Aufwände, die durch die Überbrückung von Differenzen zwischen unterschiedlichen Implementierungen entstehen, verursachen unnötige Kosten und erschweren die angestrebte Automatisierbarkeit.

Diese Arbeit hat das Ziel, Vorgehensweisen aus dem automatisierten IT-Management und der IT-Governance zu verbinden. Dabei soll die Grundlage für eine erweiterbare Automatisierung geschaffen werden, die technische Sichten (beispielsweise Network Management, Virtual Machine Management und Storage Management) ebenso wie nicht-technische Sichten berücksichtigen kann (beispielsweise Cost Management, Risk Management und Geschäftsprozesse). Diese Sichten werden auch als Domänen bezeichnet, da Aufgaben, die von Verantwortlichen in ihrem Kontext bearbeitet werden, sich in der Regel nur auf Zusammenhänge innerhalb der Domäne beziehen. Dadurch soll die Formulierung von Regelwerken für ein automatisches Managementsystem möglich werden, die Bezug sowohl auf messbare Größen, beispielsweise Monitoring-Informationen aus laufenden Systemen, als auch auf definierte strukturelle Zusammenhänge nehmen können, die auch bisher weitestgehend implizite Beziehungen aus den nicht-technischen Domänen beinhalten.

Hierdurch soll zunächst erreicht werden, dass Automatisierungsansätze auf alle Domänen angewendet werden können, die in Form von geeigneten Modellen zugefügt werden. Als Beispiel soll hier ein automatisiertes Managementsystem dienen, das als technische Domäne ein Speichersystem verwaltet, das aus unterschiedlichen Speichersubsystemen besteht. Durch entsprechende Regelwerke wird sichergestellt, dass Daten auf alternative Subsysteme migriert werden, wenn die Speicherkapazität nicht mehr ausreicht, oder wenn der Durchsatz (I/O-Operationen pro Sekunde) gesetzte Grenzen unterschreitet. Sowohl Kapazität als auch Durchsatz sind dabei Größen, die durch Monitoring des verwalteten Speichersystems erfasst werden können. Sollen nun in diesem automatisierten Management weitere, nicht-technische Aspekte berücksichtigt werden, muss das Managementsystem durch entsprechende Modelle und Regelsätze erweiterbar sein:

- Durch Compliance-Vorgaben, die entweder Entscheidungen der Unternehmensleitung oder gesetzliche Vorgaben sind, kann zum Beispiel aus Datenschutzgründen verlangt werden, dass bestimmte Daten nur in Rechenzentren an bestimmten Orten vorgehalten werden dürfen. Um diese Vorgaben im automatisierten Speichermanagement zu berücksichtigen, wird einerseits der Bezug dazu durch ein Compliance-Modell hergestellt, andererseits kann die Auszeichnung bestimmter Untermengen der Daten nötig werden, sodass eine Prüfung und Entscheidung möglich ist.

- Für eine bessere Nachvollziehbarkeit, welche Speicherressourcen für welche Anwendungen und darüber hinaus in welchen Geschäftsprozessen genutzt werden, wird der Bezug durch die Verbindung mit einem Prozessmodell hergestellt. Ist im Prozessmodell beispielsweise hinterlegt, dass eine Bearbeitung nur an bestimmten Wochentagen vorgenommen wird, kann dies in den Regeln für das Speichersystem berücksichtigt werden und leistungsbeeinträchtigende Vorgänge wie eine Spiegelung von Daten oder die Generierung von Reports zu anderen Zeitpunkten ausgeführt werden.
- Im Zuge des Risk Managements kann die Vorgabe gemacht werden, dass bestimmte Daten zur Kontrolle von Risiken durch Feuer- oder Flutschäden auf Systeme an physikalisch unterschiedlichen Orten gespiegelt werden müssen. Um diesen Bezug maschinenlesbar abzubilden, wird das Managementsystem durch ein Risk-Management-Modell, passende Regelsätze und das Mapping zum Speichermanagement-Modell ergänzt.

Dies sind nur einige Beispiele, die zeigen, wie die nicht-technischen Domänen einen direkten Einfluss nehmen auf die Entscheidungen, die ein automatisiertes Managementsystem zu treffen hat. Insbesondere die Verbindung zu Geschäftsprozessen kann bedeutend komplexer werden, wenn die Funktionalitäten von Business Process Execution Engine und automatisiertem Managementsystem sich weiter annähern. Anpassungen für Regelsysteme, die in Szenarien wie den oben genannten Beispielen zum Einsatz kommen sollen, werden heute in der Regel manuell eingepflegt und sind mit entsprechendem Aufwand verbunden.

Außerdem werden Möglichkeiten zur Abfrage über die Zusammenhänge zwischen den Domänenmodellen verbessert. Durch die zur Realisierung der Automatisierung nötige Formalisierung der Domänenmodelle wird bereits die Grundlage hierfür geschaffen. Während domänenübergreifende Zusammenhänge zwischen technischen Modellen hauptsächlich innerhalb von Automatisierungsregeln wichtig sind, können die Zusammenhänge, die nicht-technische Aspekte berücksichtigen, direkt zur IT-Governance beitragen. Damit kann zur Beantwortung von Fragen beigetragen werden, die auf Abstraktionsebenen näher an der Geschäftsseite als auf der IT-Seite liegen. Werden von der Geschäftsführung Unternehmensziele und Kennzahlen zur Messung des Erreichungsgrades dieser Ziele definiert, können die im automatisierten Managementsystem verwalteten Informationen aus verschiedenen Quellen zusammen mit den domänenübergreifenden inhaltlichen Verbindungen genutzt werden, um Kennzahlen automatisch zu berechnen.

1.2 Fragestellung und Lösungsansätze

Die Kernfrage ist, wie diese angestrebte Konvergenz von IT-Governance und automatisiertem IT-Management erreicht werden kann. Der Lösungsvorschlag in dieser Arbeit lässt sich grob in drei Einzelschritte untergliedern:

1. Es wird eine Möglichkeit benötigt, Modelle strukturell unterschiedlicher Domänen in einer Weise auszudrücken, die nicht nur eine automatisierte Verarbeitung ermöglicht, sondern auch die Beschreibung der Beziehungen zwischen Entitäten untereinander. Beziehungen können also sowohl zwischen Entitäten innerhalb einer Domäne bestehen, als auch zwischen Entitäten unterschiedlicher Domänen; die eingesetzte Formalisierung muss solche Beziehungen ausdrücken können. Für die Unterstützung der automatisierten Verarbeitung ist zudem wichtig, dass die Modelle nicht nur schematisches Wissen wie Taxonomien enthalten können, sondern auch Instanzdaten, die konkrete Entitäten repräsentieren, wie beispielsweise Rechner, Personen oder Prozesse. Nicht nur der zugrundeliegende Formalismus sondern insbesondere auch der strukturelle Aufbau der Domänenmodelle muss die nachträgliche Erweiterbarkeit ermöglichen: Das nachträgliche Zufügen weitere Domänenmodelle muss möglich sein, ohne dass die bestehende Wissensbasis dadurch in einen inkonsistenten Zustand versetzt wird, also Aussagen aus vorher vorhandenenen und neu zuzufügenden Domänenmodellen sich widersprechen.

2. Ebenfalls nötig ist eine Architektur für ein Laufzeitsystem, das diese Modelle verwalten und abfragbar machen kann. Dabei müssen Beziehungen auf der schematischen Ebene ebenso wie auf Instanzebene in Abfragen möglich sein. Die Verwaltung der Modelle beinhaltet dabei auch die Möglichkeit zur Definition und Überprüfung von Abhängigkeiten zwischen Domänenmodellen untereinander. Die Architektur muss die Infrastruktur für die Integration externer Informationsquellen bieten, aus denen zur Laufzeit aktuelle Daten für das automatisierte Management bezogen werden. Neben eben Domänenmodellen und Laufzeitinformationen muss das Laufzeitsystem auch die Regelmengen verwalten und auswerten können, die für die automatische Regelung nach dem oben beschriebenen MAPE-K-Prinzip notwendig sind.

3. Das etablierte Rahmenwerk für IT-Governance, COBIT, soll formalisiert und in einer Form verfügbar gemacht werden, die die Nutzung mit den Punkten 1. und 2. erlaubt. Das bedeutet, dass die in COBIT definierten Zusammenhänge zwischen den Abstraktionsebenen von der Geschäftssicht bis hin zur IT-Sicht so referenzierbar gemacht werden, dass weitere externe Informationsmodelle durch die Abbildung zu Entitäten der jeweils passenden COBIT-Abstraktionsebene integriert werden können. Hierdurch wird ein Referenzmodell geschaffen, das als „Rückgrat“ für die domänenübergreifende Verbindung technischer und nicht-technischer Informationsmodelle eingesetzt werden kann. Diese Modelle müssen jedoch auch, je nach Anwendungsfall, durch eine Kombination von Transformationen aus bestehenden andersweitigen Formalisierungen (z.B. XML-Schemata) und Anwendung von entsprechenden Entwicklungsprozessen in die geeignete Form gebracht werden.

Die grundsätzliche Formalisierung von Domänenmodellen wird in dieser Arbeit durch den Einsatz von *Ontologien* erzielt. Der ursprünglich aus der Philosophie entlehnte Begriff Ontologie beschreibt die Disziplin, die sich mit allem Seienden befasst. In der Informatik wird unter einer

Ontologie ein formales Informationsmodell verstanden, das Entitäten innerhalb einer Domäne sowie die Beziehungen zwischen den Entitäten beschreibt. Im Gegensatz zu Modellierungsformen wie Entity-Relationship-Diagrammen oder UML-Modellen kann dabei auch die Semantik des Modells durch Restriktionen auf Mengen sowie durch Auszeichnung von Relationen mit Eigenschaften wie Transitivität, Reflexivität oder Symmetrie exakt beschrieben werden.

Für die Beschreibung und die Verarbeitung von Ontologien existieren bereits Grundlagentechnologien wie Ontologie-Beschreibungssprachen, die auf formalen Logiken basieren, sowie Abfragesprachen, mit denen Untermengen von Informationen aus einer Ontologie selektiert werden können. Die Schwierigkeit der Umsetzung der oben beschriebenen Schritte liegt allerdings im Erfassen und einer korrekten Kodierung der Semantik der Domänenmodelle. Diese Kodierung ist dabei maßgeblich verantwortlich dafür, inwieweit die nachträgliche Anpassbarkeit und Erweiterbarkeit von Domänenmodellen möglich ist und in welcher Form eine Verarbeitbarkeit durch das Laufzeitsystem möglich ist. Dies umfasst beispielsweise Einschränkungen der Ausdrucksmächtigkeit, es kann also nötig sein, in der Modellierung auf bestimmte Strukturen zu verzichten, um bei der späteren Verarbeitung ein gewünschtes Laufzeitverhalten oder selbst Entscheidbarkeit zu erreichen.

Domänenmodelle werden in dieser Arbeit durch formale Ontologien beschrieben, da diese einerseits eine domänenunabhängige Modellierungsform darstellen und daher grundsätzlich geeignet sind für die Erfassung der Semantik der unterschiedlichen Domänen[1], andererseits bereits erfolgreich für die Informationsintegration technischer Modelle im automatisierten IT-Management verwendet wurden. Als Teil existierender Grundlagentechnologien können verschiedene Methoden des *Ontology Engineering* betrachtet werden. Diese müssen jedoch untersucht und für den Einsatz in einer Laufzeitumgebung adaptiert und erweitert werden, um anwendbar zu sein. Entwicklung und Weiterentwicklung der Methoden zur Umsetzung von Ontologien im automatisierten Management sind wesentlicher Bestandteil für die Realisierung eines formalen COBIT-Informationsmodells. Für die Erfüllung der Anforderungen in 1. wird ein Modulkonzept entwickelt, das Domänenontologien, Regelmengen und benötigte programmatische Erweiterungen des Laufzeitsystems (z.B. für Berechnungsvorschriften oder Adapter zu externen Datenquellen) als abgeschlossene Einheiten kapselt.

Das Laufzeitsystem zur Erfüllung von 2. nutzt zur Beschreibung seiner gemeinsamen Wissensbasis (Knowledge-Komponente) ebenfalls Ontologien. Um dies zu realisieren, müssen zusätzlich zu den oben beschriebenen Schritten zur Umsetzung einer MAPE-K-Regelschleife zusätzliche Herausforderungen angegangen werden: Ontologien beschreiben zunächst die schematische, strukturelle Seite eines Domänenmodells, das Laufzeitsystem muss zusätzlich aber auch Instanzdaten verwalten, die konkrete Entitäten der verwalteten Domänen repräsentieren. Dies kann Elemente eines technischen Systems beinhalten, beispielsweise die einzelnen physikalischen oder logischen Medien eines Speichersystems, aber auch konzeptuelle Elemente wie Geschäftsprozesse oder deren Bestandteile.

[1]Soweit die Semantik in der jeweils eingesetzten Ontologie-Sprache bzw. -Ausprägung ausdrückbar ist

Obwohl mit der Schaffung einer formal verarbeitbaren Version des De-Facto-Standard-Rahmenwerks für IT-Governance in 3. ein wesentlicher Schritt zur besseren domänenübergreifenden Automatisierung innerhalb von Unternehmen erreicht wird, sind die entwickelten Vorgehensweisen nicht auf die COBIT-Ontologie beschränkt. Konzepte zur Entwicklung der Ontologien, Modularisierung und die Verarbeitung durch das Laufzeitsystem sind generisch anwendbar. Dadurch ist die Erweiterung der Automatisierung auf beliebige weitere Domänen möglich, für die entsprechende Ontologien vorliegen oder entwickelt werden.

1.3 Methodischer Ansatz

In der Arbeit werden zunächst die Anforderungen definiert, die an die Gesamtarchitektur gestellt werden. Dies beinhaltet sowohl die Modellierung von Domänenontologien, insbesondere mit Hinblick auf die Realisierung der COBIT-Ontologie, als auch die Eigenschaften, die vom Laufzeitsystem erfüllt werden müssen. Dabei werden die in Abschnitt 1.2 beschriebenen Rahmenbedingungen berücksichtigt. Benötigte Anpassungen an Methodiken des Ontology Engineering werden in einer Weise vorgenommen, die Ontologien für den Einsatz im Kontext eines Laufzeitsystems anwendbar machen.

Anhand der Anforderungen werden die einzelnen Bestandteile unabhängig von konkreten Implementierungstechnologien entwickelt. Die Architektur und der Aufbau der Komponenten des Laufzeitsystems werden aus den Anforderungen abgeleitet. Die Entwicklung der COBIT-Ontologie folgt aus einer Analyse des Rahmenwerks und setzt die zuvor beschriebenen Anforderungen an die Ontologiemodellierung um.

Eine darauf folgende Beschreibung einer prototypischen Umsetzung des Laufzeitsystems bildet die Grundlage für eine Fallstudie, die zur Evaluation des Ansatzes eingesetzt wird. In der Fallstudie wird ein Projekt beschrieben, das zusammen mit Partnern aus der Industrie und der öffentlichen Verwaltung durchgeführt wurde und in dem ein System zum automatisierten Speichermanagement mithilfe der beschriebenen Architektur weiterentwickelt wurde. Die Evaluation beschreibt die Rahmenbedingungen, die Schritte zur Umsetzung einer Lösung innerhalb der Fallstudie und die erzielten Ergebnisse.

Der Aufbau des Laufzeitsystems und die praktischen Eigenschaften der entwickelten Ontologien werden durch den Einsatz in der Fallstudie sichergestellt und mit den in der Analyse aufgestellten Anforderungen abgeglichen.

1.4 Aufbau und Ergebnisse der Arbeit

Die Arbeit ist in drei Abschnitte eingeteilt: Abschnitt I stellt grundlegende Begriffe und Technologien vor, Abschnitt II beschreibt das Konzept der Architektur und der Umsetzung von COBIT als Ontologie und Abschnitt III stellt die prototypische Umsetzung sowie die Fallstudie vor. Im Folgenden wird der Inhalt der einzelnen Kapitel zusammengefasst.

Kapitel 2 beschreibt Begriffe und Unterdisziplinen des IT-Managements und der IT-Governance, sowie Gemeinsamkeiten und Unterschiede. Das Kapitel gibt außerdem einen Überblick über den Aufbau und den Inhalt des IT-Governance Rahmenwerks COBIT.

Im Kapitel 3 werden die Grundlagen von Ontologien und verwandten semantischen Technologien beschrieben. Dabei wird sowohl auf die formale Repräsentation von Wissen eingegangen, die die Grundlage für die modernen Ontologie-Beschreibungssprachen bilden, als auch auf den Stand der Technik konkreter Sprachen und Formate. Es wird eine Einführung in das Ontology Engineering gegeben, das sich, analog zum Software Engineering, mit der strukturierten Planung, Entwicklung, Anpassung und Pflege von Ontologien beschäftigt.

In Kapitel 4 wird der Stand der Technik zu den einzelnen Problembereichen untersucht, die in dieser Arbeit zum Einsatz kommen. Dazu werden verwandte Arbeiten vorgestellt, die sich mit Aspekten des automatisierten IT-Managements, der IT-Governance, der Formalisierung als Ontologie von Domänenmodellen im Allgemeinen und COBIT im Speziellen sowie aus Vereinigungen und Schnittmengen dieser Aspekte befassen.

Kapitel 5 stellt die Anforderungen an das Laufzeitsystem und zu entwickelnde Domänenontologien vor. Die Architektur, die anhand der Anforderungen entworfen wurde, wird erläutert. Es werden die Komponenten des Laufzeitsystems und die Zusammenhänge zwischen ihnen erklärt.

Das Kapitel 6 analysiert das COBIT-Rahmenwerk, und das detaillierte Vorgehen zur Konstruktion der COBIT-Ontologie wird erklärt. Dabei wird darauf eingegangen, welche Teile aus der COBIT-Spezifikation von der Ontologie abgedeckt werden und wie die abgedeckten Teile in der Ontologiebeschreibung umgesetzt werden.

Im Kapitel 7 wird die prototypische Implementierung des Laufzeitsystems beschrieben. Es wird erklärt, wie die für die Implementierung gewählten Technologien eingesetzt werden, um die in Kapitel 5 beschriebene Architektur umzusetzen.

Kapitel 8 beinhaltet die Evaluation der Architektur und beschreibt dazu die Randbedingungen, Vorgehensweisen und Ergebnisse der umgesetzten Fallstudie.

Eine Zusammenfassung der Ergebnisse der Arbeit und ein Ausblick über zukünftige Weiterentwicklungen wird schließlich in Kapitel 9 gegeben.

Teil I

Grundlagen

2 IT Service Management und IT-Governance

2.1 Ziele und Aufgaben

2.1.1 IT Service Management

Unter *Information Technology Management* (IT-Management) versteht man die Überwachung und Verwaltung aller IT-bezogenen Ressourcen in einer Organisation. Dies beinhaltet sowohl technische Ressourcen (z.B. Hardware, Software und Daten) als auch personelle Ressourcen. Dabei werden nicht nur der Betrieb, sondern auch die Planung zur langfristigen Weiterentwicklung der IT betrachtet [MBK+12].

Um mit der Vielzahl von einzelnen Aufgaben umgehen zu können, die aus dieser breiten Definition folgt, wurden im Laufe der Zeit mehrere Klassifikationen entwickelt, die das IT-Management nach unterschiedlichen Kriterien in Teilbereiche unterteilen. Dabei sollen die einzelnen Teilbereiche die Festlegung von Zuständigkeiten und eine inhaltliche Strukturierung erreichen. Viele existierende Klassifikationen sind dabei aus dem Bedarf nach technischen Umsetzungen entstanden und berücksichtigen nicht oder nur am Rande weitergehende organisatorische Aspekte, wie z.B. Geschäftsprozesse.

Ein Beispiel für eine Klassifizierung des IT-Managements ist die Open Systems Interconnection (OSI) System Management Spezifikation [ISO98]. Sie definiert die fünf Kategorien Fehler-Management (Fault Management), Konfigurations-Management (Configuration Management), Abrechnungs-Management (Accounting Management), Leistungs-Management (Performance Management) und Sicherheits-Management (Security Management).

Eine andere, häufig genutzte Möglichkeit für eine solche Klassifizierung ist nach System-Schichten. So teilen Hegering et al. [HAN99] das IT Management auf in Netzmanagement, Systemmanagement, Informationsmanagement, Anwendungsmanagement und Dienstmanagement, wobei eine Schicht von den jeweils darunter liegenden abhängt. In der logischen Fortführung liegt über diesen IT Management-Schichten das Enterprise Management. Jede Schicht muss horizontal, und alle Schichten sollen vertikal integriert werden.

Entwirft man eine Klassifikation, die sich nicht an technischen, sondern an organisatorischen Grenzen orientiert, so lässt sich als eine Unteraufgabe das *IT Service Management* (ITSM) definieren. Unter ITSM versteht man nach Van Bon eine Menge von Prozessen, die zusammenwirken, um die Qualität von eingesetzten IT-Services sicherzustellen. Dies geschieht anhand von Service-Vereinbarungen, auf die man sich mit dem Kunden (dem Nutzer der IT-Services) geeinigt hat ([Van02], Seite xiii). Innerhalb des ITSM versteht man nach Betz unter einem IT-Service die Manifestation einer IT-Fähigkeit, die aus Geschäftssicht greifbar ist und eine Funktion liefert, die einen Mehrwert darstellt ([Bet07], Seite 38). Die Abstraktion eines solchen Service liegt in diesem Verständnis also weit über der einer technischen Umsetzung, wie z.B. Services in einer Service-Orientierten Architektur.

ITSM umfasst die Planung, Umsetzung und den Betrieb von IT-Services, kurz gesagt, den gesamten Lebenszyklus. Aus diesem Grund wird das Kompendium der *IT Infrastructure Library* (ITIL) auch als *Lifecycle Suite* bezeichnet [SCL+11]. ITIL ist der De-Facto-Standard für „Best Practices“ beim ITSM [HZB04, HZB05], der seit 1989 entwickelt wird. Bis 2001 war die britische Central Computer and Telecommunication Agency (CCTA) verantwortlich, von 2001 bis 2013 die britische Behörde Office of Government Commerce (OGC) und seit Juli 2013 liegt die Verantwortlichkeit bei AXELOS, einem Joint Venture des Unternehmens Capita und des ebenfalls britischen Cabinet Office. Die aktuelle Version von ITIL ist *ITIL 2011*, die 2011 die Vorversion *ITIL V3* von 2007 ablöste. ITIL wurde und wird zwar von britischen Institutionen entwickelt, es hat es sich aber inzwischen auch international durchgesetzt.

Die Spezifikation von ITIL ist in fünf separate Bände aufgeteilt, die sich mit jeweils einer Phase des IT-Service-Lebenszyklus beschäftigen. Die einzelnen Phasen und Bände sind dabei:

- Servicestrategie (IT Service Strategy, [Can11])
- Serviceentwurf (IT Service Design, [Hun11])
- Serviceüberführung (IT Service Transition, [Ran11])
- Servicebetrieb (IT Service Operation, [Ste11])
- Kontinuierliche Serviceverbesserung (IT Continual Service Improvement), [Llo11])

Die Servicestrategie bildet den Beginn des Lebenszyklus und definiert die Rahmenbedingungen für die darauf folgenden Phasen. In der Phase Serviceentwurf werden neue Services umgesetzt oder bestehende angepasst, die anschließend für die Einführung in den Produktivbetrieb bereitstehen. Die Serviceüberführung befasst sich dann mit dieser Einführung, am Ende der Phase befinden sich die Services im operativen Betrieb. Der Servicebetrieb kümmert sich um die Aufrechterhaltung der laufenden Services, und kontinuierliche Serviceverbesserung soll schließlich die Schnittstellen, Verfahren und Prozesse im Unternehmen optimieren.

ITIL versucht also, Erkenntnisse und daraus abgeleitet sinnvolle Vorgehensweisen für alle Phasen des Lebenszyklus von IT-Services als Referenzmodell aufzubereiten. Dabei ist ITIL so angelegt, dass der Kundenfokus nicht aus den Augen verloren wird – IT-Services sollen immer unter Beachtung von wirtschaftlichen Kriterien entworfen und gepflegt werden. Das bedeutet auch, dass die umgesetzten geschäftlichen Anforderungen möglichst effektiv erreicht werden sollen. Zusätzlich dient ITIL auch als Grundlage für Planung und Organisation innerhalb der für die Services verantwortlichen IT selbst, indem Prozesse und die Beziehungen zwischen ihnen transparenter gemacht und besser aufeinander abgestimmt werden können.

Trotz der zunehmenden Verbreitung von ITIL gab es auch einige Kritik, die z.T. durch die laufenden Verbesserungen in den neueren Versionen bereits adressiert wurden. Ein wiederholter Kritikpunkt ist die unterschiedliche Qualität der einzelnen Abschnitte: In der Aufteilung von ITIL vor Version 3 wurden die Abschnitte zu *Service Delivery* und *Service Support* besser aufgenommen als *Application Management* und *Security Management*. Außerdem enthielt ITIL keinen Abschnitt zu IT Portfolio Management, also der systematischen Verwaltung von Projekten und Aktivitäten innerhalb der IT-Abteilungen ([Bet07], Seite 39). Van Bon stellt dazu fest, dass ITIL sich selbst nicht als Rahmenwerk oder kohärentes Modell versteht, sondern nur als eine Sammlung von Empfehlungen, und damit auch keinen Anspruch auf Vollständigkeit oder notwendigerweise auf gleiche Berücksichtigung aller betrachteten Aspekte stellt ([Van02], Seite 220). Seit ITIL V3 ist IT Portfolio Management als Teil der Servicestrategie enthalten. Application Management ist detaillierter als zuvor im Servicebetrieb aufgegangen, Security Management ist nun Teil des Serviceentwurfs. ITIL wird in der aktuellen Version zwar immer noch als Sammlung von Best Practices bezeichnet, aber selbst Frances Scarff, der „Head of Best Management Practice“ des Cabinet Office spricht im Vorwort des Bandes Servicestrategie vom „ITIL framework“.

Neben ITIL existiert seit 2005 der Standard ISO/IEC 20000, der nachprüfbare Anforderungen an das ITSM definiert. Der Standard besteht aus zwei Teilen: 20000-1 („Service management system requirements“, [ISO11]) enthält verbindliche Angaben darüber, wie IT-Services entworfen, eingeführt und kontinuierlich verbessert werden müssen, so dass es für Betreiber und Kunden gleichermaßen einen Gewinn darstellt. Dieser Teil wurde 2011 aktualisiert und wird in der aktuellen Version als ISO/IEC 20000-1:2011 bezeichnet. Der zweite Teil 20000-2 („Guidance on the application of service management systems“, [ISO12a]) enthält Hinweise zur Umsetzung des ersten Teils und wurde zuletzt 2012 aktualisiert; er wird daher als ISO/IEC 20000-2:2012 bezeichnet.

Prinzipiell beschreibt ITIL den Weg hin zu einem umfassenden ITSM, während ISO 20000 genutzt wird, um dieses Ziel in Form einer Zertifizierung zu bestätigen. Inhaltlich gibt es zahlreiche Ähnlichkeiten, jedoch sind die Strukturen von ITIL und ISO 20000 nicht vollkommen identisch. Dies liegt unter anderem darin begründet, dass ITIL als Sammlung von Best Practices nicht verbindlich ist, ISO 20000 als Standard dagegen schon. Beispielsweise schreibt ISO 20000 als Richtlinie für Information Security Management die Anwendung des verwandten Standards

ISO/IEC 20002 vor ([ISO12b], bis 2007 mit identischem Inhalt unter dem Namen ISO/IEC 17799 veröffentlicht), während ITIL zwar einen Security Management Guide enthält, aber hier keine verbindlichen Angaben macht. Außerdem enthält ISO 20000 Vorgaben zu Geschäftsbeziehungen (Business Relationship Management) und Lieferanten (Supplier Management), hierzu macht ITIL keine Angaben. Eine Übersicht zu den Details der Unterschiede und Gemeinsamkeiten zwischen ITIL und ISO 20000 geben Dugmore und Taylor in [DT08].

2.1.2 IT-Governance

Die *Enterprise Governance* ist nach der Definition des IT Governance Institute ([GDH+03], Seite 10) die Menge von Verantwortlichkeiten und Verfahren, die von der Unternehmensführung mit dem Ziel durchgeführt werden, eine strategische Richtung für das Unternehmen vorzugeben. Dabei sollen alle Planziele erfüllt werden, mit Risiken muss entsprechend umgegangen werden, und die Unternehmensressourcen sollen verantwortungsbewusst genutzt werden. Der verwandte Begriff der *Corporate Governance*, übersetzt etwa *Grundsätze der Unternehmensführung*, bezeichnet die Sicherstellung von Verhaltensregeln durch das Unternehmen. Dabei werden die Regeln nicht einseitig von der Unternehmensleitung gesetzt, sondern beinhalten beispielsweise auch Compliance-Vorgaben durch Aufsichtsbehörden [KAL14].

Die *IT-Governance* ist derjenige Bestandteil der Enterprise Governance, der sich mit der Abstimmung zwischen der Unternehmensstrategie und der IT-Strategie des Unternehmens beschäftigt. Aus der detaillierten Betrachtung dieser Abstimmung ergeben sich die IT-Prozesse, die mithilfe der IT umgesetzt werden müssen. Beim Entwurf der IT-Prozesse fließen also nicht nur die IT-Strategie sondern auch die operativen Geschäftsprozesse ein. Genauso wie die Geschäftsprozesse zur Unternehmensstrategie passen müssen, so müssen auch die IT-Prozesse zur IT-Strategie passen. Kernaufgabe der IT-Governance ist dabei die Organisation und Steuerung der Abstimmung zwischen diesen beiden Welten. Dabei wird die Abstimmung nicht als einmalige Aktion sondern als dauerhafte Aufgabe verstanden. Sie bezieht sich also nicht nur auf die Entwicklung sondern auch auf den Betrieb von Prozessen.

Für den Begriff der IT-Governance existieren in der Literatur mehrere zum Teil voneinander abweichende Definitionen. Diese unterscheiden sich auch in der Betrachtungsweise der Verantwortlichkeiten unterschiedlicher Interessenvertreter innerhalb des Unternehmens. Nach der Definition von Weill und Ross [WR04] ist der Kern der IT-Governance die Vergabe von Entscheidungsrechten zwischen den IT-Interessenvertretern (also zwischen Vorständen und den IT-Verantwortlichen). In dieser Betrachtung ist das Ziel, dass die sorgfältige Strukturierung und Verteilung von Entscheidungsrechten, zusammen mit einer entsprechenden Transparenz darüber, zu besseren Entscheidungen und letztlich zu einer größeren Effektivität führt.

Die Definition von IT-Governance des IT Governance Institutes [GDH+03] erweitert die Sicht von Weill und Ross und verschiebt die Verantwortlichkeit gänzlich auf die oberste Führungs-

ebene. Hiernach ist die Unternehmensleitung für die IT-Governance verantwortlich, da sie als integraler Bestandteil der Enterprise Governance betrachtet wird. Sie umfasst die Führungs- und Organisationsstrukturen sowie die Prozesse, die sicherstellen, dass die Unternehmens-IT die Strategien und Zielvorgaben der Organisation aufrechterhält und ausbaut. IT-Governance hat hier zwei Ziele: Zum einen muss die IT einen Wert liefern und das Unternehmen voranbringen, zum anderen müssen IT-bezogene Risiken abgeschwächt bzw. reduziert werden. Deshalb müssen Vorstand und Unternehmensleitung sich der Rolle und der Auswirkung der IT auf das Unternehmen bewusst sein und die Randbedingungen festlegen, innerhalb derer die IT-Fachleute operieren, Performance messen und Risiken verstehen.

Die dritte, häufig referenzierte Definition von IT-Governance stammt von De Haes und Van Grembergen [DV04]. Hier verteilt sich die Verantwortlichkeit auf Vorstand, Unternehmensführung und die Verantwortlichen für das IT-Management. Das Ziel ist die Formulierung und Implementierung der IT-Strategie, um damit die Fusion zwischen Geschäftssicht und IT sicherzustellen.

Zusammen mit der stetigen Steigerung der Wichtigkeit der IT für Unternehmen im Laufe der letzten Jahrzehnte sind allerdings auch die Relevanz und die Verfeinerung von IT-Governance-Vorgehensweisen gestiegen. Nach der Auffassung von Johannsen und Goeken gehen die Aufgaben der IT-Governance daher über den Abgleich von Geschäftsaspekten und IT hinaus; hier gehören zu den Aufgaben auch Compliance, Erfolgsmessung, Ressourcenmanagement und Risikomanagement ([JG07], Seite 22). Daher kann IT-Governance nicht alleine der Enterprise Governance zugeordnet werden, sondern deckt auch Fragestellungen der Corporate Governance ab.

Für die IT-Governance hat sich das Rahmenwerk COBIT (ursprünglich eine Abkürzung für *Control Objectives for Information and Related Technology*) durchgesetzt. Aufgrund seiner hohen Bedeutung in der Praxis und als Grundlage dieser Arbeit wird COBIT im Abschnitt 2.2 detaillierter erläutert. Zusätzlich zu COBIT existiert darüber hinaus seit 2008 der Standard ISO/IEC 38500, der 2015 aktualisiert wurde und in dieser Version als ISO/IEC 38500:2015 bekannt ist: „Information technology - Governance of IT for the organization“ [ISO15c]. Im Gegensatz zur Beziehung zwischen ITIL und ISO 20000 ist das Verhältnis zwischen COBIT und ISO 38500 loser gekoppelt: COBIT wird dadurch nicht ersetzt, sondern durch eine verstärkte Sicht auf den Bedarf der IT-Seite ergänzt [Syl11].

2.1.3 Der Zusammenhang zwischen IT Service Management und IT-Governance

Aufgrund ihrer Aufgaben hat die IT-Governance einen viel weiteren zeitlichen Rahmen als das ITSM. Der Fokus des ITSM liegt auf der (richtigen) Bereitstellung von IT-Services als Kapselung benötigter Leistungen sowie ihres operativen Betriebs. IT-Governance dagegen betrachtet

gegenwärtige und zukünftige Anforderungen der Geschäfts- oder Kundenseite an die IT und entwickelt sie entsprechend längerfristig in geeigneter Form weiter.

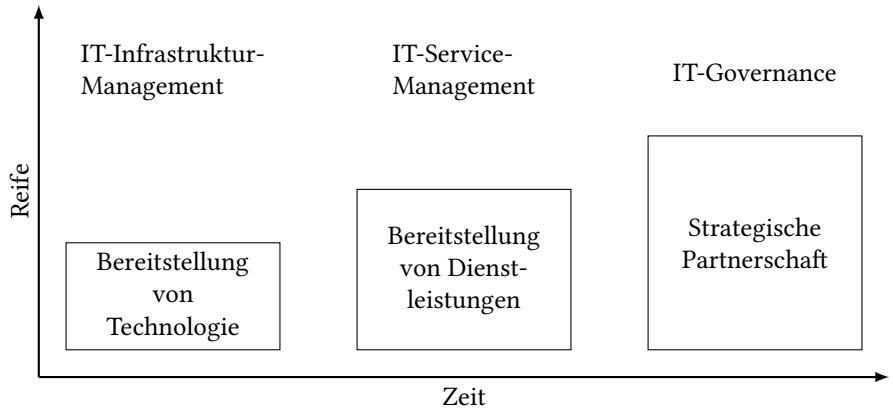

Abbildung 2.1: Evolution der IT-Governance nach Johannsen und Goeken ([JG07], Seite 23)

Abbildung 2.1 zeigt die zeitliche Entwicklung von einer reinen Bereitstellung technischer Lösungen hin zu der heutigen Ausprägung von IT-Governance. Der Schritt zum ITSM stellt sicher, dass die Ziele der Services eng mit den Auftraggebern abgestimmt werden. Mit dem weiteren Wachsen der Bedeutung der IT tritt dann vermehrt die IT-Governance in den Vordergrund: IT und ITSM bleiben wichtig, werden aber weniger als vordergründige Einheit betrachtet, sondern zunehmend als Schlüssel zur Umsetzung von Fragestellungen der Geschäftsseite.

Mit dem Fortschreiten dieses Prozesses wird der strapazierte Begriff des *Business-IT-Alignments* an Bedeutung verlieren. Betz argumentiert, ein spezielles „alignment“ für die IT zu fordern, unterstellt, dass die IT nicht Teil des Geschäfts sei ([Bet07], Seite 17). Demnach sei der Begriff, zumindest in dieser Betrachtungsweise, mängelbehaftet. Für die Lösung der eigentlichen Frage, der systematischen Verbindung der nicht-technischen Belange mit den technischen, existieren zahlreiche Vorgehen. Eine Möglichkeit ist die Betrachtung des Unternehmens auf der hohen Abstraktionsebene der Unternehmensarchitektur und der besseren Integration der IT auf dieser Ebene [ZW06], die tatsächliche Umsetzung der Lösung liegt aber im Aufgabenbereich der IT-Governance.

Während ITSM sich also eher der Frage des *wie* bei der Umsetzung der Serviceerbringung beschäftigt, ist der Kern der IT-Governance die Frage des *was*.

2.2 Aufbau von COBIT 5

Das Rahmenwerk, das sich international für die IT-Governance etabliert hat, ist COBIT. COBIT wird seit 1996 von der ISACA (Information Systems Audit and Control Association) entwickelt,

dem internationalen Verband der IT-Prüfer [isa14], und wurde daher auch zunächst explizit als Werkzeug für Auditoren entworfen. ISACA hat mehr als 110.000 Mitglieder ([GPS13], Seite 2) und sieht sich als „ein führender internationaler Anbieter für Wissensvermittlung, Zertifizierung, Förderung und Bildung in den Bereichen Prüfung und Sicherheit von Informationssystemen (IS), IT-Governance und -Management sowie IT-bezogenen Risiken und Compliance“ ([AdG+12b], Seite 2).

Bis zur Version 4.1 von COBIT, die 2007 veröffentlicht wurde, war der Name eine Abkürzung für *Control Objectives for Information and Related Technology*, übersetzt also *Kontrollziele für Informationen und zugehörige Technologien*. Seit der 2012 veröffentlichten aktuellen Version 5 wird nur noch das Akronym verwendet ([AdG+12b], Glossar), um den Übergang von einem auf Audits fokussierten zu einem umfassenden IT-Governance-Rahmenwerk zu verdeutlichen.

Strukturell gibt es große Unterschiede zwischen den Versionen 4.1 und 5: Bis zur Version 4.1 von COBIT veröffentlichte die ISACA eine Reihe von komplementären Rahmenwerken mit eigenen Publikationen:

- *ValIT* („Enterprise Value: Governance of IT Investments“), mit dem Ziel, systematisch den Wertbeitrag der IT zum Unternehmen zu erfassen;
- *RiskIT*, das sich mit dem Risiko-Management beschäftigt, also mit Geschäftsrisiken, die mit der Nutzung der IT zusammenhängen; und
- *BMIS* („Business Model for Information Security“), das Fragestellungen des Umgangs mit der Informationssicherheit behandelt.

Diese separaten Rahmenwerke wurden mit COBIT verschmolzen und werden deshalb seit COBIT 5 dort inhaltlich komplett abgedeckt. In seiner aktuellen Form besteht COBIT aus einer Menge separater Dokumente, die in Abbildung 2.2 in einer Übersicht dargestellt sind.

Den Kern bildet die Beschreibung des Rahmenwerks [AdG+12a], das ergänzt wird von zwei Kategorien von Dokumenten: Die sogenannten *Enabler-Handbücher*[1] und die *Umsetzungsleitfäden*. Während „Enabling Processes“ [AdG+12c] und „Enabling Information“ [GPS13] die Fragestellungen zur Umsetzung des Prozess-Managements bzw. des Informationsmanagements aus dem Kerndokument ergänzen und vertiefen, sind die Umsetzungsleitfäden an jeweils unterschiedliche Zielgruppen gerichtet, die COBIT aus bestimmten Blickwinkeln betrachten. COBIT Implementation [dHS12] befasst sich mit der Implementierung der Unternehmens-IT auf Basis von COBIT 5; COBIT for Risk [COB13b] legt den Fokus auf den Bereich, der zuvor von RiskIT abgedeckt wurde; COBIT for Information Security [COB12] enhält eine ganzheitliche Betrachtung des Unternehmens zur Umsetzung von Richtlinien zur Informationssicherheit; und COBIT for Assurance [COB13a] beschreibt schließlich, wie der Teil der Corporate Governance angegangen wird, der alle Interessenvertreter über den Status der Compliance gegenüber externen

[1] Dieser Begriff wird auch in der offiziellen deutschen Version von COBIT [AdG+12b] verwendet, *Enabler* wird nicht übersetzt.

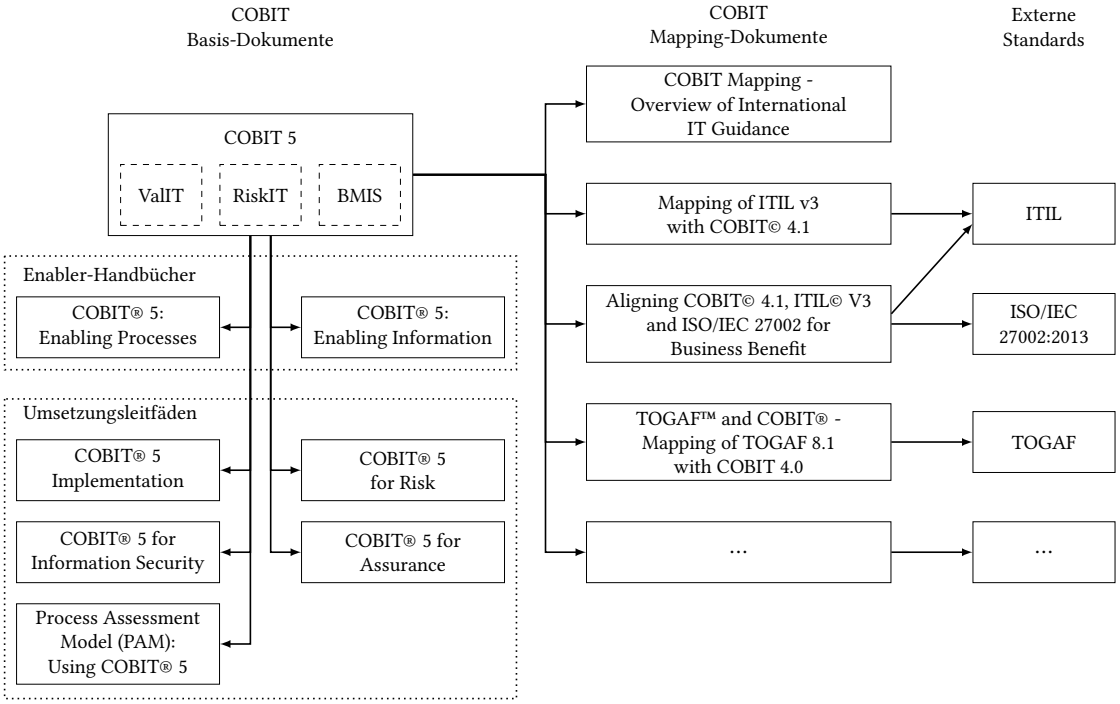

Abbildung 2.2: Übersicht: Bestandteile der COBIT-Spezifikation

Vorgaben auf dem Laufenden hält. Process Assessment Model (PAM) [BL13] gibt darüber hinaus eine Hilfestellung zur Bewertung von IT-Prozessen nach den Kriterien von COBIT.

Zusätzlich zu den Basis-Dokumenten wurde eine Reihe von Mapping-Dokumenten herausgegeben, die den Zusammenhang zwischen COBIT und existierenden externen Standards und Rahmenwerken beleuchten sowie Hinweise dazu geben, wie jeweils beide gemeinsam sinnvoll eingesetzt werden können. Da die Mapping-Dokumente nicht zu den Kern-Veröffentlichungen gehören, sind nicht alle von ihnen auf die jeweils aktuellste Version von COBIT angepasst worden. Einen generellen Überblick zu COBIT und Zusammenwirken mit externen Rahmenwerken gibt der „Overview of International IT Guidance“ [Hes06]. In separaten Dokumenten werden die Zusammenhänge zwischen COBIT und ITIL [HH08b], COBIT, ITIL und ISO/IEC 27002 [HH08a] sowie zwischen COBIT und dem Rahmenwerk der Open Group zur Unternehmensarchitektur, TOGAF („The Open Group Architecture Framework“) [Mac07] beleuchtet. Es existieren weitere Dokumente dieser Kategorie, auf die hier nicht weiter eingegangen wird.

Als übergeordnete Elemente für IT-Governance definiert COBIT auf seiner obersten Abstraktionsebene zunächst fünf Prinzipien und sieben sogenannte *Enabler*. Die Prinzipien bilden die Grundlage für den Aufbau des Informationsmodells über die Zusammenhänge, die für die IT-Governance nötig sind, wie auch für die initiale oder fortführende Umsetzung von COBIT im

Unternehmen. Sie lauten:

1. Erfüllung der Anforderungen der Anspruchsgruppen[2],
2. Abdeckung des gesamten Unternehmens,
3. Anwendung eines einheitlichen, integrierten Rahmenwerks,
4. Ermöglichung eines ganzheitlichen Ansatzes,
5. Unterscheidung zwischen Governance und Management.

Unter Enabler versteht COBIT alles, das zur Erreichung eines Unternehmensziels beiträgt und legt hierfür Kategorien fest:

1. Prinzipien, Richtlinien und Rahmenwerke,
2. Prozesse,
3. Organisationsstrukturen,
4. Kultur, Ethik und Verhalten,
5. Information,
6. Services, Infrastruktur und Anwendungen,
7. Mitarbeiter, Fähigkeiten und Kompetenzen.

Aus den Vorgehensbeschreibungen zur schrittweisen Umsetzung der Prinzipien ergibt sich unter Einbeziehung der Enabler ein Informationsmodell, das die betrachteten Entitäten und ihre Beziehungen untereinander beschreibt. Dieses Modell ist in COBIT weitestgehend implizit beschrieben: Es werden zwar die wesentlichen Elemente genannt und im Text referenziert, und die sogenannte COBIT-Zielkaskade zeigt auch den Zusammenhang zwischen Anforderungen von Anspruchsgruppen, Unternehmenszielen und IT-bezogenen Zielen auf ([AdG$^+$12b], Seite 20), aber viele Abhängigkeiten sind nur in Form von Zuordnungstabellen in Anhängen definiert (z.B. die Zuordnung von IT-bezogenen Zielen zu IT-bezogenen Prozessen, [AdG$^+$12b], Anhang C). Abbildung 2.3 zeigt den Aufbau des Informationsmodells im Überblick.

Ausgangspunkt aller Betrachtungen sind die Treiber der Anspruchsgruppen, die in konkreten Anforderungen münden. Diese werden dann in Form einer Sammlung von expliziten Geschäftszielen formuliert. Geschäftsziele haben üblicherweise einen hohen Abstraktionsgrad, z.B. die Erreichung einer bestimmten Marktdurchdringung. Aus den Geschäftszielen ergeben sich die Anforderungen an die Unterstützung durch die IT, hieraus werden die IT-bezogenen

[2]Diese Übersetzung des englischen Begriffs *stakeholder* wird hier aus der offiziellen deutschen Version von COBIT [AdG$^+$12b] übernommen.

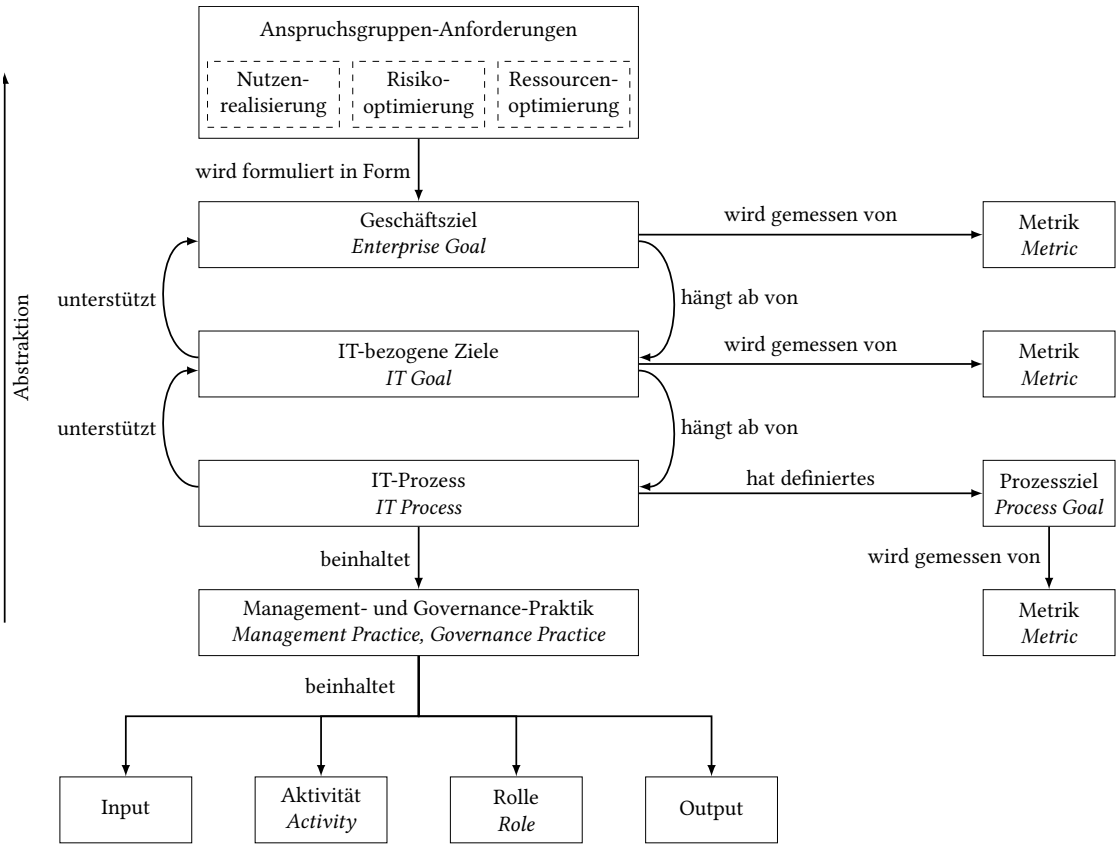

Abbildung 2.3: Konzeptueller Aufbau der COBIT Abstraktionsschichten

Ziele definiert. Dieser Schritt ist elementarer Bestandteil der IT-Governance. Das Informationsmodell ist in COBIT weiter beschrieben durch IT-Prozesse, die die Umsetzung der IT-bezogenen Ziele zur Aufgabe haben.

COBIT 5 definiert 37 konkrete Prozesse, die in fünf Kategorien eingeteilt sind. Die fünf Prozesskategorien sind zusammen mit ihrer deutschen Bezeichnung in Tabelle 2.1 gelistet. Aus den englischen Bezeichnungen der Kategorien ergeben sich Kürzel, die sich auch in den Namen der ihnen zugehörigen Prozesse wiederfinden. Zusätzlich sind die Kategorien unterteilt in die zwei Kernbereiche „Governance“ und „Management“: Governance enthält die Prozesse aus der Kategorie „Evaluate, Direct and Monitor“, die Prozesse der anderen vier Kategorien werden „Management“ zugeordnet.

Das Enabler-Handbuch „Enabling Processes“ definiert die Prozesskategorien sowie die Prozesse im Detail und legt für jeden Prozess fest, zu welcher Prozesskategorie er gehört, über welche Managementpraktiken er verfügt, was dessen Ein- und Ausgabeinformationen sind (Inputs und

Kürzel	Prozesskategorie	Deutsche Bezeichnung
EDM	Evaluate, Direct and Monitor	Evaluieren, Richtung vorgeben und Überwachen
APO	Align, Plan and Organise	Anpassen, Planen und Organisieren
BAI	Build, Acquire and Implement	Aufbauen, Beschaffen und Implementieren
DSS	Deliver, Service and Support	Bereitstellen, Betreiben und Unterstützen
MEA	Monitor, Evaluate and Assess	Überwachen, Evaluieren und Beurteilen

Tabelle 2.1: COBIT Prozesskategorien

Outputs), welche konkreten Aktivitäten zu seiner Erfüllung notwendig sind, sowie welche Rollen im Unternehmen dafür in welcher Weise zuständig sind. Eine Übersicht über die Prozesse und ihre Zuordnung zu den Prozesskategorien ist in Anhang B zusammengefasst.

Für Geschäftsziele, IT-bezogene Ziele und Prozessziele ist darüber hinaus jeweils eine Menge von Metriken definiert, die den jeweiligen Erreichungsgrad messbar machen sollen. Die Metriken sind in Form von Fließtext formuliert und bewegen sich dabei zwischen sehr abstrakten, schwer messbaren, und konkreten, messbaren Größen. Beispielsweise definiert COBIT für das IT-Ziel 17 „Knowledge, expertise and initiatives for business innovation“ die nur schwer messbare Metrik „Level of business executive awareness and understanding of IT innovation possibilities“. Für das Enterprise Goal 6 „Customer-oriented service culture“ dagegen existiert die Metrik „Number of customer service disruptions due to IT service-related incidents (reliability)“, die sich durchaus messen lässt.

Der letzte wesentliche Teil von COBIT 5 ist das sogenannte *Prozessbefähigungsmodell* (Process Capability Model). Das Ziel ist die Messung des aktuellen Reifegrades eines jeden Prozesses und der Abgleich des Ist-Zustandes mit dem Soll-Zustand. Während in COBIT 4 hierfür ein Reifegradmodell (Maturity Model) eingesetzt wurde, basiert das Prozessbefähigungsmodell auf dem Standard ISO/IEC 15504 (für den 2015 nach einer Revision die neue Version ISO/IEC 33001:2015 veröffentlicht wurde [ISO15b]). Prozesse werden in eine von sechs Befähigungsstufen eingeteilt, indem zu den Stufen gehörende Prozessattribute beurteilt werden. COBIT definiert neun solcher Prozessattribute. Abbildung 2.4 zeigt den Zusammenhang zwischen den Befähigungsstufen und den Prozessattributen. Die wesentlichen Unterschiede zwischen COBIT 4 und 5 sind hier zum einen, dass nun eine klare Trennung vorgeschrieben ist zwischen dem Inhalt eines Prozesses und seiner Messung oder Überwachung – diese Aspekte dürfen also nicht Teil der Prozessbeschreibung sein; zum anderen, dass die jeweils nächste Befähigungsstufe nur erreicht werden kann, wenn die darunterliegende Stufe vollständig erreicht wurde. Aus diesem Grund ist das Erreichen der Befähigungsstufe 1 „bereits eine wichtige Errungenschaft für ein Unternehmen“ ([AdG+12b], Seite 45).

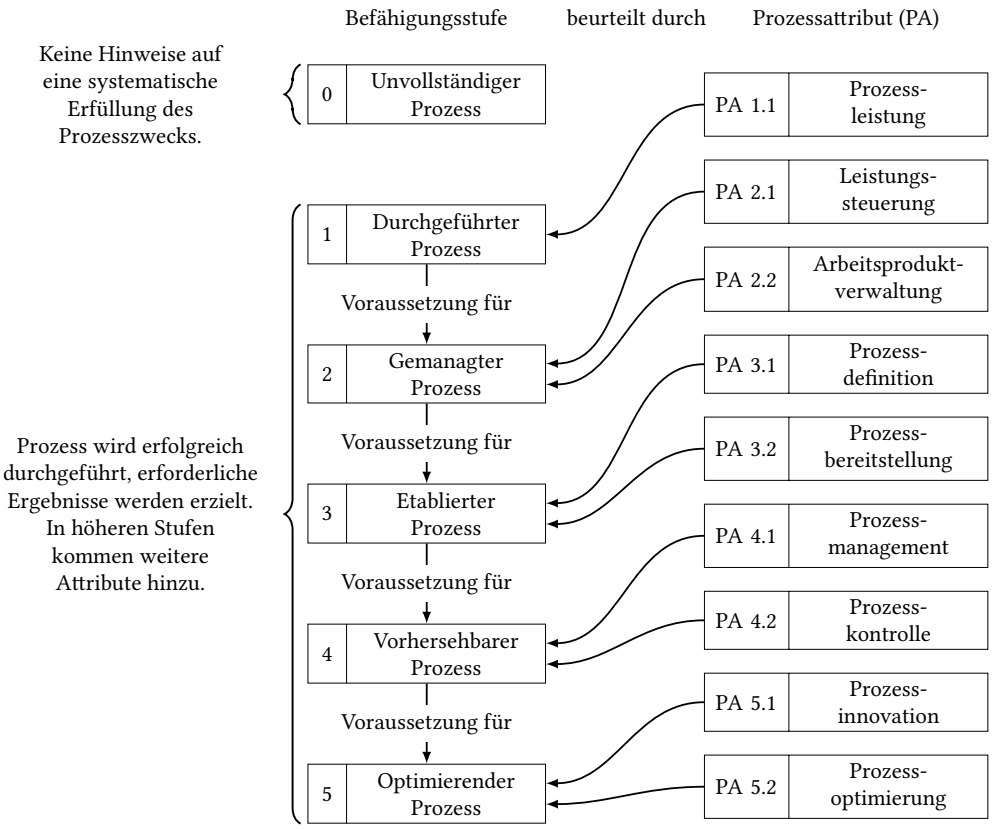

Abbildung 2.4: Befähigungsstufen und Prozessattribute des Prozessbefähigungsmodells

3 Wissensmodelle, Ontologien und Semantische Technologien

3.1 Historische Entwicklung

3.1.1 Ontologien

Der Begriff der Ontologie entstammt ursprünglich der Philosophie und bezeichnet eine Disziplin, die sich mit allem Seienden befasst. In diesem Kontext wird der Begriff ausschliesslich im Singular verwendet. In der Informatik wurde er zunächst übernommen, dann aber nach zunehmendem Einsatz in den Gebieten der Wissensrepräsentation und der Künstlichen Intelligenz mit einer von der ursprünglichen Bedeutung abweichenden versehen. Die ersten formalen Definitionen von Ontologie in der Informatik stammen aus den frühen 1990er Jahren. Hierbei wird bereits von einer oder mehreren Ontologien gesprochen. Nach Neches et al. ([NFF+91], Seite 40) beschreibt eine Ontologie die grundlegenden Begriffe und Beziehungen des Vokabulars eines bestimmten Themenfeldes sowie Regeln, um die Begriffe und Beziehungen zur Erweiterung des Vokabulars einzusetzen. Die häufig zitierte Definition von Tom Gruber [Gru93] beschreibt eine Ontologie als *eine explizite Spezifizierung einer Konzeptualisierung*. Eine Konzeptualisierung ist dabei der Prozess der Entwicklung und Klarstellung von Konzepten. Zwei Verfeinerungen führen darauf aufbauend zum heutigen Verständnis von Ontologien in der Informatik. Borst erweitert Grubers Definition wie folgt: Ontologien sind definiert als eine *formale* Spezifizierung einer *gemeinsam genutzten* Konzeptualisierung ([Bor97], Seite 12).

Letztlich erläutern Studer et al. ([SRF98], Seite 25) die Bestandteile dieser Definition: eine Konzeptualisierung bezieht sich auf das abstrakte Modell eines Phänomens der [echten] Welt, indem die relevanten Konzepte dieses Phänomens eindeutig identifiziert werden. Explizit bedeutet, dass sowohl die Konzepte als auch ihre Randbedingungen und Einschränkungen explizit definiert sind. Formal ist hier gleichbedeutend mit maschinenlesbar, die Ontologie darf also nicht nur natürlichsprachlich beschrieben sein. Eine Ontologie kann dann gemeinsam genutzt werden, wenn das in ihr beschriebene Wissen in Übereinstimmung ihrer Nutzer festgelegt wurde, es soll also nicht nur privat von einem Individuum genutzt werden können. Obwohl diese Definition heute weitestgehender Konsens ist, sollte angemerkt werden, dass sie weder festlegt, wie das gemeinsam genutzte Wissen erfasst werden soll, noch in welcher Form eine

Formalisierung zu erfolgen hat. Die Antworten auf diese Fragen sind in hohem Maße abhängig vom Ziel, das durch den Einsatz einer oder mehreren Ontologien erreicht werden soll. Die Verwendung des Ontologiebegriffs in dieser Arbeit folgt der Definition von Borst sowie den Konkretisierungen von Studer et al..

Die Definition erlaubt zwei grundlegende Sichten auf Ontologien, die abhängig von der konkreten Auslegung des Begriffes der Konzeptualisierung sind. Nach einer Sicht beinhaltet eine Ontologie demnach tatsächlich nur abstrakte Konzepte und Relationen (beispielsweise *Person* und *hatAlter*), sie beschreibt also ein Informationsschema. Nach der anderen Sicht beinhaltet eine Ontologie in erster Linie konkrete Wissensfragmente (beispielsweise *John ist eine Person*), sie übernimmt also die Rolle von gesammelten Wissen (engl. *body of knowledge*). Diese Sichtweise wird häufig in Projekten eingenommen, die den Aufbau einer umfassenden Sammlung von formalisiertem Allgemeinwissen zum Ziel haben. Aktuelle Technologien zur Repräsentation von Ontologien unterstützen in der Regel zwar die Umsetzung beider Sichten, der Unterschied muss jedoch zum richtigen Verständnis der Literatur bekannt sein, da beide Sichten implizit verwendet werden.

3.1.2 Formale Repräsentation

Wie in Abschnitt 3.1.1 beschrieben, existiert zwar ein prinzipielles gemeinsames Verständnis darüber, was eine Ontologie ist und was sie beinhaltet, es wurde jedoch eine Vielzahl von Formalismen und Sprachen zur Beschreibung von Ontologien entwickelt. Diese unterscheiden sich in grundlegenden Konzepten, ihrer Ausdrucksmächtigkeit und damit verbunden auch in ihrer Komplexität. In diesem Abschnitt soll eine Übersicht über die wichtigsten Formalismen gegeben werden. Der Einsatz des geeigneten Formalismus trägt wesentlich zum erfolgreichen Einsatz von Ontologien bei.

In *Frame-basierten Modellen* wird ein Informationsschema durch *Frames* (Rahmen) und *Slots* (Einschübe) modelliert. Ein Frame ist eine benannte Datenstruktur, die ein Konzept in der modellierten Wissensbasis darstellt; ein Slot ist eine binäre Relation, die einen Frame mit einem Wert verbindet. Eine grundlegende Eigenschaft der Frame-basierten Logik ist die Geschlossenheit: Über Dinge, die nicht explizit Teil des Modells sind, wird implizit die Aussage gemacht, dass sie nicht existieren. Es können notwendige Bedingungen für Frames definiert werden, aber es können keine hinreichenden Bedingungen definiert werden. Das bedeutet, dass automatisch die Konsistenz eines Frame-Modells überprüft werden kann, es ist aber nicht möglich, dass automatisch Unterklassen-Beziehungen inferiert werden können [WNR$^+$06]. Der bekannteste Vertreter von Frame-basierten Formalismen ist Frame Logic (auch F-Logic genannt, [KLW11]). Generell ist F-Logic unentscheidbar, viele Probleme sind allerdings entscheidbar und haben Lösungen mit polynomieller Komplexität ([Kif05], Seite 11).

Bei der Modellierung eines Informationsmodells durch *Semantische Netzwerke* (auch *Konzept-Netzwerke* genannt) wird ein Graph erstellt: Knoten repräsentieren Konzepte, Kanten repräsentieren Relationen. Da der Formalismus ursprünglich entwickelt wurde, um Konstrukte unterschiedlicher natürlicher Sprachen aufeinander abbilden zu können, beschränkt sich die Semantik in Semantischen Netzwerken auf entsprechende Relationen: es können Synonyme (Gleichheit von Konzepten) und Antonyme (Ungleichheit), sowie Meronyme (Teil-von-Beziehung) und Holonyme (Hat-Teil-Beziehung) ausgedrückt werden. Eine bekannte Anwendung von Semantischen Netzwerken ist WordNet [Fel06], eine lexikalische Datenbank der englischen Sprache. Mit ISO/IEC 13250:2015 [ISO15a] existiert eine standardisierte Version von Topic Maps, einer festgelegten Syntax für Semantische Netzwerke.

Das *Knowledge Interchange Format* (KIF, [GFB+92]) ist eine Beschreibungssprache für Wissen, die auf Prädikatenlogik erster Stufe (*First Order Logic*, kurz FOL) basiert. KIF wurde explizit mit einer hohen Ausdrucksmächtigkeit entworfen, um als Austauschformat zwischen unterschiedlichen Wissensrepräsentationen eingesetzt werden zu können und ist daher nicht entscheidbar. Als Nachfolger und Erweiterung von KIF wurde das Rahmenwerk *Common Logic* (CL) entwickelt, das ebenfalls auf FOL basiert. CL wurde 2007 im Standard ISO/IEC 24707:2007 [ISO07] formalisiert, der außer der Semantik auch drei sogenannte CL-Dialekte beschreibt, unterschiedliche syntaktische Ausprägungen: Das Common Logic Interchange Format (CLIF, das auf der ursprünglichen KIF-Syntax basiert), das Conceptual Graph Interchange Format sowie die XML-basierte Syntax XCL.

Im Gegensatz zu KIF und Common Logic wurden *Beschreibungslogiken* (Description Logics, abgekürzt DL) als Format zur Wissensrepräsentation mit dem Ziel entworfen, entscheidbar zu bleiben. Beschreibungslogiken sind eine Gruppe von Logiken, die ursprünglich entwickelt wurden, um die zunächst nur informell beschriebenen Semantischen Netzwerke und Frame-basierten Modelle zu formalisieren. Sie sind immer Fragmente von FOL, lassen aber unter Einschränkung auf bestimmte Komplexitätsklassen möglichst ausdrucksstarke Sprachelemente zu. Aufgrund ihrer Bedeutung als theoretische Grundlage für die Ontologie-Beschreibungssprache *Web Ontology Language* (OWL, [HPMW07]) werden Beschreibungslogiken in Abschnitt 3.4.1 detaillierter beschrieben.

3.1.3 Semantic Web

Das *Semantic Web* wurde Anfang der 2000er Jahre als Initiative des W3C (*World Wide Web Consortium*) ins Leben gerufen, dem Gremium zur Standardisierung von Techniken im World Wide Web. Die ursprüngliche Idee des Semantic Web umfasste die langfristige Weiterentwicklung des Web in einer Weise, in der Inhalte nicht nur wie bis dahin lesbar für Menschen, sondern auch besser algorithmisch verarbeitbar werden sollten. Dazu sollten die den Webseiten zugrunde liegenden Beschreibungen, die in erster Linie zu deren Formatierung eingesetzt wurden, durch Auszeichnungen erweitert werden, die die dargestellten Inhalte auch semantisch

beschreiben. Dabei müssen mehrere Probleme gelöst werden, beispielsweise wie solche Annotationen in bestehende Formate eingebracht werden, ohne die Kompatibilität einzuschränken. Wichtiger ist die Frage, wie die Semantik beschrieben wird: Wie wird für ein Vorkommen des Begriffs „Venus“ festgelegt, dass er sich auf einen Planeten, eine Göttin, eine Statue oder ein anderes Konzept bezieht, und wie genau werden die Beziehungen zwischen Konzepten ausgedrückt?

Um diese Fragen zu beantworten, wurde für das Semantic Web eine Reihe von Technologien entwickelt, die, soweit möglich und sinnvoll, auf bestehenden Grundlagen aufbauen. Abbildung 3.1 zeigt den sogenannten „Schichtenkuchen“ des Semantic Web, also die einzelnen Technologien und wie sie aufeinander aufbauen.

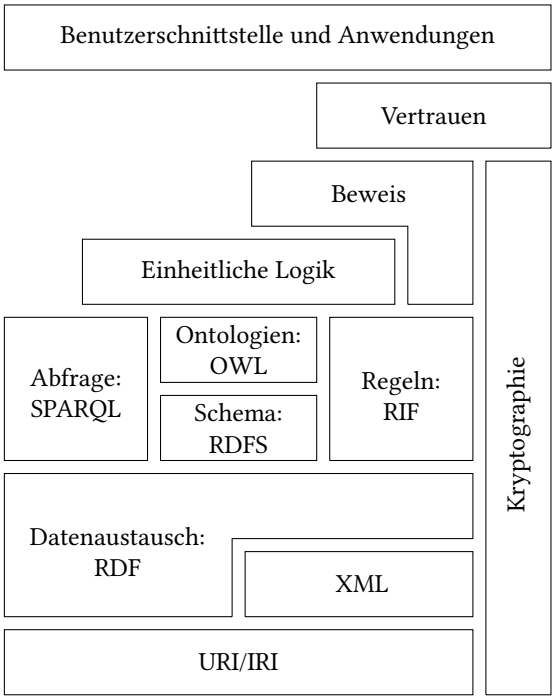

Abbildung 3.1: Der Semantic Web „Schichtenkuchen“ nach [w3c13a]

Die Grundlage zur Referenzierung von Elementen bilden die Standards URI und IRI, die in Abschnitt 3.2 detaillierter vorgestellt werden. Datenaustausch geschieht durch das Beschreibungsformat RDF und die verwandte Schema-Beschreibung RDFS, die beide in Abschnitt 3.3 beschrieben werden. Um die beschriebenen Informationen abfragbar zu machen, wurde die Abfragesprache SPARQL entwickelt, die in Abschnitt 3.5 vorgestellt wird. Für die Festlegung der Semantik wurde ein neues Beschreibungsformat festgelegt, das im Kern auf Beschreibungslogiken basiert und für das unter anderem RDF zur Serialisierung eingesetzt wird, die Sprache OWL. Diese wird im Detail in Abschnitt 3.4 beschrieben. Der vertikale Block der Kryptographie

– in der Praxis durch SSL/TLS-verschlüsselte Verbindungen im HTTPS-Protokoll realisiert – bildet zusammen mit den logischen Beschreibungen im OWL-Format die Grundlage für Vertrauen und darauf aufbauend, für Benutzer- und Programmschnittstellen.

Eng verwandt mit der der Idee des Semantic Web ist das Konzept *Linked Open Data*, das darauf abzielt, Datensätze und Schnittstellen zur Abfrage von Daten, die von allgemeinem Interesse sind (z.B. von Behörden veröffentlichte Statistiken und Informationen), mittels der Semantic-Web-Technologien zu realisieren. Das Ziel hierbei ist also nicht ein Web, das durch Daten angereichert wird, sondern Sammlungen von Daten, die analog zur Vernetzung des WWW untereinander verknüpft sind. So bietet beispielsweise die Stadt Berlin unter [Ber16] mehr als 1000 frei verfügbare Datensätze an.

3.2 Bezeichner im Semantic Web

Ressourcen im Web und im Semantic Web werden durch Zeichenketten beschrieben, die bestimmten Formaten entsprechen müssen. Für eine korrekte Zuordnung sollen daher hier die wesentlichen Begriffe beschrieben werden. Abbildung 3.2 zeigt den Zusammenhang zwischen den fünf wesentlichen Notationen.

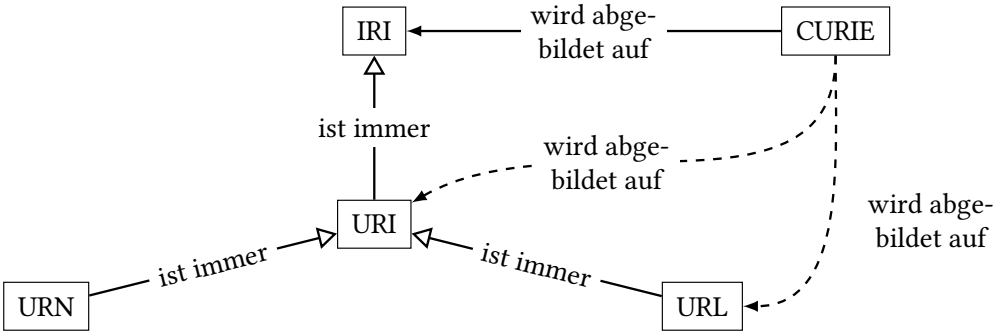

Abbildung 3.2: Zusammenhang zwischen IRI, URI, URL, URN und CURIE

Eine URL (*Uniform Resource Locator*, [BMM94b]) gibt über ein Protokoll, einen Host und zusätzliche Informationen den Namen, Ort und Zugriffsmöglichkeit auf eine Ressource im Web an. Wird der syntaktische Aufbau einer URL verallgemeinert, um Ressourcen nur noch eindeutig und strukturiert zu benennen, aber nicht mehr notwendigerweise zuzugreifen, wird von einer URI (*Uniform Resource Identifier*, [BFM05]) gesprochen. Hierbei wird nicht mehr vom Protokoll, sondern vom Schema gesprochen. Neben URL ist eine weitere Ausprägung einer URI die URN (*Uniform Resource Name*, [Moa97]). Hierbei bezeichnet das Schema lediglich einen klar definierten Namensraum, beispielsweise `isbn`, der Rest der URN wird dann als opak betrachtet und muss nur im Kontext des Namensraums interpretierbar sein. Eine URN kann also nicht

nach einem generischen Verfahren wie bei URLs aufgelöst werden. Während also jede URL und jede URN auch eine URI ist, gilt die umgekehrte Richtung nicht generell. 2005 wurde eine weitere Verallgemeinerung von URIs definiert, indem unter anderem für die alphanumerischen Teile der Zeichenkette die Begrenzung von US-ASCII-Kodierung aufgehoben wurde und die meisten Unicode-Zeichen zugelassen wurden. Dies wird als IRI (*Internationalized Resource Identifier*, [DS05]) bezeichnet. IRIs, die strukturell URLs entsprechen, aber selbst keine gültigen URLs sind (weil sie nicht-US-ASCII-Zeichen enthalten) können durch die in den entsprechenden RFCs angegebenen Übersetzungsvorschriften in gültige URLs umgewandelt werden.

In Sprachen und Datenformaten, in denen IRIs oder eine Ausprägung als Bezeichner eingesetzt werden, existiert häufig eine Möglichkeit, Präfixe häufig genutzter IRIs einmal zu definieren und dann in einer abkürzenden Schreibweise zu verwenden. Diese aus einem innerhalb der Sprache als Ersatz für einen Teil einer IRI definierten Präfix und einem lokalen Teil bestehende Kurzform wird CURIE genannt (ursprünglich eine Abkürzung für *Compact URI*, aber ebenso für IRIs eingesetzt [BM10]). CURIEs sind selbst keine gültigen IRIs, können aber zusammen mit den gegebenen Möglichkeiten der Sprache oder des Formats, in dem die CURIE eingesetzt wird, auf entsprechende gültige IRIs abgebildet werden.

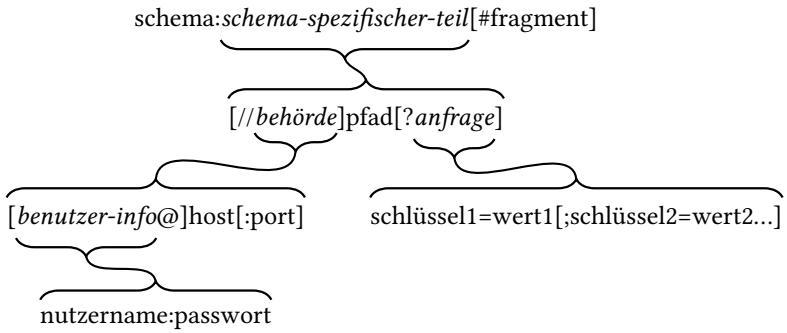

Abbildung 3.3: Aufbau einer IRI

Abbildung 3.3 zeigt den Aufbau einer IRI in vereinfachter Form. In der Abbildung stehen Elemente in eckigen Klammern für optionale Teile und die Auslassung (…) für beliebige entsprechende Wiederholungen; alle anderen nicht alphanumerischen Zeichen stehen für sich selbst. Nicht alle Formen von IRIs nutzen alle Bestandteile, beispielsweise ist die Angabe von Fragment, Behörde oder Anfrage bei URNs ungebräuchlich. Nicht in der Abbildung dargestellt sind die möglichen Werte, die für einen Host gültig sind: Hier können DNS-Namen inklusive Subdomains, IPv4-Adressen in Dezimalpunktschreibweise oder IPv6-Adressen in eckigen Klammern stehen.

3.3 RDF und RDFS

3.3.1 Resource Description Framework (RDF)

Das *Resource Description Framework* (RDF, [w3c14b]) ist ein formales Format zur Beschreibung strukturierter Informationen. Der Kern von RDF ist ein Datenmodell, mit dem gerichtete Graphen beschrieben werden. Im Gegensatz zur Baumstruktur eines XML-Dokuments kann ein RDF-Graph daher beispielsweise auch Zyklen enthalten. Neben dem Datenmodell spezifiziert RDF auch eine Reihe von gültigen Serialisierungen eines RDF-Graphen, sogenannte konkrete Syntaxen.

Ein RDF-Graph wird durch eine Menge von Aussagen (*Statements*, auch *Tripel* genannt) beschrieben, die jeweils aus einen Subjekt, einem Prädikat und einem Objekt bestehen. Subjekt und Objekt entsprechen Knoten im Graph, das Prädikat entspricht einer Kante. Es werden drei Arten von Knoten unterschieden: Benannte Knoten, unbenannte Knoten (auch anonyme Knoten, *Blank Nodes* oder *BNodes* genannt) und Literale.

Ein zentrales Merkmal von RDF ist, dass als eindeutige Bezeichner für benannte Knoten *IRIs* eingesetzt werden (vor RDF Version 1.1 sah die Spezifikation hierfür URIs vor, die Verwendung erfolgt analog). Daraus folgen wichtige Eigenschaften: Erstens können über den Knoten und das Konzept, das er repräsentiert, sowie über den Graph, dessen Teil er ist, transparent weitere Informationen bezogen werden. Die Voraussetzung hierfür ist lediglich, dass sein Bezeichner nicht nur eine IRI, sondern auch eine URL ist. Mittels Mechanismen, die bereits Teil bestehender Web-Protokolle wie HTTP sind, kann die URL aufgelöst werden und, je nachdem, wie die Anfrage gestellt wird, menschenlesbare Dokumentation zum RDF-Dokument oder weitere maschinenlesbare Aussagen oder Verweise auf externe Dokumente liefern. Dies wird durch Setzen entsprechender HTTP-Header bei der Anfrage gesteuert und als *Content Negotiation* bezeichnet. Zweitens stellt der Einsatz von IRIs als Bezeichner implizit die Verantwortlichkeit für die Einzelteile des globalen Namensraums sicher. Da der Name eines Hosts, bzw. der Domainname, einer Partei oder einem Individuum explizit zugeordnet ist, kann es nicht zu Namenskonflikten der darauf folgenden IRI-Bestandteile kommen, insbesondere des Pfads. Darüber hinaus kann der IRI-Pfad eine hierarchische Struktur abbilden und unterstützt damit analog zur Paket- oder Modulstruktur in gängigen Programmiersprachen eine sinnvolle Aufteilung. Drittens ist die konfliktfreie Identifizierung von Knoten eines RDF-Graphen dann wichtig, wenn der RDF-Graph mit weiteren Graphen zusammengeführt werden soll.

Anonyme Knoten besitzen keinen global eindeutigen Bezeichner in Form einer IRI, sie können jedoch einen Bezeichner besitzen, dessen Gültigkeitsbereich sich auf das Dokument beschränkt, in dem der Knoten definiert ist. Wie diese dokumentenlokalen Bezeichner aufgebaut sind, ist abhängig von der eingesetzten konkreten Syntax. Diese Bezeichner haben lediglich das

Ziel, den Knoten innerhalb des Dokuments referenzierbar zu machen; während der maschinellen Verarbeitung eines RDF-Dokuments werden hierfür von der verarbeitenden Software in der Regel Bezeichner automatisch generiert.

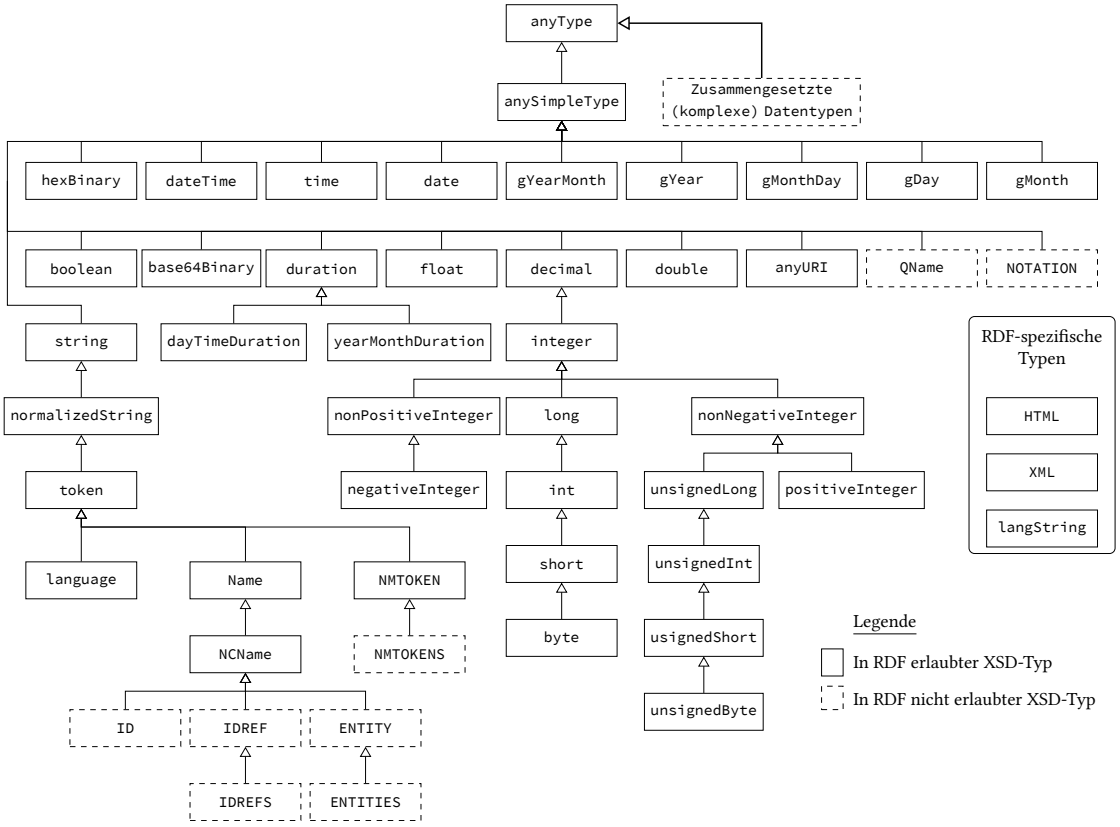

Abbildung 3.4: Datentypen in RDF und Zusammenhang mit XML Schema

Die dritte Art von Knoten in RDF sind Literale, die genutzt werden, um skalare Werte zu repräsentieren. Hierbei handelt es sich um Zeichenketten, die entweder mit einem Datentyp oder einem *Language Tag* ausgezeichnet sind[1]. Ein Language Tag ist dabei ein Kürzel für die Sprache der ausgezeichneten Zeichenkette, das den entsprechenden ISO-Standards und RFCs folgt [PD09]. Die gültigen Datentypen sind im Wesentlichen eine Untermenge der Datentypen, die in der Spezifikation der XML Schema Definition (XSD) festgelegt sind [BM04]. Die Vererbungshierarchie der Datentypen in XSD und welche davon für RDF gültig sind, zeigt Abbildung 3.4.

Generell entsprechen die Wertebereiche des jeweiligen Datentyps den aus geläufigen Program-

[1]Die Datentyp-Annotation kann auch gänzlich weggelassen werden, seit RDF 1.1 ist der Typ des Literals in diesem Fall implizit `string`.

miersprachen bekannten. Die in der Abbildung 3.4 ohne durchgezogene Linie gezeichneten Kästen bezeichnen XSD-Datentypen, die in RDF nicht eingesetzt werden dürfen. Diese Typen beziehen sich meist auf XML-spezifische Mechanismen und sind daher für RDF ungeeignet. Der Typ `QName` entspricht konzeptuell den in Abschnitt 3.2 beschriebenen CURIEs, hat aber bedingt durch die XML-Serialisierung stärkere Einschränkungen in Form eines kleineren lexikalischen Wertebereichs. Zusätzlich zu den XSD-Datentypen definiert die RDF-Spezifikation zwei weitere Typen: `HTML` und `XML`, die als Typen für Literale verwendet werden, die Dokumente oder Fragmente in der jeweiligen Markup-Sprache beinhalten können.

Das Subjekt einer RDF-Aussage darf nur ein benannter oder ein anonymer Knoten sein, das Prädikat darf nur ein benannter Knoten sein. Als Objekt sind alle drei Typen von Knoten erlaubt – benannte Knoten, anonyme Knoten und Literale.

Für die Serialisierung eines RDF-Graphen sind verschiedene Formate definiert, die sich in Syntax und technischer Grundlage unterscheiden. Abbildung 3.5 zeigt die RDF-Syntaxen die in RDF 1.0 beziehungsweise RDF 1.1 definiert sind. In RDF 1.0 waren nur die XML-basierte

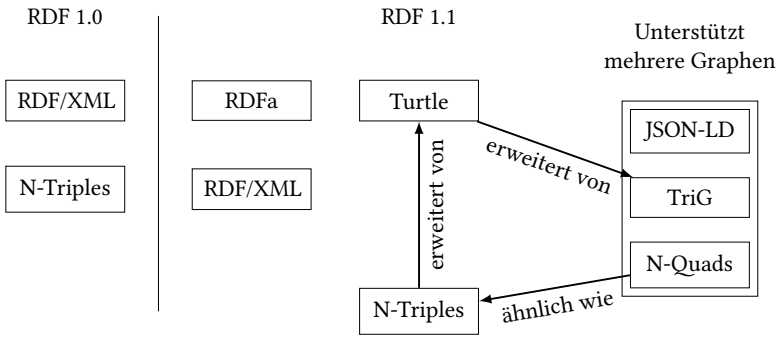

Abbildung 3.5: Übersicht der RDF-Syntaxen

RDF/XML-Syntax sowie die sogenannte N-Triples-Syntax als verbindlich festgelegt. In der N-Triples-Syntax werden die einzelnen RDF-Aussagen zeilenweise ausgegeben, dabei werden IRIs ausgeschrieben und in spitze Klammern gesetzt, für unbenannte Knoten werden Bezeichner generiert. RDF 1.1 führte eine Reihe neuer Syntaxen verbindlich ein. RDFa (für *RDF in Attributes* ermöglicht die Einbettung von RDF-Aussagen in HTML- und XHTML-Dokumenten, um diese mit besser maschinenlesbaren Varianten ihres bestehenden Inhaltes anzureichern. JSON-LD (*JavaScript Object Notation for Linked Data*) erweitert die bereits häufig eingesetzte JSON-Syntax um Konzepte aus RDF, um die ausgedrückten Daten im Sinne des Semantic Web besser miteinander vernetzen zu können.

Das Fehlen einer gut manuell bearbeitbaren RDF-Syntax wurde an RDF 1.0 häufig kritisiert, weswegen die auch vorher schon entwickelte, aber nicht standardisierte Turtle-Syntax in RDF 1.1 offiziell aufgenommen wurde. Der Name geht hervor aus der ursprünglichen Bezeichnung

Terse Triple Language, oder *TTL*. Wegen seiner leichteren Lesbarkeit im Vergleich zu anderen Syntaxen wird Turtle im Rest des Kapitels für RDF-Beispiele, und in der gesamten Arbeit für RDF-Listings verwendet. Der Beispielgraph aus der RDF-Spezifikation [w3c14a] kann in Turtle so beschrieben werden, wie in Abbildung 3.6 gezeigt.

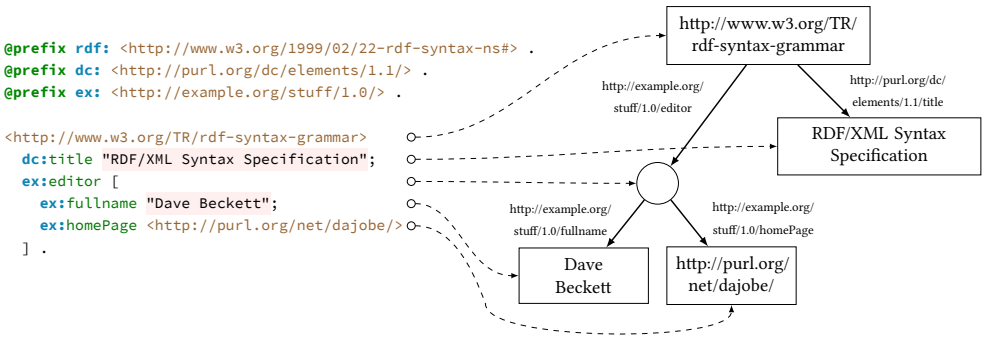

Abbildung 3.6: Beispiel: Turtle-Syntax für RDF

Grundsätzlich werden Aussagen zeilenweise geschrieben und jeweils mit einem Punkt beendet, allerdings erlaubt eine syntaktische Vereinfachung für mehrere aufeinanderfolgende Aussagen mit dem selben Subjekt dieses nur in der ersten Aussage zu nennen und in den folgenden wegzulassen; die folgenden Zeilen werden dann mit einem Semikolon beendet. Sind in aufeinanderfolgenden Aussagen jeweils Subjekt und Prädikat gleich, werden sie nur in der ersten Aussage genannt und die einzelnen Aussagen werden per Komma getrennt. IRIs werden ausgeschrieben in spitze Klammern gesetzt, können aber mittels des `@prefix`-Kommandos zu CURIEs abgekürzt werden. Die Sonderform der leeren IRI (<>) steht dabei stellvertretend für die IRI des aktuellen Dokuments. Anonyme Knoten können in Turtle, wenn kein dokumentenlokaler Bezeichner für den Knoten vergeben werden soll, durch eckige Klammern abgekürzt werden. Alternativ kann der anonyme Knoten im Dokument benannt werden, in dem er mit einer CURIE benannt wird, deren Präfix ein Unterstrich ist. Literale werden stets in Anführungszeichen geschrieben, Datentypen und Language Tags werden notiert wie in Tabelle 3.1 gezeigt. Die Nutzung der CURIEs mit `xsd`-Präfix erfordern eine entsprechende `@prefix`-Deklaration. Alle in Quellcodes genutzten IRIs und die zugehörigen Präfixe befinden sich in Anhang E.

Syntax	Typ	Beschreibung
`"Text"`	`xsd:string`	*Plain Literal*, seit RDF 1.1 implizit xsd:string
`"Text"^^xsd:string`	`xsd:string`	Explizit typisiertes Literal
`"5"^^xsd:int`	`xsd:int`	Explizit typisiertes Literal
`"Text"@en`	`rdf:langString`	Literal mit Language Tag

Tabelle 3.1: Turtle-Syntax für Literale

Die TriG- bzw. N-Quads Syntaxen sind Erweiterungen der bestehenden Turtle- bzw. N-Triples-

Syntax, die es erlauben, mehrere benannte Graphen innerhalb eines Dokuments zu spezifizieren.

3.3.2 Resource Description Framework Schema (RDFS)

RDF beschreibt zunächst nur ein Datenmodell und Syntaxen für die Notation von Graphen. Für den ursprünglich angedachten Einsatzzweck, die Zusammenführung von Informationsfragmenten im Web, sind Modell und Syntax alleine allerdings nicht ausreichend. Insbesondere muss es eine Möglichkeit geben, gemeinsam genutztes terminologisches Wissen festzulegen. Hierfür wird das Resource Description Framework Schema (RDFS, [w3c14c]) eingesetzt. RDFS kann in erster Näherung als ein RDF-Dokument verstanden werden, in dem abstrakte aber wiederkehrende Konzepte beschrieben sind, die die Strukturierung weiterer RDF-Dokumente erleichtern. Eines dieser Konzepte ist die Unterteilung von RDF-Knoten in Instanzen (Knoten, die konkrete Entitäten repräsentieren), Klassen (Mengen von Instanzen) und Relationen. Auf diese Weise können komplexere Datenmodelle intuitiv verständlich verfasst werden. Im weiteren Verlauf soll die Notation aus Abbildung 3.7 für diese Typen von Knoten verwendet werden.

Abbildung 3.7: Notation für RDF- und RDFS-Graphen

RDFS legt allerdings nicht nur dieses Vokabular fest, sondern definiert auch die Semantik der Konzepte. Ein einfaches Beispiel hierfür ist eine Unterklassen-Beziehung: Werden zwei Klassen definiert, von denen eine eine Unterklasse der anderen ist, so legt die RDFS-Semantik fest, dass alle Elemente der Unterklasse auch Elemente der Oberklasse sind, analog zu einer Untermenge einer Menge. Verarbeitungssoftware, die RDFS unterstützt, implementiert diese Semantik.

Die wesentlichen Element des RDFS-Vokabulars und ihre Beziehungen untereinander sind in Abbildung 3.8 dargestellt. Alle Bestandteile eines RDF-Graphen sind eine `rdfs:Resource`. Alle Ressourcen, die weitere Ressourcen als Menge zusammenfassen sind eine `rdfs:Class`, daher ist auch `rdfs:Resource` eine `rdfs:Class`. Die Menge aller Literale wird mit `rdfs:Literal` bezeichnet, die Menge aller Relationen mit `rdfs:Property`, sowie die Menge aller Datentypen mit `rdfs:Datatype`. Die Klasse aller Klassen (`rdfs:Class`) ist somit auch selbst eine Klasse. Die für RDF definierten Datentypen (siehe Abbildung 3.4) sind daher jeweils ein `rdfs:Datatype`

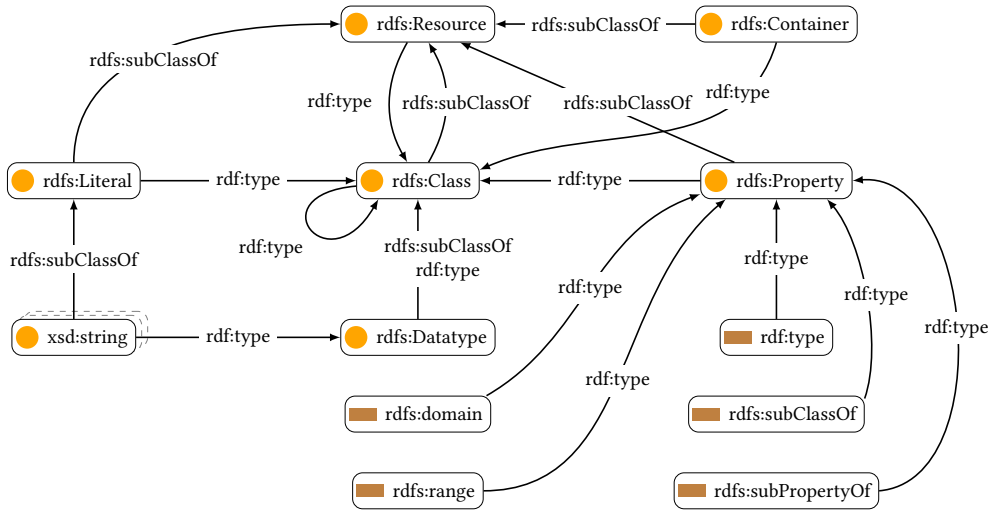

Abbildung 3.8: Kernelemente von RDFS

und gleichzeitig Unterklasse von `rdfs:Literal`. Neben den Relationen für Unterklassenbeziehungen (`rdfs:subClassOf`) und Unterrelationen (`rdfs:subPropertyOf`) definiert RDFS auch Relationen, die für eine Relation den Definitionsbereich (`rdfs:domain`) und den Wertebereich (`rdfs:range`) festlegen.

Nicht in Abbildung 3.8 gezeigt sind die Relationen `rdfs:label` und `rdfs:comment`, mit denen einer Ressource ein menschenlesbarer Name beziehungsweise ein Kommentar zugeordnet werden kann. Außerdem können mittels `rdfs:seeAlso` und `rdfs:isDefinedBy` Verweise auf IRIs extern definierter Ressourcen festgelegt werden.

RDF definiert zusammen mit RDFS drei verschiedene Typen von Datensammlungen. Die Sammlungstypen `rdf:Alt` und `rdf:Bag` sind Unterklassen von `rdfs:Container`, wobei Alt die Klasse von Sammlungen für alternative Auswahlmöglichkeiten ist und Bag die Klasse von ungeordneten Sammlungen. Der dritte Sammlungstyp, `rdf:List`, wird für die Definition einfach verketteter Listen genutzt und ist damit der einzige Typ für geordnete Sammlungen. List ist keine Unterklasse von `rdfs:Container`. Bei einer solchen Liste wird jeder Listen-Knoten durch einen anonymen Knoten repräsentiert, der durch die Relation `rdf:type` mit `rdf:List` verbunden ist, sowie durch `rdf:first` mit seinem jeweiligen Datenelement und durch `rdf:rest` mit dem folgenden Teil der Liste. Das Ende sowie eine leere Liste werden markiert durch den Knoten `rdf:nil`. In Turtle kann eine solche Liste durch die Aufzählung ihrer Elemente, eingeschlossen von runden Klammern, angegeben werden. Abbildung 3.9 zeigt eine RDF-Liste in Turtle-Syntax und den dazugehörigen RDF-Graphen.

Die Nutzung des RDFS-Vokabulars zusammen mit Software, die die RDFS-Semantik umsetzt, erfüllt die Kriterien einer Ontologie-Sprache. Da die Ausdrucksmächtigkeit von RDF mit RDFS

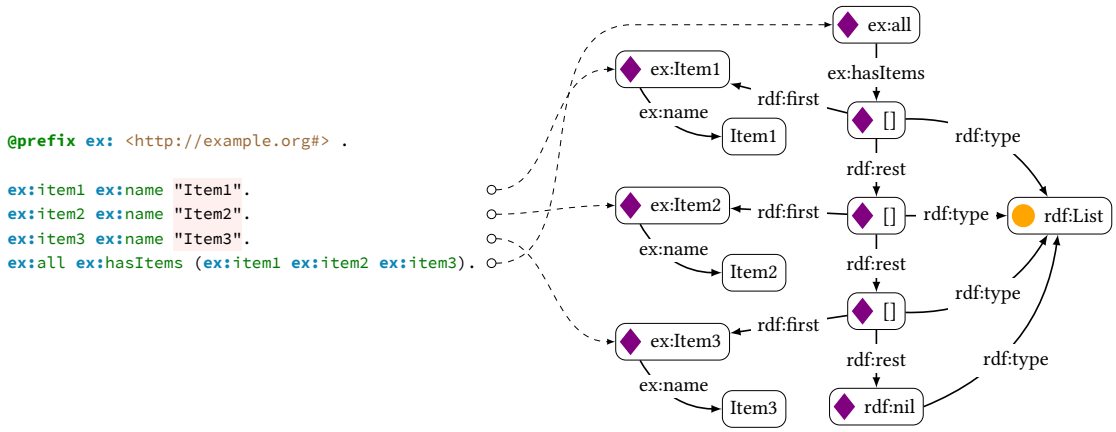

Abbildung 3.9: RDF-Listen

bedeutend schwächer ist, als die von designierten Ontologie-Beschreibungssprachen, wird hier auch von leichtgewichtigen Ontologien besprochen. Die Definition von komplexen Schnittmengen oder Vereinigungen ist zwar mit RDFS nicht möglich, eine Inferenzengine kann aber durch Unterklassenbeziehungen, Definitionsbereiche und Wertebereiche Zugehörigkeiten von Instanzen zu Klassen schlussfolgern.

Es ist in RDF auch möglich, Aussagen über Aussagen zu machen, dieses Vorgehen wird *Reifikation* genannt. Dazu wird stellvertretend für die Aussage, auf die Bezug genommen werden soll, ein anonymer Knoten angelegt, der per `rdf:type` mit der Klasse `rdf:Statement` verbunden wird. Der Knoten wird dann mit mit den Relationen `rdf:subject`, `rdf:predicate` und `rdf:object` mit den Elementen der ursprünglichen Aussage verbunden. Abbildung 3.10 zeigt einen Beispielgraph zu der Aussage, dass Person1 denkt, dass Person2 ein Buch mag.

Wie zu sehen ist, bietet die Turtle-Syntax hierfür keine syntaktische Vereinfachung (andere RDF-Syntaxen allerdings auch nicht). Da dies für größere Dokumente schnell unübersichtlich werden kann, schlagen Hartig und Thompson eine Erweiterung der RDF-Semantik und der Turtle-Syntax vor, die *Reification Done Right* genannt wird [HT14]. Diese Erweiterung ist allerdings kein Teil von offiziellen Empfehlungen des W3C.

3.4 Web Ontology Language (OWL)

Die Web Ontology Language (OWL, [HPMW07]) ist eine Sprache zur Beschreibung von Wissen oder Fakten in Form einer Ontologie und wurde vom W3C im Rahmen der Semantic Web Initiative entwickelt. Die erste Version von OWL wurde 2004 veröffentlicht [PHH04], in die aktuelle Version 2 von 2012 [HKP+12] flossen darauf hin viele Verbesserungen, die sich aus

```
@prefix ex: <http://example.org#> .

ex:person1 ex:thinks [
    rdf:type rdf:Statement;
    rdf:subject ex:person2;
    rdf:predicate ex:likes;
    rdf:object ex:book1;
] .
```

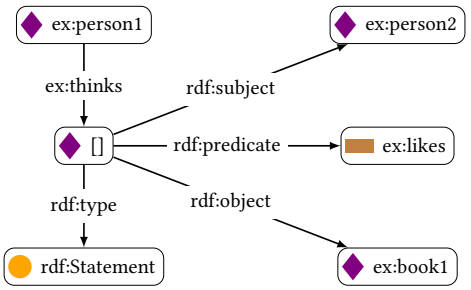

Abbildung 3.10: Reifikation in RDF

Forschung und praktischem Einsatz ergaben. Die Grundlagen für OWL bilden Beschreibungslogiken, durch die Ausdrucksmächtigkeit und Komplexitätsklasse genau festgelegt sind, sowie RDF, das für die Repräsentation einer OWL-Wissensbasis als Graph und ihre Serialisierung verwendet wird. In den folgenden Abschnitten werden zunächst die Grundlagen von Beschreibungslogiken, anschließend die Eigenschaften von OWL, Syntaxen, sowie die Unterschiede zwischen OWL 1 und OWL 2 erklärt.

3.4.1 Beschreibungslogiken

Beschreibungslogiken sind eine Klasse von Formalismen zur Beschreibung von Wissen. Ihre Semantik ist in FOL definiert, die Beschreibungslogiken selbst sind aber mit dem expliziten Ziel entworfen, entscheidbar zu bleiben. Durch die Kombination unterschiedlicher Sprachkonstrukte kann die Ausdrucksmächtigkeit und damit einhergehend ihre Komplexitätsklasse variiert werden. Heute besitzen Beschreibungslogiken insbesondere als theoretische Basis von OWL Bedeutung.

Durch die Kombination von Kürzeln, die definieren, welche Sprachkonstrukte genutzt werden dürfen, entstand eine informelle Namenskonvention, die in [NB07] und [BN07] detailliert beschrieben ist. Tabelle 3.2 beschreibt grob die wichtigsten Kürzel und ihre Bedeutung. Die Quelle hierfür ist, soweit nicht anders angegeben, [HKR08], Seite 165. Die deutschsprachigen Bezeichnungen sind zum Teil aus ([HKRS08], Seite 171f) übernommen.

Tabelle 3.2: Notation von Beschreibungslogiken

Kürzel	Beschreibung
$\mathcal{AL}$	Grundlegende Beschreibungssprache (*attributive language*) mit atomaren Konzepten, atomarer Negation, Schnittmengen und Vereinigungen von Konzepten, Wertbeschränkungen und beschränkter existentieller Quantifizierung ([BN07], Seite 52)
$\mathcal{F}$	Funktionale Rollen (*functional roles*)
$\mathcal{E}$	Existenzielle Rollenrestriktionen (*existential role restrictions*)

Tabelle 3.2: Notation von Beschreibungslogiken (fortgesetzt)

Kürzel	Beschreibung
$\mathcal{U}$	Vereinigung von Konzepten (*unification*, [BN07], Seite 53)
$\mathcal{C}$	Negation beliebiger Konzepte (*complement*, [BN07], Seite 53)
$\mathcal{S}$	Abkürzung für $\mathcal{ALC}$ plus Rollentransitivität ($\mathcal{ALC}_{\mathcal{R}+}$, [Baa07], Seite 535)
$\mathcal{H}$	Rollenhierarchie (*role hierarchy*, [Baa07], Seite 535)
$\mathcal{R}$	Generalisierte Rolleninklusion (*role inclusion*)
$\mathcal{O}$	Nominale (geschlossene Klassen mit einem Element, *nominals*)
$\mathcal{I}$	Inverse Rollen (*inverse roles*)
$\mathcal{N}$	Unqualifizierende Zahlenrestriktionen (*number restriction*)
$\mathcal{Q}$	Qualifizierende Zahlenrestriktionen (*qualified number restriction*)
$(\mathcal{D})$	Datentypen (*data types*)
$\mathcal{EL}^{++}$	Äquivalent zu $\mathcal{ELRO}$ [MMH11]

Die Aussagen innerhalb einer Beschreibungslogik werden in der Regel in zwei Gruppen geteilt, die *TBox* und die *ABox*. Die TBox (für *Terminologie*) beschreibt Hierarchien von Konzepten, oder auch Klassen, also abstrakte Konzepte und die Beziehungen zwischen ihnen. Ein Beispiel hierfür ist die Aussage „Jeder Angestellte ist eine Person“. Die ABox (für *assertionales Instanzwissen* [HKRS08], Seite 167) beschreibt Wissen über konkrete Individuen, zum Beispiel „Alice ist eine Angestellte“ oder „Alice hat das Alter 35“. Für die Beschreibungslogik $\mathcal{SROIQ}$ ist es außerdem üblich, auch die sogenannte *RBox* zu benennen. Die RBox (für *Rollen*) beinhaltet alle atomaren Rollen und ihre Inversen, sowie die universelle Rolle, von der atomaren Rollen erben. Im Beispiel ist „hat das Alter“ eine Rolle.

Für die Notation von Ausdrücken in Beschreibungslogiken werden Operatoren wie $\sqsubseteq$ für Unterklassenbeziehungen und $\equiv$ für Äquivalenz zwischen Klassen verwendet; $\sqcap$, $\sqcup$ und $\neg$ stehen für Konjunktion, Disjunktion bzw. Negation. Formal besteht die TBox aus Aussagen der Form $C \equiv D$ oder $C \sqsubseteq D$, wobei C und D Klassenausdrücke sind. Das TBox-Beispiel würde daher geschrieben werden als *Angestellte* $\sqsubseteq$ *Person*. Das erste ABox-Beispiel drückt Klassenzugehörigkeit aus und würde geschrieben werden als $\mathit{Angestellte}(\mathit{Alice})$, das zweite Beispiel als $\mathit{hatAlter}(\mathit{Alice}, 35)$. Eine Übersicht über die Notation der Beschreibungslogik-Konstrukte, die in dieser Arbeit zum Einsatz kommen, findet sich in den Tabellen 3.3 und 3.4.

3.4.2 OWL, OWL 2 und OWL 2 Profile

Im Rahmen der Bestrebungen der Semantic Web Initiative entstanden mehrere Formate zur Beschreibung von Ontologien bzw. der Annotation von Web-Dokumenten. Die *DARPA Agent*

Markup Language (DAML), die auf Basis von XML und RDF entwickelt wurde, hatte die Beschreibung von maschinenlesbaren Inhalten für das Web zum Ziel. Im gleichen Zeitraum entstand OIL (*Ontology Inference Layer*), das Konzepte aus Frame-basierten Systemen und Beschreibungslogiken verband und auf RDFS basierte. Beide Sprachen wurden kombiniert zur neuen Sprache DAML+OIL, die schließlich die Grundlage der 2004 veröffentlichten OWL (Web Ontology Language) wurde. DAML+OIL war die erste Sprache zur Wissensrepräsentation, die das Teilen von Wissen mit einer formalen Semantik verband, die durch eingeschränkte Ausdrucksmächtigkeit das korrekte und vollständige automatisierte Herleiten von Wissen (*Reasoning*) ermöglicht.

OWL hat drei Ausprägungen (auch *Spezies* genannt [Hoe09], Seite 52): OWL Full, OWL DL und OWL Lite. OWL Full ist eine semantische Erweiterung von RDFS und ist daher nicht entscheidbar, ein Reasoning ist also nicht möglich. Die Semantik von OWL DL (für Description Logics) entspricht der Beschreibungslogik $\mathcal{SHOIN}^{(\mathcal{D})}$ und ist diejenige Ausprägung von OWL, die als möglichst ausdrucksstark konzipiert wurde. Die dritte Ausprägung, OWL Lite, sollte durch weitere Einschränkung der Ausdrucksmächtigkeit das Reasoning erleichtern und der Beschreibungslogik $\mathcal{SHIF}^{(\mathcal{D})}$ entsprechen. Wie es sich herausstellte, war es aber innerhalb des syntaktischen Rahmens von OWL Lite möglich, über diese Mächtigkeit hinausgehende Ausdrücke zu formulieren ([Hoe09], Seite 53).

Die Probleme mit OWL 1 umfassten zusätzlich das Fehlen von in der Praxis benötigen Ausdrucksmöglichkeiten und führten schließlich zur Entwicklung des Nachfolgers OWL 2, der 2009 veröffentlicht wurde [HKP$^+$12, w3c12]. OWL 2 basiert auf der Beschreibungslogik $\mathcal{SROIQ}^{(\mathcal{D})}$ und führt gegenüber OWL 1 DL eine Reihe neuer Features ein, die jedoch die Komplexität und Umsetzbarkeit in Reasonern nicht einschränken [HKS06]. So können nun auch Rollen als disjunkt voneinander markiert werden (beispielsweise sind *hatMutter* und *hatSchwester* disjunkt). Ebenso können einzelne Rollen als reflexiv (Beispiel: *kennt*), irreflexiv (Beispiel: *hatGeschwister*) und antisymmetrisch (Beispiel: *istTeilVon*) ausgezeichnet werden. Es können nun auch negative Aussagen über Individuen getroffen werden (Beispiel: *Alice hat nicht das Alter 35*)[2]. Außerdem erlaubt OWL 2 die Beschreibung von komplexer Rolleninklusion, wodurch sich unter Beachtung bestimmter Regeln Rollen als Unterrollen von Ketten von Rollen definieren lassen, beispielsweise: $\mathit{hatOnkel} \sqsubseteq \mathit{hatElternteil} \circ \mathit{hatBruder}$.

Eine weitere strukturelle Änderung bringt OWL 2 mit seiner Einteilung der Sprache in *Profile* [MGH$^+$12]. Der Einsatz von OWL in der Praxis zeigte eine Zahl von typischen Szenarien, in denen bestimmte Untermengen der Sprache vermehrt oder ausschliesslich verwendet wurden. Daraus wurden Profile entwickelt, die die Sprache syntaktisch und semantisch in bestimmter Weise einschränken, um möglichst effiziente Werkzeuge für den Umgang mit dem jeweiligen Szenario entwickeln zu können. Das Profil OWL EL basiert auf der Beschreibungslogik $\mathcal{EL}^{++}$, die von Baader et al. in [BBL05] beschrieben wurde und ist besonders für das Reasoning von

[2] Aussagen der Form *Alice ist keine Angestellte* waren auch in OWL 1 schon durch die Kombination von Klassendisjunktion und Nominalen möglich: $\mathit{Angestellte} \sqcap \{\mathit{Alice}\} \sqsubseteq \bot$

Ontologien mit großen TBoxen geeignet, die beispielsweise in der biomedizinischen Forschung Verwendung finden. Das Profil OWL QL (für *Query Languages*) zielt dagegen auf den Einsatz bei relativ unkomplizierten TBoxen, aber sehr großen ABoxen ab. In solchen Fällen werden die betreffenden Informationen gerne mittels relationalen Datenbanken verwaltet, daher ist das QL-Profil so angelegt, dass die Beschreibungslogik-Ausdrücke durch Umschreiben der Abfrage in SQL übersetzt werden können. Das dritte von OWL 2 definierte Profil ist OWL RL (für *Rule Languages*). Das RL-Profil basiert auf sogenannten *Description Logic Programs* [GVD03] und ist darauf ausgelegt, mittels einer Regelengine implementierbar zu sein. Dabei ergeben sich mehrere Vorteile: Hierfür geeignete Regelengines werden auch außerhalb des Anwendungsgebiets von Ontologien entwickelt und gepflegt; Reasoning kann potentiell gut skalierbar umgesetzt werden und nicht zuletzt sind die zugrunde liegenden Regelmengen gut verständlich. Analog zu OWL 1 Full wird die Obermenge der Profile als OWL 2 Full bezeichnet, die ebenfalls nicht entscheidbar ist, sie besitzt aber in dieser Form noch weniger praktische Relevanz als OWL 1 Full.

Zur Serialisierung einer OWL-Ontologie sind, ähnlich wie bei RDF, verschiedene Syntaxen definiert. Die *OWL 2 XML Syntax* und die *Manchester Syntax* sollen hier erwähnt, aber nicht im Detail beschrieben werden; sie können in der entsprechenden Spezifikation [MPP12] nachgelesen werden. Für diese Arbeit relevant sind dagegen die *OWL Functional Syntax*, die auch für die Notationen in der OWL-Spezifikation verwendet wird, und die Abbildung von OWL auf RDF [PM12]. Der Zusammenhang von OWL und RDF, der sich bereits aus der Geschichte der Entwicklung ergibt, führt zu wesentlichen Alleinstellungsmerkmalen gegenüber anderen Formen der Wissensrepräsentation. Während Beschreibungslogiken die semantische Grundlage und die klare Abgrenzung der Ausdrucksfähigkeit von OWL bilden, führen die von RDF zugesicherte Referenzierbarkeit von Entitäten durch IRIs sowie die Verknüpfung als Graph zur Teilbarkeit und Austauschbarkeit von Wissen.

Dies wird unterstützt durch das Konzept der *Open World Assumption*, das Beschreibungslogiken und demnach auch dem Reasoning von OWL in der Regel zugrunde liegt[3]. Die Intuition des Konzepts besagt, dass das Fehlen von Wissen über eine Aussage nicht gleichbedeutend mit einer Aussage über eine entsprechende Negation ist. Würde also eine Wissensbasis, die nur die Aussage *Alice ist eine Angestellte* enthält, abgefragt werden, ob auch *Bob* ein Angestellter ist, so ist die Antwort undefiniert. Dadurch führt ein späteres Zufügen von entweder *Bob ist ein Angestellter* oder *Bob ist kein Angestellter* in keinem Fall zu einem Widerspruch, die Monotonie wird gewahrt. Diese Eigenschaft hilft bei der Ausrichtung von OWL auf den ursprünglich angedachten Einsatzzweck der Verknüpfung von Wissen, das nicht nur im Web verteilt gehalten und von unterschiedlichen Parteien bearbeitet werden kann, sondern das sich auch weiterentwickelt und ändert.

Die Serialisierung nach der Abbildung der OWL-Wissensbasis auf RDF geschieht über eine

[3]Dem entgegen steht sogenanntes *Closed-World Reasoning*, das unter bestimmten Bedingungen ebenfalls eingesetzt werden kann. Eine Übersicht zu diesem Thema geben Grimm und Hitzler in [GH07].

der RDF-Syntaxen. Die Abbildung nutzt bestimmte RDF-Konstrukte weiter, so wird zum Beispiel die Zugehörigkeit eines Individuums zu einer Klasse auch in einer als RDF serialisierten OWL-Ontologie mittels einer `rdf:type`-Beziehung ausgedrückt. Aus dem RDFS-Vokabular werden die Relationen `rdfs:domain` und `rdfs:range` übernommen, die den Definitions- bzw. Wertebereich einer Rolle einschränken. Für die meisten weiteren Ausdrücke definiert OWL ein eigenes Vokabular, insbesondere erwähnt sei hier `owl:Class`: Würde hierfür `rdfs:Class` übernommen, wäre das Modell durch die Beziehungen, die in Abbildung 3.8 dargestellt sind, für einen Reasoner nicht mehr entscheidbar. Ebenso werden separate Relationen für abstrakte und konkrete Rollen eingeführt, OWL unterscheidet hier zwischen `owl:ObjectProperty` bzw. `owl:DatatypeProperty`. Die seit OWL 2 erlaubten anonymen Individuen werden auf anonyme RDF-Knoten abgebildet.

OWL erlaubt die Auszeichnung beliebiger Entitäten und Aussagen durch Annotationen. Für Annotationen an benannten Elementen wird der Wissensbasis ein Annotations-Axiom zugefügt, die Annotation von Aussagen wird analog zur Reifikation in RDF (siehe Abbildung 3.10) durchgeführt. Hierzu wird in der RDF-Serialisierung ein anonymer Knoten vom Typ `owl:Axiom` zugefügt, der über die Relationen `owl:annotatedSource`, `owl:annotatedProperty` und `owl:annotatedTarget` die Elemente der Aussage verbindet. Annotationen werden im Reasoning-Prozess nicht berücksichtigt.

Die Tabelle 3.3 zeigt die wichtigsten, in dieser Arbeit eingesetzten OWL-Konstrukte in der Functional Syntax und ihre Beschreibungslogik-Syntax. In Tabelle 3.4 sind in der selben Form diejenigen Konstrukte aufgeführt, die gemäß der OWL-Spezifikation als Axiome allein stehen können. Auf weitere Eigenschaften von OWL 2 wird hier nicht weiter eingegangen, es sei auf die entsprechenden Spezifikationen verwiesen [HKP+12, w3c12].

Tabelle 3.3: Abbildung zwischen OWL-Ausdrücken und Description Logics

Ausdruck	**DL Syntax**	**Beschreibung**
`Class(`A`)`	A	Klasse
`OjectProperty(`R`)`	R	Abstrakte Rolle
`DataProperty(`D`)`	D	Konkrete Rolle
U	U	Wertebereich
`NamedIndividual(`o`)`	o	Individuum
v	v	Datenwert
`owl:Thing`	$\top$	Top-Konzept
`owl:Nothing`	$\bot$	Bottom-Konzept
`ObjectIntersectionOf(`$C_1 \dots C_n$`)`	$C_1 \sqcap \dots \sqcap C_n$	Schnittmenge von Konzepten
`ObjectUnionOf(`$C_1 \dots C_n$`)`	$C_1 \sqcup \dots \sqcup C_n$	Vereinigung von Konzepten
`ObjectComplementOf(`C`)`	$\neg C$	Komplement von Konzepten
`ObjectSomeValuesFrom(`$R\,C$`)`	$\exists R.C$	Existenz-Restriktion für abstrakte Rollen
`ObjectAllValuesFrom(`$R\,C$`)`	$\forall R.C$	Universal-Restriktion für abstrakte Rollen

Tabelle 3.3: Abbildung zwischen OWL-Ausdrücken und Description Logics (fortgesetzt)

Ausdruck	DL Syntax	Beschreibung
`ObjectHasValue(`$R\,o$`)`	$R \ni o$	Restriktion auf Individuen
`ObjectMinCardinality(`$n\,R$`)`	$\geqslant n\,R$	Minimal-Kardinalität für abstrakte Rollen
`ObjectMinCardinality(`$n\,R\,C$`)`	$\geqslant n\,R.C$	Qualifizierende Minimal-Kardinalität[4]
`ObjectMaxCardinality(`$n\,R$`)`	$\leqslant n\,R$	Maximal-Kardinalität für abstrakte Rollen
`ObjectMaxCardinality(`$n\,R\,C$`)`	$\leqslant n\,R.C$	Qualifizierende Maximal-Kardinalität[5]
`ObjectExactCardinality(`$n\,R$`)`	$= n\,R$	Exakte Kardinalität für abstrakte Rollen
`ObjectExactCardinality(`$n\,R\,C$`)`	$= n\,R.C$	Qualifizierende exakte Kardinalität[6]
`DataIntersectionOf(`$U_1 \ldots U_n$`)`	$U_1 \sqcap \cdots \sqcap U_n$	Schnittmenge von Wertebereichen[7]
`DataUnionOf(`$U_1 \ldots U_n$`)`	$U_1 \sqcup \cdots \sqcup U_n$	Vereinigung von Wertebereichen[8]
`DataComplementOf(`U`)`	$\neg U$	Komplement des Wertebereichs[9]
`DataSomeValuesFrom(`$U\ D$`)`	$\exists U.D$	Existenz-Restriktion für konkrete Rollen
`DataAllValuesFrom(`$U\ D$`)`	$\forall U.D$	Universal-Restriktion für konkrete Rollen
`DataHasValue(`$U\ v$`)`	$U \ni v$	Restriktion auf Datenwerte
`DataMinCardinality(`$n\,U$`)`	$\geqslant n\,U$	Minimal-Kardinalität für konkrete Rollen
`DataMaxCardinality(`$n\,U$`)`	$\leqslant n\,U$	Maximal-Kardinalität für konkrete Rollen
`DataExactCardinality(`$n\,U$`)`	$= n\,U$	Exakte Kardinalität für konkrete Rollen
`ObjectInverseOf(`R`)`	R^-	Inverse einer abstrakten Rolle
`ObjectPropertyChain(`$R_1 \ldots R_n$`)`	$R_1 \circ \cdots \circ R_n$	Komplexe Rolleninklusion[10]
`ObjectOneOf(`$o_1 \ldots o_n$`)`	$\{o_1\} \sqcup \cdots \sqcup \{o_n\}$	Aufzählung von Individuen
`DataOneOf(`$v_1 \ldots v_n$`)`	$\{v_1\} \sqcup \cdots \sqcup \{v_n\}$	Aufzählung von Datenwerten

Tabelle 3.4: Abbildung zwischen OWL-Axiomen und Description Logics

OWL-Axiom	DL Syntax	Beschreibung
`EnumeratedClass(`$A o_1 \ldots o_n$`)`	$A \equiv \{o_1\} \sqcup \cdots \sqcup \{o_n\}$	Geschlossene Klasse[11]
`SubClassOf(`$C_1\,C_2$`)`	$C_1 \sqsubseteq C_2$	Unterklassenbeziehung
`EquivalentClasses(`$C_1 \ldots C_n$`)`	$C_1 \equiv \cdots \equiv C_n$	Klassenäquivalenz
`DisjointClasses(`$C_1 \ldots C_n$`)`	$C_i \sqcap C_j \sqsubseteq \bot, i \neq j$	Klassendisjunktion
`Datatype(`D`)`	*keine Entsprechung*	Datentyp-Deklaration
`SubObjectPropertyOf(`$R_1\,R_2$`)`	$R_1 \sqsubseteq R_2$	Unterrollenbeziehung

[4] Nur in OWL 2
[5] Nur in OWL 2
[6] Nur in OWL 2
[7] Nur in OWL 2
[8] Nur in OWL 2
[9] Nur in OWL 2
[10] Nur in OWL 2
[11] `EnumeratedClass` existiert als syntaktische Vereinfachung nur in der OWL-XML-Syntax, für die Functional Syntax muss `EquivalentClasses` in Kombination mit `ObjectUnionOf` und `ObjectOneOf` verwendet werden.

Tabelle 3.4: Abbildung zwischen OWL-Ausdrücken und Description Logics (fortgesetzt)

OWL-Axiom	**DL Syntax**	**Beschreibung**
`ObjectPropertyDomain(`$C\,R$`)`	$\geqslant 1R \sqsubseteq C$	Definitionsbereich
`ObjectPropertyRange(`$C\,R$`)`	$\top \sqsubseteq \forall R.C$	Wertebereich
`InverseObjectProperties(`$R_1\,R_2$`)`	$R_1 \equiv R_2^-$	R_1 und R_2 sind inverse Rollen
`FunctionalObjectProperty(`R`)`	$\top \sqsubseteq \leqslant 1R$	Funktionale abstrakte Rolle
`InverseFunctional ObjectProperty(`R`)`	$\top \sqsubseteq \leqslant 1R^-$	Invers funktionale abstrakte Rolle
`ReflexiveObjectProperty(`R`)`	$\top \sqsubseteq \exists R.\mathit{Self}$	Reflexive Rolle[12]
`IrreflexiveObjectProperty(`R`)`	$\top \sqsubseteq \neg\exists R.\mathit{Self}$	Irreflexive Rolle[13]
`SymmetricObjectProperty(`R`)`	$R \equiv R^-$	Symmetrische Rolle
`DisjointObjectProperties(`$R_1\,R_2$`)`	$\mathit{Disjoint}(R_1, R_2)$	Abstrakte Rollen sind disjunkt[14]
`AsymmetricObjectProperty(`R`)`	$\mathit{Disjoint}(R, R^-)$	Asymmetrische Rolle[15]
`TransitiveObjectProperty(`R`)`	$R \circ R \sqsubseteq R$	Transitivität[16]
`FunctionalDataProperty(`D`)`	$\top \sqsubseteq \leqslant 1D$	Funktionale konkrete Rolle
`DisjointDataProperties(`$D_1\,D_2$`)`	$\mathit{Disjoint}(D_1, D_2)$	Konkrete Rollen sind disjunkt[17]
`ClassAssertion(`$C\,o$`)`	$C(o)$	Klassenelement
`ObjectPropertyAssertion(`$R\,o_1\,o_2$`)`	$R(o_1, o_2)$	Beziehungsaussage
`NegativeObjectProperty Assertion(`$R\,o_1\,o_2$`)`	$(o_1, o_2) : \neg R$	Negative Aussage[18]
`DataPropertyAssertion(`$D\,o\,v$`)`	$D(o, v)$	Beziehungsaussage
`NegativeDataProperty Assertion(`$D\,o\,v$`)`	$(o, v) : \neg D$	Negative Aussage[19]
`DifferentIndividuals(`$o_1 \ldots o_n$`)`	$\{o_i\} \sqcap \{o_j\} \sqsubseteq \bot,$ $1 \leq i < j \leq n$	Individuen sind paarweise disjunkt

Für die grafische Darstellung von OWL-Ontologien und Fragmenten von OWL-Ontologien soll in dieser Arbeit im Folgenden eine Notation des Autors verwendet werden, die auf der Darstellung von RDF-Graphen (siehe Abbildung 3.7) sowie der DL-Syntax basiert. OWL-spezifische Knoten und einige Beispiele sind in Abbildung 3.11 zu sehen. Die OWL-Spezifikation beschreibt selbst keine offizielle grafische Notation; in der Literatur wird hierfür daher häufig UML eingesetzt. Damit können zwar Taxonomien und Beziehungsaussagen dargestellt werden, allerdings keine erweiterten Konzepte wie Klassenrestriktionen, Schnittmengen oder Rolleneigenschaften.

[12] Nur in OWL 2. Notation aus [KSH13], alternativ auch *Ref*(R) ([HKR08], Seite 169).
[13] Nur in OWL 2. Notation aus [KSH13].
[14] Nur in OWL 2. Notation aus [KSH13].
[15] Nur in OWL 2. Notation aus [KSH13], alternativ auch *Asy*(R) ([HKR08], Seite 169).
[16] Alternative Notationen hierfür sind *Tr*(R) ([HPMW07]) sowie *Trans*(R) ([HKR08], Seite 164).
[17] Nur in OWL 2.
[18] Nur in OWL 2.
[19] Nur in OWL 2.

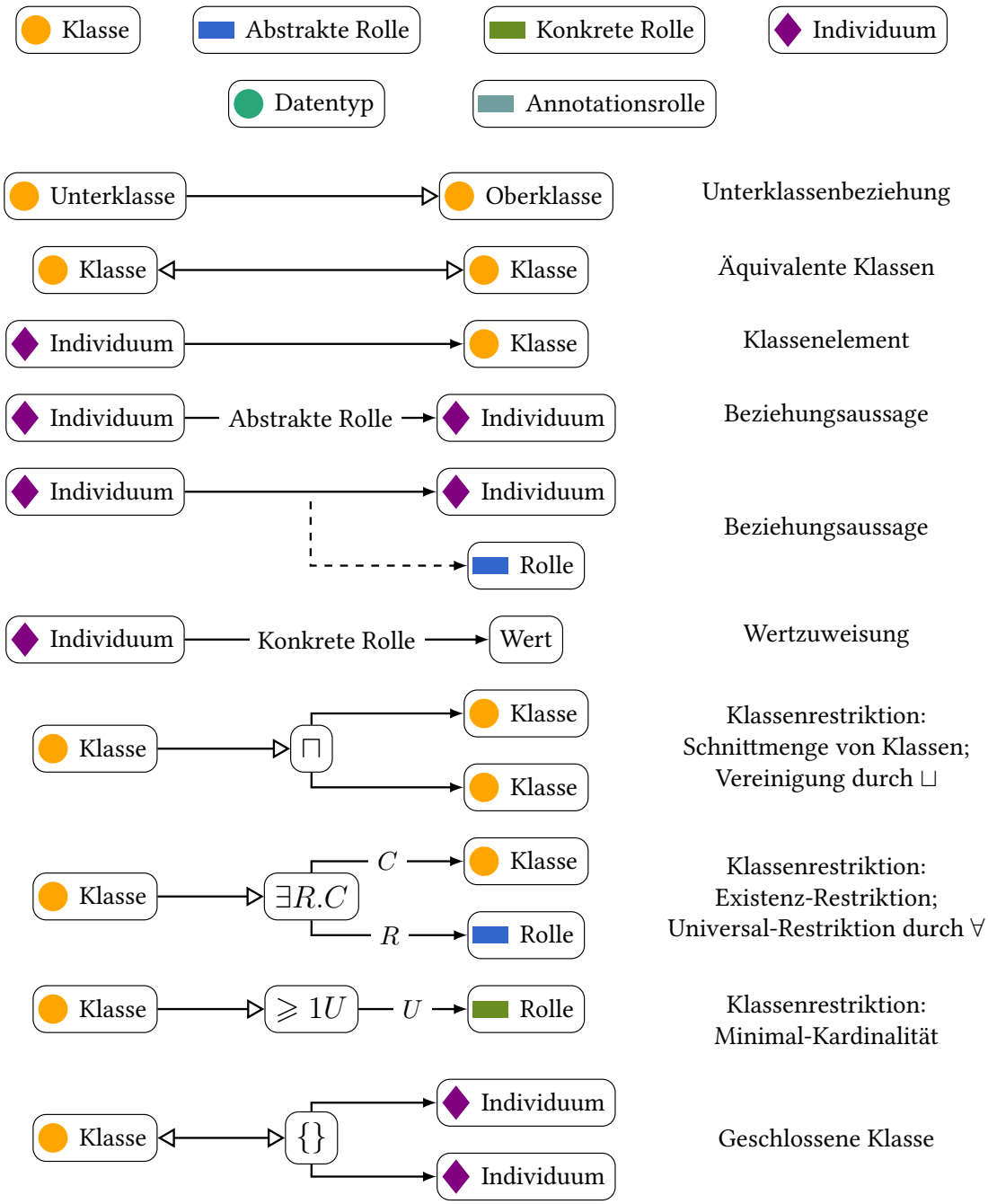

Abbildung 3.11: Grafische Notation für OWL (Fortsetzung auf Seite 44)

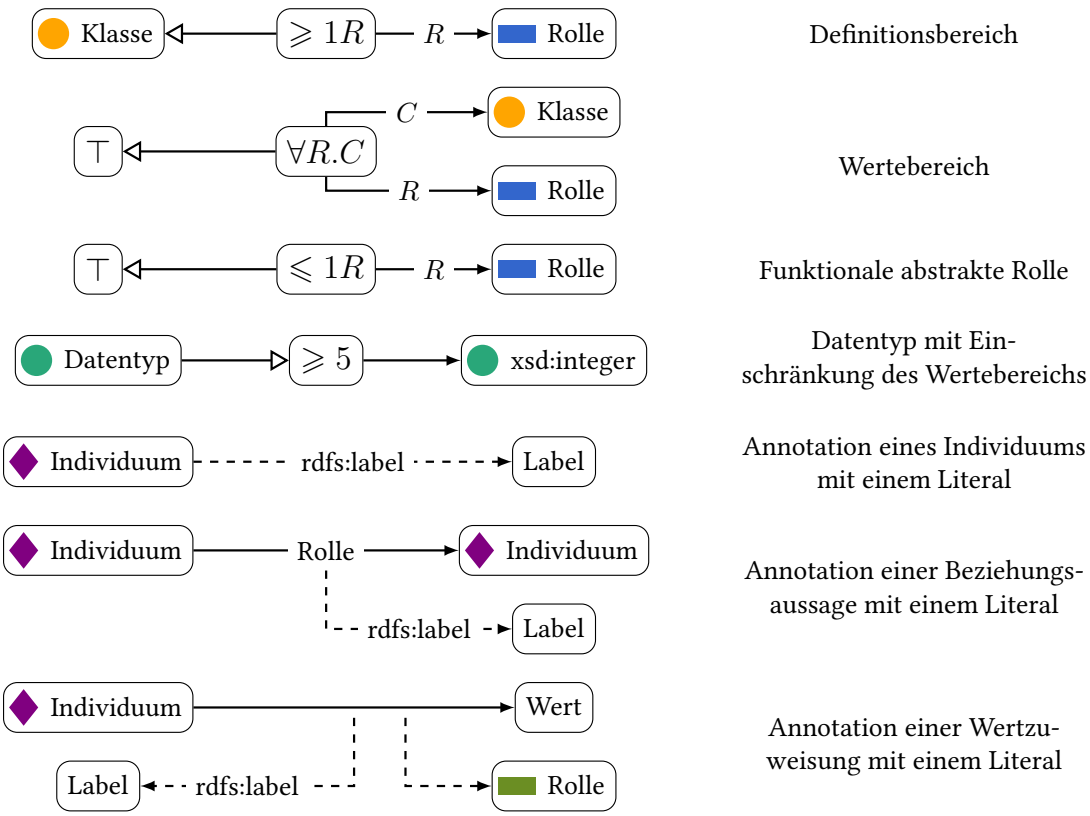

Abbildung 3.11: Grafische Notation für OWL (fortgesetzt)

3.5 SPARQL

SPARQL (*SPARQL Protocol and RDF Query Language*, [w3c13b]) ist ein umfassender Standard, der in erster Linie eine Abfragesprache für Daten im RDF-Format beschreibt. Darüber hinaus werden unterschiedliche Formate definiert, in denen Resultate einer solchen Abfrage repräsentiert werden können, sowie Details zum Aufbau von und der Kommunikation mit einem *SPARQL-Endpunkt* – einer HTTP-basierten Schnittstelle eines RDF-Datenspeichers. Die finale Version der Spezifikation von SPARQL 1.0 wurde 2007 veröffentlicht, die stark überarbeitete Version 1.1, die nicht nur die Änderungen an RDF 1.1 reflektiert, sondern selbst viele Neuerungen brachte, wurde 2013 veröffentlicht.

Quellcode 1 zeigt ein einfaches Beispiel für eine SPARQL-`SELECT`-Abfrage, die für das RDF-Dokument in Abbildung 3.6 ein tabellarisches Ergebnis mit den zwei Spalten „title“ und „name“ und einer Zeile zurückliefern würde.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ex: <http://example.org/stuff/1.0/>

SELECT ?title ?name
WHERE {
  ?document dc:title ?title .
  ?document ex:editor ?editor .
  ?editor ex:fullname ?name .
}
```

Quellcode 1: Beispiel: SPARQL-SELECT-Abfrage

Der Aufbau einer SPARQL-Abfrage ist an SQL angelehnt, die Syntax basiert auf Turtle. Neben vier Typen von lesenden Abfragen (`SELECT` für tabellarische Ergebnisse, ähnlich SQL; `CONSTRUCT` für Ergebnisse in Form eines Graphen; `ASK` zur Bestimmung, ob eine Bedingung erfüllt ist; `DESCRIBE` für eine Selbstbeschreibung des SPARQL-Endpunktes) unterstützt SPARQL seit Version 1.1 durch die Unterspezifikation *SPARQL Update* auch eine Menge von Kommandos zum Einfügen und Ändern von Daten und der Verwaltung von Graphen innerhalb eines RDF-Datenspeichers. Spezialisierte Datenbanken zur Verwaltung von RDF-basierten Datenmengen werden auch als *Triple Stores* bezeichnet.

Neben der Möglichkeit zur Gruppierung von Ergebnissen mittels `GROUP BY`, analog zu SQL, erlaubt SPARQL auch die Verwendung von Unterabfragen im `WHERE`-Block und bringt außerdem eine Vielzahl von Filter- und Umwandlungsfunktionen mit (zum Beispiel String-Funktionen zum Filtern nach regulären Ausdrücken, Zeit- und Datumskonvertierungen, und Umwandlung von IRIs in Strings). Eine umfassendere Übersicht über die Sprachkonstrukte von SPARQL gibt DuCharme in [DuC11].

Da OWL-Ontologien in RDF serialisiert und auch in Triple Stores abgelegt werden können, kann SPARQL prinzipiell auch hier zur Abfrage eingesetzt werden. Dabei muss berücksichtigt werden, dass im Gegensatz zu Abfragen über einen OWL-Reasoner, die in der Regel über einen Tableau-Algorithmus beantwortet werden, der Triple Store nur auf der RDF-Repräsentation der Wissensbasis arbeitet. Daher müssen entweder die Ergebnisse einer Inferenz, die der OWL-Semantik folgt, *materialisiert* werden, also als explizite Knoten der Datenmenge zugefügt werden, bevor die Abfrage ausgewertet werden kann, oder andere Verfahren eingesetzt werden, um die Inferenz umzusetzen. Dies kann durch Umschreiben der Abfrage oder programmatische Behandlung einzelner Fälle geschehen. Komplexe Triple Stores setzen in der Regel eine Kombination aus diesen Techniken ein, da je nach Beschaffenheit der Daten und der Abfragen unterschiedlich zwischen Ausführungszeit der Abfrage und zusätzlichem Speicherbedarf der Materialisierung abgewogen werden muss.

3.6 Ontology Engineering

In den bisherigen Abschnitten dieses Kapitels wurden der Begriff der Ontologie, die grobe historische Entwicklung sowie die etablierten Methoden zur Formalisierung einer Ontologie vorgestellt. Dieser Abschnitt betrachtet das *Ontology Engineering* (auch als *Ontological Engineering* bezeichnet), die Disziplin der systematischen Entwicklung und Pflege von Ontologien. Es bestehen dabei Überschneidungen mit Fragestellungen aus dem Gebiet der Softwaretechnik, die Unterschiede zwischen den Disziplinen sind jedoch so groß, dass sich die meisten etablierten Vorgehensweisen aus der Softwaretechnik nicht direkt übertragen lassen. Fernández-López untersucht in [Fer99] dazu die Anwendbarkeit des Softwareentwicklungsprozesses nach IEEE 1074-1995 (inzwischen abgelöst durch Version 1074-2006, [IEE06]) auf die Entwicklung von Ontologien und stellt fest, dass sich Rahmenaspekte wie ein Lebenszyklus, ein benötigter Managementprozess oder die Erfassung von Anforderungen übertragen lassen, die eigentlichen Design- und Implementierungsprozesse können jedoch nicht übernommen werden.

Nach der Definition von Gómez-Pérez et al. ([Gom04], Seite 5) umfasst das Ontology Engineering die Menge von Aktivitäten, die den Ontologie-Entwicklungsprozess, den Ontologie-Lebenszyklus sowie die Methodiken, Werkzeuge und Sprachen zum Aufbau von Ontologien betreffen. Zwischen der Veröffentlichung dieses Standardwerkes und der Untersuchung zum State of the Art im Ontology Engineering von Simperl et al. im Jahr 2010 [SMB10] stellte sich zunehmend heraus, dass es häufig nicht sinnvoll ist, Ontologien unabhängig von der Software zu betrachten, die sie nutzen oder verarbeiten soll.

Für alle Bestandteile der obigen Definition finden sich in der Literatur mehr oder weniger detaillierte Lösungsansätze, oft ist jedoch noch keine breit einsetzbare Lösung mit zufriedenstellendem Reifegrad erreicht. In den folgenden Unterabschnitten 3.6.1 und 3.6.2 soll zunächst eine Übersicht gegeben werden zur Einteilung von Ontologien in unterschiedliche Abstraktionsebenen und zum Stand der Technik der damit zusammenhängenden Methodiken zur Erstellung und Pflege von Ontologien. Abschnitt 3.6.3 betrachtet die Frage der Aufteilung von Ontologien in Einzelteile und Abschnitt 3.6.4 geht auf bewährte Vorgehensweisen bei der Erstellung von Ontologien ein.

3.6.1 Abstraktionsebenen

Ein Kernproblem des Ontology Engineering, das früh erkannt wurde aber bis heute keine definitive Lösung hat, ist die Wiederverwendbarkeit des formalisierten Wissens. Im Idealfall, analog zur Softwaretechnik, lassen sich einzelne, hinreichend abstrakte Bausteine zusammengefasst in Bibliotheken unverändert in neuen Anwendungen weiterverwenden. Dabei hängt die Nutzbarkeit solcher Bausteine nach Hoekstra ([Hoe09], Seite 93) von vier Faktoren ab, in denen sie sich unterscheiden können: der Abstraktionsebene, dem Umfang der Abdeckung der modellierten Domäne, sowie der Sprache und dem Grad der Formalisierung, in der die Ontologien

umgesetzt sind. Während die letzten drei Faktoren stärker von Entscheidungen zur Umsetzung der Ontologie im Einzelfall abhängen, kann die Einordnung der Ontologie in eine benannte Abstraktionsebene schon früh im Entwicklungsprozess den späteren Einsatzzweck eingrenzen. Hierbei unterscheidet sich das Ontology Engineering von der vergleichbaren Situation in der Softwaretechnik: Bei der Erstellung von Datenmodellen mittels UML wird klar zwischen den Meta-Ebenen M0, M1, M2 und M3 unterschieden, die die Instanzebene, die Modellebene, die Meta-Modellebene beziehungsweise die Meta-Meta-Modellebene bezeichnen. Diese Unterscheidung wird bei Modellierung einer Ontologie nicht getroffen – Instanz- und Modellebene existieren im selben Axiomraum; die Meta-Modellebene ist durch die Semantik der Sprache gegeben – stattdessen werden Abstraktionsebenen anhand der ausgedrückten *Konzepte* unterschieden. So enthält beispielsweise eine Ontologie aus der höchsten Abstraktionsebene generische Konzepte wie *Endurant* (eine Entität, die unabhängig von zeitlichen Betrachtungen ist, wie physikalische Objekte oder Personen) und *Perdurant* (ein Ereignis oder ein Verfahren), eine Ontologie aus einer niedrigen Abstraktionsebene würde spezifische konkrete Konzepte wie *HerstellerX-PlatineY* enthalten.

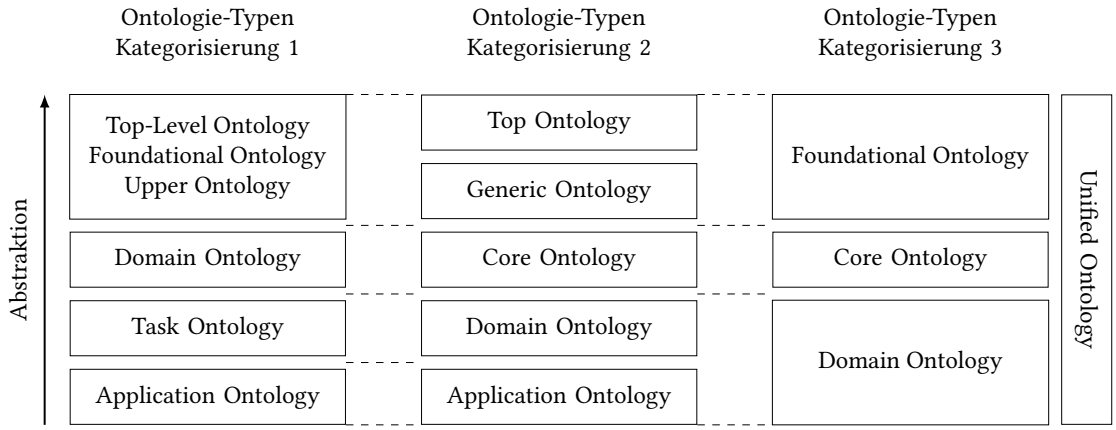

Abbildung 3.12: Kategorisierungen von Ontology-Typen

In der Literatur werden mehrere Kategorisierungen von Ontologie-Typen verwendet, deren Begrifflichkeiten sich ähneln, aber nicht immer das Gleiche meinen. Die Kategorisierungen werden hier mit 1, 2 und 3 bezeichnet und sind in Abbildung 3.12 zusammengefasst. Allen gemeinsam ist die Benennung einer obersten Abstraktionsebene, hierfür werden die untereinander austauschbaren Begriffe *Top-Level Ontology*, *Foundational Ontology*, *Upper Ontology* oder *Upper Level Ontology* verwendet (die in Kategorisierung 2 benannte *Top Ontology* beinhaltet nur eine Teilmenge dessen). Kategorie 1 wird in den Arbeiten von Garcia et al. [GOG+10] und Sure et al. [Sur03, SSS09] verwendet und beinhaltet auch eine separate *Task Ontology*, die das Vokabular zu Aufgaben und Aktivitäten der Anwendung enthält, für die die Ontologie entwickelt wird. Die *Application Ontology* enthält das anwendungsspezifische Vokabular. Die

detailliertere Einteilung der Kategorisierung 2 wurde von van Heijst et al. [vSW97] beschrieben und von Hoekstra [Hoe09] aufgegriffen, und entstammt den Anforderungen an wissensbasierte Anwendungen im biomedizinischen Kontext.

Hoekstra stellte fest, dass die in realen Anwendungen meistens eingesetzte Unterscheidung der Abstraktionsebenen eher der Aufteilung der Kategorisierung 3 entspricht: Die *Foundational Ontology* enthält alle generischen, nicht auf bestimmte Anwendungsfälle festgelegte Konzepte, die *Core Ontology* enthält die Konzepte, die generell innerhalb der betrachteten Domäne (z.B. Rechtswissenschaften oder Biologie) gelten, und die *Domain Ontology* enthält die spezialisierten Detailkonzepte der Domäne, die für die Umsetzung der Anwendung nötig sind. Einen Sonderfall bildet die Abstraktion der sogenannten *Unified Ontology*, die das Ziel verfolgt, alle Abstraktionsebenen in Form eines Gerüsts zu überspannen und darauf ausgelegt ist, in der Horizontalen erweitert zu werden, also durch Zufügen von Detailinformationen.

Es spricht auf den ersten Blick nichts dagegen, in allen Anwendungen die selbe Upper Ontology zu referenzieren, da die dort beschriebenen Konzepte per Definition nicht auf eine Anwendung oder eine Domäne beschränkt sind. Es gab mehrere Bestrebungen zur Entwicklung einer solchen generischen, wiederverwendbaren Upper Ontology. Einer der ersten Versuche hierzu wurde bereits 1977 von Bunge vorgenommen und Anfang der 1990er Jahre von Wand und Weber formalisiert [WW95] – dies wird auch als die Bunge-Wand-Weber-Ontologie bezeichnet. Aus unterschiedlichen Gründen mündeten diese und andere Bestrebungen allerdings nicht in einer einzigen, überall eingesetzten Ontologie, sondern es wurden mehrere Upper Ontologies entwickelt, in Tabelle 3.5 werden die wichtigsten vorgestellt. Eine Übersicht und Vergleich zu Einsatzzweck und Historie einiger Upper Ontologies geben Mascardi et al. in [MCR07].

Tabelle 3.5: Übersicht der wichtigsten Upper Ontologies

Name	**Beschreibung**
BFO	Die Grundlagen für die *Basic Formal Ontology* [BFO15] wurden bereits 1998 von Smith gelegt [Smi98] und später von Grenon et al. [GSG04, BS04] weiterentwickelt und ergänzt. BFO wurde für den Einsatz in der Biomedizin entwickelt und besitzt heute insbesondere in diesem Bereich Wichtigkeit: BFO ist die empfohlene Upper Ontology für neue Ontologien in der OBO (Open Biomedical Ontologies) Foundry ([GOG$^+$10], Seite 104), dem De-Facto-Standard-Repository für Ontologien in diesem Bereich. BFO umfasst ca. 40 Konzepte und ist in den OWL, OBO und CLIF-Formaten verfügbar.
Cyc	Die kommerzielle Cyc-Ontologie [Len95, Fox10, cyc14] fällt in die Kategorie einer Unified Ontology und wurde mit dem Ziel entwickelt, Allgemeinwissen zu formalisieren. Cyc umfasst ca. 240.000 Konzepte und insgesamt ca. 2 Millionen Aussagen und ist in der proprietären Cyc-Language beschrieben (eine Untermenge von Cyc ist ebenfalls im OWL-Format verfügbar).

Tabelle 3.5: Übersicht der wichtigsten Upper Ontologies (fortgesetzt)

Name	Beschreibung
DOLCE	Die *Descriptive Ontology for Linguistic and Cognitive Engineering* [GW00c, GGM+02, DOL06] wurde mit dem Ziel entwickelt, Bedeutung von Nuancen bei der Verarbeitung natürlicher Sprache klar definieren zu können. Sie wurde im Zusammenhang mit der OntoClean-Methodik zur „Bereinigung“ von Taxonomien erstellt, die in Abschnitt 3.6.2 vorgestellt wird. Heute wird häufiger als DOLCE die auf den praktischen Einsatz in Software optimierte reduzierte Version DOLCE+DnS Ultralight eingesetzt (abgekürzt auch DUL, DnS steht für Descriptions and Situations, [Gan09, MCR07]). DUL umfasst ca. 80 Konzepte und 100 Rollen. DOLCE ist im KIF-Format, DUL im OWL-Format verfügbar.
SUMO	Die *Suggested Upper Merged Ontology* [NP01, PNL02] wurde ursprünglich als Grundlage für eine Upper Ontology entworfen, die durch einen Normungsprozess ein IEEE-Standard werden sollte [PN02]. Obwohl dieser nie beendet wurde, ist SUMO mit ca. 4500 Klassen, 750 Rollen und 86.000 Individuen eine der umfangreichsten frei verfügbaren Upper Ontologies. Da teilweise Mappings zu OBO und WordNet bestehen, wird SUMO in akademischen und kommerziellen Projekten unterschiedlicher Gebiete eingesetzt. SUMO ist in einer Variante von KIF beschrieben, eine Teilmenge ist auch im OWL-Format verfügbar.
GFO	Die *General Formal Ontology* [HHB+07, Her13, GFO10] wurde für die konzeptuelle Modellierung entwickelt und enthält neben Begriffen für räumliche und zeitliche Eigenschaften auch Ausdrücke für Funktionen und Situationen. Die Ontologie umfasst ca. 80 Klassen und 70 Rollen. GFO wird im OWL-Format veröffentlicht.
Proton	Die *PROTo ONtology* [TKM05, DKSP10] ist eines der Ergebnisse des EU-Projekts SEKT (Semantically Enabled Knowledge Technologies) und ist als Upper Ontology konzipiert, die sich auf die wesentlichen Elemente beschränkt. Explizite Designziele waren Domänenunabhängigkeit, leichtgewichtige Definitionen, Anpassung an populäre Standards und eine gute Breitenabdeckung von Konzepten. Proton umfasst ca. 270 Klassen und 110 Rollen und ist in OWL beschrieben.
UMBEL	Der *Upper Mapping and Binding Exchange Layer* [BG16] ist eine Upper Ontology, die Entwickelt wurde, um Interoperabilität von Domänenontologien mit externen Datenquellen zu verbessern. Sie besteht aus einer Untermenge der Cyc-Ontologie. Obwohl UMBEL von einem Unternehmen entwickelt wird (Structured Dynamics, LLC.) wird er unter einer Creative Commons Attribution 3.0 License veröffentlicht. UMBEL enthält 70 Klassen und 60 Rollen und ist in OWL beschrieben.

Tabelle 3.5: Übersicht der wichtigsten Upper Ontologies (fortgesetzt)

Name	Beschreibung
YAGO	YAGO (ursprünglich für *Yet Another Great Ontology*, aber in der Regel nicht als Akronym verwendet, [SKW07]) ist eine Upper und Unified Ontology, die am Max-Planck-Institut für Informatik Saarbrücken mithilfe unterschiedlicher heuristischer Verfahren aus der Wikipedia extrahiert wurde. Eine manuelle Prüfung von ca. 4000 Konzepten und anschließender Hochrechnung führt nach Aussage der Autoren zu einer Fehlerfreiheit von ca. 95%. Die Ontologie beinhaltet mehr als 350.000 Klassen und ca. 10 Millionen Individuen und insgesamt mehr als 120 Millionen Aussagen.
UFO	Die *Unified Foundational Ontology* [GW04, Gui05] wurde wie GFO als Grundlage für die konzeptuelle Modellierung entwickelt und basiert im Kern aus einer Untermenge der Vereinigung von GFO und DOLCE. Im Gegensatz zu den meisten anderen Upper Ontologies ist UFO in UML beschrieben.

3.6.2 Ontology Engineering Methodiken

Die Erstellung einer Ontologie ist mit der Modellierungsphase in der Softwareentwicklung zu vergleichen: domänen- und anwendungsspezifisches Wissen müssen erfasst und in die benötigte Form gebracht werden. Die Vorgehensweisen hierzu können sich jedoch stark unterscheiden, abhängig von verschiedenen Faktoren: Welche Art von Ontologie soll entwickelt werden; wie soll das notwendige Expertenwissen erfasst werden; in welcher Granularität und nach welchem Prozess soll die Konzeptualisierung vorgenommen werden; in welchem Formalismus soll die Ontologie implementiert werden; wie soll die Ontologie evaluiert und gewartet werden; in welcher Form soll eine Dokumentation erstellt werden. Zur Frage, wie notwendiges Wissen erfasst werden soll, existieren zahlreiche Arbeiten, da vergleichbare Problemstellungen bereits in der Disziplin des *Knowledge Engineering* auftraten. Knowledge Engineering ist die systematische Herangehensweise zur Entwicklung wissensbasierter Systeme, auch als Experten-Systeme bezeichnet, unter Berücksichtigung technischer und sozialer Aspekte. Die Wissenserfassung wird in diesem Zusammenhang *Knowledge Acquisition* genannt. Abbildung 3.13 zeigt den zugehörigen Ablauf aus dem Standardwerk zu Expertensystemen von Jackson.

Der Prozess besteht aus fünf Einzelschritten, bei denen nach der Testphase ein Feedback zu vorigen Schritten möglich ist. Eine der ersten umfassenden Methodiken zum Knowledge Engineering, die zwischen der Identifikation, Konzeptualisierung und Formalisierung unterschied war KADS, bzw. der Nachfolger CommonKADS (*Knowledge Acquisition and Documentation Structuring*) von Schreiber et al. [SAA+99]. Da die Ziele des Ontology Engineering sich von denen des Knowledge Engineering unterscheiden, lässt sich CommonKADS zwar nicht unverändert hierfür einsetzen, war aber einer der Einflüsse für spätere Ontology Engineering

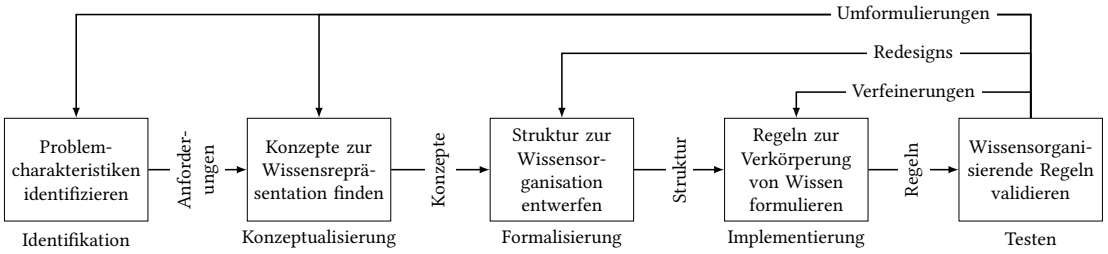

Abbildung 3.13: Prozess der Wissenserfassung nach Jackson ([Jac99], Seite 183)

Methodiken.

Für den komplexen Schritt der Konzeptualisierung existieren drei prinzipielle Ansätze ([UG96], [Gan02], Seite 85):

- Erstens, der von unten aufbauende Ansatz (*bottom-up*), bei dem zunächst die domänenspezifischen Entitäten erfasst oder aus bestehenden Systemen oder Modellen extrahiert werden. Gemeinsame Konzepte werden dann generalisiert, um zu einer Taxonomie zu gelangen. Dies führt zu besonders detaillierten und auf die spezifischen Gegebenheiten angepassten Ontologien.
- Zweitens, der von oben aufbauende Ansatz (*top-down*), bei dem mit den generischsten Konzepten begonnen wird und eine Struktur durch schrittweise Verfeinerung erstellt wird. Hierbei können leichter komplexe Ontologien mit kohärenten Strukturen entstehen, der Einsatz einer bereits bestehenden geeigneten Upper Ontology kann den Aufwand reduzieren.
- Der dritte Ansatz ist eine Mischung aus beiden, bei dem die Ontologie „aus der Mitte heraus“ entworfen wird (*middle-out*). Hier werden zunächst nur die Kernkonzepte erfasst und generalisiert, anschließend wird diese Schicht von Kernkonzepten durch Spezialisierung konkretisiert. Dabei entsteht früh eine thematische Unterteilung von Konzepten durch Gruppierung zusammengehöriger Kernkonzepte. Inhaltliche Felder können dadurch besser identifiziert werden, entsprechende Modularisierung und insgesamt bessere Stimmigkeit der Ontologie sind im Idealfall die Folge.

Keiner der Ansätze ist in allen Anwendungsfällen besser als die anderen geeignet, die Vor- und Nachteile und ihre Auswirkungen müssen jeweils abgewogen werden.

Der prinzipielle Prozess zur Wissenserfassung wurde im Laufe der Zeit von unterschiedlichen Arbeitsgruppen zu kompletten Methodiken ausgebaut. Dabei wurden in der Regel Erkenntnisse aus zuvor entwickelten Methodiken berücksichtigt, allerdings wurde häufig ein anderer Fokus gesetzt. Methodiken unterscheiden sich unter anderem in der Herangehensweise zur Modellierung einer Ontologie – mit stärkeren Einflüssen aus dem Knowledge Engineering oder

dem Software Engineering – aber auch in dem Grad der Betrachtung von Details bestimmter Schritte, z.B. der Formalisierung. Die wichtigsten Ontology Engineering Methodiken, die in der Literatur referenziert werden, werden, nach unterschiedlichen Kriterien aufgeteilt, in Abbildung 3.14 gezeigt.

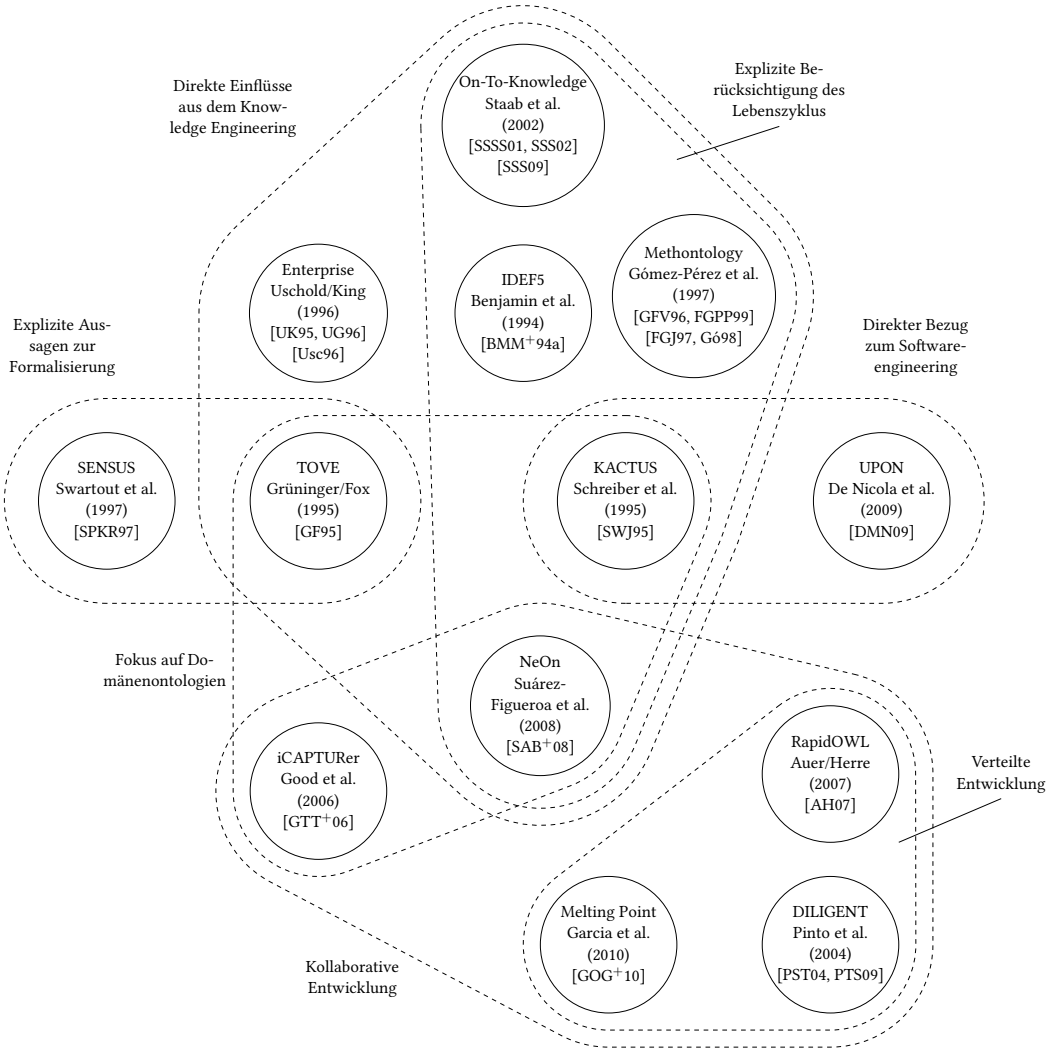

Abbildung 3.14: Übersicht über die bekanntesten Ontology Engineering Methodiken

Es lässt sich eine grobe zeitliche Unterteilung vornehmen nach Methodiken, die bis zum Jahr 2000 entstanden, und denen, die danach entstanden. Einen Vergleich der Eigenschaften von Methodiken, die bis 1999 entstanden, gibt Fernández-López in [Fer99], weitere Details beschreiben ausserdem Corcho et al. in [CFG03]. Bis zur Entwicklung und der Standardisierung von OWL

mussten Entwickler entscheiden, ob die Methodik so generisch sein soll, dass sie nicht auf bestimmte Formalismen festgelegt ist, oder ob konkret auf eine bestimmte Art der Formalisierung oder eine entsprechende Sprache hingearbeitet werden soll. Diese Entscheidung betrifft dann nicht nur den eigentlichen Schritt der Formalisierung, sondern hat bereits Einfluss auf das Vorgehen bei der Konzeptualisierung. Andere Eigenschaften von Methodiken lassen sich in dieser Form bei Software-Engineering-Methodiken nicht wiederfinden, beispielsweise die Entscheidung, für welchen Typ von Ontologien die Methodik geeignet ist (siehe Abbildung 3.12). So ist z.B. die SENSUS-Methodik [SPKR97] entstanden, während eine Domain- bzw. Unified Ontology mit ca. 50.000 Aussagen entwickelt wurde.

Repräsentativ für eine ausgereifte Ontology Engineering Methodik, die auf den zeitlichen Vorgängern aufbaut und den Lebenszyklus einer Ontologie inklusive einer Anwendungsphase berücksichtigt, ist On-To-Knowledge [SSSS01, SSS02, SSS09]. Hierbei werden ein *Knowledge-Meta-Prozess* und ein *Knowledge-Prozess* unterschieden. Der Knowledge-Meta-Prozess (Abbildung 3.15) beschreibt 14 Schritte von der Analysephase über die Entwicklung einer Ontologie bis hin zum Betrieb einer Knowledge-Management-Anwendung, die auf der Ontologie aufbaut. On-To-Knowledge ist stark beeinflusst vom Knowledge-Engineering, im Knowledge-Meta-Prozess wird daher auch auf den oben beschriebenen CommonKADS-Ansatz Bezug genommen. Der Knowledge-Prozess beschreibt die Integration von existierendem, externen Wissen in die Knowledge-Management-Anwendung, nachdem diese implementiert ist und besteht aus den vier zyklisch wiederholten Schritten der Erstellung oder dem Import von Wissen, der Erfassung (Klärung von Bedeutung und Verbindung von neuem Wissen mit dem bereits in der Wissensbasis existierenden), dem Zugriff auf Wissen (durch Bereitstellung entsprechender Schnittstellen und Abfragen) und der Nutzung des Wissens (für den jeweiligen Einsatzzweck).

Die größte Neuerung, auf die bei Ontology Engineering Methodiken nach 2004 der Augenmerk gelegt wurde, ist die Berücksichtigung der kollaborativen Entwicklung von Ontologien, im Spezialfall auch von räumlich getrennten Teams. Dieser Fokus reflektiert die Rolle, die Ontologien im Zuge der Weiterentwicklung des Webs einnehmen; weniger die einmal für eine spezifische Anwendung kreierte monolithische Wissensbasis, sondern mehr dynamisch einsetzbare Wissensmodule. Während DILIGENT [PST04, PTS09] als erste Methodik explizit die verteilte Entwicklung betrachtete, wurden im NeOn-Projekt [SAB+08] erstmals in einer Methodik auch die Wiederverwendbarkeit und Entwurfsmuster für Ontologien aufgegriffen. Die RapidOWL-Methodik [AH07] basiert dagegen als einzige auf den gleichen Prinzipien, die auch bei der agilen Software-Entwicklung zum Einsatz kommen, kürzere Zyklen mit mehr unmittelbarem Feedback werden höher bewertet als umfassende Dokumentation. Die jüngste untersuchte Methodik, *Melting Point*, versucht die besten Elemente der zeitlichen Vorgänger aufzugreifen – darunter auch sowohl Unterstützung für einen Lebenszyklus, als auch für die verteilte Entwicklung – und in eine neue, umfassende Methodik zusammenzufassen. Dazu werden sieben der in Abbildung 3.14 erwähnten Methodiken sowie zwei weitere analysiert. Der resultierende Prozess hat hohe Ähnlichkeit mit dem On-To-Knowledge-Meta-Prozess.

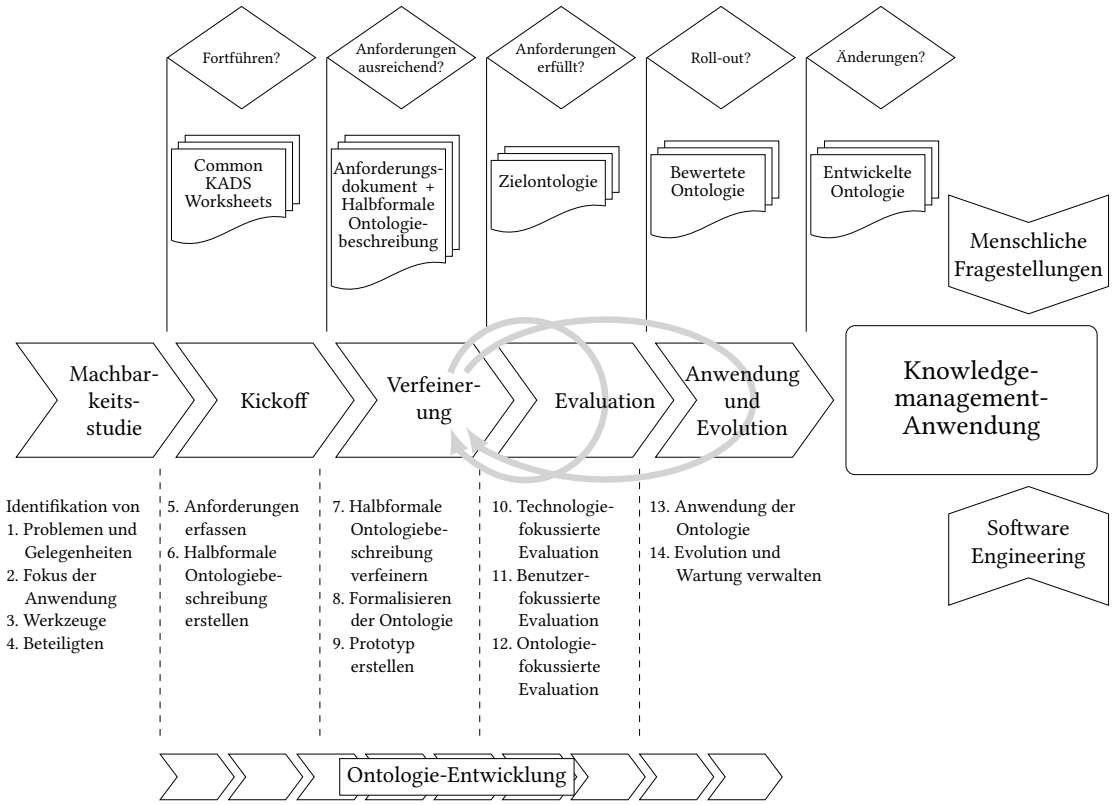

Abbildung 3.15: Der „Knowledge-Meta-Prozess“ nach Sure [Sur03, SSS09]

3.6.3 Modularisierung

Eine Fragestellung, die sowohl bei der Erstellung neuer Ontologien als auch bei der Nutzung bestehender auftritt, ist, wie genau eine Ontologie zerlegt oder aus Einzelteilen zusammengefügt werden kann. Ausgenommen von Ontologien, die einen so geringen Umfang haben, dass sie nicht sinnvoll aufgeteilt werden können, und die auch keine weiteren Ontologien referenzieren, betrifft diese Frage jeden Ontology Engineering Prozess. Unterschiedliche Teilfragestellungen wurden identifiziert und unter der Verwendung separater Terminologien in der Literatur behandelt. Die wichtigsten Begriffe und Lösungsvorschläge sollen hier zusammengefasst werden.

Parent und Spaccapietra geben in [PS09] eine Übersicht zu den Zielen, die den Begriff der *Ontologiemodularisierung* ausmachen. Ähnlich zu Modulen im Software Engineering sollen Ontologie-Module zum einen die Wartbarkeit, Übersicht und Verständlichkeit erhöhen. Zusätzlich soll aber auch die Skalierbarkeit gewährleistet werden, sowohl für Abfragen und Reasoning als auch für zukünftige Erweiterbarkeit. Darüber hinaus soll die Modularisierung zur

Wiederverwendbarkeit der Ontologien beitragen. Welche Strategie für die konkrete Umsetzung der Modularisierung verfolgt wird, hängt aber von zahlreichen Faktoren ab. So muss entschieden werden, in welchem Grad Wissensfragmente disjunkt sein sollen: Gibt es keine Überlappung von Aussagen, kann leichter ein konsistenter Gesamtzustand erreicht werden; je mehr Überlappung erlaubt wird, desto flexibler lässt sich Erweiterbarkeit realisieren (auch durch Dritte, von externen Quellen oder durch zukünftige Erweiterungen, die zum gegenwärtigen Zeitpunkt noch nicht vorgesehen sind). Es muss entschieden werden, ob Modulgrenzen anhand von semantischen oder strukturellen Elementen festgelegt werden sollen, oder einer Kombination davon. Semantische Abgrenzung von Modulen ist beispielsweise die Unterscheidung nach Abstraktionsebenen, wie in der Kategorisierung von Ontologie-Typen (siehe Abbildung 3.12). Eine strukturelle Abgrenzung erfolgt, wenn die Wissensbasis als Graph aufgefasst wird und dieser nach unterschiedlichen Kriterien in Teilgraphen zerlegt wird.

Es wird unterschieden zwischen der Zerlegung von Ontologien in Module (*Ontology Partitioning* – [dSSS09], Seite 70), dem Zusammenfügen von Modulen in eine neue Ontologie, ohne die ursprünglichen Module zu verändern (*Ontology Merging* – [ES13], Seite 39) und der Abbildung von Modulen aufeinander, ohne dabei eine neue Ontologie zu erzeugen (*Ontology Matching*, gelegentlich auch als *Ontology Mapping* bezeichnet, ebd.). Der Fall beim Ontology Matching, dass beide Ontologien in einer Weise weiterentwickelt oder angepasst werden, die explizit das Matching unterstützt, wird darüber hinaus als *Ontology Reconciliation* bezeichnet. Merging kann ebenfalls als Spezialfall von Matching betrachtet werden, bei dem die resultierende Ontologie zusätzlich die Anforderung erfüllen muss, dass sie konsistent ist und das Ergebnis eines Reasonings mindestens die Vereinigung der Reasoning-Ergebnisse der beiden Ursprungsontologien beinhaltet.

Es gibt zahlreiche Techniken zum Ontology Matching, Übersichten hierzu geben Shvaiko und Euzenat in [SE12] und Otero-Cerdeira et al. in [ORG15]. Die Klassifikation von Techniken zum Ontology Matching von Euzenat und Shvaiko [ES13] zeigt Abbildung 3.16. Jede Technik hat das Ziel, ein *Alignment* zwischen zwei oder mehr Ontologien zu finden, das aus einer Menge von *Korrespondenzen* besteht. Dabei können Korrespondenzen sowohl manuell erstellt werden (z.B. zwischen einer Domänenontologie und einer Upper Ontology), als auch durch Heuristiken (z.B. anhand der lexikalischen Ähnlichkeit der Bezeichner von Klassen oder Relationen).

Zusammen mit der Technik zum Matching muss auch entschieden werden, in welcher Form das Alignment ausgedrückt werden soll. Da diese Entscheidung vom Format abhängt, in dem die Ontologien beschrieben sind, kann dieser Schritt im Engineering Prozess erst in der Implementierungsphase durchgeführt werden. Alignments können auf unterschiedlichen Abstraktionsebenen ausgedrückt werden: Einerseits kann der dem Ontologieformat zugrunde liegende Formalismus angepasst werden. Dieser Weg wird beispielsweise mit einer Erweiterung von Descriptions Logics zu sogenannten Distributed Description Logics von Serafini und Tamilin in [ST09] beschrieben. Der Nachteil ist, dass für eine Anwendung in der Praxis zunächst alle Werkzeuge und Implementierungen entsprechend nachgezogen werden müssen. Andererseits

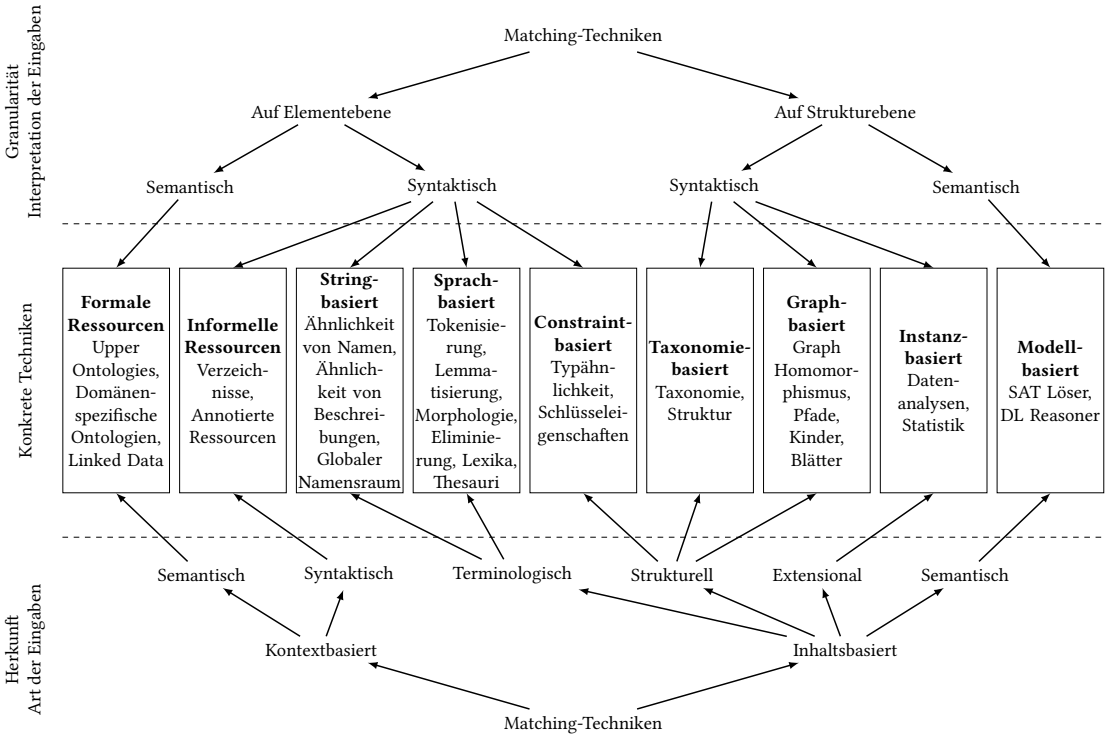

Abbildung 3.16: Klassifikation von Techniken zum Ontology Matching nach Euzenat und Shvaiko ([ES13], Seite 77)

können die bestehenden Mittel des Ontologieformats genutzt werden. OWL bietet zwar keine dedizierten Konstrukte zur Beschreibung eines Alignments, die `import` Anweisung kann aber verwendet werden, um alle Konstrukte aus einer weiteren Ontologie zu importieren; hierbei werden also alle Aussagen vereinigt, effektiv ist dies eine Ontology-Merging-Operation. Das Alignment zwischen zwei Ontologien O_1 und O_2 wird dann in einer dritten Ontologie beschrieben, die O_1 und O_2 importiert. Die Arbeit von Adamou et al. [ACGP13] verallgemeinert und formalisiert den Verband einzelner Ontologien, der hier als *Ontology Network* bezeichnet wird. Das vorgestellte Konzept eines *Virtual Ontology Network* formalisiert die Begriffe und Vorgehensweisen zur Anpassung der Gesamtmenge der Axiome (also dem Zufügen, Verändern und Entfernen von Axiomen) zur Erstellung neuer Ontology Networks. Dabei legen die Autoren Wert darauf, dass die Ontologie-Sprache selbst nicht angepasst werden muss.

Die dritte Möglichkeit ist die Einführung eines neuen Formats, einer neuen Sprache oder einer neuen Implementierung mit dem Ziel der Beschreibung des Alignments. Ein bekannter Vertreter hierfür ist die Arbeit von Scharffe et al. [SD05, SE07], die sowohl eine XML-basierte Beschreibungssprache als auch eine darauf basierende, als *Alignment API* bezeichnete Imple-

mentierung entwickelten. Insbesondere wenn Alignments nicht manuell, sondern mittels Heuristiken erzeugt werden (siehe Abbildung 3.16), gewinnt diese Möglichkeit an Bedeutung, da nur so der Grad der Übereinstimmung einer Korrespondenz ausgedrückt werden kann, mit anderen Worten, wie wahrscheinlich zwei Elemente über die errechnete Korrespondenz in Relation stehen. In OWL und OWL-Reasonern gibt es abgesehen von Annotationen, die die Semantik nicht beeinflussen, keine Möglichkeit, eine solche Wahrscheinlichkeit auszudrücken.

Ergänzend soll erwähnt werden, dass auch auf dem Gebiet der Ontologie-basierten Informationsintegration, das kein direkter Teil des Ontology Engineering ist, zahlreiche Fragen untersucht wurden, die den Einsatz von Ontologien in der Praxis stark beeinflussen. Sobald Ontologien nicht mehr zum Selbstzweck entwickelt werden (beispielsweise als Referenzmodell oder zur Wissenserfassung) sondern Teil einer umfassenderen Softwarelösung sind, kommen zusätzlich zu den bisher besprochenen Ontology-Engineering-Themen noch weitere hinzu. Ein wesentlicher Punkt ist die Abbildung zwischen Ontologien und anderen Datenquellen. Eine frühe Übersicht zu Arbeiten hierzu geben Wache et al. in [WVV+01]. Einführungen zu Lösungsmöglichkeiten geben Cruz und Xiao [CX05] sowie Semeraro et al. [SBB+13, Len13]. Speziell zu Techniken und Abbildungssprachen, die Ontologien mit RDBMSen verbinden, geben Hert et al [HRG11] eine Übersicht, Spanos et al. geben in [SSM12] die Ergebnisses eines umfangreichen Surveys zum Einsatz von RDBMSen *im* Semantic Web an.

3.6.4 Best Practices

Die meisten Ontology Engineering Methodiken, die in Abschnitt 3.6.2 vorgestellt wurden, konzentrieren sich vornehmlich auf den abstrakten Entwicklungsprozess bzw. den zugehörigen administrativen Prozess. Best Practices, wie sie für das Software Engineering etabliert sind, sind rar und beziehen sich, soweit vorhanden, zumeist auf eine technologieagnostische Ebene. So stammen die häufig zitierten Grundprinzipien für das Design von Ontologien von Gruber [Gru03] aus einem Kontext, in dem sich noch keine Standardsprache zur Beschreibung von Ontologien durchgesetzt hatte, trotzdem haben sie kaum an Bedeutung verloren ([Gom04], Seite 38ff, [Cal06], Seite 8ff). Die fünf Prinzipien lauten:

1. Klarheit (*Clarity*): Die beabsichtigte Meinung sollte durch die Terminologie der Ontologie klar und objektiv beschrieben werden. Formalisierungen sollten durch Beschreibungen in natürlicher Sprache ergänzt werden.
2. Minimale Voreingenommenheit für die Kodierung (*Minimal encoding bias*): Die Konzeptualisierung sollte nicht von einer bestimmten Kodierung auf Symbol-Ebene abhängen, da im Prinzip unterschiedliche Technologien für die Implementierung einsetzbar sein sollen.
3. Erweiterbarkeit (*Extendibility*): Neue Ausdrücke sollen zugefügt werden können, ohne bestehende anpassen zu müssen.

4. Kohärenz (*Coherence*): Aussagen, die durch Inferenz aus der Ontologie hergeleitet werden können, dürfen weder im Widerspruch zu anderen, explizit angegebenen Aussagen noch zu informell beschriebenen (beispielsweise Beispielen in der Dokumentation) stehen.

5. Minimale ontologische Festlegung (*Minimal ontological commitments*): Formalisierungen sollen auf der schwächsten Theorie, also dem „kleinsten gemeinsamen Nenner" basieren. So soll beispielsweise keine Festlegung auf ein bestimmtes Format für Datumsangaben oder Währungen erfolgen, da diese nicht in allen Systemen gleich gehandhabt werden.

Werden Best Practices zur konkreten Umsetzung von Ontologien gesucht, also nach der Auswahl einer Technologie zur Formalisierung, kann festgestellt werden, dass die Prinzipien der minimalen Voreingenommenheit und der minimalen ontologischen Festlegung hierbei nicht mehr berücksichtigt werden dürfen, sie stünden sonst im direkten Widerspruch zu den gesuchten Vorgehensweisen.

Ein strukturiertes Vorgehen für den Aufbau einer in sich schlüssigen bzw. für die Bereinigung einer Taxonomie stellen Guarino und Welty mit *OntoClean* in einer Reihe von Veröffentlichungen vor [GW00c, GW00b, GW00a]. Obwohl OntoClean als Methodik bezeichnet wird, handelt es sich nicht um eine generelle Ontology Engineering Methodik, sondern eine Unterstützung bei der Konzeptualisierung. Die Grundidee besteht aus einer Reihe von Meta-Eigenschaften, die den Konzepten der Taxonomie zugeordnet werden, und einer zugehörigen Menge von Einschränkungen, die sich daraus ergeben. So wird die Meta-Eigenschaft der *rigidity* (übersetzbar als „Festigkeit") definiert, die eine Eigenschaft oder eine Klasse so auszeichnet, dass sie sich während der Lebensdauer der ausgezeichneten Entität nicht ändern kann. So ist beispielsweise *Mensch* rigid. Kann sich die Eigenschaft für manche Individuen ändern und für andere nicht, so ist sie *semi-rigid* (Beispiel: *istRot*). Wenn sich die Eigenschaft bei jedem Individuum jederzeit ändern kann, wird sie als *anti-rigid* bezeichnet, ein Beispiel hierfür ist *Student*. Eine zu dieser Meta-Eigenschaft gehörige Einschränkung ist, dass keine als rigid gekennzeichnete Klasse Unterklasse einer als anti-rigid gekennzeichneten sein kann. Weitere Meta-Eigenschaften sind *identity*, *unity* und *dependence*, alle jeweils durch zugehörige Einschränkungen ergänzt. Werden die Schritte der Methodik befolgt, so werden zunächst alle Konzepte mit entsprechenden Meta-Eigenschaften annotiert, anschließend wird die Taxonomie anhand der Einschränkungen „gesäubert", Details dazu finden sich in [GW09].

Die Entwicklung von Entwurfsmustern, einem in der Softwaretechnik bereits seit geraumer Zeit etablierten Konzept, hat im Ontology Engineering erst nach 2005 an Bedeutung gewonnen. Erwähnenswert hierzu sind insbesondere die Arbeiten von Presutti et al. [PDGB09, PBDG10, DBG+10], die im Rahmen des NeOn-Projekts entstanden (siehe Abschnitt 3.6.2). Hier wurde erstmals eingehend untersucht, in welcher Form Muster entworfen werden können, welche Meta-Daten für eine sinnvolle Weiterverwendung erfasst werden müssen und wie sich der Einsatz von Entwurfsmustern im Ontology Engineering Prozess widerspiegelt. Entwurfsmuster für Ontologien werden hier in sechs Typen unterteilt [GP09]:

- Strukturelle Muster (*Structural Patterns*), die die grundlegende Struktur der Ontologie beeinflussen sollen;
- Umformungs- und Abbildungsmuster (*Correspondence Patterns*, siehe hierzu auch die Arbeit von Scharffe [Sch09]), die entweder Alignments zwischen zwei Ontologien oder die Umformung eines Ausgangsmodells (das eine Ontologie sein kann aber nicht muss) in eine (andere) Ontologie beschreiben;
- Inhaltliche Muster (*Content Patterns*), die Vorgaben zu domänenspezifischen Modellierungsfragestellungen machen;
- Reasoningmuster (*Reasoning Patterns*), die einzelne Aspekte beschreiben, die von einem Reasoner ausgewertet werden sollen, wie z.B. Klassifizierung oder Vererbung;
- Präsentationsmuster (*Presentation Patterns*), die die bessere Benutzbarkeit und Lesbarkeit der Ontologie für den Nutzer sicherstellen sollen;
- Linguistische Strukturen (*Lexico-Syntactic Patterns*), die strukturelle und inhaltliche Muster mit natürlichsprachlichen Sätzen in Verbindung bringen können, um die Semantik von Textfragmenten zu beschreiben.

Obwohl die Entwicklung bzw. Extraktion wiederkehrender Muster und ihre Einordnung in klar definierte Kategorien für das Ontology Engineering einen großen Schritt in Richtung einer ingenieurmäßigen Disziplin mit wiederholbarem Vorgehen und nachvollziehbaren Ergebnissen bedeutet, so haben sich bisher noch keine Entwurfsmuster für Ontologien herausgebildet, deren Grad der allgemeinen Anwendbarkeit und Verbreitung mit dem der etablierten Entwurfsmuster in der Softwaretechnik zu vergleichen wäre.

In der Literatur gibt es generell wenige Arbeiten zu spezifischen Hinweisen für die *Implementierung* einer Ontologie. Der Leitfaden von Noy und McGuinness zum Modellierungswerkzeug Protégé-2000 [NM01] bezieht sich zwar in erster Linie auf Frame-basierte Modelle, enthält aber auch viele Hinweise, die für Beschreibungslogik-basierte Ontologien gültig sind, beispielsweise die Aussage, dass es prinzipiell nicht immer nur einen richtigen Weg zur Modellierung einer Ontologie gibt, oder zur Frage, wann ein Konzept als Klasse und wann es als Instanz modelliert werden sollte. Weiterhin nennenswert sind die Arbeiten von Poveda et al. [PSG10a, PSG10b, PSG15], die eine Menge von konkreten Vorgaben definieren, die bei der Entwicklung einer Ontologie beachtet werden sollten. Diejenigen Vorgaben, bei denen dies möglich ist, können durch den zugehörigen *Ontology Pitfall Scanner*, der in Form eines Webservice bereit gestellt wird [OOP14], automatisiert überprüft werden. Es wird hier zwischen unterschiedlichen Schweregraden unterschieden, so wird die Verletzung bestimmter Vorgaben als kritisch eingestuft, bei anderen wird gewarnt.

Werden die Vorgehensweisen von einer der Ontology Engineering Methodiken befolgt, sollte nach dem Entwurf und der Umsetzung einer Ontologie eine Überprüfung ihrer Qualität stattfinden, um gegebenenfalls in der nächsten Iteration Probleme zu beheben. Hier stellt sich

die Frage, wie die Qualität gemessen werden kann. Unter dem Begriff der *Ontology Quality* bzw. *Ontology Quality Assurance* können Arbeiten zusammengefasst werden, die sich mit der Entwicklung von Verfahren zur Messbarkeit der Qualität von Ontologien beschäftigen. Ein wesentliches Ergebnis der Studie von Rogers [Rog06] ist, dass in den allermeisten Fällen nicht ein einzelnes, objektives Qualitätsmaß für eine Ontologie bestimmbar ist: Werden alle zum Entwicklungszeitpunkt relevanten Erkenntnisse zur korrekten Strukturierung (z.B. bestimmten philosophischen Schulen folgend) und der gesamte als „korrekt“ betrachtete Inhalt mit aufgenommen, so kann dies zu übermäßig komplexen und nicht mehr sinnvoll nutzbaren Ontologien führen. Metriken zur Qualitätsmessung müssen demnach abhängig von den Gegebenheiten des jeweiligen Anwendungsfalls festgelegt werden. Fang und Evermann stellen in [FE07] eine Methode zur näherungsweisen Bestimmung des Maßes an Überschneidung der Aussagen einer Ontologie mit der durch Testpersonen wahrgenommenen Realität vor, also wie gut das Modell der Ontologie den persönlichen Erkenntnissen eines Betrachters entspricht. Hierzu werden als wahr oder als unwahr bekannte Aussagen zufällig durchmischt präsentiert, die von Testpersonen bewertet werden. Die Bewertungsergebnisse können als Näherung einer Qualitätsmetrik aufgefasst werden. Die Autoren merken selbst an, dass die Ergebnisse von Anzahl und Vorkenntnissen der Testpersonen abhängig sind.

In der Arbeit von Gangemi et al. [GCCL06] wird die Validierung einer Ontologie bezüglich vorher definierter Maße zur Struktur, Funktion und dem angedachten Anwendungszweck in Form von Diagnostik-Aufgaben beschrieben. Jede solche Aufgabe hat eine Menge von Eigenschaften (z.B. zu überprüfende Wertebereiche). Die Menge von Diagnostik-Aufgaben zur Validierung einer Ontologie müssen als Teil ihres Entwicklungsprozesses definiert und anschließend überprüft werden, sie sind also nicht allgemeingültig.

Duque-Ramos et al. [DFSA11] passen das in ISO 25000 [ISO14] festgelegte Verfahren aus dem Software-Engineering an, um die Qualität von Ontologien zu beurteilen. Dazu definieren sie 14 Metriken und eine Abbildung der Wertebereiche, die sich aus der Berechnung der Metriken ergeben, auf eine der im ISO-Standard festgelegten Qualitätsstufen 1 bis 5. Während das Verfahren klar beschrieben ist und prinzipiell ohne manuelle Anpassung auf beliebigen Ontologien anwendbar ist, muss die Bedeutung der Metriken bei einer Anwendung verstanden werden: So werden beispielsweise die mittlere Anzahl von Rollen einer Klasse und das Verhältnis von konkreten zu abstrakten Rollen erfasst. Wie sinnvoll die Festlegung von Ober- oder Untergrenzen für solche Metriken ist, ist allerdings stark abhängig davon, für welchen Einsatz die Ontologie konzipiert ist und welcher Kategorie sie zugeordnet wird.

Teil II

Entwurf

4 Verwandte Arbeiten

Dieser Abschnitt stellt eine Übersicht zu den wichtigsten Arbeiten der Literatur vor, die sich mit einer Fragestellung beschäftigen, die eine Überschneidung mit den Zielen dieser Arbeit hat. Dabei werden die Arbeiten hier zunächst nur kategorisiert und vorgestellt. Ein inhaltlicher Abgleich findet in Kapitel 5 in Abschnitt 4.4 statt.

Wie Ontologien generell im IT-Management eingesetzt werden können, sowohl für die Entwicklung domänenspezifischer Informationsmodelle als auch für Laufzeitsysteme (beispielsweise im IT Operations Management), wird in Abschnitt 4.1 zusammengefasst. Abschnitt 4.2 untersucht Arbeiten, die sich mit der Formalisierung von IT-Governance-Modellen im Allgemeinen und COBIT im Speziellen befassen; wobei hier nicht notwendigerweise Ontologien zum Einsatz kommen müssen. Schließlich stellt Abschnitt 4.3 Arbeiten vor, in denen Ontologiebasierte Ansätze für den Einsatz im IT Service Management, dem IT-Governance und verwandten Gebieten entwickelt werden.

4.1 Ontologien im IT-Management

Es lassen sich zwei grobe Kategorien von Forschung zu Ontologien im IT Management definieren: Die erste Kategorie fokussiert sich auf den Einsatz im Kontext technischer Systeme: Hierbei werden unterschiedliche Informationsmodelle, Schnittstellen und offene und herstellerspezifische Werkzeuge überbrückt, um konkrete, oft direkt messbare Probleme zu lösen. Dies kann die bessere Leistung oder Resilienz eines Systems beinhalten, einen erhöhten Grad von Automatisierung oder auch die bessere Nachvollziehbarkeit, beispielsweise zur Fehleranalyse. Ontologien übernehmen hier die Rolle eines Vermittler-Modells oder werden zur formalen Repräsentation von Management-Wissen eingesetzt, das Software zum Selbstmanagement als Informationsbasis dient.

Die zweite Kategorie beinhaltet die Erfassung von Wissen, das auch außerhalb einer direkten technischen Anwendung eingesetzt werden soll. Hierbei dienen Ontologien zwar ebenfalls als Referenzmodell, die Verbesserung von Eigenschaften eines Laufzeitsystems steht aber nicht im Vordergrund. Das angestrebte Ziel ist hier alles, das sich unter dem Dachbegriff des *Business-IT-Alignments* zusammenfassen lässt. Der Begriff selbst kann zwar hinterfragt werden, wie Abschnitt 2.1.3 beschreibt, die Fragestellungen der hier angesiedelten Arbeiten sind aber durchaus relevant.

Bekannt in der ersten Kategorie sind die Arbeiten von De Vergara, Guerrero et al. [DVB04, DGVB09, GVDB05, GVD+06]. Hier wurde in einer Reihe von Projekten zunächst die Sinnhaftigkeit, dann die Machbarkeit und schließlich die exemplarische Anwendung von Ontologien und verwandten Technologien auf das Netzwerkmanagement untersucht. Strukturinformationen und Verhaltensregeln sollten aus unterschiedlichen Quellmodellen über Adapter in eine Ontologie zusammengefasst werden (u.a. aus SNMP (Simple Network Management Protocol) bzw. dem Modell SMI (Structure of Management Information), CMIP (Common Management Information Protocol) bzw. dem Modell GDMO (Guidelines for the Definition of Managed Objects) und WBEM (Web-Based Enterprise Management) bzw. dem Modellformat MOF (Managed Object Format). Für die Formalisierung der Ontologie wurde hier OWL als Ontologiesprache gewählt und durch Regeln im SWRL (Semantic Web Rule Language, [HPB+04]) Format erweitert. Die Anwendungsfälle umfassen Network Security and Policy Management sowie Network Monitoring. Die Autoren kommen zu dem Schluss, dass sich OWL für die Integration der Informationsmodelle sehr gut eignet, aber dass viele Eigenschaften von SWRL seinen praktischen Einsatz behindern, beispielsweise das Fehlen eines Negations-Operators oder die streng monotone Inferenz.

In [TSK10, TSK11, TMK12] entwickelt der Autor der vorliegenden Arbeit ein System für das automatisierte IT-Management, in dem ebenfalls OWL-Ontologien für die Beschreibung des Informationsmodells eingesetzt werden. Als Core Ontology wird dazu CIM (Common Information Model), ein umfassendes objektorientiertes Modell der DMTF (Distributed Management Task Force) zur Beschreibung von Elementen in IT-Umgebungen in das OWL-Format übersetzt. Ein prototypisches Laufzeitsystem ist darüber hinaus in der Lage, Daten von einem CIM-Agenten zur Laufzeit in entsprechende OWL-Instanzen zu abzubilden. Durch die Nutzung von CIM als Modell ist das System auf den Einsatz im Kontext entsprechender technischer Systeme begrenzt. Eine Anwendung von Ontologie-basiertem IT-Management im Bereich des Storage Managements wird ebenfalls vom Autor in [TS15] beschrieben. Dabei wird eine Ontologie für ein System zur Storage-Virtualisierung entworfen und wird zusammen mit einem entsprechenden Laufzeitsystem zur Abfrage und Steuerung des verwalteten Speichersystems eingesetzt.

In den Arbeiten von Strassner et al. [SDR+07, SvJP09, SvOD09] zum Policy-basierten und autonomen Netzwerkmanagement wurden zunächst Ontologien als ein unterstützendes Element in einer ansonsten hauptsächlich auf UML-Modellen basierenden Umgebung eingesetzt. In späteren Arbeiten gewinnen Ontologien an Wichtigkeit, da die flexible Erweiterbarkeit entwickelter Taxonomien für mehrere Teilfragestellungen genutzt wird. Es werden hauptsächlich Konzepte und Begriffe aus dem Ontology-Engineerings verwendet, allerdings keine spezifischen Technologien. So wird eine „Upper Ontology“ vorgestellt ([SvOD09], Seite 271), ein Informationsmodell das abstrakte Entitäten zum räumlichen, Verwaltungs- oder Aktivitätskontext enthält; dieses wird aber nicht mittels OWL oder anderer Ontologie-Sprachen beschrieben, sondern mithilfe eines eigenen objektorientierten Modells.

Die explizite Anwendung von Ontology-Mapping-Techniken (wie in Abschnitt 3.6.3 erläutert)

wird von Wong et al. [WRPS05] beschrieben; Strassner ist hier ebenfalls Koautor. Hier werden Konfigurationen von Routern verschiedener Hersteller unter Verwendung von Ontologien aufeinander abgebildet. In [Cv08] verwenden Carroll und van der Meer Ontologien im Kontext einer Mobilfunkanwendung, um Informationsmodelle von Servicebeschreibungen mit Modellen von Benutzerprofilen zu verbinden; in [LvD+10] stellen Latré et al. eine Anwendung zur automatischen Verwaltung von Multimedia-Netzen vor (z.B. zum Video-Streaming), in der Ontologien eingesetzt werden, um Filterregeln zu generieren. Im *Aesop*-Ansatz von O'Sullivan und Fallon [OWL07, FO14] werden automatisch Netzeinstellungen optimiert, um Dienstgüteeigenschaften für Endnutzer zu verbessern. Hierfür wird eine separate Ontologie zur Formalisierung des Service Management Modells eingeführt [FO12], die mittels RDF beschrieben wird.

Ein Beispiel für den Einsatz von Ontologien in einer nicht unmittelbar technisch orientierten Fragestellung des IT-Management ist die Arbeit von Veres et al. [VSC+10]: Hier wird ein zuvor entwickeltes Rahmenwerk zum Abgleich der IT mit der Geschäftsstrategie aus Sicht des Anforderungs-Engineerings in Form einer OWL-Ontologie formalisiert. Das Schema des sogenannten Business Motivation Model kann dann mittels SPARQL abgefragt werden, um zu Antworten auf Fragen zu den Zusammenhängen der unterschiedlichen Aspekte zu gelangen. Ein Zufügen von Instanzdaten zur Ontologie, um beispielsweise auch konkrete formulierte Geschäftsziele in die Beantwortung der Fragen mit aufzunehmen, wird bei den Autoren im Ausblick erwähnt.

4.2 Formalisierung von IT-Governance und COBIT

Goeken und Alter stellen in [GA09] fest, dass trotz des hohen Bedarfs an Unterstützung für ein methodisches Vorgehen bei der IT-Governance überraschend wenig Forschung in dieser Richtung durchgeführt wird. Angesichts der Bedeutung von COBIT als Rahmenwerk in der Praxis wird als Grundlage für die weitere *systematische* Entwicklung von Verfahren und Werkzeugen hier ein klar definiertes Meta-Modell für COBIT angesehen. Die Autoren erklären zunächst die für das Vorgehen relevanten Unterschiede zwischen den Abstraktionsebenen M0 bis M3 der Meta-Modellierung und stellen dann ein Meta-Modell für COBIT 4.1 vor. Das Modell beinhaltet 19 Konzepte und 11 Relationen und wird in Form eines Entity-Relationship-Diagramms (ER) präsentiert.

Ebenfalls mit der Entwicklung eines COBIT 4.1 Meta-Modells beschäftigen sich Souza Neto und Nunes Ferreira Neto in [SF13]. Sie verwenden andere Prinzipien zur Konstruktion eines Meta-Modells als Goeken und Alter, ihr resultierendes COBIT-Meta-Modell unterscheidet sich daher marginal. Das Meta-Modell ist mit 27 Konzepten umfangreicher, entfernt sich aber in einzelnen Aspekten etwas mehr von den Definitionen des offiziellen Rahmenwerks. Es ist ebenfalls als ER-Diagramm beschrieben.

Der Autor der vorliegenden Arbeit stellt in [Tex14, TG15] eine Herangehensweise für die Formalisierung von COBIT vor, bei der für COBIT eine OWL-Ontologie entwickelt wird, die COBITs Meta-Modell und Modell repräsentiert. Hierbei werden demnach die Struktur sowie die Menge der spezifizierten Entitäten beschrieben, die der Struktur entsprechen. Dabei werden auch Konstrukte berücksichtigt, die in der COBIT-Spezifikation nur rein natürlichsprachlich beschrieben sind, insbesondere Metriken zur Messung des Erreichungsgrades von Geschäfts- oder Prozesszielen.

In [ABE+06] setzen Andersson et al. auf einer noch allgemeineren Ebene an und entwerfen ein generelles Business-Meta-Modell als Ontologie. Hierbei ergeben sich inhaltlich bedingt Überschneidungen COBIT, mit einer Gesamtmenge von 22 Konzepten kann das entwickelte Modell jedoch eher als domänenspezifische Core Ontology betrachtet werden. Das Modell wird hier in Form eines UML-Klassendiagramms angegeben.

4.3 Entwicklung von Ontologien für IT Service Management, IT-Governance und verwandte Gebiete

Um eine bessere Nutzung von IT-Ressourcen zu erreichen, stellen Ghedini Ralha und Gostinski in [GG08] ein Framework zum „Business-IT-Alignment“ vor. COBIT wird zwar erwähnt, aber fließt nicht in den Ansatz ein. Die Autoren bewerten allerdings die Wichtigkeit für den Einsatz von semantischer Modellierung als hoch ein: Für das notwendige bessere gegenseitige Verständnis der Geschäfts- und IT-Einheiten werden daher Ontologien eingesetzt für eine gemeinsam genutzte Terminologie, die die Voraussetzung für das Alignment ist. Nach einer Analyse von Methodiken zum Ontology Engineering und Ontology Alignment werden eine auf ITIL-Konzepten basierende Ontologie und eine Ontologie für eine spezifische Geschäftsdomäne entwickelt. Diese werden durch einen Reasoner validiert und dann mittels unterschiedlicher, auf statistischen Verfahren basierenden Alignment-Techniken aufeinander abgebildet.

Mehrere Arbeiten befassen sich mit der Entwicklung eines formalen Modells für IT Service Management oder Teilaspekte davon bzw. für ITIL. Die Motivationen hierfür sind unterschiedlich aber zum Teil überlappend. Baiôco et al. betrachten speziell das IT Service Configuration Management, die als Grundlage für ein Informationssystem genutzt werden kann, das aktuelle und akkurate Informationen über die Zusammenhänge von IT-Prozessen, eingesetzten Werkzeugen und Personen liefert. Die Autoren bauen auf der Grundlage der UFO Upper Ontology auf, die in Abschnitt 3.6.1 vorgestellt wurde. Die resultierende Ontologie wird in Form von Klassendiagrammen und einzelnen FOL-Aussagen beschrieben.

In der Arbeit von Valiente et al. [Val11, VVR11, VGS12] ist das Ziel die Berücksichtigung von ITSM-Prozessen direkt bei der (modellgetriebenen) Softwareentwicklung neuer Anwendungen; hier werden daher Ontologien als Brücke zwischen einem MDA-Prozess (Model Dri-

ven Architecture) und dem ITIL-basierten ITSM eingesetzt. Dazu entwickeln die Autoren unter anderem Ontologien, die Ausschnitte von ITIL sowie der Prozessbeschreibungssprache BPMN (Business Process Model Notation, [omg11, Koc11]) enthalten. Mit der Entwicklung von BPMN-Ontologien beschäftigen sich außerdem auch Natschläger in [Nat11] und Penicina in [Pen13].

Czarnecki und Orłowski stellen ebenfalls den Mehrwert einer Formalisierung von ITIL als Ontologie vor und unternehmen erste Schritte in dieser Richtung [CS13, CO15], indem die Anwendbarkeit von Ontology Engineering Methodiken wie Methontology untersucht werden (siehe auch Abbildung 3.14), konkrete Fragmente einer Ontologie werden aber nicht vorgestellt. Die ITSMO (IT Service Management Ontology) von Fagnoni [Fag14] basiert direkt auf den Konzepten von ITIL, erhebt aber nicht den Anspruch eine direkte ITIL-Formalisierung zu sein.

Getrieben von einem komplexen Anwendungsfall, der Integration von ITSM-Prozessen über Geschäftsstellen bzw. Tochterfirmen nach einer Fusion hinweg, stellen Knittl und Schmitz in [KS13] ein Ontologie-basiertes Vorgehen hierfür vor. Dazu werden Ontologien sowohl für Teile von ITIL als auch für eine entsprechende CMDB (Configuration Management Database) entwickelt.

Zur Entwicklung von Ontologien zur IT-Governance finden sich in der Literatur bedeutend weniger Arbeiten. Wong et al. stellen in [WYRP08] bereits fest, dass für eine semantische Interoperabilität zwischen der IT, für Compliance Management und auch für semantische Datenintegration im Unternehmen eine umfassende IT-Governance-Ontologie nötig wäre. Zur Umsetzung halbautomatischer Compliance-Überprüfungen setzen sie hierzu kleine Ausschnitte von COBIT in Form einer Ontologie um, die für den untersuchten Anwendungsfall benötigt werden. Dos Santos et al. sehen ebenfalls die Notwendigkeit einer IT-Governance-Ontologie, setzen hierfür aber auf ITIL [ddTF10].

Einen ersten Schritt in Richtung Formalisierung von COBIT als Ontologie nehmen Niemann et al. in [NHSS11] vor. Grundlage der Arbeit ist COBIT 4.1; es wird mittels NLP-Verfahren (Natural Language Processing) versucht, ein Schema aus dem Spezifikationstext zu extrahieren. Durch SPARQL-Abfragen wird eine stichprobenartige Validierung vorgenommen. Eine Übersicht oder die aus dem Verfahren resultierende Ontologie werden in der Arbeit nicht vorgestellt.

Werden nicht ITSM oder IT-Governance als abgeschlossene Gebiete betrachtet, sondern weiter gefasste Untersuchungen zur Formalisierung von Zusammenhängen im Unternehmen als Ontologie vorgenommen, so finden sich hierzu ebenfalls Arbeiten, die hier berücksichtigt werden sollen: Bereits im Jahr 2000 stellen Kietz et al. in [KMV00] eine Methode vor, um halbautomatische Ontology Acquisition innerhalb eines Unternehmensnetzes zu betreiben.

Zur Unterstützung des Prozessmanagement durch Ontologien stellen Hepp und Dumitru in [HR07] ein Framework vor. Hierzu werden die Vision und Anforderungen des sogenannten

Semantic Business Process Management erläutert, sowie Ontology Engineering Methodiken und existierende Grundlagentechnologien untersucht, die für eine entsprechende Umsetzung geeignet sind. Nötige Teilontologien und die grundsätzliche Architektur werden umrissen.

Eine umfangreichere und ausgereiftere Sammlung von Arbeiten hierzu wurden unter dem Namen *Corporate Semantic Web* von der gleichnamigen Arbeitsgruppe um Paschke vorgestellt [BPP10, PCH+10, PCH+11, PCH+13]. Das Ziel beinhaltet hier eine Reihe von Einzelteilen: Intelligente semantische Suche beliebiger Strukturen im Unternehmen, Wissenstransfer und -weitergabe und fortgeschrittenes agiles ITSM und Geschäftsprozessmanagement. Auf dem Weg zu diesem Ziel untersuchen die Autoren einzelne Fragestellungen wie die Anwendung von Ontology Engineering im Geschäftsumfeld (*Corporate Ontology Engineering*), Modularisierung und Versionierung von Ontologien und Voraussetzungen für die kollaborative Weiterentwicklung von Ontologien.

4.4 Einordnung in den Stand der Forschung

Die Arbeiten, die in den Abschnitten 4.1 bis 4.3 vorgestellt wurden, zeigen die Fortschritte auf, die in den einzelnen Problembereichen erreicht wurden, die dieser Arbeit zugrunde liegen.

Dieser Abschnitt soll untersuchen, inwieweit existierende Teillösungen für einzelne Problembereiche in dieser Arbeit direkt oder nach nötigen Anpassungen anwendbar sind und welche Aspekte neu konzipiert werden müssen. Da die untersuchten Ansätze der Literatur bis auf wenige Ausnahmen andere Zielsetzungen haben als die in dieser Arbeit angestrebten, müssen in jedem Fall die Rahmenbedingungen verglichen werden. Die untersuchten Themenbereiche lassen sich sinnvoll noch weiter unterscheiden in die sechs Bereiche Governance, COBIT, Automatisierung, Ontologien, ITIL/ITSM und Business-IT-Alignment.

Unter dem Bereich Automatisierung werden diejenigen Arbeiten zusammengefasst, die ein Laufzeitsystem zum automatisierten Management zumindest als Teilaspekt berücksichtigen. Der Einsatz von Ontologien zur Wissensmodellierung ist ein separater Bereich, der dabei nicht nur Entwurf und Entwicklung von Ontologien als abgeschlossene Informationsfragmente umfasst, sondern auch die Verbesserung von hierzu genutzten Methodiken. Der Bereich ITIL/ITSM fasst Arbeiten zusammen, die sich mit IT Service Management allgemein oder speziell mit ITIL befassen, da beides nicht zum Kern dieser Arbeit zählt. Schließlich beinhaltet der Bereich Business-IT-Alignment diejenigen Arbeiten, die hierbei eine inhaltliche Unterscheidung zur Governance treffen, bzw. sich nicht in vollem Umfang auf die in Kapitel 2.1.2 beschriebenen Inhalte und Aufgaben von IT-Governance beziehen.

Die Bereiche und ihre Abdeckung durch repräsentative, vorgestellte Arbeiten sind in Tabelle 4.1 zusammengefasst. Die Spalten der Tabelle beschränken sich dabei auf die wesentlichen Arbei-

Fokus	Goeken 2009 [GA09]	Souza Neto 2013 [SF13]	Niemann 2011 [NHSS11]	De Vergara 2009 [DGVB09]	Strassner 2009 [SvOD09]	Baiôco 2009 [BCCG09]	Valiente 2011 [Val11]	Czarnecki 2015 [CO15]	dos Santos 2010 [ddTF10]	Hepp 2007 [HR07]	Rohloff 2008 [Roh08]	González 2011 [VSC+10]	Blumauer 2010 [BPP10]	Wong 2008 [WYRP08]	ISACA/ITG [AdG+12a]
Governance*	●	●	●	●						●		●		●	●	●
COBIT*	●	●	●	●											(●)	●
Automatisierung*					●	●							(●)			
Ontologien*				●	●	●	●	●	●	●	●		●	●	●	
ITIL/ITSM							●	●	●	●	●	●				●
Business-IT-Align.												●	●	●	●	

Tabelle 4.1: Fokus von Ansätzen in der Literatur. Mit * gekennzeichnete Bereiche bilden den Fokus dieser Arbeit.

ten, in denen mehrere der untersuchten Bereiche behandelt werden. Der Fokus dieser Arbeit ist in der Tabelle durch eine Markierung mit * an den ersten vier Bereichen hervorgehoben.

Die Arbeiten von Goeken und Alter [GA09], Souza Neto und Nunes Ferreira Neto [SF13] und Niemann et al. [NHSS11] stellen die bisherigen wesentlichen Ansätze in der Literatur dar, das Rahmenwerk COBIT zu formalisieren. Alle drei Arbeiten stellen die Notwendigkeit eines formalen Modells für COBIT heraus, verfolgen aber unterschiedliche Ansätze zu dessen Umsetzung. In den ersten beiden dieser drei Ansätze werden Entity-Relationship-Diagramme (ER-Diagramme) entwickelt, die die Kernkonzepte und die Beziehungen zwischen ihnen beschreiben sollen. Die Ausdrucksmöglichkeiten von ER-Diagrammen können als Untermenge der Ausdrucksmöglichkeiten von UML betrachtet werden. Diese Art von Diagramm ist zwar zur generellen Erfassung von Konzepten und Kardinalitäten von Relationen geeignet, aber zur umfassenden Beschreibung von Zusammenhängen nicht ausreichend und daher für die Erfassung der Semantik ungeeignet. Cranefield untersuchte bereits 2001 die Nutzung von UML zur Beschreibung von Ontologien [Cra01] und setzte die Object Constraint Language (OCL, [omg14]) zur Festlegung von Klassenrestriktionen ein. Neben Cranefield kommen auch Roussey et al. [RPKC11] zu dem Schluss, dass UML nicht ausreicht, um die benötigten Details für ein Reasoning ausdrücken zu können.

Niemann et al. verfolgen aus diesem Grund das Ziel, eine Ontologie für COBIT zu entwickeln,

setzen dazu aber, wie in Abschnitt 4.3 beschrieben, NLP-Verfahren für die Extraktion von Konzepten aus dem Spezifikationstext ein.

In dieser Arbeit wird nach bestem Wissensstand der erste Versuch unternommen, ein vollständiges formales Modell der strukturellen Elemente des Rahmenwerks COBIT in Form einer Ontologie zu entwickeln, das nicht auf heuristischer Textanalyse basiert, sondern auf einer inhaltlichen Analyse. Die Ergebnisse der Arbeiten von Goeken und Alter sowie Souza Neto und Nunes Ferreira Neto können dabei allerdings nicht als Grundlage verwendet werden, da beide Arbeiten sich auf COBIT 4.1 beziehen, in dieser Arbeit aber die aktuelle Version 5 von COBIT betrachtet wird. Die in Abschnitt 2.2 dargelegte weitgehende Änderung und Umstrukturierung von COBIT zwischen den Versionen 4.1 und 5 macht den Aufbau auf den Ergebnissen unmöglich. Der Beitrag dieser Arbeit ist für diesen Aspekt eine komplette Neuentwicklung einer COBIT-Ontologie.

Der zweite wesentliche Lösungsbestandteil in dieser Arbeit ist die Automatisierung von IT-Systemen und die Verwendung von Ontologien zur Beschreibung von Systemmodellen in diesem Kontext. Wesentliche Beiträge hierzu leisteten die Gruppen um De Vergara, Guerrero et al. [DVB04, DGVB09, GVDB05, GVD+06] sowie Strassner et al. [SDR+07, SvJP09, SvOD09]. Auch vergangene Arbeiten des Autors dieser Arbeit beschäftigten sich mit dem Einsatz von Ontologien zum automatisierten IT-Management [TSK10, TSK11, TMK12, TS15]. Allen Arbeiten gemein ist ihre Einschränkung auf eine rein technische Sicht des IT-Managements: Die entwickelten bzw. behandelten Modelle fokussieren sich auf eine oder mehrere unterschiedliche technische Domänen wie das Policy-basierte Netzwerkmanagement oder Storage Management, es werden aber keine expliziten Verbindungen mit Ontologien aus nicht-technischen Bereichen berücksichtigt. Während die Arbeiten von De Vergara et al. den ersten umfassenden Ansatz zu Ontologie-basiertem IT-Management darstellen, werden Ontologien in den Arbeiten von Strassner et al. im Wesentlichen unterstützend eingesetzt.

Diese Arbeit setzt nach bestem Wissensstand erstmals Ontologie-basiertes IT-Management mit dem expliziten Ziel ein, die Integration beliebiger, auch nicht-technischer Domänen, zu realisieren und eine Automatisierung über Domänengrenzen hinweg zu ermöglichen. Dabei wird hier zudem, im Gegensatz zum Ansatz der Arbeiten von Strassner et al., eine auf den offenen Standards RDF und OWL basierende Formalisierung verwendet. Die COBIT-Ontologie wird für das Informationsmodell der Automatisierung als Core Ontology eingesetzt, d.h. als Grundlage der semantischen Verbindung von Konzepten unterschiedlicher Domänenontologien. Die Nutzung einer Core Ontology im Informationsmodell eines Automatisierungssystems, die direkt auf dem formalen Modell eines Standards oder De-Facto-Standards zum IT-Government basiert, ist dabei ebenfalls neu – in der Literatur finden sich hierzu keine vergleichbaren Arbeiten.

In den Arbeiten mehrerer Gruppen spielt die Entwicklung von Ontologien zur semantischen Verbindung von Modellen unterschiedlicher Domänen im IT-Management eine Rolle, bei denen nicht-technische Sichten zumindest berücksichtigt werden. Hierzu werden in den meis-

ten Fällen Ausschnitte von ITIL als Ontologie formalisiert. Arbeiten in dieser Kategorie sind die Arbeiten von Baiôco et al. [BCCG09], die die teilweise Abdeckung von IT Service Management und IT-Governance mit einer ITIL-Ontologie anstreben, die Arbeiten von Valiente [Val11, VVR11, VGS12], sowie die Arbeiten von Czarnecki et al. [CS13, CO15]. In keinem der Ansätze spielt Automatisierung oder die Entwicklung von Automatisierungsansätzen eine Rolle, der Fokus liegt stets auf konzeptuellen Modellen und der Erfassung der Semantik.

Obwohl geeignete konzeptuelle Modelle eine wichtige Voraussetzung für die Entwicklung und Weiterentwicklung von Automatisierungssystemen sind, in denen Ontologien als Kernmodell eingesetzt werden, werden hierfür wichtige Aspekte nicht betrachtet, wie beispielsweise die Auswirkung der Komplexität der verwendeten Ontologiesprache auf die Laufzeit eines Regelkreises. Eine erwähnenswerte Ausnahme für die Berücksichtigung von Automatisierungsaspekten ist die Arbeit von González-Conejero et al. [GMF11], in der ein Ontologie-basiertes System eingesetzt wird, um mittels Ontologien zu GRC (Governance, Risk Management, Compliance), also einer Untermenge der IT-Governance, eine Automatisierung von Compliance-Evaluierungs-Prozessen zu erzielen. Das Papier verbleibt allerdings auf einer hohen Abstraktionsebene und geht nicht auf konkrete Eigenschaften der Ontologien, ihrer Erstellung oder des zugehörigen Laufzeitsystems ein.

Der Beitrag dieser Arbeit im Vergleich zu Arbeiten der oben beschriebenen Kategorie ist daher nicht nur die Einbeziehung von Laufzeitaspekten bei der Entwicklung von Ontologien zur Überbrückung unterschiedlicher Domänen, sondern auch die Entwicklung eines Laufzeitsystems, das diese nutzen kann. Darüber hinaus stellt eine detailliert dargelegte Fallstudie zur Validierung dieses Systems und der entsprechenden Konzepte eine Neuerung dar, die in dieser Form in der bestehenden Literatur ebenfalls nicht zu finden ist.

Zusammenfassend lassen sich in der Literatur sowohl Ansätze zur Formalisierung von Domänenmodellen für Systeme zur Automatisierung im IT-Management finden, als auch die Feststellung der Notwendigkeit der Entwicklung formaler Modelle für die IT-Governance. Es existiert allerdings keine Arbeit, die Informationsmodelle in einem System zum automatisierten IT-Management semantisch durch ein umfassendes Kernmodell verbindet, das auf einem etablierten Rahmenwerk basiert: Existierende Automatisierungsansätze beschränken sich auf technische Sichtweisen, selbst wenn sie Ontologien oder verwandte Technologien zur Beschreibung ihrer Wissensmodelle einsetzen, und die existierenden Versuche zur Erstellung eines auch zur Automatisierung geeigneten formalen Modells zur Überbrückung von Sichtweisen sind als rudimentär einzustufen. In dieser Arbeit wird erstmals ein vollständiges formales Modell des Rahmenwerks COBIT in Form einer Ontologie entwickelt, sowie die Konzepte, beispielhafte Implementierung und Anwendung eines Laufzeitsystems, in dem modulare Wissensmodelle geladen und zum regelbasierten, automatisierten IT-Management eingesetzt werden.

5 Konzeption

5.1 Anforderungen

Die funktionalen Anforderungen, die sich in dieser Arbeit stellen, betreffen die zwei Kernbereiche der Formalisierung für das Rahmenwerk COBIT und der Entwicklung der Architektur eines Laufzeitsystems zum Ontologie-basierten automatisierten IT-Management. In den Erläuterungen der folgenden funktionalen und nichtfunktionalen Anforderungen wird daher ausdrücklich Bezug genommen auf die „COBIT-Ontologie“ und das „Laufzeitsystem“.

5.1.1 Funktionale Anforderungen

F1 *Formales Modell für das Rahmenwerk COBIT*
Für COBIT soll ein formales Modell in Form einer OWL-Ontologie entwickelt werden. Dabei sollen alle wesentlichen Elemente der Spezifikation, die sich nicht in erster Linie auf Handlungsvorschriften beziehen, berücksichtigt und, soweit möglich, umgesetzt werden. Die resultierende Ontologie soll also das implizit beschriebene Informationsmodell formalisieren; die Richtlinien zur Einführung von COBIT im Unternehmen müssen nicht berücksichtigt werden.

F2 *Nachvollziehbarkeit der COBIT-Ontologie*
Es muss in der COBIT-Ontologie nachvollziehbar sein, aus welchen Elementen (Kapiteln, Abbildungen oder Tabellen) bestimmte Aussagen hergeleitet werden, die die Grundlage von Modellelementen bilden, bzw. auf welche Dokumente Bezug genommen wird. Dies ist notwendig, um die Überprüfbarkeit auf Fehler und Vollständigkeit, die Möglichkeit zur nachträglichen Erweiterung sowie Weiterentwicklung entsprechend künftiger Versionen von COBIT zu gewährleisten.

F3 *Identifizierbarkeit von Elementen der COBIT-Ontologie*
Da eines der Kernziele der Ontologie der Einsatz als Referenzmodell ist, müssen alle Elemente der Ontologie, die Konzepte der COBIT-Spezifikation abbilden, einem klar definierten Namensschema folgen.

Ontologien, die die COBIT-Ontologie referenzieren, müssen auf Elemente Bezug nehmen können, selbst wenn diese in der ursprünglichen Spezifikation keine eindeutigen Bezeichner besitzen. Der Einsatz von Sprachkonzepten wie anonymen RDF-Knoten bzw. anonymen OWL-Individuen muss daher begründet werden und darf diese Anforderung nicht verletzen.

F4 *Konsistenz der COBIT-Ontologie*
Die Ontologie muss konsistent im Sinne der zugrunde liegenden Description Logic sein. Die Ontologie muss im Sinne der Open World Assumption erweiterbar sein, ohne dass solche Erweiterungen unmittelbar zu einer Inkonsistenz führen. Eine Ausnahme zur Erweiterungsfähigkeit unter Berücksichtigung der Open World Assumption sind Modellelemente, die in der COBIT-Spezifikation ausdrücklich geschlossene Mengen sind (beispielsweise die fest definierten 37 Prozesse).

F5 *Formalisierung von Metriken der COBIT-Ontologie*
In COBIT existiert das Konzept einer Metrik, mit denen der Erreichungsgrad von Geschäfts-, Prozess- und IT-Zielen messbar gemacht werden soll.
Dabei wird eine Menge von Beispielmetriken definiert, es ist aber im Einsatz von COBIT explizit vorgesehen, diese durch weitere, für den jeweiligen Einsatzzweck geeignete Metriken zu erweitern. In der spezifizierten Form sind Metriken als Text gegeben. Für den Einsatz der Ontologie als Referenzmodell in einem Automatisierungssystem müssen die Werte von Metriken allerdings maschinenverarbeitbar werden. Um Metriken aus externen Ontologien und Berechnungsvorschriften im Laufzeitsystem sinnvoll referenzieren zu können, reicht es nicht aus, diesen Aspekt erst auf der Implementierungsebene des verarbeitenden Systems zu behandeln. Anders gesagt, der Typ, den ein Metrikwert annehmen kann, darf nicht nur in der Implementierung des Laufzeitsystems beschrieben sein, sondern muss bereits in der Ontologie vorhanden sein, da er selbst Bezug auf weitere Ontologie-Elemente nehmen kann und weitere, nachträglich zugefügte Ontologien auch Bezug auf ihn nehmen können müssen. Es muss daher eine Möglichkeit geschaffen werden, Metriken innerhalb der Ontologie über eine reine Wiedergabe ihres beschreibenden Textes hinaus so zu spezifizieren, dass ihre Werte innerhalb eines Automatisierungssystems ausgewertet werden können.

F6 *Laufzeitsystem zur wissensbasierten Automatisierung*
Es muss ein Laufzeitsystem geschaffen werden, um dem MAPE-K-Prinzip entsprechend ein IT-System zu überwachen, Anpassungen des Systems zu planen und gemäß Automatisierungsregeln auf das zu verwaltende System einzuwirken. Es muss daher um eine Infrastruktur für die entsprechenden Komponenten verfügen: Informationsmodelle, Regelmengen und Softwarekomponenten für die Umsetzung der Aufgaben der einzelnen MAPE-K-Schritte.

F7 *Domänenübergreifende Wissensbasis des Laufzeitsystems*
Die Informationsmodelle, die die Wissensbasis ausmachen (die Knowledge-Komponente der MAPE-K-Architektur), müssen in einer Form spezifiziert werden, die unabhängig von konkreten Domänen ist, die nicht nur taxonomisches Wissen sondern auch die semantischen Eigenschaften von Zusammenhängen beschreiben kann und deren Auswertung (Validierung, Reasoning) entscheidbar ist. Die Informationsmodelle müssen das gesamte notwendige strukturelle Wissen beschreiben können, das für die Durchführung der Regelungsaufgaben des Laufzeitsystems notwendig sind, also Entitäten, Relationen, zusätzliche, die Semantik beschreibende Axiome und Regeln. Für die Regelung benötigte Daten, die erst zur Laufzeit verfügbar sind, müssen mit den entsprechenden strukturellen Informationen verküpft verwerden können.

F8 *Modulare Wissensbasis im Laufzeitsystem*
Für die nachträgliche Erweiterbarkeit der Wissensbasis um Informationsmodelle, Softwarekomponenten für den Zugriff auf weitere Schnittstellen des zu verwaltenden Systems, sowie Regelmengen muss das System eine geeignete Abstraktion bereitstellen. Diese muss ermöglichen, die Abhängigkeiten, die explizit und implizit zwischen Informationsmodellen, Softwarekomponenten und Regeln bestehen, so zu beschreiben, dass sie vom Laufzeitsystem überprüfbar sind und damit ein konsistenter Zustand der Wissensbasis erreicht werden kann. Relationen zwischen Elementen unterschiedlicher Informationsmodelle müssen ausdrückbar sein, ebenso muss die Abfrage von Zusammenhängen, die sich über mehrere Informationsmodelle erstrecken, möglich sein.

5.1.2 Nicht-funktionale Anforderungen

N1 *Orientierung an bestehenden Standards*
Sowohl die COBIT-Ontologie (siehe funktionale Anforderungen **F1** und **F2**) als auch weitere Wissensmodelle, die mit dem Laufzeitsystem verarbeitbar sein sollen (siehe insbesondere funktionale Anforderung **F7**), sollen unter Verwendung geeigneter offener Standards umgesetzt werden. Dies ist eine notwendige Voraussetzung für die sinnvolle nachträgliche Erweiterung um zusätzliche Domänenmodelle, da, unabhängig von der konkret eingesetzten Technologie zur Formalisierung in dieser Arbeit, von Dritten bereitgestellte maschinenlesbare Beschreibungen zukünftig relevanter Modelle, Standards oder Schnittstellen zu keinem Zeitpunkt alle im selben Format verfügbar sein werden. Die daher notwendige Anpassung bzw. Konvertierung in ein gemeinsam genutztes Format muss aber von voneinander unabhängigken Parteien vorgenommen werden können, was nur mit durch einen offenen Standard ermöglicht wird.

N2 *Erweiterbarkeit des Laufzeitsystems*
Das Laufzeitsystem soll breit anwendbar sein, also nicht nur auf einem spezifischen, einmal entwickelten und implementierten Informationsmodell für einen konkreten Anwendungsfall arbeiten, sondern auch durch inkrementelles Zufügen von Informationsmodellen, Regelmengen und Softwarekomponenten zur Ansteuerung externer Schnittstellen erweiterbar sein.

N3 *Bedienbarkeit des Laufzeitsystems*
Das Laufzeitsystem muss sowohl über Programmier- als auch Bedienschnittstellen verfügen, mit denen die Kernaufgaben durchgeführt und nachvollzogen werden können und die zur Fehlersuche in der Wissensbasis genutzt werden können (wenn sich beispielsweise tatsächliche von erwarteten Reasoning-Ergebnissen unterscheiden).

N4 *Für domänenübergreifende Managementaufgaben geeignete Performance*
Um Validierung und Reasoning auf den Wissensmodellen durchführen zu können, dürfen diese nicht nur nicht untentscheidbar sein, sondern es müssen hierfür auch Algorithmen existieren, deren Komplexität eine Implementierung mit Laufzeiteigenschaften ermöglicht, die für den Anwendungsfall angemessen sind. Mit anderen Worten, Laufzeiten von Bearbeitungs- oder Auswertungszyklen müssen in einer Größenordnung liegen, die einen sinnvollen Einsatz des Systems für die jeweilige Managementaufgabe erlaubt. Ebenso muss das Laufzeitsystem diesen Kriterien entsprechende Implementierungen enthalten.

5.2 Architektur

5.2.1 Überblick

Die grundlegende Idee zur Lösung der in Abschnitt 1 beschriebenen Problematik hat zwei wesentliche Bestandteile: Einerseits soll sich die Wissensbasis, die für den Regelkreis genutzt wird, aus modularen, formalen Wissensmodellen zusammensetzen. Zwei Ziele sind dabei vorrangig: Die Informationsmodelle müssen für die Automatisierung verarbeitbar sein und die Wissensbasis muss erweiterbar sein. Auf diese Weise soll es möglich sein, weitere Systembestandteile anzubinden, aber auch die Unterstützung von weiteren Sichten hinzuzufügen. Andererseits muss ein Laufzeitsystem geschaffen werden, das dem MAPE-K-Prinzip folgend die Automatisierung von Systemen durchführen kann und anhand der Online-Auswertung von Informationen aus den Bestandteilen des zu verwaltenden Systems Entscheidungen zu deren Steuerung treffen kann. Dies entspricht dem Inhalt der funktionalen Anforderung **F6**.

Der vorgestellte Lösungsansatz ist das Ergebnis einer inkrementellen Weiterentwicklung der Grundidee, die über den Zeitraum mehrerer Jahre im Rahmen eines Anwendungsfalls mit

industrieller Kooperation vorgenommen wurde. Details des Anwendungsfalls werden in Abschnitt 8.2 erläutert, hier wird zunächst das konzeptuelle Vorgehen beschrieben. Das Vorgehen wird in vier Bestandteile unterteilt, die in Abbildung 5.1 dargestellt sind. Aufgrund ihrer Bedeutung werden die *Modularisierung* der Wissensbasis in Abschnitt 5.2.2 und das *Laufzeitsystem* in Abschnitt 5.2.3 detaillierter vorgestellt.

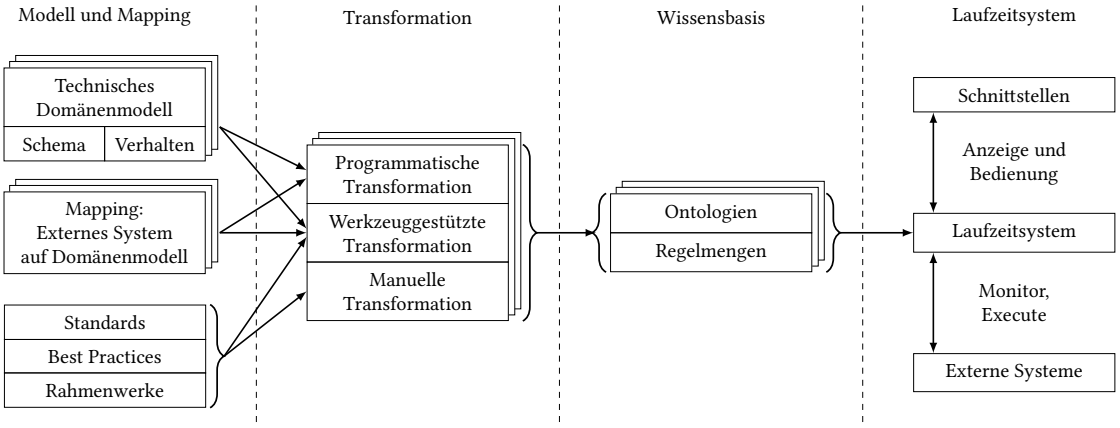

Abbildung 5.1: Vorgehen des Lösungsansatzes

Modell und Mapping

Als Ausgangsbasis des Automatisierungssystems müssen die für den jeweiligen Anwendungsfall benötigten Informationsmodelle in einer für die automatisierte Verarbeitung geeigneten Form vorliegen. Der Grad der Formalisierung, den existierende Informationsmodelle haben, variiert stark. Wie in Abschnitt 4 beschrieben, sind existierende Informationsmodelle tendenziell stärker formalisiert verfügbar, je dichter sie konzeptionell an konkreten technischen Systemen angesiedelt sind. Der Grad reicht dabei von schwachen Formalisierungen wie Listen von Bezeichnern (z.B. CSV oder Excel-Dokumente) über technologie-orientierte wie RDBMS- oder XML-Schemata bis hin zu starken Formalisierungen wie CL (siehe Abschnitt 3.1.2).

Je weiter sich eine Domäne inhaltlich von technischen Implementierungen entfernt, desto seltener sind formale Modelle verfügbar, selbst wenn in der Domäne klare Strukturen etabliert oder informell spezifiziert sind. Im „Modell und Mapping“-Schritt müssen daher zunächst diejenigen Informationsmodelle benannt und ausgewählt werden, die für den Anwendungsfall benötigt werden. Technische Domänenmodelle können hierbei auch teilweise implizit durch die Struktur oder Implementierung existierender Systeme gegeben sein.

Die Informationen, die nötig sind, um Laufzeitinformationen aus den zu verwaltenden Systembestandteilen beziehen zu können, werden ebenfalls als Informationsmodell aufgefasst und

hier als „Mapping“ bezeichnet. Dies kann einfache Konfigurationen wie eine Menge von Hosts beinhalten, aber auch komplexer sein, wie die Beschreibung der Abbildung eines relationalen Schemas auf eine Ontologie. Außerdem müssen die einzusetzenden nicht-technischen Domänenmodelle wie Standards, Best Practices und Rahmenwerke einbezogen werden. Diese sind im Prinzip allgemeingültig: Wurde ein bestimmter Standard oder ein bestimmtes Rahmenwerk einmal in ein entsprechendes formales Modell übersetzt, kann es in beliebigen Kontexten (Projekte oder Unternehmen) genutzt werden, vorausgesetzt, das Modell wird entsprechend verfügbar gemacht.

Transformation

Der zweite Schritt, der hier als Transformation bezeichnet wird, beinhaltet die Aufbereitung der Informationsmodelle. Ausgangsmodelle sollen so in Ontologien transformiert werden, dass ihre ursprünglich *angedachte* Semantik erfasst wird; das bedeutet, dass Informationen ggf. aus unterschiedlichen Quellen, wie maschinell lesbaren Daten und zugehöriger menschenlesbarer Dokumentation, zusammengeführt werden. Darüber hinaus sollen die in Abschnitt 5.1 beschriebenen Anforderungen erfüllt werden, insbesondere Anforderungen **F7** (*Domänenübergreifende Wissensbasis*), **F8** (*Modulare Wissensbasis*) und **N1** (*Orientierung an bestehenden Standards*).

Die konkreten Arbeitsschritte sind dabei abhängig von den jeweiligen Modellen, es muss in jedem Fall jedoch eine Mischung aus Software-Entwicklung und Ontology-Engineering eingesetzt werden. Bereits maschinell verarbeitbare aber sehr umfangreiche oder komplexe Modelle werden dabei durch speziell zu diesem Zweck zu erstellenden Software-Werkzeugen automatisch in das Zielformat übersetzt, dies wird hier „Programmatische Transformation“ genannt. Ein Beispiel hierfür ist ein umfangreiches XML-Modell, in dem anwendungsspezifische Semantik kodiert ist. Die Spezifikation der Semantik liegt dabei entweder explizit in der Dokumentation, oder implizit im Quellcode der verarbeitenden Anwendung vor. Eine Extraktion der Semantik des Modells in Form einer sinnvoll nutzbaren Ontologie ist hier nur möglich durch ein Software-Werkzeug, welches das Ursprungsmodell interpretiert und eine Menge äquivalenter Axiome in der verwendeten Ontologiesprache erzeugt. Programmatische Transformation ist dann anzuwenden, wenn mindestens eine von zwei Bedingungen erfüllt ist: Die Transformation soll automatisch wiederholbar sein, beispielsweise weil regelmässig aktualisierte Versionen des Ausgangsmodells veröffentlicht werden, deren Nutzung ermöglicht werden soll, oder die anderen Transformationsarten sind mit einem bedeutend höheren Aufwand verbunden.

Werkzeuggestützte und manuelle Transformation sind nicht automatisch wiederholbar. Bei der werkzeuggestützten Transformation werden alle sinnvoll nutzbaren verfügbaren Werkzeuge eingesetzt, die bei der Erstellung der Ontologie aus einem schwach formalisierten Ausgangsformat helfen, beispielsweise Textextraktion bzw. -manipulation. Dies kann von einfachen Operationen wie „Suchen und Ersetzen“ bis hin zu komplexen heuristischen Verfahren

reichen, die auch zum Ontology Matching eingesetzt werden können, wie in Abbildung 3.16 gezeigt. Die werkzeuggestützte Transformation wird durch manuelle Modellierung von Entitäten und Axiomen ergänzt. Bestandteile der Semantik, die nicht durch die initiale Extraktion erzeugt werden können, müssen nachgepflegt werden, beispielsweise Eigenschaften von Relationen oder Untermengenbeziehungen.

Eine manuelle Transformation entspricht der Neuentwicklung einer Ontologie unter Verwendung einer geeigneten Ontology-Engineering-Methodik. Dies ist die einzige nicht auf einer Heuristik basierende Möglichkeit zur Erfassung der Semantik von Ausgangsmodellen, die nur in Form einer natürlichsprachlichen Spezifikation verfügbar sind. Das konkrete Vorgehen sowie die Reproduzierbarkeit unterliegt hier der angewandten Methodik; die Qualität der resultierenden Ontologie hängt ab von der Methodik und der Erfahrung des Modellierers.

Wissensbasis

Die resultierende Wissensbasis umfasst schließlich Wissensmodelle aller benötigten Domänen, Standards und Rahmenwerke und setzt sich zusammen aus der Menge der Ontologien, sowie Regelmengen, in denen diejenigen Wissensfragemente kodiert sind, die sich nicht in der Semantik der Ontologiesprache ausdrücken lassen. Insbesondere beinhalten die Regeln das Verhalten, das zur Laufzeit des Managementsystems den Regelkreislauf steuert, und umfassen die folgenden Aspekte:

- Steuerung, unter welchen Bedingungen Adapter externe Systeme abfragen,
- Transformation und Abbildung importierter Daten auf das Schema der zugehörigen Domänenontologie,
- Umsetzung der A- und P-Komponenten des Regelkreises, ggf. durch Einbindung von Software-Erweiterungen der Regelengine,
- Durchführung von Reasoning.

Um die Erfüllung der Anforderungen **F8** (*Modulare Wissensbasis*), **N1** (*Orientierung and bestehenden Standards*) und **N4** (*Für domänenübergreifende Managementaufgaben geeignete Performance*) gewährleisten zu können, wird in dieser Arbeit als Ontologiesprache OWL 2 im RL Profil eingesetzt (siehe 3.4.2). Elemente einzelner Bestandteile der Wissensbasis müssen nach einem nachvollziehbaren und erweiterbaren Schema global eindeutiger Bezeichner benennbar sein, die gleichzeitig auch Ontology Reconciliation erlauben. Dies ist durch die Verwendung von IRIs in RDF und OWL gegeben, vergleichbare Mechanismen fehlen Formaten wie den unterschiedlichen UML-Serialisierungen, XML sowie CLIF. Obwohl RDF aus diesem Grund für die Definition zahlreicher freier Vokabulare eingesetzt wird, fehlen hier im Vergleich zu OWL

wichtige Konstrukte, die für die Erfüllung der Anforderungen nötig sind, darunter der Import-Mechanismus und die Möglichkeit der Beschreibung von Klassenrestriktionen. Schließlich zeigen Meyer und Kröger in [MK15], dass das OWL 2 RL-Profil gerade für Ontologie-basiertes IT-Management am besten geeignet ist, da die Ausdrucksmächtigkeit und die Möglichkeit der geeigneten Implementierung eines entsprechenden Reasoners hier ausgeglichen sind.

Der letzte Bereich in Abbildung 5.1 ist das eigentliche Laufzeitsystem, das die Ontologien und Regelmengen der Wissensbasis nutzt, um IT-Managementaufgaben durchzuführen. Die Aufgaben des Laufzeitsystems leiten sich aus den Anforderungen **F6**, **F7** und **F8** her und werden in Abschnitt 5.2.3 vorgestellt.

5.2.2 Modularisierung der Wissensbasis

Die zwei zentralen Anforderungen an die Wissensbasis sind Anforderung **F7** (*Domänenübergreifende Wissensbasis*) und Anforderung **F8** (*Modulare Wissensbasis*). Während die meisten Arbeiten in der Literatur, die sich mit der Modularisierung von Ontologien beschäftigen, den Fokus auf Ontology Partitioning legen (siehe Abschnitt 3.6.3), umfasst die Fragestellung zur Modularisierung in dieser Arbeit im Wesentlichen Aspekte des Ontology Matching und außerhalb des eigentlichen Ontology Engineering angesiedelte Aspekte des Software-Engineering.

Es soll hier nicht untersucht werden, wie Ontologien während ihrer Entwicklung in logische Einzelteile zerlegt werden können, sondern wie die nachträgliche, modulare Erweiterung der Wissensbasis des Laufzeitsystems auch durch Informationsmodelle von im Vorfeld nicht explizit vorgesehenden Domänen erreicht werden kann. Die Operation des Zufügens muss also ermöglichen, sowohl Axiome als auch Alignments nachzuladen. Die Strategie zur Modularisierung soll dabei aber nicht einschränken, in welcher Form Alignments entwickelt oder spezifiziert werden. Der Klassifikation von Techniken zum Ontology Matching von Euzenat und Shvaiko (Abbildung 3.16) folgend sollen insbesondere auf Ebene der Eingaben kontextbasierte sowie auf Ebene der Interpretation strukturelle Matching-Techniken einsetzbar sein; diese sind für den Aufbau einer Wissensbasis für das automatisierte Management am nützlichsten.

Für die Erfüllung von Anforderung **N1**, nach der ein erweiterbares System Standard-Technologien einsetzen muss, sind beispielsweise Erweiterungen der grundlegenden Formalismen wie Distributed Description Logics (siehe Abschnitt 3.6.3) nicht einsetzbar. Die Entwicklung von Alignments, die den Ausdruck unterschiedlicher Grade von Übereinstimmungen der Korrespondenzen zwischen Ontologien unterstützen, kommen in dieser Arbeit ebenfalls nicht zum Einsatz, ihre Nutzung ist aber im automatisierten IT-Management denkbar.

Wie in Abschnitt 5.2.1 erläutert, besteht ein Informationsmodell nicht nur aus einer *Domänenontologie* (oder einer Vereinigung mehrerer Ontologien), sondern auch aus einer oder mehreren benannten Mengen von *Domänenregeln*. Regeln können dabei Elemente aus einer oder mehreren beliebigen Ontologien der Wissensbasis referenzieren. Um Abhängigkeiten zwischen

domänenspezifischen Informationsmodellen zu beschreiben, wäre eine Beschränkung auf den OWL-Import-Mechanismus hierfür nur dann ausreichend, wenn Regeln ausschließlich in Ontologien eingebettet beschrieben würden. Techniken dieser Form existieren, beispielsweise werden SWRL-Regeln innerhalb von OWL-Ontologien serialisiert und Regeln in der sogenannten SPARQL Inferencing Notation (SPIN, [KHI11]) werden innerhalb eines RDF-Dokuments als verschachtelte anonyme Knoten gespeichert. Dies macht allerdings spezialisierte Werkzeuge zur Verarbeitung nötig; eine Erweiterung oder Änderung des Regelformats ist nur schwer möglich.

Neben der Kodierung von Abhängigkeiten zwischen Informationsmodellen müssen auch solche zwischen den logisch zusammengehörenden Informationsfragmenten innerhalb eines Informationsmodells ausdrückbar sein: Regeln können nicht nur Elemente der zugehörigen Domänenontologie referenzieren, sondern müssen auch die Möglichkeit haben, auf die oben beschriebenen Software-Erweiterungen der Regelengine Bezug zu nehmen. Diese Erweiterungen werden als *Built-Ins* bezeichnet und sind programmatische Erweiterungen der Regelengine, deren Schnittstelle in Form einer Funktion innerhalb eines Regel-Körpers oder Regel-Kopfes aufgerufen werden können. Im Regel-Körper fungiert ein Built-In als zusätzlicher Matcher, der mitentscheidet, in welchen Situationen die Regel feuern kann, im Regel-Kopf kann das Built-In zusätzliche Fakten generieren oder Seiteneffekte ausführen, wie beispielsweise Logging. Für Built-Ins können zusätzliche, anwendungsspezifische *Ressourcen* in ein Modul mit aufgenommen werden, beispielsweise Zertifikate, die für den Verbindungsaufbau zu einer Datenquelle benötigt werden.

Um zusammengehörige Informationsfragmente, also Domänenontologien, Regelmengen und Built-Ins, so zu verbinden, dass sie zur Auflösung von Abhängigkeiten als Einheit erscheinen, werden sie dazu als hier sogenannte Ontologiemodule zusammengefasst.

Abbildung 5.2 zeigt den strukturellen Aufbau eines Ontologiemoduls und die technischen Abhängigkeiten, die innerhalb der Bestandteile eines Modules existieren können, sowie die logischen Abhängigkeiten zwischen zwei Ontologiemodulen. Domänenontologien, Domänenregeln, Built-Ins und Ressourcen sind als Informationsfragmente in einem Ontologiemodul jeweils optional. Zusätzlich enthält jedes Modul das verpflichtende Fragment mit Metadaten. Welche Metadaten im Detail enthalten sein müssen, ist der Implementierung überlassen, es muss jedoch mindestens die Möglichkeit geben, die Abhängigkeiten des Moduls an andere Module zu beschreiben, also welche anderen Module referenziert und daher benötigt werden. Darüber hinaus können die Metadaten Angaben machen über Autoren, Lizenzen, Version des Moduls bzw. der Fragmente, Ursprung von Aussagen innerhalb der Domänenontologie (Provenance), Ontologiestatistiken (Anzahl von Invidiuen, genutzte Sprachfeatures), exportierte Namensräume sowie die Schnittstellen eventuell enthaltener Built-Ins.

Jede Verbindung zwischen zwei Modulen oder Fragmenten eines Moduls mit einem anderen Fragment stellt dabei eine mögliche Abhängigkeit dar. Alle Abhängigkeiten sind optional;

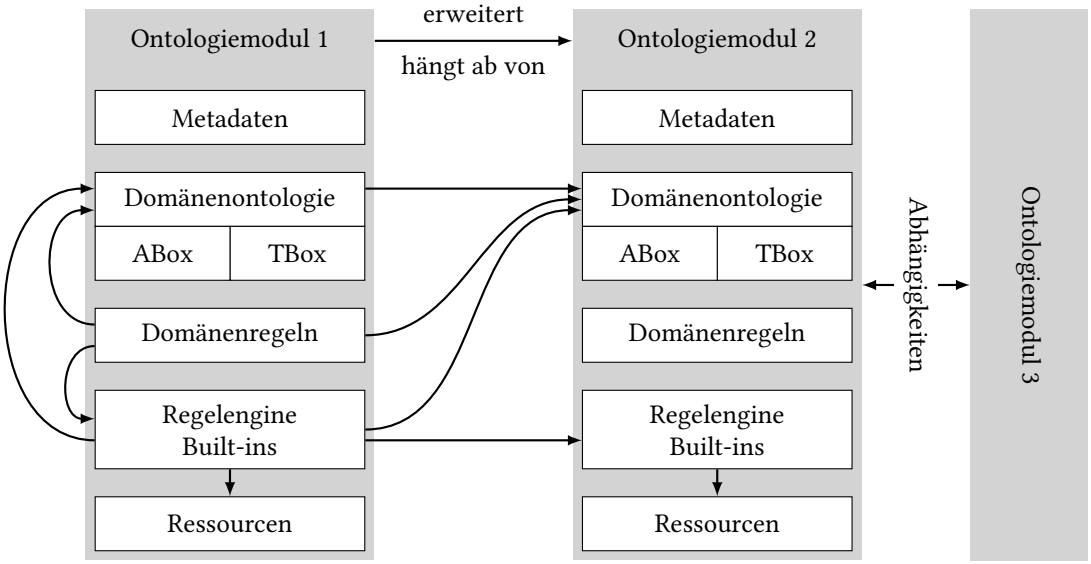

Abbildung 5.2: Aufbau von und Abhängigkeiten zwischen Ontologiemodulen

diejenigen Abhängigkeiten, die innerhalb eines Modules möglich sind, werden aber in jedem nicht-trivialen Modul genutzt, sofern die jeweiligen Fragmente vorhanden sind.

Built-Ins sind im Gegensatz zu den anderen Fragmenten Software-Artefakte und müssen die entsprechende Erweiterungs-Schnittstelle der Regelengine implementieren. Sie können die Domänenontologie innerhalb des Moduls programmatisch referenzieren, z.B. ist der Zugriff auf die ABox nötig für eine Analyse von Zusammenhängen zwischen Individuen oder die Aggregation von Attributen von Individuen. Die TBox wird genutzt, wenn das Built-In die Aufgabe eines Adapters zu einem externen System übernimmt und Informationen in die Domänenontologie hinzufügen soll.

Die Kapselung von Informationsfragmenten und Abhängigkeiten in dieser Form ermöglicht die inkrementelle Erweiterung der Wissensbasis, da nicht nur die benötigten Informationsmodelle als Module beschrieben werden können, sondern auch Alignments. Alignments zwischen Entitäten von Ontologien unterschiedlicher Module werden durch Einführung von Axiomen in der erweiternden Ontologie ausgedrückt, die Entitäten der erweiterten Ontologie referenzieren. Hierdurch werden direkt alle Möglichkeiten der Beschreibung von Ontologie-Alignments unterstützt, die auf OWL-Sprachfeatures basieren. Ebenso ist aber eine Erweiterung um externe Alignment-Beschreibungen möglich, wie die in Abschnitt 3.6.3 beschriebene Methode von Scharffe [SD05, SE07], wodurch auch Alignmentbeschreibungen mit gewichteten Korrespondenzen ausdrückbar sind. Solche Alignments werden dann als zusätzliche Ressourcen im Modul abgelegt, auf die aus den Built-Ins zugegriffen werden kann.

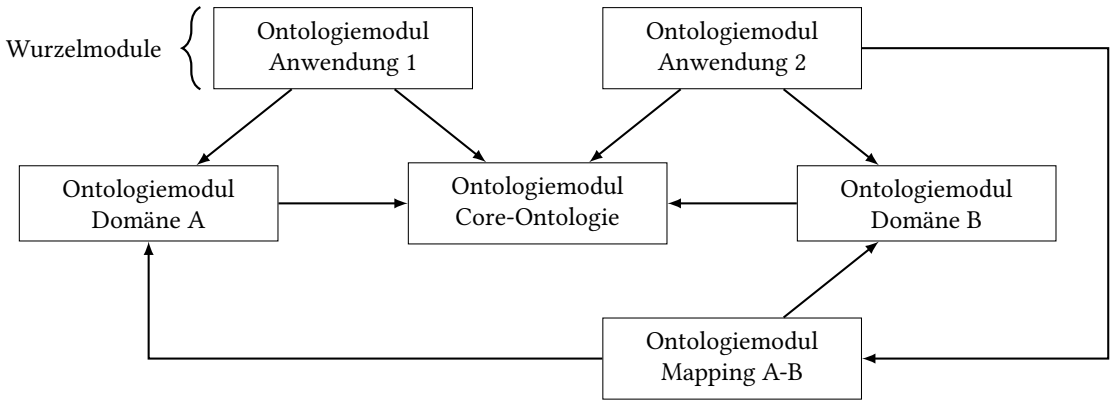

Abbildung 5.3: Zyklische Abhängigkeiten und Abbildungen zwischen Ontologiemodulen

Durch die Kombination von Ontologiemodulen für domänenspezifische Informationsmodelle und Ontologiemodulen zur Beschreibung von Abbildungen zwischen anderen Modulen können mehrere Anwendungsfälle, die jeweils unterschiedliche Module nutzen, auf der gleichen Wissensbasis agieren. Abbildung 5.3 zeigt die Konfiguration eines Beispielszenarios. Ein Anwendungsfall verfügt über eine Menge von Modulen und über genau ein Anwendungsfall-spezifisches Modul, das alle anderen benötigten referenziert, das *Wurzelmodul*. Für Anwendungsfall 1 beinhaltet dies das Modul für Domäne A und das Modul einer Core-Ontologie. Wird nun ein Anwendungsfall 2 zugefügt, der nicht nur die weitere Domäne B einbeziehen soll, sondern auch durch eine Abbildung zwischen Domänen A und B Informationen in Bezug zu Domäne A setzen soll, entsteht eine Konfiguration, wie in der Abbildung gezeigt. Das Ontologiemodul für Domäne A muss nicht angepasst werden; keines der Module im Abschluss der Abhängkeiten des Wurzelmoduls von Anwendungsfall 1 wurde geändert oder durch Abhängigkeiten ergänzt. Wichtig ist hierbei, dass domänenspezifische Module, Core-Ontologie-Module und Mappings keine Abhängigkeiten an Wurzelmodule haben. Hierdurch kann ein Modul mit dem Abschluss seiner Abhängigkeiten direkt in beliebigen weiteren Wurzelmodulen genutzt werden. Dieses Prinzip erlaubt die schrittweise Erweiterung der Wissensbasis um die Zusammenhänge zwischen Informationsmodellen unterschiedlicher Domänen, die unabhängig von den jeweiligen Abbildungen entwickelt werden können.

Die Struktur der Module ermöglicht darüber hinaus auch eine inkrementelle Entwicklung der Wissensbasis über einen längeren Zeitraum unter Verwendung versionierter Domänenontologien. Dazu werden für eine Domänenontologie *zwei* Ontologiemodule angelegt: Das erste Modul enthält die TBox der Ontologie und eine leere ABox. Das zweite Modul enthält die ABox der Ontologie, und eine TBox, die auf taxonomische Informationen reduziert wurde, also nur grundlegende strukturelle Zusammenhänge beschreibt, aber keine komplexen Sachverhalte wie Restriktionen. Das Laufzeitsystem fügt aktualisierte Informationen in Form von Individuen und Aussagen in die ABox des zweiten Moduls ein. Abfragen an die Domänenonto-

logie werden an die Vereinigung der TBox des ersten Modules und die ABox des zweiten Modules gestellt und berücksichtigen dadurch die Domänensemantik. Wird daraufhin die Domänensemantik weiterentwickelt oder ergänzt, beispielsweise in Form neuer Klassen, die durch Restriktionen Individuen neu gruppieren, kann die Version 2 dieses TBox-Moduls zusätzlich geladen werden. Abfragen werden dann auf die Vereinigung des neuen TBox-Moduls mit dem ABox-Modul gestellt. Die Ursprungsdaten bleiben die selben, da das ABox-Modul nicht verändert wurde, aber die Informationen, die durch die Vereinigung und im Rahmen der Abfrage durchgeführtes Reasoning zurückgeliefert werden, sind umfangreicher und vollständiger und können Ausschnitte oder Sichten auf die Daten beschreiben, die mit der vorigen Version der TBox in dieser Form nicht verfügbar waren.

5.2.3 Konzepte des Laufzeitsystems

Die grundlegende Aufgabe des Laufzeitsystems ist die Verwaltung der in Abschnitt 5.2.2 beschriebenen Ontologiemodule. Dies umfasst die Basisfunktionen wie das Laden und die Verwaltung von Abhängigkeiten zwischen den Modulen, die MAPE-K-zentrischen Funktionen zum Management des zu verwaltenden Systems und die Verwaltungsfunktionen zum manuellen oder programmatischen Abfragen von Informationen aus den Ontologiemodulen und der Steuerung des Laufzeitsystems selbst.

Abbildung 5.4 zeigt eine Übersicht über die Komponenten des Laufzeitsystems, die im Folgenden beschrieben werden. Aus der Anforderung **F6** (*Laufzeitsystem zur wissensbasierten Automatisierung*) kann eine Mindestmenge von Komponenten hergeleitet werden: Als Kernfunktionalität muss die Möglichkeit der Kommunikation mit dem zu verwaltenden System bestehen (mindestens das Lesen von Monitoring- und Event-Informationen und das Eingreifen in das System über entsprechende Schnittstellen). Um das Laufzeitsystem für beliebige Zielsysteme einsetzbar zu machen und Anforderung **N2** (*Erweiterbarkeit des Laufzeitsystems*) zu folgen, wird *Regelengine* benötigt, die von systemspezifischen Adaptern in Form von Built-Ins erweitert werden kann. Die Bestandteile der Ontologiemodule, die in Abbildung 5.2 gezeigt werden, müssen von entsprechenden Komponenten gehandhabt werden, daher wird außer der Regelengine und der Built-In-Schnittstelle mindestens eine Komponente zur Verwaltung von Ontologien benötigt, die hier als *Modulmanager* bezeichnet wird.

Die Aufgaben des Modulmanagers umfassen das Laden von Ontologiemodulen einschließlich des Parsens von Ontologien und die Bereitstellung eines Modells und einer API zum Zugriff auf die Inhalte des Moduls, also der Ontologie, Regelmengen, Ressourcen und Metadaten. Als gestrichelte Linien wird in der Abbildung gezeigt, an welchen Stellen der Architektur die Bestandteile der Ontologiemodule zum Einsatz kommen. Passive Informationsfragmente wie die Ontologien und Regelmengen werden Teil der Wissensbasis; die Built-Ins sind Software, die die Built-In-Schnittstelle der Regelengine implementieren.

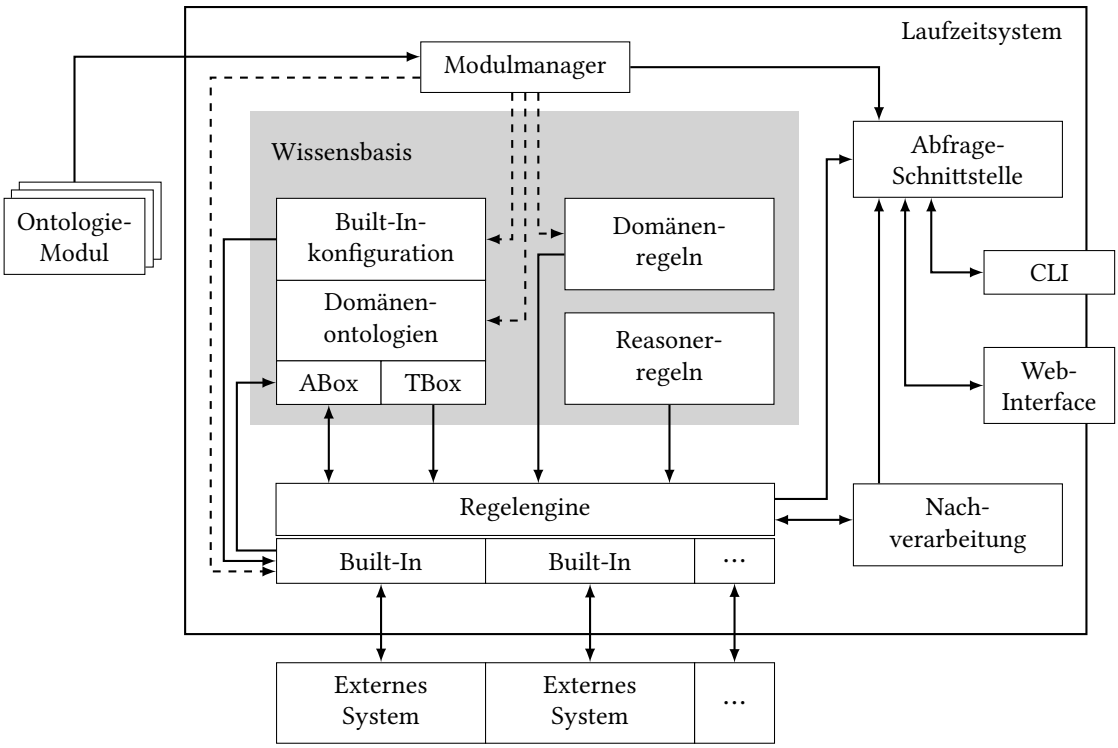

Abbildung 5.4: Architektur des Laufzeitsystems

Je nach Umfang und Komplexität eines Built-Ins wird zusätzliche Konfiguration nötig, die als passive Ressource ebenfalls in die Wissensbasis eingefügt wird. Die Konfiguration wird dabei ebenfalls in Form einer Ontologie beschrieben. Hierdurch kann zum einen die Konfiguration direkten Bezug nehmen auf Entitäten, die in der Domänenontologie beschrieben sind, ohne eine klare Trennung von Domänenmodell und implementierungsspezifischem Modell zu verletzen. Zum andern wird die Konfiguration genauso wie die Domänenontologie durch die selben Technologien abfragbar.

Die Regelengine hat die Aufgabe, die in Ontologiemodulen enthaltenen Mengen von Domänenregeln auszuwerten und nutzt zum Zugriff auf die in den Regeln referenzierten Ontologieentitäten die API der Ontologieverwaltung. Die Domänenregeln werden dazu eingesetzt, die im jeweiligen Anwendungsfall benötigten strukturellen und inhaltlichen Transformationen von Daten vorzunehmen, wo dies möglich ist. Sie können daher Manipulationen am Informationsmodell vornehmen, die ansonsten häufig hartkodiert in anwendungsspezifischem Programmcode realisiert sind. Dies kann die Wartbarkeit erhöhen, indem die Komplexität durch Einsatz von deklarativen Regeln reduziert wird.

Entsprechend Anforderung **F7** (*Domänenübergreifende Wissensbasis des Laufzeitsystems*) wird

darüber hinaus ein *Reasoner* benötigt, der die Semantik der Domänenontologien auswerten kann und Daten, die über die als Built-Ins realisierten Adapter vom zu verwaltenden System gelesen werden, unter Berücksichtigung des zugehörigen Schemas aus der TBox validieren kann. Der Designentscheidung zum Einsatz von OWL 2 RL als Ontologiebeschreibungssprache folgt, dass der eingesetzte Reasoner die OWL 2 RL-Semantik unterstützen muss. Da die diesem Profil zugrunde liegende Semantik, wie in Abschnitt 3.4.2 beschrieben, durch eine Regelmenge beschrieben werden kann, wird die für die Architektur bereits vorgesehene Regelengine durch Einsatz dieser Regelmenge gleichzeitig zum Reasoner. Daher ist die Menge der Reasoner-Regeln eine der Eingaben für die Regelengine, wie in der Abbildung gezeigt. Da die Regeln des OWL 2 RL-Profils wohldefiniert sind, müssen sie nicht wie die restlichen Domänenontologien erst aus einem Ontologiemodul geladen werden.

Die Trennung des Reasoners in eine generische Regelengine und eine Regelmenge, die die gewünschte Semantik umsetzt, hat neben der Reduktion der Menge speziell zu implementierender Komponenten und damit auch der Reduktion von Komplexität einen weiteren Vorteil. Im Hinblick auf Anforderung **N4** (*Für domänenübergreifende Managementaufgaben geeignete Performance*) kann einzelnen Ontologiemodulen auf diese Weise jeweils eine von der OWL 2 RL-Semantik abweichende Grundregelmenge zugeordnet werden. Obwohl zum vollen Ausschöpfen der Mittel, die die OWL-Modellierung bietet, also alle Features des RL-Profils einsetzt werden können, kann die bewusste Nutzung einer eingeschränkterer Ausdrucksmächtigkeit zu einem Performancegewinn beitragen. Einer der Anwendungsfälle, in der diese Möglichkeit sinnvoll eingesetzt werden kann, ist die Konstruktion von Ontologiemodulen, die im Wesentlichen als ABox-Container dienen, wie in Abschnitt 5.2.2 beschrieben. Hier kann statt OWL 2 RL eine reine RDFS-Semantik oder eine leere Regelmenge für eine komplette Deaktivierung des Reasonings eingesetzt werden. Ebenso ist eine bestimmte, auf den Anwendungsfall und die eingesetzte Ontologie angepasste Untermenge der OWL 2 RL-Semantik möglich.

Der Einsatz eines regelbasierten Reasoners bedingt eine weitere Komponente, die das Laufzeitsystem bereitstellen muss, um in beliebigen Szenarien einsetzbar zu sein; ein System zur Beschreibung von Berechnungsvorschriften, die sich nicht oder nur durch großen Modellierungsaufwand in der Ontologie und den Regelmengen ausdrücken lassen. Diese Komponente wird hier als *Nachverarbeitung* bezeichnet und wird insbesondere für Aufgaben wie die Berechnung von Aggregationsfunktionen eingesetzt, beispielsweise von Summen oder Mittelwerten von Attributen einer Menge von Individuuen. Derartige Berechnungen lassen sich in OWL bzw. in Form von Regeln nicht effizient ausdrücken, sollen aber ein Teil des Informationsmodells der Ontologiemodule und nicht der Implementierung des Laufzeitsystems sein. Konzeptuell wird also die Ausdrucksfähigkeit der Ontologiemodule erweitert, indem eine spezielle, über das Standard-Reasoning hinausgehende Funktionalität durch die Nachverarbeitung bereitgestellt wird. Dies steht nicht im Widerspruch zu Anforderung **N1** (*Orientierung an bestehenden Standards*), solange die Beschreibung der Nachverarbeitungsvorschriften ebenso wie die Ontologien basierend auf Standardformaten ausgedrückt werden.

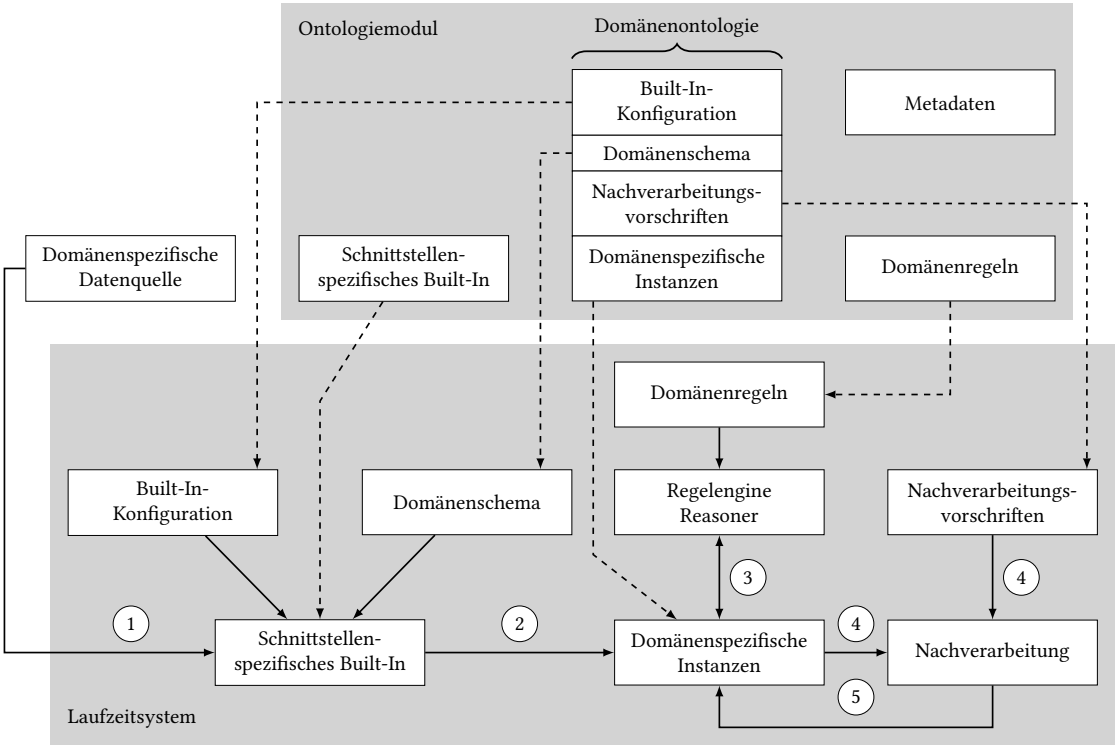

Abbildung 5.5: Datenfluss des Laufzeitsystems

Für die Erfüllung der Verwaltungsfunktionen wird schließlich noch eine *Abfrageschnittstelle* benötigt, über die auf Informationen in der Wissensbasis, des Zustandes der Nachverarbeitung und des Modulmanagers zugegriffen werden kann. Die Abfrageschnittstelle wird genutzt von den Komponenten, die Bedienschnittstellen für Benutzer anbieten, hier ein CLI und ein Web-Interface, entsprechend der Anforderung **N3** (*Bedienbarkeit des Laufzeitsystems*).

Um den Ablauf beim Einsatz des Laufzeitsystems zu verdeutlichen, wird der Datenfluss beim Abfragen des zu verwaltenden Systems in Abbildung 5.5 erklärt. Der obere Kasten steht repräsentativ für eines der zur Laufzeit geladenen Ontologiemodule und zeigt die einzelnen enthaltenen Komponenten, der untere Kasten zeigt die Zusammenhänge zwischen den Komponenten des Laufzeitsystems. Der Ausgangspunkt ist ein Zustand, in dem das Laufzeitsystem geladen ist, das für das zu verwaltende System nötige Wurzelmodul sowie alle seine Abhängigkeiten geladen und aufgelöst sind und der Zugriff auf die domänenspezifische Datenquelle besteht. Im Rest des Abschnitts erfolgt eine allgemeine Beschreibung des Datenflusses, die durch ein laufendes Beispiel unterstützt wird.

Das schnittstellenspezifische Built-In nutzt das Schema der Domänenontologie als sein Datenmodell und liest zu Beginn seines Lebenszyklus die Ontologie mit der zugehörigen Built-

In-Konfiguration. Die Konfiguration kann dabei einfache Werte enthalten wie beispielsweise Hostnamen, Datenbanknamen oder Benutzernamen, aber auch komplexere Zusammenhänge wie Abbildungsvorschriften von externen Datenstrukturen auf die Ontologie, sowie Angaben zur Konstruktion von Bezeichnern neu zu generierender Entitäten.

In Schritt ① werden Daten durch das Built-In von der Datenquelle gelesen. Die Art der Kommunikation zwischen Datenquelle und Built-In wird gegenüber dem restlichen Laufzeitsystem durch das Built-In gekapselt. Einfache Dateien können prinzipiell ebenso als Datenquelle dienen wie programmatische Schnittellen zu externen Systemen. Je nach Beschaffenheit des abzufragenden Systems muss die Built-In-Implementierung entweder aktiv Daten anfragen (Polling) oder zum Empfang von Daten bereit sein (Event-getrieben). Im zweiten Fall muss ebenfalls in der Implementierung entschieden werden, zu welchen Zeitpunkten bzw. in welchem Intervall der Datenfluss fortgesetzt wird, wieviel beispielsweise also zunächst gepuffert werden muss.

Analog zur Handhabung der Kommunikation wird auch die Vergabe von Bezeichnern für Entitäten im Built-In gehandhabt. Da alle Entitäten innerhalb der Ontologie eindeutig durch eine IRI referenzierbar sein müssen, wie in Abschnitt 3.2 beschrieben, aber nicht alle abfragbaren Datenquellen Entitäten durch IRIs oder kompatible Formen wie URLs oder URNs identifizieren, muss eine entsprechende Übersetzung im Built-In vorgenommen werden. Wie diese Übersetzung im Detail abläuft, hängt von der Datenquelle ab und muss von der Built-In-Implementierung umgesetzt werden. Für den Fall, dass die Datenquelle ein RDBMS ist, wurden in Abschnitt 3.6.3 Arbeiten vorgestellt, die sich mit einer solchen Abbildung beschäftigen. Generell muss die Abbildung nur die Kollisionsfreiheit der Bezeichner garantieren, wünschenswert ist darüber hinaus die Lesbarkeit bzw. die Möglichkeit der manuellen Zuordnung zum ursprünglichen Datensatz durch einen Domänenexperten, um Log- und Debugvorgänge zu vereinfachen. Die Abbildung muss surjektiv sein, eine Bijektivität ist allerdings nicht erforderlich.

Beispiel: Die Abbildung 5.6 zeigt im oberen linken Kasten einen exemplarischen Datensatz.

Das Built-In erzeugt eine Menge von Axiomen, die die grundlegende Struktur der gelesenen Daten mittels Ontologie-Entitäten beschreiben, also mittels Instanzen und Wertzuweisungen. In Schritt ② wird mit der Menge der Axiome die ABox der Domänenontologie in der Wissensbasis gefüllt. In einfachen Fällen, in denen wenige Referenzen zwischen den generierten Instanzen bestehen oder solche Referenzen kein Teil der Ausgangsdaten sind, können Entitäten so generiert werden, dass sie dem Schema der Domänenontologie entsprechen. Instanzen werden dann direkt als zugehörig zu benannten Klassen des Schemas ausgezeichnet und Beziehungsaussagen und Wertzuweisungen können die entsprechenden abstrakten bzw. konkreten Rollen nutzen, die im Schema beschrieben sind. Für die Übersetzung der Ausgagsdaten in Axiome nutzt ein Built-In seine Konfiguration, in der die Übersetzungsvorschriften für Namen, Struktur und Zielnamensraum von Elementen enthalten sein muss.

Beschreiben die Ursprungsdaten nur einen Ausschnitt des Modells, das nach dem Abschluss des Daten-Imports in der Domänenontologie verfügbar sein soll, oder haben die Ursprungsdaten eine vom Zielmodell abweichende Taxonomie, können stattdessen auch *transiente Entitäten* generiert werden. Solche Entitäten spiegeln die Struktur und Werte der Ausgangsdaten wider, diese entsprechen aber nicht dem Zielmodell. Sie sind weder Teil des Domänenschemas, noch sollen sie nach Abschluss des Daten-Imports Teil des Modells sein. Wertzuweisungen an transiente Entitäten können Bezeichner haben, die nur innerhalb der Built-In-Implementierung und der zugehörigen Domänenregeln Gültigkeit besitzen.

Beispiel: Ein Beispiel hierfür wird im oberen rechten Kasten in Abbildung 5.6 gezeigt. Das Modell folgt der grafischen Notation aus Abbildung 3.11. In diesem Beispiel sollen Ursprungsdaten in die Domänenontologie eingefügt werden, mit einem zugrunde liegende Datenformat, in dem aber Referenzen zwischen Entitäten nur über den Umweg von Bezeichnern ausgedrückt werden: Datensatz 2 referenziert Datensatz 1 über seine ID. Das Built-In erzeugt daher ein Modell, das die Struktur der ursprünglichen Daten reflektiert, dabei werden auch die IDs der Datensätze in Wertzuweisungen übersetzt. Die hierzu genutzte Eigenschaft `ex:_ref` ist dabei eine transiente Entität. Es wird das `name`-Attribut genutzt, um eine Instanz mit dem entsprechenden Namen im `ex`-Namensraum zu erzeugen, `id`- und `value`-Attribute werden für entsprechende Wertzuweisungen im `ex`-Namensraum und das `references`-Attribut für die `ex:_ref`-Wertzuweisung genutzt.

Nach der Verarbeitung durch die Regelengine (Schritt ③ in Abbildung 5.5), in der die OWL 2 RL-Semantik und die Domänenregeln verarbeitet werden, entspricht die ABox der Domänenontologie der gewünschten Struktur. Hierzu müssen diejenigen Domänenregeln definiert werden, die aus dem durch das Built-In erzeugte Modell die korrekte Struktur konstruieren, sofern dies nicht bereits durch die OWL 2 RL-Regeln geschieht. Insbesondere die Behandlung der transienten Entitäten muss daher durch Regeln erfolgen. Dadurch, dass ABox und TBox gemeinsam betrachtet werden, kann die noch fehlende Semantik durch den Reasoner explizit gemacht werden, ohne dass weitere spezifische Domänenregeln benötigt werden.

Beispiel: Die Menge der Domänenregeln umfasst in Abbildung 5.6 im unteren linken Kasten eine Regel, `adapter1`, die die fehlende Referenz zwischen den Instanzen herstellt. Der Regel-Körper besteht dabei aus einer Menge von Mustern, die auf Aussagen des RDF-Graphen passen, wenn die Regelengine die Variablen in den Mustern (durch `?` ausgezeichnet) binden kann. Der Regel-Kopf beschreibt dann eine Menge von Aussagen, die dem Graphen zugefügt werden. Die Zugehörigkeit der Instanzen `ex:Date1` und `ex:Date2` zur Klasse `ex:Date` kann durch den in der TBox der Domänenontologie beschriebenen Wertebereich der Rolle `ex:value` hergeleitet werden. Ebenfalls in der TBox definiert sind die konkreten Rollen `ex:id` und `ex:value`, die Wertzuweisungen in der ABox sind damit nicht mehr nur benannte Kanten im RDF-Graph, sondern drücken die Zuweisungen für die Rollen in der OWL-Semantik aus. Dies ist nur möglich, weil dazu die Abbildung von OWL auf RDF-Graphen ausgenutzt wird, wie in Abschnitt 3.4.2 beschrieben.

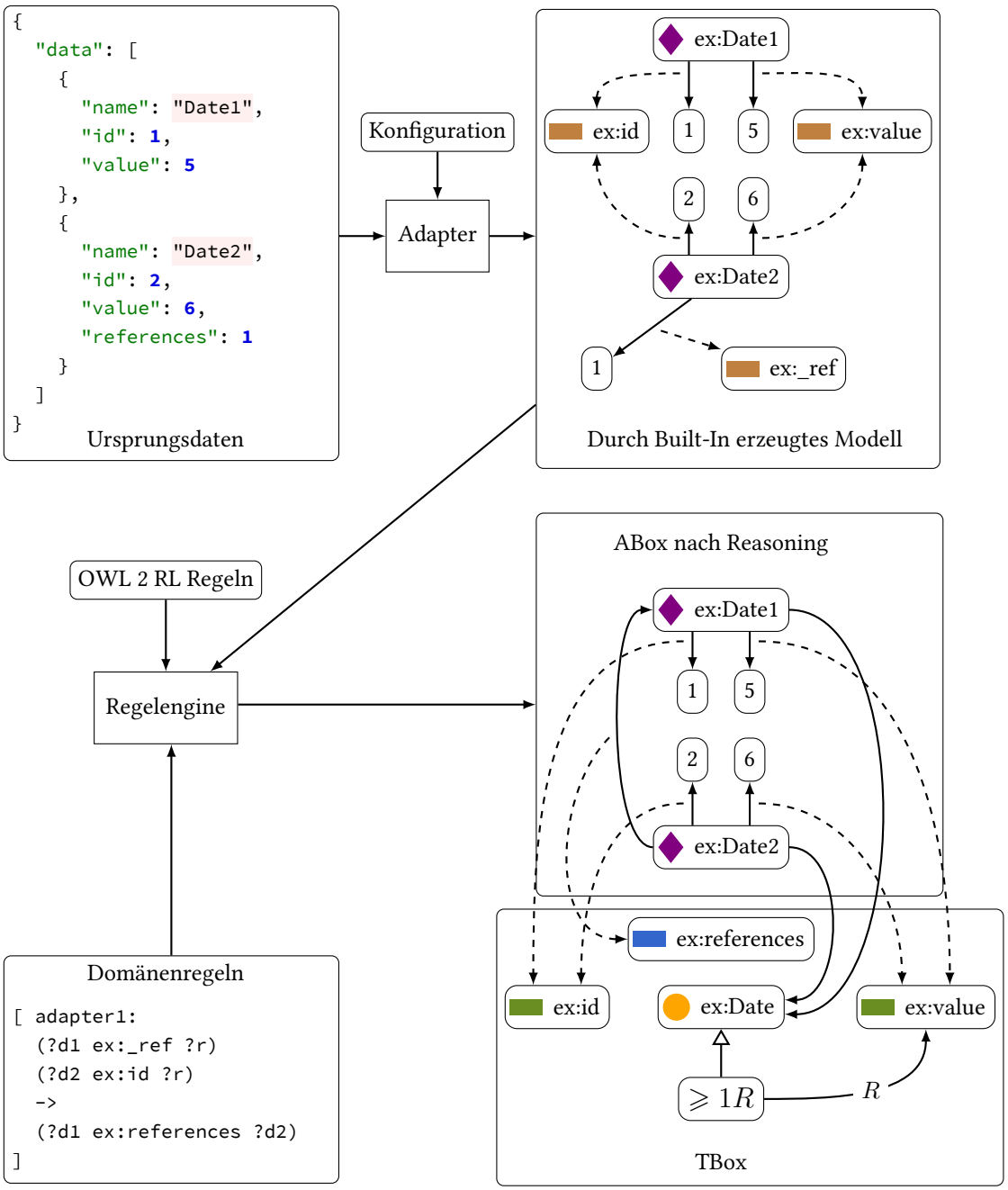

Abbildung 5.6: Arbeitsweise von Built-Ins und Domänenregeln. Die Eigenschaft `ex:_ref` ist in diesem Beispiel eine transiente Entität.

Die programmatische Entfernung der transienten Entitäten nach dem Reasoning ist eine billige Operation, wenn sie anhand des Musters ihrer Bezeichner vorgenommen wird. Die Entfernung kann daher so umgesetzt werden, dass kein neuer Reasoning-Prozess gestartet wird.

Schließlich liest die Komponente zur Nachverarbeitung in Schritt ④ in Abbildung 5.5 die Ontologie nach dem Reasoning und die Nachverarbeitungsvorschriften. Die Ergebnisse dieser Auswertung werden in Schritt ⑤ zurück in die Domänenontologie geführt. Die Nachverarbeitungsvorschriften können einerseits nicht auf Ebene der OWL-Semantik die Ontologie betrachten, sondern müssen den zugrunde liegenden RDF-Graphen lesen, um Aggregationen durchführen zu können, andererseits soll ihre Definition auf bestehenden Standards basieren. Dazu wird eine Kombination aus SPARQL zur Beschreibung der Aggregationen und OWL zur Verbindung der Aggregationen mit dem restlichen Modell vorgesehen.

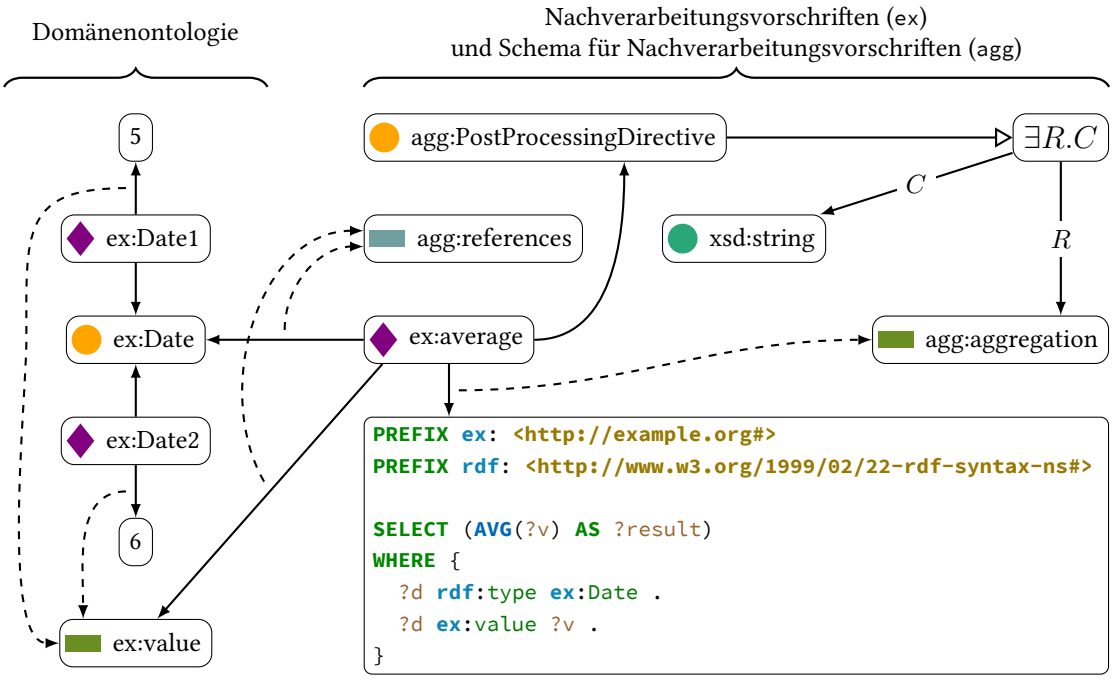

Abbildung 5.7: Definition von Nachverarbeitungsvorschriften

Beispiel: Abbildung 5.7 zeigt die Modellierung von Nachverarbeitungsvorschriften am Beispiel des Modells aus Abbildung 5.6. Im Beispiel wird der Mittelwert der Werte aller Wertzuweisungen von `ex:value` an Instanzen der Klasse `ex:Date` berechnet. Dazu existiert die Instanz `ex:average`, die die Aggregation repräsentiert und der über die konkrete Rolle `agg:aggregation` das SPARQL-Query zugewiesen ist.

Durch die Kombination einer Instanz und einer Wertzuweisung für das Query wird die Nachverarbeitungsvorschrift zu einem „First Class“-Objekt, das einen Bezeichner besitzt und da-

mit auch von anderen Stellen referenzierbar ist. Diese Struktur hat gegenüber dem Ansatz von SPIN [KHI11], in dem der Abstract Syntax Tree (AST) eines SPARQL-Queries direkt im RDF-Graph eingefügt wird, mehrere Vorteile. Erstens wird für die Auswertung des Queries keine Software benötigt, die die RDF-Repräsentation des AST in eine für die SPARQL-Engine auswertbare Speicherrepräsentation übersetzt; der in jeder SPARQL-Engine-Implementierung enthaltene Parser ist ausreichend. Zweitens kann auf diese Weise das Query nicht nur in reinen RDF-Dokumenten, sondern auch in als RDF serialisierten OWL-Ontologien eingefügt werden. In der RDF-Repräsentation von OWL-Ontologien dürfen nicht beliebige RDF-Konstrukte eingefügt werden, wenn dadurch ungültiges OWL entsteht, was bei SPIN der Fall ist. Drittens können Queries auf diese Weise auch ohne Software zum Dekodieren des AST in beliebigen RDF-, OWL oder auch Texteditoren bearbeitet werden. Der Nachteil ist, dass nicht bereits zum Zeitpunkt des Parsens des OWL-Dokuments, in dem die Nachverarbeitungsvorschrift enthalten ist, syntaktisch ungültige Queries erkannt werden, sondern Queries müssen in einem zweiten Schritt mittels eines SPARQL-Parsers überprüft werden. Da die Queries aber vor ihrer Auswertung ohnehin mindestens syntaktisch überprüft werden müssen, fällt dieser Nachteil in der Praxis nicht ins Gewicht.

Der Instanz, die die Nachverarbeitungsvorschrift repräsentiert, können neben dem Query beliebige weitere Referenzen auf andere Modellelemente zugeordnet werden. Diese Eigenschaft wird für die folgenden vier Techniken genutzt:

1. Durch die Annotationsrolle `agg:references` werden Referenzen innerhalb des Queries auf Entitäten der Domänenontologie auf Modellebene explizit gemacht. Dies macht solche Zusammenhänge auch abfragbar, ohne zunächst den `agg:aggregation`-Wert parsen zu müssen. Abfragen dieser Art sind zur Visualisierung der Vorschriften hilfreich.

2. Der Instanz kann über die abstrakte Rolle `agg:dependsOn` eine Referenz auf eine weitere Instanz einer Nachverarbeitungsvorschrift zugeordnet werden. Hierdurch lassen sich Abhängigkeitshierarchien ausdrücken, die die Reihenfolge der Auswertung der einzelnen Vorschriften steuern.

3. Der Instanz kann über die Annotationsrolle `agg:target` eine Instanz oder eine Klasse aus der Domänenontologie zugeordnet werden, der über eine Wertzuweisung das Ergebnis des Queries zugewiesen werden soll. Dazu wird der Variablennamen `result` in Queries reserviert. Die Zielrolle wird der Instanz durch eine Referenz vom Typ `agg:targetProperty` zugewiesen. Diese ist wie `agg:target` eine Annotationsrolle, da sie keine Relation zwischen Instanzen darstellt.

4. Ein Query kann die Variable `$this` enthalten und die Instanz, der dieses Query zugeordnet ist, über die Annotationsrolle `agg:context` die Referenz auf eine Instanz aus der Domänenontologie enthalten. Die Eigenschaft der SPARQL-Syntax, dass Variablen sowohl

? als auch $ als Präfix haben dürfen, wird hier genutzt, um eine Konvention zur Auszeichnung von Dependency Injection (DI) zu etablieren: Vor der Auswertung des Queries wird in der SPARQL-Engine die Variable auf die referenzierte Domänenontologie-Instanz festgelegt. Im einfachen Fall, in dem der Kontext eine Instanz ist, könnte statt des DI-Mechanismus und der Verwendung von `$this` auch die IRI der Kontext-Instanz im Query direkt angegeben werden. Als Entität, die durch `agg:context` referenziert wird, kann jedoch auch eine Klasse angegeben werden. In diesem Fall wird das Query für jede Instanz ausgewertet, die dieser Klasse zugeordnet ist; `$this` wird dazu bei jeder einzelnen Auswertung neu auf die jeweilige Instanz gebunden. Auf diese Weise können einfach Nachverarbeitungsvorschriften einer Menge von Instanzen zugeordnet werden.

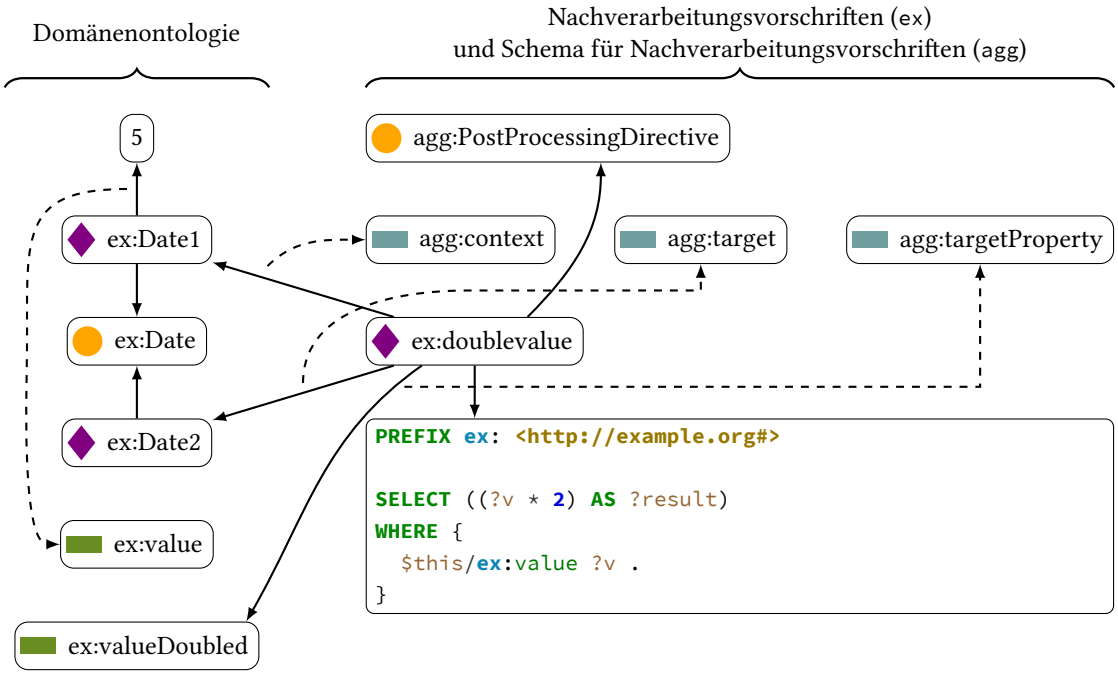

Abbildung 5.8: Dependency Injection in Nachverarbeitungsvorschriften

Beispiel: Ein Beispiel, das den Einsatz von `agg:target`, `agg:context` und Dependency Injection einsetzt, wird in Abbildung 5.8 gezeigt. Dabei wird der Instanz `ex:Date2` das Attribut `ex:valueDoubled` zugewiesen mit einem Wert, der dem doppelten `ex:value` von `ex:Date1` entspricht. Das SPARQL-Query nutzt dazu die Kurzschreibweise eines Property Path.

Statt des hier demonstrierten DI-Mechanismus ließe sich für die Generierung der Wertzuweisung ein ähnliches Ergebnis auch durch den Einsatz von SPARQL `CONSTRUCT`-Queries erreichen. Hierbei müssten allerdings alle referenzierten Entitäten innerhalb des Queries beschrieben werden; eine Änderung des Kontextes wäre weniger flexibel, eine unveränderte Nutzung

des Queries in mehreren Kontexten wäre nicht möglich. Darüber hinaus lassen sich durch die Verwendung von SELECT-Queries in der Implementierung des Laufzeitsystems leichter Optimierungen durch Memoisation oder Caching vornehmen.

Für die Modellierung und Auswertung von Nachverarbeitungsvorschriften wird also ebenso wie in der Arbeitsweise der schnittstellenspezifischen Adapter die Abbildung von OWL auf RDF-Graphen ausgenutzt. Für die Realisierung des Laufzeitsystems muss daher sichergestellt werden, dass die eingesetzte Technik (Bibliotheken zur Verarbeitung von RDF/OWL) die notwendigen Abstraktionen dazu bieten. Bei der Erstellung von Domänenontologien, Domänenregeln und Nachverarbeitungsvorschriften müssen ebenfalls die RDF-OWL-Abbildung sowie die Semantiken der beiden Sprachen berücksichtigt werden.

6 COBIT-Ontologie

6.1 Methodik und Abdeckung des Rahmenwerks

Um die Anforderung **F1** (*Formales Modell für das Rahmenwerk COBIT*) zu erfüllen, wird eine Ontologie entworfen, die als Referenzmodell dienen kann, um Domänenontologien unterschiedlicher technischer und nichttechnischer Sichten verknüpfen zu können. COBIT als Rahmenwerk erfüllt diesen Anspruch, stellt jedoch keine formale Version zur Verfügung. Die dem Entwicklungsprozess resultierende Ontologie soll im Laufzeitsystem einsetzbar sein, das in Abschnitt 5.2.3 beschrieben wurde. Der Prozess umfasst die Schritte der Auswahl einer geeigneten Entwicklungsmethodik, der Auswahl einer Strategie der Konzeptualisierung und schließlich der Anwendung der einzusetzenden Entwicklungsmethodik.

Als Ausgangsbasis für die Gegenüberstellung von Entwicklungsmethodiken des Knowledge Engineering und Ontology Engineering in Hinblick auf den Grad ihrer Einsetzbarkeit für die Entwicklung der COBIT-Ontologie muss zunächst die Kategorie der Ontologie etabliert werden. Von den Kategorisierungen von Ontologie-Typen, die in Abbildung 3.12 gegenübergestellt werden, entspricht die Kategorisierung 3, d.h., eine Unterteilung in Foundational Ontology, Core Ontology und Domain Ontology, am ehesten der Granularität, in der Ontologien allgemein entwickelt werden. Die COBIT-Ontologie soll als Referenz dienen, die explizit unterschiedliche Domänen miteinander verbindet, ist aber kein Modell zur Beschreibung generischen allgemeingültigen Wissens, somit kann sie nicht als Foundational Ontology oder Domain Ontology angelegt werden. Obwohl die Umsetzung als Unified Ontology denkbar ist, soll sie nicht von einer spezifischen Foundational Ontology oder einer Untermenge davon abhängen, damit sich die Ontologie durch Formulierung entsprechender Abbildungen auch nachträglich in Kombination mit einer oder mehrerer Foundational Ontologies einsetzen lässt. Durch die Beschränkung auf die Abstraktionsebene einer Core Ontology können solche Abbildungen leichter entwickelt werden, ohne dass Inkonsistenzen entstehen, oder, was wahrscheinlicher ist, unerwünschte Inferenzen entstehen. Der Verzicht, die Semantik der Konzepte in der COBIT-Ontologie auf einer bestehenden Foundational Ontology zu basieren, schließt dabei nicht die Einbindung und Nutzung existierender Vokabulare aus.

Allen Entwicklungsmethoden für Ontologien, die in Abschnitt 3.6 vorgestellt werden, liegt eine ähnliche Struktur in drei Schritten zugrunde:

- *Spezifikation* - Es wird festgelegt, welche Anforderungen bestehen, welchen Umfang die Ontologie haben soll und durch welchen Formalismus sie implementiert werden soll,
- *Konzeptualisierung* - Der Prozess, in dem die zu beschreibenden Konzepte erarbeitet werden, und
- *Implementierung* - Die Beschreibung der Konzepte mittels des vorgesehenen Formalismus sowie Test gegenüber den Anforderungen.

Zusätzlich kann eine Rückkopplung zwischen den Einzelschritten vorgenommen werden, die für inkrementelle Entwicklung genutzt wird. Die Struktur entspricht im Wesentlichen auch dem Prozess der Wissenserfassung von Jackson, wie in Abbildung 3.13 gezeigt, wobei bestimmte Aspekte von moderneren Methodiken noch ergänzt und erweitert wurden. Diese Vorgehensstruktur ist gut begründeter Konsens und soll für die COBIT-Ontologie als Grundlage dienen. Dabei wird sie ergänzt durch diejenigen Aspekte aus unterschiedlichen Vorgehensweisen, die hier als relevant eingestuft werden, denn keine der Vorgehensweisen berücksichtigt alle Aspekte. Auf die wichtigsten dieser zusätzlichen Aspekte und ihre Relevanz für diese Arbeit soll hier knapp eingegangen werden.

- Mehrere Methodiken machen konkrete Vorgaben über Details der Zwischenschritte und Zwischenergebnisse des Entwicklungsprozesses. Ein Beispiel dafür ist der Knowledge-Meta-Prozess nach Sure in Abbildung 3.15. Solche Vorgaben werden hier im Folgenden als Empfehlungen betrachtet, da nicht alle Zwischenschritte und -ergebnisse erforderlich für die Entwicklung der COBIT-Ontologie sind. So werden hier beispielsweise KADS-Worksheets nicht als erforderlich eingestuft, während der Zyklus zwischen der Erstellung einer halbformalen Ontologiebeschreibung, ihrer Formalisierung und einer Evaluationsphase als sinnvoll betrachtet wird.
- Der zweite Aspekt ist die Berücksichtigung von Vorgehensweisen, die in der Softwareentwicklung etabliert sind und für die Ontologie-Entwicklung auch eingesetzt werden sollen: Entwickeln von Anforderungen, Dokumentation, Configuration Management und Testen des Ergebnisses. Ein bekannter Vertreter einer Methodik, die Argumente hierfür liefert, ist Methontology von Gómez-Pérez et al. [GFV96, FGPP99, FGJ97, Gó98]. Diese Punkte sind für die Entwicklung der COBIT-Ontologie ebenfalls naheliegend und sinnvoll und werden umgesetzt, soweit dies möglich ist.
- Die Einbeziehung von Entwicklung durch geographisch verteilt arbeitende Teams oder Communities wurde zuerst in DILIGENT von Pinto et al. [PST04, PTS09] beschrieben, aber auch durch Melting Point von Garcia et al. [GOG$^+$10] beeinflusst. Der Aspekt ist für diese Arbeit nicht maßgeblich, kann aber in einer zukünftigen Weiterentwicklung der Ontologie Berücksichtigung finden, die idealerweise parallel zur Entwicklung der COBIT-Spezifikation stattfindet.

- Zur initialen Festlegung des Umfangs und groben Inhalts der Ontologie stellen Grüninger und Fox in der TOVE-Methotik [GF95] sogenannte *Competency Questions* (CQ) vor, die u.a. auch von De Nicola et al. für UPON [DMN09] adaptiert werden. CQs sind natürlichsprachliche Fragen auf konzeptueller Ebene, die durch entsprechende Abfragen der Ontologie beantwortet werden sollen. Durch einen separaten Prozess müssen die CQs und mögliche Abfragen an die Ontologie so aneinander angenähert werden, dass sie auch auswertbar sind, da nicht beliebig komplexe Abfragen in jedem Formalismus möglich sind. Die Resultate solcher Abfragen können Mengen von Instanzen sein (vergleichbar mit SQL-Abfragen), aber auch Konzepte wie Klassen oder Rollen umfassen. Bei der Entwicklung von Domänenontologien werden CQs in der Regel durch Interviews mit Domänenexperten und Nutzern der geplanten Ontologie erarbeitet. Bei der Entwicklung einer Referenzontologie, die auf einem bestehenen Standard basiert, können CQs aus der Spezifikation abgeleitet werden, sofern diese umfassend genug ist. Der Entwicklungsgrad der Ontologie soll dann mit der Menge der beantwortbaren CQs korrelieren.

Für den wesentlichen Schritt der Konzeptualisierung wird zunächst eine geeignete Strategie gewählt, wie in Abschnitt 3.6.2 beschrieben. Die *top-down*-Strategie, bei der mit den generischsten Konzepten begonnen wird, ist für diese Arbeit nicht geeignet, da durch die Analyse des Rahmenwerks zunächst konkrete Konzepte herausgestellt werden; eventuell vorhandene Verallgemeinerungen müssen erst erarbeitet werden. Der Einsatz der *middle-out*-Strategie ist zwar denkbar, müsste aber auch auf entsprechenden Vorarbeiten basieren. Eine mögliche thematische Unterteilung der zuerst herausgearbeiteten Kernkonzepte ist für die COBIT-Ontologie nicht vorteilhaft, weil die COBIT-Spezifikation auf einen hohen Grad von Integration zwischen allen darin beschriebenen Bestandteilen ausgelegt ist. Es ist also wenig sinnvoll, die Ontologie in Einzelteile anhand logischer Grenzen zerlegen zu wollen, die nicht schon in der Spezifikation beschrieben sind. Aus diesem Grund wird hier die *bottom-up*-Strategie verfolgt: Beschriebene Konzepte der Spezifikation werden erfasst und anschließend, wo möglich und sinnvoll, verallgemeinert.

Die meisten Ontology Engineering Methodiken machen für den Schritt der Konzeptualisierung keine Detailvorgaben, er kann nach wie vor eher als „handwerklicher“ Prozess verstanden werden. Daher werden die in Abschnitt 3.6.4 vorgestellten Best-Practice-Hinweise für den Schritt hier mit Hinblick auf ihre Anwendbarkeit betrachtet:

- Von den Grundprinzipien für das Design von Ontologien von Gruber [Gru03] werden hier Klarheit, Erweiterbarkeit und Kohärenz für die Konzeptualisierung übernommen. Minimale Voreingenommenheit für die Kodierung und minimale ontologische Festlegung sind nicht anzuwenden, weil als Zielformat der Formalisierung bereits OWL und die verwandten Technologien festgelegt wurden (siehe Abschnitt 4.4). Insbesondere die Verfolgung der minimalen ontologischen Festlegung soll damit nicht berücksichtigt werden.

- Nach einer Evaluierung der OntoClean-Methode von Guarino und Welty lassen sich mehrere Ergebnisse feststellen: Obwohl die Methode vor der Spezifizierung von OWL entwickelt wurde, ist sie auch beim Entwurf von OWL-Ontologien anwendbar, indem Meta-Eigenschaften als Annotationsrollen definiert und an den auszuzeichnenden Elementen vermerkt werden können. Allerdings muss die Auswertung der Einschränkungen, die für die Meta-Eigenschaften beschrieben sind, entweder manuell erfolgen oder durch zu diesem Zweck zu entwickelnde spezifische Software-Werkzeuge geschehen. Dies ist nicht mit einem OWL-Reasoner möglich: Semantik, die die Modell- und Meta-Modellebene in dieser Form verbindet, würde in OWL 2 Full liegen und wäre damit unentscheidbar. Darüber hinaus ist die Methode besonders für die Entwicklung von Upper Ontologies geeignet, die einerseits zahlreiche Konzepte in der Breite abdecken und andererseits durch uneindeutige oder widersprüchliche Aussagen entstandene Zyklen in der Vererbungshierarchie der Konzepte auflösen müssen. Derartige Zyklen sind kein prinzielles Problem und nicht vergleichbar mit Mehrfachvererbung in der objektorientierten Programmierung, können aber in diesen Fällen zu nicht-intuitiven bzw. der *angedachten* Bedeutung von Konzepten widersprechenden und damit letztlich falschen Schlussfolgerungen durch einen Reasoner führen. Bei der Konstruktion der COBIT-Ontologie sind die Voraussetzungen für solche Problemfälle nicht zu erwarten, da keine Upper Ontology entwickelt werden soll und derartige Inkonsistenzen im Informationsmodell eines etablierten Rahmenwerks wie COBIT äußerst unwahrscheinlich sind. Aus diesen Gründen wird OntoClean für die Entwicklung der Ontologie nicht eingesetzt.
- Entwurfsmuster für die Formulierung wiederkehrender Modellierungsaufgaben sind das Hilfsmittel, das am ehesten einen direkten Einfluss auf die Konzeptualisierung hat. Nicht alle Kategorien von Entwurfsmustern, die in Abschnitt 3.6.4 beschrieben wurden, haben den gleichen Grad von Wichtigkeit für die Entwicklung der COBIT-Ontologie, beispielsweise sind linguistische Strukturen hier unerheblich. Die wichtigste Kategorie ist die der inhaltlichen Muster (*Content Patterns* (CP)). CPs sind am ehesten zu vergleichen mit Software Design Patterns, die für bestimmte Modellierungssituationen eine geeignete, wiedererkennbare Funktion übernehmen. Es können sowohl domänenspezifische als auch domänenunabhängige CPs beschrieben werden. Der Einsatz von Entwurfsmustern, insbesondere CP, wird hier als nicht zwingend erforderlich aber hilfreich eingestuft. Es sollte im Einzelfall mindestens in der Literatur recherchiert werden, ob ein Entwurfsmuster existiert, das für den gewünschten Zweck anwendbar ist und zu welchem Grad. Im zweiten Schritt kann dann entschieden werden, ob das Muster unverändert angewendet wird, einzelne Strukturen oder Ideen übernommen werden oder eine eigene Lösung gefunden werden muss. Hier besteht ein Unterschied zwischen Entwurfsmustern für Software und solchen für Ontologien: Für die Struktur von Software wird auch beim Einsatz der selben, benannten Muster jeweils eine eigene Implementierung vorgenommen, die im Namensraum der entwickelten Sofware abgelegt wird, während Entwurfsmuster für Ontologien in OWL formalisiert und durch IRIs referenzierbar werden können. Dadurch kann ein

Katalog von Mustern erstellt werden, die als Bausteine direkt eingebunden werden können. Der größte, öffentlich zugängliche Katalog von Entwurfsmustern für Ontologien wird von der „Association for Ontology Design & Patterns“ (ODPA) gepflegt, die sich hauptsächlich aus Wissenschaftlern zusammensetzt, die sich mit dem Thema auseinandergesetzt haben; er ist unter [Ass08] zu finden. Im Folgenden wird an den Stellen des Entwicklungsprozesses, an denen Entwurfsmuster sinnvoll angewendet werden können, ein Abgleich mit diesem Katalog vorgenommen und vorhandene Muster für den Einsatz evaluiert.

Vor der Formalisierung des Rahmenwerks muss festgestellt werden, welche Teile von COBIT analysiert und in die Ontologie mit aufgenommen werden sollen. Wie in Abschnitt 2.2 geschildert, sind als übergeordnete Elemente der IT-Governance fünf Prinzipien und sieben Enabler festgelegt. Diese sind grundlegend für das Verständnis und die angedachte Anwendung von COBIT als Sammlung von Best Practices der IT-Governance. Für die Formalisierung und den Einsatz als Referenzmodell zur Verknüpfung von Domänenontologien sind die Vorgehensbeschreibungen allerdings zweitrangig. Diese Beschreibungen sind auf einer hohen Abstraktionsebene angelegt und sind eher Richtlinien oder Unterstützung als abzuarbeitende Prozessvorschriften. Aus den Prinzipien und den detaillierten Angaben zu ihrer Umsetzung, die den großen Teil der Spezifikation ausmachen, lässt sich dagegen das zugrunde liegende Informationsmodell herleiten. Dieses ist, wie bereits erwähnt, im Wesentlichen implizit enthalten, ist aber der wichtigste Bestandteil von COBIT für den Einsatz als Referenz. Der Fokus der Analyse richtet sich daher auf dieses Informationsmodell. In Abbildung 2.3 wurden die grundlegenden Elemente des Informationsmodells und die Zusammenhänge zwischen ihnen vorgestellt, die sich bereits aus den einführenden Beschreibungen der Spezifikation ergeben. Darüber hinausgehende Details über Konzepte und Instanzen müssen extrahiert werden. Außerdem wurde in Abbildung 2.4 das Prozessbefähigungsmodell von COBIT vorgestellt, dieses soll ebenfalls in der Ontologie umgesetzt werden.

6.2 Entwicklung des Schemas

Da die COBIT-Spezifikation aus einer Reihe von Dokumenten besteht, die teilweise unterschiedliche Zielgruppen und Einsatzzwecke haben (siehe Abbildung 2.2), erstrecken sich Teile des Informationsmodells in variierendem Detaillierungsgrad über mehrere der Handbücher und Umsetzungsleitfäden. Dies beinhaltet nicht nur ergänzende Informationen wie zusätzliche Eigenschaften oder Beschreibungen, sondern betrifft in Teilen selbst die Definition von Begriffen zur Benennung bestimmter Strukturen. Aus diesem Grund müssen mindestens die Basis-Dokumente untersucht werden, um zu einer über das gesamte Rahmenwerk hinweg konsistenten Benennung von Konzepten zu gelangen. Ein zentrales Element von COBIT sind die Prozesse, da sie die Verbindung zwischen den Fragen aus Sicht der Betriebsführung und der Umsetzung auf IT-Seite bilden. Nebem dem Hauptdokument von COBIT [AdG$^+$12a], das die

übergeordneten Zusammenhänge beschreibt, ist der größte Teil des Informationsmodells, der die Prozesse im Detail beschreibt, im Enabler-Handbuch *COBIT® 5: Enabling Processes* zu finden [AdG+12c]. Dieses Dokument ist daher neben dem Hauptdokument das wichtigste der Basis-Dokumente.

Die Mapping-Dokumente sind besonders hilfreich, wenn die Zusammenhänge zwischen COBIT und anderen, externen Standards ebenfalls formalisiert werden sollen. In Abschnitt 4.3 vorgestellte Ansätze, die beispielsweise ITIL als Ontologie formalisieren, können unter Zuhilfenahme der Mapping-Beschreibung auch auf Ontologie-Seite mit COBIT verbunden werden. Dazu werden zuerst die Informationsmodelle beider Ausgangsstandards (in diesem Fall COBIT und ITIL) als Ontologie formalisiert, anschließend wird das Mapping selbst als eine dritte Ontologie umgesetzt. Dies bietet Möglichkeiten zur Abfrage, die über die Zielsetzungen dieser Arbeit hinausgehen; erste Schritte zu einer solchen Umsetzung wurden in [TG15] beschrieben. Die Inhalte der Mapping-Dokumente sind allerdings zur Entwicklung des Kerns der COBIT-Ontologie selbst nicht besonders zu berücksichtigen, da sie keine Konzeptbeschreibungen enthalten, die in den Basis-Dokumenten nicht auch zu finden sind.

Die Ontologie kann hier grob in zwei Teilen betrachtet werden: Einerseits die Beschreibung der 37 Prozesse und den mit ihnen verbundenen Entitäten, die ihre Eigenschaften beschreiben, andererseits die Entitäten, die dem Informationsmodell seine Struktur geben. Dieser zweite Teil soll hier als *Schema* bezeichnet werden. Die Grenze zwischen Prozessbeschreibungen und Schema ist nicht identisch mit der Trennung zwischen ABox und TBox der Ontologie, denn beide Teile können den vollen Modellierungsumfang von OWL nutzen und damit sowohl Klassen als auch Instanzen beinhalten. Die Grenze soll der besseren Verständlichkeit in der Erläuterung dienen, hat aber keine Repräsentation in der Ontologie. Wegen seiner Funktion als Referenzmodell ist es ebenfalls nicht sinnvoll, das Schema und die Prozessbeschreibungen als zwei separate Ontologien zu entwerfen.

Für die Konstruktion der Bezeichner von Entitäten in der Ontologie werden hier die häufig genutzten Konventionen von Horridge [Hor11] angewendet: Bezeichner werden in CamelCase geschrieben; Klassennamen und Individuen beginnen mit einem Großbuchstaben; Rollennamen beginnen mit einem Kleinbuchstaben. Darüber hinaus gilt für abstrakte Rollen, dass der Bezeichner mit `is` oder `has` beginnen soll, je nach Bedeutung der Rolle. Bezeichner von konkreten Rollen oder Annotationsrollen sollen kein `has`-Präfix haben. Die Ontologie wird mit dem Namensraum `http://purl.org/atextor/ontology/cobit5` entwickelt, die vollen IRIs aller Elemente setzen sich daher durch die Verkettung dieses Namensraums mit einer Raute und den Bezeichnern nach den oben beschriebenen Regeln zusammen. Die Benennung von Entitäten, die in Abbildungen oder im Fließtext erwähnt werden, erfolgt in der Arbeit weiterhin durch die folgende Konvention: Soweit nicht anders angegeben, wird der oben genannte Namensraum bzw. seine CURIE `cobit5` angenommen. Die vollen IRIs von CURIEs, die sich auf andere Vokabulare oder Ontologien beziehen, sind in Anhang E aufgeführt.

Um Anforderung **F3** (*Identifizierbarkeit von Elementen der COBIT-Ontologie*) zu erfüllen, wird außerdem festgelegt, dass keine anonymen OWL-Individuen in der Modellierung eingesetzt werden sollen; alle Individuen sollen einen dem entsprechenden Namensschema folgenden Bezeichner besitzen. Anonyme Knoten auf RDF-Ebene ergeben sich automatisch durch die Serialisierung der OWL-Ontologie (z.B. bei Restriktionen), repräsentieren aber keine Entitäten und widersprechen daher nicht der Anforderung. Die Namensschemata für die einzelnen Typen von Entitäten werden im Folgenden bei deren Beschreibung festgelegt.

Der erste der Teil der Spezifikation, der in das Informationsmodell übernommen werden soll, findet sich bereits vor der Erläuterung der Enabler, in der Beschreibung des Prinzip 5, *Unterscheidung zwischen Governance und Management* ([AdG$^+$12a], Seite 31ff). Bei der Etablierung der Prozesskategorien (vergleiche auch Tabelle 2.1) werden die übergeordneten Kategorien „Processes for Governance of Enterprise IT“ und „Processes for Management of Enterprise IT“ festgelegt, in denen die restlichen Prozesskategorien zugeordnet werden ([AdG$^+$12a], Abbildung 16). Eine Besonderheit dabei ist, dass die Kategorie der Managementprozesse als Unterkategorie der Kategorie der Governance-Prozesse aufgefasst wird, was auch dem allgemeinen Verständnis der Begriffe entspricht. Damit entsteht für jeden Prozess eine Hierarchie von Kategorien, denen er zugeordnet sein kann. Abbildung 6.1 zeigt den Ausschnitt der Ontologie, der diese Zusammenhänge beschreibt.

Als Konzepte ergeben sich zunächst die Klassen der `ProcessCategory` und des `Process`. An dieser Stelle wird eine Designentscheidung getroffen, die Hierarchie von Prozesskategeorien nicht durch eine Klassenhierarchie zu repräsentieren, sondern durch eine Menge von Individuen. Hierdurch soll die Wahrnehmung der Kategorien als separate Entitäten unterstrichen werden, ermöglicht aber auch die Formulierung weiterer Zusammenhänge, die andernfalls so nicht möglich wären: `Processes` und `ProcessCategorys` können über die Rollen `hasProcess` bzw. ihre Inverse `hasProcessCategory` verbunden werden. Die Hierarchie der `ProcessCategorys` kann durch die transitiven Rollen `hasParentProcessCategory` und `hasSubProcessCategory` beschrieben werden; die Transitivität erreicht hier eine Semantik, die vergleichbar mit einer auf Vererbung basierenden Klassenhierarchie wäre. Zusätzlich kann `ProcessCategory` als eine geschlossene Klasse definiert werden, der nur die spezifizierten Kategorien angehören. Wird die Ontologie von Dritten im Rahmen einer unternehmensspezifischen Implementierung erweitert, würde eine fälschlicherweise zugefügte, d.h., der COBIT-Spezifikation widersprechende, `ProcessCategory` auch zu einer inkonsistenten Ontologie führen. Die Einführung von Klassen zur Repräsentation von weiteren, anwendungsspezifischen Kategorien von Prozessen ist dagegen problemlos möglich. Die Prozesskategorien folgen in COBIT keinem Namensschema, es wird daher auch für die Individuen in der Ontologie als Bezeichner der Name der Kategorie als Bezeichner verwendet.

Als Ausgangspunkt für die weiteren Zusammenhänge des Informationsmodells wird die *Goals Cascade* verwendet ([AdG$^+$12a], Seite 17ff). Darunter wird in COBIT die Abfolge von Einflüssen verstanden, die bei den Anforderungen der Anspruchsgruppen beginnen und sich bis zu

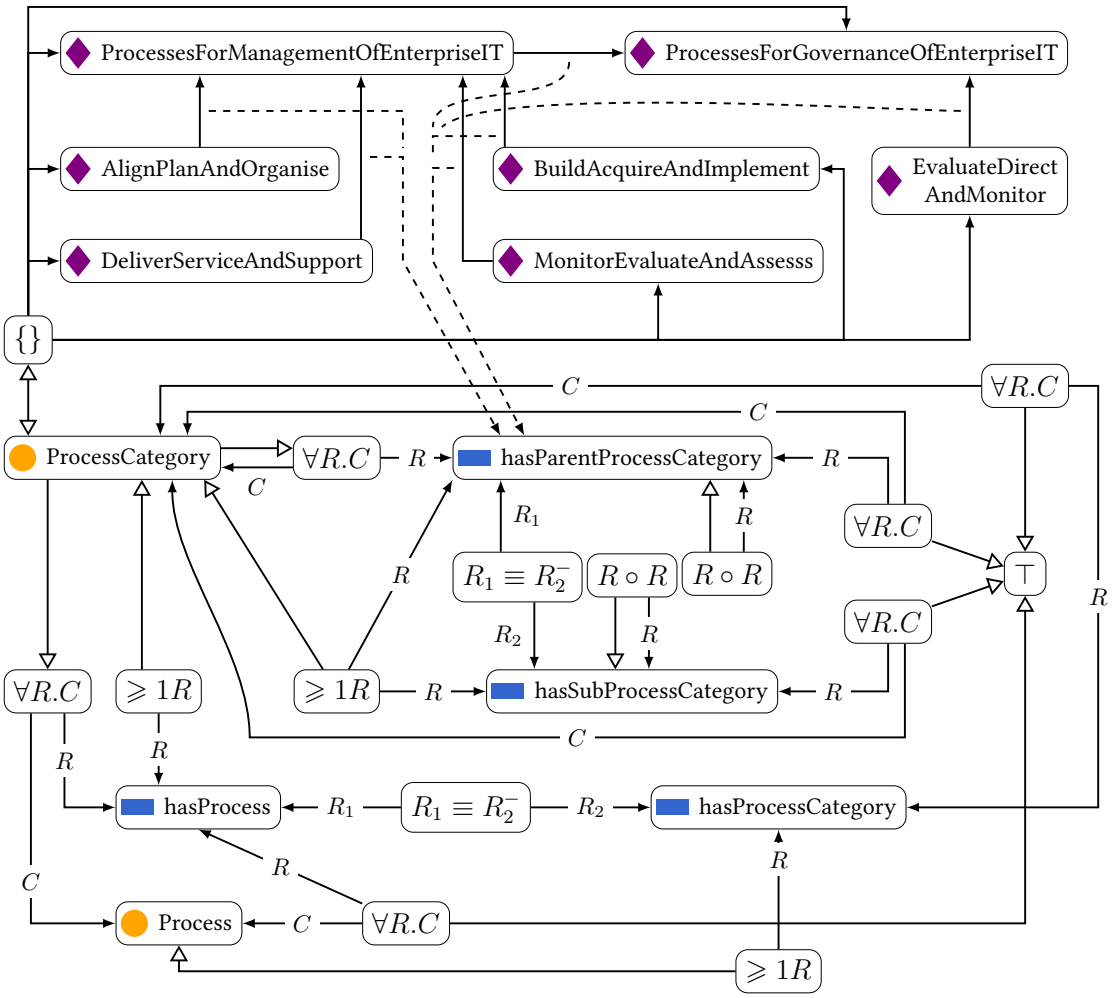

Abbildung 6.1: Zusammenhang von Prozesskategorien und Prozessen. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

den Zielen einzelner Enabler erstrecken. Diese besteht aus vier Schritten:

1. Anspruchsgruppen-Treiber beeinflussen Anspruchsgruppen-Anforderungen. Mit Treibern sind hier innere und äußere Veränderungen gemeint, beispielsweise „Strategieänderungen, veränderte[...] unternehmerische[...] oder behördliche[...] Bedingungen oder neue[...] Technologien“ ([AdG+12b], Seite 19).
2. Anspruchsgruppen-Anforderungen kaskadieren zu Unternehmenszielen.
3. Unternehmensziele kaskadieren zu IT-bezogenen Zielen.

4. IT-bezogene Ziele kaskadieren zu Enabler-Zielen.

Die konkrete Beschreibung von Entitäten in COBIT erfolgt erst beginnend mit Schritt 3 der Goals Cascade: Anspruchsgruppen-Treiber und Anspruchsgruppen-Anforderungen sind als Konzepte so generisch, dass es wenig sinnvoll ist, hierfür innerhalb des Rahmenwerks Vorgaben zu machen. Daher werden sie nicht in der Ontologie formalisiert, eine kontext- bzw. unternehmensspezifische Formalisierung und nachträgliche Verbindung mit der COBIT-Ontologie ist aber denkbar. Als Konzepte ergeben sich die Klassen `EnterpriseGoal` und `ITGoal`, sowie Instanzen der in der Spezifikation beschriebenen 17 konkreten Unternehmensziele und 17 IT-bezogenen Ziele. Die Ziele werden durch ihre laufenden Nummern aus dem Dokument referenziert und haben abgesehen davon keine eindeutigen Bezeichner. Für die Ontologie wird daher das Namensschema `EG`i für Unternehmensziele und `ITG`j für IT-bezogene Ziele eingesetzt, wobei i und j den jeweiligen laufenden Nummern entsprechen.

Wie Unternehmensziele von der Erfüllung von IT-bezogenen Zielen abhängen, ist in Form einer Matrix beschrieben ([AdG+12a], Seite 50), in der zwischen primären und sekundären Abhängigkeiten unterschieden wird. Die Goal Cascade in Schritt 4 erwähnt den weiteren Zusammenhang von IT-bezogenen Zielen und Enabler-Zielen, allerdings werden in der restlichen Spezifikation keine direkten diesbezüglichen Aussagen gemacht. Stattdessen werden die Zusammenhänge zwischen IT-bezogenen Zielen und einem spezifischen Typ von Enabler, den Prozessen, beschrieben; ebenfalls in Form einer Matrix ([AdG+12a], Seite 52), die primäre und sekundäre Abhängigkeiten unterscheidet. Für die Abhängigkeiten werden die abstrakten Rollen `isPrimarySupportFor` und ihre Inverse `hasPrimaryDependencyOn` eingeführt, sowie die analogen `secondary`-Gegenstücke. Diese erfassen die in der Spezifikation festgelegte abstrakte Ebene; spezifischere Versionen können in der Implementierung von konkreten Anwendungsfällen durch Unterrollenbeziehungen umgesetzt werden.

Entsprechend der Beschreibung des Prozessmodells im Enabler-Handbuch Enabling Processes besitzen Prozesse eigene Ziele. Daraus ergibt sich als Konzept die Klasse `ProcessGoal`. Jedem der drei Typen von Zielen (`EnterpriseGoal`, `ITGoal`, `ProcessGoal`) werden im Rahmenwerk zugehörige Metriken zugeordnet. Die Menge der Metriken jedes Ziels soll dazu dienen, den Erreichungsgrad des Ziels zu messen, COBIT spezifiziert aber nicht, wie die Messung im einzelnen vorgenommen wird. Daraus werden die Klassen `Metric` sowie ihre Unterklassen `EnterpriseGoalMetric`, `ITGoalMetric` und `ProcessGoalMetric`, und die zugehörige abstrakte Rolle `hasMetric` hergeleitet. Der obere Teil von Abbildung 6.2 zeigt die Klassen zur Beschreibung von Zielen, Metriken und die Zusammenhänge mit Prozessen.

Im unteren Teil von Abbildung 6.2 werden die wesentlichen Zusammenhänge von Konzepten gezeigt, die Prozesse und ihre Eigenschaften detailliert beschreiben. Diese Konzepte sind ebenfalls im Enabler-Handbuch Enabling Processes definiert. Zentrales Element im Aufbau von Prozessen sind die Governance-Praktiken bei den Prozessen der Kategorie EDM und die analog dazu verstandenen Managementpraktiken bei den Prozessen der anderen Kategorien. Die Praktiken repräsentieren einzelne Aufgaben, die auf dem Weg der Umsetzung des Prozesses

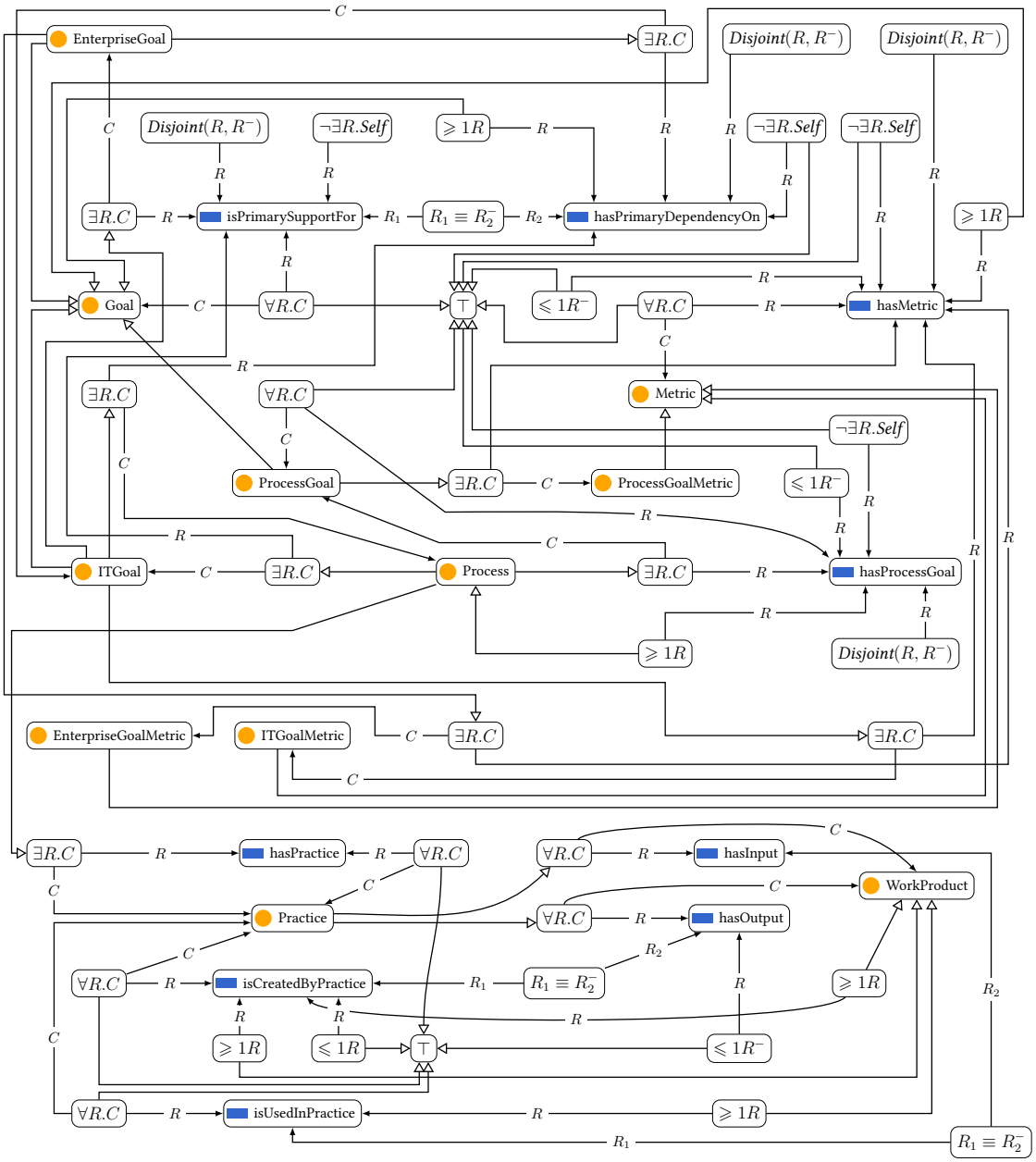

Abbildung 6.2: Zusammenhang zwischen Prozessen, Zielen und Managementpraktiken. Analog zu den abstrakten Rollen `isPrimarySupportFor` und `hasPrimaryDependencyOn` existieren auch `isSecondarySupport` und `hasSecondaryDependencyOn` mit analoger Semantik. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

auszuführen sind. In der Ontologie werden dafür die Klassen `Practice` und ihre Unterklassen `ManagementPractice` und `GovernancePractice` definiert (`ManagementPractice` und `GovernancePractice` werden nicht in der Abbildung gezeigt).

Jede einzelne der Praktiken hat weitere Eigenschaften. Um den Informationsfluss bei der Ausführung der Prozesse zu beschreiben, definiert COBIT für die Praktiken eine Menge von *Input*s und *Output*s. Dabei kann es sich um unterschiedliche Informationsfragmente handeln, wie Richtlinien, Dokumentation oder Modelle. Für Inputs und Outputs wird im COBIT-Hauptdokument und in Enabling Processes kein übergeordneter Begriff verwendet, im Umsetzungsleitfaden *Process Assessment Model* [BL13] wird dafür aber der Begriff Work Product verwendet, der daher in der Ontologie auch zur Benennung der Klasse `WorkProduct` eingesetzt wird.

Weitere Eigenschaften, die für Praktiken definiert sind, sind einerseits die sogenannten *Aktivitäten*, die in noch feinerer Granularität Durchführungsrichtlinien für die Praktik definieren und andererseits die Zuordnung von Verantwortlichkeiten. Für die Repräsentation von Verantwortlichkeiten werden in COBIT insgesamt 26 Rollen beschrieben, beispielsweise *CEO*, *COO* oder *Head of Human Resources*, und für jede Praktik durch eine RACI-Matrix als Verantwortlichkeit zugeordnet. RACI steht dabei für Responsible, Accountable, Consulted und Informed. Es wird hier nicht als vorteilhaft angesehen, zu versuchen, die Matrixstruktur in der Ontologie abzubilden, da sie selbst nur ein Hilfsmittel ist, um die Zuordnung verständlich aufzuzeigen. Stattdessen wird die Zurordnung durch die abstrakten Rollen `hasRoleConsulted`, `hasRoleAccountable`, `hasRoleInformed` und `hasRoleResponsible` ausgedrückt, für die jeweils eine Inverse Rolle `isXFor` existiert (nicht in der Abbildung gezeigt). Diese abstrakten Rollen werden also verwendet, um Individuen der Klasse `Practice` mit Individuen der Klasse `Role` zu verbinden, um damit die in der Spezifikation definierten Zuordnungen auf *Schema-Ebene* zu beschreiben, aber nicht, um eine Rollenzuordnung für Individiuen, die Personen oder konkrete Funktionseinheiten repräsentieren, vorzunehmen. Abbildung 6.3 zeigt die Konzepte in der Ontologie, die Praktiken, Aktivitäten, Rollen und die RACI-Matrizen umsetzen.

Für Rollen muss dabei eine Designentscheidung getroffen werden: Entweder werden Rollen als Unterklassen der Klasse `Role` beschrieben, oder als Individuen von `Role`. Die erste Strategie erlaubt es, Individuen, die konkrete Personen oder Funktionseinheiten repräsentieren, durch Zuordnung zu der jeweiligen Rollen-Klassen zu gruppieren. Die zweite Strategie verdeutlicht, dass Rollen als Konstrukte erster Klasse wahrgenommen werden, die selbst wieder Eigenschaften haben können. Es wird hier die zweite Strategie verwendet: Personen oder Funktionseinheiten werden hierbei durch Relationen ausgedrückt, außerdem können somit beliebige weitere, zumeist unternehmenspezifische Eigenschaften mit den Rollen-Individuen verbunden werden. `Role` ist, im Gegensatz zu beispielsweise `ProcessCategory`, eine offene Klasse, wodurch weitere Rollen bei Bedarf zugefügt werden können, ohne Inkonsistenzen zu erzeugen.

Für die Modellierung der Aktivitäten, die in COBIT jeder Management- und Governance-Praktik als Listen zugeordnet sind, wird in der Abbildung 6.3 die Klasse `Activity` gezeigt,

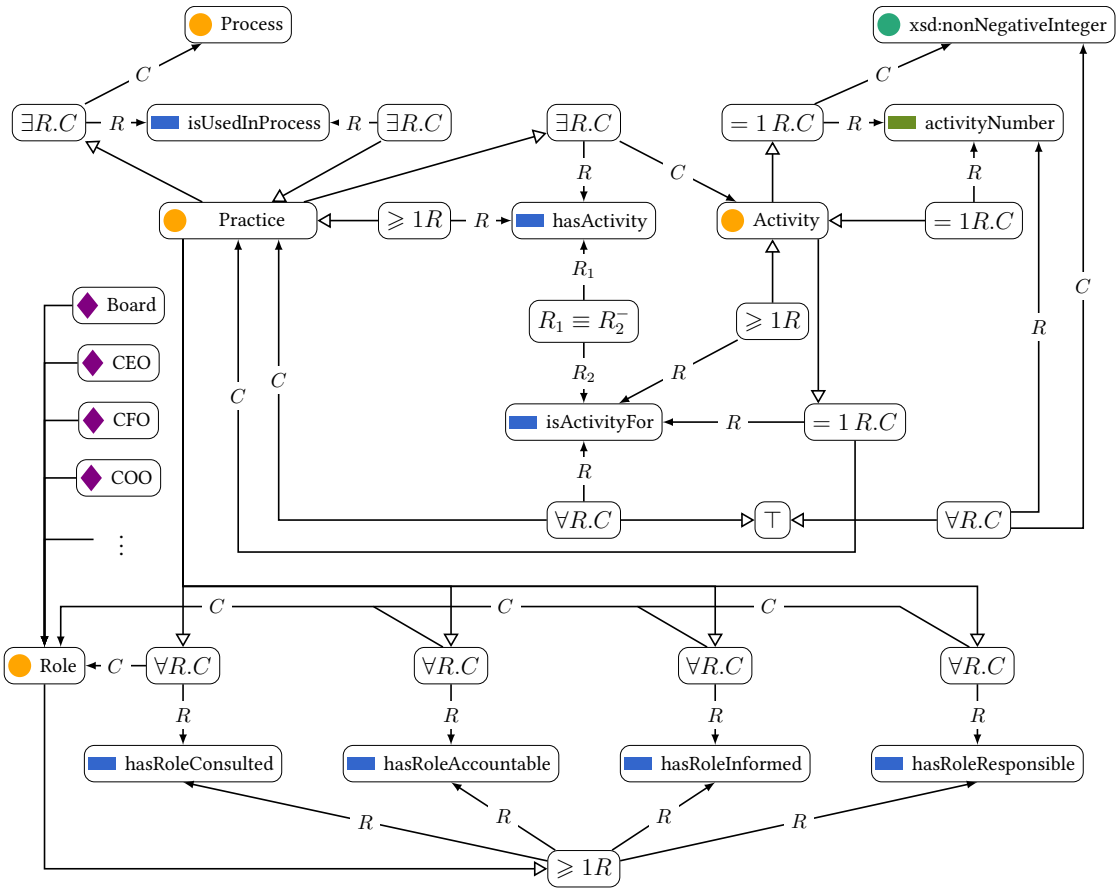

Abbildung 6.3: Zusammenhang zwischen Managementpraktiken, Rollen und Aktivitäten. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

die eine Kardinalitätsrestriktion auf die konkrete Rolle `activityNumber` besitzt. Die laufenden Nummern von Aktivitäten werden hierdurch zwar erfasst, semantisch ist aber eine Menge von Beziehungsaussagen zwischen Praktiken und Aktivitäten immer noch eine ungeordnete Sammlung. Im Gegensatz zu RDFS ist in der Spezifikation von OWL kein Sammlungstyp definiert, der eine Ordnung berücksichtigt (vergleiche Abschnitt 3.3.2). Um die Aktivitätslisten als solche zu erhalten, muss daher auf OWL-Ebene eine entsprechende Listenstruktur definiert werden. Die Semantik der Liste soll in der Ontologie beschrieben sein, Abfragen über die Liste müssen aber sowohl über einen Reasoner als auch auf der unterlagerten RDF-Ebene über SPARQL möglich sein.

Um dieses Problem anzugehen, existieren in der Literatur mehrere Ansätze, von denen keiner, wie sich herausgestellt hat, in unveränderter Form einsetzbar ist.

- Eine einfache Lösung hierzu besteht aus der Konstruktion von verketteten Knoten, analog zur Struktur von RDF-Listen, wie in Abbildung 3.9 gezeigt, allerdings unter Nutzung eines separaten Vokabulars. Wird eine OWL-Ontologie in einer RDF-Syntax serialisiert, werden RDF-Listen in einigen Typen von Axiomen genutzt, daher darf ein OWL-Dokument auf Modellebene keine RDF-Listen enthalten. `rdfs:Sequence` wäre erlaubt, ist aber für einen OWL-Reasoner unsichtbar. Dieser Ansatz wird beispielsweise für die Modellierung von Sequenzen in der OWL-S-Spezifikation [MBH+04] genutzt, einer Standard-Ontologie für Web Services. Diese Lösung ist allerdings problematisch, da der direkte Bezug zwischen Ausgangselement (hier: Praktik) und Listenelement (hier: Aktivität) verloren geht. Auf OWL-Ebene sind keine Abfragen oder Schlussfolgerungen möglich, die die Listensemantik berücksichtigen. Eine Abfrage über SPARQL ist zwar möglich, auch um einzelne Elemente der Liste zu selektieren, damit wird aber die Semantik nicht mehr in der Ontologie, sondern in der Abfrage festgelegt.
- Der ODPA-Katalog enthält Muster für die Beschreibung von Bags, Sequenzen [Gan10] und Listen [Cic10]. Dabei baut das Muster für Listen auf denen für Bags und für Sequenzen auf. Es entspricht in der Grundidee doppelt verknüpften Knoten, ähnlich des oben beschriebenen Ansatzes, erweitert die Semantik aber so, dass bei der Beschreibung einer Liste durch Listenknoten und ihre Content-Knoten durch den Reasoner geschlussfolgert werden kann, dass alle dem ersten Content-Knoten folgenden auch Elemente der Liste sind. Allerdings ist die Ontologie fehlerhaft definiert: Wird beispielsweise *Monday nextItem Tuesday* zugesichert, werden fälschlicherweise *Monday follows Tuesday* und *Monday directlyFollows Tuesday* daraus geschlussfolgert. Das Muster ist daher für den Einsatz in der COBIT-Ontologie unbrauchbar.
- Drummond et al. [DRS+06] stellen ebenfalls eine Ontologie für verkettete Listen vor, in der Elemente, die dem ersten Element transitiv folgen, durch einen Reasoner als Elemente der Liste zugeordnet werden können, also die Zugehörigkeit eines Elements zu der Liste abgefragt werden kann. Über SPARQL können diese Listen nur abgefragt werden, indem der konkreten Listenstruktur gefolgt wird; aggregierende Abfragen, beispielsweise zum Zählen der Elemente einer Liste, können nicht formuliert werden.
- Hoekstra entwickelt das Konzept von *Marker Properties* ([Hoe09], Seite 168ff), um Sequenzen in OWL zu modellieren: Zu jeder Klasse wird eine zugehörige abstrakte, reflexive Rolle definiert: zu Klasse `A` wird die Rolle `isA` definiert, zu Klasse `B` die Rolle `isB` usw. Die Sequenz kann dann durch eine komplexe Rolleninklusion der Form $isA \circ isB \circ \cdots \sqsubseteq sequenz$ beschrieben werden. Der Vorteil gegenüber dem Ansatz von Drummond et al. ist die geringere Anzahl von Individuen in der Ontologie, da die Modellierung von Listenknoten entfällt. Allerdings sind die so formulierten Listen über SPARQL überhaupt nicht abfragbar, da die Listenstruktur auf RDF-Ebene nicht sichtbar ist. Die Entkopplung der Semantik von der repräsentierenden Struktur kann für reine

OWL-Modellierung als Vorteil gewertet werden, macht die Lösung für den Einsatz im Laufzeitsystem aber ebenfalls unbrauchbar.

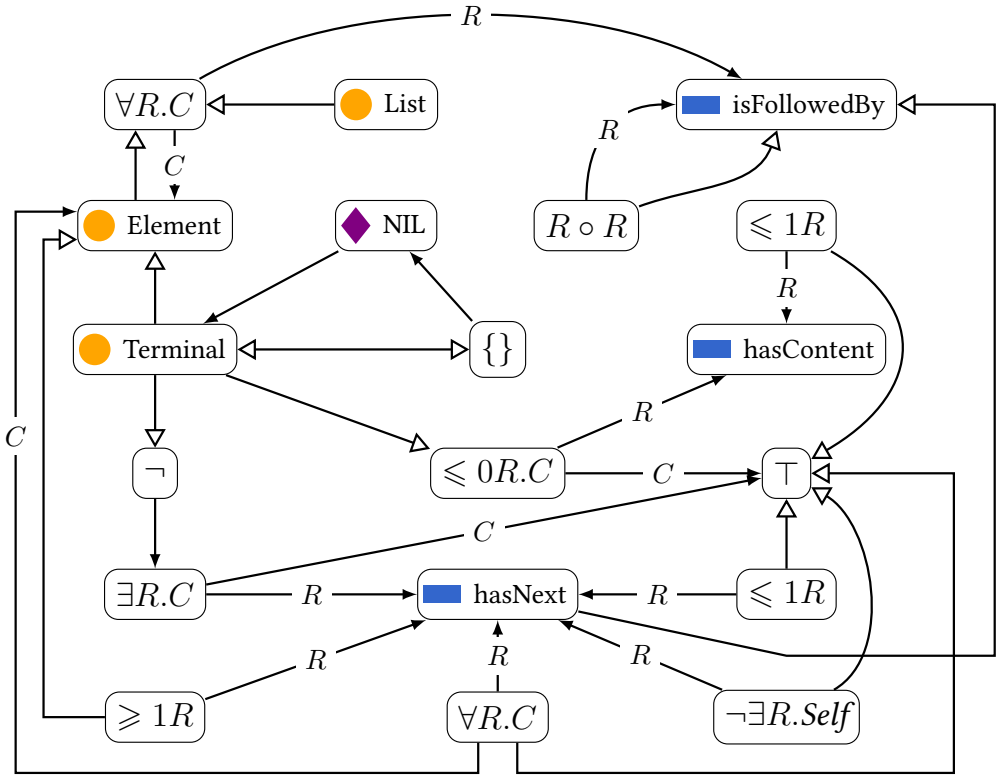

Abbildung 6.4: Modellierung von Listen in OWL. Der Standardnamensraum in dieser Abbildung ist `list`.

Die Lösung für die Modellierung von OWL-Listen in dieser Arbeit lehnt sich daher am Ansatz von Drummond et al. an und ist auch vergleichbar mit dem Listen-Muster aus dem ODPA-Katalog, berücksichtigt aber einerseits die korrekte Listensemantik auf OWL-Ebene und erlaubt andererseits auch die Abfrage mittels SPARQL, inklusive der Möglichkeit von Aggregationen der Listenelemente. Abbildung 6.4 zeigt die Struktur der Listen-Ontologie. Eine Liste wird hier im Gegensatz zum Ansatz von Drummond et al. als separate Klasse `List` modelliert, eine Listen-Instanz ist also nicht identisch mit der Instanz des ersten Listenknotens. Dies erlaubt es, über Unterklassen von `List` anwendungsspezifische Listen als Konzepte benennen zu können. Ein Listenknoten wird durch eine Instanz der Klasse `Element` ausgedrückt und bekommt über die abstrakte Rolle `hasContent` den Inhalt und über die abstrakte Rolle `hasNext` den folgenden Listenknoten zugewiesen. Das letzte Listenelement zeigt über `hasNext` stets auf das Terminalsymbol `NIL`; dies wird für Aggregationen genutzt. Die Rolle `hasNext` entspricht der Idee der Rolle `directlyPrecedes` aus dem ODPA-Listenmuster; `isFollowedBy` entspricht

der Rolle `precedes`.

Abbildung 6.5 zeigt den Ausschnitt der Ontologie, der das Prozessbefähigungsmodell beschreibt. Zur Umsetzung des Prozessbefähigungsmodells, dessen Struktur in Abbildung 2.4 vorgestellt wurde, werden als Kernelemente die Klasse `ProcessCapabilityLevel` für Befähigungsstufen und die Klasse `PerformanceAttribute` für Prozessattribute eingeführt. Beide sind geschlossene Klassen, die genau die Menge der entsprechenden Individuen umfassen. Als Namensschema für die Befähigungsstufen werden die Namen eingesetzt, die in der Spezifikation in dem Abschnitt verwendet werden, der das Maturity Model für Prozesse aus COBIT 4.1 mit dem Prozessbefähigungsmodell aus COBIT 5 vergleicht ([AdG+12a], Seite 44). Befähigungsstufen sind in COBIT textuell beschrieben und besitzen als eindeutigen Bezeichner eine Identifikationsnummer. Die Identifikationsnummer wird in der Ontologie durch die konkrete Rolle `capabilityLevel` und eine Existenz-Restriktion der Klasse `ProcessCapabilityLevel` ausgedrückt. Jeder Befähigungsstufe können zusätzlich Prozessattribute zugeordnet sein. Das Namensschema für die Prozessattribute wird aus ihrer Beschreibung übernommen ([AdG+12a], Seite 42): PA*i.j*, wobei *i* die Identifikationsnummer der Befähigungsstufe, der das Prozessattribut zugeordnet ist, und *j* die laufende Nummer des Prozessattributs ist.

Für eine Vereinfachung der Verwendung der COBIT-Ontologie als Referenz wird in der Ontologie die explizite paarweise Disjunktion aller Individuen festgelegt, indem ein `DifferentIndividuals`-Axiom mit allen beschriebenen Individuen als Argument zugefügt wird. Ausgenommen hiervon sind diejenigen Individuen, für die mittels `owl:sameAs` explizit eine Gleichheit festgelegt wurde. Vergleichbar mit der in Abschnitt 3.4.2 beschriebenen Open World Assumption folgt OWL nicht der sogenannten Unique Name Assumption (UNA), nach der automatisch unterschiedlich benannte Konzepte disjunkt sind, daher muss diese Aussage explizit zugefügt werden. Da `DifferentIndiviuals`-Axiome mit einer großen Anzahl von Argumenten in einigen Reasonern zu Performance-Problemen führen können, kann eine Entfernung dieses Axioms aus der Ontologie vor dem Einsatz im jeweiligen Reasoner nötig sein; dies muss im Rahmen des jeweiligen Einsatzes der Ontologie abgewogen werden.

6.3 Umsetzung der Prozesse

Eine weitere Designentscheidung betrifft die Modellierung der Prozesse. Hierbei lässt sich als Argument für eine Umsetzung der Prozesse in Form von Klassen anführen, dass dadurch beim Einsatz der COBIT-Ontologie eine Implementierung der Prozesse durch entsprechende Individuen ausgedrückt werden kann. Allerdings wären hierdurch die Prozesse entsprechend ihrer Beschreibung in COBIT keine unmittelbar wahrnehmbaren Entitäten. Komplexe Zusammenhänge, wie beispielsweise die primäre oder sekundäre Unterstützung bestimmter IT-Ziele, müssten in diesem Fall ausschließlich in Form von Klassenrestriktionen beschrieben werden. Dies würde wiederum bedingen, dass zwischen den Individuen der späteren Implementierung

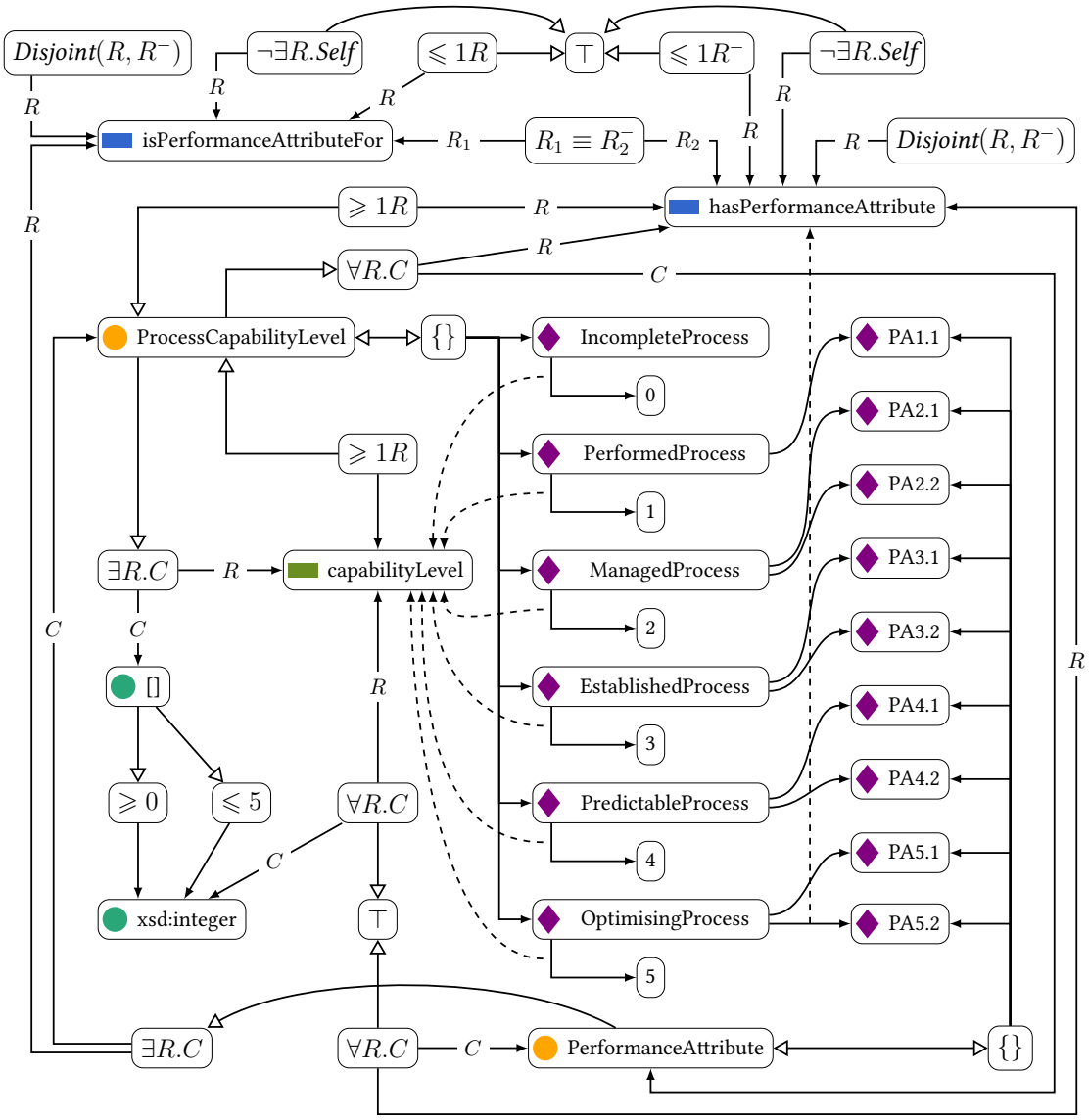

Abbildung 6.5: OWL-Formalisierung des COBIT Prozessbefähigungsmodells. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

vielfach die selben Beziehungsaussagen wiederholt werden müssten, die eigentlich Teil der Schemabeschreibung sein sollten, um zu dem Schema konsistent zu sein. Daher werden hier die spezifizierten Prozesse und alle zugehörigen Entitäten wie Ziele und Management- und Governance-Praktiken als Individuen modelliert.

Als Namensschema für die Prozesse wird die Benennung aus der Spezifikation übernommen, bei der das Kürzel der Prozesskategorie mit der laufenden Nummer des Prozesses verbunden wird, beispielsweise `APO01`. Die vollständige Liste der Prozesse und ihre Zugehörigkeiten zu Prozesskategorien kann im Anhang in Abbildungen B.1 und B.2 gefunden werden. Ebenso wird für die Individuen der Management- und Governance-Praktiken das Namensschema aus Enabling Processes übernommen, die erste Praktik des Prozesses `APO01` heißt daher `APO01.01`. Für die Aktivitäten, die den Praktiken zugeordnet sind, wird in COBIT kein Namensschema vorgegeben, da sie in Enabling Processes nur tabellarisch aufgelistet sind. Aus diesem Grund wird an dieser Stelle ein Namensschema definiert: Für die $k$te Aktivität der $j$ten Management- oder Governance-Praktik des $i$ten Prozesses der Kategorie C wird als Name $Ci.j.\mathtt{A}k$ festgelegt.

Work Products werden als Ein- und Ausgaben von Praktiken in Enabling Processes zwar gelistet aber nicht identifiziert. Eine eindeutige Benennung von Work Products wird nur in Process Assessment vorgenommen, wo ihnen eine sogenannte Work Product ID (WID) zugewiesen wird. Die WID hat die Form $Ci\text{-}\mathtt{WP}w$, wobei w eine laufende Nummer mit Geltungsbereich innerhalb des Prozesses ist. Dies wird als Grundlage des Namensschemas von Work Products in der Ontologie genutzt, ist allerdings hierzu nicht ausreichend, da die COBIT-Spezifikation an dieser Stelle inkonsistent ist: Neben kleineren Abweichungen in der Beschreibung der Work Products zwischen den beiden Dokumenten, die für die Identifikation unerheblich sind, existiert auch eine Menge von Work Products, die in Enabling Processes referenziert bzw. als Ein- und Ausgaben in Praktiken genutzt, in Process Assessment aber nicht benannt werden. Für diese Work Products wird daher ein Ersatz-Namensschema wie folgt definiert: $Ci\text{-}\mathtt{WPX}w\alpha$, wobei w die laufende Nummer des zuvor definierten Work Products ist und α ein Suffix aus $a, b, c, \ldots$. Als Ordnung wird die Definitionsreihenfolge in Enabling Processes beibehalten. Muss also zum Beispiel eine Ersatz-WID für ein Work Product vergeben werden, das in Enabling Processes zwischen den Work Products `APO06-WP10` und `APO06-WP11` erstmals verwendet, dem in Process Assessments aber keine WID zugeordnet wird, so wird dieses `APO06-WPX10a` benannt. Die Sonderfälle bei der Benennung von Work Products, die sich durch Inkonsistenzen in der COBIT-Spezifikation ergeben, sowie ihre Behandlung in der Ontologie werden in Tabelle 6.1 zusammengefasst.

Tabelle 6.1: Sonderfälle bei der Benennung von Work Products

Betrifft	**Beschreibung**	**Kommentar**
`APO06-WPX10a`	Operational procedures	Keine WID definiert
`BAI01-WPX11a`	Programme audit plans	Keine WID definiert
`BAI03-WPX2a`	SLAs and OLAs	Keine WID definiert

Tabelle 6.1: Sonderfälle bei der Benennung von Work Products (fortgesetzt)

Betrifft	Beschreibung	Kommentar
BAI09-WPX12a	Action plan to adjust licence numbers and allocations	Keine WID definiert
MEA02-WPX6a	Control deficiencies	Keine WID definiert
MEA02-WPX6b	Remedial actions	Keine WID definiert
MEA02-WPX6c	Results of assurance provider evaluations	Keine WID definiert
MEA02-WPX6d	High-level assessments	Keine WID definiert
MEA02-WPX6e	Assurance plans	Keine WID definiert
MEA02-WPX6f	Assessment criteria	Keine WID definiert
BAI03-WP16	Periodic maintenance analyses	In Enabling Processes nicht referenziert; `cobit5:isCreatedByManagementPractice` und `cobit5:isUsedInPractice` Beziehungsaussagen werden aus der BAI03-Beschreibung in [BL13], Seite 75 hergeleitet
APO09-WP4, BAI03-WP18	Updated service portfolio	Doppelt vergebene WIDs für gleiches Work Product; wird in der Ontologie durch `owl:sameAs` versachlicht

Der Modellierung von Verantwortlichkeiten folgend werden die einzelnen Rollen, die in COBIT beschrieben sind, ebenfalls als Individuen modelliert und jeder Management- und Governance-Praktik durch eine entsprechende Beziehungsaussage zugeordnet. Zusätzlich zu den Beziehungsaussagen zwischen `Practice` und `Activity` mittels der abstrakten Rolle `hasActivity`, deren Schema in Abbildung 6.3 gezeigt wurde, wird jeder Praktik eine Liste ihrer Aktivitäten zugeordnet. Hierzu wird das in Abbildung 6.4 vorgestellte Listenschema eingesetzt. Die Klasse `ActivityList` wird als Unterklasse von `List` definiert und für jede Aktivität wird ein Individuum angelegt, das eine `ActivityList` ist und dem Namensschema $Ci.j$.`Activities` folgt. Die Listenknoten werden analog zur jeweils als Inhalt referenzierten Aktivität nach dem Namensschema $Ci.j$.`A.N`k benannt. Ein Beispiel, das die Zusammenhänge zwischen dem Prozess 01 der Kategorie APO, seiner Managementpraktik 01 und der zugehörigen Aktivitäten zeigt, wird in Abbildung 6.6 veranschaulicht.

Für die Erfüllung von Anforderung **F2** (*Nachvollziehbarkeit der COBIT-Ontologie*) wird eine Möglichkeit benötigt, einen Bezug zwischen Entitäten, Relationen und dadurch entstehenden Strukturen zu den sie bedingenden Beschreibungen der Spezifikation herstellen zu können. Eine textuelle Dokumentation der Ontologie kann zu dieser Erfüllung nur teilweise beitragen, da

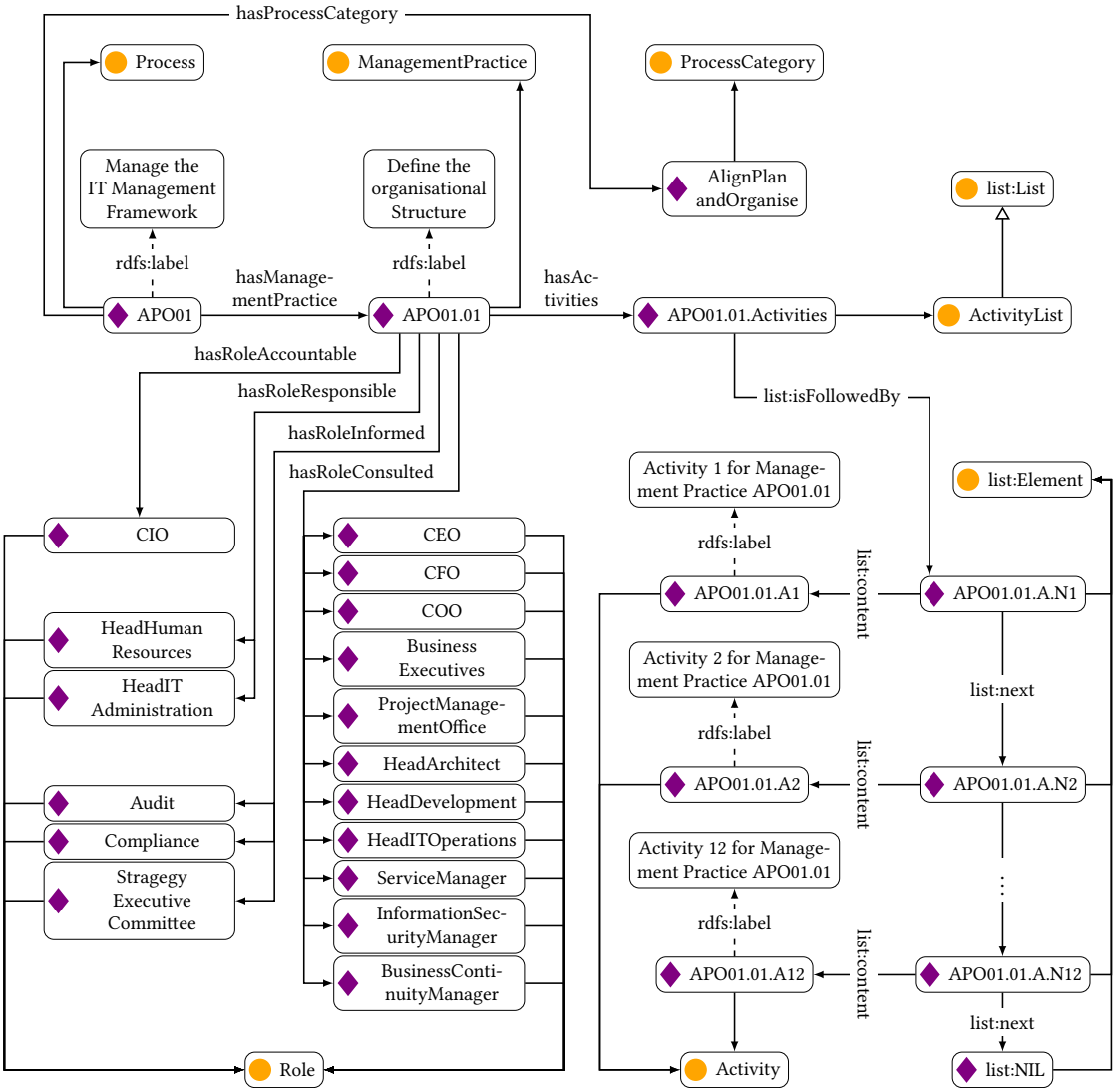

Abbildung 6.6: Ausschnitt: ManagementPractice 01 des Prozesses 01 mit Rollen und Aktivitäten. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

sie aus umfassenden Zuordnungstabellen bestehen würde, die letztlich der ursprünglichen Spezifikation ähneln würden. Die Bezüge sollen stattdessen in der Ontologie selbst hergestellt werden, wo sie sowohl in Modellierungswerkzeugen bei Untersuchung und Erweiterung der Ontologie als auch bei der programmatischen Verarbeitung Berücksichtigung finden können.

Die Nachvollziehbarkeit wird durch eine Kombination von Reifikation impliziter Konzepte, Annotationen und benötigter Vokabulare erzielt. Im ersten Schritt werden Individuen ange-

legt, die die COBIT-Spezifikation selbst, sowie referenzierte Elemente (Abschnitte, Tabellen und Abbildungen) der Spezifikation repräsentieren. Der zweite Schritt besteht aus einer Modellierung von referenzierten, externen Standards, und, soweit nötig, ihren Einzelteilen als Individuen. Im dritten Schritt werden dann die eigentlichen Individuen und Beziehungsaussagen der Ontologie den repräsentativen Individuen zugeordnet. Die Annotation eines Individums, einer Rolle oder einer Klasse wird analog zu einer Beziehungsaussage zwischen zwei Individuen modelliert, außer dass statt einer abstrakten Rolle eine Annotationsrolle verwendet wird. Die Annotation einer Beziehungsaussage zwischen zwei Individuen stellt ein separates Axiom dar, das eine mit der Reifikation von Aussagen in RDF vergleichbare Struktur hat (siehe Abbildung 3.10).

Für die Referenzen auf repräsentative Individuen wird ein Standardvokabular eingesetzt, das für die Beschreibung von Metadaten an Dokumenten weit verbreitet ist, das Vokabular der Dublin Core Metadata Initiative (DCMI) [DCM15]. DCMI unterscheidet zwischen mehreren Namensräumen, die zum Teil historische und zum Teil technische Hintergründe haben; für den Einsatz von DCMI in der COBIT-Ontologie werden die `dcterms` und `dcmi` Namensräume genutzt. Da Dublin Core in Form eines RDFS-Vokabulars veröffentlicht wird, durch dessen direkten Import in eine OWL-Ontologie diese in OWL 2 Full liegen würde, wird die angepasste Version von Dublin Core von Reinhardt [Rei09] eingesetzt. Hierbei werden die ursprünglichen URIs von DCMI beibehalten, um abwärtskompatibel mit auf RDF-Ebene arbeitenden Werkzeugen zu bleiben, `rdfs:Classes` werden aber durch `owl:Classes` ersetzt und `rdf:Propertys` durch `owl:AnnotationPropertys`. Auf diese Weise bleibt die OWL-Ontologie konsistent; Annotationen können aber an allen Elementen, inklusive Beziehungsaussagen, angebracht werden. Abbildung 6.7 zeigt die Annotation von COBIT-Elementen durch Referenzen auf ihre Quellen sowie die Modellierung von Bezügen zu in der Spezifikation referenzierten externen Standards. Dabei werden `cobit5:COBIT5` und `cobit5:COBIT5EnablingProcesses` repräsentativ für die jeweiligen Publikationen als Instanzen der Klasse `dcmi:Collection` modelliert. Ihre Bestandteile, wie zum Beispiel `cobit5:COBIT5Figure16`, werden durch die Annotation `dcterms:isPartOf` ausgedrückt.

6.4 Modellierung von Metriken

Die in COBIT beschriebenen Metriken für die Messung der Erreichung von Geschäfts-, Prozess- und IT-Zielen sind, wie in Abschnitt 5.1.1 beschrieben, nur textuell definiert. Nach Anforderung **F5** (*Formalisierung von Metriken der COBIT-Ontologie*) sollen Metriken allerdings nicht erst auf Implementierungsebene des verarbeitenden Systems, sondern bereits auf konzeptueller Ebene formal beschreibbar sein. Eine Lösung hierfür muss daher bereits Teil der Ontologie-Ebene sein. Nur so kann sichergestellt werden, dass Elemente wie Metriken und Meta-Infor-

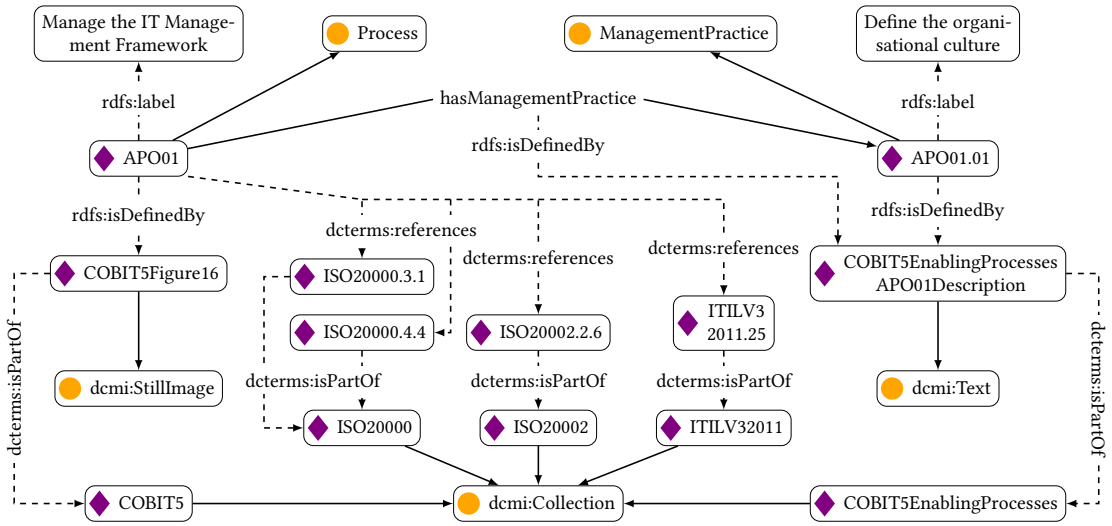

Abbildung 6.7: Quellreferenzen durch Annotationen. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

mationen über Metriken als Konzepte auch über die Grenzen unterschiedlicher Implementierungen hinweg einheitlich referenzierbar bleiben.

Die Aufgabe, die die Formalisierung der Metriken hat, ist analog zur Aufgabe der gesamten COBIT-Ontologie: Die in der Spezifikation enthaltenen, im Wesentlichen menschenlesbar beschriebenen Angaben über die Metriken sollen maschinenlesbar und -verarbeitbar werden. Insbesondere für die Aufgabe als Referenzmodell, das die Integration weiterer Domänenmodelle ermöglicht und vereinfacht, müssen die Metriken der COBIT-Ontologie damit zum einen referenzierbar werden, zum anderen müssen die im Metrik-Text erwähnten Entitäten referenzierbar gemacht werden. Hierdurch wird sichergestellt, dass alle Implementierungen zur Berechnung, Bearbeitung oder Betrachtung eines Metrikwerts sich auf die korrekten Modellelemente beziehen, unter der Annahme, dass die Ontologie hierfür auch als zentrales Informationsmodell eingesetzt wird. Dabei ist zu beachten, dass nicht alle Entitäten, die in Metriken referenziert werden, innerhalb des Kontextes von COBIT zu finden sind, sondern in beliebigen IT-Untersystemen eines Unternehmens zu finden sein können, wie beispielsweise in CMDBs, CRM- oder MES-Systemen – genau aus diesem Grund wird COBIT als verbindendes Referenzmodell eingesetzt. Es können daher an dieser Stelle keine Detailvorgaben gemacht werden, wie die zur Berechnung von aktuellen Metrikwerten benötigten Daten beschafft werden, die konzeptuelle Verbindung von Typen kann aber beschrieben und dadurch maschinenlesbar gemacht werden.

Für die in COBIT beschriebenen Metriken exisitiert kein definiertes Namensschema, daher wird an dieser Stelle ein entsprechendes Schema festgelegt: Für jede nummerierte Metrik j

von Geschäftsziel i wird als Schema `EGM`$i.j$ festgelegt; für jede nummerierte Metrik j von IT-Ziel i wird als Schema `ITG`$i.j$ festgelegt; und für die $k$te Metrik des $j$ten Prozessziels des $i$ten Prozesses aus Prozesskategorie C wird als Schema `PGM`$Ci.j.k$ festgelegt. Jede Metrik wird als Individuum der Klasse `cobit5:Metric` und dem beschriebenen Namensschema folgend definiert. Die textuellen Beschreibungen der Metriken aus der Spezifikation werden dem jeweiligen Individuum über eine `rdfs:label`-Annotation zugewiesen.

Da das Konzept einer formal beschriebenen Metrik in der COBIT-Spezifikation selbst nicht vorhanden ist, soll es auch kein Teil der eigentlichen COBIT-Ontologie werden. Stattdessen wird hier eine *Formalisierungs-Ontologie* (FO) eingeführt, die die COBIT-Ontologie importiert und unter Verwendung von Standard-OWL-Sprachelementen erweitert. Kernkonzept der FO ist die Klasse `fo:FormalMetric`, deren Instanzen die Brücke zwischen den ursprünglichen Metrik-Individuuen aus der COBIT-Ontologie und den in der Metrik-Beschreibung referenzierten externen Entitäten bilden. Für jedes `cobit5:Metric`-Indivuum wird ein FO-Individuum angelegt, das über eine Beziehungsaussage des Typs `fo:formalizes` mit dem ursprünglichen Indivuum verbunden wird. Das FO-Individuum folgt dabei im Namensschema dem Metrik-Individuum; ist der Bezeichner des Metrik-Indiviuums I, so ist der Bezeichner des FO-Individuums `FM`I. Durch die Trennung in `cobit5:Metric` und `fo:FormalMetric` wird die unterschiedliche Semantik beider Konzepte unterstrichen und lässt sich leichter in den Einschränkungen der zugehörigen Relationen ausdrücken; außerdem kann die Verarbeitung zur Laufzeit oder die Weiterentwicklung der FO auch erfolgen, ohne die COBIT-Ontologie laden zu müssen.

Erst durch die Semantik der abstrakten Rolle, die einer Beziehungsaussage zwischen einer `fo:FormalMetric`-Instanz und einer weiteren Entität zugrunde liegt, wird das jeweilige Verhältnis eindeutig beschrieben. Daher wird in der FO die Rolle `fo:hasUnit` definiert, und `fo:FormalMetric` durch eine Existenz-Restriktion so eingeschränkt, dass der formalen Metrik die Beschreibung einer Einheit zugeordnet sein muss. Die Einheit legt fest, welchen Typ von Wert die Metrik beschreibt; hierdurch werden automatische Überprüfungen von Constraints, die Generierung von Oberflächen zur Betrachtung, Bearbeitung und Aggregation von Metrikwerten sowie das programmatische Nachvollziehen von Verbindungen zwischen Metrik und Ausgangsdatenquellen möglich.

Da COBIT explizit die Einführung eigener, unternehmens- und anwendungsspezifischer Metriken unterstützt, ist für die Modellierung der Einheiten von Metriken die Erstellung eines festgelegten, aus der Spezifikation extrahierten Einheitenkatalogs nicht ausreichend. Die Erweiterbarkeit durch selbst definierte Einheiten wird in anderen Rahmenwerken häufig durch eine Aufweichung der formalen Beschreibung erreicht, beispielsweise wird in CIM für die Beschreibung der Einheit eines Attributs nur ein String vergeben. Für die Modellierung von Einheiten in OWL existieren dagegen mehrere Bestrebungen, Rahmenwerke zur erweiterbaren, umfassenden Beschreibung von Einheiten zu schaffen. Die bekanntesten Rahmenwerke dieser Kategorie sind QUDT (Quantities, Units, Dimensions and Types [HKHS14]; im Rahmen eines Datenintegrationsprojekts der NASA entwickelt und veröffentlicht), QUDV (Quantities, Units,

Dimensions, Values [dRB+14]; für den Einsatz zur Beschreibung der Einheiten von Sensordaten im Zusammenhang mit OMG SysML (Object Management Group Systems Modeling Language) entwickelt) und QUOMOS (Quantities and Units of Measure Ontology Standard [QUO14]; ein geplantes Rahmenwerk der OASIS (Organization for the Advancement of Structured Information Standards), dessen Entwicklung allerdings 2014 ohne direkt nutzbares Endergebnis eingestellt wurde). Von diesen Rahmenwerken ist QUDT das ausgereifteste und am flexibelsten erweiterbare, weswegen es hier als Grundlage der Definition von Einheiten von Metriken eingesetzt wird; eine Adaption auf QUDV ist bei Bedarf aber ebenfalls denkbar.

QUDT unterscheidet zwischen fünf Basiskonzepten:

- Die *Quantity Kind* beschreibt die Art einer Einheit, beispielsweise „Länge“. Jede Quantity Kind hat ein Symbol, für Länge ist das Symbol L, für Zeit ist das Symbol T.
- Die *Dimension* beschreibt den Vektor von Exponenten einer zusammengesetzten Quantity Kind, beispielsweise L/T für Geschwindigkeit.
- Die *Unit* beschreibt eine benannte Einheit, beispielsweise „Meter“, und hat immer eine Quantity Kind und zugehörige Dimension, darüber hinaus können Eigenschaften wie Standardsymbol (im Beispiel: m) und Umrechnungsfaktoren festgelegt werden.
- Ein *Quantity Value* ist ein skalarer Wert in Kombination mit einer Unit, beispielsweise $5m$.
- Ein *Quantity Type* ist eine Aufzählung von Einheiten der selben Quantity Kind, für Länge also beispielsweise Meter, Inch, und Kilometer.

Eine `fo:FormalMetric` wird als Unterklasse von `qudt:QuantityValue` definiert. Während ein umfassender Katalog von im Wesentlichen physikalischen Einheiten Bestandteil von QUDT ist, werden für die Formalisierung von COBIT-Metriken eigene Quantity Kinds und Units angelegt. Ein Beispiel für die Modellierung einer Formal Metric und den Zusammenhang zwischen COBIT-Ontologie, FO und QUDT wird in Abbildung 6.8 gezeigt: Für die erste Metrik des Geschäftsziels 6, EGM6.1, werden passende Quantity Kind, Einheit, Formal Metric und die entsprechenden Beziehungsaussagen definiert.

6.5 Zusammenfassung und Fazit

Durch Analyse und Gegenüberstellung der Basis-Dokumente der COBIT-Spezifikation wurde erstmals ein umfassendes formales Modell in Form einer OWL-Ontologie erstellt, das die Kernkonzepte sowohl auf schematischer Ebene, wie auch die benannten und beschriebenen Entitäten und die Beziehungen zwischen ihnen beinhaltet. Die Ontologie adaptiert, soweit vorhanden, existierende Namensschemata für Bezeichner von Konzepten und Entitäten; in den restlichen Fällen nutzt sie hierzu entwickelte geeignete Namensschemata. Die Modellierung der

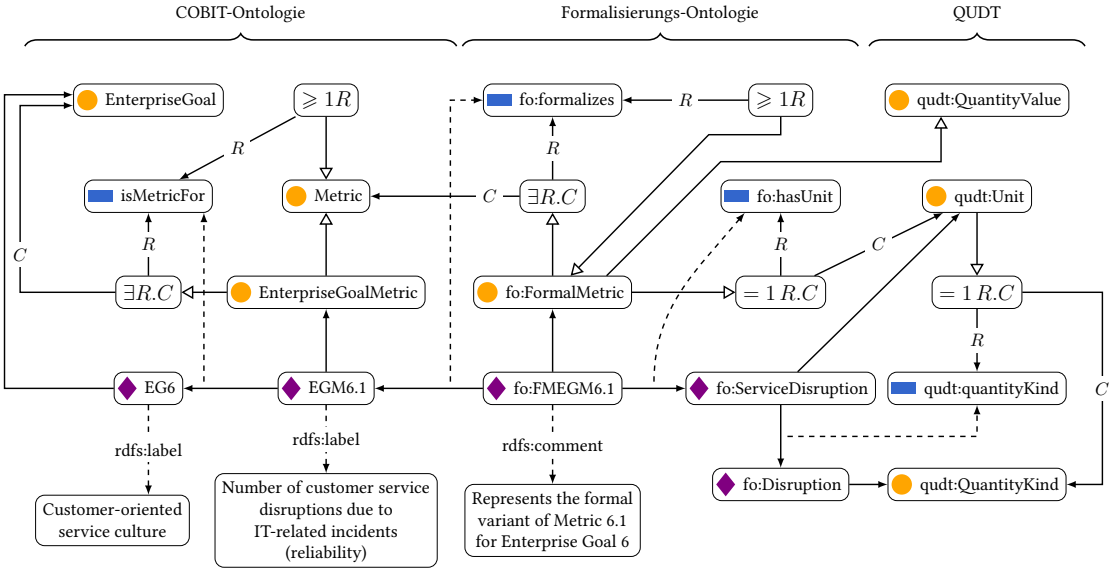

Abbildung 6.8: Formalisierung von COBIT-Metriken am Beispiel der Metrik EGM6.1. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

Ontologie berücksichtigt die Erweiterbarkeit: Klassenrestriktionen sind so definiert, dass im Rahmenwerk zur Anpassung angedachte Konstrukte durch Zufügen entsprechender Axiome in der Ontologie erweitert werden können, ohne Inkonsistenzen auszulösen, beispielsweise bei Geschäfts- oder Prozesszielen. Auf der anderen Seite drückt die Ontologie diejenigen Konstrukte, die explizit als geschlossene Mengen aufzufassen sind, auch als solche aus, beispielsweise die Menge der 37 Prozesse. Die Entwicklung und der Einsatz eines Schemas zur Beschreibung von Listen stellt die Einhaltung der vordefinierten Reihenfolge von Aktivitäten in der Ontologie sicher. Beziehungsaussagen zwischen Entitäten sind durch Annotationen mit Referenzen auf reifizierte Rahmenwerk-Elemente ausgezeichnet, wodurch sich der Zusammenhang zwischen Ontologie und zugrunde liegender Spezifikation direkt nachvollziehen lässt.

Zusätzlich zur Umsetzung des eigentlichen Rahmenwerks erlaubt ein hierzu entwickeltes Vorgehen, die in COBIT natürlichsprachlich beschriebenen Metriken zur Messung des Erreichungsgrades von Geschäfts-, Prozess- und IT-Zielen formal auszudrücken. Hierdurch wird eine bessere Verarbeitbarkeit des Modells in einem Laufzeitsystem ermöglicht, die es erlaubt, Messwerte, die zur Berechnung eines Metrikwerts beitragen, mit einem direkten Bezug zum Rahmenwerk zu korrelieren. Alle Beziehungen, angefangen bei Datenquellen von Messwerten über die formalen Beschreibungen von Metriken bis hin zu den Zielen, zu denen sie beitragen, werden auf diese Weise online abfragbar.

Die Formalisierung hat einige Inkonsistenzen zwischen den einzelnen Spezifikationsdokumen-

ten aufgezeigt; solche Inkonsistenzen könnten durch eine parallel zur Veröffentlichung der Spezifikationstexte vorgenommene Entwicklung eines entsprechenden formalen Modells durch die ISACA selbst reduziert werden.

Zur Erfüllung von Anforderung **F4** (*Konsistenz der COBIT-Ontologie*) wurde die Ontologie mittels des etablierten OWL-DL-Reasoners Pellet [SPG+07] erfolgreich verifiziert. Die Erweiterbarkeit offener Klassen bzw. die Nicht-Erweiterbarkeit der geschlossenen Klassen wurde durch Zufügen entsprechender Axiome und erneute Verifikation durch Pellet sichergestellt. Aufgrund des Umfangs der gesamten Ontologie ist es nicht möglich, diese hier wiederzugeben, unabhängig davon, ob dazu DL-Syntax, eine RDF-Syntax oder eine OWL-Syntax eingesetzt wird. In Anhang F wird daher nur das Schema der Ontologie in DL-Syntax aufgeführt. Die vollständige Ontologie kann über ihre Bezeichner-URL http://purl.org/atextor/ontology/cobit5 bezogen werden.

Obwohl aus den in Abschnitt 6.1 erläuterten Gründen die Ontologie ohne Bezug auf eine Foundational Ontology entworfen wurde, ist die Entwicklung eines Mappings auf eine solche eine mögliche zukünftige Erweiterung. Abhängig vom Einsatzzweck sind jedoch nicht alle Foundational Ontologies gleichermaßen hierfür geeignet. Denkbar für den Einsatz als Referenz und in einem Laufzeitsystem sind Abbildungen auf DUL, GFO, Proton oder UMBEL; ungeeignet sind dagegen BFO, Cyc und YAGO, da sie wegen ihres unterschiedlichen Einsatzzweckes keine geeignete Abstraktionen bieten und aufgrund ihres großen Umfangs nicht unangepasst in jedem Laufzeitsystem einsetzbar sind. UFO ist ebenfalls nicht geeignet, da es in UML spezifiziert ist und damit selbst nach einer möglichen Transformation in OWL keine über die Taxonomie herausgehenden Zusammenhänge beschreibt. SUMO ist dann geeignet, wenn die aus der ursprünglichen FOL-Beschreibung generierte, reduzierte OWL-Ontologie als umfangreich und nachvollziehbar genug eingestuft wird.

Teil III

Realisierung

7 Prototypische Umsetzung

7.1 Implementierung der COBIT-Ontologie

Für die Implementierung der COBIT-Ontologie wurde als Syntax die RDF Turtle Syntax ausgewählt. Von den möglichen RDF-Syntaxen ist Turtle am besten geeignet, um die Ontologie sowohl werkzeuggestützt als auch auf textueller Ebene zu bearbeiten, daher wird sie hier eingesetzt. Eine Serialisierung in einer der nativen OWL-Syntaxen wird ausgeschlossen, da die spätere Verwendung der Ontologie im Laufzeitsystem, wie in Abschnitt 5.2.3 beschrieben, eine Verarbeitung auch auf RDF-Ebene voraussetzt.

Ergebnisse der Konzeptualisierungsphase wurden in einem halbformalen Zwischenformat erfasst, das durch textuelle Bearbeitung in ein Ontologie-Gerüst im Turtle-Format konvertiert wurde. Die weitere Bearbeitung erfolgte zunächst mittels der Ontologie-Entwicklungsumgebung Protégé [Mus15], mit der Konzepte, Entitäten und Zuweisungen zugefügt wurden. Für Validierungen, die über die Möglichkeiten von Protégé hinausgehen, wurde zudem Apache Jena Fuseki [apa14c] eingesetzt, ein auf Apache Jena [apa14b] basierender integrierter SPARQL-Server. Zwischenversionen der Ontologie wurden zunächst in den SPARQL-Server geladen und dann durch Online-SPARQL-Abfragen und den Abfrageergebnissen zum Teil nachgeschalteter Filter und Textumformungen validiert:

- Es wurde sichergestellt, dass alle Literale, die Beschreibungen enthalten, den Typ `rdf:langString` haben und mit dem korrekten Sprachcode ausgezeichnet sind (`@en`).
- Alle Literale mit Beschreibungen wurden extrahiert und einer Rechtschreibprüfung unterzogen.
- Es wurde sichergestellt, dass Entitäten den in Abschnitt 6.2 beschriebenen Namensschemata folgen.
- Es wurde sichergestellt, dass alle Annotations-Axiome einen durch `rdfs:isDefinedBy` festgelegten Verweis besitzen.

Schließlich wurde für die Validierung der COBIT-Ontologie der Prozess eingesetzt, der in Algorithmus 1 gezeigt wird.

1: **function** ONT(i) ▷ i: `owl:Import`-Axiom
2: **return** Von i referenzierte Ontologie
3: **function** UNIQUE(O) ▷ O: Ontologie
4: $R \leftarrow$ RDF-Repräsentation von O
5: $A \leftarrow$ Anonyme Knoten in R
6: $U \leftarrow R \setminus A$
7: **for** $a \leftarrow A$ **do**
8: $a' \leftarrow \text{CONCAT}(\text{ID}(a), \text{ID}(O))$
9: $U \leftarrow U \sqcup \{a \text{ mit neuem Bezeichner } a'\}$
10: **return** U
11: **function** RESOLVE(O) ▷ O: Ontologie
12: $I \leftarrow$ `owl:Import`-Axiome in O
13: $J \leftarrow \varnothing$
14: **for** $i \leftarrow I$ **do**
15: $J \leftarrow J \sqcup \text{RESOLVE}(\text{ONT}(i))$
16: $U \leftarrow \text{UNIQUE}(O)$
17: **return** $(U \setminus I) \sqcup J$
18: $C \leftarrow$ RESOLVE(COBIT)
19: ENTAIL(C, $\bot$) ▷ Reasoner-Aufruf: Test auf Widersprüche in C

Algorithmus 1: Erstellung des transitiven Abschlusses der COBIT-Ontologie mit ihren Abhängigkeiten

Ziel des Prozesses ist es, die Menge aller Axiome der COBIT-Ontologie und des transitiven Abschlusses ihrer Abhängigkeiten zu bilden (Zeile 18) und die Existenz des Bottom-Konzepts $\bot$ (bzw. von `owl:Nothing`) zu widerlegen (Zeile 19), wodurch die Konsistenz bestätigt wird. Die RESOLVE-Funktion löst dabei rekursiv alle `owl:Import`-Direktiven auf. Da die bei der Serialisierung einer Ontologie in einer RDF-Syntax vergebenen internen Bezeichner für anonyme Knoten nur lokal innerhalb der Ontologie eindeutig sind, wird die UNIQUE-Funktion verwendet, um die Bezeichner aller anonymen Knoten global eindeutig zu machen. Als neuer Bezeichner wird hierzu jeweils der bisherige Bezeichner mit dem der entsprechenden Ontologie konkateniert (Zeile 8) und dem Knoten zugewiesen (Zeile 9).

Voraussetzung für die erfolgreiche Ausführung des Prozesses ist die Zyklenfreiheit im Import-Graphen der Ontologien, was im Falle der COBIT-Ontologie gegeben ist. Für den Test auf Widersprüche wird der OWL-DL-Reasoner Pellet eingesetzt, der im Gegensatz zum regelbasierten Reasoner des Laufzeitsystems ein OWL 2 DL- und kein OWL 2 RL-Reasoner ist; sowohl Reasoning als auch Validierung werden daher über einen Tableau-Algorithmus [MH09] ausgeführt. Dies erlaubt im Falle einer Inkonsistenz eine vollständige Auflistung aller Axiome, die zu der Inkonsistenz beitragen, und ist daher für die Entwicklung besser geeignet. Die Überprüfung

der Konsistenz mittels Pellet dauert für die Ontologie auf einem Rechner mit Intel Core i7-3770 Prozessor ca. sieben Minuten. Diese vergleichbar lange Dauer des Reasonings ist zum Entwicklungszeitpunkt unerheblich.

Tabelle 7.1 fasst Statistiken über die entwickelte Ontologie zusammen.

Tabelle 7.1: Statistiken über die COBIT-Ontologie

Typ	**Anzahl**
Anzahl Axiome	41.040
Klassen	40
Abstrakte Rollen	49
Konkrete Rollen	2
Individuen	3.910
Zeilen in Turtle-Syntax	204.410

7.2 Implementierung des Laufzeitsystems

7.2.1 Übersicht über eingesetzte Technologien

Die Implementierung des Prototypen des Laufzeitsystems wurde in Java vorgenommen, da bis auf wenige Ausnahmen die De-Facto-Standard-Software zur Verarbeitung der Ontologie-Standardformate ebenfalls in Java implementiert ist, wie beispielsweise frei verfügbare Parser und Schnittstellen. Aufbauend auf der Entscheidung für den Einsatz von Java wird ein Rahmenwerk benötigt, das Abstraktionen sowohl für Modularisierung unter Berücksichtigung transitiver Abhängigkeiten zu Compile- und Laufzeit als auch für die Verwaltung von Service-Schnittstellen bietet. Letztere beziehen sich dabei nicht in erster Linie auf entfernt nutzbare Schnittstellen wie Web-Services oder REST, sondern auf diejenigen Services, die von Modulen bereitgestellt und genutzt werden. Populäre Build-Werkzeuge wie Apache Maven [apa14d] haben als primäres Ziel nur die Beschreibung von Komponenten und ihren Abhängigkeiten, bieten aber keine Möglichkeit zur Beschreibung von Services. Die Lösung, die hierfür unter dem Java Specification Request (JSR) 376 entwickelt wird, ist zum aktuellen Zeitpunkt noch kein Bestandteil von Java [Rei16] und steht hier daher nicht zur Verfügung. Stattdessen wird als Rahmenwerk für die Modularisierung das zwar nicht offiziell in Java integrierte, aber seit 1998 entwickelte und als Grundlage zahlreicher Anwendungen dienende OSGi (Open Services Gateway initiative, [OSG15]) eingesetzt. OSGi ist eine Spezifikation für eine modulare Service-Plattform mit dynamischem Komponentenmodell; einzelne Komponenten können in einer OSGi-Anwendung also auch zur Laufzeit geladen und entladen werden. Obwohl dieses Feature für den Protoypen keine vorrangige Wichtigkeit besitzt, ist es verbunden mit einem wohldefinierten Lebenszyklus-Modell für Komponenten.

Die OSGi-Kernkonzepte, die in der OSGi-Spezifikation [Ope14, Ope15] definiert sind und für die Architekturbeschreibung unterschieden werden müssen, umfassen:

- *Bundle*: Ein Bundle ist eine Menge von Java-Klassen, zusätzlichen Ressourcen und einem Manifest mit Informationen über die Aktivierung und die Services des Bundles, die als Zip-Datei mit der Endung `.jar` gepackt sind.
- *Manifest*: Der Standard-Mechanismus von Java, um Metadaten in einer `.jar`-Datei abzulegen. OSGi beschreibt eine Menge von zum Teil verpflichtenden zusätzlichen Schlüsseln im Manifest, die Bundle-Name, -Beschreibung und -Version umfassen, sowie Angaben über exportierte und importierte Klassen machen.
- *Bundle-Activator*: Diejenige Klasse, die für ein Bundle nach dem Laden als Einstiegspunkt dient und die beispielsweise programmatisch Services registrieren kann.
- *Service*: Der OSGi-Mechanismus zum Auffinden und Binden von Funktionalität in ggf. zur Laufzeit dynamisch geladenen Bundles. Services werden durch Java Interfaces festgelegt und sollen so wenige Einschränkungen für ihre Implementierungen wie möglich vorgeben. Die sogenannte *Service Registry* ist diejenige von OSGi spezifizierte Schnittstelle, die von jeder OSGi-Implementierung bereitgestellt werden muss, über die programmatisch der jeweils aktuelle Zustand aller Services abgefragt und bearbeitet werden kann. Neben den zu entwickelnden Services, die die Funktionalität der Anwendung ausmachen, existieren auch eine Reihe von Standard-OSGi-Services, unter anderem der Log-Service für die zentrale Verarbeitung von Log-Meldungen aller Services, der Configuration Admin Service für die zentrale Konfiguration von Bundle-Instanzen und der HTTP-Service für den Empfang und Versand von Daten über HTTP.
- *Component*: Eine Component ist eine Java-Klasse in einem Bundle, die zusätzlich in einer zugehörigen XML-Datei deklariert ist. Die Deklaration erlaubt unter anderem die Festlegung von Richtlinien zur Instanziierung der Component, also ob sie vom Rahmenwerk unverzüglich oder erst bei Bedarf instanziiert werden soll. Außerdem kann festgelegt werden, ob nur eine Instanz erzeugt werden soll, die zwischen Aufrufern geteilt wird, oder für jeden Aufrufer eine eigene Instanz. Jede Implementierung eines Sevices ist eine Component, aber nicht notwendigerweise anders herum. Im Folgenden wird *Component* explizit im Sinne der OSGi-Nomenklatur verstanden, während der Begriff *Komponente* allgemeiner einen Bestandteil des Systems bezeichnet, ohne Implementierungsdetails zu implizieren.
- *Reference* oder *Service Reference*: Eine Component kann deklarativ Abhängigkeiten an Services definieren. Das OSGi-Rahmenwerk nimmt die Auflösung vor und übergibt der Component mittels Callback passende Service-Instanzen. Dabei werden die konfigurierbaren Kardinalitäten `0..1`, `1`, `0..n` und `1..n` unterstützt.

OSGi beinhaltet eine Menge weitere Konzepte und Standard-Services, auf die hier nicht weiter eingegangen wird. Es existieren mehrere freie Implementierungen der OSGi-Spezifikation. Als OSGi-Implementierung wird hier die Community-getriebene vollständige Implementierung Apache Felix [apa14a] eingesetzt, die über den Standard hinausgehende, optionale Zusatzfunktionalitäten in Form konformer und implementierungsagnostischer Bundles bereitstellt. Ein Beispiel hierfür, das auch in der Implementierung des Prototypen Anwendung findet, ist die Gogo Shell, ein generischer Kommandointerpreter, über den die OSGi-Anwendung abgefragt und gesteuert, und der um eigene Kommandos erweitert werden kann.

Für die Generierung von Service-Deskriptoren und Bundles werden darüber hinaus die bndtools [BND14] eingesetzt: Dieses Projekt, das früh parallel zur Enwicklung des OSGi-Standards gestartet wurde, ermöglicht unter anderem die Auszeichnung von Service-Implementierungen durch Java-Annotationen, sodass keine Service-Deskriptoren manuell erstellt werden müssen, und die automatische Registrierung von Services anhand von Annotationen, sodass im einfachsten Fall kein Bundle-Activator geschrieben werden muss.

Als Rahmenwerk für die Verarbeitung von RDF und verwandter Formate kommt Apache Jena [apa14b] zum Einsatz. Jena wurde seit Anfang der 2000er Jahre als Open-Source-Produkt von HP Labs entwickelt und ist seit 2009 ein Apache-Projekt. Es ist die am längsten und bis heute aktiv entwickelte Software für RDF auf der Java-Plattform und umfasst mehrere Bestandteile, von denen einige im Prototypen verwendet werden: RDF-Modell, Parser und Serialisierer für alle gängigen RDF-Syntaxen; SPARQL-Modell, -Parser, und Auswertungsengine (ARQ); eine Regelengine mit Unterstützung für Backward- und Forward-Chaining, ein Parser für eine einfache Regel-Syntax und eine Schnittstelle für programmatische Erweiterungen sowie eine Sammlung von Kommandozeilenwerkzeugen. Da sowohl Unterstützung für RDF und SPARQL als auch eine erweiterbare Regelengine im Prototypen benötigt werden, ist Jena hierfür besonders geeignet. Eine Unterstützung für OWL ist in Jena nur rudimentär in Form eines Vokabulars von OWL 1 vorhanden, eine programmatische Bearbeitung einer Ontologie auf OWL-Ebene ist mittels Jena nicht möglich. Da jedoch alle programmatischen Modellbearbeitungen innerhalb des Prototyps auf RDF-Ebene stattfinden, wie in Abschnitt 5.2.3 beschrieben, fällt diese Einschränkung nicht ins Gewicht.

Die Jena Regelengine hat keine Unterstützung für die Auswertung von Inline-Ausdrücken, wie beispielsweise arithmetischer Ausdrücke oder einfacher String-Interpolation. Eine Umsetzung der Auswertung solcher Ausdrücke ist prinzipiell auch durch die SPARQL-Engine möglich, dies würde aber großen syntaktischen Mehraufwand bedeuten, der Entwicklung und Wartung fehleranfällig macht. Aus diesem Grund wird hierzu eine weitere Apache-Bibliothek eingesetzt: Apache Commons JEXL [apa11] (Java Expression Language) ist eine einbettbare Sprache zur Beschreibung einfacher Ausdrücke, deren Syntax von denen der Apache Velocity Template Engine und der Java Server Pages Standard Tag Library (JSTL) inspiriert wurde.

Als Entwicklungshilfsmittel und zur Bedienbarkeit des Laufzeitsystems wird wird außerdem eine Komponente für die Realisierung einer Web-Oberfläche zur Inspektion des Status des Lauf-

zeitsystems, des Reasoners und der geladenen Module benötigt. Als Grundlage hierfür wird das Rahmenwerk Vaadin [vaa16] eingesetzt, bei dem die gesamte Beschreibung von sowohl serverseitiger Verarbeitung als auch Aufbau der clientseitigen Oberfläche in Java erfolgt. Hierdurch lässt sich die Oberfläche gut als optionales OSGi-Bundle integrieren, ohne dabei von weiteren Werkzeugen wie beispielsweise JavaScript-Crosscompilern abzuhängen.

7.2.2 Abbildung von Ontologie-Modulen auf OSGi

Für die Umsetzung der grundlegenden Struktur von Ontologie-Modulen aus Abbildung 5.2 auf die OSGi-Architektur wurden mehrere Ansätze untersucht. Um die Möglichkeiten von OSGi sinnvoll zu nutzen, muss insbesondere eine Abbildung von Abhängigkeiten, die zwischen den Informationsfragmenten eines Moduls bestehen können, auf die Beschreibung von Abhängigkeiten gefunden werden, wie sie innerhalb von OSGi erfolgt. Nur so können beispielsweise Abfragen über den Lebenszyklus von Bundles genutzt werden. Dabei müssen zwei Probleme gelöst werden, die in Abbildung 7.1 zusammengefasst sind.

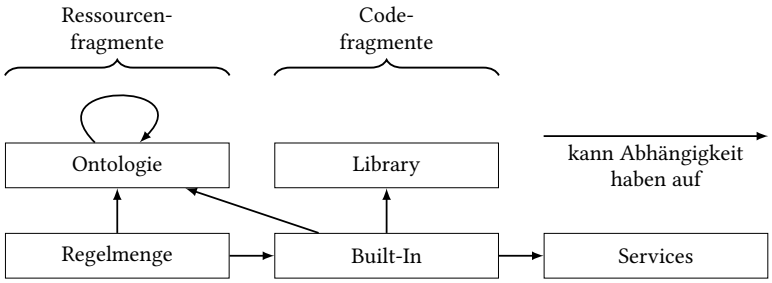

Abbildung 7.1: Mögliche Abhängigkeiten zwischen Informationsfragmenten

Einerseits erlaubt OSGi zwar, wie zuvor beschrieben, in ein Bundle auch Bestandteile aufzunehmen, die keine Java-Klassen sind, diese sind in der Abbildung als Ressourcenfragmente zusammengefasst. Die Mechanismen zur Beschreibung und Auflösung von Abhängigkeitsgraphen sind aber nur auf Java-Code ausgelegt und damit für Ressourcen nicht direkt einsetzbar. Andererseits ist die Auflösung von Abhängigkeiten nur in einem zyklenfreien Graphen möglich. Es gibt diesbezüglich allerdings für Ontologien keine Einschränkung, denn ein Import-Graph kann immer Zyklen enthalten. Die Einführung eines neuen, vom OSGi-Mechanismus separaten Abhängigkeitssystems für alle Informationsfragmente, das seinerseits die Import-Graphen der Ontologien analysiert und aufzulösen versucht, wird hier als nicht sinnvoll erachtet, da ein solches System den konfliktfreien Einsatz der Bundles in anderen OSGi-basierten Anwendungen außer dem Laufzeitsystem nicht erlauben würde. Ebenso muss die Einbeziehung beliebiger Ontologien in Ontologie-Module möglich sein, ohne die Ontologien so anpassen zu müssen, dass keine Zyklen in ihrem Import-Graphen mehr bestehen, da in diesem Fall die flexible nach-

trägliche Erweiterung durch zusätzliche Domänenmodule, wie in Abbildung 5.3 gezeigt, nicht mehr uneingeschränkt möglich wäre.

Stattdessen werden die beiden Abbildungsprobleme hier wie folgt gelöst: Zunächst werden für Codefragmente unverändert normale OSGi-Service-Abhängigkeiten verwendet, und für Regelmengen und Ontologien werden repräsentative Service-Schnittstellen geschaffen, die die jeweiligen Informationsfragmente aus OSGi-Sicht referenzierbar machen. Der Service-Schnittstelle, die Ontologien repräsentiert, wird dabei über eine deklarative Komponenten-Eigenschaft eine bijektive Abbildung auf die IRI zugeordnet, die als Ontologie-Bezeichner dient. Über die Service-Schnittstelle kann eine implementierende Component programmatisch Angaben über die Abhängigkeiten machen, die in der von ihr repräsentierten Ontologie als Import-Statements enthalten sind. Dies wird als Grundlage für einen verzögerten Ladevorgang der Ontologien in Modulen genutzt, bei dem die zyklischen Abhängigkeiten aufgelöst werden können. Dabei wird registriert, wenn Ontologie-Module verfügbar werden, die zur Auflösung von Abhängigkeiten beitragen. Die Auflösung wird dann solange rekursiv versucht, bis eine Abhängigkeit endgültig nicht aufgelöst werden kann oder alle Abhängigkeiten aufgelöst sind und die Ontologien geladen werden können.

Um den Schritt der Erstellung der repräsentativen Service-Schnittstellen zu automatisieren und damit die Entwicklung zu vereinfachen und mögliche Fehlerquellen zu reduzieren, wurde ein Hilfswerkzeug entwickelt, das für eine Ontologie als Eingabe ihren Import-Graphen analysiert und eine zugehörige OSGi-Service-Beschreibung generiert. Das Werkzeug nutzt zur Code-Generierung die Apache Velocity Template Engine [apa10].

7.2.3 Umsetzung der Architektur mittels OSGi

Aus den konzeptuellen Elementen der Architektur in Abbildung 5.4 werden für die Umsetzung des Prototypen entsprechende Komponenten entwickelt. Einzelne Funktionalitäten werden dabei als Services und sie implementierende Components entwickelt und jeweils in OSGi-Bundles gepackt, externe Bestandteile werden in separate Bundles gepackt. Abbildung 7.2 zeigt eine Übersicht über die wesentlichen Komponenten und ihre Abhängigkeiten untereinander. Die Abbildung folgt dabei der Notation eines UML-Komponentendiagramms, der Stereotyp «component» ist daher nicht mit dem Begriff der OSGi-Component zu verwechseln. Die Abbildung gruppiert zusammengehörige Komponenten grau hinterlegt: OSGi-Standard-Services werden durch den OSGi-Standard beschrieben [Ope15]; Basis-Komponenten machen den Kern des Laufzeitsystems aus; Jena-Komponenten abstrahieren die Funktionalität aus der Jena-Bibliothek, Built-In-Komponenten erweitern die Regelengine. Die Implementierung der OSGi-Standard-Komponenten wird von Apache Felix bereitgestellt, Jena- und JEXL-Komponenten von den jeweiligen Bibliotheken, die restlichen Komponenten sind selbst implementiert.

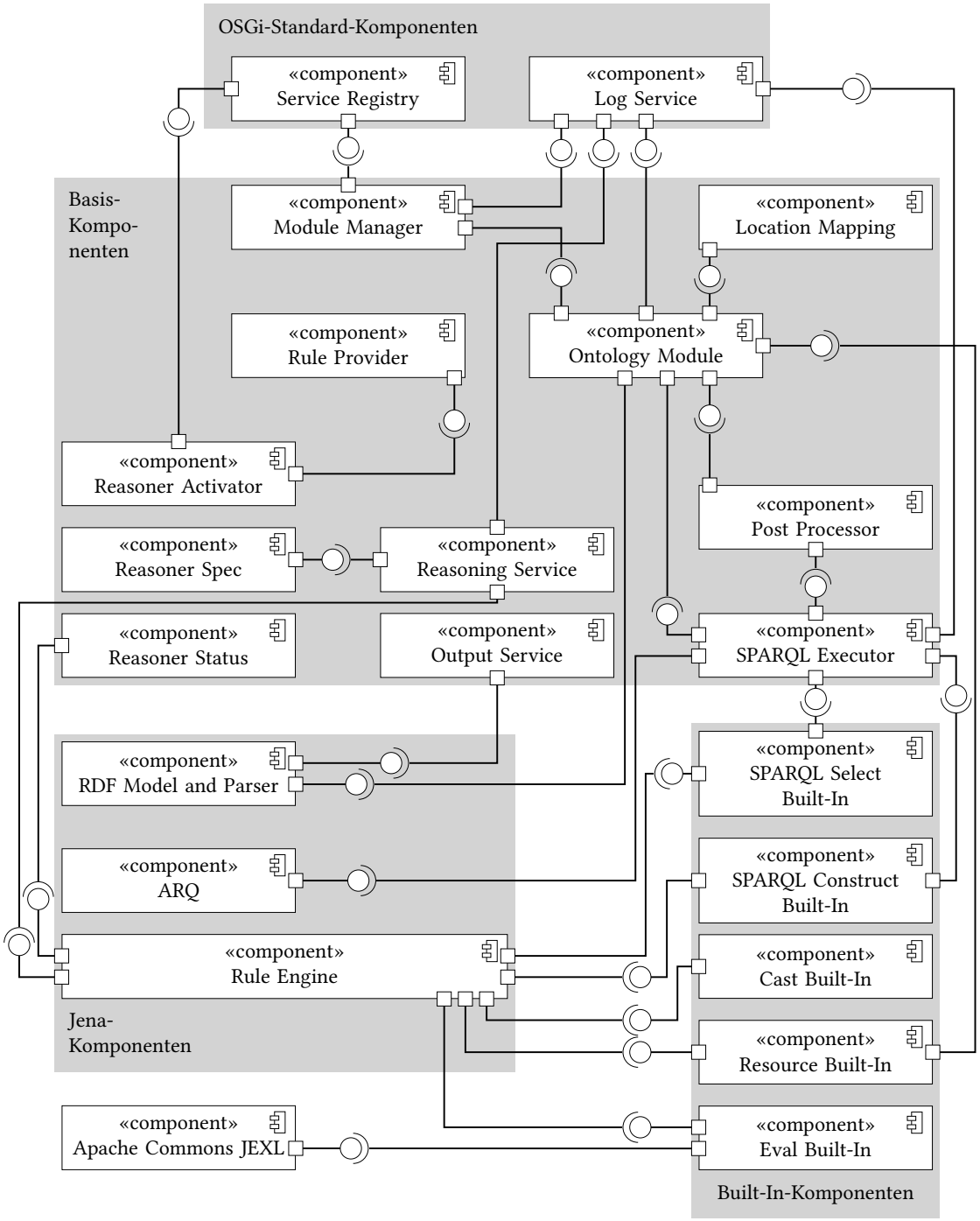

Abbildung 7.2: Grundlegende Komponenten des Prototypen

Die grundlegende Kommunikation zwischen den Komponenten verwendet das sogenannte Whiteboard Pattern [KH04], dessen Ziel vergleichbar ist mit dem des Listener-Patterns, aber bei dem die entsprechende Infrastruktur abstrahiert und zentral vom OSGi-Rahmenwerk verwaltet wird. Werden Services durch entsprechende Deklarationen beim Laden ihres Bundles oder durch äquivalente programmatische Aufrufe aktiv, werden ihre Schnittstellen in der OSGi Service Registry sichtbar und alle Komponenten, die entsprechende Abhängigkeiten definiert haben, werden vom Rahmenwerk benachrichtigt. Hierdurch wird die Umsetzung der Kardinalitäten zwischen Components erreicht. Im Folgenden werden die Basis- und Built-In-Komponenten und ihre Aufgaben mit Bezug zur Architektur knapp beschrieben.

Basis-Komponenten

- Das *Ontology Module* beschreibt die Schnittstelle, die die zu ladenden Module implementieren müssen. Die Abbildung von Ontologie-Modulen auf die OSGi-Architektur wurde im Detail in Abschnitt 7.2.2 beschrieben.
- Jede in einem Modul enthaltene Ontologie wird durch eine IRI identifiziert, ebenso wie jede darin beschriebene Entität. Sowohl RDF und OWL als auch SPARQL haben die eingebaute Möglichkeit zur Festlegung von Präfixen, um Entitäten durch CURIEs zu referenzieren. In der Syntax der Jena-Regelengine existiert diese Möglichkeit nur implizit, d.h. CURIEs können verwendet werden, die Abbildung auf die zugehörige IRI muss aber im die Regel verarbeitenden Programmcode geschehen. Die *Location Mapping*-Komponente hat daher die Aufgabe, die Abbildung zwischen IRIs und CURIEs über die Grenzen aller Ontologie-Module hinweg in RDF-Modellen, SPARQL-Queries und Regelmengen einheitlich vorzuhalten. Mit anderen Worten: egal an welcher Stelle eine CURIE eingesetzt wird, wird sie immer zur korrespondierenden selben IRI aufgelöst.
- Der *Module Manager* hat die Aufgabe, über die Notifikation durch die Service Registry beim Neuladen eines Ontologie-Moduls nötige Bearbeitungen anzustoßen. Dies umfasst im Wesentlichen die korrekte Handhabung der Informationsfragemente des Moduls durch die Location Mapping Komponente, kann bei Bedarf aber beliebig erweitert werden. Laden und Auflösen von Abhängigkeiten ist explizit keine Aufgabe des Module Manager, da diese Aufgaben durch das OSGi-Rahmenwerk abgedeckt sind.
- Der *Rule Provider* stellt die Schnittstelle zur Verfügung, die ein Service implementieren muss, der Regelmengen zur Verarbeitung in der Regelengine ergänzen will. Eine Implementierung des Rule Provider wird durch das Laufzeitsystem selbst beigetragen und umfasst die Basis-Regelmengen der RDFS- und OWL-Semantik. Jedes Ontologie-Modul kann darüber hinaus weitere Rule Provider mit modulspezifischen Regelmengen zufügen. Regelmengen werden durch Mengen von URIs spezifiziert, wodurch sowohl

Dateien, die als Ressourcen in einem Ontologie-Modul enthalten sind, als auch extern aufzulösende Regelmengen durch Bundle-URIs referenziert werden können.

- Der *Reasoner Activator* hat die Aufgabe, nötige Vorbedingungen zu überprüfen und im erfolgreichen Fall den Reasoning Service über die Service Registry verfügbar zu machen. Die Vorbedingungen umfassen dabei das Parsen und erfolgreiche Auflösen aller Referenzen, die in Regelmengen vorkommen: Regeln enthalten CURIEs, die sich auf Modellelemente beziehen, deren Modelle geladen und verfügbar sein müssen; außerdem können Regeln auch Aufrufe von Built-Ins enthalten, die ebenfalls geladen sein müssen. Dadurch, dass der Reasoning Service nicht deklarativ, sondern erst programmatisch nach erfolgreicher Vorbereitung durch den Reasoner Activator verfügbar gemacht wird, können bei seinem Aufruf keine Laufzeitfehler durch fehlerhafte Regeln oder fehlende Referenzen auftreten.
- Die *Reasoner Spec* beschreibt Eigenschaft und Fähigkeiten des Reasoners und stellt eine Factory zur Erzeugung einer Instanz bereit; diese Funktionalität wird für die Erfüllung der entsprechenden Schnittstelle in der Regelengine benötigt.
- Der *Reasoner Status* bietet eine Schnittstelle zum Abfragen des Status des regelbasierten Reasoners: Welche Regelmengen sind geladen, welche Fehler sind beim Laden von Regeln aufgetreten (beispielsweise nicht auflösbare Referenzen), welche Built-Ins sind verfügbar.
- Der *Reasoning Service* ist der zentrale Service zur Nutzung des Reasoners; die Inferenz eines RDF- bzw. OWL-Modells kann hier angestoßen und resultierende Ergebnisse können abgefragt werden.
- Der *SPARQL Executor* ist der zentrale Service zur Auswertung von SPARQL-Abfragen auf den von Ontologie-Modulen bereitgestellten Modellen, kann aber auch auf den Modellen arbeiten, die Ergebnis des Reasonings sind. Dabei werden `SELECT`-, `CONSTRUCT`- und `ASK`-Queries unterstützt. Der SPARQL Executor nutzt die Jena ARQ-Engine.
- Der *Post Processor* ist die Komponente zur Auswertung der Nachverarbeitungsvorschriften, die in Ontologie-Modulen enthalten sein können, wie in Abschnitt 5.2.3 beschrieben. Der Post Processor nutzt für die Auswertung den SPARQL Executor Service.
- Der *Output Service* hat die Aufgabe, unterschiedliche Modellelemente für die Ausgabe in Strings zu serialisieren; dies umfasst Statements, Literale unterschiedlichen Typs, IRI-identifizierte Elemente und gesamte Modelle.

Built-In-Komponenten

- Das *SPARQL Select Built-In* stellt eine Erweiterung der Regelengine dar, die, ausgelöst durch reguläre Vorbedingungen in Regeln, die Resultate von SPARQL-Abfrageergebnissen direkt an Knoten des Graphen binden kann. Diese Erweiterung wird in erster Linie für die Auswertung von Aggregationen eingesetzt, die sich durch Verwendung der nativen Mittel der Regelengine gar nicht oder nur umständlich ausdrücken lassen. Die ausgewerteten Ausdrücke sind dabei alleinstehend keine gültigen SPARQL-Ausdrücke, da die Definitionen von Präfixen wegfallen. Diese werden vor der Auswertung zur Laufzeit gebunden und umfassen alle Standard-Präfixe wie `xsd`, `rdf`, `rdfs` usw., sowie die CURIEs der Modelle aus den geladenen Ontologie-Modulen, die im Location Mapping Service registriert sind. Quellcode 2 zeigt die beispielhafte Verwendung des Built-In, wie sie in einer Regel vorkommen kann: Hier wird ein Knoten des Typs `ex:Date` als Variable `?d` gebunden, anschließend wird ein SPARQL-Ausdruck zur Berechnung eines Wertes basierend auf der `ex:value`-Property von `?d` ausgewertet. Das Ergebnis der Abfrage wird dann über die Property `ex:value2` an den Ursprungsknoten gebunden.

```
[ multiply:
  (?d rdf:type ex:Date),
  sparql-select(?d, 'SELECT ((xsd:int(?v) * 10) AS ?result) WHERE {
    ?this ex:value ?v }', ?vmultiplied)
  ->
  (?d ex:value2 ?vmultiplied)
]
```

Quellcode 2: Verwendung des SPARQL Select Built-In

- Das *SPARQL Construct Built-In* arbeitet analog zum SPARQL Select Built-In, kann aber nicht nur skalare Werte als Ergebnis liefern sondern beliebige Graph-Strukturen erzeugen. Um mehr als einen Ergebnisknoten binden zu können, nimmt das Built-In daher eine beliebige Anzahl von Knoten als Argument, die in der Reihenfolge ihrer Definition im Query als Variablen `$1`, `$2`, ... benannt sind. Eine beispielhafte Verwendung des Built-In wird in Quellcode 3 gezeigt: Hierbei werden zwei Eingabeknoten anhand ihrer `ex:id` in der Regel gebunden und als Argumente in das Construct-Built-In übergeben. Im Query wird jedem der beiden Knoten ein individueller Wert über das Property `ex:value` zugewiesen.

```
[ setvalues:
  (?d1 rdf:type ex:Date),
  (?d1 ex:id "1"),
  (?d2 rdf:type ex:Date),
  (?d2 ex:id "2")
  ->
  sparql-construct(?d1, ?d2, 'CONSTRUCT {
    $1 ex:value "5"^^xsd:int .
    $2 ex:value "6"^^xsd:int . } WHERE {}')
]
```

Quellcode 3: Verwendung des SPARQL Construct Built-In

- Das *Cast Built-In* wird verwendet, um getypte Literale zu erzeugen. Beispielsweise könnte im Schema der Ontologie aus Abbildung 5.6 die Eigenschaft `ex:value` eine Restriktion auf dem Wertebereich haben, die nur Werte vom Typ `xsd:int` erlaubt. Ein vom Adapter aus den Ursprungsdaten erzeugter Wert ist aber zunächst ein ungetyptes Literal und hat damit den Typ `xsd:int`. In einem solchen Fall wäre das korrekte Vorgehen eine Behandlung des Werts als transiente Entität, z.B. `ex:_val` und eine Ergänzung der Domänenregeln wie in Quellcode 4.

```
[ adapter2:
  (?d1 ex:_val ?v)
  cast(?v, ?v2, xsd:int)
  ->
  (?d1 ex:value ?v2)
]
```

Quellcode 4: Verwendung des Cast Built-In

- Das *Resource Built-In* erlaubt es, über den Dateinamen einer Resource in einem Ontologie-Modul die ihr entsprechende Bundle-URI abzufragen. Auf diese Weise ist für jede Ressource zur Laufzeit eine eindeutige, physikalische URI verfügbar, über die sie aus dem Graph referenziert werden kann. Der Resolver für Bundle-URIs ist Teil des OSGi-Rahmenwerks und erlaubt es, alle URIs aufzulösen, die dem `bundle:`-Schema folgen. Das Resource Built-In kann dann in Verbindung mit weiteren anwendungs- und formatspezifischen Built-Ins genutzt werden, die anhand einer URI Informationen aus Ressourcen strukturiert abfragen können. Die Abstraktion der URI als Bezeichner der Datenquelle macht eine einfache Entkopplung von konkreten Formaten oder Protokollen möglich.

Ein Beispiel, in dem mittels des zusätzlichen formatspezifischen `xpath-content` Built-Ins einem gebundenen Knoten der Wert eines XML-Knotens aus einer XML-Datei im gleichen Ontologie-Modul über eine Eigenschaft zugewiesen wird, ist in Quellcode 5 dargestellt.

```
[ augment_value:
  (?d1 rdf:type ex:Date),
  (?d1 ex:id "1")
  # people.xml ist eine Ressource im gleichen Modul
  resource('people.xml', ?people)
  # Als erstes Argument kann eine beliebige URI angegeben werden
  xpath-content(?people, '//people/person[1]/name', ?name)
  ->
  (?d1 ex:name ?name)
]
```

Quellcode 5: Verwendung des Resource Built-In

- Das *Eval Built-In* macht die Möglichkeiten der JEXL-Engine innerhalb der Regeln verfügbar. Der Aufruf des Built-Ins nimmt wie das SPARQL Construct Built-In eine beliebige Anzahl von Argumenten, die die entsprechenden Knoten auf die Variablen `$1`, `$2`, ... bindet. Das Built-In ist insbesondere für die erweiterbare Umformung von Werten geeignet, wie die beispielhaften Aufrufe in Quellcode 6 zeigen. Dabei wird keine ganze Regel gezeigt, sondern nur Aufrufe des Built-Ins unter der Voraussetzung, dass passende Knoten an die Eingabevariablen gebunden sind.

```
# Eingabe hat Werte 'yes' oder 'no', die in 'true' und 'false' konvertiert
# werden sollen
eval(?in, ?out, '$1 == "yes" ? "true" : "false"')
# Eingabe ist ein UNIX timestamp, der in ein gültiges Literal des Typs
# xsd:date konvertiert werden soll
eval(?in, ?out, "new(\"java.text.SimpleDateFormat\",
  \"yyyy-MM-dd'T'HH:mm:ssXXX\").format(Long.parseLong($1, 10) * 10)")
```

Quellcode 6: Verwendung des Eval Built-In

7.2.4 Regelbasierter Reasoner

Wie in Abschnitt 5.2.3 beschrieben wurde, wird die Regelmenge von OWL 2 RL als Grundlage des regelbasierten Reasoners im Laufzeitsystem verwendet, da Ausdrucksmächtigkeit und

flexible Erweiterbarkeit für einen Anwendungsfall wie das Laufzeitsystem besonders ausgeglichen sind. Während ein RDFS-Reasoner in Form einer entsprechenden Regelmenge ein Teil von Jena ist, muss eine Umsetzung der OWL 2 RL-Regeln für den Prototypen vorgenommen werden. Die Regeln, die in der OWL-Spezifikation für die Umsetzung des RL-Profils vorgegeben sind, können als Pseudocode aufgefasst werden, der zum Teil nicht direkt übernommen werden kann.

Abbildung 7.3 zeigt daher für die drei Typen von Regeln jeweils eine exemplarische Umsetzung in der Jena-Syntax. Die *IF*- und *THEN*-Abschnitte sind aus der RL-Spezifikation [MGH+12] übernommen. Die Regel `eq-trans` (Transitive Äquivalenz) steht repräsentativ für den ersten Typ von Regeln, die sich direkt auf die Jena-Syntax abbilden lassen. Die Regel `cls-uni` (Vereinigung von Klassen) dient als Beispiel für den zweiten Typ von Regeln, deren Kopf oder Körper beliebig wachsen kann. Die Umsetzung der Regel wird rekursiv definiert, wobei die Teilregel `cls-unia` die Abbruchbedingung behandelt, wenn die Liste der vereinigten Klassen die minimale Länge von zwei hat. Die Teilregel `cls-unib` behandelt den Rekursionsschritt. Die Umsetzung der Regel nutzt dabei über das Pattern Matching hinausgehende Möglichkeiten der Regelengine, unter anderem Built-Ins zur Abfrage von RDF-Listen, die Bestandteil von Jena sind. Der dritte Typ von Regeln beinhaltet Validierungs- bzw. Konsistenzregeln, bei denen der Regelkopf durch `false` immer eine Inkonsistenz ausdrückt. Hierfür kommen ebenfalls Built-Ins zum Einsatz, die es erlauben, zentral zu konfigurieren, ob validiert werden soll oder nicht, und die hilfreiche Fehlermeldungen bereitstellen, falls eine Inkonsistenz auftritt.

7.2.5 Schnittstellen

Für die Interaktion mit dem Laufzeitsystem und die Erfüllung der Anforderung **N3** (*Bedienbarkeit des Laufzeitsystems*) werden zusätzlich zu den in Abbildung 7.2 gezeigten Basis- und Built-In-Komponenten noch Komponenten eingesetzt, die einerseits den Kommandointerpreter um geeignete Kommandos erweitern und andererseits durch einen eingebetteten Web-Server eine Web-Oberfläche zur Verfügung stellen. Die jeweiligen Komponenten hierzu und ihre Abhängigkeiten an die Basis-Komponenten werden in Abbildung 7.4 gezeigt.

Kommandos nutzen die Erweiterungsschnittstelle der Apache Felix Gogo Shell und kapseln die Funktionalität der entsprechenden Basis-Komponenten. Die Komponente zur Bereitstellung der Web-Oberfläche nutzt den OSGi-HTTP-Service, um einen HTTP-Endpunkt zu öffnen, den Jetty Web-Server [jet16] als Implementierung des HTTP-Service, das Vaadin-Rahmenwerk zur Umsetzung des Modells und Client-Views sowie die Services der Basis-Komponenten für die zugrunde liegenden Abfragen. Obwohl der Web-Server hier eingebettet verwendet wird, das Laufzeitsystem also als unabhängige Anwendung gestartet wird, die ihrerseits die HTTP-Schnittstelle öffnet, ist durch die Verwendung des OSGi-HTTP-Service auch ein Einsatz des

Regel `eq-trans`

IF

```
T(?x, owl:sameAs, ?y)
T(?y, owl:sameAs, ?z)
```

THEN

```
T(?x, owl:sameAs, ?z)
```

Umsetzung

```
[eq-trans:
 (?x owl:sameAs ?y),
 (?y owl:sameAs ?z)
 ->
 (?x owl:sameAs ?z)
]
```

Regel `cls-uni`

IF

```
T(?c, owl:unionOf, ?x)
LIST[?x, ?c1, ..., ?cn]
T(?y, rdf:type, ?ci)
```

THEN

```
T(?y, rdf:type, ?c)
```

Umsetzung

```
[cls-unia:
  (?c owl:unionOf ?x),
  listLength(?x, ?len),
  equal(?len, 2),
  listEntry(?x, 0, ?c1),
  (?y rdf:type ?cx1),
  equal(?c1, ?cx1)
  ->
  (?y rdf:type ?c)
]
```

```
[cls-unib:
  (?c owl:unionOf ?x),
  listLength(?x, ?len),
  greaterThan(?len, 2),
  listEntry(?x, 0, ?c1),
  (?y rdf:type ?cx1),
  equal(?c1, ?cx1),
  (?x rdf:rest ?tl)
  ->
  (?y rdf:type ?c),
  (?c owl:unionOf ?tl)
]
```

Regel `cls-com`

IF

```
T(?c1, owl:complementOf, ?c2)
T(?x, rdf:type, ?c1)
T(?x, rdf:type, ?c2)
```

THEN

```
false
```

Umsetzung

```
[cls-com:
  (?v rb:validation on()),
  (?c1 owl:complementOf ?c2),
  (?x rdf:type ?c1),
  (?x rdf:type ?c2)
  ->
  (?x rb:violation error('inconsistent class
    assertions', 'Can not belong to both
    complement classes', ?x))
]
```

Abbildung 7.3: Umsetzung von OWL 2 RL Regeln (Beispiel)

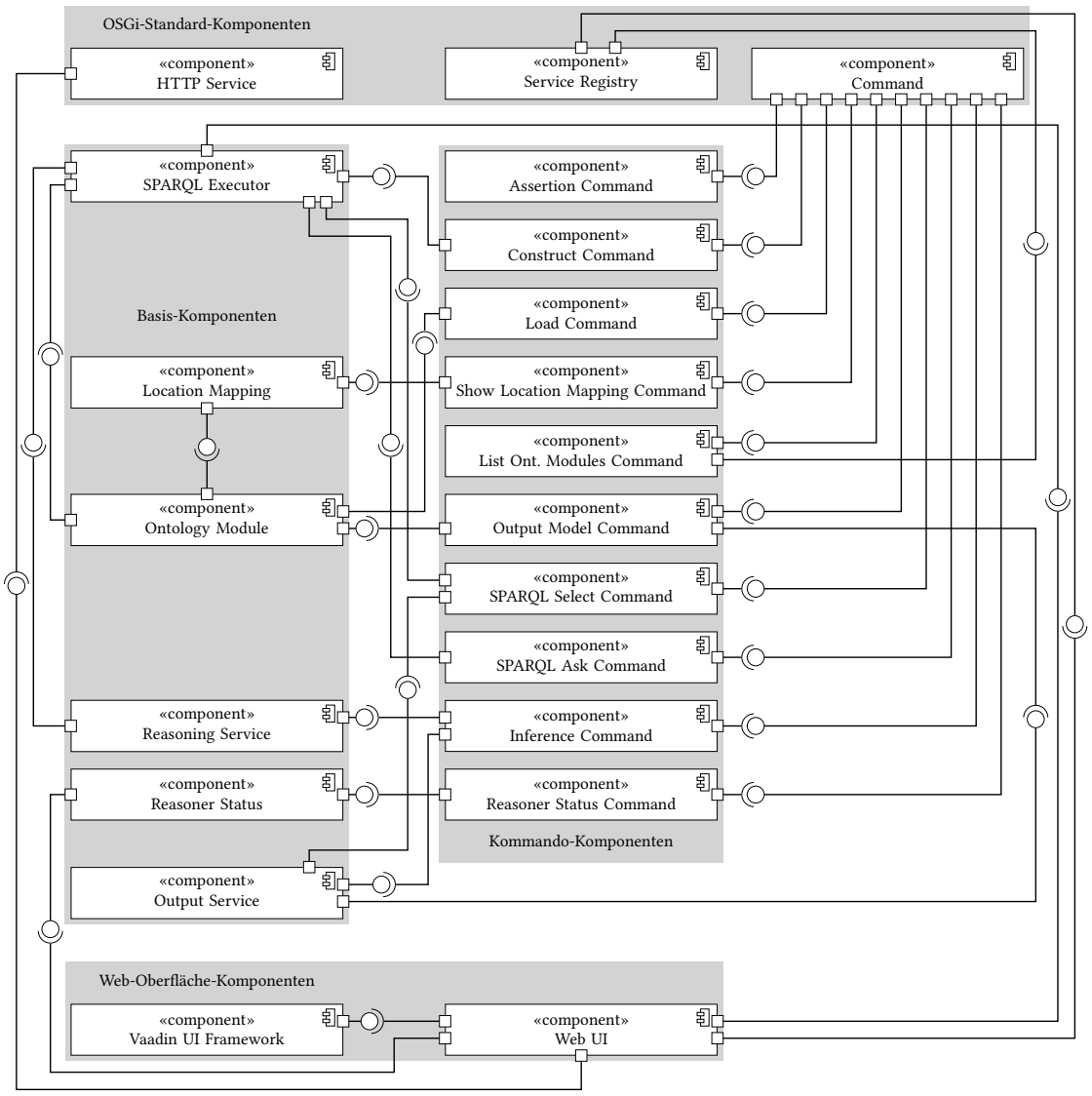

Abbildung 7.4: Prototypen-Komponenten für CLI- und Web-UI

Laufzeitsystems als Servlet innerhalb eines Servlet Containers oder Application Servers denkbar. Über die Web-Oberfläche sind Abfragen des Reasoner-Status, Informationen über die geladenen Ontologie-Module und Built-Ins und Abfragen an die Modelle möglich.

Tabelle 7.2 fasst Statistiken über den implementierten Prototyp zusammen.

Tabelle 7.2: Statistiken über die prototypische Implementierung des Laufzeitsystems

Typ	**Anzahl Zeilen**
Java-Code	7.596
Bnd Bundle-Beschreibungen	383
Velocity Templates	47
CSS	17

8 Evaluation

8.1 Betrachtung im Hinblick auf die definierten Anforderungen

Der Erfüllungsgrad der funktionalen Anforderungen aus Abschnitt 5.1.1 und der nicht-funktionalen Anforderungen aus Abschnitt 5.1.2 nach Konzeption und Umsetzung der COBIT-Ontologie und des prototypischen Laufzeitsystems wird im Folgenden bewertet.

F1 *Formales Modell für das Rahmenwerk COBIT*
Die Anforderung wird als vollständig erfüllt betrachtet, da das implizite Informationsmodell der Spezifikation komplett konzeptualisiert wurde (vgl. Abschnitt 6.1) und in OWL umgesetzt wurde (vgl. Abschnitt 7.1).

F2 *Nachvollziehbarkeit der COBIT-Ontologie*
Wie in Abbildung 6.7 beispielhaft gezeigt, werden in der Ontologie die Zusammenhänge zwischen Modell und ihm zugrundeliegenden Fragmenten der Spezifikation durch Annotationen reifiziert. Die jeweiligen Fragmente sind selbst als repräsentative Entitäten in der COBIT-Ontologie enthalten. Annotationen sind insbesondere auch an den Zuweisungen von Relationen angebracht, sodass beispielsweise nicht nur nachvollzogen werden kann, wo die Elemente *X* und *Y* definiert sind, sondern auch, wo eine Aussage *X steht in Beziehung zu Y* definiert ist. Die Vollständigkeit dieser Annotationen kann programmatisch überprüft werden: Quellcode 7 zeigt die hierzu verwendete SPARQL-Abfrage, die alle OWL-Axiome zählt, bei denen keine Quelle in Form einer `rdfs:isDefinedBy`-Annotation angegeben ist. Zusätzlich wird sichergestellt, dass die vorhandenen Annotationen Elemente im `cobit5`-Namensraum referenzieren. Das Ergebnis der Abfrage ist `0`, die Anforderung wird damit als vollständig erfüllt betrachtet.

F3 *Identifizierbarkeit von Elementen der COBIT-Ontologie*
Alle Elemente der Ontologie sind durch IRIs eindeutig identifizierbar. Verwendete Namensschemata entsprechen denen der Spezifikation, sofern vorhanden. Die Umsetzung der Ontologie verzichtet außerdem auf den Einsatz von anonymen Individuen. Die Anforderung verlangt zusätzlich den Verzicht auf anonyme RDF-Knoten, soweit diese nicht zwingend benötigt werden. Die Abfrage in Quellcode 8 stellt dies sicher, indem die Anzahl von anonymen Knoten gezählt wird, die nicht Basis-RDF- oder OWL-Konstrukte

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX cobit5: <http://purl.org/atextor/ontology/cobit5#>

SELECT (COUNT(?axiom) AS ?result)
WHERE {
  ?axiom rdf:type owl:Axiom .
  OPTIONAL {
    ?axiom rdfs:isDefinedBy ?source
    BIND (!STRSTARTS(STR(?source), STR(cobit5:))
      AS ?isNotDefinedByCobit)
  }
  FILTER (NOT EXISTS { ?axiom rdfs:isDefinedBy [] } ||
    COALESCE(?isNotDefinedByCobit, true))
}
```

Quellcode 7: Query: Test auf fehlende Quellreferenzen

kodieren: Beispielsweise werden Annotationsaxiome als anonyme RDF-Knoten serialisiert. Das Ergebnis der Abfrage ist `0`, die Anforderung wird daher ebenfalls als vollständig erfüllt betrachtet.

F4 *Konsistenz der COBIT-Ontologie*
Eine Überprüfung durch den OWL-DL-Reasoner Pellet bestätigt die Konsistenz des transitiven Abschlusses der Ontologie mit den von ihr importierten Abhängkeiten, wie in Abschnitt 7.1 gezeigt. Die Erweiterbarkeit der nicht als geschlossene Mengen modellierten Konzepte ist, wie in der Anforderung vorgesehen, ebenfalls möglich. Die Wahrscheinlichkeit für unerwartete Reasoning-Ergebnisse, die sich durch das Zufügen von Individuen zu den bestehenden COBIT-Klassen ergeben könnten, kann reduziert werden, indem für alle neuen Individuen konsequent entsprechende `DifferentIndividuals`-Axiome zugefügt werden. Die Anforderung wird als vollständig erfüllt betrachtet.

F5 *Formalisierung von Metriken der COBIT-Ontologie*
Abschnitt 6.4 stellt die Möglichkeit zur formalen Beschreibung der Metriken durch die Verbindung des Einheiten-Rahmenwerks QUDT mit der COBIT-Ontologie vor. Dieses Vorgehen lässt sich für direkt messbare Größen einsetzen, indem die benötigten Einheiten mittels QUDT modelliert werden und zur Laufzeit entsprechende Werte durch geeignete Adapter aus externen Datenquellen in die ABox der Ontologie importiert werden. Dieser Schritt ist stark von den abzufragenden Systemen und der Organisationsstruktur des jeweiligen Unternehmens abhängig und lässt sich daher auf der Abstraktions-

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT (COUNT(?s) AS ?result)
WHERE {
  ?s ?_ [] .
  FILTER(isBlank(?s))
  FILTER(
    NOT EXISTS { ?s rdf:rest [] } &&
    NOT EXISTS { ?s rdf:first [] } &&
    NOT EXISTS { ?s rdf:type owl:Axiom } &&
    NOT EXISTS { ?s rdf:type owl:Restriction } &&
    NOT EXISTS { ?s rdf:type owl:AllDifferent } &&
    NOT EXISTS { ?s rdf:type owl:Class } &&
    NOT EXISTS { ?s rdf:type rdfs:Datatype } &&
    NOT EXISTS { ?d rdf:type rdfs:Datatype .
                 ?d owl:withRestrictions/rdf:rest*/rdf:first ?s }
  )
}
```

Quellcode 8: Query: Test auf anonyme RDF-Knoten

ebene, auf der COBIT angelegt ist, nicht ohne Weiteres im Vorfeld durchführen. Noch wichtiger als die Behandlung der stärker umsetzungsbezogenen Fragen ist allerdings der Umgang mit nicht direkt messbaren Metriken. Für die Eingliederung von Metriken wie das in Abschnitt 2.2 beispielhaft vorgestellte „Level of business executive awareness and understanding of IT innovation possibilities“ in ein Laufzeitsystem ist die Entwicklung eines Prozesses erforderlich, der stärker an der klassischen Knowledge Acquisition angelehnt ist. Mehrere, teilweise auch kombinierbare Vorgehensweisen sind hierzu denkbar: Als Wertebereich für eine entsprechende Einheit könnte beispielsweise eine ganze Zahl zwischen 1 und 10 gewählt werden; eine Einstufung könnte daraufhin durch Fragebögen an die jeweiligen Führungskräfte (und/oder andere Mitarbeiter) oder automatisiert durch Klassifizierung von formal erfassten Managemententscheidungen vorgenommen werden. Für die weitere Verarbeitung derartiger Metriken müssten Werte hierzu außerdem mit Gewichtungen versehen werden. Die Entwicklung eines solchen Prozesses ist außerhalb des Rahmens dieser Arbeit. Aus diesen Gründen werden nur einzelne, in der COBIT-Spezifikation beschriebene Metriken als Proof of Concept umgesetzt (siehe Abbildung 6.8), auf eine vollständige Formalisierung aller Metriken wird verzichtet. Die Anforderung wird daher nur als im Kern erfüllt betrachtet.

F6 *Laufzeitsystem zur wissensbasierten Automatisierung*
Abschnitt 5.2.3 stellt die Architektur eines Laufzeitsystems und Abschnitt 7.2 eine entsprechende prototypische Implementierung vor, das die Anforderung prinzipiell erfüllt. Das Laufzeitsystem ist so erweiterbar, dass Komponenten für die Bearbeitung der MAPE-K-Aufgaben beliebig zugefügt werden können. Die Architektur sieht dabei keine speziellen Strukturen oder Schnittstellen vor, nach denen Komponenten einem der MAPE-K-Schritte zugeordnet werden, stattdessen tragen Erweiterungen durch beliebige Informationsmodelle oder Software-Fragmente in Form von Built-Ins zur Funktionalität bei. Dies kann die Umsetzung von Bestandteilen des MAPE-K-Prinzips für die Integration im Laufzeitsystem zwar unter Umständen komplexer machen, bietet aber die nötige Flexibilität, unterschiedliche Strategien insbesondere für die Analyze- und Plan-Phasen zu realisieren. Hierbei ist es möglich, die bestehende Regelengine zu nutzen, indem entsprechende Regelmengen beigetragen werden, aber es ist ebenso denkbar, dass beispielsweise Komponenten für Machine Learning zugefügt und genutzt werden. Die Optimierung jeweiliger Planungsaufgaben ist jedoch ebenfalls außerhalb des Rahmens dieser Arbeit. Die Anforderung wird als vollständig erfüllt betrachtet.

F7 *Domänenübergreifende Wissensbasis des Laufzeitsystems*
Durch den Einsatz von OWL 2 RL als Ontologiesprache ist die Grundlage für die Nutzung eines Informationsmodells geschaffen, das nachträglich um Unterstützung für beliebige weitere Domänen erweiterbar ist. Erst in Kombination mit dem Modulsystem ist es jedoch möglich, dass eine der Anforderung entsprechende Wissensbasis geschaffen wird, da hierdurch die Verbindung zwischen Informationsmodell und zur Laufzeit verfügbaren Daten ermöglicht wird. Die logische Kapselung von Fragmenten einzelner Domänen unterstützt den Aufbau eines umfassenderen Modells, das eine Core-Ontologie nutzt, wie in Abbildung 5.3 gezeigt. Die COBIT-Ontologie kann als ein Beispiel einer solchen Core-Ontologie betrachtet werden. Die Architektur des Laufzeitsystems erlaubt die Beschreibung von Alignments zwischen Domänenontologien durch Zufügen von Mapping-Ontologien, andere Techniken zum Ontology Matching, wie in Abbildung 3.16 vorgestellt, sind aber mittels entsprechender Module ebenso umsetzbar. Die Anforderung wird als vollständig erfüllt betrachtet.

F8 *Modulare Wissensbasis im Laufzeitsystem*
Die Abhängigkeiten, die zwischen Informationsmodellen, Regelmengen und Softwarekomponenten in Form von Built-Ins zwischen Ontologie-Modulen und innerhalb eines Moduls bestehen können (siehe Abbildung 5.2), können explizit definiert und damit für das Laufzeitsystem auswertbar gemacht werden. Die Wissensbasis kann somit, wie in der Anforderung verlangt, nachträglich beliebig erweitert werden. Durch eine Abbildung der Bestandteile der Module auf die OSGi-Architektur wird sichergestellt, dass zur Laufzeit der Abschluss aller benötigten Fragmente zur Verfügung steht oder das entsprechende Modul bereits auf Ebene des Rahmenwerks als nicht auflösbar markiert wird. Die Anforderung wird damit als vollständig erfüllt betrachtet.

N1 *Orientierung an bestehenden Standards*
Die für die Beschreibung der Modelle in den Ontologie-Modulen vorgesehene Sprache OWL ist die am weitesten verbreitete Standardsprache zur Beschreibung von Ontologien. OWL ist ebenso wie die eingesetzten Technologien RDF und SPARQL ein Standard des W3C. Da die Sprachen in dieser Arbeit unverändert gegenüber dem Standard eingesetzt werden, wird die Anforderung als vollständig erfüllt betrachtet.

N2 *Erweiterbarkeit des Laufzeitsystems*
Die prinzipielle Anwendbarkeit des Laufzeitsystems für beliebige Informationsmodelle ergibt sich durch die Verwendung von OWL als domänenunabhängiger Sprache und der Ontologie-Module, die die nachträgliche Erweiterbarkeit ermöglichen. Die Anforderung wird als vollständig erfüllt betrachtet, es wird an dieser Stelle zur Bestätigung auf den Anwendungsfall verwiesen, der mittels des prototypischen Laufzeitsystems umgesetzt wurde und der in Abschnitt 8.2 im Detail beschrieben ist.

N3 *Bedienbarkeit des Laufzeitsystems*
Die Implementierung des Laufzeitsystems stellt eine Kommandozeilen- und eine Web-Schnittstelle bereit, über die auf die Wissensbasis zugegriffen werden kann, wie in Abschnitt 7.2.5 beschrieben. Durch die Kombination von Kommandos der Apache Felix Gogo Shell, die zur Introspektion der OSGi-Instanz und der geladenen Bundles genutzt werden können, und der spezifischen Kommandos zum Zugriff auf die Wissensbasis wird die Anforderung vollständig erfüllt.

N4 *Für domänenübergreifende Managementaufgaben geeignete Performance*
Die Eignung der Performance des Laufzeitsystems für einen spezifischen Anwendungsfall hängen von dessen Bedingungen ab, daher kann zur Erfüllung der Anforderung an dieser Stelle kein generelle Aussage gemacht werden. Stattdessen wird hierzu ebenfalls auf die Beschreibung des Anwendungsfalls in Abschnitt 8.2 verwiesen.

8.2 Use-Case: Ontologie-basiertes Storage Management

8.2.1 Kontext

Die vorgestellten Konzepte und die prototypische Implementierung des Laufzeitsystems wurden im Rahmen eines umfassenden Anwendungsfalles eingesetzt und evaluiert sowie anhand der währenddessen gewonnenen Erkenntnisse inkrementell weiterentwickelt. Umfang und Vorgehensweisen der Lösung berücksichtigen daher die Erfordernisse, die sich im Laufe des Einsatzes gestellt haben. Die Anpassungen, die sich daraus ergeben, wurden weitestgehend generalisiert und kommen so auch zukünftigen Einsatzszenarien zugute.

Der Anwendungsfall wurde im Kontext von drei Forschungsprojekten mit Beteiligung von Kooperationspartnern aus der Wirtschaft und der öffentlichen Verwaltung durchgeführt, die

in Tabelle 8.1 zusammengefasst sind. Der Autor war an allen Projekten maßgeblich beteiligt und die hier vorgestellten Inhalte wurden, sofern nicht anders vermerkt, von ihm erarbeitet.

Tabelle 8.1: Forschungsprojekte des Use-Case

	Beschreibung
OntoStor	
Titel	Ontologie-basiertes Storage Management
Laufzeit	05.2012 - 12.2012
Förderung	„Forschung für die Praxis“, Förderung durch das Land Hessen[1]
Projektbeteiligte	Labor für Verteilte Systeme, Hochschule RheinMain SVA Systemvertrieb Alexander GmbH, Wiesbaden IBM Systems & Technology Group, Mainz
Kurzbeschreibung	Machbarkeitsstudie und Proof of Concept zum Einsatz von Semantic-Web-Technologien für Storage Management
OntoStorM	
Titel	Ontologie-basiertes Storage Management
Laufzeit	07.2013 - 12.2014
Förderung	„LOEWE 3 ModellProjekte“[2], Förderung durch das Land Hessen
Projektbeteiligte	Labor für Verteilte Systeme, Hochschule RheinMain SVA Systemvertrieb Alexander GmbH, Wiesbaden Hessische Zentrale für Datenverarbeitung
Kurzbeschreibung	Entwicklung eines Ansatzes für Ontologie-basiertes Storage Management und Integration in ein bestehendes Produkt; aufbauend auf den Ergebnissen des OntoStor-Projekts
TOMATO	
Titel	TOMATO Ontology Management Toolkit
Laufzeit	01.2015 - 12.2015
Förderung	„LOEWE 3 ModellProjekte“[3], Förderung durch das Land Hessen
Projektbeteiligte	Labor für Verteilte Systeme, Hochschule RheinMain SVA Systemvertrieb Alexander GmbH, Wiesbaden Hessische Zentrale für Datenverarbeitung
Kurzbeschreibung	Erweiterung des Ansatzes aus dem OntoStorM-Projekt um die Möglichkeit zur Integration zusätzlicher Domänenmodelle

[1] Siehe auch https://wissenschaft.hessen.de/wissenschaft/forschung/kampagne-forschung-fuer-die-praxis

[2] Dieses Projekt (HA-Projekt-Nr.: 383/13-22) wurde im Rahmen von Hessen ModellProjekte aus Mitteln der LOEWE - Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbundvorhaben gefördert

[3] Dieses Projekt (HA-Projekt-Nr.: 455/14-45) wurde im Rahmen von Hessen ModellProjekte aus Mitteln der LOEWE - Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbundvorhaben gefördert

Die Projekte bauen inhaltlich aufeinander auf und haben als projektübergreifendes Kernziel die Weiterentwicklung des Standes der Technik im Storage Management, der auf die Verwaltung von Datenspeichersystemen fokussierten Unterdisziplin des IT-Managements. Dabei sind die Aufgaben innerhalb der Forschungskooperation entsprechend dem eingebrachten Fachwissen der einzelnen Teilnehmer verteilt: Das Labor für Verteilte Systeme der Hochschule RheinMain bearbeitet die Forschungsfragestellung und entwickelt entsprechende Software-Prototypen, die Firma SVA GmbH stellt als Grundlage hierfür ihr Storage-Management-Produkt und entsprechendes Domänenwissen zur Verfügung und übernimmt weitere Entwicklungsaufgaben zur Integration und zum späteren Ausbau der Ergebnisse in ein Produkt, die IBM Systems & Technology Group trägt als Hersteller der eingesetzten Speichersysteme Fachwissen bei, und berät und die Hessische Zentrale für Datenverarbeitung tritt als Nutzer der IBM- und SVA-Produkte mit einer konkreten Fragestellung im Anwendungsfall auf.

Da mit stark wachsenden Datenmengen die für ihre Speicherung benötigten Systeme umfangreicher und so viel komplexer werden, dass sie manuell nicht mehr verwaltbar sind, steigen auch die Anforderungen an entsprechende Managementsysteme. Der erste Schritt hierzu besteht aus der in der Praxis weit verbreiteten Storage-Virtualisierung, bei der die gesamte Menge der physikalischen Speichermedien durch eine zentrale Schnittstelle abstrahiert wird. Storage-Virtualisierung wird in der Regel in Form einer Appliance bereitgestellt, die direkt in ein SAN integriert wird und Speicher auf Blockebene über Fiber Channel, iSCSI, SAS oder vergleichbare Schnittstellen bereitstellt. Zusätzlich stellt die Appliance eine Managementschnittstelle bereit, über die beispielsweise die Topologie der Speichermedien angepasst oder Performance-Statistiken abgefragt werden können.

Ein am Markt etabliertes Produkt zur Storage-Virtualisierung und die hierzu im Anwendungsfall genutzte technische Grundlage bildet der IBM SAN Volume Controller (SVC, [IBM15]). Der SVC bietet viele Features, welche die feingranulare Konfiguration komplexer Speicherumgebungen ermöglichen, wie beispielsweise die logische Gruppierung von Speichermedien in sogenannte Volumes, die transparente Abbildung zwischen physikalischen und logischen LUNs, automatisches Tiering (automatisches Verschieben selten genutzter Daten auf günstigere aber langsamere Speichermedien und häufig genutzter Daten auf schnellere Speichermedien) und automatisches Spiegeln von Daten auf andere Storage Controller oder Datacenter an anderen Standorten. Über eine Kommandozeilenschnittstelle können alle Einstellungen der Features vorgenommen und aktuelle Performance-Daten abgerufen werden.

Obwohl durch den SVC ein zentraler Punkt zur Verwaltung der Speicherumgebung geschaffen wird, sind weitere Werkzeuge erforderlich, um den Umgang damit für einen Administrator effektiv zu machen. So werden Möglichkeiten benötigt, die Speichertopologie zu visualisieren (was für Speicherumgebungen mit mehreren 10.000 Entitäten eine komplexe Aufgabe sein kann), Änderungen an der Umgebung atomar auszuführen, Kapazitäten und Auslastungen von Volumes zu berechnen und den Ist-Zustand des Systems mit Vorgaben wie Replikationsregeln

oder Service Level Agreements abzugleichen. Der zweite Schritt zu einem umfassenden Managementsystem für Speicherumgebungen besteht daher aus Werkzeugen, die Lösungen für Aufgaben dieser Art bereitstellen. Das Produkt BVQ [bvq17] der Firma SVA GmbH stellt ein solches Managementwerkzeug dar. SVA ist ein Systemintegrator mit Fokus auf Storage, der SVC-Systeme vertreibt und supportet. BVQ ist ein SVA-intern entwickeltes Produkt zur Verwaltung von Speicherumgebungen, in erster Linie für SVC-basierte Systeme.

BVQ wird von zahlreichen SVA-Kunden eingesetzt, darunter auch durch die Hessische Zentrale für Datenverarbeitung (HZD), den zentralen IT-Dienstleister für hessische Behörden. Mit zunehmendem Bedarf an Integration unterschiedlicher technischer und nicht-technischer Sichten sowie an Automatisierung zur Reduktion von manuellem Aufwand ist allerdings eine grundsätzliche Weiterentwicklung der Einsatzmöglichkeiten von BVQ nötig. Die Untersuchung der Machbarkeit einer solchen Entwicklung, die Ausarbeitung nötiger Konzepte und ihre Umsetzung als Proof of Concept sind der Hauptgegenstand der Forschungsprojekte. Obwohl der Proof of Concept produktspezifisch auf Basis von BVQ umgesetzt ist, liegt bei vergleichbaren Produkten anderer Hersteller eine änliche Situation vor.

Die zentrale Fragestellung ist die Möglichkeit zur flexiblen Weiterentwicklung, Ergänzung und Anpassung des Datenmodells des Werkzeugs: Die alleinige Verwaltung der Speicherumgebung reicht nicht mehr aus, um die sich stellenden Anforderungen zu erfüllen, da hierbei in zunehmendem Maße die Schnittmengen mit angrenzenden Domänen betrachtet werden müssen. So besteht insbesondere der Bedarf, die Storage-Virtualisierung gemeinsam mit der Software-Virtualisierung zu betrachten. Geschäftsanwendungen vieler BVQ-Nutzer werden zur flexiblen Bereitstellung in virtuellen Maschinen (VMs) betrieben, denen Volumes des SVC als Laufwerke zugeordnet sind. Das Werkzeug zur Verwaltung des Storage-Systems, in diesem Fall BVQ, hat allerdings keine weiteren Informationen über die Anwendung, die in der VM läuft. Managementwerkzeuge zur Verwaltung der VMs „sehen“ dagegen nur das Laufwerk, haben aber keine detaillierten Informationen darüber, in welcher Qualitätsklasse bezüglich garantiertem Durchsatz oder Verfügbarkeit es sich befindet oder welche sonstigen Eigenschaften des zugrundeliegenden Speichersystems das Volume hat. Ein zusammengeführtes, durch Monitoring der jeweiligen Systeme um Laufzeitdaten angereichertes Informationsmodell, das dem Ansatz des integrierten IT-Managements folgt, ist daher notwendig. Sowohl für ein umfassendes Monitoring und die Ergründung von Ursachen im Fehlerfall oder bei Performance-Problemen als auch für die Definition und Auswertung von Regeln zum automatisierten Management, die sich über Domänengrenzen erstrecken können, wird ein solches integriertes Modell benötigt.

Neben der Integration der Sichten auf Storage und Virtualisierung gewinnt mittelfristig auch die Berücksichtigung nicht-technischer Sichten im Informationsmodell an Wichtigkeit. Bei den typischen Anwendungsszenarien der HZD ist dabei insbesondere die Verbindung mit einem Modell für die Abrechnung von Leistungen (Accounting) relevant. Hauptsächlich bedingt durch die Schwierigkeit der Integration des Managements von Storage- und Software-Virtualisierungssystemen haben Kunden der HZD bei der Nutzung von bereitgestellter Infra-

struktur nur die Auswahl zwischen wenigen vordefinierten „Paketen“, die jeweils eine bestimmte Kombination von Merkmalen garantieren. Merkmale umfassen dabei beispielsweise den maximalen zugesicherten Speicherplatz, garantierte IOPS (Input/Output Operationen pro Sekunde) und die Anzahl von VMs inklusive Betriebssystem-Lizenzen. Die kleinste zeitliche Einheit für Bereitstellung und Abrechnung ist dabei ein Monat. Wünschenswert und den Anforderungen entsprechend, die sich in der Praxis stellen, müssten zukünftig aber sowohl Merkmale freier kombinierbar sein als auch die Zeiträume für eine Bereitstellung im Minuten- statt im Wochenbereich liegen. So erfordern beispielsweise einige Anwendungen eine hohe Datenrate aber einen geringen Bedarf an Speicherplatz (oder andersherum), und insbesondere für Entwicklungs- und Testaufgaben werden vergleichsweise kurzlebige VM-Instanzen benötigt. Die Reduktion von unter den aktuellen Gegebenheiten erforderlicher aber eigentlich unnötiger Überprovisionierung spart Geld und unterstützt eine bessere Nutzung vorhandener Ressourcen. Eine Erfüllung derartiger Anforderungen kann allerdings nur durch einen hohen Grad von Integration und Automatisierung erreicht werden, die sich über alle betroffenen Bereiche erstrecken. Während die HZD-Kunden hessische Behörden sind, stehen IT-Abteilungen in Unternehmen vor ähnlichen Problemen, wenn sie die unterschiedlichen Organisationseinheiten versorgen.

8.2.2 Architektur

Das Vorgehen des Lösungsansatzes, wie in Abbildung 5.1 vorgestellt, wird als Grundlage verwendet und für den Anwendungsfall um die benötigten spezifischen Details ergänzt. Abbildung 8.1 zeigt die resultierende Übersicht.

Im *Modell und Mapping*-Schritt werden technische Domänenmodelle für die Repräsentation der Storage- und der Software-Virtualisierung benötigt. Daher wird zunächst untersucht, welche existierenden Modelle relevanter Standards bzw. Schnittstellen hierzu verfügbar sind. Das Standard-Informationmodell zur Interoperabilität von Storage-Systemen ist SMI-S (Storage Management Initiative Specification, [Sto17]) und wird von der SNIA (Storage Networking Industry Association) spezifiziert, es baut auf dem objektorientierten CIM-Modell der DMTF auf. Da der SVC über eine SMI-S-basierte Schnittstelle verfügt und in einer vorherigen Arbeit des Autors eine Übersetzung von CIM in OWL vorgenommen wurde (siehe Abschnitt 4.1), die auch SMI-S umfasst, liegt der Einsatz dieser Schnittstelle und des entsprechenden Modells nahe. Eine genauere Untersuchung der gebotenen Möglichkeiten des SVC in der Analysephase der Umsetzung des Anwendungsfalls zeigte jedoch, dass die SMI-S-basierte Schnittstelle des SVC nicht den Umfang hat, der für die Abfrage und Steuerung erforderlich ist: Einige von den Partnern als besonders relevant eingeschätzte Elemente und Beziehungen des SVC-internen Modells sind in der betrachteten SVC-Version 5 hierüber nicht abfragbar. Daher wird ein SVC-spezifisches Datenmodell entwickelt, das der Beschreibung der Strukturen in der SVC-Dokumentation entspricht. Als zugehörige Schnittstelle zur Abfrage von Laufzeitdaten und zum Ausführen von

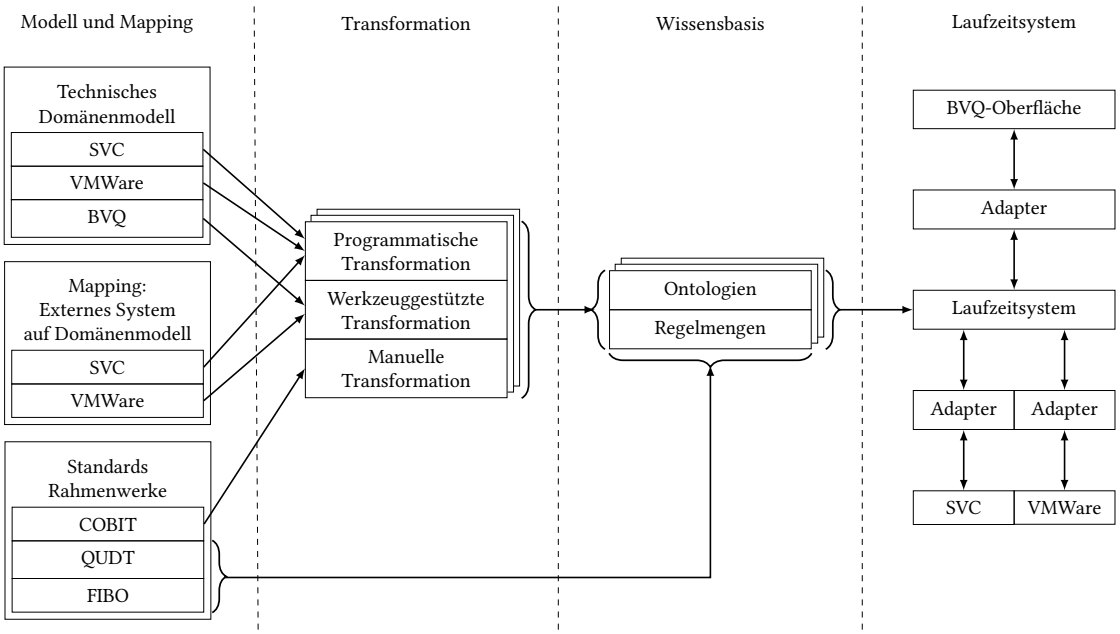

Abbildung 8.1: Anwendung des Lösungsansatzes auf den Use-Case

Managementkommandos muss demnach die ursprünglich für die manuelle Administration vorgesehene SSH-basierte Kommandozeilenschnittstelle genutzt werden. Die Abbildung zwischen den SSH-Kommandos zur Administration des SVC und ihren Ausgaben mit dem SVC-Datenmodell wird in einer separaten Mapping-Ontologie beschrieben.

Als zweites technisches Domänenmodell ist eine Repräsentation der Software-Virtualisierung notwendig. Das im Anwendungsfall vom BVQ-Kunden HZD eingesetzte Virtualisierungsprodukt ist VMware vSphere Server [vmw17]. Eine maschinell verarbeitbare Version des hierfür einsetzbaren Datenmodells ist in Form eines XML-Schemas der REST-basierten Management-Schnittstelle von vSphere Server verfügbar. Für eine automatisierte Transformation eines XML-Schemas in OWL wird in der Literatur zumeist auf die *XML Semantics Reuse Methodology* (XSD2OWL) von García [Gar05] verwiesen, die hier allerdings nicht einsetzbar ist, da dabei XSD-`elements`, die sowohl einen `simpleType` als auch einen `complexType` haben können, in `rdf:Propertys` übersetzt werden. Dies führt zu Ontologien, die in OWL Full liegen. Stattdessen wird hierfür im Rahmen der Fallstudie ein Werkzeug entwickelt, das die Untermenge der im vSphere-XSD eingesetzten Sprachkonstrukte in OWL übersetzt und dabei für den oben beschriebenen Problemfall sowohl eine `owl:DatatypeProperty` als auch eine `owl:ObjectProperty` erzeugt, die durch ein Namenssuffix unterscheidbar gemacht werden sowie durch eine Annotationsrolle mit dem ursprünglichen Namen des XSD-`elements` ausgezeichnet werden. Hierdurch wird der Bezug zum Ursprungsmodell explizit gemacht.

Analog zu SMI-S als Standardmodell zur Interoperabilität zwischen Storage-Systemen spezifiziert die DMTF für Software-Virtualisierung eine Reihe von Standards, beispielsweise ein Format zum Austausch von VM-Metadaten, unter dem Dachbegriff VMAN (Virtualization Management, [dmt17]). Die relevanten Standards bauen wie SMI-S auf CIM auf, es stellt sich aber eine ähnliche Problematik wie beim SVC ein, indem sich das standardkonforme und das produktspezifische Modell unterscheiden. Auf eine Unterstützung von VMAN wird daher verzichtet, es ist jedoch naheliegend, in einer zukünftigen Erweiterung des Proof of Concepts durch die Transformation von VMAN in eine Ontologie und Mapping mit CIM- bzw. SMI-S-Ontologien den Lösungsansatz um Ontologien einer hersteller- und produktunabhängigen Abstraktion zu ergänzen. Dies liegt außerhalb des Rahmens dieser Arbeit. Ebenso wie für den SVC wird für das VMware-Modell im Rahmen dieser Arbeit ein entsprechendes Mapping-Modell entwickelt.

Da BVQ als existierendes Produkt bereits zahlreiche Funktionen bietet, die über die vom SVC bereitgestellten Möglichkeiten hinausgehen, ist auch sein Datenmodell deutlich umfangreicher. Insbesondere ist es im BVQ-Modell möglich, Aggregationsvorschriften zu beschreiben, die auf komplexe Pfade durch den das Speichersystem darstellenden Objektgraphen Bezug nehmen können. Das BVQ-spezifische Modell stellt daher das dritte technische Domänenmodell dar. Ein separates Mapping-Modell wird hier nicht benötigt, da das Modell keine externe Datenquelle beschreibt.

Als Ontologien für genutzte Rahmenwerke kommen die COBIT-Ontologie als Core Ontology und das QUDT-Rahmenwerk, wie in Abschnitt 6 vorgestellt, zum Einsatz. Für die Entwicklung des Modells zur Abrechnung bereitgestellter Ressourcen wird als Core Ontology darüber hinaus FIBO (Financial Industries Business Ontology) genutzt, ein gemeinsam von der OMG (Object Management Group) und dem EDMC (Enterprise Data Management Council, [edm17]) publiziertes Ontologie-Rahmenwerk, das mit mehr als 500 Klassen und 300 Relationen Zusammenhänge der Finanzindustrie und verwandter Themen beschreibt [omg17]. FIBO wird in vier einzelnen Spezifikationen veröffentlicht, die als „Foundations“ (FND), „Business Entities“ (BE), „Financial Business and Commerce“ (FBC) und „Indices and Indicators“ (IND) bezeichnet werden.

Im *Transformation*-Schritt werden alle benötigten Ontologien aus verfügbaren Formaten entweder direkt übernommen, wie QUDT und FIBO, die bereits im OWL-Format vorliegen, oder durch geeignete Transformationen in Ontologien und Regelmengen für die *Wissensbasis* übersetzt.

Das *Laufzeitsystem* ist im Use-Case keine unabhängige Anwendung, sondern wird als Komponente in BVQ integriert, welche die ursprüngliche Datenmodellschicht ersetzt. Durch dieses Vorgehen kann die bestehende Bedienoberfläche in zunächst unveränderter Form weitergenutzt werden, indem ein Adapter entwickelt wird, der die Schnittstelle der ursprünglichen Datenmodellschicht gegenüber der Oberfläche bereitstellt. Das bedeutet, dass die gegenüber dem ursprünglichen BVQ-Datenmodell neue Funktionalität zur Integration weiterer Domänen durch Ontologie-Module zunächst nicht über die grafische Oberfläche genutzt werden

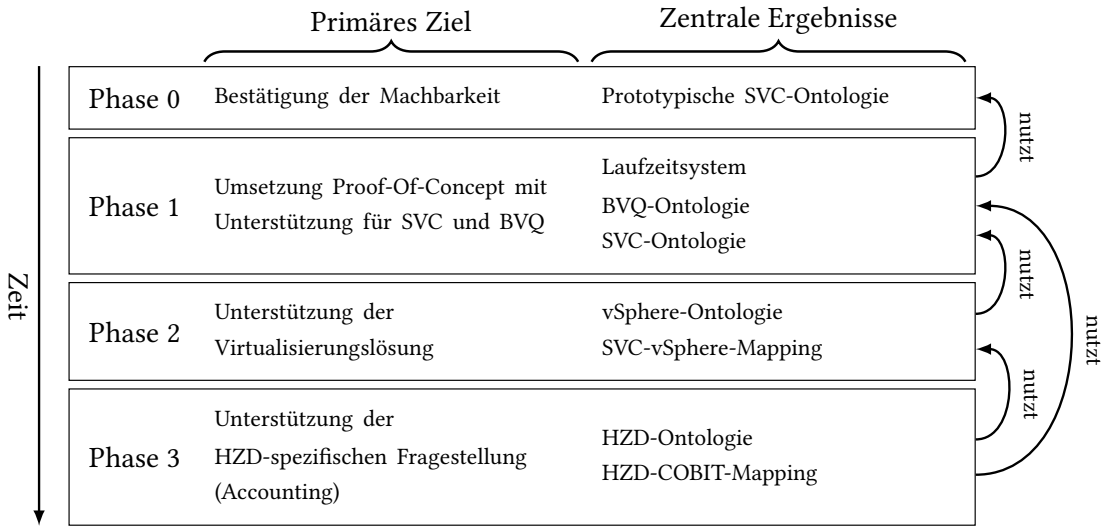

Abbildung 8.2: Entwicklungsphasen des Use-Case

kann, sondern durch sukzessive Anpassung des Programmcodes der Oberfläche dieser Adapter überflüssig gemacht und der volle Funktionsumfang des Laufzeitsystems den BVQ-Nutzern zugänglich gemacht wird. Diese Weiterentwicklung der Bedienoberfläche ist kein Bestandteil der Forschungskooperation, sondern wird durch das SVA-Entwicklungsteam verfolgt.

8.2.3 Ontologien und Umsetzung

Die Entwicklung der Teilergebnisse, die für den Use-Case benötigt werden, wurde in vier Phasen aufgeteilt, die inhaltlich aufeinander aufbauen und zeitlich nacheinander bearbeitet wurden. Abbildung 8.2 zeigt die Phasen, ihr jeweiliges primäres Ziel und ihre zentralen Ergebnisse als Übersicht.

In Entwicklungsphase 0 wurde zunächst die generelle Machbarkeit des Ansatzes untersucht, bei der neben der Überprüfung von Verfügbarkeit und Nutzbarkeit der eingesetzten Software auch eine prototypische Version der SVC-Ontologie entwickelt wurde. Die Umsetzung beginnt mit der Bereitstellung aller benötigten Modelle als Ontologie-Module. Die tatsächliche Anzahl von Einzelontologien ist durch die gegebenen technischen Anforderungen größer als die Anzahl der konzeptuellen Bestandteile aus Abbildung 8.1. Tabelle 8.2 gibt daher einen Überblick über die Ontologien, die im Use-Case und damit auch in den in diesem Abschnitt folgenden Abbildungen verwendet werden. Die in der Tabelle genannten Namensräume sind als CURIE-Prefixes angegeben, die vollen IRIs sind in Anhang E aufgeführt. Soweit nicht anders angegeben, wurden die Ontologien vom Autor dieser Arbeit entwickelt. Der wesentliche Inhalt dieser Ontologien und die Zusammenhänge zwischen ihnen werden im Folgenden erläutert.

Tabelle 8.2: Ontologien im Use-Case

Namensraum	Beschreibung
`cobit5`	COBIT-Ontologie, wie in Abschnitt 6 beschrieben.
`list`	Listen-Ontologie als Abhängkeit der COBIT-Ontologie, wie in Abschnitt 6.2 beschrieben.
`fo`	Formalisierungs-Ontologie, wie in Abschnitt 6.4 beschrieben.
`agg`	Ontologie mit Vokabular zur Beschreibung von Aggregationen, wie in Abschnitt 5.2.3 beschrieben.
`fibo-fnd-dt-fd`	Teilnamensraum aus dem Foundations-Modell von FIBO, der Daten und Zeiten beschreibt. Diese Ontologie wird direkt aus der FIBO-Spezifikation übernommen.
`fibo-fnd-agr-ctr`	Teilnamensraum aus dem Foundations Modell von FIBO, der Vereinbarungen und Verträge beschreibt. Diese Ontologie wird direkt aus der FIBO-Spezifikation übernommen.
`bvq-cls`	Namensraum für Klassen des SVC- und des BVQ-Modells. Ein Großteil der Taxonomie des ursprünglichen BVQ-Datenmodells, das auch die SVC-Entitäten enthält, liegt durch Beschreibungen in einem implementierungsspezifischen XML-Format vor. Dieses Ausgangsformat wird durch ein hierzu entwickeltes Werkzeug des Autors in eine OWL-Ontologie transformiert. Das Ausgangsformat folgt dabei einer Namenskonvention für Bezeichner, bei der nicht explizit zwischen Entitäten und Relationen unterschieden wird. Daher werden auf Ontologie-Ebene separate Namensräume für Klassen, abstrakte und konkrete Rollen eingeführt. Durch die Beibehaltung existierender Namen für Elemente (statt einer Umbenennung entsprechend einer für Ontologien gängigen Benennungsstrategie) wird die Anbindung an die bestehende Oberfläche erleichtert.
`bvq-obp`	Namensraum für abstrakte Rollen des SVC- und des BVQ-Modells.
`bvq-dbp`	Namensraum für konkrete Rollen des SVC- und des BVQ-Modells.
`bvq-agg`	Namensraum für Instanzen der Nachverarbeitungsvorschriften.
`bvq-map`	Enthält die Abbildung der Darstellung externer Systeme auf das jeweilige Domänenmodell.
`vmware`	Namensraum für das Datenmodell der VMware vSphere Schnittstelle. Diese Ontologie wird durch ein hierzu entwickeltes Werkzeug des Autors aus einem XML-Schema generiert.
`hzd`	Namensraum für die Beschreibung HZD-spezifischer Entitäten. Diese Ontologie wurde manuell entwickelt.
`hzd-map`	Namensraum für die Abbildung HZD-spezifischer Entitäten auf die COBIT-Ontologie. Diese Ontologie wurde manuell entwickelt.

Tabelle 8.2: Ontologien im Use-Case (fortgesetzt)

Namensraum	Beschreibung
`uc0`	Namensraum für die Individuen, die konkrete Konzepte im Use-Case repräsentieren. Diese Ontologie wurde manuell entwickelt.
`qudt`	Namensraum des Hauptvokabulars des QUDT-Rahmenwerks. Die Ontologien des Rahmenwerks mit anderen Namensräumen (`unit`, `quantity` und `dimension`) werden im Use-Case nur als Abhängigkeiten importiert. Die Ontologie wird inhaltlich direkt übernommen, es werden zuvor jedoch Inkonsistenzen in der Ontologie entfernt. Details über die nötigen Anpassungen an QUDT finden sich in Anhang C.

Die Kernelemente der Ontologie, die das SVC-Schema enthält, sind in Abbildung 8.3 dargestellt. Die Ontologie ist das Ergebnis der Entwicklungsphase 1. Auf eine Erläuterung aller Elemente wird an dieser Stelle verzichtet, da ihre Semantik die entsprechende Struktur aus der SVC-Dokumentation [IBM15] wiedergibt, die folgenden Erläuterungen zu Klassen sind allerdings zum weiteren Verständnis nötig:

- Ein `Node` bezeichnet eine physikalische Maschine, die Speichermedien enthält, die in der Storage-Virtualisierung eingebunden sind.
- Eine `MDisk` (Managed Disk) ist ein physikalisches Speichermedium.
- `VDisk` (Virtual Disk) ist die SVC-Terminologie für logische Volumes, d.h. logische Gruppierungen von MDisks, die den Speicher für Speicherkonsumenten bereitstellen. Konsumenten haben keinen Zugriff auf `MDisks`. In der SVC-Ontologie ist jeder `VDisk` über die konkrete Rolle `name` ihr SVC-interner eindeutiger Bezeichner zugeordnet.
- Ein `Extent` ist ein zusammenhängender Speicherbereich auf einer `MDisk` und ist die kleinste verwaltete Speichermenge.
- Jede `VDisk` besitzt eine oder zwei `VDiskCopys`, die jeweils unabhängige physikalische Kopien der Daten der `VDisk` darstellen. `VDiskCopys` sind definiert durch eine Menge von `Extents` auf `MDisks`.

Entwicklungsphase 2 hat die Ontologie für das Software-Virtualisierungssystem und das Mapping mit der SVC-Ontologie als Ergebnis. Die Ontologie für das Modell des Software-Virtualisierungssystems wird analog zur SVC-Ontologie erstellt und nutzt ebenfalls die Namensräume `bvq-cls`, `bvq-obp` und `bvq-dbp`. Die Kernelemente dieser Ontologie werden in Abbildung 8.4 gezeigt und umfassen im Wesentlichen Klassen für die `vm_VM`, den `vm_Host`, auf dem die `vm_VM` läuft, und `vm_VirtualDisks`, die als Laufwerke den `vm_VMs` zugeordnet sind. Da die Zuordnung über weitere Attribute verfügen kann, ist sie reifiziert und ebenfalls als Klasse modelliert. Jede

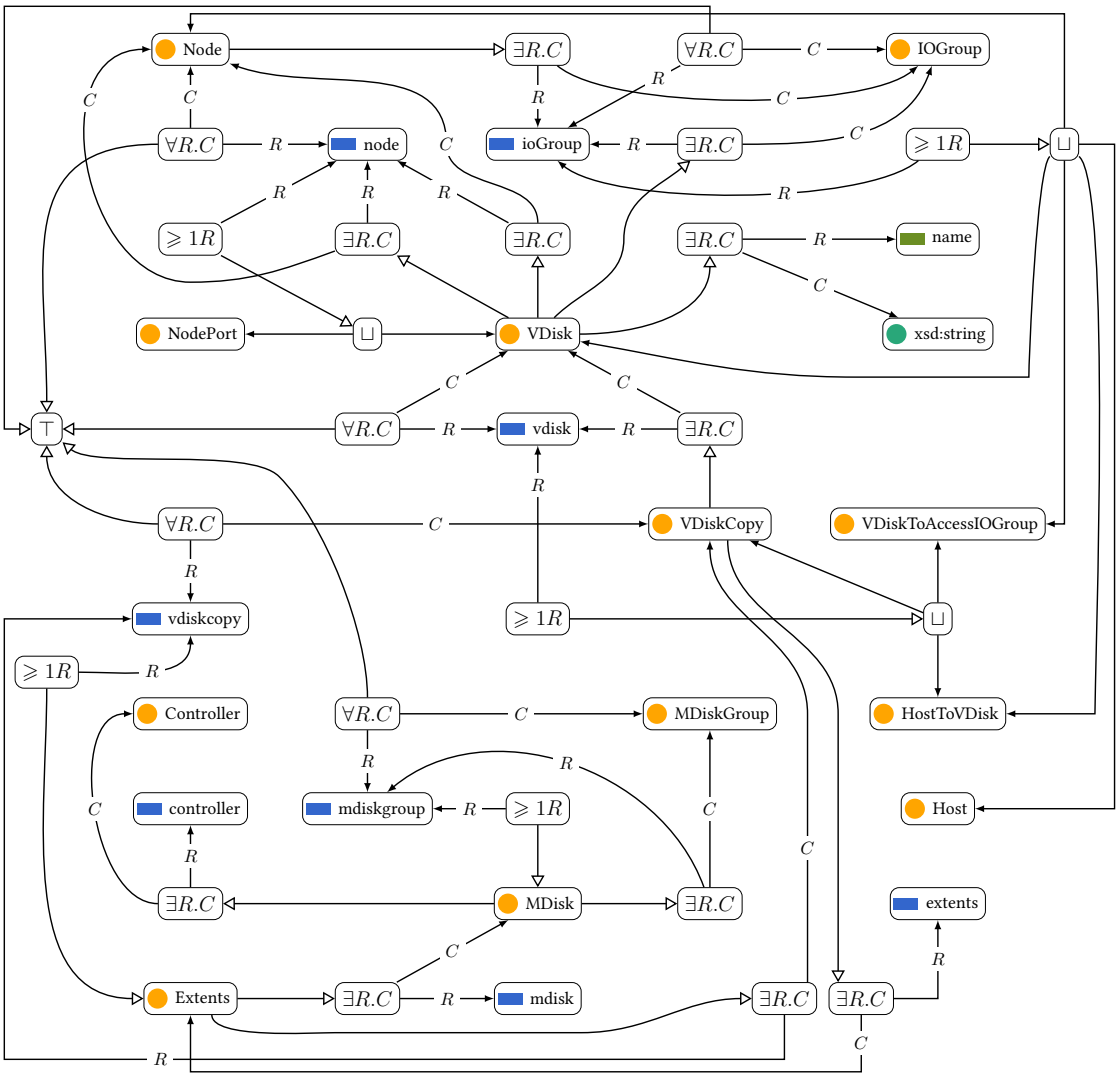

Abbildung 8.3: Kernelemente der SVC-Ontologie. Der Standardnamensraum in dieser Abbildung ist `bvq-cls` für Klassen, `bvq-obp` für Object Properties und `bvq-dbp` für Data Properties.

`vm_VirtualDisk` wird über die konkrete Rolle `vdisk_label` mit dem Bezeichner versehen, der sie innerhalb des Virtualisierungsssystems eindeutig benennt.

Das Mapping zwischen der SVC-Ontologie und der Virtualisierungsontologie wird über die separate, abstrakte Rolle `isAssignedTo` vorgenommen, über die eine SVC-`VDisk` mit einer `vm_VirtualDisk` verbunden werden kann. Eine `owl:sameAs`-Beziehung wird hier explizit nicht eingesetzt, da die jeweiligen Entitäten konzeptuell unterschiedlich sind und daher eine entsprechende Gleichsetzung zu unerwünschten Inferenzen führen würde. Eine Beziehungsaussage der `isAssignedTo`-Rolle zwischen einer `VDisk` und einer `vm_VirtualDisk` wird basierend auf einer Regel zugesichert, welche die Werte der `name`- und `vdisk_label`-Attribute vergleicht. Das Mapping zwischen der generischen Virtualisierungsontologie und der technologiespezifischen vSphere-Ontologie kann dagegen durch eine `owl:sameAs`-Beziehung zwischen `bvq-cls:vm_VirtualDisk` und `vmware:VirtualDisk` hergestellt werden, da hierbei jeweils dieselbe Entität bezeichnet wird.

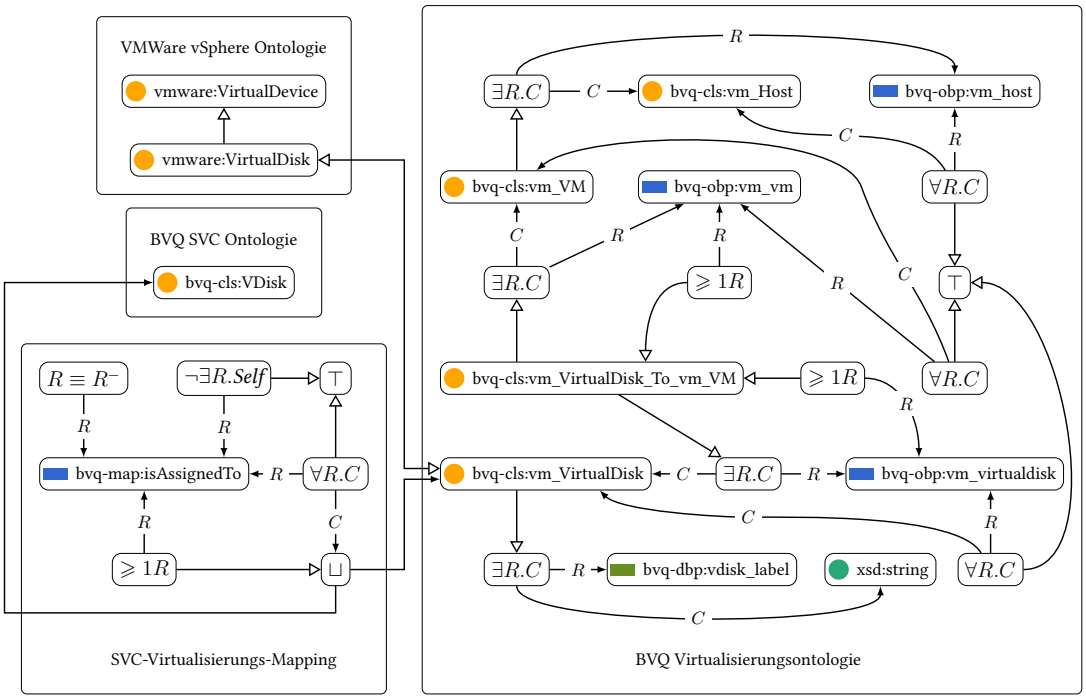

Abbildung 8.4: Kernelemente der Virtualisierungsontologie und Mapping zwischen SVC-, Virtualisierungs- und VMware-Ontologien.

Im Rahmen des existierenden BVQ-Modells sind neben der Taxonomie des SVC auch Aggregationsvorschriften definiert, die anhand von Pfaden durch den zugrunde liegenden Objektgraphen Werte zur besseren Verständlichkeit des aktuellen Zustands des Speichersystems aufbereiten. Die Ausdrucksmöglichkeiten des Formats, in dem die Aggregationen beschrieben wer-

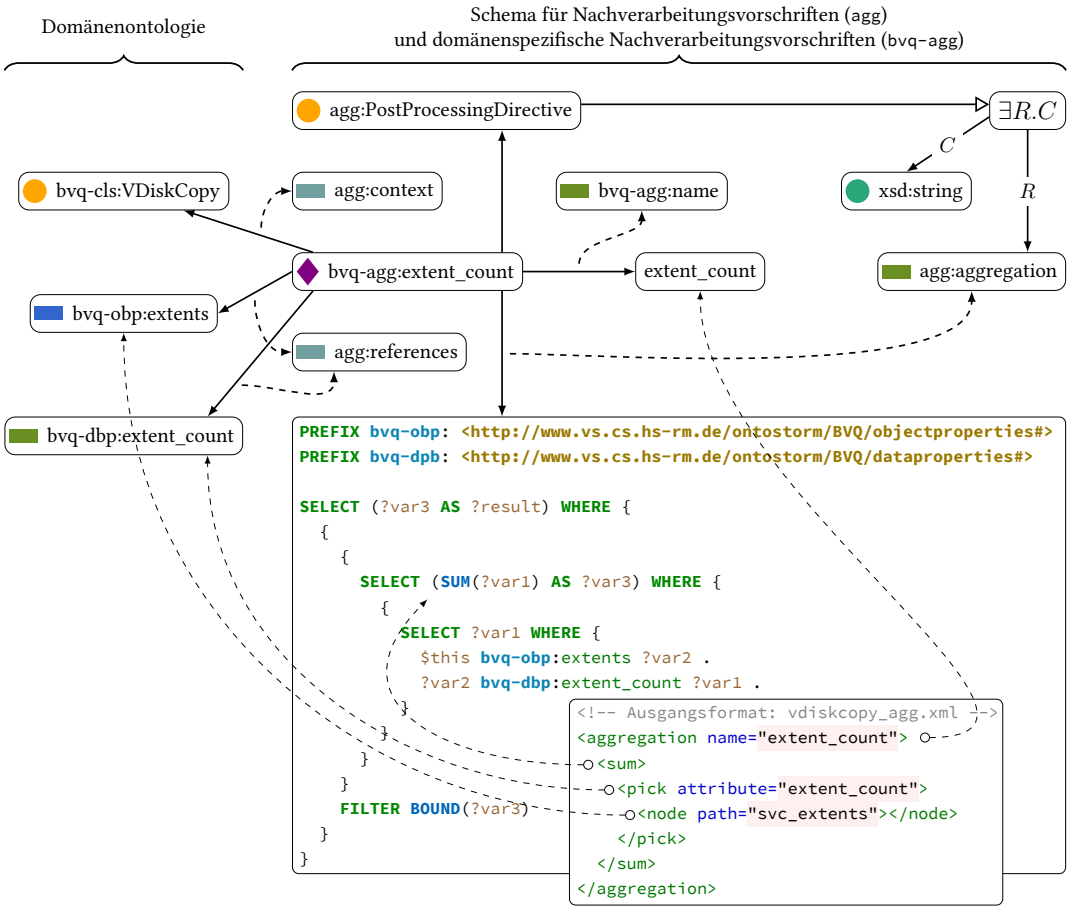

Abbildung 8.5: Beispiel: Nachverarbeitungsvorschriften für Aggregationen des SVC-Modells

den, können als Untermenge der in SPARQL verfügbaren Ausdrucksmöglichkeiten betrachtet werden, es wird jedoch ein proprietäres Format in XML-Syntax zu ihrer Definition eingesetzt. Für den Übergang zu einem Ontologie-basierten Modell wird daher eine Transformation der bestehenden Aggregationsvorschriften in die Struktur der Nachverarbeitungsvorschriften des Laufzeitsystems vorgenommen, wie in Abschnitt 5.2.3 beschrieben. Da in BVQ mehr als 4.000 solcher Aggregationen definiert sind, wird die Transformation programmatisch vorgenommen. Abbildung 8.5 zeigt anhand eines Beispiels, wie das Ausgangsformat übersetzt wird. Dabei wird die im Kontext der Klasse `VDiskCopy` definierte Aggregationsvorschrift `extent_count` in eine Instanz von `agg:PostProcessingDirective` übersetzt. Über die Annotationsrolle `agg:context` wird eine explizite Referenz auf die Klasse und über die Annotationsrolle `agg:references` Referenzen auf die in der Nachverarbeitungsvorschrift genutzten konkreten Rollen der Domänenontologie zugefügt.

Die Semantik des generierten SPARQL-Queries der Vorschrift entspricht weitestgehend der Semantik der Auswertung des Ausgangsformates. Das Ausgangsformat wird durch rekursiven Abstieg des DOMs in SPARQL-Ausdrücke übersetzt, bei dem Bezüge auf Knoten des Objektgraphen aus äußeren Kontexten in den rekursiven Aufruf übergeben werden. Die resultierenden einzelnen Ausdrücke werden dann, je nach Typ, entweder als Graph-Patterns, als Summen- bzw. Aggregationsausdrücke oder als Subqueries zu einem Ergebnisquery zusammengesetzt. Hierdurch können alle wesentlichen Sprachkonstrukte direkt übersetzt werden, lediglich ein Sprachkonstrukt des Ausgangsformates, das die Auswertung eines Unterausdrucks an eine zu implementierende Java-Klasse delegiert, kann nicht in SPARQL übersetzt werden. Dieses Sprachkonstrukt wird von vier Aggregationsvorschriften genutzt, die daher manuell übersetzt werden. In der Beispielaggregation in Abbildung 8.5 werden, ausgehend von einer `VDiskCopy`, alle Knoten selektiert, die über die Kante `svc_extents` erreichbar sind, und über diese Knoten wird das Attribut `extent_count` aufsummiert.

Für die Use-Case-spezifische Fragestellung der HZD wird in Entwicklungsphase 3 eine separate Ontologie entwickelt, die die BVQ-Ontologie und Standard-Entitäten des FIBO-Rahmenwerks nutzt und die in Abbildung 8.6 gezeigt wird. Kernelement der Fragestellung ist das HZD-Verfahren, das den Vorgang der Bereitstellung von IT-Ressourcen darstellt und dem die vertraglichen Aspekte wie Vertragsparteien zugeordnet sind. Beliebige bereitgestellte Entitäten, hier als Abrechnungsentitäten bezeichnet, sollen innerhalb eines festgelegten Zeitraums einem Verfahren zugeordnet werden. Ein Zeitraum wird hier als Unterklasse von `DatePeriod` aus den FIBO Foundations definiert, eine zusätzliche Abbildung auf das ODPA-Muster `TimeInterval` [Pre08] ist ebenfalls denkbar. Als Abrechnungsentität wird hier beispielhaft die Klasse `hzd:VM` definiert, die durch die Unterklassenbeziehung zu `vm_VM` den Bezug zum technischen Modell herstellt, die Nutzung anderer Modellelemente als Abrechnungsentitäten ist aber ebenso möglich. Für die Erstellung und Verwaltung von Abrechnungen werden schließlich den Klassen entsprechende konkrete Instanzen angelegt und durch Beziehungsaussagen verbunden.

Sowohl das Laufzeitsystem als neue Datenmodellschicht in BVQ als auch die Ontologie-basierte Verknüpfung von Storage-, Virtualisierungs- und Abrechnungsontologien wurden neu entwickelt, daher wurden entsprechende Instanzen, die nicht Storage-Entitäten repräsentieren, im Rahmen eines Tests manuell angelegt. Bei einer längerfristigen Anwendung bzw. der Weiterentwicklung der BVQ-Oberfläche können diese durch passende Adapter analog zur Funktionsweise des SVC-Adapters automatisch zu den ABoxen der jeweiligen Ontologiemodule zugefügt werden.

Durch die Formalisierung der Abrechnungsfragestellung wird auch die Verbindung der HZD-Ontologie mit der COBIT-Ontologie möglich, die durch eine separate Mapping-Ontologie realisiert wird, wie in Abbildung 8.7 gezeigt. Die Verbindung wird in diesem Beispiel über die abstrakte Rolle `providedBy` zwischen der Use-Case-spezifischen Instanz eines Verfahrens und dem Work Product 9 („Cost allocation model“) der Management Practice APO06.04 („Model

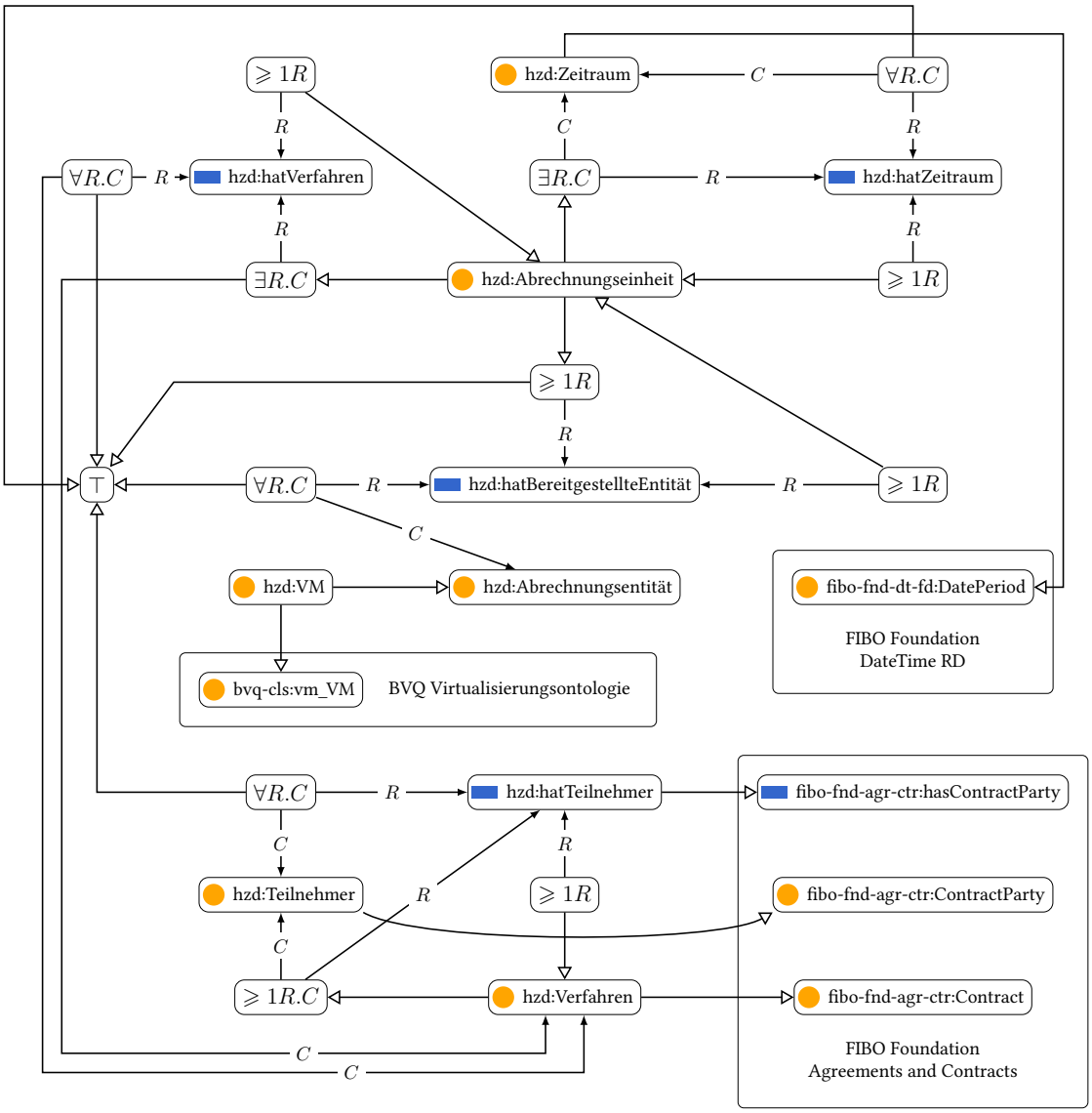

Abbildung 8.6: HZD-Ontologie und Relation zu FIBO-Entitäten

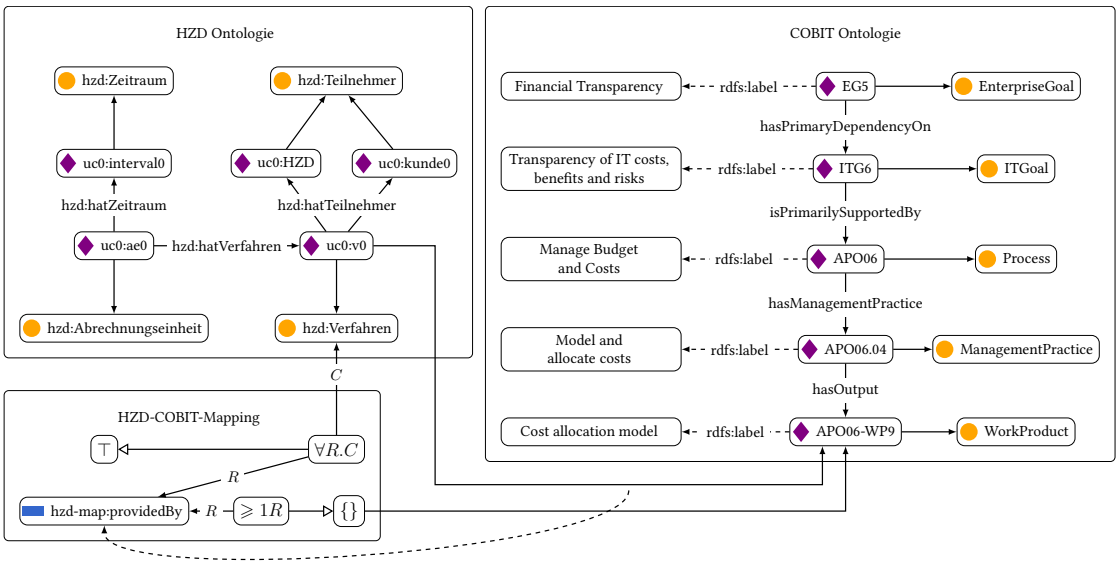

Abbildung 8.7: Verbindung zwischen HZD-Ontologie und COBIT-Ontologie. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

and allocate costs“) hergestellt, das Verfahren dient aus COBIT-Sicht also als Modell zur Kostenzuordnung. Auf diese Weise werden die zur Beantwortung von IT-Governance-Fragen notwendigen Zusammenhänge durch die Verbindung der Ontologien formal erfasst.

Durch die Einbeziehung der Formalisierung von Metriken in COBIT, wie in Abschnitt 6.4 vorgestellt, können darüber hinaus auch die verfügbaren formalisierten anwendungsspezifischen Informationen zur Beantwortung von IT-Governance-Fragen genutzt werden. Abbildung 8.8 zeigt den kompletten Zusammenhang der HZD-Ontologie inklusive der Use-Case-spezifischen Beispielinstanzen mit der COBIT-Ontologie, der Mapping-Ontologie, der Formalisierungs-Ontologie sowie der dabei genutzten QUDT-Ontologie. Ausgangspunkt ist die Definition von formalen Metriken für die COBIT-Metriken PGDSS01.1.2 („Number of incidents caused by operational problems“) und PGDSS01.2.1 („Ratio of events compared to the number of incidents“). Diese werden hier exemplarisch betrachtet, andere COBIT-Metriken könnten analog behandelt werden.

Zunächst werden `fo:Event` und `fo:Incident` als Einheiten unter Nutzung von QUDT sowie entsprechende Instanzen für die formalen Metriken definiert. Für die Rate von Events zu Incidents werden eine passende Quantity Kind und eine Einheit angelegt, die die Quantity Kind verwendet. Domänenspezifische Konzepte, die Abhängigkeiten an die COBIT- oder die Formalisierungs-Ontologie haben, werden im Kontext der Mapping-Ontologie definiert, die Einführung eines weiteren Unternamensraums der Domänenontologie ist aber ebenfalls

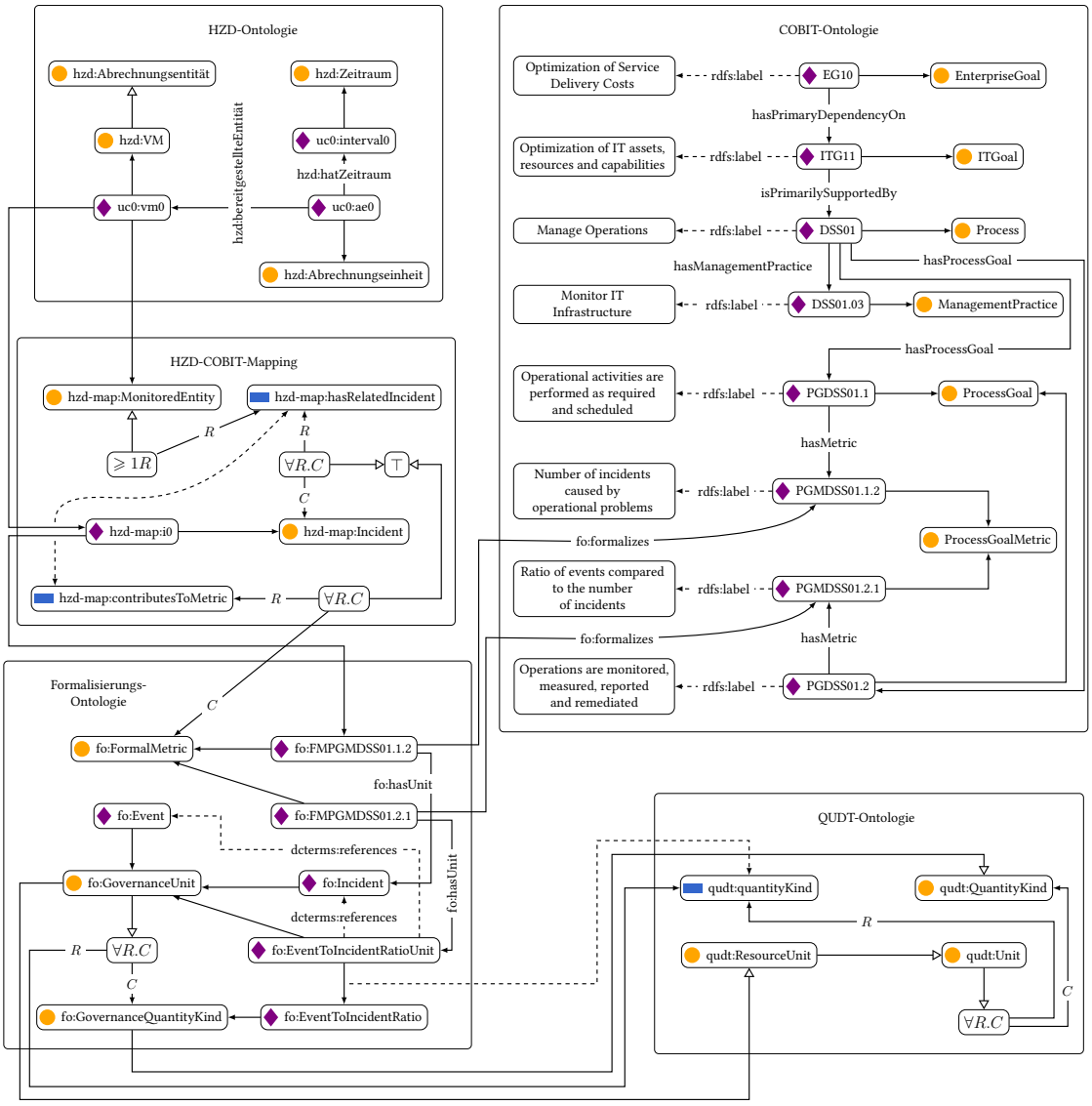

Abbildung 8.8: Verbindung zwischen HZD-Ontologie, COBIT- und Formalisierungs-Ontologien. Der Standardnamensraum in dieser Abbildung ist `cobit5`.

möglich. Die Mapping-Ontologie wird um zwei Klassen erweitert: `hzd-map:Incident` (nicht zu verwechseln mit der oben beschriebenen QUDT-Einheit `fo:Incident`) repräsentiert die Menge der aufgetretenen und in der Berechnung der Metriken zu berücksichtigenden Incidents, `hzd-map:MonitoredEntity` enthält die Entitäten der Domänenontologie, denen die Incidents zugeordnet werden, im Use-Case also VMs. Zusätzlich werden zwei abstrakte Rollen zugefügt: `hzd-map:hasRelatedIncident` stellt die Verbindung zwischen einer VM und einem Incident her, `hzd-map:contributesToMetric` macht die Verbindung zwischen Incident und formaler Metrik explizit. Die Beispielinstanz eines Incidents, `hzd-map:i0` zeigt die Zuordnung zu VM und Metrik. Sowohl Monitored Entities als auch Incidents können über weitere Rollen Attribute und Relationen zu anderen Teilmodellen enthalten, beispielsweise hat die VM Verbindungen zum Storage-System.

Zur Beschreibung der Berechnungsvorschriften der formalen Metriken können schließlich `agg:PostProcessingDirectives` angelegt werden, analog zur Berechnung der Aggregationen des SVC-Modells in Abbildung 8.5. Die Berechnungsvorschriften können auf alle Teilmodelle Bezug nehmen und sind damit nicht auf die Berücksichtigung von Werten innerhalb einer Domäne beschränkt. Obwohl alle Aspekte in inhaltlich getrennten Ontologien bzw. Ontologie-Modulen angelegt sind, kann jederzeit der Bezug zwischen Monitored Entities, den Metriken, zu deren Berechnung sie beitragen und darüber auch der Beitrag zu Zielen aus COBIT-Sicht nachvollzogen werden.

8.2.4 Ergebnisse

Entwicklungsaufwand

Entsprechend des Vorgehenskonzepts aus Abbildung 8.1 wurden Transformationswerkzeuge entwickelt, welche die vorliegenden aber proprietären und schlecht für eine Weiterentwicklung und Integration geeigneten Formate für Modelle und Verhaltensvorschriften in Ontologien und Regelmengen des Laufzeitsystems übersetzen. Die Konstellation, in der Ausgangsmodelle und -Formate auf Eigenentwicklungen basieren, ist repräsentativ für viele potentielle Einsatzszenarien, in denen semantische Technologien zum Einsatz kommen können. Der Anwendungsfall ist daher gut geeignet, beispielhaft den nötigen Aufwand für die Adaption des Ansatzes und des generischen Laufzeitsystems auf eine konkrete Problemstellung zu untersuchen. Insbesondere aus Sicht der Kooperationspartner ist diese Aufwandsuntersuchung im Hinblick auf zukünftige Erweiterungen hilfreich. Während innerhalb der Forschungskooperation in Entwicklungsphase 1 alle technischen Grundlagen geschaffen wurden, wurde die Erweiterbarkeit des Systems in Phase 2 um technische und in Phase 3 um nicht-technische Domänen exemplarisch untersucht. Diese Phasen und die darin entwickelten Teilergebnisse sind daher diejenigen, deren Umsetzbarkeit und Entwicklungsaufwand mit der auf dem ursprünglichen System basierenden entsprechenden Alternative verglichen wurden.

Es fand eine Anpassung des Laufzeitsystems für den Einsatz als Komponente in BVQ statt, sowie eine Entwicklung der entsprechenden Programmteile. Außerdem wurden die nicht durch Werkzeuge generierten Ontologien, wie die COBIT-Ontologie, mittels der Ontologie-Entwicklungsumgebung Protégé umgesetzt. Tabelle 8.3 zeigt Statistiken über den Umfang der für den Use-Case entwickelten Ontologien, Regelmengen und Code-Fragmente. Für Programmcode wurden Java und Scala eingesetzt, für Ontologien OWL unter Verwendung der Turtle-Syntax (TTL) und für Regelmengen die Syntax der Jena-Regelengine. Die Tabelle zeigt die Zugehörigkeit der Teilergebnisse zu den jeweiligen Entwicklungsphasen.

Tabelle 8.3: Statistiken über die Umsetzung des Use-Case. Mit * markierte Einträge sind durch Werkzeuge generiert.

Beschreibung	**Typ**	**Anzahl Zeilen**	**Phase**
Transformer für SVC- und BVQ-Modell	Scala-Code	996	1
Transformer für Aggregationsvorschriften	Scala-Code	2.401	1
Transformer für vSphere-XSD	Scala-Code	306	2
Laufzeitsystem und Adapter	Java-Code	2.506	1
Editor für Jena-Regelsprache	Java-Code	1.077	1
Adapterschicht für BVQ-Oberfläche[4]	Java-Code	1.857	1
Serialisierung in Neo4j-Graphdatenbank[5]	Java-Code	1.730	2
SVC-Ontologie*	Ontologie (TTL)	2.157	1
BVQ-Ontologie*	Ontologie (TTL)	40.044	1
Nachverarbeitungsvorschriften*	Ontologie (TTL)	48.570	1
VMware vSphere Ontologie*	Ontologie (TTL)	87.954	2
SVC-Adapter Ontologie	Ontologie (TTL)	90	1
Ontologien für Wurzelmodul	Ontologie (TTL)	56	1
HZD-Ontologie	Ontologie (TTL)	181	3
OWL 2 RL Regelsatz	Regelmenge	778	1
Regeln für BVQ-Modell*	Regelmenge	1.895	1
Regeln für SVC-Adapter*	Regelmenge	831	1

Da die Entwicklung und Integration des Laufzeitssystems in die bestehende BVQ-Software sowie die Umsetzung von Transformationswerkzeugen von Ausgangsformaten in Ontologien und Regelmengen in Phase 1 stattfinden, ist diese Phase erwartungsgemäß mit ca. 18 Monaten Dauer die umfangreichste. Für die Modelle für SVC und BVQ wurden spezifische Transformationswerkzeuge entwickelt, da die jeweiligen Ausgangsformate auf Eigenentwicklungen der Firma SVA basieren. Insbesondere das Format zur Definition der Aggregationsvorschriften entspricht im Umfang einer deklarativen Programmiersprache wie XSLT, ist aber nur in Form

[4]Nicht vom Autor dieser Arbeit entwickelt
[5]Nicht vom Autor dieser Arbeit entwickelt

der auswertenden Implementierung spezifiziert. Daher ist dieser Transformer für Aggregationsvorschriften mit ca. 2.400 Zeilen Scala-Code der umfangreichste in Phase 1.

In der ursprünglichen Implementierung von BVQ sind die Taxonomien der SVC- und BVQ-Modelle und die Nachverarbeitungsvorschriften in zusammen ca. 37.000 Zeilen von XML-basierten Formaten beschrieben. Die Semantik der SVC- und BVQ-Modelle ist weitestgehend implizit im Programmcode des Verarbeitungskerns von BVQ enthalten. Der Transformer für die Übersetzung dieser Modelle nutzt die Taxonomien aus den XML-Formaten, reichert sie mit der entsprechenden Semantik aus dem Verarbeitungskern an und generiert Ontologien und Regelmengen. Der ursprüngliche Verarbeitungskern von BVQ umfasste ca. 27.000 Zeilen Java-Code. Die Erweiterungen des Laufzeitsystems, die für den Use-Case und den vorigen Verabeitungskern entwickelt wurden, machen etwa 2.500 Zeilen Java-Code und 1.900 Zeilen generierter Regeln aus. Dabei nicht eingerechnet sind der Umfang des generischen Teil des Laufzeitsystems (siehe Tabelle 7.2) sowie der nötigen Transformationswerkzeuge.

In Phase 2 war der wesentliche Entwicklungsaufwand für das Transformationswerkzeug des XML-Schemas der vSphere-Schnittstelle in eine entsprechende Ontologie nötig. Dieses Transformationswerkzeug ist weit weniger komplex als die in Phase 1 entwickelten, da das Schema nur die Datenstrukturen der Schnittstelle beschreibt, aber keine weiteren Restriktionen oder Verhaltens- oder Berechnungsvorschriften. Zusammen mit der nachfolgenden manuellen Erstellung der Mapping-Ontologie zwischen den SVC- und vSphere-Ontologien sowie der Integration in das Laufzeitsystem durch Kapselung in Ontologie-Module betrug der Entwicklungsaufwand für diese Phase ca. zwei Monate. Basierend auf der ursprünglichen Implementierung des Verarbeitungskerns von BVQ hätte diese Erweiterung nach Schätzungen des BVQ-Entwicklungchefs zwischen sechs und zehn Monaten gedauert.

Für die Realisierung von Phase 3 wurde hauptsächlich die HZD-Ontologie entwickelt und Ontologie-Module für sie und die von ihr genutzten externen Abhängigkeiten umgesetzt. Die Entwicklung von Transformationswerkzeugen bzw. Adaptern an externe Datenquellen fand in dieser Phase nicht statt. Die Entwicklungszeit für die HZD-Ontologie betrug zusammen mit der Integration in die SVC-, FIBO- und COBIT-Ontologien nicht mehr als einen Monat – eine vergleichbare Entwicklung des Modells auf Basis der ursprünglichen Implementierung hätte nach Schätzungen des BVQ-Entwicklungchefs ca. sechs Monate gedauert.

Performance-Messungen

Die prototypische Implementierung wird anhand des Use-Case quantitativ und qualitativ evaluiert. Es werden die Laufzeiten eines Abfrage- und Reasoningzyklus und der Speicherverbrauch des Laufzeitsystems in zwei unterschiedlichen Konfigurationen gemessen. Für jede Konfiguration werden Messungen an jeweils zwei SVC-Hostsystemen durchgeführt.

Die erste Konfiguration, *Debug* genannt, ist für Entwicklung und Debugging des Laufzeitsystems sowie der eingesetzten Ontologien hilfreich. Dabei werden das Logging von Schlussfolgerungen durch die Regelengine sowie die Serialisierung der durch Abfrage und Reasoning erzeugten Ontologie inklusive ABox aktiviert. Das Logging von Schlussfolgerungen erfolgt auf dem Heap, da Rückschlüsse aller Teilschlussfolgerungen vorgehalten werden müssen. Auf diese Weise können alle Einzelschritte, die zu einer Entscheidung durch die Regelengine beitragen, nachvollzogen werden, es handelt sich aber um eine für Laufzeit und Speicherverbrauch teure Operation. Die Debug-Konfiguration hat sich während der Entwicklung des Laufzeitsystems und des Use-Cases als so hilfreich herausgestellt, dass ihre Performanceeigenschaften hier separat betrachtet werden sollen. Analog dazu wird in der Konfiguration *Produktion* das Laufzeitsystem auf den Betrieb für den Einsatz bei BVQ-Kunden eingestellt. Logging von Schlussfolgerungen der Regelengine und die Serialisierung von durch Reasoning erzeugen Ontologien wird deaktiviert. Hierdurch sinkt sowohl die Bearbeitungsdauer für einen Reasoning-Zyklus als auch der Speicherbedarf.

Das erste Hostsystem, das zur Messung verwendet wird, im Folgenden als *Host 1* bezeichnet, ist ein SVC in Version 6.4, der bei der Entwicklung und zum Test von BVQ eingesetzt wird. Sein Datensatz besteht im Wesentlichen aus synthetischen Daten und Testdaten. Der Umfang des Datensatzes beträgt nach Serialisierung durch das Laufzeitsystem in RDF ca. 32.500 Tripel. Das zweite Hostsystem, im Folgenden *Host 2* genannt, ist ebenfalls ein SVC in Version 6.4, dessen Datensatz das Modell des Speichersystems des nach Datenumfang größten BVQ-Kunden enthält und ca. 116.000 Tripel umfasst.

Abbildung 8.9 zeigt die Laufzeiten zur Bearbeitung eines Zyklus, der Schritte ① bis ③ des Datenflusses aus Abbildung 5.5 umfasst. Dabei werden die Taxonomie des Speichersystems vom SVC über die SSH-basierte Schnittstelle abgefragt, das resultierende RDF durch die modellspezifischen Regeln in eine dem OWL-Schema entsprechende Form gebracht, OWL 2 RL-Reasoning ausgeführt und transiente Entitäten und Rollen entfernt. Für die Messung wurde nur das Laufzeitsystem mit den in dieser Arbeit vorgestellten Bestandteilen alleinstehend getestet, also nicht die gesamte BVQ-Software, die das Laufzeitsystem als Modul lädt.

Die Schritte der Abfrage des Speichersystems, der Vorverarbeitung des daraus resultierenden RDF und das OWL 2 RL-Reasoning laufen weitestgehend parallel ab und können nicht sinnvoll separat vermessen werden. Sie werden daher in Abbildung 8.9 gemeinsam als ein Schritt („Polling und Reasoning“) dargestellt. In Debug- und Produktions-Konfigurationen beträgt er bei Host 1 ca. 10 Sekunden und bei Host 2 ca. 30 Sekunden. In jedem Reasoning-Zyklus ist dieser Schritt zentral, während die Entfernung der transienten Entitäten und Rollen mit ca. 180 ms bei Host 1 und ca. 500 ms bei Host 2 sowie die Serialisierung der Ontologie mit ca. 500 ms bei Host 1 und ca. 1.900 ms bei Host 2 insgesamt kaum ins Gewicht fallen. In der Debug-Konfiguration kommt zusätzlich die Laufzeit für das Logging von Schlussfolgerungen hinzu: Für Host 1 beträgt das Logging mit ca. 26 Sekunden mehr als zwei Drittel der gesamten Laufzeit, bei Host 2 ist der Anteil mit ca. 200 von 230 Sekunden ca. 87%.

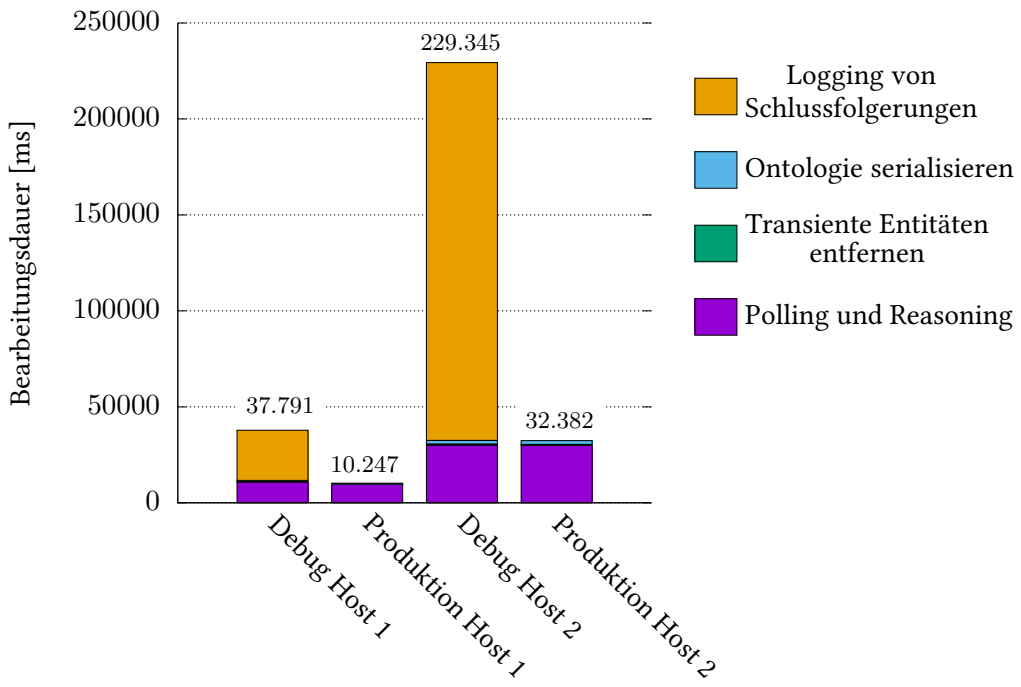

Abbildung 8.9: Laufzeiten zur Bearbeitung eines Reasoning-Zyklus zweier SVC-Systeme

Es wird nicht die gesamte Menge der Nachverarbeitungsvorschriften (Schritte ④ und ⑤ in Abbildung 5.5) in jedem Reasoning-Zyklus ausgewertet. Stattdessen wird abhängig von der Auswahl in der Bedienoberfläche immer nur diejenige Untermenge ausgewertet, die zur Anzeige der entsprechenden Werte notwendig ist. Die Berechnung aller Nachverarbeitungsvorschriften beträgt für Host 1 ca. drei Minuten, für Host 2 ca. zehn Minuten.

In einem realistischen Anwendungsszenario werden SVC-Systeme und Managementwerkzeug am selben Standort eingesetzt, die hier durchgeführten Tests wurden jedoch auf einem räumlich von den Hostsystemen getrennten, ca. 10 km entfernten und per VPN angebundenen Rechner durchgeführt. Hieraus resultiert eine Latenz zwischen etwa 20 und 100 Millisekunden, die angesichts der Gesamtdauern vernachlässigbar ist. Die Messungen wurden auf einer Workstation mit Intel Core i7-3770 CPU und 16 GB Arbeitsspeicher ausgeführt. Die eingesetzte Java-Version ist 1.7.0 Revision 45. Die Messung wurde durch Logging des Laufzeitsystems durchgeführt.

Abbildung 8.10 zeigt für beide Hosts und Konfigurationen den Speicherverbrauch. Dabei wird der durch die Java Virtual Machine allokierte maximale Heap (Heap Max) sowie der tatsächlich benötigte, maximale Heap (Heap Used Max) angegeben. Die Messung des Speichers wurde

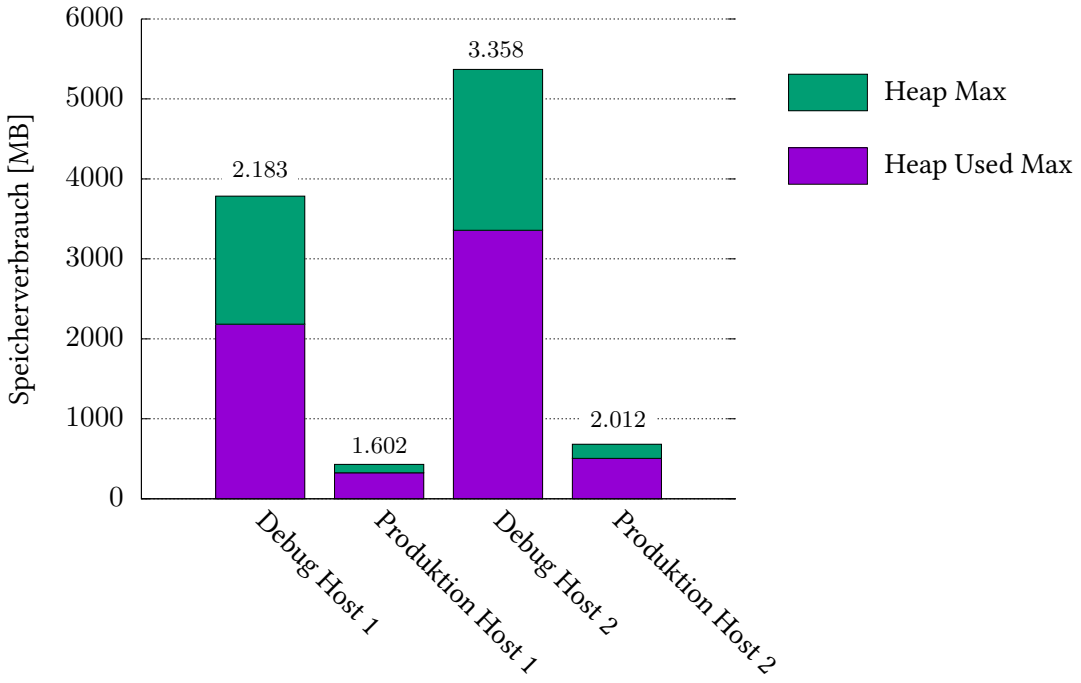

Abbildung 8.10: Speicherverbrauch bei Bearbeitung eines Reasoning-Zyklus zweier SVC-Systeme

mittels JVisualVM vorgenommen. Bei der Messung des Speicherverbrauchs wurde ebenfalls das Laufzeitsystem alleinstehend gemessen.

Laufzeiten und Speicherverbrauch verdeutlichen die unterschiedlichen Anforderungen an das System zum Entwicklungszeitpunkt und für den produktiven Einsatz. Die wesentliche Erkenntnis, insbesondere unter Betrachtung der Laufzeit bei Bearbeitung des Modells von Host 2, ist die Einsetzbarkeit des Ansatzes in einer interaktiven Anwendung wie BVQ: Die Laufzeiten bei Einsatz der Produktions-Konfiguration sind hier dem Augenschein nach sehr ähnlich zu denen der ursprünglichen Implementierung in BVQ. Eine konkrete Messung bzw. ein direkter Vergleich konnte mangels der Instrumentierbarkeit der ursprünglichen Implementierung nicht vorgenommen werden. Dabei ist zu berücksichtigen, dass das Laufzeitsystem und die eingesetzte Regelengine generische Softwarekomponenten sind, während die ursprüngliche Implementierung speziell für den Anwendungszweck entwickelt wurde.

8.2.5 Diskussion

Das Ziel des Ansatzes, die einfache Erweiterbarkeit um zusätzliche Domänenmodelle zu gewährleisten, die in dieser Form in der ursprünglichen Implementierung nicht umzusetzen wäre, wurde im Rahmen des Use-Case bestätigt. Wie sich während der Bearbeitung des Use-Case gezeigt hat, wird die angestrebte Verbesserung der Erweiterbarkeit in erster Linie durch die vereinheitlichte Kapselung der Domänensemantik in Form von zusammengefasster Module erreicht. Während im vorigen System die Semantik von Modell und zugehörigen Berechnungen verteilt war auf Strukturen in spezifischen XML-Formaten und hartkodiert in der zugehörigen Anwendung, wurde die Darstellung von Informationen in der neuen Lösung vereinheitlicht. Dadurch, dass die neue Repräsentation auf Standardformaten basiert, können zu ihrer Bearbeitung auch Standardwerkzeuge wie Protégé eingesetzt werden, was die Aufrechterhaltung der Konsistenz des Modells stark vereinfacht, insbesondere im Vergleich mit der Wartung von entsprechendem Programmcode. Ebenso trägt die deklarative Natur der Ontologien und Regelmengen zur Erweiterbarkeit bei: bei ausreichender Kenntnis der Formalismen und entsprechender Werkzeugunterstützung können nicht nur die Ontologien bereits zum Entwicklungszeitpunkt von Inkonsistenzen befreit werden, sondern auch die Ontologie-Module so angelegt werden, dass eine nachträgliche Ergänzung beliebiger Konzepte widerspruchsfrei möglich ist. Würde das Modell als Teil des Programmcodes gepflegt, wären vergleichbar flexible nachträgliche Zusätze zumeist nicht ohne Änderung der Schnittstellen möglich. Hier konnte das Ontologie-Modul zur Unterstützung des vSphere-Modells sowie des Mappings auf die SVC-Ontologie nach dem erfolgreichen Funktionstest des Laufzeitsystems entwickelt und zugefügt werden, ohne dass Änderungen am bestehenden Programmcode nötig wurden.

Zusätzlich erlaubt der Einsatz von OWL-Ontologien als zentralem Format für das Informationsmodell auch die Einbindung und Nutzung von extern verfügbaren Modellen, wie formalen Varianten relevanter Standards bzw. Industriestandards, sofern solche verfügbar sind. Dies wurde im Use-Case demonstriert durch den Einsatz von FIBO, QUDT und COBIT und ermöglichte die Umsetzung und Integration des HZD-spezifischen Informationsmodells in viel kürzerer Zeit, als eine äquivalente Anpassung des Programmcodes benötigt hätte.

Laufzeit und Speicherverbrauch des prototypischen Laufzeitsystems können für den Einsatz im Use-Case generell als zufriedenstellend beurteilt werden, da sie vergleichbare Werte wie das vorige, weitestgehend hartkodierte System erreichen. Bei den Laufzeiten der Auswertung der Nachverarbeitungsvorschriften sind hingegen zahlreiche Optimierungen realisierbar. Wie die Beispielabfrage in Abbildung 8.5 zeigt, entsteht durch die rekursive Übersetzung des Ausgangsformats eine Struktur mit vielen Subqueries, die durch eine Kombination von Verbesserungen am Transformationswerkzeug und manuellen Anpassungen reduziert werden könnte. Nachverarbeitungsvorschriften, die nur Graph-Patterns aber keine Aggregationsfunktionen von SPARQL nutzen, könnten zudem automatisch in Regeln umgeschrieben werden, die während des Reasoning-Prozesses ausgewertet werden können. Schließlich wäre für die schnellere

Auswertung häufig benutzter Vorschriften auch eine Generierung von Bytecode denkbar, der sowohl das Parsen der Abfrage als auch die Navigation durch den Graph abkürzen könnte.

Es hat sich gezeigt, dass die Weiternutzung existierender Ontologien, bzw. in Ontologien transformierbare Modelle in anderen Formaten, nicht in allen Fällen ohne Einschränkungen möglich ist, so waren beispielsweise die von der DMTF spezifizierten Modelle für Speicher- und Software-Virtualisierung im Use-Case nicht unmittelbar einsetzbar, weil sie durch die Schnittstellen des SVC nicht in ausreichendem Umfang unterstützt wurden, wie in Abschnitt 8.2.2 beschrieben.

Da COBIT als Planungshilfe bei der HZD noch an Bedeutung gewinnt und mangels einer bestehenden Infrastruktur zur Verarbeitung und Nutzung des formalen COBIT-Modells zur automatischen Abrechnung augenblicklich noch nicht eingesetzt werden kann, kann dieser Teil der Arbeit nur indirekt bewertet werden. Nach Gesprächen mit dem Storage Manager der HZD wird für diesen Einsatz ein großes Potenzial bestehen, den bestehenden Prozess zu beschleunigen und flexiblere Konfigurationsmöglichkeiten für die HZD-Kunden anbieten zu können. Eine notwendige Voraussetzung hierfür ist jedoch die Erweiterung der grafischen Oberfläche von BVQ, sodass die Verwaltung, Visualisierung und Abfrage der nunmehr generischeren Wissensmodelle (also nicht nur SVC, sondern auch kunden- bzw. anwendungsspezifische Modelle) auch ohne tiefe Kenntnisse über Ontologien und semantische Technologien möglich ist.

9 Zusammenfassung und Ausblick

9.1 Ergebnisse der Arbeit

Mit steigender Komplexität von IT-Systemen wird deren Management ebenfalls komplexer, weswegen sowohl *automatisiertes IT-Management*, also die Reduktion von manuellen Schritten bei der Verwaltung, als auch *integriertes IT-Management*, also die gemeinsame Betrachtung von Anwendungen, Informationen und Teilsystemen weitreichende Anwendung finden. Dabei sollen auch Zusammenhänge zwischen Elementen der einzelnen Aspekte betrachtet werden, die implizit bestehen. Es existieren bereits Methoden und Werkzeuge, die automatisiertes und integriertes IT-Management sowohl separat als auch in Kombination betrachten. Auf der anderen Seite ist ein stetiges Wachstum der Bedeutung der Integration von IT-Management und *IT-Governance* zu beobachten. Während sich das IT-Management weitestgehend mit der technischen Sicht auf Systeme beschäftigt, setzt sich die IT-Governance damit auseinander, die Zusammenhänge zwischen den Anforderungen, die sich aus Sicht der Unternehmensführung an die IT stellen, und den Systemen, welche die IT letztlich befähigen, methodisch umzusetzen. Für die IT-Governance existieren etablierte Vorgehensweisen, diese sind jedoch für eine menschliche Durchführung entworfen, lassen sich also nicht ohne Weiteres algorithmisch verarbeiten.

Das Ziel dieser Arbeit ist ein Beitrag zur Konvergenz der beiden Disziplinen, so dass einerseits Vorgehensweisen des automatisierten IT-Managements so erweitert werden, dass sie nicht mehr nur auf technische Aspekte des Gesamtsystems anwendbar sind, sondern auch die in der IT-Governance angesiedelten nicht-technischen Aspekte unterstützt werden können. Andererseits müssen die Kernkonzepte der IT-Governance formal so erfasst werden, dass sie durch ein System für automatisiertes IT-Management verarbeitbar werden. Erst durch diese Konvergenz wird es möglich, implizit bestehende Zusammenhänge zwischen der IT und der durch sie unterstützten Fragen der Geschäftssicht in der Formulierung von Automatisierungsregeln zu nutzen, die auf alle Teile des Gesamtsystems und alle Abstraktionsebenen der Informationen eines Unternehmens Bezug nehmen können. Schließlich wird es hierdurch möglich, eine Automatisierung auch über die Grenzen der technischen Sichten hinaus zu realisieren. Da automatisiertes IT-Management sowie IT-Governance eine Vielzahl inhaltlich separater und strukturell unterschiedlicher Domänen abdecken, muss das jeweilige Domänenwissen modular erfassbar sein und das Vorgehen für die Verküpfung von Domänenwissen die Erweiterung um zusätzliche Domänenmodelle erlauben.

Der Lösungsansatz der Arbeit basiert auf der formalen Beschreibung der Semantik von Teilmodellen des Gesamtsystems in Form von Ontologien und Regelmengen. Dabei werden nicht nur die Entitäten und Zusammenhänge innerhalb einer Sichtweise oder Domäne beschrieben, sondern auch die sukzessive nachträgliche Erweiterung um weitere Domänenmodelle und Abbildungen zwischen bestehenden Domänenmodellen ermöglicht, ohne bestehende Systembestandteile anpassen zu müssen. Eine wichtige Eigenschaft ist dabei der Einsatz offener Standards zur Formalisierung der Ontologien, wodurch die Wiederverwendbarkeit allgemeingültiger Teile und Modelle zur Repräsentation relevanter formaler Standards und von Industriestandards erreicht wird.

In der Arbeit wurden zunächst die Begriffe und Konzepte des IT Service Managements und der IT-Governance erklärt, sowie der Aufbau von COBIT, dem De-Facto-Standard-Rahmenwerk für IT-Governance. Anschließend wurde aufgrund seiner Wichtigkeit für die Arbeit ein umfassender Überblick über die formalen Grundlagen, die maßgeblichen Formate und Standards zur Definition von Ontologien sowie den Stand der Forschung zum Ontology Engineering gegeben.

Nach einer Aufarbeitung von Literatur zu den einzelnen inhaltlichen Bestandteilen der Fragestellung wurde festgestellt, dass, obwohl von mehreren Seiten ein entsprechender Bedarf begründet wurde, die gemeinsame Betrachtung aus formalem Modell für die IT-Governance und Verfahren für das automatisierte IT-Management weitestgehend unbearbeitet ist. Aus dieser Aufarbeitung ergeben sich die Anforderungen zur Bewältigung der größten Hürden hierbei. Erstens wird eine Möglichkeit benötigt, die Informationsmodelle unterschiedlicher technischer und nicht-technischer Domänen in einer Weise auszudrücken, welche die exakte Beschreibung der Zusammenhänge zwischen ihnen erlaubt. Zweitens besteht die Notwendigkeit für Konzepte und Architektur eines geeigneten Softwaresystems zur Verarbeitung der Verknüpfung von Domänenmodellen, die auch die Automatisierung auf Basis der Verbindung von „klassischen“ IT-Management-Modellen mit IT-Governance-Modellen umfasst. Drittens wird ein umfassendes formales Modell für einen etablierten IT-Governance-Standard benötigt, idealerweise für COBIT.

Das erste zentrale Ergebnis der Arbeit ist daher ein Konzept für die Modularisierung einer formalen Wissensbasis, in der Wissen über einzelne Domänen verknüpft und verarbeitet werden kann. Die Erfassung des jeweiligen Domänenwissens wird in Form einer Menge von OWL-Ontologien realisiert, die zusammen mit Regelmengen zur Beschreibung von Verarbeitungsvorschriften, Software-Fragmenten als Adapter zu externen Datenquellen sowie Metadaten zur Spezifikation von Abhängigkeiten zu *Ontologie-Modulen* zusammengefasst werden.

Erst durch diese Kapselung inhaltlich stark gekoppelter Informationsfragmente in abgeschlossene Einheiten wird die saubere Trennung der Wissensbasis an *Domänengrenzen* möglich. Während etablierte Vorgehensweisen des Software Engineering Grenzen von Domänenmodellen in der Modellierungsphase auf konzeptueller Ebene identifizieren, erlaubt die Strukturierung in Ontologie-Module darüber hinaus auch die Aufrechterhaltung dieser Grenzen in

den darauf folgenden Entwicklungs- und Betriebsphasen. Eine Ontologie ist per Definition die formale Spezifizierung einer gemeinsam genutzten Konzeptualisierung (siehe Seite 23), soll also explizit den Austausch von erfasstem Wissen ermöglichen. Im Kontext einer Verarbeitung folgt auf die Zusammenstellung und Verknüpfung geteilter Ontologien notwendigerweise eine Ergänzung um die Beschreibung domänenspezifischer Verhaltensweisen. Obwohl die Verhaltensweisen impliziter Bestandteil des Domänenmodells sind, führt ihre explizite Einführung außerhalb der eigentlichen Ontologie zu einer Aufweichung des Domänenmodells; das Domänenmodell existiert nicht mehr als eine Einheit, sondern ist verteilt auf Ontologie und sie verarbeitende Implementierung. Ontologie-Module erweitern das Konzept der gemeinsamen Nutzung über ein reines Informationsmodell hinaus so, dass zusammen mit dem Informationsmodell alle für den praktischen Einsatz nötigen Bestandteile ebenfalls mitgeliefert werden. Hierdurch bleibt das Domänenmodell auch *nach* der Zusammenstellung und Verknüpfung als Einheit erhalten und kann als solches weiterentwickelt, geteilt oder abgefragt werden.

Das zweite zentrale Ergebnis ist ein erweiterbares Laufzeitsystem zum regelbasierten automatisierten IT-Management. Das System verwaltet im Kern eine Menge von Ontologie-Modulen, die separat entwickelt werden können und sich zur Laufzeit untereinander referenzieren können. Für Module können Abhängigkeiten an weitere Module festgelegt werden, wodurch auch Abhängigkeiten zwischen Modul-Bestandteilen unterschiedlicher Module abgedeckt werden. Obwohl also Abhängigkeitsmechanismen für Ontologien und Software auch vorher verfügbar waren, wird die Definition von Abhängigkeiten durch das Modulsystem erstmals auf Ebene der logischen Einheiten von Modul-Bestandteilen unterschiedlichen Typs möglich. Auf diese Weise kann eine Instanz des Laufzeitsystems also zu beliebigen nachträglichen Zeitpunkten um Module erweitert werden, die Ontologien bestehender Module ergänzen, erweitern oder Konzepte unterschiedlicher Module zueinander in Relation setzen. Das Laufzeitsystem stellt einen regelbasierten OWL 2 RL-Reasoner und die Möglichkeit zur Auswertung von Nachverarbeitungsvorschriften zur Verfügung, über die in Kombination der Datenfluss einer Management-Regelschleife deklarativ und unter Nutzung von Standardformaten beschrieben werden kann.

Die prototypische Implementierung des Laufzeitsystems basiert auf dem OSGi-Rahmenwerk, das unter anderem wohldefinierte Schnittstellen sowie ein Lebenszyklusmodell für Service-Komponenten hat. Die Struktur der Ontologie-Module wird in der Architektur des Laufzeitsystems daher auf OSGi-Komponenten abgebildet. Dies reduziert nicht nur den Entwicklungsaufwand, sondern ermöglicht auch die nahtlose Integration von Ontologie-Modulen mit weiteren Softwarekomponenten und Bibliotheken, die beispielsweise für die Umsetzung von Adaptern zu externen Systemen benötigt werden. Im Vergleich mit der hierzu üblichen Entwicklung separater Softwarekomponenten sind dadurch zusammengehörige Informationen besser gekapselt und besser erweiterbar.

Das dritte zentrale Ergebnis, die entwickelte COBIT-Ontologie, stellt die erste vollständige Formalisierung der Kernstrukturen von COBIT dar, die in der ausschließlich aus textuellen Dokumenten bestehenden offiziellen Spezifikation des Rahmenwerks implizit enthalten sind. Bei der

Umsetzung als Ontologie im OWL-Format wurden in mehreren Teilschritten an der Spezifikation orientierte und ergänzende Nomenklaturen für Entitäten entwickelt und Klassen, Rollen, Individuen und Beziehungsaussagen eingeführt, die das Informationsmodell darstellen. Außerdem wurden Entitäten und Beziehungsaussagen über Annotationen in Beziehung gesetzt mit stellvertretenden Individuen, die Abschnitte, Tabellen und Abbildungen der ursprünglichen Spezifikation repräsentieren. Auf diese Weise kann der Zusammenhang zwischen Aussagen des formalen Modells und der zugruende liegenden Beschreibung des Rahmenwerks nachvollzogen werden, bei Bedarf auch programmatisch. Für die Umsetzung der COBIT-Ontologie wurde nach Untersuchung bestehender Ansätze als ein Teilergebnis eine neue Ontologie für die Formulierung verketteter Listen in OWL entwickelt. Außerdem wurde ein Konzept vorgestellt, mit dem unter Verwendung des QUDT-Rahmenwerks zur Beschreibung von Einheiten in OWL-Ontologien die in COBIT als Fließtext formulierten Metriken zur Messung des Erreichungsgrades von Geschäfts-, IT- und Prozesszielen programmatisch auswertbar umgesetzt werden können. Hierdurch können COBIT-Metriken so formalisiert werden, dass explizite Referenzen auf die in der Metrik erwähnten Entitäten hergestellt werden können. Dadurch, dass in dieser Formalisierung Entitäten aus beliebigen Domänenontologien referenziert werden können, wird die direkte Auswertung von Metriken möglich, die sich inhaltlich über mehrere Domänen erstrecken, ohne dass hierzu bei jeder Auswertung Übersetzungen zwischen den einzelnen Domänenmodellen nötig werden.

In einem umfassenden Use-Case wurden Konzepte und Implementierung eingesetzt und anhand der Fragen, die sich während des praktischen Einsatzes stellten, auch zur Bewertung des gesamten Ansatzes weiterentwickelt. Insgesamt konnte das Vorgehen des Lösungsansatzes, bei dem zwischen Modell und Mapping, Transformation, Wissensbasis und Laufzeitsystem unterschieden wird, erfolgreich umgesetzt werden (siehe Abbildung 5.1). Sowohl die Konzepte der Ontologie-Module als auch des regelbasierten Laufzeitsystems wurden im Rahmen des Use-Case evaluiert. Hierbei entstanden weitere konkrete Ergebnisse, die in erster Linie für den Use-Case relevant sind, wie die Ontologie für den IBM SAN Volume Controller, aber auch indirekte Ergebnisse in Form von Erkenntnissen, die breitere Anwendung finden können.

Eine wichtige Erkenntnis, die sich durch Leistungsmessungen ergeben hat, ist, dass ein System basierend auf der Kombination von Ontologien und Regelmengen als zentralem Informationsmodell eine vergleichbare Laufzeitperformance erreichen kann wie ein System, in dem die entsprechenden Teile hartkodiert sind, gleichzeitig aber eine geringere Menge an Programmcode erforderlich ist. Ebenso hat sich am Beispiel der VMware-Ontologie gezeigt, dass durch den Ansatz, semantische Domänenmodelle als Module gekapselt zu beliebigen nachträglichen Zeitpunkten zufügen zu können, die Integration der Unterstützung für zusätzliche Domänen mit deutlich geringerem zeitlichen Aufwand realisierbar ist als bei einer Lösung basierend auf reiner Softwareentwicklung. Hierfür sind jedoch tiefgehende Kenntnisse der zugrunde liegenden Formalismen nötig sowie gegenüber der reinen Softwareentwicklung zusätzliche technische Vorkehrungen für das Debugging der Verknüpfung von Modellen. Schließlich wurde im Use-Case erfolgreich demonstriert, dass semantische Modelle technischer und nicht-technischer

Domänen gemeinsam in einem Laufzeitsystem integriert werden können, so dass Beziehungen zwischen ihnen so explizit ausgedrückt werden können, dass Automatisierungsvorschriften sich über ihre Verknüpfung erstrecken können. Während dies im Use-Case mit einer Kombination von Modellen für Storage-Virtualisierung, Software-Virtualisierung, Accounting und IT-Governance gezeigt wurde, ist prinzipiell auch die Erweiterung um beliebige weitere Domänen möglich.

9.2 Ausblick

Auf dem Weg, all diejenigen Informationen innerhalb eines Unternehmens verknüpfen zu können, die zur größeren Transparenz, Reduktion des Aufwandes von Prozessen und Routineaufgaben und Vereinfachung von Auditing beitragen können, ist ein formales Modell für die IT-Governance nur ein Bestandteil. Die COBIT-Ontologie kann als eine Brücke zwischen IT und der Sicht der Geschäftsführung auf die IT verstanden werden, es gibt aber zahlreiche weitere Aspekte innerhalb eines Unternehmens, die in einer umfassenden Integration von Informationen berücksichtigt werden sollten. In der Literatur existieren Arbeiten zur Formalisierung einzelner dieser Aspekte durch Ontologien; so entwickelt beispielsweise Dittmann in [Dit07] eine Ontologie für FMEA (Failure Mode and Effects Analysis, dt. *Fehlermöglichkeits- und -einflussanalyse*), einer etablierten analytischen Methode zur Berechnung der Wahrscheinlichkeiten von Auftritt und Entdeckung von Produktfehlern, bei der in der Regel nur abgeschlossene Berichte zur Risikoanalyse geschaffen werden. Hierbei bestehen Schnittmengen mit dem Risk Management, das seit Version 5 Teil von COBIT ist. Eine Verbindung beider Ontologien durch eine Abbildungsontologie könnte die Anzahl der Einsatzmöglichkeiten erhöhen und zusammen mit dem vorgestellten Laufzeitsystem die Bedingungen zur Automatisierung des Risk Managment verbessern, indem die Erstellung von abgeschlossenen Berichten ergänzt bzw. sukzessive ersetzt wird durch eine Onlineanalyse basierend auf jeweils aktuellen Messwerten. Eine derartige Weiterentwicklung wäre nur durch eine entsprechende Verknüpfung der Ontologien für FMEA mit den Ontologien der betrachteten technischen Systeme möglich.

Eine technische Domäne, deren Integration sich in einer gemeinsamen Betrachtung anbietet, ist der Aufbau und die Konfiguration eines Netzwerks. Mit dem Anstieg der Verbreitung und Bedeutung von SDN (Software-defined Networking), bei dem Netzwerke programmatisch initialisiert, gesteuert und geändert werden, wachsen auch die Möglichkeiten für automatisierte Konfiguration und Monitoring. Werden diese Aspekte des Netzwerks nicht mehr isoliert betrachtet, sondern mit den Modellen der Anwendungen und Prozesse verknüpft, die das Netzwerk nutzen, ergeben sich hierbei neue Möglichkeiten der Automatisierung auf einer hohen Abstraktionsebene, beispielsweise die dynamische Umkonfiguration anhand von Indikatoren auf Prozessebene. Martínez et al. erfassen in [MYD+15] erstmals die Semantik der Konfiguration von Switches und Routern in einer OWL-Ontologie, um die Konfiguration von hybridem SDN – einer Mischung von SDN und „klassischen“ Netzwerkgeräten – zu erleichtern. Dies könnte als Grundlage der Integration der Netzwerkdomäne dienen.

Für die Unternehmensarchitektur (Enterprise Architecture), die durch eine Verknüpfung von Modellen angrenzender Domänen bzw. relevanter Standards ebenfalls profitieren könnte, existieren ebenfalls hierzu nutzbare Vorarbeiten: Nicht nur spezifiziert die ISACA in [Mac07] ein COBIT-Mapping zu TOGAF, dem Rahmenwerk für Unternehmensarchitektur der Open Group, sondern es stellen auch Gerber et al. in [GKv10] eine Formalisierung des TOGAF-Content-Modells als Ontologie vor. Beide Arbeiten könnten zusammengenommen zu einer Erstellung einer formalen Abbildung zwischen den Ontologien für COBIT und TOGAF genutzt werden. Damit würden, in Ergänzung zum Modell der Abhängigkeiten zwischen den Abstraktionsebenen in COBIT, auch die in TOGAF im Vordergrund stehenden strukturellen Eigenschaften eines Unternehmens, wie die Aufteilung von Organisationseinheiten, in die Berücksichtigung der Automatisierung einfließen.

Auch eine Formalisierung der Zusammenhänge zwischen ITIL und COBIT ist denkbar, da eine Abbildung beider Rahmenwerke bereits von der ISACA [HH08b], aber teilweise auch von unabhängigen Analysten [gle14] behandelt wurde. Zusätzlich liegen bereits Arbeiten vor, in denen ITIL als Ontologie formalisiert wurde, beispielsweise von Valiente et al. [Val11, VVR11, VGS12]. Der Zusammenhang zwischen IT Service Management bzw. dem De-Facto-Standard ITIL und IT-Governance bzw. COBIT wurde in Abschnitt 2.1.3 bereits grundlegend betrachtet, eine formale Integration beider Modelle könnte aber insbesondere in Unternehmen, in denen bereits beide Rahmenwerke eingesetzt werden, große Vorteile bringen. Nicht nur könnten Begrifflichkeiten klarer abgegrenzt werden, sondern es ließen sich auch die gemeinsamen Schnittstellen, beispielweise im Portfolio-Management, dem Demand Management oder dem Service Level Management klar definieren und in gemeinsamen Automatisierungsregeln nutzen.

In der Literatur existieren demnach Arbeiten zur Formalisierung einzelner Aspekte eines Unternehmens durch Ontologien, es sind dem Autor jedoch keine Bestrebungen bekannt, diese koordiniert zu einem frei nutzbaren Meta-Rahmenwerk zusammenzufassen. Der erste Schritt hierzu sollte aus einer Steigerung der Wahrnehmung von Ontologien als geeignetem Mittel zur Formalisierung von geteilten, unter Umständen sogar standardisierten, Informationsmodellen durch die jeweiligen Normungsgremien sein. Die Entwicklung formaler Modelle, die die Inhalte der eigentlichen Rahmenwerke repräsentieren, sollte nicht mehr nur durch Dritte geschehen, sondern durch Expertengruppen, die innerhalb der jeweiligen offiziellen Entwicklungsprozesse agieren und nebem dem Normungsgremium auch Experten für die Wissensmodellierung umfassen. Dies gilt für technische Bereiche, für diejenigen Bereiche, die sich mit Aufbau und Führung von Unternehmen beschäftigen und für den Bereich von Normen und Vorschriften. Der zweite Schritt sollte schließlich die Schaffung eines Baukastens der jeweiligen formalen Modelle sein, der von unabhängiger Stelle gepflegt wird, die einzelnen Bestandteile aktuell hält und ihre Kompatibilität untereinander sicherstellt. Die in dieser Arbeit vorgestellten Ansätze zur Verknüpfung der einzelnen Domänen können als erster Schritt in diese Richtung verstanden werden.

Literaturverzeichnis

[ABE+06] Andersson, Birger ; Bergholtz, Maria ; Edirisuriya, Ananda ; Ilayperuma, Tharaka ; Johannesson, Paul ; Gordijn, Jaap ; Grégoire, Bertrand ; Schmitt, Michael ; Dubois, Eric ; Abels, Sven ; Hahn, Axel ; Wangler, Bengt ; Weigand, Hans: Towards a Reference Ontology for Business Models. In: Embley, David W. (Hrsg.) ; Olivé, Antoni (Hrsg.) ; Ram, Sudha (Hrsg.): *Conceptual Modeling — ER 2006* Bd. 4215, Springer, 2006 (Lecture Notes in Computer Science). – ISBN 978–3–540–47224–7, S. 482–496 Referenziert auf Seite 66.

[ACGP13] Adamou, Alessandro ; Ciancarini, Paolo ; Gangemi, Aldo ; Presutti, Valentina: The foundations of virtual ontology networks. In: *Proceedings of the 9th International Conference on Semantic Systems - I-SEMANTICS '13* (2013), S. 49–56. ISBN 9781450319720 Referenziert auf Seite 56.

[AdG+12a] Ampe, Floris ; du Preez, Gert ; Grijp, Stefanie ; Hardy, Gary ; Peeters, Bart ; Poels, Geert ; Steuperaert, Dirk: *COBIT 5: A Business Framework for the Governance and Management of Enterprise IT.* Information Systems Audit and Control Association (ISACA), 2012. – ISBN 978–1–60420–237–3 Referenziert auf Seiten 17, 69, 99, 101, 103 und 109.

[AdG+12b] Ampe, Floris ; du Preez, Gert ; Grijp, Stefanie ; Hardy, Gary ; Peeters, Bart ; Poels, Geert ; Steuperaert, Dirk: *COBIT 5: Rahmenwerk für Governance und Management der Unternehmens-IT.* Information Systems Audit and Control Association (ISACA), 2012. – ISBN 978–1–60420–245–8 Referenziert auf Seiten 17, 19, 21 und 102.

[AdG+12c] Ampe, Floris ; du Preez, Gert ; Grijp, Stefanie ; Hardy, Gary ; Peeters, Bart ; Steuperaert, Dirk: *COBIT® 5: Enabling Processes.* Information Systems Audit and Control Association (ISACA), 2012. – ISBN 978–1–60420–241–0 Referenziert auf Seiten 17 und 100.

[AH07] Auer, Sören ; Herre, Heinrich: RapidOWL — An Agile Knowledge Engineering Methodology. In: Virbitskaite, Irina (Hrsg.) ; Voronkov, Andrei (Hrsg.): *Perspectives of Systems Informatics* Bd. 4378, Springer, 2007 (Lecture Notes in Computer Science). – ISBN 978–3–540–70880–3, S. 424–430 Referenziert auf Seiten 52 und 53.

[apa10] Apache: *The Apache Velocity Project.* `http://velocity.apache.org/`, 2010. – Abgerufen am 16.07.2017. Referenziert auf Seite 129.

[apa11] Apache Commons: *Java Expression Language (JEXL).* `http://commons.apache.org/proper/commons-jexl/`, 2011. – Abgerufen am 16.07.2017. Referenziert auf Seite 127.

[apa14a] Apache: *Felix OSGi Framework and Service platform.* `http://felix.apache.org/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seite 127.

[apa14b] Apache: *Jena.* `https://jena.apache.org/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seiten 123 und 127.

[apa14c] Apache: *Jena Fuseki SPARQL Server.* `https://jena.apache.org/documentation/fuseki2/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seite 123.

[apa14d] Apache: *Maven Project.* `https://maven.apache.org/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seite 125.

[Ass08] Association for Ontology Design & Patterns (ODPA): *Ontology Design Patterns Portal.* `http://ontologydesignpatterns.org/`, 2008. – Abgerufen am 16.07.2017. Referenziert auf Seite 99.

[Baa07] Baader, Franz: Description Logic Terminology. In: Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation and Applications (Second Edition).* Cambridge University Press, 2007. – ISBN 978–0521150118, S. 525–536 Referenziert auf Seite 37.

[BBL05] Baader, Franz ; Brandt, Sebastian ; Lutz, Carsten: Pushing the $\mathcal{EL}$ Envelope. In: Kaelbling, Leslie P. (Hrsg.) ; Saffiotti, Alessandro (Hrsg.): *Proceedings on the 19th Joint International Conference on Artificial Intelligence (IJCAI 2005).* Edinburgh, Schottland : Morgan Kaufman Publishers, Juli 2005. – ISBN 0938075934, S. 364–369 Referenziert auf Seite 38.

[BCCG09] Baiôco, Gleison ; Costa, André Cypriano M. ; Calvi, Camilo Z. ; Garcia, Anilton S.: IT Service Management and Governance - Modeling an ITSM Configuration Process: a Foundational Ontology Approach. In: *Proceedings of the '09. IFIP/IEEE International Symposium on Integrated Network Management (IM) - Workshops*, IEEE, 2009. – ISBN 9781424439249, S. 24–33 Referenziert auf Seiten 69 und 71.

[Ber16] *Berlin Open Data.* `http://daten.berlin.de/`, 2016. – Abgerufen am 16.07.2017. Referenziert auf Seite 27.

[Bet07] Betz, Charles T.: *Architecture and Patterns for IT Service Management, Resource Planning and Governance: Making Shoes for the Cobbler's Children.* Morgan Kaufman Publishers, 2007. – ISBN 978–0–12–370593–8 Referenziert auf Seiten 12, 13 und 16.

[BFM05] Berners-Lee, Tim ; Fielding, Roy T. ; Masinter, Larry: *RFC 3986: Uniform Resource Identifier (URI): Generic Syntax.* `http://www.ietf.org/rfc/rfc3986.txt`, 2005 Referenziert auf Seite 27.

[BFO15] Institute for Formal Ontology and Medical Information Science (IFOMIS): *Basic Formal Ontology (BFO).* `http://www.ifomis.org/bfo/`, 2015 Referenziert auf Seite 48.

[BG16] Bergman, Michael K. ; Giasson, Frédérick: Upper Mapping and Binding Exchange Layer (UMBEL) ontology / Structured Dynamics LLC. 2016. – Forschungsbericht Referenziert auf Seite 49.

[BL13] Banister, Gary A. ; Lewis, Barry D.: *Process Assessment Model (PAM): Using COBIT® 5.* Information Systems Audit and Control Association (ISACA), 2013. – ISBN 978–1–60420–264–9 Referenziert auf Seiten 18, 105 und 112.

[BM04] Biron, Paul V. ; Malhotra, Ashok: *XML Schema Part 2: Datatypes Second Edition.* `https://www.w3.org/TR/xmlschema-2/`, 2004 Referenziert auf Seite 30.

[BM10] Birbeck, Mark ; McCarron, Shane: *CURIE Syntax 1.0 – A syntax for expressing Compact URIs.* `https://www.w3.org/TR/curie/`, 2010 Referenziert auf Seite 28.

[BMM+94a] Benjamin, Perakath C. ; Menzel, Christopher P. ; Mayer, Richard J. ; Fillion, Florence ; Futrell, Michael T. ; DeWitte, Paula S. ; Lingineni, Madhavi: IDEF5 Method Report - Information Integration for Concurrent Engineering (IICE). 1994 (409). – Forschungsbericht Referenziert auf Seite 52.

[BMM94b] Berners-Lee, Tim ; Masinter, Larry ; McCahill, Mark: *RFC 1739: Uniform Resource Locators (URL).* `http://www.ietf.org/rfc/rfc1738.txt`, 1994 Referenziert auf Seite 27.

[BN07] Baader, Franz ; Nutt, Werner: Basic Description Logics. In: Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation and Applications (Second Edition).* Cambridge University Press, 2007. – ISBN 978–0521150118, S. 47–104 Referenziert auf Seiten 36 und 37.

[BND14] *bnd and bndtools.* `http://bnd.bndtools.org/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seite 127.

[Bor97] Borst, Willem N.: *Construction of Engineering Ontologies for Knowledge Sharing and Reuse*, Universiteit Twente, Diss., 1997 Referenziert auf Seite 23.

[BPP10] Blumauer, Andreas ; Pellegrini, Tassilo ; Paschke, Adrian: Corporate Semantic Web – der Einsatz von Semantic-Web-Technologien im Unternehmen. In: *HMD Praxis der Wirtschaftsinformatik* 275 (2010), S. 105–114 Referenziert auf Seiten 68 und 69.

[BS04] Bittner, Thomas ; Smith, Barry: Normalizing Medical Ontologies using Basic Formal Ontology. In: *Kooperative Versorgung, Vernetzte Forschung, Ubiquitäre Information (Proceedings of GMDS Innsbruck)*, videel, Niebüll, 2004. – ISBN 3–8990–6750–9, S. 199–201 Referenziert auf Seite 48.

[bvq17] *Business Volume Qualicision (BVQ).* `https://www.bvq-software.de/en/home/`, 2017. – Abgerufen am 16.07.2017. Referenziert auf Seite 148.

[Cal06] *Ontologies for Software Engineering and Software Technology.* Springer, 2006. – ISBN 978–3–540–34517–6 Referenziert auf Seite 57.

[Can11] Cannon, David: *ITIL Service Strategy.* The Stationery Office Ltd, 2011. – ISBN 978–0113313044 Referenziert auf Seite 12.

[CFG03] Corcho, Oscar ; Fernández-López, Mariano ; Gómez-Pérez, Asunción: Methodologies, tools and languages for building ontologies. Where is their meeting point? In: *Data & Knowledge Engineering* 46 (2003), Nr. 1 Referenziert auf Seite 52.

[Cic10] Ciccarese, Paolo: *List Ontology Design Pattern.* `http://ontologydesignpatterns.org/wiki/Submissions:List`, 2010. – Abgerufen am 16.07.2017. Referenziert auf Seite 107.

[CO15] Czarnecki, Adam ; Orłowski, Cezary: Application of Ontology in the ITIL Domain. In: *Information Systems Architecture and Technology: Service Oriented Networked Systems.* Oficyna Wydawnicza Politechniki Wrocławskiej, 2015 Referenziert auf Seiten 67, 69 und 71.

[COB12] *COBIT 5 for Information Security.* Information Systems Audit and Control Association (ISACA), 2012. – ISBN 978–1–60420–255–7 Referenziert auf Seite 17.

[COB13a] *COBIT 5 for Assurance.* Information Systems Audit and Control Association (ISACA), 2013. – ISBN 978–1–60420–340–0 Referenziert auf Seite 17.

[COB13b] *COBIT 5 for Risk.* Information Systems Audit and Control Association (ISACA), 2013. – ISBN 978–1–60420–458–2 Referenziert auf Seite 17.

[Cra01] Cranefield, Stephen: UML and the Semantic Web. In: *The International Semantic Web Working Symposium.* Palo Alto, USA, 2001 Referenziert auf Seite 69.

[CS13] Czarnecki, Adam ; Sitek, Tomasz: Ontologies vs. Rules - Comparison of Methods of Knowledge Representation Based on the Example of IT Services Management. In: Borzemski, Leszek (Hrsg.) ; Grzech, Adam (Hrsg.) ; Świątek, Jerzy (Hrsg.) ; Wilimowska, Zofia (Hrsg.): *Information Systems Architecture and Technology: Intelligent Information Systems, Knowledge Discovery, Big Data and High Performance Computing*. Oficyna Wydawnicza Politechniki Wrocławskiej, 2013, S. 99–109 Referenziert auf Seiten 67 und 71.

[Cv08] Carroll, Ray ; van der Meer, Sven: Semantic Integration of User Data - Models and Processes Enabling Seamless Mobility. In: *32nd Annual IEEE International Computer Software and Applications Conference* (2008), S. 1103–1109. ISBN 978–0–7695–3262–2 Referenziert auf Seite 65.

[CX05] Cruz, Isabel F. ; Xiao, Huiyong: The Role of Ontologies in Data Integration. In: *Journal of Engineering Intelligent Systems* 13 (2005), Nr. 4 Referenziert auf Seite 57.

[cyc14] Cycorp: *OpenCyc*. `http://www.cyc.com/platform/opencyc`, 2014. – Abgerufen am 01.10.2014. Referenziert auf Seite 48.

[DBG+10] Daga, Enrico ; Blomqvist, Eva ; Gangemi, Aldo ; Montiel, Elena ; Nikitina, Nadejda ; Presutti, Valentina ; Villazón-Terrazas, Boris: Pattern based ontology design: methodology and software support (NeOn Deliverable D2.5.2). 2010. – Forschungsbericht Referenziert auf Seite 58.

[DCM15] Association for Information Science and Technology (ASIS&T): *Dublin Core Metadata Initiative*. `http://dublincore.org/`, 2015 Referenziert auf Seite 114.

[ddTF10] dos Santos, Marcos H. ; dos Santos, Joni H. ; Todesco, Jose L. ; Fileto, Renato: ITIL Ontology-based Model for IT Governance: A prototype demonstration. (2010) Referenziert auf Seiten 67 und 69.

[DFSA11] Duque-Ramos, Astrid ; Fernández-Breis, Jesualdo T. ; Stevens, Robert ; Aussenac-Gilles, Nathalie: OQuaRE: A SQuaRE-based Approach for Evaluating the Quality of Ontologies. In: *Journal of Research and Practice in Information Technology* 43 (2011), Nr. 2 Referenziert auf Seite 60.

[DGVB09] De Vergara, Jorge E. L. ; Guerrero, Antonio ; Villagrá, Víctor A. ; Berrocal, Julio: Ontology-Based Network Management: Study Cases and Lessons Learned. In: *Journal of Network and Systems Management* 17 (2009), September, Nr. 3, S. 234–254. – ISSN 1064–7570 Referenziert auf Seiten 64, 69 und 70.

[dHS12] du Preez, Gert ; Hardy, Gary ; Steuperaert, Dirk: *COBIT 5 Implementation*. Information Systems Audit and Control Association (ISACA), 2012. – ISBN 978–1–60420–240–3 Referenziert auf Seite 17.

[Dit07] Dittmann, Lars U.: *OntoFMEA - Ontologiebasierte Fehlermöglichkeits- und Einflussanalyse*. Springer, 2007. – ISBN 978–3–8350–9572–4 Referenziert auf Seite 175.

[DKSP10] Damova, Mariana ; Kiryakov, Atanas ; Simov, Kiril ; Petrov, Svetoslav: Mapping the Central LOD Ontologies to PROTON Upper-Level Ontology. In: Shvaiko, Pavel (Hrsg.) ; Euzenat, Jérôme (Hrsg.) ; Giunchiglia, Fausto (Hrsg.) ; Stuckenschmidt, Heiner (Hrsg.) ; Mao, Ming (Hrsg.) ; Cruz, Isabel F. (Hrsg.): *Proceedings of The Fifth International Workshop on Ontology Matching (OM-2010)*. Shanghai, China, 2010. – ISSN 1613–0073 Referenziert auf Seite 49.

[DMN09] De Nicola, Antonio ; Missikoff, Michele ; Navigli, Roberto: A software engineering approach to ontology building. In: *Information Systems* 34 (2009), April, Nr. 2, S. 258–275. – ISSN 03064379 Referenziert auf Seiten 52 und 97.

[dmt17] Distributed Management Taskforce (DMTF): *Virtualization Management (VMAN)*. `http://www.dmtf.org/standards/vman`, 2017 Referenziert auf Seite 151.

[DOL06] Laboratory for Applied Ontology (ISTC-CNR): *Descriptive Ontology for Linguistic and Cognitive Engineering (DOLCE)*. `http://www.loa.istc.cnr.it/old/DOLCE.html`, 2006 Referenziert auf Seite 49.

[dRB+14] de Koning, Hans P. ; Rouquette, Nicolas ; Burkhart, Roger ; Espinoza, Huascar ; Lefort, Laurent: *Library for Quantities, Units, Dimensions and Values (QUDV)*. `http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu`, 2014. – Abgerufen am 24.04.2014. Referenziert auf Seite 117.

[DRS+06] Drummond, Nick ; Rector, Alan ; Stevens, Robert ; Moulton, Georgina ; Horridge, Matthew ; Wang, Hai H. ; Seidenberg, Julian: Putting OWL in Order: Patterns for Sequences in OWL. In: Grau, Bernardo C. (Hrsg.) ; Hitzler, Pascal (Hrsg.) ; Shankey, Conor (Hrsg.) ; Wallace, Evan (Hrsg.): *Proceedings of the OWLED*06 Workshop on OWL: Experiences and Directions* Bd. 216. Athens, Georgia, USA, November 2006. – ISSN 1613–0073 Referenziert auf Seite 107.

[DS05] Dürst, Martin ; Suignard, Michel: *RFC 3987: Internationalized Resource Identifiers (IRI)*. `http://www.ietf.org/rfc/rfc3987.txt`, 2005 Referenziert auf Seite 28.

[dSSS09] d'Aquin, Mathieu ; Schlicht, Anne ; Stuckenschmidt, Heiner ; Sabou, Marta: Criteria and Evaluation for Ontology Modularization Techniques. In: Stuckenschmidt, Heiner (Hrsg.) ; Parent, Christine (Hrsg.) ; Spaccapietra, Stefano (Hrsg.): *Modular Ontologies* Bd. 5445. Springer, 2009. – ISBN 978–3–642–01906–7, S. 67–89 Referenziert auf Seite 55.

[DT08] Dugmore, Jenny ; Taylor, Sharon: *ITIL® V3 and ISO/IEC 20000 - Alignment White Paper*. Office of Government Commerce (OGC), 2008 Referenziert auf Seite 14.

[DuC11] DuCharme, Bob: *Learning SPARQL*. O'Reilly, 2011. – ISBN 978–1449306595 Referenziert auf Seite 45.

[DV04] De Haes, Steven ; Van Grembergen, Wim: IT Governance and its mechanisms. In: *Information Systems Control Journal* 1 (2004) Referenziert auf Seite 15.

[DVB04] De Vergara, Jorge E. L. ; Villagrá, Víctor A. ; Berrocal, Julio: Applying the Web ontology language to management information definitions. In: *IEEE Communications Magazine* 42 (2004), Juli, Nr. 7, S. 68–74. – ISSN 0163–6804 Referenziert auf Seiten 64 und 70.

[edm17] *Enterprise Data Management Council (EDMC)*. `https://www.edmcouncil.org/`, 2017. – Abgerufen am 16.07.2017. Referenziert auf Seite 151.

[ES13] Euzenat, Jérôme ; Shvaiko, Pavel: *Ontology Matching - Second edition*. Springer, 2013. – ISBN 978–3–642–38720–3 Referenziert auf Seiten 55 und 56.

[Fag14] Fagnoni, Enrico: *ITSMO IT Service Management Ontology*. `http://ontology.it/itsmo/v1/itsmo.html`, 2014. – Abgerufen am 24.04.2014. Referenziert auf Seite 67.

[FE07] Fang, Jennifer ; Evermann, Joerg: Evaluating Ontologies: Towards a Cognitive Measure of Quality. In: *Proceedings of the 2nd International Workshop on Vocabularies, Ontologies and Rules for the Enterprise (VORTE), co-located with The Eleventh International IEEE Enterprise Distributed Object Computing Conference (EDOC) Conference*, 2007. – ISBN 9780769533384, S. 109–116 Referenziert auf Seite 60.

[Fel06] Fellbaum, Christiane: WordNet(s). In: *Encyclopedia of Language & Linguistics (Second Edition)* (2006), S. 665–670. ISBN 978–0–08–044854–1 Referenziert auf Seite 25.

[Fer99] Fernández-López, Mariano: Overview Of Methodologies For Building Ontologies. In: *IJCAI99 Workshop on Ontologies and Problem-Solving Methods: Lessons Learned and Future Trends*. Stockholm, Schweden, 1999 Referenziert auf Seiten 46 und 52.

[FGJ97] Fernández-López, Mariano ; Gómez-Pérez, Asunción ; Juristo, Natalia: METHONDOLOGY: From Ontological Art Towards Ontological Engineering. In: *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*. Stanford, USA, März 1997 Referenziert auf Seiten 52 und 96.

[FGPP99] Fernández-López, Mariano ; Gómez-Pérez, Asunción ; Pazoz Sierra, Juan ; Pazoz Sierra, Alejandro: Building a Chemical Ontology Using Methontology and the Ontology Design Environment. In: *Intelligent Systems and their Appliations* 14 (1999), Nr. 1, S. 37–46. – ISSN 1094–7167 Referenziert auf Seiten 52 und 96.

[FO12] Fallon, Liam ; O'Sullivan, Declan: Using a semantic knowledge base for communication service quality management in Home Area Networks. In: *2012 IEEE Network Operations and Management Symposium*. Maui, Hawaii, USA : IEEE, April 2012. – ISBN 978–1–4673–0267–8, S. 43–51 Referenziert auf Seite 65.

[FO14] Fallon, Liam ; O'Sullivan, Declan: The Aesop Approach for Semantic-Based End-User Service Optimization. In: *IEEE Transactions on Network and Service Management* 11 (2014), Juni, Nr. 2, S. 220–234. – ISSN 1932–4537 Referenziert auf Seite 65.

[Fox10] Foxvog, Douglas: Cyc. In: Poli, Roberto (Hrsg.) ; Healy, Michael (Hrsg.) ; Kameas, Achilles (Hrsg.): *Theory and Applications of Ontology: Computer Applications*. Springer, 2010. – ISBN 978–90–481–8846–8, S. 259–278 Referenziert auf Seite 48.

[GA09] Goeken, Matthias ; Alter, Stefanie: Towards Conceptual Metamodeling of IT Governance Frameworks Approach - Use - Benefits. In: *Proceedings of the 42nd Hawaii International Conference on System Sciences*, 2009. – ISBN 9780769534503 Referenziert auf Seiten 65 und 69.

[Gan02] Gandon, Fabien: *Distributed Artificial Intelligence and Knowledge Management: Ontologies and Multi-Agent Systems for Corporate Semantic Web*, INRIA and University of Nice - Sophia Antipolis — Doctoral School of Sciences and Technologies of Information and Communication (S.T.I.C.), Diss., 2002 Referenziert auf Seite 51.

[Gan09] Gangemi, Aldo: *Ontology:DOLCE+DnS Ultralite*. `http://ontologydesignpatterns.org/wiki/Ontology:DOLCE+DnS_Ultralite`, 2009. – Abgerufen am 16.07.2017. Referenziert auf Seite 49.

[Gan10] Gangemi, Aldo: *Sequence Ontology Design Pattern*. `http://ontologydesignpatterns.org/wiki/Submissions:Sequence`, 2010. – Abgerufen am 16.07.2017. Referenziert auf Seite 107.

[Gar05] García, Roberto: *A Semantic Web approach to Digital Rights Management*, Universitat Pompeu Fabra, Diss., 2005 Referenziert auf Seite 150.

[GCCL06] Gangemi, Aldo ; Catenacci, Carola ; Ciaramita, Massimiliano ; Lehmann, Jos: Modelling Ontology Evaluation and Validation. In: Sure, York (Hrsg.) ; Domingue, John (Hrsg.): *The Semantic Web: Research and Applications* Bd. 4011, Springer,

2006 (Lecture Notes in Computer Science). – ISBN 978–3–540–34544–2, S. 140–154 Referenziert auf Seite 60.

[GDH+03] Guldentops, Erik ; De Haes, Steven ; Hardy, Gary ; Ormsby, Jacqueline ; Ramos, Daniel F. ; Singleton, Jon ; Williams, Paul A.: *Board Briefing on IT Governance, 2nd Edition*. IT Governance Institute (ITGI), 2003. – ISBN 1–893209–64–4 Referenziert auf Seite 14.

[GF95] Grüninger, Michael ; Fox, Mark S.: Methodology for the Design and Evaluation of Ontologies. In: *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995 Referenziert auf Seiten 52 und 97.

[GFB+92] Genesereth, Michael ; Fikes, Richard ; Bobrow, Daniel ; Brachman, Ronald J. ; Gruber, Thomas ; Hayes, Patrick ; Letsinger, Reed ; Lifschitz, Vladimir ; MacGregor, Robert ; McCarthy, John ; Norvig, Peter ; Patil, Ramesh ; Schubert, Len: *Knowledge Interchange Format Version 3.0 Reference Manual*. 1992 Referenziert auf Seite 25.

[GFO10] Onto-Med Research Group: *General Formal Ontology*. `http://www.onto-med.de/ontologies/gfo/`, 2010. – Abgerufen am 16.07.2017. Referenziert auf Seite 49.

[GFV96] Gómez-Pérez, Asunción ; Fernández-López, Mariano ; Vicente Rodríguez, Antonio J.: Towards a Method to Conceptualize Domain Ontologies. In: Wahlster, Wolfgang (Hrsg.): *ECAI96 Workshop on Ontological Engineering*. Budapest, Ungarn, 1996. – ISBN 0471968099 Referenziert auf Seiten 52 und 96.

[GG08] Ghedini Ralha, Célia ; Gostinski, Rafael: A Methodological Framework for Business-IT Alignment. In: Bartolini, Claudio (Hrsg.) ; Sahai, Akhil (Hrsg.) ; Sauvé, Jacques P. (Hrsg.): *Proceedings of the 3rd IEEE/IFIP International Workshop on Business-driven IT Management*. Salvador, Brasilien : IEEE, April 2008. – ISBN 978–1–4244–2191–6, S. 1–10 Referenziert auf Seite 66.

[GGM+02] Gangemi, Aldo ; Guarino, Nicola ; Masolo, Claudio ; Oltramari, Alessandro ; Schneider, Luc: Sweetening Ontologies with DOLCE. In: Richard Benjamins, V. (Hrsg.): *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web* Bd. 2473. Sigüenza, Spanien : Springer, Oktober 2002 (Lecture Notes in Computer Science). – ISBN 978–3–540–44268–4, S. 166–181 Referenziert auf Seite 49.

[GH07] Grimm, Stephan ; Hitzler, Pascal: Semantic Matchmaking of Web Resources with Local Closed-World Reasoning. In: *International Journal of Electronic Commerce* 12 (2007), Nr. 2, S. 89–126. – ISSN 1086–4415 Referenziert auf Seite 39.

[GKv10] Gerber, Aurona ; Kotzé, Paula ; van der Merwe, Alta: Towards the Formalisation of the TOGAF Content Metamodel using Ontologies. In: Filipe, Joaquim (Hrsg.) ; Cordeiro, José (Hrsg.): *Proceedings of the 12th International Conference on Enterprise Information Systems*. Funchal, Portugal : SciTe Press, 2010. – ISBN ISBN 978–989–8425–05–8, S. 54–64 Referenziert auf Seite 176.

[gle14] glenfis AG: *ITIL© Edition 2011 - COBIT® 5 Mapping*. `http://www.glenfis.ch/files/3513/9956/1349/ITIL_Edition_2011_-_COBIT_5_-Mapping_Glenfis_AG_v1.2.pdf`, 2014. – Abgerufen am 20.10.2014. Referenziert auf Seite 176.

[GMF11] González-Conejero, Jorge ; Meroño-Peñuela, Albert ; Fernández, David: Ontologies for Governance, Risk Management and Policy Compliance. In: *Workshop on Modelling Policy-making (MPM 2011) in conjunction with The 24th International Conference on Legal Knowledge and Information Systems (JURIX 2011)*, 2011. – ISBN 4300002010, S. 3–7 Referenziert auf Seiten 69 und 71.

[GOG+10] Garcia, Alexander ; O'Neill, Kieran ; Garcia, Leyla J. ; Lord, Phillip ; Stevens, Robert ; Corcho, Oscar ; Gibson, Frank: Developing ontologies within decentralized settings. In: Chen, Huajun (Hrsg.) ; Wang, Yimin (Hrsg.) ; Cheung, Kei-Hoi (Hrsg.): *Semantic e-Science* Bd. 11, Springer, 2010 (Annals of Information Systems). – ISBN 978–1–4419–5902–7, S. 99–139 Referenziert auf Seiten 47, 48, 52 und 96.

[Gom04] *Ontological Engineering*. Springer, 2004. – ISBN 978–1–85233–551–9 Referenziert auf Seiten 46 und 57.

[GP09] Gangemi, Aldo ; Presutti, Valentina: Ontology Design Patterns. In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 201–220 Referenziert auf Seite 58.

[GPS13] Grijp, Stefanie ; Peeters, Bart ; Steuperaert, Dirk: *COBIT® 5: Enabling Information*. Information Systems Audit and Control Association (ISACA), 2013. – ISBN 978–1–60420–350–9 Referenziert auf Seite 17.

[Gru93] Gruber, Thomas: A Translation Approach to Portable Ontology Specification. In: *Knowledge Acquisition* 5 (1993), Juni, Nr. 2. – ISSN 1042–8143 Referenziert auf Seite 23.

[Gru03] Gruber, Thomas: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. In: Guarino, Nicola (Hrsg.) ; Poli, Roberto (Hrsg.): *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Kluwer Academic Publishers, 2003 Referenziert auf Seiten 57 und 97.

[GSG04] Grenon, Pierre ; Smith, Barry ; Goldberg, Louis: Biodynamic Ontology: Applying BFO in the Biomedical Domain. In: *Ontologies in Medicine* (2004), S. 20–38. ISBN 978–1–58603–418–4 Referenziert auf Seite 48.

[GTT+06] Good, Benjamin M. ; Tranfield, Erin M. ; Tan, Poh C. ; Shehata, Marlene ; Singhera, Gurpreet K. ; Gosselink, John ; Okon, Elena B. ; Wilkinson, Mark D.: Fast, Cheap and Out of Control: A Zero Curation Model for Ontology Development. In: *Pacific Symposium on Biocomputing 11*. Maui, Hawaii, USA, 2006, S. 128–139 Referenziert auf Seite 52.

[Gui05] Guizzardi, Giancarlo: *Ontological Foundations for Structural Conceptual Models*, Universiteit Twente, Diss., 2005 Referenziert auf Seite 50.

[GVD03] Grosof, Benjamin ; Volz, Raphael ; Decker, Stefan: Description Logic Programs: Combining Logic Programs with Description Logic. In: *Proceedings of the 12th International Conference on World Wide Web*. Budapest, Ungarn : ACM Press, 2003. – ISBN 1–58113–680–3, S. 48–57 Referenziert auf Seite 39.

[GVD+06] Guerrero, Antonio ; Villagrá, Víctor A. ; De Vergara, Jorge E. L. ; Sánchez-Macián, Alfonso ; Berrocal, Julio: Ontology-Based Policy Refinement Using SWRL Rules for Management Information Definitions in OWL. In: State, Radu (Hrsg.) ; van der Meer, Sven (Hrsg.) ; O'Sullivan, Declan (Hrsg.) ; Pfeifer, Tom (Hrsg.): *Large Scale Management of Distributed Systems - 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2006* Bd. 4269. Irland : Springer, Oktober 2006 (Lecture Notes in Computer Science). – ISBN 978–3–540–47659–7, S. 227–232 Referenziert auf Seiten 64 und 70.

[GVDB05] Guerrero, Antonio ; Villagrá, Víctor A. ; De Vergara, Jorge E. L. ; Berrocal, Julio: Ontology-Based Integration of Management Behaviour and Information Definitions Using SWRL and OWL. In: Schönwälder, Jürgen (Hrsg.) ; Serrat, Joan (Hrsg.): *Ambient Networks* Bd. 3775, Springer, 2005 (Lecture Notes in Computer Science). – ISBN 978–3–540–29388–0, S. 12–23 Referenziert auf Seiten 64 und 70.

[GW00a] Guarino, Nicola ; Welty, Christopher A.: A Formal Ontology of Properties. In: Dieng, Rose (Hrsg.) ; Corby, Olivier (Hrsg.): *Knowledge Engineering and Knowledge Management. Methods, Models, and Tools. 12th International Conference, EKAW 2000, Juan-les-Pins, France, October 2-6, 2000 Proceedings* Bd. 1937. Frankreich : Springer, Oktober 2000 (Lecture Notes in Computer Science). – ISBN 978–3–540–41119–2, S. 97–112 Referenziert auf Seite 58.

[GW00b] Guarino, Nicola ; Welty, Christopher A.: Identity, Unity and Individuality: Towards a formal toolkit for ontological analysis. In: Horn, W. (Hrsg.): *Procee-*

dings of the 14th European Conference on Artificial Intelligence (ECAI 2000). Berlin : IOS Press, 2000, S. 219–223 Referenziert auf Seite 58.

[GW00c] Guarino, Nicola ; Welty, Christopher A.: Ontological Analysis of Taxonomic Relationships. In: Laender, Alberto Henrique F. (Hrsg.) ; Liddle, Stephen W. (Hrsg.) ; Storey, Veda C. (Hrsg.): *Conceptual Modeling — ER 2000* Bd. 1920, Springer, 2000 (Lecture Notes in Computer Science). – ISBN 978–3–540–41072–0, S. 210–224 Referenziert auf Seiten 49 und 58.

[GW04] Guizzardi, Giancarlo ; Wagner, Gerd: A Unified Foundational Ontology and some Applications of it in Business Modeling. In: Missikoff, Michele (Hrsg.): *Proceedings of the Open InterOp Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI - INTEROP 2004), Co-located with CAiSE'04 Conference*. Riga, Lettland, 2004, S. 129–143 Referenziert auf Seite 50.

[GW09] Guarino, Nicola ; Welty, Christopher A.: An Overview of OntoClean. In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 201–220 Referenziert auf Seite 58.

[Gó98] Gómez-Pérez, Asunción: Knowledge Sharing and Reuse. In: Liebowitz, Jay (Hrsg.): *Handbook of Applied Expert Systems*. New York, USA : CRC Press, 1998. – ISBN 0849331064, Kapitel 10 Referenziert auf Seiten 52 und 96.

[HAN99] Hegering, Heinz-Gerd ; Abeck, Sebastian ; Neumair, Bernhard: *Integriertes Management vernetzter Systeme*. Dpunkt Verlag, 1999. – ISBN 978–3932588167 Referenziert auf Seite 11.

[Her13] Herre, Heinrich: Formal Ontology and the Foundation of Knowledge Organization. In: *Knowledge Organization* 40 (2013), Nr. 5, S. 332–339 Referenziert auf Seite 49.

[Hes06] Heschl, Jimmy: *COBIT Mapping - Overview of International IT Guidance*. IT Governance Institute (ITG), 2006. – ISBN 1–933284–31–5 Referenziert auf Seite 18.

[HH08a] Hardy, Gary ; Heschl, Jimmy: *Aligning COBIT© 4.1, ITIL© V3 and ISO/IEC 27002 for Business Benefit*. IT Governance Institute (ITG), 2008 Referenziert auf Seite 18.

[HH08b] Heschl, Jimmy ; Hardy, Gary: *COBIT© Mapping: Mapping of ITIL v3 with COBIT© 4.1*. IT Governance Institute (ITG), 2008. – ISBN 978–1–60420–035–5 Referenziert auf Seiten 18 und 176.

[HHB+07] Herre, Heinrich ; Heller, Barbara ; Burek, Patryk ; Hoehndorf, Robert ; Loebe, Frank ; Michalek, Hannes: General Formal Ontology (GFO) - A Foundational Ontology Integrating Objects and Processes. 2007. – Forschungsbericht Referenziert auf Seite 49.

[HKHS14] Hodgson, Ralph ; Keller, Paul J. ; Hodges, Jack ; Spivak, Jack: *QUDT - Quantities, Units, Dimensions and Data Types Ontologies*. http://www.qudt.org/, 2014. – Abgerufen am 24.04.2014. Referenziert auf Seite 116.

[HKP+12] Hitzler, Pascal ; Krötzsch, Markus ; Parsia, Bijan ; Patel-Schneider, Peter F. ; Rudolph, Sebastian: *OWL 2 Web Ontology Language Primer (Second Edition)*. http://www.w3.org/TR/owl2-primer/, 2012 Referenziert auf Seiten 35, 38 und 40.

[HKR08] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian: *Foundations of Semantic Web Technologies*. Berlin : Springer, 2008. – ISBN 978–1–4200–9050–5 Referenziert auf Seiten 36 und 42.

[HKRS08] Hitzler, Pascal ; Krötzsch, Markus ; Rudolph, Sebastian ; Sure, York: *Semantic Web - Grundlagen*. Berlin : Springer, 2008. – ISBN 978–3–540–33993–9 Referenziert auf Seiten 36 und 37.

[HKS06] Horrocks, Ian ; Kutz, Oliver ; Sattler, Ulrike: The Even More Irresistible $\mathcal{SROIQ}$. In: *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning*, 2006, S. 57–67 Referenziert auf Seite 38.

[Hoe09] Hoekstra, Rinke: *Ontology Representation - Design Patterns and Ontologies that Make Sense*, University of Amsterdam, Diss., 2009 Referenziert auf Seiten 38, 46, 48 und 107.

[Hor11] Horridge, Matthew: A Practical Guide To Building OWL Ontologies Using Protégé 4 and CO-ODE Tools. (2011) Referenziert auf Seite 100.

[HPB+04] Horrocks, Ian ; Patel-Schneider, Peter F. ; Boley, Harold ; Tabet, Said ; Grosof, Benjamin ; Dean, Mike: *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. http://www.w3.org/Submission/SWRL/, 2004 Referenziert auf Seite 64.

[HPMW07] Horrocks, Ian ; Patel-Schneider, Peter F. ; McGuinness, Deborah L. ; Welty, Christopher A.: OWL: a Description-Logic-Based Ontology Language for the Semantic Web. In: Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation and Applications (Second*

Edition). Cambridge University Press, 2007. – ISBN 978–0521150118, S. 458–486 Referenziert auf Seiten 25, 35 und 42.

[HR07] Hepp, Martin ; Roman, Dumitru: An Ontology Framework for Semantic Business Process Management. In: Oberweis, Andreas (Hrsg.) ; Weinhardt, Christof (Hrsg.) ; Gimpel, Henner (Hrsg.) ; Koschmider, Agnes (Hrsg.) ; Pankratius, Victor (Hrsg.) ; Schnizler, Björn (Hrsg.): *Proceedings of the 8th International Conference Wirtschaftsinformatik*, Universitätsverlag Karlsruhe, 2007, S. 423–440 Referenziert auf Seiten 67 und 69.

[HRG11] Hert, Matthias ; Reif, Gerald ; Gall, Harald C.: A Comparison of RDB-to-RDF Mapping Languages Categories and Subject Descriptors. In: *Proceedings of the 7th International Conference on Semantic Systems (I-SEMANTICS).* Graz, Österreich, September 2011. – ISBN 9781450306218, S. 25–32 Referenziert auf Seite 57.

[HT14] Hartig, Olaf ; Thompson, Bryan: *Foundations of an Alternative Approach to Reification in RDF.* `http://arxiv.org/pdf/1406.3399.pdf`, Juni 2014 Referenziert auf Seite 35.

[Hun11] Hunnebeck, Lou: *ITIL Service Design.* The Stationery Office Ltd, 2011. – ISBN 978–0113313051 Referenziert auf Seite 12.

[HZB04] Hochstein, Axel ; Zarnekow, Rüdiger ; Brenner, Walter: ITIL als Common-Practice-Referenzmodell für das IT-Service-Management - Formale Beurteilung und Implikationen für die Praxis. In: *Wirtschaftsinformatik* 46 (2004), Nr. 5, S. 382–389. – ISSN 0937–6429, 1861–8936 Referenziert auf Seite 12.

[HZB05] Hochstein, Axel ; Zarnekow, Rüdiger ; Brenner, Walter: ITIL as Common Practice Reference Model for IT Service Management: Formal Assessment and Implications for Practice. In: *IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05)*, IEEE Computer Society, 2005, S. 407–410 Referenziert auf Seite 12.

[IBM06] IBM Corporation: *An Architectural Blueprint for Autonomic Computing, Technical Whitepaper (Fourth Edition).* Juni 2006 Referenziert auf Seite 1.

[IBM15] IBM Corporation: *IBM System Storage SAN Volume Controller.* `http://www-03.ibm.com/systems/storage/software/virtualization/svc/overview.html`, 2015. – Abgerufen am 16.07.2017. Referenziert auf Seiten 147 und 154.

[IEE06] IEEE: *IEEE Standard for Developing a Software Project Life Cycle Process.* `http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1665059&contentType=Standards`, 2006 Referenziert auf Seite 46.

[isa14] Information Systems Audit and Control Association (ISACA): *About ISACA*. http://www.isaca.org/about-isaca/Pages/default.aspx, 2014. – Abgerufen am 24.04.2014. Referenziert auf Seite 17.

[ISO98] International Organization for Standardization: *ISO/IEC 10040:1998: Information technology – Open Systems Interconnection – Systems management overview*. http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?ics1=35&ics2=100&ics3=70&csnumber=24406, 1998 Referenziert auf Seite 11.

[ISO07] International Organization for Standardization: *ISO/IEC 24707:2007: Information technology – Common Logic (CL): a framework for a family of logic-based languages*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39175, 2007 Referenziert auf Seite 25.

[ISO11] International Organization for Standardization: *ISO/IEC 20000-1:2011: Information Technology – Service Management – Part 1: Service management system requirements*. http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=51986, 2011. – Abgerufen am 24.04.2014. Referenziert auf Seite 13.

[ISO12a] International Organization for Standardization: *ISO/IEC 20000-2:2012: Information Technology – Service Management – Part 2: Guidance on the application of service management systems*. http://www.iso.org/iso/catalogue_detail?csnumber=51987, 2012. – Abgerufen am 24.04.2014. Referenziert auf Seite 13.

[ISO12b] International Organization for Standardization: *ISO/IEC TR 20002:2012: Information technology – Telecommunications and information exchange between systems – Managed P2P: Framework*. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=50950, 2012 Referenziert auf Seite 14.

[ISO14] International Organization for Standardization: *ISO/IEC 25000:2014: Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuaRE*. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=64764, 2014. – Abgerufen am 01.07.2014. Referenziert auf Seite 60.

[ISO15a] International Organization for Standardization: *ISO/IEC 13250-5:2015: Information technology – Topic Maps – Part 5: Reference model*. http://www.iso.org/iso/catalogue_detail.htm?csnumber=40757, 2015 Referenziert auf Seite 25.

[ISO15b] International Organization for Standardization: *ISO/IEC 33001:2015: Information technology – Process assessment – Concepts and terminology*.

http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=54175, 2015 Referenziert auf Seite 21.

[ISO15c] International Organization for Standardization: *ISO/IEC 38500:2015: Information technology – Governance of IT for the organization.* http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=62816, 2015 Referenziert auf Seite 15.

[Jac99] Jackson, Peter: *Introduction to Expert Systems.* Addison Wesley, 1999. – ISBN 978–0201876864 Referenziert auf Seite 51.

[jet16] Eclipse Foundation: *Jetty Web Server.* http://www.eclipse.org/jetty/, 2016 Referenziert auf Seite 136.

[JG07] Johannsen, Wolfgang ; Goeken, Matthias: *Referenzmodelle für IT-Governance. Strategische Effektivität und Effizienz mit COBIT, ITIL & Co.* Heidelberg : Dpunkt Verlag, 2007. – ISBN 3898643972 Referenziert auf Seiten 15 und 16.

[KAL14] Knolmayer, Gerhard ; Asprion, Petra M. ; Loosli, Gabriela: IT-Governance. In: Gronau, Norbert (Hrsg.) ; Becker, Jörg (Hrsg.) ; Kurbel, Karl (Hrsg.) ; Sinz, Elmar (Hrsg.) ; Suhl, Leena (Hrsg.): *Enzyklopädie der Wirtschaftsinformatik - Online-Lexikon*, 2014. – Abgerufen am 29.09.2014. Referenziert auf Seite 14.

[KH04] Kriens, Peter ; Hargrave, BJ: Listeners Considered Harmful: The "Whiteboard" Pattern. 2004. – Forschungsbericht Referenziert auf Seite 131.

[KHI11] Knublauch, Holger ; Hendler, James A. ; Idehen, Kingsley: *SPIN - Overview and Motivation.* https://www.w3.org/Submission/spin-overview/, 2011 Referenziert auf Seiten 81 und 92.

[Kif05] Kifer, Michael: Rules and Ontologies in F-Logic. In: Eisinger, Norbert (Hrsg.) ; Małuszyński, Jan (Hrsg.): *Reasoning Web - First International Summer School* Bd. 3564. Msida, Malta : Springer, Juli 2005 (Lecture Notes in Computer Science). – ISBN 978–3–540–27828–3, S. 22–34 Referenziert auf Seite 24.

[KLW11] Kifer, Michael ; Lausen, Georg ; Wu, James: Logical Foundations of Object-Oriented and Frame-Based Languages. In: *International Journal of Service Science, Management, Engineering, and Technology* 2 (2011), April, Nr. 2 Referenziert auf Seite 24.

[KMV00] Kietz, Jörg-Uwe ; Maedche, Alexander ; Volz, Raphael: A Method for Semi-Automatic Ontology Acquisition from a Corporate Intranet. In: *Workshop on Ontologies and Text, Co-located with EKAW 2000.* Frankreich, 2000 Referenziert auf Seite 67.

[Koc11] Kocian, Claudia: Geschäftsprozessmodellierung mit BPMN 2.0: Business Process Model and Notation im Methodenvergleich. (2011), Juli Referenziert auf Seite 67.

[KS13] Knittl, Silvia ; Schmitz, David: An Ontology-Based Approach for Semantic Interoperability in Interorganizational IT Service Management. In: *Proceedings of the 8th IFIP/IEEE International Workshop on Business-driven IT Management*. Gent, Belgien, 2013. – ISBN 9783901882500, S. 1218–1224 Referenziert auf Seite 67.

[KSH13] Krötzsch, Markus ; Simančík, František ; Horrocks, Ian: *A Description Logic Primer*. http://arxiv.org/pdf/1201.4089, Juni 2013 Referenziert auf Seite 42.

[Len95] Lenat, Douglas: Cyc: A Large-Scale Investment in Knowledge Infrastructure. In: *Communications of the ACM* 11 (1995), Nr. 38. – ISSN 0001–0782 Referenziert auf Seite 48.

[Len13] Lenzerini, Maurizio: *Ontology-based Data Management*. http://wp.sigmod.org/?p=871, Mai 2013. – Abgerufen am 24.04.2014. Referenziert auf Seite 57.

[Llo11] Lloyd, Vernon: *ITIL Continual Service Improvement*. The Stationery Office Ltd, 2011. – ISBN 978–0113313082 Referenziert auf Seite 12.

[LvD+10] Latré, Steven ; van der Meer, Sven ; De Turck, Filip ; Strassner, John ; Hong, James W.: Ontological Generation of Filter Rules for Context Exchange in Autonomic Multimedia Networks. In: *Network Operations and Management Symposium (NOMS), 2010 IEEE*, 2010, S. 575–582 Referenziert auf Seite 65.

[Mac07] Macgregor, Stuart: *TOGAF™ and COBIT® - Mapping of TOGAF 8.1 with COBIT 4.0*. IT Governance Institute (ITG), 2007. – ISBN 978–1–933284–82–X Referenziert auf Seiten 18 und 176.

[MBH+04] Martin, David ; Burstein, Mark ; Hobbs, Jerry ; Lassila, Ora ; McDermott, Drew ; McIlraith, Sheila ; Narayanan, Srini ; Paolucci, Massimo ; Parsia, Bijan ; Payne, Terry ; Sirin, Evren ; Srinivasan, Naveen ; Sycara, Katia: *OWL-S: Semantic Markup for Web Services*. http://www.w3.org/Submission/OWL-S/, 2004 Referenziert auf Seite 107.

[MBK+12] Mertens, Peter ; Bodendorf, Freimut ; König, Wolfgang ; Picot, Arnold ; Schumann, Matthias ; Hess, Thomas: *Grundzüge der Wirtschaftsinformatik*. Springer, 2012. – ISBN 978–3–642–30514–6 Referenziert auf Seite 11.

[MCR07] Mascardi, Viviana ; Cordì, Valentina ; Rosso, Paolo: A Comparison of Upper Ontologies. In: Baldoni, Matteo (Hrsg.) ; Boccalatte, Antonio (Hrsg.) ; De Paoli, Flavio (Hrsg.) ; Martelli, Maurizio (Hrsg.) ; Mascardi, Viviana (Hrsg.): *Proceedings of the 8th AI*IA/TABOO Joint Workshop From Objects to Agents: Agents*

and Industry: Technological Applications of Software Agents (WOA 2007). Genua, Italien, 2007. – ISBN ISBN 978–88–6122–061–4, S. 55–64 Referenziert auf Seiten 48 und 49.

[MGH+12] Motik, Boris ; Grau, Bernardo C. ; Horrocks, Ian ; Wu, Zhe ; Fokoue, Achille ; Lutz, Carsten: *OWL 2 Web Ontology Language Profiles (Second Edition)*. `http://www.w3.org/TR/owl2-profiles/`, 2012 Referenziert auf Seiten 38 und 136.

[MH09] Möller, Ralf ; Haarslev, Volker: Tableau-Based Reasoning. In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 509–528 Referenziert auf Seite 124.

[MK15] Meyer, Fabian ; Kroeger, Reinhold: A Framework for Autonomic, Ontology-based IT Management. In: Tortonesi, Mauro (Hrsg.) ; Schönwälder, Jürgen (Hrsg.): *Proceedings of the 11th International Conference on Network and Service Management*. Barcelona, Spanien, 2015. – ISBN 978–3–901882–77–7, S. 78–84 Referenziert auf Seite 80.

[MMH11] Maier, Frederick ; Mutharaju, Raghava ; Hitzler, Pascal: Distributed Reasoning with $\mathcal{EL}^{++}$ using MapReduce / Wright State University. 2011. – Forschungsbericht Referenziert auf Seite 37.

[Moa97] Moats, Ryan: *RFC 2141: URN Syntax*. `http://www.ietf.org/rfc/rfc2142.txt`, 1997 Referenziert auf Seite 27.

[MPP12] Motik, Boris ; Patel-Schneider, Peter F. ; Parsia, Bijan: *OWL 2 Web Ontology Language – Structural Specification and Functional-Style Syntax (Second Edition)*. `https://www.w3.org/TR/owl2-syntax/`, 2012 Referenziert auf Seite 39.

[Mus15] Musen, Mark: The Protégé project: A look back and a look forward. In: *AI Matters* 1 (2015), Nr. 4, S. 4–12. – ISSN 2372–3483 Referenziert auf Seite 123.

[MYD+15] Martínez, Anny ; Yannuzzi, Marcelo ; De Vergara, Jorge E. L. ; Serral-Gracià, René ; Ramírez, Wilson: An Ontology-Based Information Extraction System for bridging the configuration gap in hybrid SDN environments. In: Badonnel, Remi (Hrsg.) ; Xiao, Jin (Hrsg.) ; Ata, Shingo (Hrsg.) ; De Turck, Filip (Hrsg.) ; Groza, Voicu (Hrsg.) ; dos Santos, P. Carlos R. (Hrsg.): *Proceedings of the 14th IFIP/IEEE Symposium on Integrated Network and Service Management*. Ottawa, Kanada : IEEE, 2015. – ISBN 978–3–901882–76–0, S. 441–449 Referenziert auf Seite 175.

[Nat11] Natschläger, Christine: Towards a BPMN 2.0 Ontology. In: Dijkman, Remco (Hrsg.) ; Hofstetter, Jörg (Hrsg.) ; Koehler, Jana (Hrsg.): *Business Process Model and Notation* Bd. 95, Springer, November 2011 (Lecture Notes in Business Information Processing). – ISBN 978–3–642–25159–7 Referenziert auf Seite 67.

[NB07] Nardi, Daniele ; Brachman, Ronald J.: An Introduction to Description Logics. In: Baader, Franz (Hrsg.) ; Calvanese, Diego (Hrsg.) ; McGuinness, Deborah L. (Hrsg.) ; Nardi, Daniele (Hrsg.) ; Patel-Schneider, Peter F. (Hrsg.): *The Description Logic Handbook: Theory, Implementation and Applications (Second Edition)*. Cambridge University Press, 2007. – ISBN 978–0521150118, S. 1–43 Referenziert auf Seite 36.

[NFF+91] Neches, Robert ; Fikes, Richard ; Finin, Tim ; Gruber, Thomas ; Patil, Ramesh ; Senator, Ted ; Swartout, William: Enabling Technology for Knowledge Sharing. In: *AI Magazine* 12 (1991), Nr. 3 Referenziert auf Seite 23.

[NHSS11] Niemann, Michael ; Hombach, Sascha ; Schulte, Stefan ; Steinmetz, Ralf: Das IT-Governance-Framework CObIT als Wissensdatenbank - Entwurf, Umsetzung und Evaluation einer Ontologie. 2011. – Forschungsbericht Referenziert auf Seiten 67 und 69.

[NM01] Noy, Natalya ; McGuinness, Deborah L.: Ontology Development 101: A Guide to Creating Your First Ontology. (2001) Referenziert auf Seite 59.

[NP01] Niles, Ian ; Pease, Adam: Towards a standard upper ontology. In: *Proceedings of the international conference on Formal Ontology in Information Systems - FOIS '01* 2001 (2001), S. 2–9. ISBN 1581133774 Referenziert auf Seite 49.

[omg11] Object Management Group (OMG): *Business Process Model and Notation (BPMN), Version 2.0*. `http://www.omg.org/spec/BPMN/2.0`, Januar 2011. – Abgerufen am 10.03.2015. Referenziert auf Seite 67.

[omg14] Object Management Group (OMG): *Object Constraint Language™ (OCL™), Version 2.4*. `http://www.omg.org/spec/OCL/2.4/`, Februar 2014. – Abgerufen am 30.11.2016. Referenziert auf Seite 69.

[omg17] Object Management Group (OMG): *EDMC - Financial Industry Business Ontology® (FIBO®)*. `http://www.omg.org/spec/EDMC-FIBO/`, März 2017. – Abgerufen am 04.04.2017. Referenziert auf Seite 151.

[OOP14] *OOPS! OntOlogy Pitfall Scanner*. `http://oops.linkeddata.es/`, 2014. – Abgerufen am 16.07.2017. Referenziert auf Seite 59.

[Ope14] Open Services Gateway initiative Alliance: *OSGi Core Release 6*. 2014 Referenziert auf Seite 126.

[Ope15] Open Services Gateway initiative Alliance: *OSGi Compendium Release 6*. 2015 Referenziert auf Seiten 126 und 129.

[ORG15] Otero-Cerdeira, Lorena ; Rodríguez-Martínez, Francisco J. ; Gómez-Rodríguez, Alma: Ontology Matching: A literature review. In: *Expert Systems with Applications* 42 (2015), S. 949–971 Referenziert auf Seite 55.

[OSG15] Open Services Gateway initiative Alliance: *Open Services Gateway initiative*. `https://www.osgi.org/`, 2015. – Abgerufen am 16.07.2017. Referenziert auf Seite 125.

[OWL07] O'Sullivan, Declan ; Wade, Vincent ; Lewis, David: Understanding as We Roam. In: *IEEE Internet Computing* 11 (2007), März, Nr. 2, S. 26–33. – ISSN 1089–7801 Referenziert auf Seite 65.

[PBDG10] Presutti, Valentina ; Blomqvist, Eva ; Daga, Enrico ; Gangemi, Aldo: Pattern-Based Ontology Design. In: Suárez-Figueroa, Mari C. (Hrsg.) ; Gómez-Pérez, Asunción (Hrsg.) ; Motta, Enrico (Hrsg.) ; Gangemi, Aldo (Hrsg.): *Ontology Engineering in a Networked World*. Springer, 2010. – ISBN 978–3–642–24793–4, S. 35–64 Referenziert auf Seite 58.

[PCH+10] Paschke, Adrian ; Coskun, Gökhan ; Heese, Ralf ; Luczak-Rösch, Markus ; Oldakowski, Radoslaw ; Schäfermeier, Ralph ; Streibel, Olga: Corporate Semantic Web: Towards the Deployment of Semantic Technologies in Enterprises. In: Du, Weichang (Hrsg.) ; Ensan, Faezeh (Hrsg.): *Canadian Semantic Web: Technologies and Applications*. Springer, 2010. – ISBN 978–1–4419–7334–4, S. 105–131 Referenziert auf Seite 68.

[PCH+11] Paschke, Adrian ; Coskun, Gökhan ; Heese, Ralf ; Oldakowski, Radoslaw ; Rothe, Mario ; Schäfermeier, Ralph ; Streibel, Olga ; Teymourian, Kia ; Todor, Alexandru: Corporate Semantic Web Report IV / Freie Universität Berlin, Department of Mathematics and Computer Science, Corporate Semantic Web. 2011. – Forschungsbericht Referenziert auf Seite 68.

[PCH+13] Paschke, Adrian ; Coskun, Gökhan ; Harasic, Marko ; Heese, Ralf ; Oldakowski, Radoslaw ; Schäfermeier, Ralph ; Streibel, Olga ; Teymourian, Kia ; Todor, Alexandru: Corporate Semantic Web Report VI: Validation and Evaluation / Freie Universität Berlin, Department of Mathematics and Computer Science, Corporate Semantic Web. 2013. – Forschungsbericht Referenziert auf Seite 68.

[PD09] Phillips, Addison ; Davis, Mark: *Best Current Practices 47: Tags for Identifying Languages*. `https://tools.ietf.org/html/bcp47`, 2009 Referenziert auf Seite 30.

[PDGB09] Presutti, Valentina ; Daga, Enrico ; Gangemi, Aldo ; Blomqvist, Eva: eXtreme Design with Content Ontology Design Patterns. In: *Proceedings of the Workshop on Ontology Patterns*. Washington DC, USA, 2009 Referenziert auf Seite 58.

[Pen13] Penicina, Ludmila: Choosing a BPMN 2.0 Compatible Upper Ontology. In: *Proceedings of eKNOW 2013: The Fifth International Conference on Information, Process and Knowledge Management.* Nizza, Frankreich, 2013 Referenziert auf Seite 67.

[PHH04] Patel-Schneider, Peter F. ; Hayes, Patrick ; Horrocks, Ian: *OWL Web Ontology Language Semantics and Abstract Syntax.* `http://www.w3.org/TR/owl-semantics/`, 2004 Referenziert auf Seite 35.

[PM12] Patel-Schneider, Peter F. ; Motik, Boris: *OWL 2 Web Ontology Language – Mapping to RDF Graphs (Second Edition).* `https://www.w3.org/TR/owl2-mapping-to-rdf/`, 2012 Referenziert auf Seite 39.

[PN02] Pease, Adam ; Niles, Ian: IEEE standard upper ontology: a progress report. In: *The Knowledge Engineering Review* 1 (2002), Nr. 17, S. 65–70 Referenziert auf Seite 49.

[PNL02] Pease, Adam ; Niles, Ian ; Li, John: The Suggested Upper Merged Ontology: A Large Ontology for the Semantic Web and its Applications. In: *Proceedings of the AAAI-2002 Workshop on Ontologies and the Semantic Web.* Edmonton, Alberta, Kanada, 2002 Referenziert auf Seite 49.

[Pre08] Presutti, Valentina: *TimeInterval Ontology Design Pattern.* `http://ontologydesignpatterns.org/wiki/Submissions:TimeInterval`, 2008. – Abgerufen am 16.07.2017. Referenziert auf Seite 158.

[PS09] Parent, Christine ; Spaccapietra, Stefano: An Overview of Modularity. In: Stuckenschmidt, Heiner (Hrsg.) ; Parent, Christine (Hrsg.) ; Spaccapietra, Stefano (Hrsg.): *Modular Ontologies* Bd. 5445. Springer, 2009. – ISBN 978–3–642–01906–7, S. 5–24 Referenziert auf Seite 54.

[PSG10a] Poveda, María ; Suárez-Figueroa, Mari C. ; Gómez-Pérez, Asunción: Common Pitfalls in Ontology Development. In: *Proceedings of the Current Topics in Artificial Intelligence, and 13th Conference on Spanish Association for Artificial Intelligence.* Sevilla, Spanien : Springer, 2010. – ISBN 978–3–642–14263–5, S. 91–100 Referenziert auf Seite 59.

[PSG10b] Poveda-Villalón, María ; Suárez-Figueroa, Mari C. ; Gómez-Pérez, Asunción: A Double Classification of Common Pitfalls in Ontologies. In: *Workshop on Ontology Quality (OntoQual 2010), Co-located with EKAW 2010.* Lissabon, Portugal, Oktober 2010. – ISSN 1613–0073 Referenziert auf Seite 59.

[PSG15] Poveda-Villalón, María ; Suárez-Figueroa, Mari C. ; Gómez-Pérez, Asunción: Did you validate you ontology? OOPS! In: Simperl, Elena (Hrsg.) ; Norton, Barry (Hrsg.) ; Mladenic, Dunja (Hrsg.) ; Valle, Emanuele D. (Hrsg.) ; Fundulaki, Irini (Hrsg.) ; Passant, Alexandre (Hrsg.) ; Troncy, Raphaël (Hrsg.): *The Semantic Web:*

ESWC 2012 Satellite Events Bd. 7540, Springer, 2015 (Lecture Notes in Computer Science). – ISBN 978–3–662–46640–7, S. 402–407 Referenziert auf Seite 59.

[PST04] Pinto, Helena S. ; Staab, Steffen ; Tempich, Christoph: DILIGENT: Towards a fine-grained methodology for DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies. In: *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI 2004)*, IOS Press, 2004 Referenziert auf Seiten 52, 53 und 96.

[PTS09] Pinto, Helena S. ; Tempich, Christoph ; Staab, Steffen: Ontology Engineering and Evolution in a Distributed World Using DILIGENT. In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 153–176 Referenziert auf Seiten 52, 53 und 96.

[QUO14] Organization for the Advancement of Structured Information Standards (OASIS): *OASIS Quantities and Units of Measure Ontology Standard (QUOMOS) TC*. `https://wiki.oasis-open.org/quomos`, 2014. – Abgerufen am 21.02.2017. Referenziert auf Seite 117.

[Ran11] Rance, Stuart: *ITIL Service Transition*. The Stationery Office Ltd, 2011. – ISBN 978–0113313068 Referenziert auf Seite 12.

[Rei09] Reinhardt, Simon: *Dublin Core in OWL 2*. `http://bloody-byte.net/rdf/dc_owl2dl/index.html`, 2009. – Abgerufen am 16.07.2017. Referenziert auf Seite 114.

[Rei16] *The State of the Module System*. `http://openjdk.java.net/projects/jigsaw/spec/sotms/`, 2016. – Abgerufen am 16.07.2017. Referenziert auf Seite 125.

[Rog06] Rogers, Jeremy: Quality Assurance of Medical Ontologies. In: *Methods Inf Med* 45 (2006), Nr. 3, S. 267–274 Referenziert auf Seite 60.

[Roh08] Rohloff, Michael: An integrated view on business- and IT-architecture. In: *Proceedings of the 2008 ACM symposium on Applied computing - SAC '08* (2008), S. 561. ISBN 9781595937537 Referenziert auf Seite 69.

[RPKC11] Roussey, Catherine ; Pinet, Francois ; Kang, Myoung A. ; Corcho, Oscar: An Introduction to Ontologies and Ontology Engineering. In: Falquet, Gilles (Hrsg.) ; Métral, Claudine (Hrsg.) ; Teller, Jacques (Hrsg.) ; Tweed, Christopher (Hrsg.): *Ontologies in Urban Development Projects*. Springer, 2011 (Advanced Information and Knowledge Processing). – ISBN 978–0–85729–723–5, S. 9–38 Referenziert auf Seite 69.

[SAA+99] Schreiber, Guus ; Akkermans, Hans ; Anjewierden, Anjo ; De Hoog, Robert ; Shadbolt, Nigel R. ; Van de Velde, Walter ; Wielinga, Bob: *Knowledge Engineering and Management: The CommonKADS Methodology*. MIT Press, 1999. – ISBN 978–0262193009 Referenziert auf Seite 50.

[SAB+08] Suárez-Figueroa, Mari C. ; Aguado de Cea, Guadalupe ; Buil, Carlos ; Dellschaft, Klaas ; Fernández-López, Mariano ; García, Andrés ; Gómez-Pérez, Asunción ; Herrero, German ; Montiel-Ponsoda, Elena ; Sabou, Marta ; Villazón-Terrazas, Boris ; Yufei, Zheng: NeOn Methodology for Building Contextualized Ontology Networks (NeOn Deliverable D5.4.1). 2008. – Forschungsbericht Referenziert auf Seiten 52 und 53.

[SBB+13] Semeraro, Giovanni ; Basile, Pierpaolo ; Basili, Roberto ; De Gemmis, Marco ; Ghidini, Chiara ; Lenzerini, Maurizio ; Lops, Pasquale ; Moschitti, Alessandro ; Musto, Cataldo ; Narducci, Fedelucio ; Pipitone, Arianna ; Pirrone, Roberto ; Poccianti, Piero ; Serafini, Luciano: Semantic technologies for industry: From knowledge modeling and integration to intelligent applications. In: *Intelligenza Artificiale* 7 (2013), S. 125–137 Referenziert auf Seite 57.

[Sch09] Scharffe, François: *Correspondence Patterns Representation*, University of Innsbruck, Diss., 2009 Referenziert auf Seite 59.

[SCL+11] Steinberg, Randy A. ; Cannon, David ; Lloyd, Vernon ; Hunnebeck, Lou ; Rance, Stuart: *ITIL Lifecycle Suite 2011*. The Stationery Office Ltd, 2011. – ISBN 978–0113313235 Referenziert auf Seite 12.

[SD05] Scharffe, François ; De Bruijn, Jos: A Language to Specify Mappings Between Ontologies. In: Chbeir, Richard (Hrsg.) ; Dipanda, Albert (Hrsg.) ; Yétongnon, Kokou (Hrsg.): *Proceedings of the 1st International Conference on Signal-Image Technology and Internet-Based Systems (SITIS 2005)*. Yaoundé, Kamerun, 2005, S. 267–271 Referenziert auf Seiten 56 und 82.

[SDR+07] Strassner, John ; De Souza, José N. ; Raymer, David ; Samudrala, Srini ; Davy, Steven ; Barrett, Keara: The Design of a New Policy Model to Support Ontology-Driven Reasoning for Autonomic Networking. In: *2007 Latin American Network Operations and Management Symposium* (2007), September, S. 114–125. ISBN 978–1–4244–1181–8 Referenziert auf Seiten 64 und 70.

[SE07] Scharffe, François ; Euzenat, Jérôme: Expressive alignment language and implementation. 2007. – Forschungsbericht Referenziert auf Seiten 56 und 82.

[SE12] Shvaiko, Pavel ; Euzenat, Jérôme: Ontology matching: state of the art and future challenges. In: *IEEE Transactions on Knowledge and Data Engineering* (2012) Referenziert auf Seite 55.

[SF13] Souza Neto, João ; Ferreira Neto, Arthur N.: Metamodel of the IT Governance Framework COBIT. In: *Journal of Information Systems and Technology Management* 10 (2013), Dezember, Nr. 3, S. 521–540. – ISSN 1807–1775 Referenziert auf Seiten 65 und 69.

[SKW07] Suchanek, Fabian M. ; Kasneci, Gjergji ; Weikum, Gerhard: Yago: A Core of Semantic Knowledge. In: *Proceedings of the 16th international World Wide Web conference (WWW 2007)*, ACM Press, 2007 Referenziert auf Seite 50.

[SMB10] Simperl, Elena ; Malgorzata, Mochol ; Bürger, Tobias: Achieving Maturity: the State of Practice in Ontology Engineering in 2009. In: *International Journal of Computer Science and Applications* 7 (2010), Nr. 1, S. 45–65 Referenziert auf Seite 46.

[Smi98] Smith, Barry: Basic Concepts of Formal Ontology. In: Guarino, Nicola (Hrsg.): *Proceedings of Formal Ontology in Information Systems (FOIS'98)*, IOS Press, 1998 Referenziert auf Seite 48.

[SPG+07] Sirin, Evren ; Parsia, Bijan ; Grau, Bernardo C. ; Kalyanpur, Aditya ; Katz, Yarden: Pellet: A practical OWL-DL reasoner. In: *Web Semantics* 5 (2007), Nr. 2 Referenziert auf Seite 119.

[SPKR97] Swartout, Bill ; Patil, Ramesh ; Knight, Kevin ; Russ, Tom: Toward Distributed Use of Large-Scale Ontologies. In: *Proceedings of the AAAI97 Spring Symposium Series on Ontological Engineering*. Stanford, USA, März 1997 Referenziert auf Seiten 52 und 53.

[SRF98] Studer, Rudi ; Richard Benjamins, V. ; Fensel, Dieter: Knowledge engineering: Principles and methods. In: *Data & Knowledge Engineering* 25 (1998), Nr. 1, S. 191–197 Referenziert auf Seite 23.

[SSM12] Spanos, Dimitrios-Emmanuel ; Stavrou, Periklis ; Mitrou, Nikolas: Bringing Relational Databases into the Semantic Web: A Survey. In: *Semantic Web* 3 (2012), Nr. 2, S. 169–209 Referenziert auf Seite 57.

[SSS02] Sure, York ; Staab, Steffen ; Studer, Rudi: Methodology for Development and Employment of Ontology Based Knowledge Management Applications. In: *SIGMOD Record* Bd. 31. ACM, Dezember 2002. – ISSN 0163–5808, S. 18–23 Referenziert auf Seiten 52 und 53.

[SSS09] Sure, York ; Staab, Steffen ; Studer, Rudi: Ontology Engineering Methodology. In: Staab, Steffen (Hrsg.) ; Studer, Rudi (Hrsg.): *Handbook on Ontologies*. Springer, 2009. – ISBN 978–3–540–70999–2, S. 135–152 Referenziert auf Seiten 47, 52, 53 und 54.

[SSSS01] Staab, Steffen ; Schnurr, Hans-Peter ; Studer, Rudi ; Sure, York: Knowledge Processes and Ontologies. In: *Intelligent Systems* 16 (2001), Nr. 1. – ISSN 1541–1672 Referenziert auf Seiten 52 und 53.

[ST09] Serafini, Luciano ; Tamilin, Andrei: Composing Modular Ontologies with Distributed Description Logics. In: Stuckenschmidt, Heiner (Hrsg.) ; Parent, Christine (Hrsg.) ; Spaccapietra, Stefano (Hrsg.): *Modular Ontologies* Bd. 5445. Springer, 2009. – ISBN 978–3–642–01906–7, S. 321–347 Referenziert auf Seite 55.

[Ste11] Steinberg, Randy A.: *ITIL Service Operation*. The Stationery Office Ltd, 2011. – ISBN 978–0113313075 Referenziert auf Seite 12.

[Sto17] Storage Networking Industry Association (SNIA): *Storage Management Initiative Specification (SMI-S)*. `https://www.snia.org/tech_activities/standards/curr_standards/smi`, 2017. – Abgerufen am 16.07.2017. Referenziert auf Seite 149.

[Sur03] Sure, York: *Methodology, Tools and Case Studies for Ontology based Knowledge Management*, Universität Karlsruhe, Diss., 2003 Referenziert auf Seiten 47 und 54.

[SvJP09] Strassner, John ; van der Meer, Sven ; Jennings, Brendan ; Ponce de Leon, Miguel: Semantic Mediation to Enable Network and Service Management Interoperability in Future Internet Networks. In: Cunningham, Paul (Hrsg.) ; Cunningham, Miriam (Hrsg.): *eChallenges e-2009 Conference Proceedings*, 2009. – ISBN 978–1–905824–13–7 Referenziert auf Seiten 64 und 70.

[SvOD09] Strassner, John ; van der Meer, Sven ; O'Sullivan, Declan ; Dobson, Simon: The Use of Context-Aware Policies and Ontologies to Facilitate Business-Aware Network Management. In: *Journal of Network and Systems Management* 17 (2009), September, Nr. 3, S. 255–284. – ISBN 1092200991 Referenziert auf Seiten 64, 69 und 70.

[SWJ95] Schreiber, Guus ; Wielinga, Bob ; Jansweijer, Wouter: The KACTUS View on the 'O'Word. In: *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995 Referenziert auf Seite 52.

[Syl11] Sylvester, Delton: ISO 38500 - Why Another Standard? In: *COBIT Focus - Using COBIT, Val IT, Risk IT, BMIS and ITAF* 2 (2011), April Referenziert auf Seite 15.

[Tex12] Textor, Andreas: Bi-Directional Ontology Updates Using Lenses. In: Cacciagrano, Diletta R. (Hrsg.) ; Dini, Petre (Hrsg.): *Proceedings of the Sixth International Conference on Advances in Semantic Processing (SEMAPRO)*. Barcelona, Spanien, 2012. – ISBN 978–1–61208–240–0, S. 71–74

[Tex14] Textor, Andreas: Ein Ansatz für eine Ontologie-basierte Verbindung von IT Monitoring und IT Governance. In: Plödereder, Erhard (Hrsg.) ; Grunske, Lars (Hrsg.) ; Schneider, Erik (Hrsg.) ; Ull, Dominik (Hrsg.): *44. Jahrestagung der Gesellschaft für Informatik, Workshop Management komplexer IT-Systeme und Anwendungen* Bd. 232, Gesellschaft für Informatik, 2014. – ISBN 978–3–88579–626–8, S. 845–856 Referenziert auf Seite 66.

[TG15] Textor, Andreas ; Geihs, Kurt: Calculation of COBIT Metrics Using a Formal Ontology. In: Appleton, Owen (Hrsg.) ; Brenner, Michael (Hrsg.) ; Schaaf, Thomas (Hrsg.) ; Tortonesi, Mauro (Hrsg.) ; De Turck, Filip (Hrsg.) ; Groza, Voicu (Hrsg.) ; dos Santos, P. Carlos R. (Hrsg.): *Proceedings of the 10th International Workshop on Advanced IT Service Management (BDIM).* Ottawa, Kanada : IEEE, 2015. – ISBN 978–3–901882–76–0, S. 1384–1390 Referenziert auf Seiten 66 und 100.

[TGK17] Textor, Andreas ; Geihs, Kurt ; Kroeger, Reinhold: Semantic Models for Bridging Domains in Automated IT Management: Lessons Learned. In: *Proceedings of 2017 International Conference on Networked Systems (NetSys).* Göttingen : IEEE, März 2017. – ISBN 978–1–5090–4394–1

[TKM05] Terziev, Ivan ; Kiryakov, Atanas ; Manov, Dimitar: Base upper-level ontology (BULO) Guidance. 2005. – Forschungsbericht Referenziert auf Seite 49.

[TMK12] Textor, Andreas ; Meyer, Fabian ; Kroeger, Reinhold: Automated IT Management using Ontologies. In: *International Journal on Advances in Intelligent Systems* 5 (2012), Dezember, Nr. 3/4, S. 291–301. – ISSN 1942–2679 Referenziert auf Seiten 64 und 70.

[TS15] Textor, Andreas ; Sikora, Thomas: An Ontology-Based Architecture for Storage Management. In: Cuel, Roberta (Hrsg.) ; Young, Robert (Hrsg.): *Formal Ontologies Meet Industry - 7th International Workshop* Bd. 225, Springer, August 2015 (Lecture Notes in Business Information Processing). – ISBN 978–3–319–21544–0, S. 63–74 Referenziert auf Seiten 64 und 70.

[TSK10] Textor, Andreas ; Stynes, Jeanne ; Kroeger, Reinhold: Transformation of the Common Information Model to OWL. In: Daniel, Florian (Hrsg.) ; Facca, Federico M. (Hrsg.): *Current Trends in Web Engineering - 10th International Conference on Web Engineering ICWE 2010 Workshops* Bd. 6385, Springer, Juli 2010 (Lecture Notes in Computer Science). – ISBN 978–3–642–16984–7, S. 163–174 Referenziert auf Seiten 64 und 70.

[TSK11] Textor, Andreas ; Stynes, Jeanne ; Kroeger, Reinhold: IT Management Using a Heavyweight CIM Ontology. In: Luttenberger, Norbert (Hrsg.) ; Peters, Hagen (Hrsg.): *17th GI/ITG Conference on Communication in Distributed Systems*

(KiVS'11) Bd. 17, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, März 2011 (OpenAccess Series in Informatics (OASIcs)). – ISBN 978–3–939897–27–9, S. 73–84 Referenziert auf Seiten 64 und 70.

[UG96] Uschold, Mike ; Grüninger, Michael: Ontologies: Principles, Methods and Applications. In: *Knowledge Engineering Review* 11 (1996), Juni Referenziert auf Seiten 51 und 52.

[UK95] Uschold, Mike ; King, Martin: Towards a Methodology for Building Ontologies. In: *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjunction with IJCAI-95*, 1995 Referenziert auf Seite 52.

[Usc96] Uschold, Mike: Building Ontologies: Towards a Unified Methodology. In: *Proceedings of Expert Systems '96, the 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems*, 1996 Referenziert auf Seite 52.

[vaa16] *Vaadin User Interface Components*. `https://vaadin.com/home`, 2016 Referenziert auf Seite 128.

[Val11] Valiente-Blásquez, María-Cruz: *Improving IT Service Management using an Ontology-Based and Model-Driven Approach*, Universidad de Alcalá, Diss., 2011 Referenziert auf Seiten 66, 69, 71 und 176.

[Van02] Van Bon, Jan: *The Guide to IT Service Management 2002: Vol. 1.* Addison Wesley, 2002. – ISBN 978–0201737929 Referenziert auf Seiten 12 und 13.

[VGS12] Valiente, María-Cruz ; Garcia-Barriocanal, Elena ; Sicilia, Miguel Ángel: Applying Ontology-Based Models for Supporting Integrated Software Development and IT Service Management Processes. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42 (2012), Januar, Nr. 1, S. 61–74. – ISSN 1094–6977 Referenziert auf Seiten 66, 71 und 176.

[vmw17] *VMWare vSphere*. `http://www.vmware.com/products/vsphere.html`, 2017. – Abgerufen am 16.07.2017. Referenziert auf Seite 150.

[VSC+10] Veres, Csaba ; Sampson, Jennifer ; Cox, Karl ; Bleistein, Steven ; Verner, June: An Ontology-Based Approach for Supporting Business-IT Alignment. In: Xhafa, Fatos (Hrsg.) ; Barolli, Leonard (Hrsg.) ; Papajorgji, Petraq J. (Hrsg.): *Complex Intelligent Systems and Their Applications* Bd. 41. Springer, 2010. – ISBN 978–1–4419–1635–8, S. 21–43 Referenziert auf Seiten 65 und 69.

[vSW97] van Heijst, Gertjan ; Schreiber, Guus ; Wielinga, Bob: Using explicit ontologies for KBS development. In: *International Journal of Human-Computer Studies* 2 (1997), Nr. 42, S. 183–292 Referenziert auf Seite 48.

[VVR11] Valiente, María-Cruz ; Vicente-Chicote, Cristina ; Rodríguez, Daniel: An Ontology-Based and Model-Driven Approach for Designing IT Service Management Systems. In: *International Journal of Service Science, Management, Engineering, and Technology* 2 (2011), April, Nr. 2, S. 65–81 Referenziert auf Seiten 66, 71 und 176.

[w3c12] W3C OWL Working Group: *OWL 2 Web Ontology Language Document Overview (Second Edition)*. `http://www.w3.org/TR/owl2-overview/`, 2012 Referenziert auf Seiten 38 und 40.

[w3c13a] World Wide Web Consortium (W3C): *Semantic Web Layer Cake*. `https://www.w3.org/2001/sw/`, 2013 Referenziert auf Seite 26.

[w3c13b] The SPARQL Working Group: *SPARQL Protocol and RDF Query Language (SPARQL)*. `http://www.w3.org/TR/sparql11-overview/`, 2013. – Abgerufen am 05.01.2015. Referenziert auf Seite 44.

[w3c14a] World Wide Web Consortium (W3C): *RDF sample graph*. `https://www.w3.org/TR/rdf-syntax-grammar/#figure1`, 2014 Referenziert auf Seite 32.

[w3c14b] World Wide Web Consortium (W3C): *Resource Description Framework (RDF)*. `http://www.w3.org/RDF/`, 2014. – Abgerufen am 16.12.2014. Referenziert auf Seite 29.

[w3c14c] World Wide Web Consortium (W3C): *Resource Description Framework Schema (RDFS)*. `http://www.w3.org/TR/rdf-schema/`, 2014. – Abgerufen am 16.12.2014. Referenziert auf Seite 33.

[WNR+06] Wang, Hai H. ; Noy, Natasha ; Rector, Alan ; Musen, Mark ; Redmond, Timothy ; Rubin, Daniel ; Tu, Samson ; Tudorache, Tania ; Drummond, Nick ; Horridge, Matthew ; Seidenberg, Julian: Frames and OWL Side by Side. In: *Proceedings of the 9th International Protégé Conference*. Stanford, USA, Juli 2006 Referenziert auf Seite 24.

[WR04] Weill, Peter ; Ross, Jeanne: *IT Governance - How Top Performers Manage IT Decision Rights for Superior Results*. Boston, Massachusetts, USA : Harvard Business School Press, 2004. – ISBN 1–59139–253–5 Referenziert auf Seite 14.

[WRPS05] Wong, Alfred Ka Y. ; Ray, Pradeep ; Parameswaran, N. ; Strassner, John: Ontology Mapping for the Interoperability Problem in Network Management. In: *IEEE Journal on Selected Areas in Communications* 23 (2005), Oktober, Nr. 10, S. 2058–2068 Referenziert auf Seite 65.

[WVV+01] Wache, Holger ; Vögele, Thomas ; Visser, Ubbo ; Stuckenschmidt, Heiner ; Schuster, Gerhard ; Neumann, Holger ; Hübner, Sebastian: Ontology-Based Integration of Information - A Survey of Existing Approaches. In: *Proceedings of IJCAI'01 Workshop on Ontologies and Information Sharing*, 2001, S. 108–117 Referenziert auf Seite 57.

[WW95] Wand, Yair ; Weber, Ron: On the deep structure of information systems. In: *Information Systems* 3 (1995), Nr. 5, S. 203–223 Referenziert auf Seite 48.

[WYRP08] Wong, Alfred ; Yip, Frederick ; Ray, Pradeep ; Paramesh, Nandan: Towards Semantic Interoperability for IT Governance: An Ontological Approach. In: *Computing and Informatics* 27 (2008), S. 131–155 Referenziert auf Seiten 67 und 69.

[ZW06] Zarvić, Novica ; Wieringa, Roel: An Integrated Enterprise Architecture Framework for Business-IT Alignment. In: Latour, Thibaud (Hrsg.) ; Petit, Michaël (Hrsg.): *Proceedings of Workshops and Doctoral Consortium, The 18th International Conference on Advanced Information Systems Engineering - Trusted Information Systems (CAiSE'06)*. Namur University Press, Juni 2006. – ISBN 2–87037–525–5, S. 262–270 Referenziert auf Seite 16.

A Publikationen

Im Zusammenhang mit dieser Dissertation sind folgende wissenschaftliche Veröffentlichungen entstanden:

1. Textor, Andreas ; Geihs, Kurt ; Kroeger, Reinhold: Semantic Models for Bridging Domains in Automated IT Management: Lessons Learned. In: *Proceedings of 2017 International Conference on Networked Systems (NetSys)*. Göttingen : IEEE, März 2017. – ISBN 978–1–5090–4394–1.

2. Textor, Andreas ; Sikora, Thomas: An Ontology-Based Architecture for Storage Management. In: Cuel, Roberta (Hrsg.) ; Young, Robert (Hrsg.): *Formal Ontologies Meet Industry - 7th International Workshop* Bd. 225, Springer, August 2015 (Lecture Notes in Business Information Processing). – ISBN 978–3–319–21544–0, S. 63–74.

3. Textor, Andreas ; Geihs, Kurt: Calculation of COBIT Metrics Using a Formal Ontology. In: Appleton, Owen (Hrsg.) ; Brenner, Michael (Hrsg.) ; Schaaf, Thomas (Hrsg.) ; Tortonesi, Mauro (Hrsg.) ; De Turck, Filip (Hrsg.) ; Groza, Voicu (Hrsg.) ; dos Santos, P. Carlos R. (Hrsg.): *Proceedings of the 10th International Workshop on Advanced IT Service Management (BDIM)*. Ottawa, Kanada : IEEE, 2015. – ISBN 978–3–901882–76–0, S. 1384–1390.

4. Textor, Andreas: Ein Ansatz für eine Ontologie-basierte Verbindung von IT Monitoring und IT Governance. In: Plödereder, Erhard (Hrsg.) ; Grunske, Lars (Hrsg.) ; Schneider, Erik (Hrsg.) ; Ull, Dominik (Hrsg.): *44. Jahrestagung der Gesellschaft für Informatik, Workshop Management komplexer IT-Systeme und Anwendungen* Bd. 232, Gesellschaft für Informatik, 2014. – ISBN 978–3–88579–626–8, S. 845–856.

5. Textor, Andreas: Bi-Directional Ontology Updates Using Lenses. In: Cacciagrano, Diletta R. (Hrsg.) ; Dini, Petre (Hrsg.): *Proceedings of the Sixth International Conference on Advances in Semantic Processing (SEMAPRO)*. Barcelona, Spanien, 2012. – ISBN 978–1–61208–240–0, S. 71–74.

6. Textor, Andreas ; Meyer, Fabian ; Kroeger, Reinhold: Automated IT Management using Ontologies. In: *International Journal on Advances in Intelligent Systems* 5 (2012), Dezember, Nr. 3/4, S. 291–301. – ISSN 1942–2679.

B COBIT 5 Prozesse

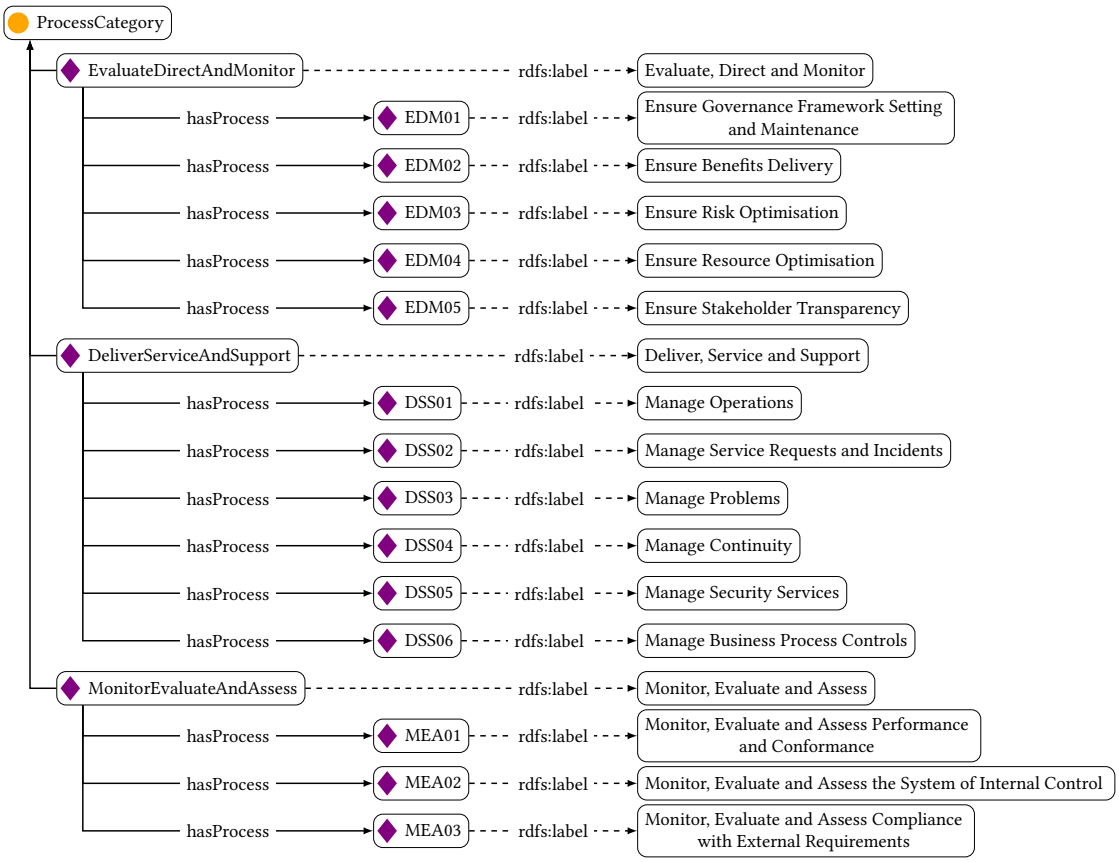

Abbildung B.1: COBIT Prozesskategorien und zugehörige Prozesse: EDM, DSS, MEA

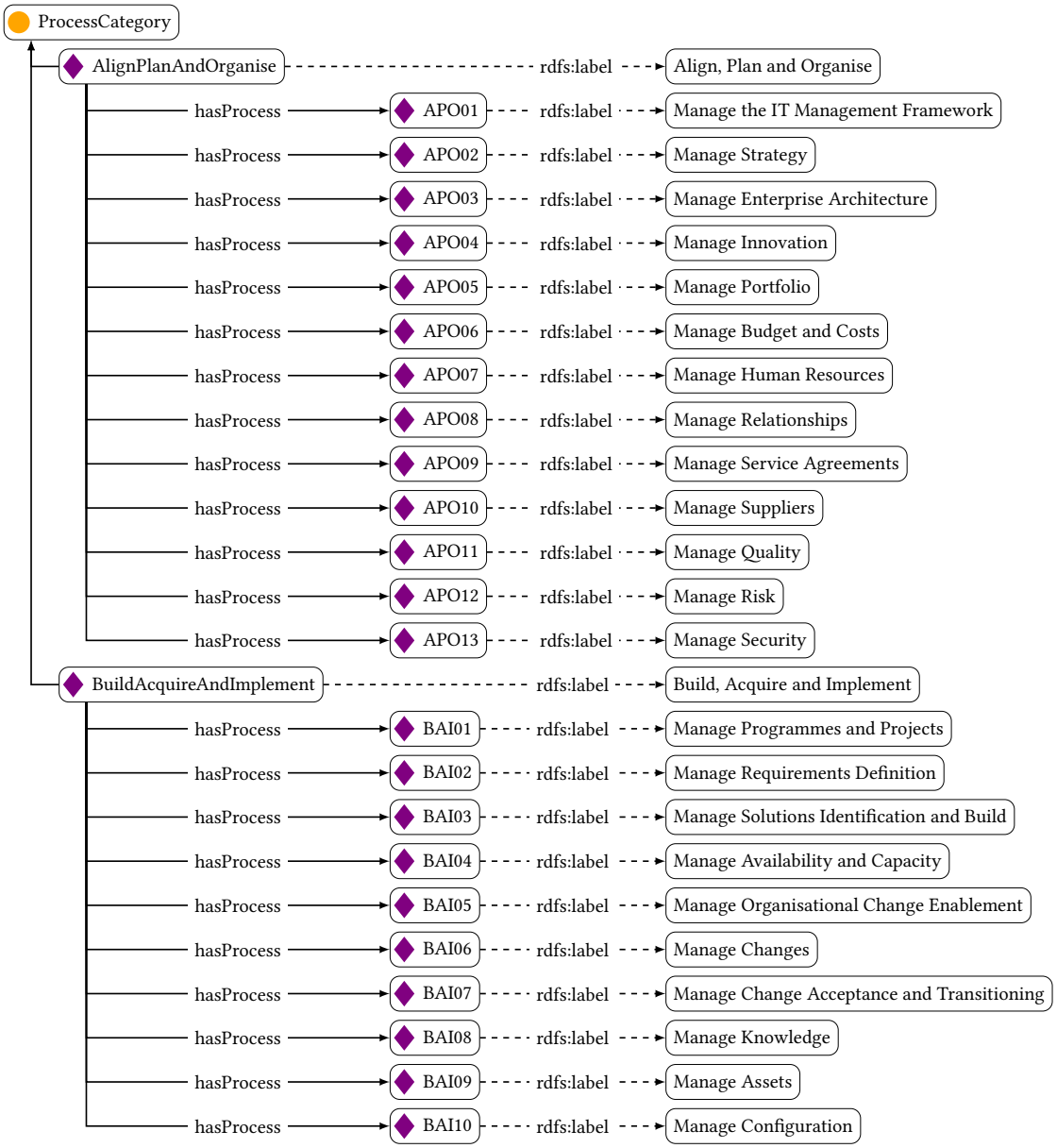

Abbildung B.2: COBIT Prozesskategorien und zugehörige Prozesse: APO, BAI

C Nötige Anpassungen an QUDT

Die Ontologien, in denen QUDT Version 1.0 verfügbar ist, enthalten Inkonsistenzen, die ein Reasoning unmöglich machen. In diesem Abschnitt werden die Inkonsistenzen sowie die nötigen Anpassungen zu ihrer Auflösung beschrieben.

QUDT enthält unter anderem die folgenden Axiome:

$$\begin{aligned} \text{Dimension} &\sqsubseteq \text{EnumerationElement} \\ \text{EnumerationElement} &\sqsubseteq = 1\ \text{literal.Literal} \\ \text{EnumerationElement} &\sqsubseteq \leq 1\ \text{symbol.Literal} \\ \text{symbol} &\sqsubseteq \text{literal} \end{aligned}$$

Gleichzeitig sind aber für zahlreiche Individuuen der Klasse *Dimension* sowohl ein *symbol* als auch *literal* festlegt. Dies führt zu einer Inkonsistenz, da die Kardinalitätsrestriktion auf *symbol* verletzt wird. Die hier umgesetzte Lösung besteht aus der Löschung der Unterrollenbeziehung von *symbol* und *literal*, die die beiden Rollen effektiv disjunkt macht. Für den Einsatz in dieser Arbeit hat diese Anpassung keinen nachteiligen Effekt, kann aber nicht als generelle Lösung angesehen werden.

Außerdem sind zahlreiche Zahlenwerte des Typs `xsd:double`, die im QUDT-Katalog insbesondere physikalische Konstanten repräsentieren[1], keine lexikalisch gültigen Werte des Datentyps, da sie Auslassungspunkte beinhalten, z.B.:

```
<qudt:quantityValue>
  <nist:CODATAValue rdf:about="http://physics.nist.gov/cuu/CODATA-Value_MagneticConstant">
    <qudt:unit rdf:resource="http://data.nasa.gov/qudt/owl/unit#FaradPerMeter"/>
    <qudt:numericValue rdf:datatype="http://www.w3.org/2001/XMLSchema#double"
    >12.566370614...e-7</qudt:numericValue>
    <rdfs:label>CODATA value for magnetic constant</rdfs:label>
  </nist:CODATAValue>
</qudt:quantityValue>
```

Die umgesetze Lösung ist eine Löschung der Auslassungen in `xsd:double` Literalen.

[1] Siehe `http://data.qudt.org/qudt/owl/1.0.0/nist-constants.owl`

D Abkürzungen

API	Application Programming Interface
APO	Align, Plan and Organise
AST	Abstract Syntax Tree
BAI	Build, Acquire and Implement
BFO	Basic Formal Ontology
BMIS	Business Model for Information Security
BPMN	Business Process Model Notation
BVQ	Business Volume Qualicision
CEO	Chief Executive Officer
CFO	Chief Financial Officer
CIM	Common Information Model
CIO	Chief Information Officer
CISO	Chief Information Security Officer
CL	Common Logic
COO	Chief Operating Officer
CRM	Customer Relationship Management
CRO	Chief Risk Officer
CLI	Command Line Interface
CLIF	Common Logic Interchange Format
CMDB	Configuration Management Database
CMIP	Common Management Information Protocol
COBIT	Control Objectives for Information and Related Technology
CP	Content Pattern
CQ	Competency Question

CSV Comma Separated Value

CURIE Compact URI

DAML DARPA Agent Markup Language

DCMI Dublin Core Metadata Initiative

DI Dependency Injection

DILIGENT DIstributed, Loosely-controlled and evolvInG Engineering of oNTologies

DL Description Logics

DMTF Distributed Management Task Force

DNS Domain Name System

DnS Descriptions and Situations

DOLCE Descriptive Ontology for Linguistic and Cognitive Engineering

DOM Document Object Model

DSS Deliver, Service and Support

DUL DOLCE+DnS Ultralight

EDM Evaluate, Direct and Monitor

EDMC Enterprise Data Management Council

EL Basierend auf Beschreibungslogik $\mathcal{EL}^{++}$ (OWL 2 Profil)

ER Entity Relationship

FIBO Financial Industry Business Ontology

FMEA Failure Mode and Effects Analysis

FO Formalisierungs-Ontologie

FOL First Order Logic

GDMO Guidelines for the Definition of Managed Objects

GFO General Formal Ontology

GRC Governance, Risk Management and Compliance

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS HTTP Secure

HZD Hessische Zentrale für Datenverarbeitung

iSCSI	Internet Small Computer Systems Interface
ID	Identifier
IDEF5	Integration Definition 5
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IOPS	Input/output operations per second
IP	Internet Protocol
IRI	Internationalized Resource Identifier
ISACA	Information Systems Audit and Control Association
ISO	International Standardisation Organisation
ITGI	IT Governance Institute
ITIL	IT Infrastructure Library
IT	Information Technology
ITSM	IT Service Management
ITSMO	IT Service Management Ontology
JEXL	Java Expression Language
JSON	JavaScript Object Notation
JSON-LD	JSON for Linked Data
JSTL	Java Server Pages Standard Tag Library
KACTUS	Knowledge about Complex Technical Systems for Multiple Use
KADS	Knowledge Acquisition and Documentation Structuring
KIF	Knowledge Interchange Format
LUN	Logical Unit Number
MAPE-K	Monitor, Analyze, Plan, Execute, Knowledge
MDA	Model Driven Architecture
MEA	Monitor, Evaluate and Assess
MES	Manufacturing Execution System
MOF	Managed Object Format
NeOn	Networked Ontologies

NLP Natural Language Processing
OASIS Organization for the Advancement of Structured Information Standards
OBO Open Biomedical Ontologies
OCL Object Constraint Language
ODPA Association for Ontology Design & Patterns
OIL Ontology Inference Layer
OMG Object Management Group
OSGi Open Services Gateway initiative
OWL Web Ontology Language
PA Process Attribute
PAM Process Assessment Model
Proton PROTo ONtology
QL Query Languages (OWL 2 Profil)
QName Qualified Name
QUDT Quantities, Units, Dimensions and Types
QUDV Quantities, Units, Dimensions, Values
QUOMOS Quantities and Units of Measure Ontology Standard
RACI Responsible, Accountable, Consulted, Informed
RDBMS Relational Database Management System
RDF Resource Description Framework
RDFa RDF in Attributes
RDFS RDF Schema
REST Representational state transfer
RFC Request for Comments
RIF Rule Interchange Format
RL Rule Languages (OWL 2 Profil)
RiskIT Risk IT Framework for Management of IT Related Business Risks
SAN Storage Area Network
SAS Serial Attached SCSI

SCSI Small Computer Systems Interface

SDN Software-defined Networking

SLA Service Level Agreement

SMI Structure of Management Information

SMI-S Storage Management Initiative Specification

SNIA Storage Networking Industry Association

SNMP Simple Network Management Protocol

SPARQL SPARQL Protocol and RDF Query Language

SPIN SPARQL Inferencing Notation

SQL Structured Query Language

SSH Secure Shell

SSL Secure Socket Layer

SUMO Suggested Upper Merged Ontology

SVC SAN Volume Controller

SWRL Semantic Web Rule Language

TLS Transport Layer Security

TOGAF The Open Group Architecture Framework

TOVE TOronto Virtual Enterprise project

TTL Terse Triple Language (auch *Turtle*)

UFO Unified Foundational Ontology

UMBEL Upper Mapping and Binding Exchange Layer

UML Unified Modeling Language

UNA Unique Name Assumption

UPON Unified PRocess for ONtology building

URI Uniform Resource Identifier

URL Uniform Resource Locator

URN Uniform Resource Name

US-ASCII American Standard Code for Information Interchange

ValIT Enterprise Value: Governance of IT Investments

VM	Virtual Machine
VMAN	Virtualization Management
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WBEM	Web-Based Enterprise Management
WID	Work Product ID
XHTML	Extensible Hypertext Markup Language
XSLT	Extensible Stylesheet Language Transformations
XML	Extensible Markup Language
XSD	XML Schema Definition
YAGO	Yet Another Great Ontology

E Verwendete Namensräume

Präfix	IRI
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs	http://www.w3.org/2000/01/rdf-schema#
owl	http://www.w3.org/2002/07/owl#
xsd	http://www.w3.org/2001/XMLSchema#
ex	http://example.org#
cobit5	http://purl.org/atextor/ontology/cobit5#
fo	http://wwwvs.cs.hs-rm.de/cobit5-fo#
list	http://purl.org/atextor/ontology/list#
agg	http://wwwvs.cs.hs-rm.de/aggregation#
dcterms	http://purl.org/dc/terms/
dcmi	http://purl.org/dc/dcmitype/
fibo-fnd-dt-fd	http://www.omg.org/spec/EDMC-FIBO/FND/DatesAndTimes/FinancialDates/
fibo-fnd-agr-ctr	http://www.omg.org/spec/EDMC-FIBO/FND/Agreements/Contracts/
bvq-cls	http://www.vs.cs.hs-rm.de/ontostorm/BVQ/classes#
bvq-obp	http://www.vs.cs.hs-rm.de/ontostorm/BVQ/objectproperties#
bvq-dbp	http://www.vs.cs.hs-rm.de/ontostorm/BVQ/dataproperties#
bvq-agg	http://www.vs.cs.hs-rm.de/ontostorm/BVQ/aggregations#
bvq-map	http://www.vs.cs.hs-rm.de/ontostorm/BVQ/mapping#
hzd	http://www.vs.cs.hs-rm.de/ontostorm/hzd#
hzd-map	http://www.vs.cs.hs-rm.de/ontostorm/hzd/mapping#
uc0	http://www.vs.cs.hs-rm.de/ontostorm/hzd/usecase0#
vmware	http://www.vs.cs.hs-rm.de/ontostorm/VMware#
qudt	http://data.nasa.gov/qudt/owl/qudt#
unit	http://data.nasa.gov/qudt/owl/unit#
quantity	http://data.nasa.gov/qudt/owl/quantity#
dimension	http://data.nasa.gov/qudt/owl/dimension#

F Schema der COBIT-Ontologie

$$
\begin{aligned}
\text{Activity} &\sqsubseteq = 1\ \text{activityNumber.Literal}\\
\text{Activity} &\sqsubseteq = 1\ \text{isActivityFor.Practice}\\
\text{BSCDimension} &\equiv \{\text{Customer}, \text{Financial}, \text{Internal}, \text{LearningAndGrowth}\}\\
\text{BSCDimension} &\sqsubseteq \forall\ \text{hasGoal.}(\text{EnterpriseGoal} \sqcup \text{ITGoal})\\
\text{BSCDimension} &\sqsubseteq \exists\ \text{hasGoal.Goal}\\
\text{Goal} &\sqsubseteq \forall\ \text{hasBSCDimension.BSCDimension}\\
\text{Goal} &\sqsubseteq \exists\ \text{hasBSCDimension.BSCDimension}\\
\text{Goal} &\sqsubseteq \forall\ \text{hasMetric.Metric}\\
\text{Goal} &\sqsubseteq \exists\ \text{hasMetric.Metric}\\
\text{EnterpriseGoal} &\sqsubseteq \text{Goal}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{hasMetric.EnterpriseGoalMetric}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{hasMetric.EnterpriseGoalMetric}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{hasPrimaryDependencyOn.ITGoal}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{hasPrimaryDependencyOn.ITGoal}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{hasSecondaryDependencyOn.ITGoal}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{hasSecondaryDependencyOn.ITGoal}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{hasPrimaryGovernanceObjective.GovernanceObjective}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{hasPrimaryGovernanceObjective.GovernanceObjective}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{hasSecondaryGovernanceObjective.GovernanceObjective}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{hasSecondaryGovernanceObjective.GovernanceObjective}\\
\text{EnterpriseGoal} &\sqsubseteq \forall\ \text{isAnswerToStakeholderNeed.StakeholderNeed}\\
\text{EnterpriseGoal} &\sqsubseteq \exists\ \text{isAnswerToStakeholderNeed.StakeholderNeed}\\
\text{ITGoal} &\sqsubseteq \text{Goal}\\
\text{ITGoal} &\sqsubseteq \forall\ \text{hasMetric.ITGoalMetric}\\
\text{ITGoal} &\sqsubseteq \exists\ \text{hasMetric.ITGoalMetric}\\
\text{ITGoal} &\sqsubseteq \forall\ \text{hasPrimaryDependencyOn.Process}
\end{aligned}
$$

$$
\begin{aligned}
\text{ITGoal} &\sqsubseteq \exists\ \text{hasPrimaryDependencyOn.Process}\\
\text{ITGoal} &\sqsubseteq \forall\ \text{hasSecondaryDependencyOn.Process}\\
\text{ITGoal} &\sqsubseteq \exists\ \text{hasSecondaryDependencyOn.Process}\\
\text{ITGoal} &\sqsubseteq \forall\ \text{isPrimarySupportFor.EnterpriseGoal}\\
\text{ITGoal} &\sqsubseteq \exists\ \text{isPrimarySupportFor.EnterpriseGoal}\\
\text{ITGoal} &\sqsubseteq \forall\ \text{isSecondarySupportFor.EnterpriseGoal}\\
\text{ITGoal} &\sqsubseteq \exists\ \text{isSecondarySupportFor.EnterpriseGoal}\\
\text{ProcessGoal} &\sqsubseteq \text{Goal}\\
\text{ProcessGoal} &\sqsubseteq \forall\ \text{hasMetric.ProcessGoalMetric}\\
\text{ProcessGoal} &\sqsubseteq \exists\ \text{hasMetric.ProcessGoalMetric}\\
\text{ProcessGoal} &\sqsubseteq \forall\ \text{isProcessGoalFor.Process}\\
\text{ProcessGoal} &\sqsubseteq \exists\ \text{isProcessGoalFor.Process}\\
\text{GovernanceObjective} &\equiv \{\text{BenefitsRealisation}, \text{ResourceOptimisation}, \text{RiskOptimisation}\}\\
\text{GovernanceObjective} &\sqsubseteq \forall\ \text{isPrimaryGovernanceObjectiveFor.EnterpriseGoal}\\
\text{GovernanceObjective} &\sqsubseteq \exists\ \text{isPrimaryGovernanceObjectiveFor.EnterpriseGoal}\\
\text{GovernanceObjective} &\sqsubseteq \forall\ \text{isSecondaryGovernanceObjectiveFor.EnterpriseGoal}\\
\text{GovernanceObjective} &\sqsubseteq \exists\ \text{isSecondaryGovernanceObjectiveFor.EnterpriseGoal}\\
\text{Metric} &\sqsubseteq \forall\ \text{isMetricFor.Goal}\\
\text{Metric} &\sqsubseteq \exists\ \text{isMetricFor.Goal}\\
\text{EnterpriseGoalMetric} &\sqsubseteq \text{Metric}\\
\text{EnterpriseGoalMetric} &\sqsubseteq \forall\ \text{isMetricFor.EnterpriseGoal}\\
\text{EnterpriseGoalMetric} &\sqsubseteq \exists\ \text{isMetricFor.EnterpriseGoal}\\
\text{ITGoalMetric} &\sqsubseteq \text{Metric}\\
\text{ITGoalMetric} &\sqsubseteq \forall\ \text{isMetricFor.ITGoal}\\
\text{ITGoalMetric} &\sqsubseteq \exists\ \text{isMetricFor.ITGoal}\\
\text{ProcessGoalMetric} &\sqsubseteq \text{Metric}\\
\text{ProcessGoalMetric} &\sqsubseteq \forall\ \text{isMetricFor.ProcessGoal}\\
\text{ProcessGoalMetric} &\sqsubseteq \exists\ \text{isMetricFor.ProcessGoal}\\
\text{PerformanceAttribute} &\equiv \{\text{PA1.1}\} \sqcup \{\text{PA}i.j \mid i \in \{2 \ldots 5\}, j \in \{1, 2\}\}\\
\text{PerformanceAttribute} &\sqsubseteq \forall\ \text{isPerformanceAttributeFor.ProcessCapabilityLevel}\\
\text{PerformanceAttribute} &\sqsubseteq \exists\ \text{isPerformanceAttributeFor.ProcessCapabilityLevel}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasAccountableRole.Role}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasAccountableRole.Role}
\end{aligned}
$$

$$
\begin{aligned}
\text{Practice} &\sqsubseteq \forall\ \text{hasActivity.Activity}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasActivity.Activity}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasConsultedRole.Role}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasConsultedRole.Role}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasInformedRole.Role}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasInformedRole.Role}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasResponsibleRole.Role}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasResponsibleRole.Role}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasInput.WorkProduct}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasInput.WorkProduct}\\
\text{Practice} &\sqsubseteq \forall\ \text{hasOutput.WorkProduct}\\
\text{Practice} &\sqsubseteq \exists\ \text{hasOutput.WorkProduct}\\
\text{Practice} &\sqsubseteq \forall\ \text{isUsedInProcess.Process}\\
\text{Practice} &\sqsubseteq \exists\ \text{isUsedInProcess.Process}\\
\text{GovernancePractice} &\sqsubseteq \text{Practice}\\
\text{ManagementPractice} &\sqsubseteq \text{Practice}\\
\text{Process} &\equiv \{\text{APO}i \mid i \in \{01,\ldots,12\}\} \sqcup \{\text{BAI}i \mid i \in \{01,\ldots,10\}\}\\
&\quad \sqcup \{\text{DSS}i \mid i \in \{01,\ldots,06\}\} \sqcup \{\text{EDM}i \mid i \in \{01,\ldots,05\}\}\\
&\quad \sqcup \{\text{MEA}i \mid i \in \{01,02,03\}\}\\
\text{Process} &\sqsubseteq \exists\ \text{hasProcessCategory.ProcessCategory}\\
\text{Process} &\sqsubseteq \exists\ \text{hasGoal.ProcessGoal}\\
\text{Process} &\sqsubseteq \exists\ \text{isPrimarySupportFor.ITGoal}\\
\text{Process} &\sqsubseteq \exists\ \text{isSecondarySupportFor.ITGoal}\\
\text{Process} &\sqsubseteq \exists\ \text{hasManagementPractice.ManagementPractice}\\
\text{ProcessCapabilityLevel} &\equiv \{\text{EstablishedProcess}, \text{IncompleteProcess}\}\\
&\quad \sqcup \{\text{ManagementProcess}, \text{OptimisingProcess}\}\\
&\quad \sqcup \{\text{PerformedProcess}, \text{PredictableProcess}\}\\
\text{ProcessCapabilityLevel} &\sqsubseteq \exists\ \text{capabilityLevel.}\{\geq 0, \leq 5\}\\
\text{ProcessCapabilityLevel} &\sqsubseteq \forall\ \text{hasPerformanceAttribute.PerformanceAttribute}\\
\text{ProcessCategory} &\sqsubseteq \forall\ \text{hasParentProcessCategory.ProcessCategory}\\
\text{ProcessCategory} &\sqsubseteq \forall\ \text{hasProcess.Process}\\
\text{ProcessCategory} &\sqsubseteq \forall\ \text{hasSubProcessCategory.ProcessCategory}\\
\text{Role} &\sqsubseteq \exists\ \text{isAccountableFor.Practice}
\end{aligned}
$$

$$\begin{aligned}
\text{Role} &\sqsubseteq \exists\, \text{isConsultedAbout.Practice}\\
\text{Role} &\sqsubseteq \exists\, \text{isInformedAbout.Practice}\\
\text{Role} &\sqsubseteq \exists\, \text{isResponsibleFor.Practice}\\
\text{StakeholderNeed} &\sqsubseteq \exists\, \text{isAnsweredByEnterpriseGoal.EnterpriseGoal}\\
\text{StakeholderNeed} &\sqsubseteq \forall\, \text{isAnsweredByEnterpriseGoal.EnterpriseGoal}\\
\text{WorkProduct} &\sqsubseteq \forall\, \text{isCreatedByPractice.Practice}\\
\text{WorkProduct} &\sqsubseteq \exists\, \text{isCreatedByPractice.Practice}\\
\text{WorkProduct} &\sqsubseteq \forall\, \text{isUsedInPractice.Practice}\\
\text{WorkProduct} &\sqsubseteq \exists\, \text{isUsedInPractice.Practice}
\end{aligned}$$

$$\top \sqsubseteq \leqslant 1\, \text{hasEnterpriseGoal}$$
$$\mathit{Disjoint}(\text{hasEnterpriseGoal}, \text{hasEnterpriseGoal}^-)$$
$$\top \sqsubseteq \neg\exists\, \text{hasEnterpriseGoal}.\mathit{Self}$$
$$\geqslant 1\, \text{hasEnterpriseGoal} \sqsubseteq \text{BSCDimension}$$
$$\text{hasEnterpriseGoal} \sqsubseteq \text{hasGoal}$$
$$\top \sqsubseteq \forall\, \text{hasEnterpriseGoal.EnterpriseGoal}$$
$$\top \sqsubseteq \leqslant 1\, \text{hasGoal}$$
$$\mathit{Disjoint}(\text{hasGoal}, \text{hasGoal}^-)$$
$$\top \sqsubseteq \neg\exists\, \text{hasGoal}.\mathit{Self}$$
$$\geqslant 1\, \text{hasGoal} \sqsubseteq \text{BSCDimension}$$
$$\top \sqsubseteq \forall\, \text{hasGoal.Goal}$$
$$\text{hasGoal} \equiv \text{hasBSCDimension}^-$$
$$\top \sqsubseteq \leqslant 1\, \text{hasITGoal}$$
$$\mathit{Disjoint}(\text{hasITGoal}, \text{hasITGoal}^-)$$
$$\top \sqsubseteq \neg\exists\, \text{hasITGoal}.\mathit{Self}$$
$$\geqslant 1\, \text{hasITGoal} \sqsubseteq \text{BSCDimension}$$
$$\text{hasITGoal} \sqsubseteq \text{hasGoal}$$
$$\top \sqsubseteq \forall\, \text{hasITGoal.ITGoal}$$
$$\top \sqsubseteq \geqslant 1\, \text{isActivityFor}$$
$$\mathit{Disjoint}(\text{isActivityFor}, \text{isActivityFor}^-)$$
$$\top \sqsubseteq \neg\exists\, \text{isActivityFor}.\mathit{Self}$$
$$\geqslant 1\, \text{isActivityFor} \sqsubseteq \text{Activity}$$
$$\top \sqsubseteq \forall\, \text{isActivityFor.Practice}$$
$$\text{isActivityFor} \equiv \text{hasActivity}^-$$

$\geqslant 1$ isCreatedByPractice $\sqsubseteq$ WorkProduct

$\top \sqsubseteq \forall$ isCreatedByPractice.Practice

isCreatedByPractice $\equiv$ hasOutput$^-$

$\top \sqsubseteq \geqslant 1$ isCreatedByManagementPractice

Disjoint(isCreatedByManagementPractice, isCreatedByManagementPractice$^-$)

$\top \sqsubseteq \neg\exists$ isCreatedByManagementPractice.*Self*

$\geqslant 1$ isCreatedByManagementPractice $\sqsubseteq$ WorkProduct

$\top \sqsubseteq \forall$ isCreatedByManagementPractice.ManagementPractice

isCreatedByManagementPractice $\sqsubseteq$ isCreatedByPractice

isCreatedByManagementPractice $\equiv$ hasOutput$^-$

$\top \sqsubseteq \geqslant 1$ isCreatedByGovernancePractice

Disjoint(isCreatedByGovernancePractice, isCreatedByGovernancePractice$^-$)

$\top \sqsubseteq \neg\exists$ isCreatedByGovernancePractice.*Self*

$\geqslant 1$ isCreatedByGovernancePractice $\sqsubseteq$ WorkProduct

$\top \sqsubseteq \forall$ isCreatedByGovernancePractice.GovernancePractice

isCreatedByGovernancePractice $\sqsubseteq$ isCreatedByPractice

isCreatedByGovernancePractice $\equiv$ hasOutput$^-$

Disjoint(isAnswerToStakeholderNeed, isAnswerToStakeholderNeed$^-$)

$\top \sqsubseteq \neg\exists$ isAnswerToStakeholderNeed.*Self*

$\geqslant 1$ isAnswerToStakeholderNeed $\sqsubseteq$ EnterpriseGoal

$\top \sqsubseteq \forall$ isAnswerToStakeholderNeed.StakeholderNeed

isAnswerToStakeholderNeed $\equiv$ isAnsweredByEnterpriseGoal$^-$

Disjoint(hasPrimaryGovernanceObjective, hasPrimaryGovernanceObjective$^-$)

$\top \sqsubseteq \neg\exists$ hasPrimaryGovernanceObjective.*Self*

$\geqslant 1$ hasPrimaryGovernanceObjective $\sqsubseteq$ EnterpriseGoal

$\top \sqsubseteq \forall$ hasPrimaryGovernanceObjective.GovernanceObjective

hasPrimaryGovernanceObjective $\equiv$ isPrimaryGovernanceObjectiveFor$^-$

Disjoint(hasSecondaryGovernanceObjective, hasSecondaryGovernanceObjective$^-$)

$\top \sqsubseteq \neg\exists$ hasSecondaryGovernanceObjective.*Self*

$\geqslant 1$ hasSecondaryGovernanceObjective $\sqsubseteq$ EnterpriseGoal

$\top \sqsubseteq \forall$ hasSecondaryGovernanceObjective.GovernanceObjective

hasSecondaryGovernanceObjective $\equiv$ isSecondaryGovernanceObjectiveFor$^-$

Disjoint(hasPrimaryDependencyOn, hasPrimaryDependencyOn$^-$)

$\top \sqsubseteq \neg\exists$ hasPrimaryDependencyOn.*Self*

$\geqslant 1$ hasPrimaryDependencyOn $\sqsubseteq$ Goal

hasPrimaryDependencyOn $\equiv$ isPrimarySupportFor$^-$

Disjoint(hasSecondaryDependencyOn, hasSecondaryDependencyOn$^-$)

$\top \sqsubseteq \neg\exists$ hasSecondaryDependencyOn.*Self*

$\geqslant 1$ hasSecondaryDependencyOn $\sqsubseteq$ Goal

hasSecondaryDependencyOn $\equiv$ isSecondarySupportFor$^-$

$\top \sqsubseteq \geqslant 1$ hasBSCDimension

Disjoint(hasBSCDimension, hasBSCDimension$^-$)

$\top \sqsubseteq \neg\exists$ hasBSCDimension.*Self*

$\top \sqsubseteq \forall$ hasBSCDimension.BSCDimension

$\geqslant 1$ hasBSCDimension $\sqsubseteq$ Goal

$\top \sqsubseteq \leqslant 1$ hasMetric

Disjoint(hasMetric, hasMetric$^-$)

$\top \sqsubseteq \neg\exists$ hasMetric.*Self*

$\geqslant 1$ hasMetric $\sqsubseteq$ Goal

$\top \sqsubseteq \forall$ hasMetric.Metric

Disjoint(hasAccountableRole, hasAccountableRole$^-$)

$\top \sqsubseteq \neg\exists$ hasAccountableRole.*Self*

$\geqslant 1$ hasAccountableRole $\sqsubseteq$ Practice

$\top \sqsubseteq \forall$ hasAccountableRole.Role

Disjoint(hasConsultedRole, hasConsultedRole$^-$)

$\top \sqsubseteq \neg\exists$ hasConsultedRole.*Self*

$\geqslant 1$ hasConsultedRole $\sqsubseteq$ Practice

$\top \sqsubseteq \forall$ hasConsultedRole.Role

Disjoint(hasInformedRole, hasInformedRole$^-$)

$\top \sqsubseteq \neg\exists$ hasInformedRole.*Self*

$\geqslant 1$ hasInformedRole $\sqsubseteq$ Practice

$\top \sqsubseteq \forall$ hasInformedRole.Role

Disjoint(hasResponsibleRole, hasResponsibleRole$^-$)

$\top \sqsubseteq \neg\exists$ hasResponsibleRole.*Self*

$\geqslant 1$ hasResponsibleRole $\sqsubseteq$ Practice

$\top \sqsubseteq \forall$ hasResponsibleRole.Role

$\top \sqsubseteq \geqslant 1\ \text{isProcessGoalFor}$

$\mathit{Disjoint}(\text{isProcessGoalFor}, \text{isProcessGoalFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isProcessGoalFor}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isProcessGoalFor}.\text{Process}$

$\geqslant 1\ \text{isProcessGoalFor} \sqsubseteq \text{ProcessGoal}$

$\text{isProcessGoalFor} \equiv \text{hasProcessGoal}^-$

$\mathit{Disjoint}(\text{isAccountableFor}, \text{isAccountableFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isAccountableFor}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isAccountableFor}.\text{Practice}$

$\geqslant 1\ \text{isAccountableFor} \sqsubseteq \text{Role}$

$\text{isAccountableFor} \equiv \text{hasAccountableRole}^-$

$\top \sqsubseteq \geqslant 1\ \text{isActivityListFor}$

$\top \sqsubseteq \leqslant 1\ \text{isActivityListFor}$

$\mathit{Disjoint}(\text{isActivityListFor}, \text{isActivityListFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isActivityListFor}.\mathit{Self}$

$\geqslant 1\ \text{isActivityListFor} \sqsubseteq \text{ActivityList}$

$\top \sqsubseteq \forall\ \text{isActivityListFor}.\text{Practice}$

$\mathit{Disjoint}(\text{isConsultedAbout}, \text{isConsultedAbout}^-)$

$\top \sqsubseteq \neg\exists\ \text{isConsultedAbout}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isConsultedAbout}.\text{Practice}$

$\geqslant 1\ \text{isConsultedAbout} \sqsubseteq \text{Role}$

$\text{isConsultedAbout} \equiv \text{hasConsultedRole}^-$

$\mathit{Disjoint}(\text{isInformedAbout}, \text{isInformedAbout}^-)$

$\top \sqsubseteq \neg\exists\ \text{isInformedAbout}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isInformedAbout}.\text{Practice}$

$\geqslant 1\ \text{isInformedAbout} \sqsubseteq \text{Role}$

$\text{isInformedAbout} \equiv \text{hasInformedRole}^-$

$\mathit{Disjoint}(\text{isResponsibleFor}, \text{isResponsibleFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isResponsibleFor}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isResponsibleFor}.\text{Practice}$

$\geqslant 1\ \text{isResponsibleFor} \sqsubseteq \text{Role}$

$\text{isResponsibleFor} \equiv \text{hasResponsibleRole}^-$

$\mathit{Disjoint}(\text{isPrimarySupportFor}, \text{isPrimarySupportFor}^-)$

$\top \sqsubseteq \neg\exists$ isPrimarySupportFor.*Self*

$\top \sqsubseteq \forall$ isPrimarySupportFor.Goal

Disjoin(isPrimarySupportFor, isSecondarySupportFor)

Disjoint(isSecondarySupportFor, isSecondarySupportFor$^-$)

$\top \sqsubseteq \neg\exists$ isSecondarySupportFor.*Self*

$\top \sqsubseteq \forall$ isSecondarySupportFor.Goal

$\top \sqsubseteq \leqslant 1$ hasActivity

$\top \sqsubseteq \neg\exists$ hasActivity.*Self*

Disjoint(hasActivity, hasActivity$^-$)

$\top \sqsubseteq \forall$ hasActivity.Activity

$\geqslant 1$ hasActivity $\sqsubseteq$ Practice

$\top \sqsubseteq \geqslant 1$ hasActivityList

$\top \sqsubseteq \leqslant 1$ hasActivityList

Disjoint(hasActivityList, hasActivityList$^-$)

$\top \sqsubseteq \neg\exists$ hasActivityList.*Self*

$\top \sqsubseteq \forall$ hasActivityList.ActivityList

$\geqslant 1$ hasActivityList $\sqsubseteq$ Practice

hasActivityList $\equiv$ isActivityListFor$^-$

Disjoint(hasInput, hasInput$^-$)

$\top \sqsubseteq \neg\exists$ hasInput.*Self*

$\top \sqsubseteq \forall$ hasInput.WorkProduct

$\geqslant 1$ hasInput $\sqsubseteq$ Practice

$\top \sqsubseteq \leqslant 1$ hasOutput

$\top \sqsubseteq \neg\exists$ hasOutput.*Self*

Disjoint(hasOutput, hasOutput$^-$)

$\top \sqsubseteq \forall$ hasOutput.WorkProduct

$\geqslant 1$ hasOutput $\sqsubseteq$ Practice

$\top \sqsubseteq \neg\exists$ isUsedInProcess.*Self*

Disjoint(isUsedInProcess, isUsedInProcess$^-$)

$\geqslant 1$ isUsedInProcess $\sqsubseteq$ Practice

$\top \sqsubseteq \forall$ isUsedInProcess.Process

isUsedInProcess $\equiv$ hasPractice$^-$

$\top \sqsubseteq \geqslant 1$ isMetricFor

$\mathit{Disjoint}(\text{isMetricFor}, \text{isMetricFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isMetricFor}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isMetricFor}.\text{Goal}$

$\geqslant 1\ \text{isMetricFor} \sqsubseteq \text{Metric}$

$\text{isMetricFor} \equiv \text{hasMetric}^-$

$\top \sqsubseteq \geqslant 1\ \text{isPerformanceAttributeFor}$

$\mathit{Disjoint}(\text{isPerformanceAttributeFor}, \text{isPerformanceAttributeFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isPerformanceAttributeFor}.\mathit{Self}$

$\geqslant 1\ \text{isPerformanceAttributeFor} \sqsubseteq \text{PerformanceAttribute}$

$\top \sqsubseteq \forall\ \text{isPerformanceAttributeFor}.\text{ProcessCapabilityLevel}$

$\mathit{Disjoint}(\text{isPrimaryGovernanceObjectiveFor}, \text{isPrimaryGovernanceObjectiveFor}^-)$

$\top \sqsubseteq \neg\exists\ \text{isPrimaryGovernanceObjectiveFor}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{isPrimaryGovernanceObjectiveFor}.\text{EnterpriseGoal}$

$\geqslant 1\ \text{isPrimaryGovernanceObjectiveFor} \sqsubseteq \text{GovernanceObjective}$

$\top \sqsubseteq \geqslant 1\ \text{hasProcessCategory}$

$\mathit{Disjoint}(\text{hasProcessCategory}, \text{hasProcessCategory}^-)$

$\top \sqsubseteq \neg\exists\ \text{hasProcessCategory}.\mathit{Self}$

$\geqslant 1\ \text{hasProcessCategory} \sqsubseteq \text{Process}$

$\top \sqsubseteq \forall\ \text{hasProcessCategory}.\text{ProcessCategory}$

$\text{hasProcessCategory} \equiv \text{hasProcess}^-$

$\top \sqsubseteq \leqslant 1\ \text{hasPerformanceAttribute}$

$\mathit{Disjoint}(\text{hasPerformanceAttribute}, \text{hasPerformanceAttribute}^-)$

$\top \sqsubseteq \neg\exists\ \text{hasPerformanceAttribute}.\mathit{Self}$

$\top \sqsubseteq \forall\ \text{hasPerformanceAttribute}.\text{PerformanceAttribute}$

$\geqslant 1\ \text{hasPerformanceAttribute} \sqsubseteq \text{ProcessCapabilityLevel}$

$\text{hasPerformanceAttribute} \equiv \text{isPerformanceAttributeFor}^-$

$\text{hasParentProcessCategory} \circ \text{hasParentProcessCategory} \sqsubseteq \text{hasParentProcessCategory}$

$\top \sqsubseteq \forall\ \text{hasParentProcessCategory}.\text{ProcessCategory}$

$\geqslant 1\ \text{hasParentProcessCategory} \sqsubseteq \text{ProcessCategory}$

$\text{hasParentProcessCategory} \equiv \text{hasSubProcessCategory}^-$

$\top \sqsubseteq \leqslant 1\ \text{hasProcess}$

$\mathit{Disjoint}(\text{hasProcess}, \text{hasProcess}^-)$

$\top \sqsubseteq \neg\exists\ \text{hasProcess}.\mathit{Self}$

$\top \sqsubseteq \forall$ hasProcess.Process

$\geqslant 1$ hasProcess $\sqsubseteq$ ProcessCategory

hasSubProcessCategory $\circ$ hasSubProcessCategory $\sqsubseteq$ hasSubProcessCategory

$\geqslant 1$ hasSubProcessCategory $\sqsubseteq$ ProcessCategory

$\top \sqsubseteq \forall$ hasSubProcessCategory.ProcessCategory

$\top \sqsubseteq \leqslant 1$ hasProcessGoal

Disjoint(hasProcessGoal, hasProcessGoal$^-$)

$\top \sqsubseteq \neg\exists$ hasProcessGoal.*Self*

$\geqslant 1$ hasProcessGoal $\sqsubseteq$ Process

$\top \sqsubseteq \forall$ hasProcessGoal.ProcessGoal

$\top \sqsubseteq \neg\exists$ hasPractice.*Self*

Disjoint(hasPractice, hasPractice$^-$)

$\top \sqsubseteq \forall$ hasPractice.Practice

$\geqslant 1$ hasPractice $\sqsubseteq$ Process

$\top \sqsubseteq \neg\exists$ hasManagementPractice.*Self*

Disjoint(hasManagementPractice, hasManagementPractice$^-$)

$\top \sqsubseteq \forall$ hasManagementPractice.ManagementPractice

hasManagementPractice $\sqsubseteq$ hasPractice

$\geqslant 1$ hasManagementPractice $\sqsubseteq$ Process

$\top \sqsubseteq \neg\exists$ hasGovernancePractice.*Self*

Disjoint(hasGovernancePractice, hasGovernancePractice$^-$)

$\top \sqsubseteq \forall$ hasGovernancePractice.ManagementPractice

hasGovernancePractice $\sqsubseteq$ hasPractice

$\geqslant 1$ hasGovernancePractice $\sqsubseteq$ Process

Disjoint(isSecondaryGovernanceObjectiveFor, isSecondaryGovernanceObjectiveFor$^-$)

$\top \sqsubseteq \neg\exists$ isSecondaryGovernanceObjectiveFor.*Self*

$\top \sqsubseteq \forall$ isSecondaryGovernanceObjectiveFor.EnterpriseGoal

$\geqslant 1$ isSecondaryGovernanceObjectiveFor $\sqsubseteq$ GovernanceObjective

Disjoint(isAnsweredByEnterpriseGoal, isAnsweredByEnterpriseGoal$^-$)

$\top \sqsubseteq \neg\exists$ isAnsweredByEnterpriseGoal.*Self*

$\top \sqsubseteq \forall$ isAnsweredByEnterpriseGoal.EnterpriseGoal

$\geqslant 1$ isAnsweredByEnterpriseGoal $\sqsubseteq$ StakeholderNeed

Disjoint(isUsedInPractice, isUsedInPractice$^-$)

$\top \sqsubseteq \neg\exists$ isUsedInPractice.*Self*

$\geqslant 1$ isUsedInPractice $\sqsubseteq$ WorkProduct

$\top \sqsubseteq \forall$ isUsedInPractice.Practice

isUsedInPractice $\equiv$ hasInput^{-}

$\top \sqsubseteq \forall$ activityNumber.xsd:integer

$\geqslant 1$ activityNumber $\sqsubseteq$ Activity

$\top \sqsubseteq \forall$ capabilityLevel.xsd:integer

$\geqslant 1$ capabilityLevel $\sqsubseteq$ ProcessCapabilityLevel

Die Individuen der Ontologie werden hier aus Platzgründen nicht aufgeführt; es sollen hier nur exemplarisch die Axiome für den Prozess `APO01` und die Managementpraktik `APO01.01` gelistet werden. Die vollständige Ontologie kann unter http://purl.org/atextor/ontology/cobit5 bezogen werden.

Process(APO01)

$\{$hasManagementPractice(APO01, APO01.i) $\mid i \in \{01, \ldots, 08\}\}$

$\{$isPrimarySupportFor(APO01, ITGi) $\mid i \in \{1, 2, 9, 11, 15, 16, 17\}\}$

$\{$isSecondarySupportFor(APO01, ITGi) $\mid i \in \{3, 4, 7, 10, 12, 13, 14\}\}$

$\{$hasGoal(APO01, PGAPO01.i) $\mid i \in \{1, 2\}\}$

hasProcessCategory(APO01, AlignPlanAndOrganise)

ManagementPractice(APO01.01)

hasAccountableRole(APO01.01, CIO)

$\{$hasConsultedRole(APO01.01, r) $\mid r \in \{$BusinessContinuityManager, BusinessExecutives, CEO, CFO, COO, HeadArchitect, HeadDevelopment, HeadITOperations, InformationSecurityManager, ProjectManagementOffice, ServiceManager$\}\}$

$\{$hasInformedRole(APO01.01, r) $\mid r \in \{$Audit, Compliance, StrategyExecutiveCommittee$\}\}$

$\{$hasResponsibleRole(APO01.01, r) $\mid r \in \{$HeadHumanResources, HeadITAdministration$\}\}$

$\{$hasActivity(APO01.01, APO01.01.Ai) $\mid i \in \{1 \ldots 12\}\}$

hasActivityList(APO01.01, APO01.01.Activities)

$\{$hasInput(APO01.01, w) $\mid w \in \{$APO01-WP1, APO01-WP13, APO01-WP11, APO02-WP14, APO03-WP5, APO11-WP4, APO11-WP9, APO11-WP12, APO11-WP13, APO11-WP14, APO13-WP3, EDM01-WP1, EDM01-WP2, MEA01-WP3, MEA01-WP5, MEA01-WP6, MEA02-WP2, MEA02-WP3, MEA02-WP4, MEA02-WP6, MEA02-WPX6a, MEA02-WPX6b, MEA02-WPX6e, MEA02-WP10, MEA02-WP11, MEA02-WP12, MEA03-WP4$\}\}$

$\{$hasOutput(APO01.01, w) $\mid w \in \{$APO01-WP7, APO01-WP5, APO01-WP6$\}\}$

isUsedInProcess(APO01.01, APO01)