Automatisiertes, wissensbasiertes IT-Operations-Management

Dissertation
zur Erlangung des Grades
„Doktor der Naturwissenschaften“
am Fachbereich Physik, Mathematik und Informatik
der Johannes Gutenberg-Universität
in Mainz

Fabian Meyer
geb. in Erlangen

Mainz, den 3. Juli 2017

Abstract

Moderne IT-Systeme sind komplexe Konglomerate verschiedenster physischer und logischer Komponenten, die einer stetigen Dynamik unterliegen. Dies stellt das IT-Operations-Management bei der Gewährleistung einer zugesicherten Service-Qualität vor neue Herausforderungen, die mit manuellen Prozessen kaum noch zu bewerkstelligen sind. Eine Automatisierung der Prozesse ist heute jedoch nur für isolierte Teilbereiche möglich, da Informationen über unterschiedlichste Datenquellen verteilt sind und eine zusammenhängende, abstraktionsebenenübergreifende Systemgesamtsicht nur in den Köpfen der Administratoren entsteht. Die in den letzten Jahren im Kontext des Semantic Webs entstandenen Standards und Technologien bieten nun jedoch neue Möglichkeiten, Informationen domänenübergreifend zu verknüpfen und gemeinsam zu verarbeiten. Erste, in der Literatur existierende Ansätze eines ontologiebasierten, automatisierten IT-Operations-Managements sind vielversprechend, ein allgemeingültiges und auf arbiträre Anwendungsfälle adaptierbares Framework existiert jedoch nicht.

In dieser Arbeit wird untersucht, wie sich die als Standardarchitektur des automatisierten IT-Operations-Managements etablierte Monitor Analyze Plan Execute Knowledge (MAPE-K) Loop mit Semantic-Web-Technologien verknüpfen und daraus ein automatisiertes, wissensbasiertes IT-Operations-Management-Framework ableiten lässt. Dazu werden zunächst für die definierten Anforderungen und identifizierten Probleme bei der Wissensrepräsentation und -verarbeitung existierende Lösungsansätze auf ihre Eignung hin untersucht. Bereiche, für die keine adäquaten Lösungen gefunden werden konnten, werden anschließend mit eigenen Lösungen besetzt. Die ausgewählten und entwickelten Teillösungen werden dann konzeptionell zu einem generischen Management-Framework verknüpft. Abschließend werden das Framework und dessen Implementierung in einem Proof of Concept auf zwei Anwendungsfälle aus den Bereichen Storage- und Radar-Management angewandt und daran evaluiert.

Abstract (English)

Modern IT systems are complex conglomerates of diverse physical and logical components, all subject to a constant change while supporting applications over time. Delivering a guaranteed Quality of Service (QoS) for services offered by those systems presents a difficult task for IT operations management and this is barely accomplished by employing manual management processes. Yet, automation is only attainable for isolated domains, since the information required for a holistic management is distributed over several data sources, and a coherent and comprehensive view of the system under management today only originates in the administrators' minds. Standards and technologies emerging from the semantic web context are now offering new possibilities for a domain spanning data modeling and processing. First approaches which transferred those techniques to automated IT operations management were promising. Yet, no general framework, adaptable to a wide range of use cases, exists so far.

This thesis examines how the Monitor Analyze Plan Execute Knowledge (MAPE-K) loop as the standard architecture for automated IT operations management can be combined with semantic web technologies in order to create a general IT operations management framework. Existing approaches are revised as partial solutions for the defined requirements and identified problems in modeling and processing. For all problem areas lacking appropriate solutions, new methods are developed. Subsequently, a novel concept is designed, combining the selected and developed parts into a generic management framework. In a proof of concept, the framework and its implementation are applied to a radar management and a storage management use case, upon which the presented approach is evaluated.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die fortschreitende Digitalisierung aller Aspekte des täglichen Lebens macht IT-Dienste zu einem immer wichtigeren Bestandteil unserer modernen Gesellschaft. Schon heute sind sie sowohl im privaten als auch im beruflichen Umfeld nicht mehr wegzudenken. Es beginnt bei grundlegenden Dingen wie Telefonie, Internet oder E-Mail, die wir tagtäglich wie selbstverständlich einsetzen, und mündet in vollständig automatisierten Unternehmensabläufen.

Die für die Bereitstellung solcher Dienste verantwortlichen IT-Systeme sind komplexe Konglomerate verschiedenster physischer und logischer Komponenten, die miteinander verknüpft und aufeinander abgestimmt werden müssen. In solchen Komponentenstrukturen herrscht eine stetige Dynamik, die durch geplante Eingriffe und unvorhergesehene Ereignisse getrieben wird. Es werden regelmäßig neue Komponenten zum System hinzugefügt, die automatisch erkannt und eingebunden werden müssen, oder obsolete Komponenten entfernt, deren Funktionen von anderen Komponenten übernommen werden müssen. Darüber hinaus sorgen externe Einflüsse wie Hardwarefehler oder Infrastrukturausfälle zu transienten oder persistenten Ausfällen ganzer Teilsysteme. Der Trend, einzelne Dienste oder die gesamte Infrastruktur in die Cloud auszulagern, verstärkt diesen Effekt weiter. Dort werden bei Bedarf dynamisch neue Dienstinstanzen erzeugt und in das Gesamtsystem eingebunden oder nicht mehr benötigte Dienstinstanzen entfernt, sodass sich die Komponenten kontinuierlich adaptieren müssen.

Trotz dieser Dynamik gilt es, die vom System erbrachte Dienstgüte (Quality of Service, QoS) stetig auf einem im Rahmen einer Dienstvereinbarung (Service Level Agreement, SLA) zwischen Dienstleister und Dienstnutzer vereinbarten Niveau zu halten. Eine solche Vereinbarung setzt sich meist aus mehreren Einzelvereinbarungen zusammen, in denen für unterschiedliche Einflussgrößen (Service Level Objectives, SLOs) wie Verfügbarkeit oder Antwortzeit, festgelegt ist, wie sie zu messen und zu bewerten sind (Metriken). Der Bruch eines SLAs kann mit einer Vertragsstrafe belegt sein, die es für Dienstleister zu vermeiden gilt. Die bereitgestellten Dienste sind meist als serviceorientierte Architekturen realisiert, in denen höherwertige Dienste

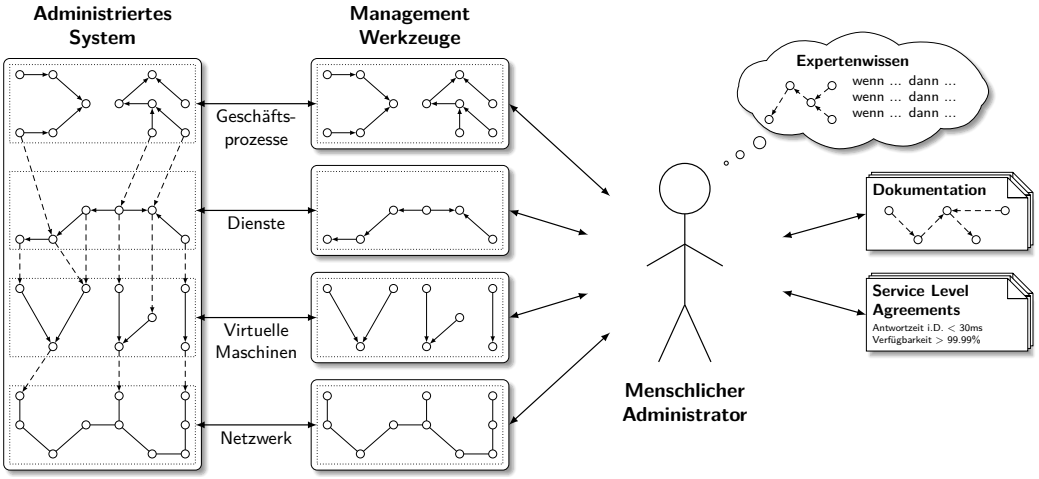


Abbildung 1.1.: Ist-Zustand beim abstraktionsebenenübergreifenden IT-Operations-Management heterogener Systeme.

aus niederwertigen Diensten komponiert werden, sodass mehrschichtige Dienstgüteabhängigkeiten entstehen. Ein Dienstleister verknüpft dabei typischerweise nicht nur eigene Dienste, sondern tritt selbst anderen Dienstleistern als Konsument gegenüber. Dienstgütevereinbarungen sind nicht nur auf extern erbrachte Dienste beschränkt, sondern werden häufig auch innerhalb eines Unternehmens zwischen der IT- und der Fachabteilung ausgehandelt.

Um einen reibungslosen IT-Betrieb zu gewährleisten und damit sicherzustellen, dass die vereinbarte Dienstgüte zu jedem Zeitpunkt erbracht wird, existiert als Teilgebiet des allgemeinen IT-Managements das sogenannte IT-Operations-Management, dessen Aufgabe es ist, sich während des laufenden Betriebs anbahnende Fehler und Dienstgüteeinbrüche durch ständige Überwachung frühzeitig zu erkennen und proaktiv gegenzusteuern. Das Operations Management wird meist von erfahrenen Administratoren durchgeführt, die ihr über die Jahre angeeignetes Expertenwissen dazu nutzen, Laufzeitinformationen unterschiedlicher Management-Werkzeuge (Kommandozeilenwerkzeuge, grafische Oberflächen, etc.) mit Dokumentationsquellen wie Konfigurationsmanagementwerkzeugen oder Excel Sheets manuell zu verknüpfen, um Rückschlüsse über den Systemzustand ziehen und Management-Entscheidungen treffen zu können (siehe Abb. 1.1). Stellt man dieses Vorgehen der Komplexität moderner IT-Systeme und der Informationsflut in Zeiten von Big Data gegenüber, wird schnell klar, dass ein Umdenken erforderlich ist.

1.2 Bedarf nach neuen Lösungen

Um weiterhin in der Lage zu sein, moderne IT-Systeme effizient zu administrieren, bedarf es neuartiger Management-Werkzeuge, die in der Lage sind, automatisiert Daten unterschiedlicher Informationsquellen zu akquirieren, miteinander zu verknüpfen, zu analysieren und eigenständig Lösungsansätze zu entwickeln. Dem Administrator wird eine aufbereitete Gesamtsicht bereitgestellt, die erkannte Probleme aufzeigt und Lösungsvorschläge unterbreitet. Zusätzlich können Richtlinien für ein selbstständiges Eingreifen des Management-Systems definiert und so ein vollständig automatisiertes Management erzielt werden.

Zwar existieren entsprechende Management-Werkzeuge, sie sind jedoch in der Regel nur für ein einzelnes Produkt, eine Klasse von gleichartigen Produkten (Datenbanken, Virtual Maschine Hypervisors, etc.), oder mehrere Produkte einzelner Hersteller (Produkte von IBM, HP, etc.) einsetzbar. In einer homogenen Systemlandschaft können solche Management-Werkzeug hinreichend sein, meist trifft man jedoch heterogene, über die Jahre gewachsene Systeme an, die aus verschiedensten, teilweise proprietären Komponenten bestehen. Dies resultiert in Insellösungen, die jeweils einen Teilbereich des Systems überwachen und steuern, jedoch kein Verständnis des Gesamtzusammenhangs haben.

Ein Gesamtbild entsteht so nur in den Köpfen der Administratoren, und komplexe Fragestellungen wie „Welcher Geschäftsprozess ist von der Wartung meiner Datenbank betroffen?“, „Ist mein System nach dem Umzug der Virtuellen Maschine weiterhin konform zu den im Service Level Agreement vereinbarten Redundanzmechanismen?“ und „Wie verhält sich die Antwortzeit meiner Anwendung, wenn ihr virtueller Speicher auf einen langsameren physikalischen Speicher umgezogen wird?“ können nur mit hohem manuellen und fehleranfälligen Aufwand beantwortet werden.

Daher bedarf es neuartiger Management-Werkzeuge, die in der Lage sind, Wissen über mehrere Domänen und Abstraktionsebenen hinweg zu verknüpfen und so eine umfassende Management-Sicht zu schaffen, die es erlaubt, Probleme global zu betrachten und automatisiert zu lösen. Neben der reinen Verknüpfung technischer Domänen (Netzwerk, Virtuelle Maschinen, Dienste, etc.) spielt die Integration nichttechnischer Domänen (Geschäftsprozesse, Unternehmensziele, Richtlinien, etc.) eine wichtige Rolle, da erst durch sie ein ganzheitliches Management möglich wird.

1.3 Ontologiebasiertes IT-Operations-Management

Als Ontologie bezeichnet man in der Informatik eine formale Konzeptualisierung einer Domäne mit fest definierter Semantik [110]. Ontologien haben im Zuge der Entwicklung des Semantic Web (siehe Kapitel 2.3) in den letzten Jahren einen Renaissance bei der Wissensmodellierung erlebt. Mit der auf Basis des Resource Description Frameworks (RDF) [33] entwickelten Web Ontology Language (OWL) [57] hat das W3C einen Modellierungsstandard geschaffen, der es erlaubt, arbiträre Domänen formal zu beschreiben und miteinander zu verknüpfen.

Eine OWL-Ontologie setzt sich aus einer Terminologie- (Terminological Box, TBox) und einer Instanzkomponente (Assertion Box, ABox) zusammen. Die TBox umfasst Konzepte, Rollen und Terminologieaxiome, die ABox die konkreten Instanzen und Instanzaxiome der beschriebenen Domäne. OWL-Ontologien folgen der Annahme einer offenen Welt (Open-world assumption), die es erlaubt, Modelle nachträglich zu erweitern und domänenübergreifend zu verknüpfen, solange die Konsistenz gewahrt wird. Die OWL-Semantik basiert auf Standardbeschreibungslogiken, sodass etablierte Validierungs- und Schlussfolgerungsalgorithmen eingesetzt werden können, deren Korrektheit und Vollständigkeit bewiesen und deren Komplexität wohlbekannt ist. Diese Eigenschaften machen OWL zu einem vielversprechenden Kandidaten als Grundlage eines IT-Operations-Management-Informationsmodells.

Bereits 2004 setzte die Gruppe um Jorge E. López de Vergara Ontologien für das IT-Operations-Management ein [121]. In ihrem Ansatz übersetzten sie die Informationsmodelle unterschiedlicher IT-Management-Standards in OWL, verknüpften sie miteinander und formulierten auf dem resultierenden Gesamtmodell abstraktionsebenenübergreifende Policies [58]. Von Technologieadaptern bereitgestellte Laufzeitinformationen wurden zyklisch auf Ontologieinstanzen abgebildet, gemeinsam verarbeitet, die definierten Policies dagegen ausgewertet und davon abgeleitete Management-Aktionen auf das überwachte System angewandt, sodass ein vollständiger, automatisierter Regelkreis umgesetzt wurde.

Auf Grundlage der Arbeiten von de Vergara et al. entstanden in den letzten Jahren verschiedene Ansätze, die Ontologien als zentrales IT-Operations-Management-Informationsmodell einsetzen. Sie konnten deutlich den resultierenden Mehrwert zeigen (siehe Abschnitt 3.1, Seite 44), ein allgemeingültiges IT-Operations-Management-Framework, mit dem sich ein automatisiertes, ontologiebasiertes Management arbiträrer Domänen umsetzen lässt, entstand bisher jedoch nicht.

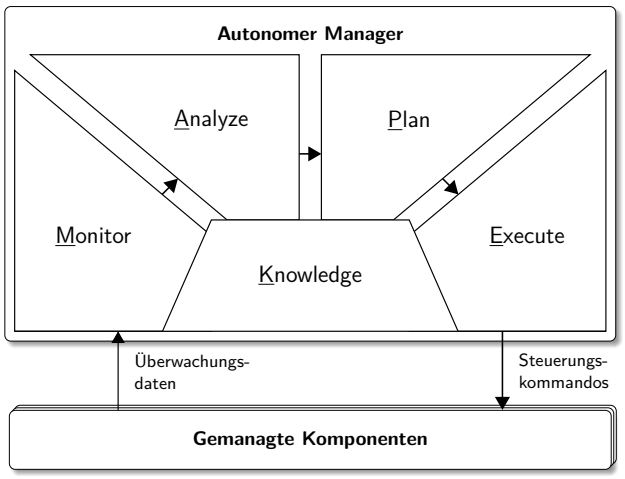


Abbildung 1.2.: Aufbau des Monitor Analyze Plan Execute Knowledge (MAPE-K)-Vorgehensmodells aus dem Bereich des Autonomic Computings.

1.4 Ziel der Arbeit

Als Standardarchitekturmodell des automatisierten IT-Operations-Managements hat sich die aus dem Autonomic Computing stammende Monitor Analyze Plan Execute Knowledge (MAPE-K) Loop [72] etabliert (siehe Abbildung 1.2). Ihre Komponenten bilden einen Regelkreis, der Überwachungsdaten von gemanagten Komponenten schrittweise analysiert, bewertet und Aktionen plant, die in Steuerungskommandos umgesetzt werden. Ein System, das diese Architektur für das IT-Operations-Management umsetzt, bezeichnet man auch als autonomen Manager und einen Durchlauf des Regelkreises als Management-Zyklus.

Kern dieses Modells bildet das Wissenssubsystem (Knowledge). Es kapselt eine subsystemübergreifende Wissensbasis, die in jedem Verarbeitungsschritt Kontextinformationen bereitstellt und die Weltsicht des autonomen Managers repräsentiert. Das Überwachungssubsystem (Monitor) bildet die Schnittstelle zu den überwachten Komponenten. Es nimmt deren Überwachungsdaten entgegen, vereinheitlicht sie und reicht sie anschließend an das Analysesubsystem (Analyze) weiter. Dieses analysiert die aufbereiteten Daten und zieht aus ihnen Rückschlüsse über den Zustand der überwachten Komponenten und des Gesamtsystems. Die Analyseergebnisse werden im Planungssubsystem (Plan) genutzt, um den Ist-Zustand des Systems mit dem Soll-Zustand zu vergleichen, etwaige Abweichungen festzustellen und autonom einen Rekonfigurationsplan zu erarbeiten, der das System in einen validen Zustand zurückführt. Das Umsetzungssubsystem (Execute) bildet das Gegenstück zum Überwachungssubsystem und ist dafür zuständig, den Regelkreis zu schließen, indem es die geplanten Management-Aktionen als Steuerungskommandos auf die Komponenten anwendet.

In dieser Arbeit soll untersucht werden, in wie weit sich das MAPE-K-Architekturmodell auf das ontologiebasierte IT-Operations-Management übertragen lässt. Dazu müssen für die unterschiedlichen Subsysteme generische Methoden und Verfahren gefunden oder entwickelt werden, die es erlauben, die Architektur mit geringem Integrationsaufwand auf beliebige Anwendungsfälle anzuwenden. Den Kern des Regelkreises soll eine Wissensbasis bilden, die als modularer Zusammenschluss mehrerer Ontologien realisiert ist. Aus ihr können die in den anderen Subsystemen angesiedelten Algorithmen Kontextinformationen beziehen oder den darin repräsentierten Systemzustand anpassen, um die folgenden Kernfunktionalitäten bereitzustellen:

- Eine kontextsensitive Vorverarbeitung, die vom gemanagten System empfangene Überwachungsdaten vor der Analyse filtert, anreichert und aggregiert (Semantic Complex Event Processing).
- Eine kontextsensitive Abbildung der vorverarbeiteten Überwachungsdaten, die beobachtbare Zustandsänderungen des gemanagten Systems auf die Wissensbasis überträgt (Semantic Lifting und Mapping).
- Analyseverfahren, die nicht beobachtbare Zustände oder implizite Qualitätsindikatoren des gemanagten Systems aus den Zustandsänderungen ableiten (Verhaltens- und Analysemodell).
- Planungsverfahren, die anhand des von der Ontologie repräsentierten Systemzustands autonom entscheiden, welche Management-Aktionen auf dem gemanagten System ausgeführt werden sollen (Steuerungsmodell).
- Ein Management-Modell, mit dem alle Management-Aspekte in einheitlicher Form beschrieben werden können.
- Ein Adaptionsverfahren, das die Verarbeitungsverfahren automatisch auf die im Management-Modell beschriebene Logik adaptiert.

Darüber hinaus müssen Lösungen für Probleme gefunden werden, die sich allgemein aus dem Einsatz von Ontologien als zentrales IT-Operations-Management-Informationsmodell ergeben:

- Die Validierungs- und Schlussfolgerungsalgorithmen der OWL unterlagerten Beschreibungslogik skalieren auf Grund der exponentiellen Laufzeit schlecht. Stellt man dies der Instanzfülle gegenüber, die bei der Repräsentation des Systemzustands über mehrere Domänen und Abstraktionsebenen hinweg entsteht, führt dies unausweichlich zu langen Management-Zyklen bis hin zu nicht verarbeitbaren Modellen.

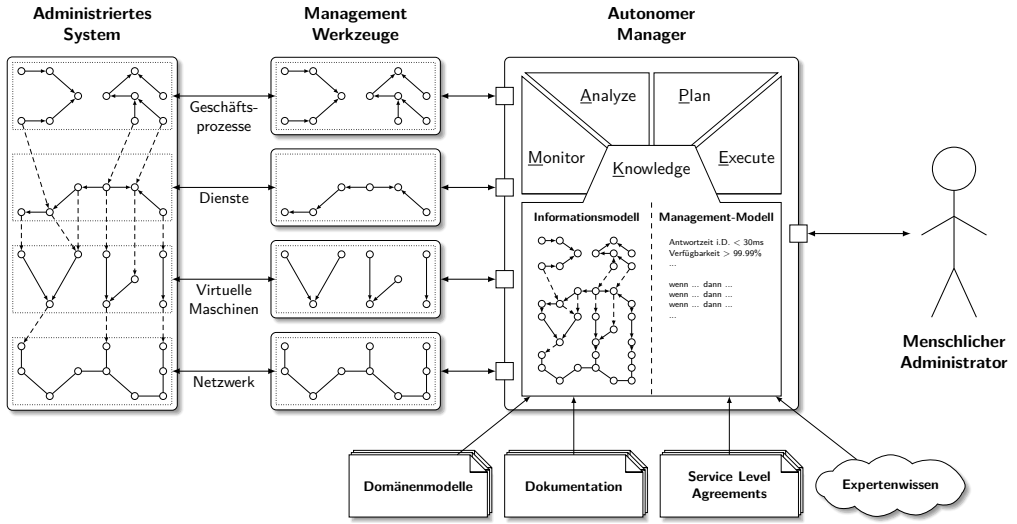


Abbildung 1.3.: Vision eines autonomes, ontologiebasierten IT-Operations-Management Frameworks.

- OWL-Ontologien haben kein designiertes Zeitkonzept. Alle darin enthaltenen Axiome beschreiben einen universellen Zustand. Dadurch können bei Analyse und Planung keine vergangenen Zustände einbezogen werden, die jedoch bei vielen Management-Fragestellungen essentiell sind.

Ziel dieser Arbeit ist es, dedizierte Lösungen für die beschriebenen Funktionen und Probleme zu finden, diese in einer generischen Management-Architektur zu verknüpfen und ein darauf basierendes Framework zu entwickeln, mit dem ein automatisiertes, wissensbasiertes IT-Operations-Management arbiträrer Anwendungsfälle umgesetzt werden kann.

Als Vision (siehe Abb. 1.3) soll so ein autonomer Manager realisiert werden können, dessen ontologiebasiertes Informationsmodell Daten unterschiedlichster Domänen und Ereignisquellen zu einer domänen- und abstraktionsebenenübergreifenden Management-Sicht verknüpft. Die im Management-Zyklus eingesetzten Verfahren werden automatisch auf das im Management-Modell formalisierte Analyse- und Handlungswissen des Administrators adaptiert und auf Basis der Monitoring-Informationen in ein automatisiertes Management des überwachten Systems umgesetzt. Der Administrator muss nur noch mit dem autonomen Manager interagieren, der ihm eine domänenübergreifende und aufbereitete Sicht des Gesamtsystems bereitstellt. Er überprüft geplante Rekonfigurationen, führt manuelle Management-Aktionen durch und passt das Informations- und Management-Modell auf neue Situationen an.

1.5 Aufbau der Arbeit

Kapitel 2 beschreibt die zum Verständnis der Arbeit wesentlichen Grundlagen des IT-Managements, von Ontologien, Semantik Web Technologien, regelbasierten Systemen und der Analyse kontinuierlicher Event-Datenströme. Eine Detailanalyse der einzelnen Problemfelder und Betrachtung existierender Lösungsansätze findet in Kapitel 3 statt. In Kapitel 4 werden geeignete Ansätze ausgewählt und für Bereiche, für die keine adäquaten Lösungen existieren, neue Methoden entwickelt. Anschließend werden die existierenden und neu entwickelten Verfahren zu einer neuartigen Management-Architektur verknüpft, ein dazu passendes Management-Framework entwickelt und ein prototypisches Laufzeitsystem umgesetzt. Das Framework wird in Kapitel 5 anhand zweier Fallstudien evaluiert und die Erfüllung der definierten Ziele überprüft. Kapitel 6 fasst die Arbeit zusammen und gibt einen Ausblick über mögliche Anschlussarbeiten.

2 Grundlagen

In diesem Kapitel werden die für das Verständnis der weiteren Arbeit vorausgesetzten Grundlagen vorgestellt. In Abschnitt 2.1 werden die Funktionen und Aufgaben des IT-Managements erläutert. Abschnitt 2.2 stellt Ontologien und deren Adaption in die Informatik vor. In Abschnitt 2.3 werden das Resource Description Framework (RDF) als graphbasiertes Ressourcenbeschreibungsformat, die darin serialisierte Web Ontologie Language (OWL) als Ontologiebeschreibungssprache und die SPARQL Query Language for RDF (SPARQL) als RDF-Abfragesprache vorgestellt. Sie bilden als Semantik Web Technologien die Grundlage des entwickelten Frameworks. Abschnitt 2.4 stellt die regelbasierte Verarbeitung und das Rule Interchange Format (RIF) vor, das als Regelaustauschformat für das Semantik Web entwickelt wurde und später die Grundlage des Management-Modells bildet. Abschließend werden in Abschnitt 2.5 Datenstromanalyse und Complex Event Processing erläutert, die bei der Vorverarbeitung von Überwachungsdaten eine zentrale Rolle spielen.

2.1 IT-Management

Unter IT-Management versteht man nach [64] alle Maßnahmen, die einen an den Unternehmenszielen ausgerichteten, effektiven und effizienten Systembetrieb sicherstellen, sodass Dienste und Anwendungen in gewünschter Güte bereitgestellt und ihre Verfügbarkeit gewährleistet werden können. Dazu gehören Personal, Verfahren, Programme und Werkzeuge, mit denen Ressourcen (Objekte des Managements) über mehrere Ebenen hinweg (Netzwerk-, System-, Informations-, Anwendungs-, Dienst- und Geschäftsebene) verwaltet werden.

Die Anforderungen an ein modernes IT-Management setzen sich laut [101] aus Alignment als Anpassung der IT an (interne oder externe) Kunden- und Organisationsbedürfnisse, Enabling als aktive Verbesserung der Unternehmensprozesse durch Einführung neuer Technologien, IT-Strategie als Entwicklung neuer Konzepte für die Zukunft der Unternehmens-IT und IT-Controlling als Bereitstellung von Kennzahlen zur Steuerung der Strategie zusammen. Sie werden durch ein Zusammenspiel der Aufgabenbereiche IT-Service-Management, IT-Governance-, Risk- und Compliance-Management, IT-Ressource-Management, und IT-Programm- und Portfolio-Management realisiert (siehe Abb. 2.1):

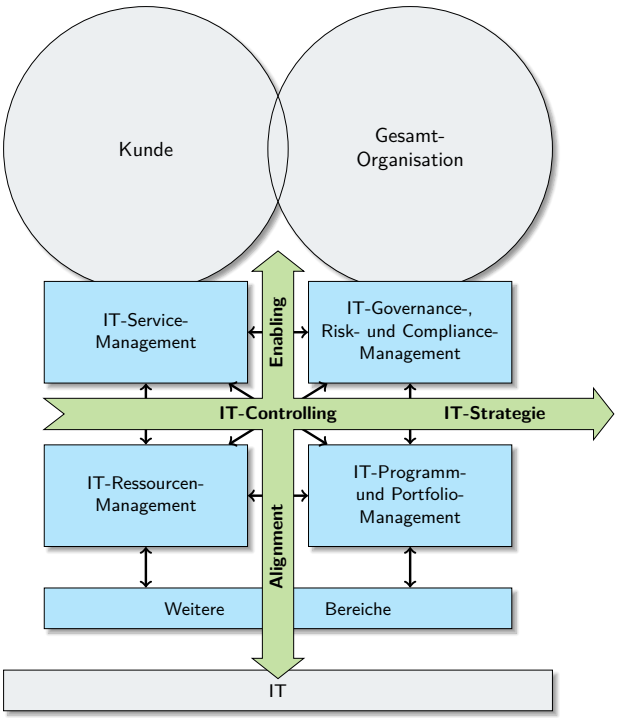


Abbildung 2.1.: Anforderungen und Aufgabenbereiche des IT-Managements nach [101].

- Das IT-Service-Management befasst sich mit Prozessen und Methoden zur bestmöglichen Unterstützung der Geschäftsprozesse eines Unternehmens durch die IT.
- IT-Governance ist in der Unternehmensführung angesiedelt und umfasst Strukturen und Prozesse, die sicherstellen, dass die IT die Unternehmensziele unterstützt.
- Das IT-Risk-Management erfasst, bewertet und behandelt Risiken des IT-Betriebs.
- Die IT-Compliance stellt sicher, dass Rahmenbedingungen gegenüber Kunden und Gesetzgebern, und unternehmensinterne Richtlinien eingehalten werden.
- Das IT-Ressourcen-Management plant den Einsatz bzw. das Outsourcing von IT-Ressourcen (Hardware, Software, Personal).
- Das IT-Programm- und Portfolio-Management bündelt IT-Leistungen in Leistungsbereiche (Anwendungsentwicklung, Netzwerkbetrieb, etc.), prüft ihre Konformität und stellt so die Verbindung zwischen Strategie und Leistungserbringung her.

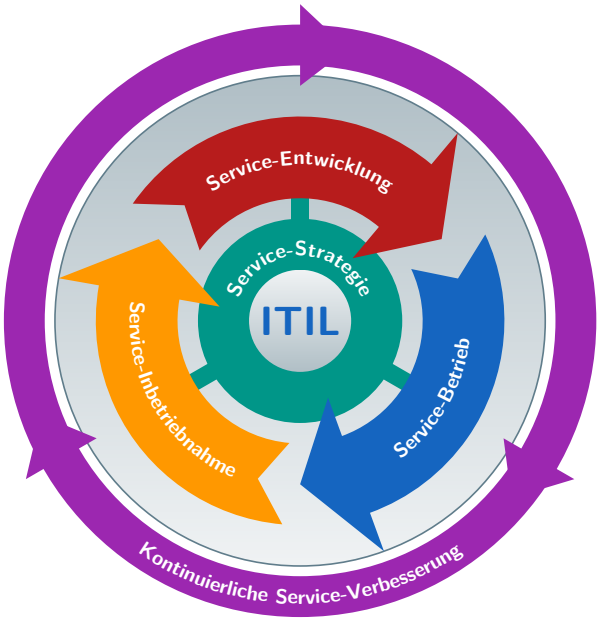


Abbildung 2.2.: ITIL-Service-Lebenszyklus nach [5].

Zwar stellen all diese Aufgabenbereiche einen wichtigen Beitrag zum IT-Management dar, auf Grund der Relevanz für die vorliegende Arbeit wird im Weiteren jedoch nur das IT-Service-Management betrachtet.

Das de-facto Standardmodell des IT-Service-Managements ist die IT Infrastructure Library (ITIL) [5]. Sie umfasst Service-Management-Prozesse, -Funktionen und -Rollen, die den Phasen Service-Strategie, Service-Entwicklung, Service-Inbetriebnahme, Service-Betrieb und kontinuierliche Service-Verbesserung des Service-Lebenszyklus (siehe Abb. 2.2) zugeteilt sind. Die Service-Strategie (Service Strategy) befasst sich mit Organisationszielen und Kundenbedürfnissen. Sie hilft IT-Organisationen, sich langfristig zu verbessern und zu entwickeln, indem dem Kunden ein Mehrwert durch neue oder modifizierte Services geboten wird. Die Service-Entwicklung (Service Design) entwirft Konzepte für die von der Strategie vorgesehenen Services und setzt sie um. Von der Service-Inbetriebnahme (Service Transition) werden die Dienste anschließend in die Laufzeitumgebung integriert. Während der Betriebsphase (Service Operation) findet ein operatives Service-Management (Service Operations Management) statt, bei dem die Service-Güte kontinuierlich überprüft und ggf. korrigierend eingegriffen wird. Die Service-Verbesserung (Continual Service Improvement) geht Hand in Hand mit der Betriebsphase und verbessert die Service-Qualität stetig, indem z. B. neue Software-Versionen eingepflegt werden. Mit dem ISO/IEC 20000 [105] existiert ein Standard, nach dem Service-Management-Systeme zertifiziert werden können.

Forschungsgegenstand dieser Arbeit ist das IT-Operations-Management, dessen Kern Fehler- und Leistungs-Management bilden. Aufgabe des Fehler-Managements (Fault Management) ist es, Fehler im laufenden Betrieb festzustellen, die Ursache einzugrenzen und zu beseitigen und so die Verfügbarkeit von Diensten sicherzustellen. Das Leistungs-Management (Performance Management) stellt sicher, dass die von einem bereitgestellten Dienst erbrachte Dienstgüte (Quality of Service, QoS) auf einem zugesicherten Leistungsniveau gehalten wird. Die Dienstgüte leitet sich aus quantifizierbaren QoS-Parametern ab, für die Nutzwerte, Mittelwerte und Grenzwerte festgelegt sind [64]. Die QoS-Parameter ergeben sich wiederum aus Metriken, die Berechnungsvorschriften und Überwachungsaktivität (z. B. Messhäufigkeit und -dauer) beschreiben.

Dienstgütevereinbarungen (Service Level Agreements, SLAs) sind Verträge, die zwischen Service-Anbietern und -Kunden geschlossen werden und eine Mindestdienstgüte (z. B. eine minimale prozentuale Verfügbarkeit oder maximale durchschnittliche Antwortzeit eines Dienstes) vereinbaren. Ein Vertragsbruch ist häufig mit einer Vertragsstrafe belegt. Ziel des Operations-Managements ist es daher, zur Laufzeit sicherzustellen, dass die vertraglich zugesicherte Leistung stetig erbracht wird. Dies umfasst zum einen die Nachweisführung dem Kunden gegenüber (Messung und Dokumentation), zum anderen eine ständige Überwachung und Rekonfiguration, um die Dienstgüte aufrecht zu erhalten.

Bei der Rekonfiguration werden zwei Arten von Management-Eingriffen unterschieden: reaktiv und proaktiv. Als reaktiv bezeichnet man korrigierende Eingriffe, die auf Grund von Abweichungen zwischen Ist- und Soll-Zustand eines Systems durchgeführt werden (z. B. im Fehlerfall), als proaktiv Eingriffe, die schon vor dem Eintritt unerwünschter Zustände durchgeführt werden und deren Vermeidung zum Ziel haben. Als Entscheidungsgrundlage des proaktiven Handelns dienen meist Vorhersagen (z. B. die wahrscheinliche I/O-Rate einer Virtuellen Maschine in zwei Stunden), oder Schwellenwertüberschreitungen (z. B. die dritte von fünf erlaubten Antwortzeitüberschreitungen pro Monat).

Die Management-Komplexität hängt dabei laut [64] stark von der Anzahl, Heterogenität und räumlichen- bzw. organisatorischen Verteilung der Objekte und Dienste ab, sodass eine spezifische und integrierte Management-Lösung für jeden Anwendungsfall benötigt würde. Das integrierte Management basiert auf dem Konzept einer globalen Datenbasis, die ebenenübergreifend alle Management-relevanten Aspekte umfasst und dafür einheitliche Schnittstellen und Oberflächen bereitstellt. Die benötigten Informationen werden über herstellerunabhängige und wohldefinierte Protokolle von den heterogenen Komponenten bezogen. Die zugrundeliegenden

Standards werden auch Management-Architekturen genannt und setzten sich aus Informations-, Kommunikations-, Funktions- und Organisationsmodell zusammen.

Das Informationsmodell legt fest, wie Ressourcen und Informationen der Domäne syntaktisch und semantisch beschrieben werden, das Kommunikationsmodell, wie Informationen zugegriffen werden, das Funktionsmodell, welche generischen Management-Funktionen existieren und das Organisationsmodell, welche Rollen und Kooperationsmodelle existieren. Management-Plattformen sind Realisierungen von Management-Architekturen und stellen Werkzeuge bereit, die in Management-Prozesse integriert werden. Nutzer der Management-Werkzeuge sind in der Regel auf der technischen Ebene Systemadministratoren, die die Systemkomponenten und erbrachten Dienste überwachen und durch Eingriffe die Service-Güte und Verfügbarkeit gewährleisten, auf der Geschäftsebene Controller, die aus den technischen Ebenen gewonnene Kennzahlen nutzen, um daraus Unternehmenskennzahlen abzuleiten.

Unter automatisiertem IT-Operations-Management versteht man die Automatisierung der manuellen Anteile des Administrationsprozesses. Dabei ersetzt eine Software-Komponente den menschlichen Administrator bei der Analyse, Entscheidungsfindung und Umsetzung. Dazu muss die Software über ein Systemverständnis und Handlungswissen verfügen, die entweder durch Formalisierung des in den Köpfen der Administratoren vorhandenen Expertenwissens (Modelle, Regeln, etc.), einen Lernprozess (z. B. Training eines neuronalen Netzes) oder die Kombination beider Ansätze gewonnen werden. Setzt die Software getroffene Management-Entscheidungen nicht eigenständig auf dem überwachten System um, sondern legt sie einem Administrator zur manuellen Prüfung vor, spricht man von einem Assistenzsystem.

Als Umsetzungsmodell von entsprechenden Laufzeitsystemen hat sich die aus dem Autonomic Computing stammende Monitor Analyze Plan Execute Knowledge (MAPE-K) Loop [72] etabliert (siehe Abb. 1.2, Seite 5). Sie sieht vor, dass in einer zentralen Wissensbasis (Knowledge) ein Modell des überwachten Systems gehalten und den anderen Subsystemen als Kontextwissen bereitgestellt wird. Das Überwachungssubsystem (Monitor) nimmt zur Laufzeit Daten des überwachten Systems entgegen und bereitet sie auf, das Analysesubsystem (Analyze) kombiniert die aufbereiteten Daten mit dem Systemmodell und leitet daraus höherwertiges Wissen (Kennzahlen, Systemzustände etc.) ab. Das Planungssubsystem (Plan) trifft dann auf Basis der Analyseergebnisse Management-Entscheidungen, die vom Umsetzungssubsystem (Execute) auf das überwachte System angewandt werden.

2.2 Ontologien

Der Ursprung des Worts Ontologie stammt aus der theoretischen Philosophie. Bereits zur Zeit von Aristoteles wurden Ontologien eingesetzt, um Entitäten und Sachverhalte zu beschreiben und zu klassifizieren. Klassifikation spielt auch in der Künstlichen Intelligenz eine wichtige Rolle und so ist es nicht verwunderlich, dass der Begriff zu Beginn der 1990er Jahre in die Informatik adaptiert wurde. Dort stellt eine Ontologie eine Art Vokabular mit wohldefinierter Semantik dar, das eine einheitliche Sicht auf eine Domäne ermöglicht, die sowohl von Menschen als auch von Computern verstanden werden kann.

Eine der gängigsten Definitionen einer Ontologie ist „eine formale, explizite Spezifikation einer gemeinsamen Konzeptualisierung“ [110]. Sie ist explizit, da die enthaltenen Entitäten und Einschränkungen klar definiert sind, formal, da sie ohne den Einsatz von Verfahren wie linguistischer Datenverarbeitung maschinenlesbar ist, und gemeinsam, da sie den Konsens einer relevanten Gruppe über die Domäne darstellt.

Eine Ontologie setzt sich aus Konzepten, Rollen, Instanzen, Axiomen und Einschränkungen zusammen. Konzepte gruppieren Instanzen anhand von Charakteristiken und werden häufig auch als Klassen bezeichnet. Rollen sind Beziehungskonzepte, über die Entitäten in Relation gesetzt werden können. Instanzen sind konkrete Objekte einer betrachteten Domäne. Axiome sind Aussagen, die Konzepte, Rollen und Instanzen in Beziehung setzen. Einschränkungen legen zusätzliche Kriterien fest, die für einen konsistenten Zustand der Ontologie erfüllt sein müssen. Als Taxonomie bezeichnet man den Teil einer Ontologie, der das statische Domänenwissen über Konzepte, Rollen und Einschränkungen umfasst. Dieses Wissen wird durch Axiome ausgedrückt, die Entitätsbeziehungen und -eigenschaften wie Konzept- und Relationshierarchien beschreiben.

Laut [110] herrscht zum großen Teil Einigkeit darüber, vier Arten von Ontologien zu unterscheiden:

- Kernontologien erfassen das Wissen, das als Grundlage über mehrere Domänen hinweg gültig ist.
- Domänenontologien erfassen das Wissen, das für einen bestimmten Typ von Domäne gültig ist.
- Anwendungsontologien erfassen das Wissen einer konkreten Anwendungsdomäne.
- Repräsentationsontologien definieren domänenunabhängige Repräsentationsentitäten (z. B. für eine objektorientiere Modellierung).

Eines der grundlegenden Entwurfsparadigmen von Ontologien ist die Modularität, die es ermöglicht, existierende Ontologien wiederzuverwenden und zu kombinieren. Um dies zu gewährleisten, sollten Ontologien laut [110] kleine Module aus stark zusammenhängenden Entitäten sein, die nur beschränkt mit anderen Modulen interagieren. Für die Verbindung von Ontologien existieren unterschiedliche Verfahren wie die Inklusion [110], bei der Ontologien vollständig miteinander vereinigt werden (Konzepte, Relationen, etc.), die Restriktion [43], die eine eingeschränkte Inklusion darstellt, und die polymorphe Verfeinerung [13], bei der inkludierte Entitäten überladen werden können.

Die Implementierung von Ontologien geschieht typischerweise mit Beschreibungslogiken [6], wodurch automatische Mechanismen zur Konsistenzprüfung und Schlussfolgerung genutzt werden können.

2.3 Semantic-Web-Technologien

2.3.1 Resource Description Framework

Das World Wide Web Consortium (W3C) hat sich im Zuge der Weiterentwicklung des Internets das Ziel gesetzt, ein einheitliches, maschinenlesbares Datenaustauschformat zur Ressourcenbeschreibung im Netz zu schaffen. Dadurch sollen Software-Komponenten in der Lage sein, untereinander Informationen auszutauschen, unterschiedliche Quellen miteinander zu verknüpfen und neues Wissen aus vorhandenem Wissen abzuleiten. Als Resultat entstand das Resource Description Framework (RDF) [33], dessen Standard 1999 erstmals veröffentlicht wurde. Es bildet die syntaktische und semantische Grundlage von OWL, das im späteren für die Domänenmodellierung eingesetzt wird.

Das Grundprinzip von RDF besteht darin, Ressourcen und deren Beziehungen zueinander durch Aussagen (Statements) zu beschreiben. Eine Aussage ist ein Tripel aus Subjekt, Prädikat und Objekt, die jeweils eine Ressource der beschriebenen Problemdomäne referenzieren. Ein RDF-Dokument besteht aus einer Menge von Aussagen und wird als Ressourcengraph interpretiert, in dem jede Aussage eine mit der Prädikatressource getypte, gerichtete Kante von der Subjekt- zur Objektressource darstellt.

Im Gegensatz zu anderen Graphbeschreibungsformaten wie der Graph Modelling Language (GraphML) [21], der Graph Exchange Language (GXL) [125], oder dem

Trivial Graph Format (TGF) definiert RDF nicht nur die Syntax, sondern auch die Semantik der darin modellierten Graphen.

In den folgenden Abschnitten werden die einzelnen Bestandteile von RDF im Detail erläutert.

Ressourcen

Die Ressourcen (Knoten) eines RDF-Graphen untergliedern sich in benannte Ressourcen, Literale und anonyme Knoten. Benannte Ressourcen stellen einen eindeutigen Referenten der modellierten Problemdomäne dar und referenzieren diesen über einen Internationalized Resource Identifier (IRI) [38]. Oft wird in RDF der Begriff IRI stellvertretend für benannte Ressourcen genutzt. Ein IRI ist ein um Unicode-Symbole erweiterter Uniform Resource Identifier (URI) [14], der sich aus einem verpflichtenden Schema- und Hierarchie- und einem optionalen Abfrage- und Fragmentteil zusammensetzt. Der Hierarchieteil besteht aus einer optionalen Zuständigkeit und einem Pfad. URIs untergliedern sich in Uniform Resource Locators (URLs) [15] und Uniform Resource Names (URNs) [88]. Ein URL beinhaltet den Ort der Ressource und stellt eine verfolgbare Verknüpfung zum Referenten dar. Ein URN ist ein ortsunabhängiger Bezeichner einer Ressource. Abbildung 2.3 zeigt anhand zweier Beispiele den Aufbau von URLs und URNs.

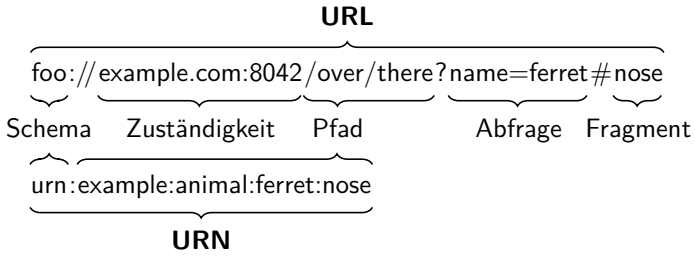


Abbildung 2.3.: Beispielhafter Aufbau der URI-Unterklassen URL und URN (aus [14]).

Literale dienen zur Repräsentation skalarer Werte im Graph. Sie haben eine lexikalische Darstellung, die sich aus einem als Zeichenkette kodierten Wert und einem über einen IRI referenzierten Datentyp zusammensetzt. Grundsätzlich werden in RDF die Datentypen der XML Schema Definition (XSD) [95] unterstützt. Ist ein Literal mit keinem Datentyp versehen (Plain Literal), wird es als Zeichenkette (`xsd:string`) interpretiert. Alternativ zum Datentyp kann ein Literal mit einem Sprachkennzeichen [96] versehen werden und wird dann als Zeichenkette in der annotierten Sprache (`rdf:langString`) interpretiert. Abbildung 2.4 zeigt anhand eines Beispiels den Aufbau von Literalen.

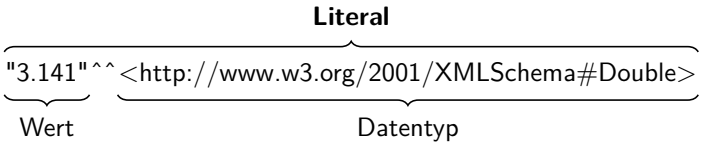


Abbildung 2.4.: Beispielhafter Aufbau eines RDF-Literals, das sich aus einem lexikalischen Wert und einem Datentyp zusammensetzt.

Anonyme Knoten sind eine Komfortfunktion von RDF, die es ermöglicht, Ressourcen zu definieren, ohne ihnen einen eindeutigen Namen zu geben. Sie sind lediglich durch ihre Beziehungen zu anderen Ressourcen des Graphen charakterisiert. Im Gegensatz zu benannten Ressourcen können anonyme Knoten nicht dokumentenübergreifend referenziert werden. In einigen RDF-Serialisierungsformaten wird ein anonymer Knoten durch einen sogenannter Blank Node Identifier identifiziert, dessen Darstellung disjunkt zu IRIs und Literalen sein muss.

Die Ressourcen eines RDF-Graphen werden nicht explizit definiert, sondern nur implizit durch deren Nutzung in Aussagen deklariert. Sie können sich stetig verändern, indem Kanten hinzugefügt oder entfernt werden.

Aussagen

Aussagen sind Tripel von Subjekt-, Prädikat- und Objektressourcen. Sie definiert, dass das Subjekt über die vom Prädikat repräsentierte Rolle in Beziehung zum Objekt steht. Die eingesetzten Ressourcen dürfen dabei nicht beliebig gewählt werden: Subjekt einer Aussage darf eine benannte Ressource oder ein anonymer Knoten, Prädikat ausschließlich eine benannte Ressource und Objekt eine benannte Ressource, ein anonymer Knoten oder ein Literal sein.

Eine Menge von Aussagen kann zu einem RDF-Dokument zusammengefasst werden, das als Ressourcengraph zu interpretieren ist, dessen Kanten von den Aussagen beschrieben werden. Mehrere RDF-Graphen, die in unterschiedlichen Dokumenten serialisiert sind, können beliebig miteinander verknüpft werden. Dies geschieht durch die Vereinigung der Aussagen aller beteiligten Dokumente. Das resultierende Dokument repräsentiert einen neuen Graph, der sich aus allen Ressourcen und Aussagen der vereinigten Dokumente zusammensetzt. Benannte Ressourcen, die in Aussagen mehrerer vereinigter Dokumente referenziert sind, bilden die Schnittpunkte, an denen die Teilgraphen miteinander verknüpft sind. Dies ermöglicht es, unterschiedliche Teilmodelle einer Domäne gemeinsam zu betrachten.

RDF-Vokabular und -Semantik

Zwar kann man mit RDF im Linked-Data-Sinn [63] Ressourcen und deren Beziehungen maschinenlesbar beschreiben und austauschen, die Bedeutung hinter dem modellierten Wissen ist für eine Maschine jedoch nicht ersichtlich. Daher wurden als Teil des RDF-Standards zwei Basisvokabulare mit zugehöriger Semantik definiert, die eine einheitliche Interpretationsbasis schaffen. Die Semantik wird von sogenannten Entailment-Regeln festgelegt.

Das RDF-Vokabular beinhaltet die grundlegenden RDF-Konzepte, die vom RDF Schema (RDFS) erweitert werden. Beide Vokabulare referenzieren sich gegenseitig und nutzen Konzepte des jeweils anderen, um sich selbst zu beschreiben. Dabei gilt, dass die Semantik des RDF-Vokabulars ohne RDFS-Semantik umgesetzt werden kann, jedoch nicht umgekehrt.

Das Grundprinzip der RDF-Semantik besteht darin, die Ressourcen des Graphen in Klassen zu kategorisieren. Eine Klasse wird selbst von einer Ressource repräsentiert und die Klassenzugehörigkeit einer Ressource über die `rdf:type`-Rolle ausgedrückt. RDF definiert eine Menge von Standardklassen und deren Semantik. Die Klasse `rdf:Property` ist die Klasse aller Rollen, der jede im Graph als Prädikat einer Aussage genutzte Ressource implizit zugeordnet ist:

$$\frac{\text{s p o .}}{\text{p rdf:type rdf:Property .}}\ \textbf{rdfD2}$$

Da Literale in RDF nicht als Subjekte von Aussagen genutzt werden dürfen, wird für jedes Literal, das als Objekt einer Aussage vorkommt, ein anonymer Knoten erzeugt und mit dem Datentyp des Literals getypt:

$$\frac{\text{s p “l”\^\^d .}}{\text{s p \_:n .} \wedge \text{\_:n rdf:type d .}}\ \textbf{rdfD1}$$

Die Klasse `rdf:XMLLiteral` ist die Klasse aller XML-Literale, für die gilt, dass jedes Objekt einer Aussage deren Element ist, gdw. es ein wohldefiniertes XML-Literal ist, also einen gültigen XSD-Datentyp referenziert und eine valide lexikalische Darstellung des Datentyps verkörpert.

Um in RDF ungeordnete Mengen, geordnete Mengen und Alternativen zu modellieren, gibt es die Klassen `rdf:Bag`, `rdf:Seq` und `rdf:Alt`, denen über die Rolle `rdf:_i` für alle $i \in \mathbb{N}$ jeweils das i-te Element zugeordnet werden kann. Für ungeordnete Mengen wird die Reihenfolge der Elemente bei der Betrachtung vernachlässigt, bei

Alternativen gilt per Konvention das erste Element als Standardauswahl. Eine terminierte Elementliste wird als Instanzen von `rdf:List` modelliert. Dem Prinzip einer einfach verketteten Liste folgend, zeigt jeder Listeneintrag über die `rdf:first`-Rolle auf das gekapselte Element und über die `rdf:rest`-Rolle auf den nachfolgenden Eintrag bzw. am Listenende auf `rdf:nil`.

Da Aussagen selbst keine Ressourcen sind, sondern nur Kanten des Graphen repräsentieren, können sie selbst nicht in Aussagen referenziert werden. Um dennoch Aussagen über Aussagen machen zu können, existiert in RDF die sogenannte Versachlichung (Reification), bei der Aussagen als Instanzen der Klasse `rdf:Statement` als Knoten des Graphen materialisiert und über die `rdf:subject`-, `rdf:predicate`- und `rdf:object`-Rollen mit den entsprechenden Ressourcen verknüpft werden. Versachlichte Aussagen unterliegen nicht der Semantik regulärer Aussagen, sondern werden als einfache Ressourcen des Graphen interpretiert.

RDFS-Vokabular und -Semantik

Durch das RDF-Vokabular lassen sich zwar einfache Strukturen beschreiben, eine echte Modellierung der Domänenterminologie ist jedoch erst mit dem Resource Description Framework Schema (RDFS) [23] möglich. Das RDFS-Vokabular definiert Klassen und Rollen, die zu einer systematischeren Kategorisierung von Ressourcen genutzt werden können. Die Klasse `rdfs:Resource` stellt die Oberklasse aller Ressourcen dar, der implizit jeder Knoten des Graphen zugeordnet wird, der als Subjekt oder Objekt einer Aussage verwendet wird:

$$\frac{\text{s p o .}}{\text{s rdf:type rdf:Resource .} \wedge \text{o rdf:type rdf:Resource .}} \textbf{ rdfs4}$$

Analog dazu ist `rdfs:Class` die Klasse aller Klassen. Über die Relation `rdfs:subClassOf` können Klassenhierarchien gebildet werden. Jede Klasse ist Unterklasse von `rdfs:Resource` und sich selbst (reflexiv).

$$\frac{\text{c rdf:type rdfs:Class .}}{\text{c rdfs:subClassOf rdfs:Resource .}} \textbf{ rdfs8}$$

Die Unterklassenbeziehung ist transitiv und jede Instanz einer Unterklasse auch implizit Instanz derer Oberklassen.

$$\frac{\text{x rdfs:subClassOf y .} \qquad \text{y rdfs:subClassOf z .}}{\text{x rdfs:subClassOf z .}} \textbf{ rdfs11}$$

$$\frac{\text{x rdfs:subClassOf y .} \qquad \text{z rdf:type x .}}{\text{z rdf:type y .}}\ \textbf{rdfs9}$$

Analog dazu können über die transitive und reflexive Rolle `rdfs:subPropertyOf` Rollenhierarchien gebildet werden. Beziehungen über Unterrollen werden implizit auf deren Oberrollen reproduziert.

$$\frac{\text{x rdfs:subPropertyOf y .} \qquad \text{y rdfs:subPropertyOf z .}}{\text{x rdfs:subPropertyOf z .}}\ \textbf{rdfs5}$$

$$\frac{\text{p rdfs:subPropertyOf y .} \qquad \text{s p o .}}{\text{s y o .}}\ \textbf{rdfs7}$$

Über die `rdfs:domain`- und `rdfs:range`-Rollen können Definitions- und Wertebereich einer Rolle festgelegt werden. Dabei gilt, dass das Subjekt einer Aussage implizit Instanz der Definitionsbereichsklasse und das Objekt Instanz der Wertebereichsklasse ist.

$$\frac{\text{p rdfs:domain d .} \qquad \text{s p o .}}{\text{s rdf:type d .}}\ \textbf{rdfs2}$$

$$\frac{\text{p rdfs:range r .} \qquad \text{s p o .}}{\text{o rdf:type r .}}\ \textbf{rdfs3}$$

Weitere von RDFS definierte Klassen sind `rdfs:Literal`, `rdfs:Datatype`, `rdfs:Container` und `rdfs:ContainerMembershipProperty`. `rdfs:Literal` ist die Klasse aller Literale und somit die Oberklasse aller Datentypen.

$$\frac{\text{d rdf:type rdfs:Datatype .}}{\text{d rdfs:subClassOf rdfs:Literal .}}\ \textbf{rdfs13}$$

Die Klasse `rdfs:Container` ist die Oberklasse von `rdf:Bag`, `rdf:Seq` und `rdf:Alt`, die Rolle `rdfs:member` die Oberrolle der `rdfs:ContainerMembershipProperty`-Rollen `rdf:_i` $i \in \mathbb{N}$.

$$\frac{\text{p rdf:type rdfs:ContainerMembershipProperty .}}{\text{p rdfs:subPropertyOf rdfs:member .}}\ \textbf{rdfs12}$$

Der Standard definiert neben den Ableitungsregeln mehrere axiomatische Tripel, durch die sich das Vokabular selbst beschreibt, indem es beispielsweise den Definitions- und Wertebereich der eingeführten Rollen festlegt. Diese Tripel bilden die Grundlage eines korrekten Reasonings und sind Teil der RDF- und RDFS-Dokumente.

Formate

RDF-Dokumente können in unterschiedlicher Syntax serialisiert werden. Am verbreitetsten sind RDF-XML [50] und die Terse RDF Tripel Language (Turtle) [12]. RDF-XML ist besonders gut maschinenlesbar und wird deshalb meist als Austauschformat verwendet, Turtle besonders gut menschenlesbar und wird daher meist als Präsentationsyntax genutzt. In dieser Arbeit wird zur RDF-Darstellung Turtle genutzt, dessen Syntax im Folgenden kurz erläutert wird. Es erlaubt zur verkürzten Schreibweise die Definition eines Basis-IRI und mehrerer Präfixe.

```
1  @base          <http://example.org> .
2  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
```

Aussagen werden durch einen Punkt getrennt, vollqualifizierte oder zum Basis-IRI relative IRIs in spitze Klammern gestellt, gepräfixte IRIs werden nicht geklammert, und ganzzahlige und boolesche Literale können ohne lexikalische Schreibweise genutzt werden.

```
1  <#Bob> foaf:age   42 .
2  <#Bob> foaf:name  "Bob"^^xsd:string .
3  <#Bob> foaf:knows <http://example.org#Alice> .
```

Prädikatlisten erlauben es, mehrere Aussagen desselben Subjekts durch Semikola getrennt zu gruppieren. Analog bieten Objektlisten die Möglichkeit, mehrere Aussagen desselben Subjekts und Prädikats durch Kommata getrennt zu gruppieren.

```
1  <#Alice> foaf:age  37 ;
2           foaf:name "Alice" , "Ally" .
```

In eckigen Klammern können anonyme Knoten durch eine Prädikatliste beschrieben werden.

```
1  <#Alice> foaf:knows [ foaf:age  41 ;
2                        foaf:name "Carl" ] .
```

2.3.2 Web Ontology Language

Mit RDF und RDFS lassen sich einfache Terminologien und Ressourcenstrukturen auf einem Niveau von Entity Relation (ER)-Diagrammen modellieren, für eine umfassende, ontologische Beschreibung fehlen jedoch wichtige Sprachekonzepte. Mehrere Forschungsgruppen begannen daher Ende der 90er Jahre, durch theoretische Grundlagen untermauerte Beschreibungslogikkonzepte auf RDF zu übertragen und so die Modellierung ausdrucksmächtigerer Terminologien zu ermöglichen.

Die DARPA Agent Markup Language (DAML) [66] wurde 2000 als RDF-basierte Auszeichnungssprache für Agenten entwickelt. Ihr Vokabular umfasst Sprachkonstrukte um Klassendisjunktion, Rollencharakteristiken, Individuenäquivalenz und einfache Klasseneinschränkungen auszudrücken. Parallel dazu wurde mit dem Ontology Inference Layer (OIL) [45] ein Modell entwickelt, mit dem ontologische Inferenzmechanismen beschrieben werden können. Die starke Überschneidung beider Ansätze führte dazu, dass sie 2001 unter dem Dach des W3C zum Standard DAML+OIL [68] zusammengeführt wurden.

Um die Entwicklungen einer standardisierten Ontologiesprache für das Semantic Web weiter voran zu treiben, wurde Ende 2001 die W3C Web Ontology Working Group (WebOnt) gegründet. Sie veröffentlichte 2004 erstmals die stark von DAML+OIL geprägt Web Ontology Language (OWL) [82]. OWL setzt auf RDFS auf und erweitert dessen Vokabular und Semantik um Konzepte, die die Ausdrucksmächtigkeit auf das Niveau einer $\mathcal{SHOIN}$-Beschreibungslogik [6] heben. Die Erweiterungen des 2006 veröffentlichten OWL 1.1 [94] hoben die Ausdrucksmächtigkeit auf $\mathcal{SROIQ}$ [6], es kam jedoch nie zur abschließenden Standardisierung. Ende 2012 wurde mit OWL 2 [57] der aktuelle OWL-Standard veröffentlicht, der auf dem Niveau einer $\mathcal{SROIQ}(\mathcal{D})$-Beschreibungslogik [6] angesiedelt ist. Die entscheidendsten Neuerungen sind die umfangreiche Unterstützung von Datentypen und die Einführung von Sprachprofilen.

Aufbau

Ein Grundbaustein von OWL-Ontologien sind Entitäten, die Konzepte und Objekte der Problemdomäne darstellen und sich in die sechs Kategorien Klassen, Individuen, Datentypen, Objekt-, Daten-, Ontologie- und Annotationsrollen untergliedern. Alle Entitäten sind über einen IRI eindeutig identifizierbar und die Kategorien untereinander disjunkt. Bei den in einer Ontologie verwendeten Entitäten spricht man auch von der Signatur der Ontologie.

Anders als in RDF gruppieren OWL-Klassen keine beliebigen Ressourcen, sondern ausschließlich Individuen, die konkrete Domänenobjekte repräsentieren. Alle OWL-Klassen sind Unterklassen von `owl:Thing`, das dem Top Concept ($\top$) in der Beschreibungslogik entspricht.

Rollen werden in OWL in Objekt-, Daten- und Annotationsrollen untergliedert. Eine Objektrolle entspricht einer abstrakten Rolle in der Beschreibungslogik und setzt zwei Individuen in Beziehung. Eine Datenrolle entspricht einer konkreten Rolle in der Beschreibungslogik und setzt Individuen mit Literalen in Beziehung. Annotationsrollen zeichnen Entitäten und Axiome mit Metainformationen aus.

Eine OWL-Ontologie setzt sich aus Axiomen zusammen, die atomare Aussagen über die Entitäten der Problemdomäne darstellen. Anders als in RDF wird in OWL klar zwischen Taxonomie- und Instanzebene unterschieden. Zwar befinden sich alle Axiome physikalisch gemeinsam in einem Dokument, logisch werden sie jedoch in drei Gruppen unterteilt. Die Terminological Box (TBox) ist die terminologische Komponente der Ontologie und umfasst alle Axiome über die Klassen der Domäne. Die Role Box (RBox) ist die Rollenkomponente der Ontologie und umfasst alle Axiome über die Rollen der Domäne. In der Assertion Box (ABox) befinden sich alle Axiome über die Individuen der Domäne. Axiome werden als Kombinationen mehrerer RDF-Statements serialisiert.

Im Folgenden werden die unterschiedlichen Axioms- und Ausdruckstypen von OWL vorgestellt.

Klassen

Der Begriff Klasse wird in OWL auch als Synonym für Klassenbeschreibung (Class Description) genutzt. Die einfachste Form der Klassenbeschreibung ist die benannte Klasse, die ein direktes Domänenkonzept darstellt und eindeutig über einen IRI identifizier- und referenzierbar ist. Deutlich mehr Modellierungstiefe und Ausdrucksmächtigkeit bieten die unterschiedlichen Formen komplexer Klassen.

Aufzählungen sind abgeschlossene Klassen, die ausschließlich eine Menge von explizit benannten Instanzen umfassen. Universal- und existenzquantifizierte Klassen beschränken die Zielmenge einer Rolle. Ein Individuum ist genau dann Instanz der universalquantifizierten Klasse, wenn es über die Rolle ausschließlich mit Elementen der Zielmenge in Beziehung steht, und genau dann Instanz der existenzquantifizierten Klassen, wenn es über die Rolle mit mindestens einem Element der Zielmenge

Typ	OWL-Konstruktor	DL-Äquivalent
Benannte Klasse	$\mathrm{Class}(C)$	C
Aufzählungen	$\mathrm{OneOf}(x_1 \dots x_n)$	$\{x_1\} \sqcap \dots \sqcap \{x_n\}$
Universalquantifizierte Klasse	$\mathrm{AllValuesFrom}(R\ C)$	$\forall R.C$
Existenzquantifizierte Klasse	$\mathrm{SomeValuesFrom}(R\ C)$	$\exists R.C$
Wertquantifizierte Klasse	$\mathrm{HasValue}(Rx)$	$\exists R.\{x\}$
Min. Kard. Klasse	$\mathrm{MinCardinality}(n\ R\ C)$	$\geq nR.C$
Max. Kard. Klasse	$\mathrm{MaxCardinality}(n\ R\ C)$	$\leq nR.C$
Exakt Kard. Klasse	$\mathrm{ExactCardinality}(n\ R\ C)$	$= nR.C$
Reflexive Klasse	$\mathrm{HasSelf}(R)$	$\exists R.Self$
Klassenschnitt	$\mathrm{IntersectionOf}(C_1 \dots C_n)$	$C_1 \sqcap \dots \sqcap C_n$
Klassenvereinigungen	$\mathrm{UnionOf}(C_1 \dots C_n)$	$C_1 \sqcup \dots \sqcup C_n$
Klassenkomplement	$\mathrm{ComplementOf}(C)$	$\neg C$
Unterklassen	$\mathrm{SubClassOf}(C_1\ C_2)$	$C_1 \sqsubseteq C_2$
Klassenäquivalenz	$\mathrm{EquivalentClasses}(C_1 \dots C_n)$	$C_1 \equiv \dots \equiv C_n$
Klassendisjunktheit	$\mathrm{DisjointClasses}(C_1 \dots C_n)$	$C_i \sqcap C_j \sqsubseteq \bot, i \neq j$

Tabelle 2.1.: Abbildung von OWL-Klassenbeschreibungen und -Axiomen auf äquivalente Beschreibungslogikkonstrukte nach [69].

in Beziehung steht. Wertquantifizierte Klassen sind Spezialisierungen existenzquantifizierter Klassen, bei denen die Zielmenge eine einelementige Aufzählung ist. Kardinalitätseingeschränkte Klassen verallgemeinern existenzquantifizierte Klassen, indem für die Rollen-Zielmenge-Kombination eine minimale, maximale oder exakte Mächtigkeit definiert werden kann. Reflexive Klassen legen eine Rolle fest, für die gilt, dass jedes Individuum Element der Klasse ist, gdw. es darüber zu sich selbst in Beziehung steht. Klassenbeschreibungen können über unterschiedliche Operatoren kombiniert werden, um deren Schnittmenge, Vereinigung oder Komplement zu bilden.

Die vorgestellten Klassenbeschreibungen sind keine eigenständigen Axiome, sondern können nur als Teil von Klassenaxiomen oder Instanzaxiomen in der Ontologie genutzt werden. Klassenaxiome sind das Unterklassenaxiom, mit dem Klassenhierarchien gebildet werden können, das Äquivalenzaxiom, das die Äquivalenz von Klassen ausdrückt, und das Disjunktionsaxiom, das die Disjunktheit von Klassen aussagt.

Tabelle 2.1 zeigt, wie die OWL-Klassenbeschreibungen und -Axiome auf Beschreibungslogikkonzepte abgebildet werden.

Rollen

Der Begriff Rolle wird in OWL auch als Synonym für Rollenbeschreibung (Property Description) genutzt. Im Gegensatz zu Klassenbeschreibungen sind die Beschreibungsmöglichkeiten komplexer Rollen jedoch sehr beschränkt. Eine Objektrollenbeschreibung kann eine benannte Objektrolle, die Inverse einer benannten Objektrolle, oder eine Objektrollenkette sein. Eine Datenrollenbeschreibung kann ausschließlich eine benannte Datenrolle sein.

Wie auch bei Klassen, können Rollenbeschreibungen ausschließlich in Axiomen verwendet werden. Rollenaxiome sind das Unterrollenaxiom, das Rollen hierarchisch anordnet, das Äquivalenzrollenaxiom, das die Äquivalenz von Rollen ausdrückt, das Disjunktionsrollenaxiom, das die Disjunktheit von Rollen beschreibt, und das Inversenaxiom, das eine symmetrische Inversität zwischen Rollen ausdrückt. Für jede Rolle kann ein globaler Definitions- und Wertebereich festgelegt werden. Rollen können darüber hinaus mit verschiedenen Charakteristiken ausgestattet werden, indem sie als Unterrollen spezieller Rollen des OWL-Vokabulars ausgewiesen werden. Objekt- und Datenrollen können als funktional (rechtseindeutig) und inversfunktional (linkseindeutig, injektiv) deklariert werden. Objektrollen können zudem reflexiv, irreflexiv, symmetrisch, asymmetrisch oder transitiv sein. Objektrollenketten dürfen ausschließlich als Unterrollen in Unterrollenaxiomen genutzt werden und unterliegen weiteren Einschränkungen, die im Detail aus [67] entnommen werden können.

Tabelle 2.2 zeigt, wie die OWL-Rollenbeschreibungen und -Axiome auf Beschreibungslogikkonzepte abgebildet werden.

Individuen

Als Individuen werden in OWL alle Ressourcen bezeichnet, die Instanz der Klasse `owl:Thing` sind. Sie können anonym oder benannt sein und in ABox-Axiomen verwendet werden. Klassenzugehörigkeitsaxiome weisen Individuen explizit Klassen zu, Rollenzuweisungsaxiome setzen zwei Domänenobjekte über eine Rolle in Beziehung, das negative Rollenzuweisungsaxiom bildet dessen Gegenstück und schließt explizit aus, dass jemals über die Rolle eine Beziehung zwischen zwei Objekten existieren kann. Über das Individuenäquivalenzaxiom kann ausgesagt werden, dass zwei Individuen dieselbe Identität haben, also dasselbe Domänenobjekt repräsentieren. Analog dazu kann mit einer Individuenambivalenz explizit ausgeschlossen werden, dass zwei Individuen identisch sind.

Typ	OWL-Konstruktor	DL-Äquivalent
Objektrolle	$\mathrm{Property}(R)$	R
Datenrolle	$\mathrm{DataProperty}(R)$	R
Inverse Rolle	$\mathrm{InverseOf}(R)$	R^-
Rollenkette	$\mathrm{PropertyChain}(R_1 \dots R_n)$	$R_1 \circ \dots \circ R_n$
Unterrollen	$\mathrm{SubPropertyOf}(R_1\ R_2)$	$R_1 \sqsubseteq R_n$
Rollenäquivalenz	$\mathrm{EquivalentProperties}(R_1 \dots R_n)$	$R_1 \equiv \dots \equiv R_n$
Rollendisjunktheit	$\mathrm{DisjointProperties}(R_1 \dots R_n)$	$\bot \sqsubseteq \exists R_i \sqcap \exists R_j, i \neq j$
Inverse Rollen	$\mathrm{InverseProperties}(R_1 \dots R_n)$	$R_1 \equiv R_2^{-1}$
Definitionbereich	$\mathrm{PropertyDomain}(R \dots C)$	$\top \sqsubseteq \forall R^-.C$
Wertebereich	$\mathrm{PropertyRange}(R \dots C)$	$\top \sqsubseteq \forall R.C$
Funktionale Rolle	$\mathrm{FunctionalProperty}(R)$	$\top \sqsubseteq \leq 1R$
Invers-funktionale Rolle	$\mathrm{InverseFunctionalProperty}(R)$	$\top \sqsubseteq \leq 1R^-$
Reflexive Rolle	$\mathrm{ReflexiveProperty}(R)$	$\top \sqsubseteq \exists R.Self$
Irreflexive Rolle	$\mathrm{IrreflexiveProperty}(R)$	$\bot \sqsubseteq \exists R.Self$
Symmetrische Rolle	$\mathrm{SymmetricProperty}(R)$	$R \equiv R^-$
Assymetrisch Rolle	$\mathrm{AsymmetricProperty}(R)$	$\bot \sqsubseteq \exists R \sqcap \exists R^-$
Transitive Rolle	$\mathrm{TransitiveProperty}(R)$	$R \sqsubseteq R \circ R$

Tabelle 2.2.: Abbildung von OWL-Rollenbeschreibungen und -Axiomen auf die äquivalenten Beschreibungslogikkonstrukte nach [69].

Axiom	OWL-Konstruktor	DL-Äquivalent
Klassenzugehörigkeit	$\mathrm{ClassAssertion}(C\ x)$	$x \in C$
Rollenzuweisung	$\mathrm{PropertyAssertion}(R\ x\ y)$	$\langle x, y \rangle \in R$
Negative Rollenzuweisung	$\mathrm{NegativePropertyAssertion}(C\ x)$	$\langle x, y \rangle \notin R$
Individuenäquivalenz	$\mathrm{SameIndividual}(x_1 \dots x_n)$	$\{x_1\} \equiv \dots \equiv \{x_n\}$
Individuenambivalenz	$\mathrm{DifferentIndividuals}(x_1 \dots x_n)$	$\{x_i\} \sqcap \{x_j\} \sqsubseteq \bot, i \neq j$

Tabelle 2.3.: Abbildung von OWL-Individuenaxiomen auf die äquivalenten Beschreibungslogikkonstrukte nach [69].

Neben der expliziten Definition identischer Individuen gibt es in OWL die Möglichkeit, eine implizite Identität durch Schlüsselmerkmale auszudrücken. Dazu werden für eine Klasse Rollen festgelegt, deren Vereinigung einen Klassenschlüssel darstellt. Sind zwei Individuen Instanz der ausgezeichneten Klasse und alle ihre Schlüsselmerkmale referenzieren dasselbe Objekt, sind sie identisch.

Tabelle 2.3 zeigt, wie die OWL-Individuenaxiome auf Beschreibungslogikkonzepte abgebildet werden.

Datentypen

Grundsätzlich werden in OWL alle gängigen XSD-Datentypen unterstützt. Darüber hinaus können über Datenbereiche (Data Ranges) neue Datentypen gebildet werden.

Analog zu Aufzählungsklassen können Aufzählungsdatenbereiche als Aufzählung einer endlichen Menge von Literalen definiert werden. Zudem können mit den im XML-Schema definierten Einschränkungsfacetten Basisdatentypen auf einen Teilwertebereich beschränkt werden. So können z. B. obere und untere Grenzen für Zahlenwerte oder Muster von Zeichenketten festgelegt werden. Datenbereiche können über Operatoren vereinigt, geschnitten oder negiert werden. Sie können entweder direkt in Klassen- und Rollenaxiomen verwendet oder über ein Definitionsaxiom als neue Datentypen definiert werden.

Annotationen

Über Annotationen können Ontologieentitäten und Axiome mit Metainformationen ausgestatten werden, ohne die Semantik zu beeinflussen. Dazu können Annotationsrollen definiert, hierarchisch angeordnet, mit einem Definitions- und Wertebereich versehen und über ein Zuweisungsaxiom instanziiert werden.

OWL bietet Standardannotationen, über die für eine Ontologie eine Version und ein direkter Vorgänger definiert werden können. Zudem kann eine Abwärtskompatibilität oder Inkompatibilität zu früheren Versionen angegeben werden. Über die Import-Annotation können andere Ontologien über ihren IRI importiert und so deren Entitäten in der importierenden Ontologie nutzbar gemacht werden. Zudem können aus RDF geerbte Annotationen genutzt werden, um Entitäten und Axiome zu kommentieren, mit einem Bezeichner zu versehen oder Querreferenzen zu anderen Entitäten zu definieren.

Untersprachen

OWL definiert verschiedene Untersprachen, die die Ausdrucksmächtigkeit einschränken, im Gegenzug jedoch Zusicherungen zur Entscheidbarkeit und Komplexität des Reasoning machen.

OWL-2-Full beinhaltet den gesamten Sprachumfang von OWL und erlaubt es, RDF-Konstrukte uneingeschränkt zu nutzen. Dies führt dazu, dass die Klassen `owl:Class` und `rdfs:Class`, `owl:Property` und `rdfs:Property`, und `owl:Thing` und `rdfs:Resource` äquivalent werden. Daraus folgt, dass jede Ressource des Graphen ein OWL-Individuum ist und die unterschiedlichen Sprachkonzepte nicht disjunkt sind. Dies führt wiederum zur Unentscheidbarkeit von OWL-Full [71], was die Sprache zwar für theoretische Untersuchungen und reine Metamodellierung interessant, für den praktischen Einsatz jedoch ungeeignet macht.

OWL-2-DL ist eine Sprachuntermenge von OWL-2-Full, deren Einschränkungen entscheidbare Inferenzmechanismen der Beschreibungslogik anwendbar machen. Die grundlegendste Einschränkungen ist die paarweise Disjunktion von Klassen, Datentypen, Datenrollen, Objektrollen, Annotationsrollen, Individuen und Literalen. Lediglich Individuen und Klassen dürfen sich überschneiden, um ein sogenanntes Punning zu ermöglichen.

Durch die Einschränkung, dass Datenrollen keine Inverse haben dürfen und weder invers-funktional, symmetrisch oder transitiv sein können, wird die Disjunktion von Individuen und Literalen sichergestellt. Weder transitive Rollen noch deren Inverse dürfen in Kardinalitätseinschränkungen verwendet werden. Alle Axiome einer Ontologie müssen wohlgeformt sein, was bedeutet, dass alle referenzierten Klassen und Rollen explizit als solche ausgezeichnet sein müssen. Als letzte Einschränkung gilt, dass eine Äquivalenz oder Ambivalenz nur zwischen benannten Individuen bestehen darf.

Die resultierende Sprache lässt sich auf eine $\mathcal{SROIQ}(\mathcal{D})$-Beschreibungslogik abbilden, für die sich der analytische Tableau [8] als effizienter Reasoning-Algorithmus herausgestellt hat. Er ermöglicht eine Konsistenz- und Konzepterfüllbarkeitsprüfung in NEXPTIME-vollständig [103, 79].

In OWL 1 existiert zusätzlich die Sprachklasse OWL-Lite. Sie ist eine weiter eingeschränkte Untermenge von OWL-DL, in der Aufzählungen, Vereinigungen, Komplementen, Zusicherungen und Disjunktionen untersagt sind. Bei einer Klassenäquivalenz- oder Unterklassenrolle muss das Subjekt eine benannte Klasse und das Objekt eine benannte Klassen oder eine Einschränkung sein. Schnittmengen dürfen nur mit benannten Klassen und Einschränkungen gebildet werden und müssen sich aus mehr als einer Klasse zusammensetzen. Existenz- und universalquantifizierte Restriktionen dürfen als Zielmenge nur benannte Klassen oder Datentypen haben. Ressourcen können nur Instanzen von Klassen oder Restriktionen sein, der Definitionsbereich einer Rolle darf nur eine benannte Klasse und der Wertebereich nur eine benannte Klasse oder ein Datentyp sein. OWL-Lite ist eine $\mathcal{SHIF}$-Beschreibungslogik, für die Konsistenz- und Konzepterfüllbarkeitsprüfung ExpTime-vollständig sind [119].

Profile

Die OWL-Sprachuntermengen sind auf Grund ihrer unterschiedlichen Entscheidbarkeit und Komplexitätsklassen zwar aus einem wissenschaftlichen Aspekt heraus

interessant, bei der realen Anwendung von Ontologien spielen jedoch andere Kriterien eine Rolle. Daher wurden für OWL 2 drei Sprachprofile definiert, die auf bestimmte Anwendungsfelder gemünzt sind.

Das EL-Profil basiert auf $\mathcal{EL}^{++}$-Beschreibungslogiken [7] und eignet sich besonders für die Modellierung und Untersuchung von Ontologien mit vielen Klassen und Rollen. Beim Entwurf des Profils wurde die Ausdrucksmächtigkeit verschiedener Ontologien dieser Kategorie untersucht und eine entscheidbare OWL 2 Untermenge gebildet, die in der Lage ist, die Standardinferenzaufgaben in polynomialer Zeit umsetzen zu können.

OWL EL untersagt die Nutzung von universalquantifizierten Klassen, kardinalitätseingeschränkten Klassen, Klassen-, Rollen- und Datentypdisjunktionen, Klassenkomplementen, Aufzählungen mit mehr als einem Element, und irreflexiven, inversen, funktionalen, invers-funktionalen, symmetrischen und asymmetrischen Objektrollen.

Das QL-Profil basiert auf DL-Lite [26] und eignet sich für eine Domänenmodellierung mit einer Ausdrucksmächtigkeit auf dem Niveau von Entity-Relationship- und UML-Diagrammen. Ziel ist die schnelle Beantwortung von Abfragen auf großen Instanzmengen. Das Profil wurde so entworfen, dass die Daten in einer Relationalen Datenbank abgelegt werden können und sich die Standardinferenzaufgaben durch Query Rewriting in SQL-Abfragen realisieren lassen, wodurch eine logarithmische Antwortzeit erzielt werden kann.

In OWL QL ist die Nutzung von existenzquantifizierten Klassen als Unterklasse, universalquantifizierten Klassen, reflexiven Klassen, kardinalitätseingeschränkten Klassen, Aufzählungen, Klassen-, Rollen- und Datentypdisjunktionen, Rollenketten und funktionalen, invers-funktionalen und transitiven Rollen untersagt. Zudem können Individuen niemals denselben Referenten haben, also keine Identität teilen.

Das RL-Profil wurde mit dem Ziel entworfen, skalierbares Reasoning zu ermöglichen, ohne die Ausdrucksmächtigkeit zu stark einzuschränken. Es wurde von Description Logic Programs (DLPs) [56] inspiriert und definiert eine Menge von Deduktionsregeln, die von einer Regel-Engine umgesetzt werden können. Die Standardinferenzaufgaben können damit in polynomialer Zeit durchgeführt werden.

In OWL-RL existieren einige Einschränkungen, durch die ein Schließen auf Individuenexistenz außerhalb der Wissensbasis vermieden wird. Dafür dürfen OWL-Klassen nur Unterklassen von benannten Klassen, Aufzählungen, Schnittmengen, Vereinigungen und existenzquantifizierten Klassen und nur Oberklasse von benannten Klassen,

Schnittmengen, Komplementen, universalquantifizierten Klassen, existenzquantifizierten Klassen und maximalen Kardinalitätseinschränkungen von null oder eins sein.

Grafische Notation

Der OWL-Standard definiert keine grafische Ontologienotation. Daher wird in dieser Arbeit eine an die Darstellung des protégé-Editors[1] angelehnte Notation verwendet. In Abschnitt A.1 (siehe Seite 203) ist dargestellt, wie Klassen, Individuen, abstrakte und konkrete Rollen, Datentypen, Hierarchien, Äquivalenzen, Zuweisungen und Einschränkungen dargestellt werden.

2.3.3 SPARQL Query Language for RDF

Die manuelle Traversierung von RDF-Graphen bei der Informationsabfrage und -verarbeitung ist sehr aufwändig. Daher hat es sich das W3C zum Ziel gesetzt, eine Abfragesprache zu entwickeln, die das Linked-Data-Pendant zu SQL in Relationalen Datenbanken darstellt. Es entstand die SPARQL Query Language for RDF (SPARQL) [61], die auf Basis einer Menge von Anwendungsfällen entwickelt wurde, die in den RDF Data Access Use Cases and Requirements [28] definiert wurden. Die Struktur der Sprache ist sehr stark an SQL angelehnt; statt jedoch Tabellen über Fremdschlüssel miteinander zu verknüpfen, werden über Tripel und Operatoren Muster im Graph beschrieben. Da OWL in RDF serialisiert ist, kann SPARQL auch zur Abfrage und Verarbeitung von OWL-Ontologien eingesetzt werden. SPARQL unterscheidet grundsätzlich vier Query-Typen, die sich über ihre Kopfklausel und ihre Ergebnisform differenzieren; ihr Körper ist nahezu identisch.

Der Körper aller Query-Typen beginnt mit einer optionalen From-Klausel, in der die Eingabegraphen über IRIs referenziert werden können. Die Abfragen werden auf der Vereinigung dieser Graphen durchgeführt. Ohne Angabe eines Datensatzes wird die Abfrage auf dem Standarddatensatz des SPARQL-Endpoints ausgeführt.

In der Where-Klausel wird das zu bindende Muster definiert. Das Muster besteht aus einer Menge von Tripeln, deren Subjekt, Prädikat oder Objekt mit Variablen oder Konstanten belegt sein können. Zusätzlich gibt es Operatoren, um die Ergebnismenge durch Filter einzuschränken, optionale Tripel zu binden, Vereinigungen und Differenzen zu bilden, und Aufzählungen zu definieren.

[1] http://protege.stanford.edu/

```
@prefix exp: <http://example.org#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

exp:Alice   a          foaf:Person .
exp:Alice   foaf:name  "Alice" .
exp:Alice   foaf:knows exp:Bob .
exp:Bob     foaf:knows exp:Alice .
exp:Bob     foaf:name  "Bob" .
exp:Alice   foaf:knows exp:Charlie .
exp:Charlie foaf:knows exp:Alice .
exp:Charlie foaf:name  "Charlie" .
exp:Alice   foaf:knows exp:Snoopy .
exp:Snoopy  foaf:name  "Snoopy"@en .
```

Abbildung 2.5.: Leicht abgewandeltes Beispielquelldokument aus dem SPARQL-Standard [61].

Ein Filter ist ein logischer Ausdruck, der gebundene Variablen testet und gegebenenfalls Ergebniskombinationen eliminiert. Dafür stehen eingebaute Operatoren und Funktionen zur Verfügung, die auf XPath und XQuery zurückzuführen sind [80] und bei Bedarf erweitert werden können. Optionale Tripel sorgen dafür, dass entsprechende Variablen bei einem nicht existenten Tripel mit einem Nullwert belegt werden, anstatt die gesamte Kombination zu eliminieren. Vereinigungen und Differenzen verknüpfen Ergebnisuntermengen. Mit Aufzählungen kann eine Variable vorab mit einer Menge von Werten belegt werden. Darüber hinaus können Select-Queries in die Where-Klausel eingebettet und deren Ergebnisse genutzt werden.

Das Prädikat eines Tripels kann neben einer einzelnen Relation auch ein Pfadausdruck sein, der eine Aneinanderreihung von Relationen darstellt, die mit Modifikatoren versehen werden können. Subjekt und Objekt werden jeweils an das Start- und Zielobjekt des Pfades gebunden. Die Modifikatoren schränken die Länge der Teilpfade auf genau einmal, null oder einmal, beliebig oft oder mindestens einmal ein.

Nach der Where-Klausel können verschiedene Ergebnismodifikatoren genutzt werden, die eine Ordnung definieren, die Anzahl der Ergebnisse einschränken, globale Filter definieren oder Ergebnisse über eine Gruppenklausel gruppieren.

Im Gegensatz zum Körper, unterscheiden sich die vier Query-Typen in der Kopfklausel. Die im Folgenden gezeigten Beispiele sind leichte Abwandlungen der im SPARQL-Standard [61] genutzten Beispiele.

Select-Queries sind das Pendant zu SQL-Abfragen. Sie erzeugen eine Ergebnistabelle, deren Struktur durch eine Auswahlklausel als Menge von Ausdrücken definiert wird. Jede mögliche Variablenbelegung des Graphmusters resultiert in einer Ergebniszeile

in der Tabelle. Variablen, die über eine Gruppenklausel zusammengefasst wurden, können im Kopf von Select- oder Construct-Queries als Parameter für Aggregationsfunktionen genutzt werden. So kann beispielsweise für das in Abbildung 2.5 dargestellte Quelldokument der Name jeder Person und die Anzahl ihrer Freunde abgefragt werden:

Select-Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name (COUNT(?friend) AS ?count)
WHERE {
    ?person foaf:name ?name .
    ?person foaf:knows ?friend .
} GROUP BY ?person ?name
```

Tabellarisches Ergebnis

?name	?count
„Alice“	3
„Bob“	1
„Charlie“	1

Construct-Queries sind stärker mit dem Graphgedanken verknüpft. Sie verstehen sich als Transformationsfunktionen, die einen oder mehrere Eingabegraphen auf einen Ausgabegraph abbilden. Die Konstruktionsklausel besteht aus Tripelausdrücken, deren Subjekte, Prädikate und Objekte entweder Variablen oder Konstanten sein können. Für jede aus der Abfrage resultierende Variablenbelegung wird die Menge dieser Tripel im Ausgabegraph materialisiert. So kann beispielsweise ein W3C-VCard-Dokument für Alice erzeugt werden:

Construct-Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX exp: <http://example.org#> .
PREFIX vcard: <http://www.w3.org/[...]#>
CONSTRUCT   { exp:Alice vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

RDF-Ergebnis

```
@prefix exp: <http://example.org#> .
@prefix vcard: <http://www.w3.org/[...]#> .

exp:Alice vcard:FN "Alice" .
```

Describe-Queries sind eine abgeschwächte Form von Construct-Queries und bilden ebenfalls einen oder mehrere Eingabegraphen auf einen Ausgabegraphen ab. Der Ausgabegraph stellt jedoch eine Untermenge der Vereinigung der Eingabegraphen dar. In der Beschreibungsklausel können Konstanten oder Variablen des Graphmusters angegeben werden, die beschrieben werden sollen. Im Ausgabegraph werden dann alle Tripel materialisiert, in denen die Konstante oder eine Belegung der Variable als Subjekt vorkommt.

Describe-Query	RDF-Ergebnis

`PREFIX foaf: <http://xmlns.com/foaf/0.1/>`
`DESCRIBE ?person`
`WHERE { ?person foaf:name "Alice" . }`
	`@prefix exp: <http://example.org#> .`
	`@prefix foaf: <http://xmlns.com/foaf/0.1/>`
	
	`exp:Alice a foaf:Person .`
	`exp:Alice foaf:name "Alice" .`
	`exp:Alice foaf:knows exp:Bob .`
	`exp:Alice foaf:knows exp:Charlie .`
	`exp:Alice foaf:knows exp:Snoopy .`

Ask-Queries bilden die Eingabegraphen auf einen Wahrheitswert ab. Sie haben keinen Kopf, sondern nur ein Graphmuster als Körper. Der resultierende Wahrheitswert spiegelt wider, ob für das Graphmuster mindestens eine gültige Variablenbelegung existiert.

Asd-Query	Boolesches Ergebnis
`PREFIX foaf: <http://xmlns.com/foaf/0.1/>` `ASK { ?x foaf:name "Alice" }`	true

In der später entwickelten Management-Architektur wird SPARQL als Hauptabfragesprache eingesetzt.

2.4 Regelbasierte Systeme

2.4.1 Regelbasierte Verarbeitung

Bei der klassischen Datenverarbeitung gibt ein Algorithmus die Ausführungspfade, die bei der Modellanalyse und -manipulation durchlaufen werden, unter Einsatz von Kontrollstrukturen fest vor. Lassen sich die als Teil des Algorithmus auf die Daten angewandten Funktionen eindeutig ordnen, sodass Folgefunktionen immer nur von den Resultaten vorangehender Funktionen abhängen, stellt eine solche sequentielle Verarbeitung ein effektives Mittel dar. Sobald jedoch zyklische Abhängigkeiten zwischen den Ein- und Ausgabedaten der Operationen bestehen, sind der Entwurf und die Umsetzung eines korrekten, vollständigen und terminierenden Algorithmus komplex.

Bei der regelbasierten Verarbeitung ist die Ausführungsreihenfolge der einzelnen Funktionen nicht fest definiert. Der Algorithmus wird in eine Menge von Wenn-Dann-Regeln (Bedingung und Konsequenz) zerlegt, die keine expliziten Abhängigkeiten

aneinander haben, sondern nur implizit durch ihre Ein- und Ausgabedaten miteinander verknüpft sind. Alle Regeln arbeiten auf einer gemeinsamen Faktenmenge, der sogenannten Wissensbasis.

Grundsätzlich existieren bei der Regelumsetzung zwei Verarbeitungsmodelle: das datengetriebene Forward-Chaining und das zielgetriebene Backward-Chaining. **Forward-Chaining** wird meist bei der Verarbeitung sogenannter Produktionsregeln eingesetzt. Die Bedingung einer Produktionsregel definiert die als Eingabe benötigten Fakten, die Konsequenz legt fest, welche Operationen auf der Wissensbasis ausgeführt werden sollen. Ein Kontrollalgorithmus prüft bei jeder Wissensbasisänderung, ob eine neue Faktenkombination existiert, die eine Regelbedingung erfüllt und führt ggf. die als Konsequenz definierten Operationen aus.

Als Standardverfahren für Forward-Chaining hat sich der Rete-Algorithmus (Rete lat. für Netz oder Netzwerk) [46] etabliert, der Speicher für Geschwindigkeit opfert. Dabei wird jede Regel als Baum modelliert, dessen Knoten Verarbeitungsoperationen sind. Neue Fakten werden an den Blättern des Baumes eingefügt und die Resultate der Operationsknoten jeweils an die Elternknoten propagiert, bis ein Datentupel die Wurzel des Baumes erreicht und die Regel aktivierbar wird. Innerhalb des Baumes wird in sogenannte Alpha- und Beta-Knoten unterschieden.

Alpha-Netzwerke (Ketten von Alpha-Knoten) stellen die Blätter des Baumes dar. Sie filtern Fakten anhand einfacher Attributtests, die deren Belegung gegen Konstanten oder Attribute desselben Fakts prüfen. Jeder Alpha-Knoten führt dabei genau einen Test durch. Hält ein Fakt gegen den Test, wird er an den nachfolgenden Knoten weitergereicht. Der initiale Alpha-Knoten des Netzwerks filtert in der Regel auf Basis des Faktentyps (dem Prädikatensymbol).

Am Ende eines Alpha-Netzwerks steht der sogenannte Alpha-Memory, in den jeder Fakt eingefügt wird, der gegen die Tests aller Alpha-Knoten des Pfades gehalten hat. Wird ein Fakt später aus der Wissensbasis entfernt oder in der Form modifiziert, dass er nicht mehr gegen das Alpha-Netzwerk hält, wird er aus dem Alpha-Memory entfernt.

Beta-Knoten dienen als Verknüpfungsoperatoren zweier Unternetzwerke. Jeder Knoten testet die aus seinen beiden Unternetzwerken resultierenden Datentupel paarweise und fügt bei Erfolg das aus deren Vereinigung resultierende Datentupel in seinen Beta-Memory ein. Dabei speichert er sich partielle Übereinstimmungen, sodass nur relative Änderungen betrachtet werden müssen. Ein Beta-Knoten kann als Eingabe entweder die Fakten eines Alpha-Memories, die er als einelementige Tupel auffasst, oder die Faktentupel von Beta-Memories nutzen.

Um regelübergreifende Redundanzen zu vermeiden, werden äquivalente Knoten von mehreren Regelbäumen geteilt. Dies führt dazu, dass mehrere Rete-Netzwerke entstehen, die jeweils gerichtete, azyklische Graphen sind. Die ehemaligen Wurzelelemente der Bäume werden im Netzwerk als Terminalknoten (oder p-Nodes für Production Nodes) bezeichnet. Durch die Redundanzvermeidung ist die Performance des Rete-Algorithmus theoretisch unabhängig von der Regelanzahl [46].

Macht eine Wissensbasisänderung mehrere Regeln aktivierbar, wird eine Konfliktlösungsstrategie benötigt. Häufig werden dazu die Regeln mit Prioritäten versehen und dementsprechend abgearbeitet. Dabei kann es durchaus vorkommen, dass die Konsequenz einer höherprioren Regel dazu führt, dass eine bereits aktive, niederpriore Regel wieder deaktiviert wird.

Beim zielgetriebenen **Backward-Chaining** ist das Regelsystem passiv und beantwortet Anfragen. Dabei werden neben den in der Wissensbasis materialisierten Fakten auch Schlussfolgerungen von logischen Implikationen berücksichtigt, die als Modus-Ponens-Inferenzregeln formuliert sind.

Um Abfragen an die Wissensbasis zu stellen, formuliert der Nutzer eine Hypothese, die das Regelsystem versucht zu beweisen. Ein Standardverfahren ist dabei die Selective Linear Definite (SLD) Clause Resolution [49], die für Horn-Klauseln korrekt und vollständig ist. Dabei wird ein Suchbaum aufgespannt, dessen Wurzelelement die Hypothese darstellt. Für jede Inferenzregel (Fakten werden als bedingungslose Inferenzregeln betrachtet), die ein Literal der Klausel eines Knotens oder dessen Unifikation impliziert, wird ein Kindknoten hinzugefügt, dessen Klausel die Resolution der Elternklausel mit der Inferenzregel darstellt. Wird ein Knoten mit leerer Klausel erzeugt, wurde eine SLD-Resolution für die Hypothese gefunden. Da die Suchstrategie bei der SLD-Resolution nicht festgelegt ist, ist der Algorithmus prinzipiell nichtdeterministisch.

Der Vorteil von Backward- gegenüber Forward-Chaining besteht darin, dass nur zielführende Ableitungsschritte durchgeführt werden. Im Gegensatz dazu leitet ein Forward-Chaining-basiertes System immer alle herleitbaren Fakten ab, was zur Folge hat, dass möglicherweise niemals benötigte Fakten in der Wissensbasis materialisiert werden. Bei der sogenannten Truth Maintenance muss dann stetig geprüft werden, ob die Ableitungsgrundlage des Fakts weiterhin gegeben ist oder dieser entfernt werden muss. Der Vorteil von Forward-Chaining liegt hingegen in dessen Datengetriebenheit, bei der das System selbstständig auf Änderungen reagiert und dem Benutzer neue Schlussfolgerungen unverzüglich mitteilt.

Hybridansätze kombinieren die Vorteile beider Verfahren und haben sich in den letzten Jahren verstärkt etabliert. Dabei greifen Produktionsregeln, die die Verarbeitungslogik treiben und mit Forward-Chaining ausgewertet werden, auf Fakten zurück, die die Regel-Engine versucht bei Bedarf per Backward-Chaining aus logischen Implikationen herzuleiten. Dadurch wird die Anzahl an materialisierten Fakten beschränkt und trotzdem eine datengetriebene Verarbeitung ermöglicht.

In der später entwickelten Management-Architektur spielt die regelbasierte Verarbeitung bei der Umsetzung der Management-Logik eine zentrale Rolle.

2.4.2 Rule Interchange Format

Das W3C gründete 2005 eine neue Arbeitsgruppe, deren Aufgabe es war, ein standardisiertes Regelaustauschformat als Teil der Infrastruktur des Semantic Webs zu entwickeln. Dadurch sollte eine Interoperabilität zwischen den am Markt existierenden Regelsystemen und -sprachen geschaffen werden. 2010 veröffentlichte die Gruppe die erste Empfehlung des Rule Interchange Formats (RIF). Die Sprache ist modular und besteht aus aufeinander aufbauenden Dialekten mit formal definierter Syntax und Semantik. Ein RIF-Dokument setzt sich aus Metainformationen und Regelgruppen zusammen. In den Metainformationen können ein Basis-IRI und IRI-Präfixe definiert und andere RIF-Dokumente über ihren IRI importiert werden. Regelgruppen enthalten vom gewählten Dialekt abhängige Regeln oder weiter verschachtelte Regelgruppen. Das primäre Serialisierungsformat zum Austausch von RIF-Dokumenten ist XML, es existiert jedoch auch eine menschenlesbare Präsentationssyntax.

Der RIF Core Dialekt (RIF-Core) [18] ist der minimale RIF-Dialekt, der die Grundlage aller Dialekte bildet. Er ist im Prinzip ein Datalog [48] mit speziellen RIF-Built-Ins (RIF-DTB) [97] und definiert die Struktur von Horn-Formeln ohne Funktionssymbole mit einer herkömmlichen Prädikatenlogiksemantik. Eine Regel in RIF-Core kann ein atomarer Term (= Fakt) oder eine Implikation (= Deduktionsregel) sein, die in eine Variablendeklaration eingebettet sein können.

Ein atomarer Term ist entweder ein Atom oder ein Frame. Atome sind sogenannte Positionsterme, die aus einem konstanten Bezeichner (Prädikatsymbol oder Funktionsname) und einer Menge von Argumenttermen bestehen. Ein Frame besteht aus einem Term, der den Framekontext festlegt, und einer Menge von Abbildungen, die dessen Attribute an Terme binden. Terme sind Konstanten, Variablen, Termlisten oder externere Terme.

```
(* Variablendeklaration der Implikation *)
Forall ?item ?deliverydate ?scheduledate ?diffduration ?diffdays (
  (* Implizierter Positionsterm und Bedingungskonjunktion *)
  cpt:reject(<John> ?item) :- And (
    cpt:delivered(?item ?deliverydate <John>)
    cpt:scheduled(?item ?scheduledate)
    (* Aequivalenz mit externem Term *)
    ?diffduration = External(
      (* Built-In *)
      func:subtract-dateTimes(?deliverydate ?scheduledate)
    )
    ?diffdays = External(func:days-from-duration(?diffduration))
    External(pred:numeric-greater-than(?diffdays 10))
  )
)
```

Abbildung 2.6.: Beispiel einer Implikation in der RIF-Präsentationssyntax nach [19].

Eine Implikation setzt sich aus Bedingung und Konsequenz zusammen. Die Konsequenz definiert Atome, die halten, wenn die Bedingung der Implikation hält. Die Bedingung ist eine Formel, die eine Konjunktion oder Disjunktion von Formeln, eine Variablendefinition für eine Formel, ein atomarer Term, eine Termäquivalenz, eine Klassenzugehörigkeit oder ein externes Atom sein kann. Abbildung 2.6 zeigt eine Beispielimplikation, die aussagt, dass John jedes an ihn gelieferte Paket ablehnt, das mehr als zehn Tage Verspätung hat.

Der Basic Logic Dialect (RIF-BLD) [19] ist eine Erweiterung des Core-Dialekts, der als zusätzliche Sprachkonstrukte die Unterklassenformel und den Term mit benannten Argumenten einführt. Die Unterklassenformel erlaubt es, Terme auf deren Klassenhierarchie hin zu prüfen. Der Term mit benannten Argumenten stellt eine Erweiterung des Positionsterms dar, bei dem die Attribute, an die die Argumente gebunden werden sollen, explizit benannt werden, anstatt sie implizit aus der Reihenfolge abzuleiten.

Der Production Rule Dialect (RIF-PRD) [102] erweitert den Core-Dialekt um Produktionsregeln, deren Konsequenz im Gegensatz zu Deduktionsregeln keine atomaren Terme ableitet, sondern Aktionen ausführt. Dadurch wechselt das Regelsystem von einer passiven, Anfragen beantwortenden Position in eine aktive, reagierende Position, in der es seine Wissensbasis und seine Umwelt beeinflussen kann.

Die Konsequenz einer Produktionsregel ist ein Aktionsblock, der aus einer Menge atomarer Aktionen besteht. Eine atomare Aktion kann das Hinzufügen (Assert), Entfernen (Retract) oder Modifizieren (Modify) eines Fakts oder das Ausführen einer externen Aktion (Execute) sein. Abbildung 2.7 zeigt eine Beispielproduktionsregel,

```
(* Variablendeklaration der Produktionsregel *)
Forall ?customer ?purchasesYTD (
   (* Bedingung als Konjunktion *)
   If And (
     (* Klassenzugehoerigkeit *)
     ?customer#ex:Customer
     (* Frame *)
     ?customer [ ex:purchasesYTD -> ?purchasesYTD ]
     (* Externer Term *)
     External ( pred:numeric-greater-than( ?purchasesYTD 5000 ) )
   )
   (* Konsequenz als Aktionsblock *)
   Then Do (
     (* Modifikation *)
     Modify ( ?customer [ ex:status -> "Gold" ] )
  )
)
```

Abbildung 2.7.: Beispiel einer Produktionsregel in der RIF-Präsentationssyntax nach [102].

die einem Kunden den Goldstatus zuweist, wenn er seit Beginn des Jahres für mindestens 5000 Dollar eingekauft hat.

Neben den drei standardisierten RIF-Dialekten existiert das Framework for Logic Dialects (RIF-FLD) [20], das einen kohärenten Weg darstellt, auf Basis von RIF-Core neue, ausdrucksmächtigere Dialekte zu definieren.

2.5 Event-Verarbeitung

Der Physiker A.P. French beschreibt ein Event als Verknüpfung eines Punkts im Raum-Zeit-Kontinuum mit einer physikalischen Situation oder einem physikalischen Ereignis [47]. Im IT-Operations-Management wird der Event-Begriff meist als Synonym einer strukturierten Nachricht genutzt, die als Folge eines Ereignisses manifestiert wurde. Sie umfasst in der Regel die Ereignisart, dessen Zeitpunkt und ereignisbedingte Kontextinformationen. Ein wichtiger Aspekt der Informationsverarbeitung ist seit jeher die Analyse und Interpretation solcher Events.

Ein klassisches Vorgehensmodell bei der Event-Verarbeitung besteht darin, die Nachrichten in einer Datenbank zu persistieren und zyklisch oder bei Bedarf Analyseverfahren darauf anzuwenden, die höherwertige Informationen ableiten. Dabei handelt es sich um eine Form von Data Mining auf strukturierten Daten, bei dem neben Standardverfahren zur Ausreißererkennung, Clusteranalyse, Klassifikation, Assoziationsanalyse und Regressionsanalyse auch spezielle Methoden zur Verarbeitung

zeitbehafteter Daten wie Zeitreihenanalyse und Musterabgleich (Pattern Matching) eingesetzt werden. Auf Grund von Echtzeitanforderungen und/oder der ständig wachsenden Informationsflut sind solche Methoden jedoch heutzutage an vielen Stellen nicht mehr anwendbar.

Die sogenannte Datenstromanalyse (Data Stream Mining) umfasst Verfahren, die nicht auf einer statischen Informationsbasis arbeiten, sondern höherwertige Informationen in Echtzeit aus Datenströmen gewinnen. Die dabei eingesetzten Algorithmen bauen sich ein lokales Gedächtnis auf, in dem nur benötigte Informationen gehalten werden, und verwerfen die restlichen Daten. Dadurch wird der Speicherbedarf deutlich reduziert und aktuelle Kenngrößen sind ständig verfügbar.

Eine besondere Form des Data Stream Minings stellt das Complex Event Processing (CEP) dar. Etzion und Niblett definieren die Kernaufgaben von CEP unter anderem im Musterabgleich, der Event-Filterung, -Aggregation und -Transformation [39]. Das Grundprinzip besteht darin, aus einem Verbund niederwertiger Events höherwertige Informationen abzuleiten und diese als komplexe Events zu materialisieren.

Beim Musterabgleich wird versucht, ein bestimmtes Event-Muster im Datenstrom zu erkennen und so beispielsweise aus einer beobachtbaren Einzelschritten einen nicht unmittelbar beobachtbaren Prozess abzuleiten. Wird eine entsprechende Folge erkannt, wird sie als komplexes Event materialisiert, das wiederum als Eingabe weiterer Analysen dienen kann. Häufig spielt beim Musterabgleich der zeitliche Zusammenhang der Events eine wichtige Rolle, der in der Regel mit den Relationen von Allens Intervallalgebra [1] ausgedrückt wird. Sie definiert dreizehn Beziehungen, in denen zwei Intervalle zueinder stehen können (siehe Tab. 2.4).

Die Event-Filterung dient der Ausdünnung des Event-Stroms. Dabei werden Events anhand ihrer Charakteristiken klassifiziert und irrelevante Daten verworfen. Dieser Prozess kann auch kontextsensitiv geschehen, indem höherwertige Events als Entscheidungsgrundlage hinzugezogen werden.

Event-Aggregationen spielen bei der Datenverdichtung eine zentrale Rolle. Dabei werden Events mit gleichen Charakteristiken (gleicher Event-Typ, gleiche Schlüsselattribute) gruppiert und Aggregationsfunktionen (Summe, Minimum, Maximum, Mittelwert, etc.) auf den Inhalt numerischer Attribute angewandt oder die Mächtigkeit der Gruppe bestimmt (Elementanzahl). Dabei spielen Event-Fenster eine wichtige Rolle. Sie bilden eine Art Event-Strom-Analogie zu Datenbanksichten, indem sie einen bestimmten Abschnitt des Stroms repräsentieren, auf den Operatoren angewandt werden. Man unterscheidet in gleitende (sliding) und schubweise (batch)

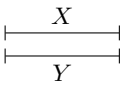
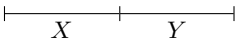
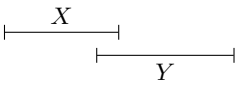
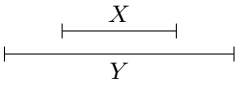
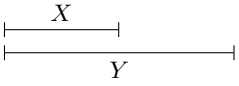
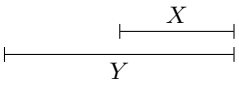
Beziehung	Symbol	Inverse	Bildhaftes Beispiel
X before Y	$<$	$>$	
X equal Y	$=$	$=$	
X meets Y	m	mi	
X overlaps Y	o	oi	
X during Y	d	di	
X starts Y	s	si	
X finishes Y	f	fi	

Tabelle 2.4.: Intervallbeziehungen aus Allen's Intervallalgebra [1].

Fenster (Windows), die entweder eine bestimmte Zeitspanne oder Eventanzahl umfassen (siehe Abb. 2.8).

Sliding Windows bewegen sich kontinuierlich mit der Zeit und umfassen alle Events vom Jetzt bis in die für das Fenster definierte, relative Vergangenheit. Betritt oder verlässt ein Event das Fenster, müssen die auf das Fenster angewandten Operatoren neu evaluiert werden.

Batch Windows repräsentieren für eine fest definierte Verweildauer (Zeit oder Anzahl Events) einen Abschnitt des Event-Stroms und bewegen sich anschließend in einer atomaren Bewegung nach vorne. Während das Fenster an einem Abschnitt verweilt, sind die Events, die es umfasst, statisch. Daher müssen die Operatoren nur einmal pro Bewegung neu evaluiert werden, Zwischenzustände werden nicht betrachtet.

Eine standardisierte Complex-Event-Processing-Sprache existiert nicht. Viele der eingesetzten Sprachen basieren jedoch auf der Continuous Query Language (CQL) [4], deren SQL-artige Syntax genutzt werden kann, um Queries an Datenströmen zu registrieren. CQL bietet sogenannte Stream-to-Relation-, Relation-to-Relation- und Relation-to-Stream-Operatoren. Stream-to-Relation-Operatoren bilden Event-Datenströme anhand gleitender Fenster auf Relationsmengen ab. Relation-to-Relation-Operatoren sind traditionelle SQL-Queries, die Relationsmengen verknüpfen und daraus neue Relationsmengen ableiten. Relation-to-Stream-Operatoren bilden die

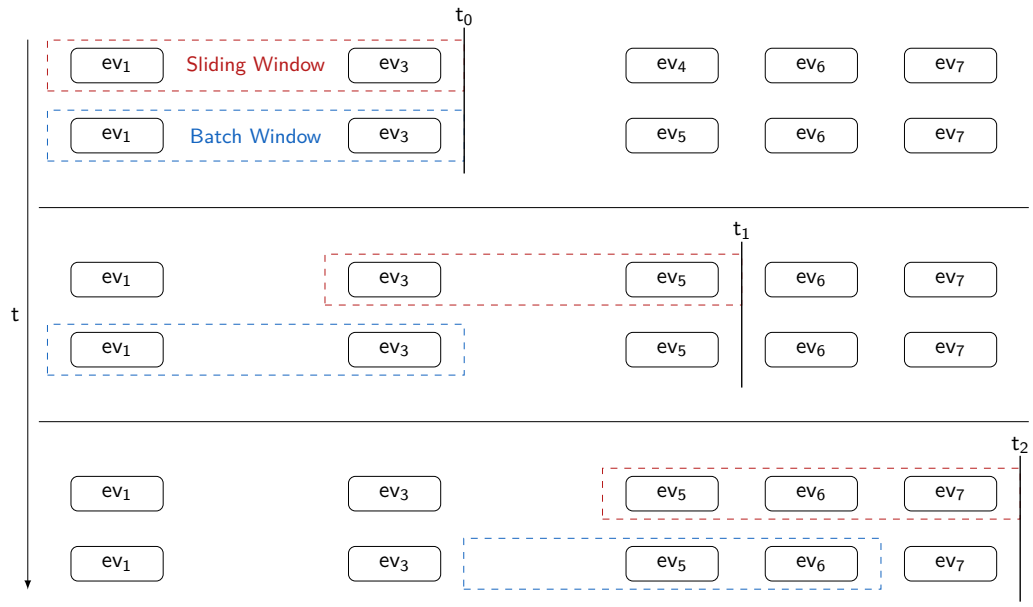


Abbildung 2.8.: Gleitende und sich schubweise fortbewegende Zeitfenster.

```
1 Select Distinct L.vehicleId, L.segNo, L.dir, L.hwy
2 // Stream-to-Relation über gleitendes Zeitfenster
3 From SegSpeedStr [Range 30 Seconds] as A,
4 // Stream-to-Relation über partitioniertes Elementfenstern
5 SegSpeedStr [Partition by vehicleId Rows 1] as L
6 // Relation-to-Relation als Join
7 Where A.vehicleId = L.vehicleId
```

Abbildung 2.9.: Beispiel eines CQL-Queries aus [4].

transformierten Relationsmengen auf Datenstromoperationen ab. So können mit dem Istream (insert Stream)-Operator Events zum Datenstrom hinzugefügt und mit dem Dstream (delete Stream)-Operator Events aus dem Datenstrom entfernt werden; der Rstream (relation Stream)-Operator fügt bei jeder Anwendung alle Relationen als Events zum Datenstrom hinzu, unabhängig davon, ob sie bereits existieren. Abbildung 2.9 zeigt ein aus [4] stammendes CQL-Query, das alle aktiven Fahrzeuge selektiert, die aktuell das High-Way-System nutzen. Für alle Fahrzeuge, die in den letzten 30 Sekunden erfasst wurden, wird aus dem letzten Erfassungs-Event die Fahrzeugnummer, die Segmentnummer, die Richtung und der High Way extrahiert.

Im IT-Operations-Management bieten sich durch den Einsatz von Complex Event Processing viele Möglichkeiten. Durch die Entwicklung von reaktiven Systemen, die Datenströme unterschiedlicher Quellen konsumieren und analysieren können, ist ein datengetriebenes Management möglich, das deutlich kürzere Reaktionszeiten

erlaubt als eine zyklische Abarbeitung eines fest definierten Management-Kreislaufs. In der später entwickelten Management-Architektur werden CEP-Mechanismen für die kontextsensitive Event-Vorverarbeitung und -Abbildung eingesetzt.

3 Analyse

In Abschnitt 1.4 (siehe Seite 5) wurde dargelegt, welche Subsysteme benötigt werden, um ein automatisiertes, wissensbasiertes IT-Operations-Management auf Basis der MAPE-K-Architektur umsetzen zu können. Zusätzlich wurden Probleme bezüglich der Repräsentation und Verarbeitung von IT-Management-Informationen in einer ontologischen Wissensbasis aufgezeigt. Ziel dieses Kapitels ist es, für die Problemfelder im Detail zu untersuchen, welchen Stellenwert sie in der Gesamtarchitektur haben, welchen Anforderungen sie unterliegen und welche Lösungsansätze dazu existieren. Dabei ist die Analyse logisch in die Teilbereiche Wissensbasis (siehe Abschnitt 3.2) und autonomer Regelkreis (siehe Abschnitt 3.3) untergliedert.

Im Wissensbasisabschnitt wird untersucht, welche Aufgabe die semantische Wissensbasis hat, wie sie strukturiert ist, wie darin temporale Zusammenhänge repräsentiert werden können, welche Bedeutung Ontologie-Reasoning hat und wie dessen Performance verbessert werden kann, und wie die Management-Logik darin beschrieben werden kann. Im Regelkreisabschnitt wird untersucht, wie Management-Informationen semantisch vorverarbeitet und auf das Informationsmodell abgebildet werden können, wie das mit Instanzdaten angereicherte Informationsmodell analysiert werden kann, und wie auf Basis des resultierenden Modells Rekonfigurationen geplant und umsetzt werden können.

Da das automatisierte, wissensbasierte IT-Operations-Management kein völlig neues Forschungsgebiet darstellt, werden in Abschnitt 3.1 zunächst Arbeiten untersucht, die eine direkte Verwandtschaft zu dieser Arbeit aufweisen. Dazu werden die darin beschriebenen Ansätze vorgestellt und anhand der definierten Anforderungen charakterisiert. Die Charakterisierung dient zum einen dazu, die Ansätze von dem hier vorgestellten Ansatz abzugrenzen, zum anderen zeigt es, welche Arbeiten Lösungsansätze für die Problemfelder bereitstellen und daher bei der anschließenden Detailbetrachtung berücksichtigt werden müssen.

Bei der Detailanalyse der einzelnen Problemfelder werden neben den unmittelbar verwandten Arbeiten auch Arbeiten herangezogen, die keinen direkten Bezug zum IT-Management haben, jedoch einen isolierten Lösungsansätz darstellen.

3.1 Verwandte Arbeiten

In diesem Abschnitt wird eine kurze Übersicht verwandter Arbeiten gegeben, die ebenfalls das automatisierte, wissensbasierte IT-Operations-Management thematisieren. Die darin vorgestellten Ansätze werden anschließend anhand der definierten Anforderungen und Ziele charakterisiert und die hier vorgestellte Arbeit von ihnen abgegrenzt.

3.1.1 Übersicht

De Vergara et al. stellen in [58] eine Management-Architektur vor, in der die zentrale Wissensbasis in OWL realisiert ist. Sie umfasst mehrere Domänenontologien, die auf konzeptueller Ebene verknüpft sind. Domänenspezifische Technologie-Gateways fragen Laufzeitinformationen von überwachten Komponenten ab und übertragen deren Zustände auf die Wissensbasis. Eine Kombination aus OWL-Semantik und SWRL-Regeln [70] sorgt während des anschließenden Reasonings dafür, dass die Entitäten domänenübergreifend verknüpft, neues Wissen geschlussfolgert und als OWL-S-Prozesse [81] modellierte Management-Aktionen aktiviert und parametrisiert werden. Die aktivierten Aktionen werden anschließend in Dienstaufrufe auf den überwachten Komponenten umgesetzt, sodass ein vollständiger Management-Zyklus eines Policy-basierten Managements (PBM) [123] umgesetzt wird. In [59] wird gezeigt, wie die Architektur im Netzwerk-Management eingesetzt werden kann, indem eine Low-Level-Ontologie, die einen Netzwerkdienst auf technischer Ebene beschreibt, und eine High-Level-Ontologie, die ein Service Level Agreement auf betrieblicher Ebene beschreibt, miteinander verknüpft werden. Der Zustand des Netzwerks wird über die Gateways auf die Low-Level-Ontologie abgebildet und daraus der Service-Status für die High-Level-Ontologie abgeleitet. Fällt die Leitung eines Premiumkunden aus, schaltet der autonome Manager selbstständig eine Backup-Leitung. In [122] und [36] wird der Ansatz auf weitere Anwendungsfälle übertragen, um dessen Adaptierbarkeit zu zeigen.

Cuppens-Boulahia et al. stellen in [32] die „Reaction after Detection (ReD)“-Architektur vor, die Netzwerkattacken erkennen und geeignete Gegenmaßnahmen einleiten soll. Der umgesetzte Management-Zyklus besteht aus einer Alert Correlation Engine (ACE), die Netzwerkknoten-Alerts korreliert und daraus aktuell stattfindende Attacken ableitet, einer Policy Instantiation Engine (PIE), die die Attacken analysiert und geeignete Netzwerk-Policies als Gegenmaßnahmen auswählt, und Policy Enforcement Points (REP, z. B. Firewalls), die die ausgewählten Policies

umsetzen. Die Policy Instantiation Engine nutzt eine OWL-Ontologie, auf die die im Intrusion Detection Message Exchange Format (IDMEF) [37] kodierten Attacken als Instanzen einer IDMEF-Ontologie abgebildet werden. Die Ontologie beinhaltet neben den IDMEF-Nachrichten Instanzen einer Organization Based Access Control (Or-BAC)-Ontologie [31, 30], in der kontextsensitive Zugriffsrichtlinien modelliert sind. Beim OWL-Reasoning wird über SWRL-Regeln aus den IDMEF-Instanzen der aktuelle Bedrohungskontext abgeleitet, auf dessen Grundlage die Policy-Auswahl geschieht.

In [40] und [41] stellen Fallon et al. mit Aesop ein Framework für die semantische, automatisierte Service-Qualitätsoptimierung in Kommunikationsnetzwerken vor. Die zentrale Wissensbasis der darin umgesetzten MAPE-K Loop ist in OWL realisiert und umfasst Instanzen der End-User Service Management Ontologie (EUSAO) [42]. Sie ist in Partitionen unterteilt, in die Daten anhand ihrer Charakteristiken eingeordnet werden. Zur Laufzeit wird die Wissensbasis initial mit der Netzwerkkonfiguration befüllt. Anschließend wird sie kontinuierlich um Service-Reports angereichert, die von einer Überwachungskomponente empfangen und abgebildet werden. Durch eine Kombination aus OWL-Reasoning und SPARQL-Queries wird zyklisch die Service-Güte aller aktiven Sitzungen abgeleitet. Erfüllt eine Sitzung die im SLA vereinbarten Anforderungen nicht, werden niederpriore Sitzungen automatisch gedrosselt. Erfüllen alle Sitzungen ihre SLAs, werden gedrosselte Sitzungen wieder entdrosselt.

Rana et al. stellen in [98] einen autonomen Manager vor, der den aufkommenden Anforderungen des Heimnetzwerk-Managements gerecht werden soll, indem er automatisiert neue Situationen erkennt und entsprechende Regeln für ein Policy-basiertes Management etabliert. Damit soll er der in [100] beschriebenen Konfigurationskomplexität, die das Know-How des durchschnittlichen Heimnetzwerkbetreibers überschreitet, und der rasant steigende Gerätezahl entgegentreten. Der Manager verfügt über eine OWL-Wissensbasis, die initial mit einer das Heimnetzwerk beschreibenden Domänenontologie befüllt wird. Zur Laufzeit werden Netzwerkpakete durch „Semantic Lifting“ auf die Ontologie abgebildet und durch Reasoning mit Kontextinformationen der Domänenontologie verknüpft. Bei einem zweiten Reasoning-Schritt werden zusätzlich SWRL-Regeln hinzugezogen, die aus vom Benutzer auf einem abstrakten Modell definierten Constraints generiert wurden [99]. Für jede Regelaktivierung wird die Variablenbelegung extrahiert und von einer Transformationskomponente aus dem abstrakten Constraint eine konkrete Netzwerk-Policy generiert und etabliert.

In [126] stellen Xiao et al. einen Ansatz vor, um der Konfigurationskomplexität der stetig wachsenden IP-Netzwerke gerecht zu werden, die unter Anbetracht der

Geräte- und Standardvielfalt manuell nicht mehr zu bewältigen sei [127]. Ihr Ansatz untergliedert das Management-Modell in drei Teile: Das Informationsmodell ist in OWL modelliert und beschreibt die Systemstruktur und deren Taxonomie. Das Verhaltensmodell ist in SWRL-Regeln modelliert, die die Informationsmodelltaxonomie nutzen, um zu beschreiben, wie Komponenten auf Systemzustände reagieren. Das Steuerungsmodell besteht ebenfalls aus SWRL-Regeln, die Dienste des in OWL-S modellierten Service-Modells aktivieren und parametrisieren, wenn bestimmte Zustände eingetreten sind. Eine Management-Komponente füllt zur Laufzeit das Informationsmodell mit Instanzdaten des überwachten Systems, leitet mit einer Inferenz-Engine ab, welche Dienste mit welchen Parametern ausgeführt werden müssen und setzt die entsprechenden Aufrufe auf dem realen System um. Die Autoren beschreiben einen Anwendungsfall, in dem das Informationsmodell aus einer SNMP Management Information Base (MIB) und das Verhaltens- und Steuerungsmodell aus einer Structure of Policy Provisioning Information (SPPI) Policy Information Base abgeleitet ist.

Keeney et al. stellen in [75] ein Policy-basiertes Management Framework vor, das in der Lage ist, heterogene Datenquellen automatisch zu erkennen und zu analysieren. Das Framework besteht aus einer Runtime Correlation Engine (RTCE), die Überwachungsdaten von Services und deren unterlagerten Hosts entgegennimmt und darauf Complex-Event-Processing-Mechanismen anwendet. Dabei werden die Event-Ströme gegen Muster aus einer Symptomdatenbank abgeglichen, um bekannte Fehler zu erkennen. Die Anomaly Detection wendet Kalman-Filter auf Hardware- und Netzwerk-Level-Überwachungsdaten an, um das Normalverhalten des Systems zu erlernen und anschließend Anomalien erkennen zu können. Von der RTCE erkannte Fehler und von der Anomaly Detection erkannte Anomalien werden gemeinsam mit den Low-Level-Daten an die Semantic Attribute Reconciliation Architecture (SARA) [60] weitergeleitet. Dort werden sie mit sogenannten Semantic Attributes ausgestattet, indem von Domänenexperten definierte Anreicherungsregeln angewandt werden. Die annotierten Daten werden in eine RDF-Datenbank geschrieben, die von einer Visualisierungskomponente zur Anzeige dynamischer Views genutzt wird [29]. In [75] wird die Visualisierungskomponente um einen Editor für Management-Regeln erweitert, die aus Policy Condition Clauses (PCC) und Service Actions bestehen. Die Semantic-based Service Control Engine (2SCE) setzt die Regeln zur Laufzeit um, indem sie die von SARA angereicherten Daten darauf anwendet. Es werden zwei Anwendungsbeispiele für die Überwachung von Voice over IP im Firmenumfeld und Media Streaming im Heimnetzwerk vorgestellt.

In [107] präsentieren Strassner et al. die Foundation, Observation, Comparison, Action, and Learning Environment (FOCALE)-Architektur. Sie soll die aktuellen Probleme des Netzwerk-Managements lösen, indem geschäftliche und technologische

Informationen verknüpft und harmonisiert werden. Um diese vertikale Integration zu erreichen, setzt FOCALE Ontologien ein, da UML durch fehlende Synonym- und Äquivalenzkonzepte unzureichend sei [109]. Die beschriebene Kontrollschleife sammelt heterogene Informationen der überwachten Ressourcen und transformiert sie über einen Mediator in das Ontologie-Modell. Anschließend wird der Ist-Zustand mit einem definierten Soll-Zustand verglichen und bei Abweichungen versucht, die Ursache des Problems zu finden, indem bei der Analyse Ontologie-Reasoning, endliche Automaten und Machine-Learning-Algorithmen eingesetzt werden. Der Manager wählt anschließend eine Menge von Operationen aus, die auf das System angewandt werden, um es in den Soll-Zustand zurückzuführen. Ein Policy Server steuert die Aktionen der Komponenten und ist Schnittstelle zum Menschen. Die Architektur sieht vor, dass mehrere autonome Manager über einen Event Service zu einer Management-Domäne zusammengefasst werden können. Die Autoren stellen ein Anwendungsbeispiel aus dem Funknetzwerkbereich vor, gehen jedoch nicht auf Realisierungsdetails ein. In [108] wird gezeigt, wie durch Nutzung von FOCALE und Ontologien hochrangige Business-Policies und niederrangige Netzwerkinformationen zu einem ganzheitlichen Management-Ansatz verknüpft werden können. Durch Mapping werden die unterschiedlichen Sichten miteinander verknüpft und aufeinander abgebildet.

3.1.2 Charakterisierung und Abgrenzung

Tabelle 3.1 zeigt eine Charakterisierung der vorgestellten Arbeiten bezüglich der in Abschnitt 1.4 (siehe Seite 5) definierten Anforderungen und Ziele. Dabei bedeutet ein leerer Kreis beim Informationsmodell, dass ein einfaches semantisches Modell eingesetzt wird (z. B. RDFS), ein voller Kreis, dass es sich um eine echte Ontologie handelt. Bei der temporalen Darstellung zeigt ein leerer Kreis, dass in der Wissensbasis ein grundlegendes Zeitkonzept vorhanden ist, ein voller Kreis, dass zeitliche Verläufe und Zustandsänderungen modellierbar sind. Beim Reasoning bedeutet ein leerer Kreis, dass Reasoning-Mechanismen eingesetzt werden, ein voller Kreis, dass dafür zusätzliche Optimierungsverfahren existieren. Für das Management-Modell zeigt ein leerer Kreis, dass die Management-Logik fast ausschließlich in einem einheitlichen Modell beschrieben wird, ein voller Kreis, dass davon alle relevanten Aspekte abgedeckt werden. Ein leerer Kreis bei der Vorverarbeitung bedeutet, dass Monitoring-Events vorverarbeitet werden, ein voller Kreis, dass dies semantisch geschieht, indem Kontextinformationen aus der Wissensbasis hinzugezogen werden. Bei der Informationsabbildung zeigt ein leerer Kreis, dass kontinuierlich Management-Informationen vom überwachten System auf die Wissensbasis abgebildet werden, ein

	de Vergara et al.	Cuppens-Boulahia et al.	Fallon et al.	Rana et al.	Xiao et al.	Keeney et al.	Strassner et al.
Informationsmodell	●	●	●	●	●	○	●
Temporale Darstellung			○				
Reasoning	●	○	●	○			○
Management-Modell	○				○		
Vorverarbeitung		○		○		○	
Informationsabbildung	○	●	●	●	○	●	○
Modellanalyse	●	●	●	●	●	●	●
Planung und Steuerung	○	○	○	○	○	○	●

Tabelle 3.1.: Charakterisierung verwandter Arbeiten anhand der in Abschnitt 1.4 (siehe Seite 5) definierten Ziele.

voller Kreis, dass dies reaktiv und nicht zyklisch geschieht. Bei der Modellanalyse zeigt ein leerer Kreis, dass grundsätzliche eine Analyse durchgeführt wird, ein voller Kreis, dass dabei weiterführende Mechanismen wie Ableitungsregeln und endliche Automaten eingesetzt werden. Ein leerer Kreis bei Planung und Steuerung zeigt, dass ein automatisiertes Management mit statischen Vorschriften umgesetzt wird, ein voller Kreis, dass das Management-System autonom Entscheidungen trifft.

Die Charakterisierung verwandter Arbeiten zeigt deutlich, dass das automatisierte, wissensbasierte IT-Operations-Management ein relevantes Forschungsgebiet darstellt, auf dem mehrere Forschungsgruppen aktiv sind. Die Charakterisierung der vorgestellten Arbeiten zeigt jedoch auch, dass die meisten Ansätze ausschließlich für einen spezifischen Anwendungsfall entwickelt wurden oder wichtige Aspekte vernachlässigen. Diese Lücke soll von dem in dieser Arbeit entwickelten Framework geschlossen werden, indem es eine allgemeingültige Lösung für ein automatisiertes, wissensbasiertes IT-Operations-Management darstellt, die auf arbiträre Anwendungsfälle adaptiert werden kann.

3.2 Wissensbasis

Der entscheidende Vorteil des ontologiebasierten gegenüber dem klassischen IT-Operations-Management ist die semantische Wissensbasis, die als domänenübergreifendes Informationsmodell Daten unterschiedlicher Quellen verknüpft. Sie bildet die Weltsicht des Management-Systems und damit dessen Analyse- und Entscheidungsgrundlage. Daher ist die Auswahl geeigneter Domänenmodelle, Entwurfsmuster und Technologien, die gemeinsam alle für das Management relevanten Informationen repräsentieren und verarbeiten können, essentiell.

In diesem Abschnitt wird untersucht, wie beim IT-Operations-Management das Informationsmodell der Wissensbasis aufgebaut sein muss, wie Daten zur Laufzeit darin gehalten werden können, wie sich temporales Wissen ausdrücken lässt, wie die Reasoning Performance verbessert werden kann und wie sich die Management-Logik darin in einheitlicher Form beschreiben lässt.

3.2.1 Informationsmodell und Datenhaltung

Beim IT-Operations-Management umfassen die Domänenontologien in der Regel nur die Taxonomie. Um mehrere Domänen miteinander zu verbinden, müssen die entsprechenden Modelle mit Ontology-Matching-Methoden [104] miteinander verknüpft und aufeinander abgebildet werden. Ontology Matching ist ein eigenes Forschungsgebiet der Wissensmodellierung (Knowledge Engineering) und wird in dieser Arbeit nicht im Detail betrachtet.

Instanzdaten werden erst zur Laufzeit in die Wissensbasis eingepflegt, hängen stark vom überwachten System ab und ändern sich stetig. Zur Ablage und Verarbeitung der Laufzeitdaten existieren grundsätzlich zwei Ansätze: (1) Die Wissensbasis ist ein Teil des autonomen Managers und wird In-Memory gehalten. Durch diese enge Kopplung ist eine sehr effiziente Verarbeitung möglich, die Wissensbasisgröße ist jedoch stark eingeschränkt. Zudem existieren meist keine Transaktions- und Persistenzmechanismen. (2) Es wird ein dedizierter Triplestore eingesetzt, der wie eine Datenbank als eigenständiger Prozess fungiert, über ein Protokoll (z. B. SPARQL) entkoppelt ist, Transaktionen unterstützt und die Daten in nicht flüchtigem Speicher persistiert. Dadurch verlängern sich jedoch auch die Zugriffszeiten und eine reaktive Verarbeitung ist auf Grund von fehlenden Schnittstellen häufig nicht möglich. Im folgenden Abschnitt wird untersucht, wie das Informationsmodell und die Datenhaltung bei den relevanten Arbeiten realisiert wurde.

Existierende Ansätze

Die von de Vergara et al. vorgestellte Management-Architektur [59] setzt als Informationsmodell eine als „Common Model“ bezeichnete OWL-Ontologie ein, die eine Verschmelzung mehrerer technischer und geschäftlicher Domänenontologien, sowie OWL-S darstellt. Technische Ontologien repräsentieren IT-Management-Standards wie SNMP und CIM, geschäftliche Ontologien Verträge und SLAs. OWL-S wird eingesetzt, um Dienste zu modellieren, über die der autonome Manager steuernd in das überwachte System eingreifen kann.

In der von Cuppens-Boulahia et al. vorgestellten „Reaction after Detection (ReD)“-Architektur [32] umfasst das Informationsmodell der in der Policy Instantiation Engine angesiedelte Wissensbasis zwei OWL-Domänenontologien. Die Organization Based Access Control (Or-BAC)-Ontologie [31, 30] definiert eine anwendungsfallunabhängige Taxonomie, die mit einer anwendungsfallspezifischen Domänenontologie verknüpft wird, um kontextsensitive Zugriffsrichtlinien zu modellieren. Im konkreten Fall nutzen sie eine eigens entwickelte Intrusion Detection Message Exchange Format (IDMEF)-Ontologie, um die darin bereitgestellten Nachrichtentypen und -attribute mit Firewall-Regeln zu verknüpfen, die in bestimmten Situationen auf den Netzwerkkomponenten etabliert werden. Zur Laufzeit bleiben die modellierten Zugriffsrichtlinien statisch. Die Wissensbasis wird ausschließlich angepasst, indem neue IDMEF-Instanzen eingefügt werden. Dabei wird jede Nachricht isoliert behandelt, es wird also kein Bezug zu vorherigen Nachrichten hergestellt.

Das von Fallon et al. vorgestellte Aesop-Framework [41] nutzt als Informationsmodell der zentralen Wissensbasis die in [42] vorgestellte End-User Service Management Ontologie (EUSAO), die auf existierende Vokabulare wie Friend of a Friend (FOAF) [24] und WGS84 Geo Positioning [22] zurückgreift, aus denen sie Konzepte wie Zeit, physische Netzwerkkomponenten, Orte, Benutzer, Sessions und Metriken übernimmt. Die Wissensbasis ist in unterschiedliche Partitionen untergliedert, in die Individuen zur Laufzeit anhand von Charakteristiken eingeordnet werden. Dies geschieht über Annotationen, die zum Design-Zeitpunkt an Domänenkonzepten angebracht werden und deren Instanzen bezüglich ihrer Dynamik und Temporalität kennzeichnen. Die Dynamik unterteilt Konzepte in statisch, wenn Instanzen nur bei Rekonfiguration des überwachten Systems angepasst werden, und dynamisch, wenn sich Instanzen während des Betriebs verändern. Die Temporalität unterteilt Konzepte in global, wenn Instanzen beim Einlesen einer Konfiguration einmalig erzeugt werden und für den gesamten Betrieb bestehen bleiben, und temporär, wenn Instanzen periodisch erzeugt und gelöscht werden. Zur Laufzeit existieren in der Wissensbasis drei Partitionen: (1) Die Metadatenpartition enthält die Terminologie

und die Regeln des überwachten Systems. Die globale Partition untergliedert sich in einen (2) statischen Teil, in dem alle statischen Individuen mit globaler Temporalität abgelegt werden, und einen (3) dynamischen Teil, in dem alle dynamischen Individuen mit globaler Temporalität abgelegt werden. Die temporäre Partition ist wiederum in Zeitabschnitte unterteilt, in die alle temporären Individuen eingeordnet werden. Individuen eines Zeitabschnitts dürfen nur Individuen desselben Zeitabschnitts oder der globalen Partition referenzieren.

In dem von Rana et al. vorgestellten Ansatz [98] wird in der OWL-basierten Wissensbasis ein Informationsmodell eingesetzt, das die Taxonomie eines Heimnetzwerks beschreibt. Zur Laufzeit umfasst die Wissensbasis die Instanzen des überwachten Netzwerks. Bei der Datenverarbeitung wird ein ähnlicher Ansatz wie in der ReD-Architektur verfolgt, indem jedes Datenpaket auf Ontologieebene gehoben, auf die Wissensbasis abgebildet und isoliert verarbeitet wird.

Xiao et al. [126] setzen eine OWL-Wissensbasis ein, deren Informationsmodell sich aus zwei Domänenontologien zusammensetzt. Das aus den für den Anwendungsfall relevanten SNMP Management Information Bases (MIBs) abgeleitete Domänenmodell stellt die Systemmodelltaxonomie bereit, OWL-S wird eingesetzt, um vom autonomen Manager aufrufbare Rekonfigurationsdienste zu modellieren.

Im von Keeney et al. vorgestellten Ansatz [75] dient ein Triplestore als zentrale Wissensbasis. Das Informationsmodell besteht aus mehreren RDF-Vokabularen, OWL kommt nicht zum Einsatz.

Die von Strassner et al. vorgestellte FOCALE-Architektur [108] beschreibt die Wissensbasis nur sehr abstrakt. Für das Informationsmodell werden zwar Ontologiemodelle verwendet, jedoch keine konkrete Technologie genannt. Die Informationsmodelltaxonomie wird in einem mehrstufigen Prozess aus den relevanten Domänenmodellen gewonnen, indem sie auf das DEN-ng [106] Kontextmodell abgebildet werden. Dabei wird ein „Universelles Lexikon“ genutzt, das systematisch um neue Konzepte und Rollen erweitert wird. Die vorgestellte DEN-ng Upper Ontology kennt als abstrakte Konzepte gemanagte Entitäten, Orte, Zeit, Aktivitäten und Personen bzw. Gruppen.

Zusammenfassung

Die existierenden Arbeiten zeigen, dass sich das in der Wissensbasis eingesetzte Informationsmodell in der Regel aus mehreren Domänenmodellen zusammensetzt, die jeweils einen Teilausschnitt der Realität darstellen. Dadurch ist es möglich, dass

Experten zunächst isoliert ein detailliertes Modell ihrer Domäne anfertigen; die Ontologiemodularität erlaubt es anschließend, die Teilmodelle auf semantischer Ebene zu einer Gesamtsicht zu verknüpfen. Ein solcher Ansatz schafft einen sehr hohen Flexibilitätsgrad, durch den existierende Domänenontologien bei der Adaption des autonomen Managers auf neue Anwendungsfälle wiederverwendet werden können.

Zur Laufzeitdatenhaltung wird in den meisten Ansätzen keine Aussage gemacht. Lediglich Keeney et al. geben explizit an, zur Speicherung einen Triplestore einzusetzen. Bei den anderen Ansätzen ist anzunehmen, dass die Wissensbasis In-Memory umgesetzt ist, da die Architekturen keinen Triplestore vorsehen und die Wissensbasis meist als integraler Bestandteil des autonomen Managers dargestellt wird.

3.2.2 Temporale Darstellung

Die Analyse zeitbehafteter Daten spielt beim IT-Operations-Management in vielen Situationen eine wichtige Rolle. Oft müssen historische Daten herangezogen werden, um durch Zeitreihenanalyse relevante Kenngrößen wie die mittlere Antwortzeit oder Verfügbarkeit eines Dienstes zu bestimmen, oder um damit Vorhersagemodelle wie Neuronalen Netze zu trainieren, deren Aussagen über zukünftige Zustände die Grundlage eines proaktiven Managements bilden. Neben der reinen Betrachtung skalarer Messwerte spielen auch Veränderungen in der Systemkonfiguration eine wichtige Rolle. Dadurch kann bei nachträglichen Fehleranalysen nachvollzogen werden, welche Systemkonfiguration zu einem bestimmten Zeitpunkt herrschte und welche Eingriffe als Ursache in Frage kommen. Zudem können bei der Planung von Management-Aktionen zeitliche Zusammenhänge erkannt, berücksichtigt und in Relation gesetzt werden. Wurde beispielsweise kürzlich eine virtuelle Festplatte auf einen physikalischen Host umgezogen, sollte bis zur Systemstabilisierung keine weitere virtuelle Platte darauf verschoben werden.

RDF und OWL bieten grundsätzlich keine Möglichkeit, temporales Wissen auszudrücken. Ein RDF-Graph sei zeitlos und repräsentiere immer eine Momentaufnahme; um dennoch zeitliche Zusammenhänge darstellen zu können, werde ein entsprechendes Vokabular und Entwurfsmuster benötigt [23].

Existierende Ansätze

Zwar nutzen Fallon et al. in Aesop [41] ein Verfahren, bei dem Service-Report-Individuen mit Zeitstempel versehen und so innerhalb einer Zeitpartition temporal

in Relation gesetzt werden können, eine zeitabhängige Zustandsmodellierung ist damit jedoch nicht möglich.

Unabhängig vom IT-Management existieren jedoch Spracherweiterungen und Entwurfsmuster, die temporale Beziehungen in OWL ausdrückbar machen. Krieger führt in [77] eine umfangreiche Untersuchung unterschiedlicher Modellierungsansätze durch. Neben speziell für OWL entwickelten Ansätzen untersucht er darin, wie sich Verfahren anderer Domänen auf OWL übertragen lassen. Von den insgesamt sieben vorgestellten Ansätzen stuft er drei als vielversprechend ein:

Der Quintupel-Ansatz entspricht nach Aussage von Krieger dem Vorgehen in temporalen Datenbanken und der Logikprogrammierung. Dieses Konzept könne auf OWL übertragen werden, indem Tripel als kleinste Informationseinheit von Quintupeln abgelöst würden, die neben Subjekt, Prädikat und Objekt einen Start- und Endzeitpunkt umfassen. Wenn auch am natürlichsten erscheinend, sei dieser Ansatz jedoch schwer auf OWL übertragbar, da Beschreibungslogiken im Allgemeinen auf unäre Klassenaxiome und binäre Beziehungen beschränkt seien, um die Entscheidbarkeit zu gewährleisten. Zudem könnten existierende Werkzeuge und Reasoner nicht damit umgehen.

Der N-äre-Ansatz nutzt das von der W3C Semantic Web Best Practices and Deployment Working Group in [93] vorgeschlagene Entwurfsmuster zum Ausdruck N-ärer Relationen. Dabei setzt eine Beziehung ein Subjekt nicht unmittelbar mit dem Zielobjekt in Relation, sondern mit einem Proxy-Individuum. Der Proxy steht wiederum mit dem eigentlichen Zielobjekt in Beziehung und hat darüber hinaus Beziehungen zu einem Start- und Endzeitpunkt, über die dessen Gültigkeitsintervall definiert ist. Um den Ansatz nutzbar zu machen, muss die Domänenontologie, auf die das Entwurfsmuster angewandt werden soll, angepasst werden. Alle auszuzeichnenden Datenrelationen müssen in Objektrelationen umgewandelt und alle bereits existierenden Einschränkungen über auszuzeichnende Rollen auf den neuen Sachverhalt umgeschrieben werden.

Der Fluents-Ansatz stellt eine von Krieger et al. entwickelte Erweiterung [78] der von Welty und Fikes entwickelten 4D-Fluents [124] dar. Bei den ursprünglichen Fluents existiert für jede Entität eine perdurante Sicht, die deren Zustand während eines Zeitabschnitts beschreibt. Temporale Beziehungen existieren nicht zwischen den eigentlichen Entitäten, sondern zwischen deren zeitlichen Ausprägungen (Temporal Parts). In [78] wird der Ansatz dahingehend erweitert, dass zeitliche Ausprägungen wie ihre Ursprungsentität getypt sind. Dadurch kann der Ansatz als Entwurfsmuster auf existierende Ontologien angewandt werden, ohne diese anzupassen.

Krieger unterzieht die drei Ansätze einer detaillierteren Evaluation bezüglich Anwendbarkeit, Speicherbedarf und Abfrageantwortzeit. Er stellte fest, dass Quintupel den Einsatz von Standard-Reasonern und -Werkzeugen vollständig ausschließen und es immer spezialisierter Lösungen bedarf. Die N-ären Relationen bedürfen einer Transformation der zeitlich auszuzeichnenden Ontologie, könnten dann jedoch von Standardwerkzeugen verarbeitet werden. Der Fluent-Ansatz erwies sich als am besten adaptierbar, da er als reines Modellierungsentwurfsmuster auf existierende Domänenontologien angewendet werden könne und mit Standardwerkzeugen und -Reasonern verarbeitbar sei.

Bei der Bewertung des Speicherbedarfs und der Abfrageantwortzeit zeigte Krieger, dass der Quintupelansatz deutlich weniger Speicherplatz benötigt, da er keiner zusätzlichen Individuen und Beziehungen bedarf und auch bei der Abfrageantwortzeit klar besser sei, da keine zusätzlichen Kanten traversiert werden müssten. Am schlechtesten schnitt der Fluent-Ansatz ab, da er den größten Datenmehraufwand bedeutet.

Krieger kommt zu dem Schluss, dass der Quintupelansatz den anderen Ansätzen überlegen und die Zeit reif für eine entsprechende Weiterentwicklung der Semantic-Web-Technologien sei.

Zusammenfassung

Keiner der ontologiebasierten Management-Ansätze verfügt über ein Konzept zur temporalen Modellierung, es existieren jedoch Ansätze abseits des IT-Managements. Krieger stellt in seiner umfangreichen Studie die vielversprechendsten Ansätze vor und kommt zu dem Schluss, dass es am sinnvollsten sei, den OWL-Standard von einer Tripel- auf eine Quintupeldarstellung zu erweitern.

3.2.3 Reasoning

Beim Einsatz einer semantischen Wissensbasis spielt Reasoning eine wichtige Rolle. Dabei wird die Konsistenz geprüft und anhand der in den Domänenontologien modellierten Semantik neues Wissen geschlussfolgert. Selbst ein trivial erscheinender Fakt, wie die Klassenzugehörigkeit der Instanz einer Klasse zu deren Oberklasse, muss erst vom Reasoner abgeleitet werden. In der Regel wird die Management-Logik auf so abstrakten Konzepten und Rollen formuliert wie möglich, sodass entsprechende Schlussfolgerungen essentiell bei deren Auswertung sind.

Beim ontologiebasierten IT-Operations-Management wird Reasoning meist in zwei Phasen eingesetzt: in der Design-Phase, um Modellierungsfehler in Domänen- und Verknüpfungsontologien aufzudecken, und in der Laufzeitphase, um die Wissensbasiskonsistenz sicherzustellen und neues Wissen zu schlussfolgern. Dabei werden beim OWL-Reasoning $\mathcal{SHOIN}$- bzw. $\mathcal{SROIQ}(\mathcal{D})$-Beschreibungslogik-Reasoner eingesetzt, die die Standardaufgaben Konzepterfüllbarkeitsprüfung und ABox-Konsistenzprüfung umsetzen. Die Konzepterfüllbarkeitsprüfung deckt nicht instanziierbare Klassen auf, indem sie Widersprüche in der Taxonomie findet, die ABox-Konsistenzprüfung stellt sicher, dass Modellinstanzen der definierten Taxonomie genügen. Für die OWL-relevanten Beschreibungslogiken lässt sich die ABox-Konsistenz- auf die Konzepterfüllbarkeitsprüfung reduzieren [103]. Als Standardverfahren für die Konzepterfüllbarkeitsprüfung hat sich der analytische Tableau [8] etabliert, der `NEXPTIME`-vollständig [120, 79] ist und über den neues Wissen mittels indirektem Beweis geschlussfolgert werden kann.

Die exponentielle Reasoning-Laufzeit führt dazu, dass nur kleine Systeme oder isolierte Ausschnitte, die sich durch wenige hundert Instanzen und Beziehungen repräsentieren lassen, effizient verarbeitet werden können. Das Management komplexer IT-Systeme über mehrere Domänen und Abstraktionsebenen hinweg bedarf jedoch schnell mehrerer zehntausend Instanzen und Beziehungen. Paart man dies mit der Dynamik moderner IT-Systeme, wird schnell klar, dass ein stetiges und uneingeschränktes Reasoning der Wissensbasis nicht praktikabel ist.

Existierende Ansätze

In der von de Vergara et al. vorgestellten Management-Architektur [59] findet Reasoning in zwei Phasen Einsatz: Während der Entwurfsphase der Domänen- und Verknüpfungsontologien wird das volle OWL-Sprachspektrum genutzt, um die Konsistenz der modellierten Taxonomien sicherzustellen. Zur Laufzeit wird ein reduziertes Reasoning durchgeführt, das auf eine Sprachuntermenge zurückgreift, die nach Aussage der Autoren mit Prädikatenlogik erster Stufe umgesetzt werden könne [36]. Details zu der ausgewählten Untermenge und den eingesetzten Verarbeitungsmechanismen werden nicht erläutert.

Das von Fallon et al. vorgestellte Aesop-Framework [41] führt für jeden Zeitabschnitt der partitionierten Wissensbasis zyklisch ein isoliertes Reasoning durch, für das dessen Inhalt mit der Metapartition und der globalen Partition vereinigt wird. Durch diese isolierte Zeitabschnittsbetrachtung kann die Individuen- und Beziehungsanzahl deutlich reduziert und dadurch die Reasoning Performance verbessert werden.

Die Arbeiten von Cuppens-Boulahia et al., Rana et al. und Strassner et al. nutzen zwar OWL-Reasoning zur Wissensbasisverarbeitung, es werden jedoch keine Optimierungsverfahren eingesetzt.

Die im Dezember 2012 veröffentlichte OWL-2-Empfehlung [57] umfasste erstmals die sogenannten Sprachprofile (siehe Abschnitt 2.3.2, Seite 28). Sie definieren Einschränkungen, die die Ausdrucksmächtigkeit der Sprache reduzieren und im Gegenzug effizientere Reasoning-Algorithmen einsetzbar machen. Bisher gibt es jedoch keine Untersuchungen zur Anwendbarkeit der Profile im IT-Operations-Management. Daher wird im folgenden Abschnitt eine Profilanalyse durchgeführt, die die Ausdrucksmächtigkeit der OWL-Profile den Anforderungen des IT-Operations-Managements gegenüberstellt.

Profilanalyse

De Vergara et al. haben in [34] bereits untersucht, über welche Ausdrucksmächtigkeit ein Modell verfügen muss, um die etablierten IT-Management-Informationsmodelle GDMO, SMIv2, MIF, IDL, MOF/CIM und SMIng vollständig beschreiben zu können. Dazu haben sie das in [55] vorgestellte Klassifikationsschema für semantische Modelle auf die Informationsmodelle angewandt und deren Ausdrucksmächtigkeit hinsichtlich Konzepten, Taxonomien, Rollen und Instanzen untersucht. Das Ergebnis wurde in [121] dazu genutzt, eine fast vollständige Äquivalenztabelle zwischen den ITM-Modellen und OWL zu definieren. Eine Ergänzung um die fehlenden Modellkonzepte wurde in [111] vorgenommen.

Anhand allgemeiner IT-Operations-Management-Anforderungen und den auf OWL-Seite der Äquivalenztabelle genutzten Sprachkonstrukte wird im Folgenden untersucht, inwiefern auch die von den Profilen verwendeten Sprachuntermengen als Informationsmodellbasis in Frage kommen.

Das EL-Profil basiert auf der Familie der $\mathcal{EL}++$-Beschreibungslogiken und ist speziell für die Analyse komplexer Terminologien entwickelt worden. Dabei werden jedoch keine Instanzdaten berücksichtigt, was das Profil für den IT-Operations-Management-Einsatz, bei dem die Laufzeitmodellanalyse eine zentrale Rolle spielt, disqualifiziert.

Die Fülle an Instanzdaten, die zur Laufzeit das überwachte System in der Wissensbasis repräsentieren, lässt das QL-Profil als besonders geeignet erscheinen. Es ist in der Lage, Queries in `LOGSPACE` zu beantworten und damit auch extrem großen Wissensbasen mit mehreren Millionen Tripeln performant zu verarbeiten. Betrachtet

man jedoch das Komplement der eingeschränkten Sprach-Features des QL-Profils in Bezug auf die Sprach-Features der Modellabbildung, offenbart sich, dass einige grundlegende Fähigkeiten fehlen. So können keine kardinalitätseingeschränkten, existenzquantifizierten und universalquantifizierten Klassen gebildet und keine funktionalen und invers-funktionalen Rollen definiert werden. Zudem schließt das QL-Profil Individuenäquivalenz aus, wodurch eine entscheidende Grundlage für die domänenübergreifende Instanzdatenverknüpfung fehlt. Dies führt dazu, dass auch das QL-Profil für den Einsatz im ontologiebasierten Management als ungeeignet eingestuft wird.

Das RL-Profil verspricht ein skalierbares Reasoning, ohne die Ausdrucksmächtigkeit zu stark einzuschränken. Es basiert auf einer Regelmenge in Prädikatenlogik erster Stufe, die direkt auf der RDF-Repräsentation des Modells arbeiten, wodurch eine Konsistenzprüfung in PTIME-vollständig und eine Konzepterfüllbarkeitsprüfung in co-NP-vollständig (PTIME-vollständig für atomare Klassenausdrücke) möglich wird [89]. Das Komplement der RL-Features in Bezug auf die Management-Modell-Features offenbart jedoch auch hier Mängel in Form von nur eingeschränkt nutzbaren Kardinalitätseinschränkungen. Zwar erlaubt das RL-Profil im Gegensatz zum QL-Profil kardinalitätseingeschränkte Klassen, beschränkt die Kardinalitäten jedoch auf null oder eins.

Zusammenfassung

Die Analyse verwandter Arbeiten hat gezeigt, dass Reasoning einen festen Bestandteil des ontologiebasierten IT-Operations-Managements darstellt. Lediglich im Ansatz von Keeney et al. wird nicht explizit erwähnt, dass Reasoning zur Laufzeit eingesetzt wird, um die Wissensbasis zu validieren und neue Fakten abzuleiten. Es ist jedoch durchaus naheliegend, dass der von ihnen eingesetzte Triplestore über Reasoning-Mechanismen verfügt.

Grundsätzlich existieren zwei Ansätze, um die Reasoning Performance zu verbessern. Der erste Ansatz besteht darin, die Anzahl der zu verarbeitenden Individuen und Beziehungen zu minimieren. Bei den Ansätzen von Cuppens-Boulahia et al. und Rana et al. geschieht dies implizit, indem jedes Monitoring-Event isoliert abgebildet und verarbeitet wird. Einen deutlich komplexeren Ansatz verfolgen Fallon et al. in ihrem Aesop-Framework, das die Wissensbasis in unterschiedliche Partitionen unterteilt. Beim Reasoning wird jeweils nur ein Fragment der Gesamtwissensbasis betrachtet, indem eine Zeitpartition mit dem globalen Systemmodell vereinigt wird.

Dies führt dazu, dass die Anzahl der zu verarbeitenden Individuen und Beziehungen deutlich reduziert wird.

Der zweite Ansatz besteht darin, die Ausdrucksmächtigkeit des Ontologiemodells einzuschränken und so die Reasoning-Komplexität zu reduzieren. De Vergara et al. nutzen ein Verfahren, bei dem nur zum Modellierungszeitpunkt ein vollständiges OWL-DL-Reasoning durchgeführt wird, zur Laufzeit wird ein nicht näher erläuterter Regelsatz mit reduzierter Semantik eingesetzt. Eine standardisierte Form dieses Ansatzes stellen die OWL-Profile dar, die als Sprachuntermengen die Ausdrucksmächtigkeit reduzieren und so effizientere Reasoning-Algorithmen einsetzbar machen. Bei der durchgeführten Untersuchung zeigte sich jedoch, dass keines der Profile alle betrachteten IT-Management-Informationsmodelle vollständig repräsentieren kann.

3.2.4 Management-Modell

Ein autonomer Regelkreis besteht aus mehreren Subsystemen, die ineinandergreifen und aufeinander abgestimmt werden müssen. Stellt dabei jedes Subsystem eine isolierte Einheit dar, die separat konfiguriert und auf den Anwendungsfall adaptiert werden muss, entsteht schnell ein intransparentes und fehleranfälliges Konglomerat aus Logikfragmenten.

Ein ontologiebasiertes Management-Framework sollte daher nicht nur ein einheitliches Informationsmodell, sondern auch ein einheitliches Management-Modell bereitstellen. Darin sollte die Logik aller am Regelkreis beteiligten Subsysteme unter Bezugnahme auf die Modellentitäten beschrieben werden können, sodass ein transparentes und validierbares Modell entsteht, bei dem der Gesamtzusammenhang von der Datenerhebung bis hin zur getroffenen Management-Entscheidung sichtbar wird.

Existierende Ansätze

De Vergara et al. stellen in [59] einen Ansatz vor, bei dem die gesamte Management-Logik in einheitlicher Form beschrieben wird. Die Abbildung zwischen den domänenspezifischen Ontologien und der domänenübergreifenden Ontologie wird durch in die Ontologie eingebettete Abbildungsregeln beschrieben. Das Analyse-, Planungs- und Steuerungsmodell ist in Form von SWRL-Regel genauso Teil der Ontologie wie die als OWL-S-Prozesse modellierten Management-Aktionen. Diese einheitliche

Repräsentation erlaubt eine transparente Betrachtung des gesamten Management-Prozesses, bei dem der Zusammenhang aller beteiligten Entitäten sofort ersichtlich wird. Xiao et al. verfolgen in [126] dasselbe Prinzip, ohne dabei die Modellabbildung zu berücksichtigen.

Bei den anderen in Abschnitt 3.1 vorgestellten Ansätzen ist die Management-Logik über unterschiedliche Quellen und Komponenten verteilt.

Zusammenfassung

Zwar stellen de Vergara et al. einen Ansatz vor, bei dem die Management-Logik in einheitlicher Form beschrieben werden kann, dies umfasst jedoch nicht die Event-basierte Vorverarbeitung und Abbildung. Darüber hinaus ziehen sie in [36] das Fazit, dass das an vielen Stellen eingesetzte SWRL nicht ausdrucksmächtig genug sei. Dennoch stellt die Idee der Integration des Management-Modells mit der Ontologie einen vielversprechenden Ansatz dar.

3.3 Autonomer Regelkreis

Dem MAPE-K-Modell folgend, besteht ein autonomer Regelkreis aus einer zentralen Wissensbasis und den Monitor-, Analyze-, Plan- und Execute-Subsystemen. Die Wissensbasis wurde bereits im vorherigen Abschnitt betrachtet. In diesem Abschnitt sollen nun die von den Subsystemen des autonomen Regelkreises erbrachten Funktionen erläutert und Umsetzungsmöglichkeiten aufgezeigt werden. Dazu wird untersucht, wie Management-Informationen eines überwachten Systems vorverarbeitet und auf das in der Wissensbasis liegende Systemmodell abgebildet werden können (Monitor), wie daraus mit Analyseverfahren neues Wissen abgeleitet und der Systemzustand bewertet werden kann (Analyze) und wie anschließend Rekonfigurationen geplant und umgesetzt werden können (Plan und Execute).

3.3.1 Vorverarbeitung

Eine zentrale Anforderung an das automatisierte IT-Operations-Management ist es, dass neue Situationen in kürzester Zeit erkannt werden und so ein schnelles Handeln ermöglicht wird. Dabei spielt die Online-Event-Verarbeitung (siehe Abschnitt 2.5, Seite 38) eine entscheidende Rolle, da es durch sie möglich wird, trotz

der in modernen IT-Systemen herrschenden Informationsflut, bei der pro Sekunde mehrere hundert oder tausend Events verarbeitet werden müssen, Management-relevante Daten herauszufiltern, zu korrelieren und zu verdichten. Dadurch wird ein reaktives, durch die Vorverarbeitung getriebenes IT-Operations-Management möglich, bei dem höherwertige Ereignisse als Auslöser genutzt werden, um komplexe Sachverhalte auf das Informationsmodell abzubilden und tiefgreifende Analyse- und Planungsmechanismen anzustoßen. Durch die Wissensbasis besteht dabei die Möglichkeit, den Datenstrom nicht nur rein syntaktisch zu betrachten, sondern den Verarbeitungsprozess semantisch anzureichern, indem Kontextwissen des überwachten Systems herangezogen wird. Damit kann beispielsweise schon bei der Vorverarbeitung erkannt werden, dass ein Host, der plötzlich ein unnatürliches I/O-Verhalten zeigt, nicht angegriffen wird, sondern kürzlich eine neue Virtuelle Maschine darauf umgezogen wurde, für die das Verhaltensmuster typisch ist.

Existierende Ansätze

Cuppens-Boulahia et al. setzen in der in [32] vorgestellten Architektur eine Alert Correlation Engine (ACE) ein. Sie agiert als Complex Event Processing Engine und verarbeitet die im IDMEF-Format über eine Publish Subscribe Middleware empfangenen Netzwerk-Alerts, um daraus Netzwerkattacken abzuleiten. Dazu kombinieren sie unterschiedliche Diagnostiken, die nicht im Detail erläutert werden. Erst die aggregierten Angriff-Events werden zur weiteren Analyse auf Ontologieebene gehoben.

In [75] verknüpfen Keeney et al. zwei Vorverarbeitungsverfahren. Ein mit dem Normalverhalten trainierter Kalman-Filter wird genutzt, um auf einem Hardware- und Netzwerk-Monitoring-Event-Strom Anomalien zu erkennen und in höherwertigen Events zu materialisieren. Die Run-time Correlation Engine (RTCE) nutzt Muster aus einer Symptomdatenbank, um über einen Musterabgleich bekannte Fehler zu erkennen und entsprechende Events zu erzeugen. Die abgeleiteten Events werden dann in der nächsten Verarbeitungsstufe mit semantischen Attributen angereichert und zur Analyse herangezogen.

Rana et al. beschreiben in [98] ein Filterverfahren, bei dem der sogenannte Data Processor (DP) Überwachungsdaten von Routern empfängt, die per Deep Packet Inspection gewonnen werden und aus Name-Wert-Paaren bestehen. Auf jede empfangene Nachricht wird ein Attributfilter angewandt und irrelevante Pakete werden verworfen. Nur Nachrichten, die dem Filter genügen, werden im weiteren Regelkreis betrachtet.

Obwohl das Informationsmodell beim ontologiebasierten IT-Operations-Management bereits in semantischer Form vorliegt, nutzt keine der verwandten Arbeiten diese Informationen zur Vorverarbeitung. Unabhängig vom IT-Management existieren jedoch Ansätze für eine semantische Event-Verarbeitung.

Teymourian et al. präsentieren in [118] und [117] einen Ansatz, der Complex Event Processing mit semantischen Modellen verknüpft, um komplexe Verhaltensmuster abzuleiten und automatisierte Reaktionsregeln auszuführen. So soll beispielsweise in einem E-Health-System ein kontinuierlicher Vitaldatenstrom mit der Krankenakte eines Patienten verknüpft werden, um komplexe medizinische Sachverhalte zu erkennen, oder in einem Aktienhandelssystem Aktienkurse mit Firmenbeziehungen verknüpft werden, um Kaufentscheidungen zu treffen. Eine entwickelte Event Processing Engine wertet dafür Regeln aus, die Event-Informationen mit Hintergrundwissen eines Triplestores verknüpfen. Dazu werden sogenannte sQuery-Rules (semantic Query Rules) genutzt, die SPARQL-Query-Templates beinhalten, die bei der Verarbeitung mit Event-Attributen parametrisiert und auf dem Triplestore ausgeführt werden. Die in der Query gebundenen Variablen können dann bei der weiteren Regelauswertung berücksichtigt werden. Der Ansatz sieht unterschiedliche Caching-Verfahren vor, um den Query-Aufwand zu reduzieren. Zudem können Queries automatisch in Fragmente zerlegt und mit dem benötigten Hintergrundwissen, das bei Updates der Wissensbasis revalidiert werden muss, auf Agenten verteilt werden, um eine effizientere Verarbeitung zu ermöglichen. In [116] stellen Teymourian et al. einen weiteren Ansatz vor, bei dem Events von speziellen Event-Mapping-Agenten um semantische Informationen angereichert werden, bevor sie das Verarbeitungsnetzwerk betreten. Dadurch tragen sie das benötigte Kontextwissen bereits in sich, sodass die Verarbeitungsagenten nicht mehr mit der Wissensbasis interagieren müssen.

Souleiman Hasan et al. präsentieren in [62] einen generischen Ansatz zur semantischen Event-Anreicherung. Sie stellen ein Anreicherungsmodell und einen dazugehörigen Formalismus vor, mit denen es möglich ist, externe Datenquellen einzubinden. Die externen Quellen seien dabei nicht an einen festen Informationstyp gekoppelt; so könnten zum Beispiel Wikis, Relationale Datenbanken oder Graphdatenbanken angebunden werden. Das Anreicherungsmodell sieht vier Klauseln vor, über die die Quelle der Hintergrundinformationen, der Beschaffungsmechanismus bzw. das Protokoll, die Extraktionsstrategie und die Datenfusion beschrieben werden. Die Anreicherung findet in einer dem Event-Verarbeitungsmechnismus vorgeschalteten Komponente statt. Events werden als RDF-Dokumente repräsentiert, denen bei der Anreicherung Tripel hinzugefügt werden.

Darüber hinaus gibt es mehrere Ansätze, die SPARQL (siehe Abschnitt 2.3.3, Seite 30) um CEP-Konzepte (siehe Abschnitt 2.5, Seite 38) erweitern und auf einem Tripeldatenstrom arbeiten. Im Folgenden werden die zwei verbreitetsten Ansätze vorgestellt.

In [9] und [10] stellen Davide Francesco Barbieri et al. die Sprache C-SPARQL vor, die eine Verarbeitung von kontinuierlichen Daten im Umfeld des Semantic Web ermöglichen soll. Es erweitert SPARQL um zusätzliche Sprachkonstrukte und Operatoren zur Datenstromverarbeitung. Über sie können Tripelströme in das Query eingebunden werden, bei denen jedes Tripel um eine vierte Komponente erweitert wird, die dessen Erzeugungszeitstempel repräsentiert. Für jede Quelle kann ein Zeit- oder Elementfenster festgelegt werden, das sich entweder kontinuierlich bewegt (Sliding Window) oder in festen Schritten voranschreitet (Batch Window). Construct-C-SPARQL-Queries können als Datenquelle für andere Queries genutzt werden. Über eine spezielle Funktion können die Zeitstempel der Tripel ausgelesen werden, um sie untereinander in zeitliche Relation zu setzen. Bei der Auswertung werden die Queries in ihre statischen und dynamischen Bestandteile zerlegt. Der statische Teil wird als SPARQL-Query von einem entsprechenden Prozessor verarbeitet, der dynamische Teil wird in die Continuous Query Language (CQL) [3] übersetzt und von einem Data Stream Management System (DSMS) ausgewertet.

In [2] stellen Darko Anicic et al. EP-SPARQL vor, das SPARQL um eine Event-Processing-Komponente erweitert, durch die es möglich sein soll, der veränderten Informationssicht vieler Domänen von einem „Sack voll Daten“ hin zu Datenströmen gerecht zu werden. Sie untergliedern die verarbeiteten Informationen in Datenstrom- und Hintergrundwissen. Im Datenstrom befinden sich RDF-Statements, die mit einem Start- und Endzeitpunkt versehen sind und Events darstellen. Das Hintergrundwissen besteht aus einem klassischen, zeitlosen RDF-Graph, aus dem Kontextinformation bezogen werden können. Über die Sequence- und Equal-Operatoren kann eine zeitliche Beziehung zwischen Statements ausgedrückt, über Filterausdrücke Zeitfenster definiert werden. Spezielle Funktionen erlauben es, auf die zeitlichen Komponenten der Tripel zuzugreifen und diese im Query zu verwenden. Von Construct-Queries erzeugte Statements werden in den verarbeiteten Datenstrom zurückgeführt. Im Kontext des ETALIS-Projekts wurde eine EP-SPARQL-Engine entwickelt, die die Queries und die RDF-Wissensbasis in Prolog übersetzt, um damit Datenströme auszuwerten. Zusätzlich existiert ein Mechanismus, der die temporalen Einschränkungen der Queries untersucht und daraus Regeln generiert, die irrelevante Statements aus dem Datenstrom entfernen.

Zusammenfassung

Die Komplexität und damit einhergehende Informationsflut moderner IT-Systeme machen es unerlässlich, dass ein automatisiertes Management-System in der Lage ist, Daten vor einer tiefgreifenden Analyse zu filtern und zu aggregieren. Zwar existieren ontologiebasierte Management-Ansätze, die Event-Vorverarbeitungsverfahren einsetzen, sie arbeiten jedoch rein syntaktisch auf dem Datenstrom, ohne das vorhandene Kontextwissen zu berücksichtigen.

Außerhalb des IT-Managements existieren Ansätze für eine semantische Event-Verarbeitung. Bei den SPARQL-Erweiterungen wird jede Aussage als Event aufgefasst und mit einem Zeitstempel versehen, der bei der Query-Verarbeitung berücksichtigt werden kann. Die Ansätze von Teymourian et al. und Barbieri et al. fassen Events als RDF-Dokumente auf, die mit Kontextinformationen verknüpft und angereichert werden können. Dabei erlaubt es der Ansatz von Barbieri et al. beliebige externe Datenquellen einzubinden; RDF-Graphen stellen nur eine mögliche Quelle dar. Die Ausdrucksmächtigkeit der Abfragen ist jedoch sehr beschränkt, da ein großer Teil der Verarbeitungslogik von Adapterkomponenten umgesetzt werden muss. Bei Teymourian et al. werden RDF-Triplestores über SPARQL angefragt. Dadurch existiert eine standardisierte Schnittstelle zwischen Verarbeitungssystem und Datenquellen, über die das volle SPARQL-Spektrum genutzt werden kann.

3.3.2 Informationsabbildung

IT-Management-relevante Informationen werden von Datenquellen in der Regel zwar schon in strukturierter, nicht jedoch in semantischer Form bereitgestellt. Um diese Daten in einem ontologiebasierten Management-System nutzbar zu machen, muss eine Brücke zwischen beiden Welten geschlagen werden. Grundsätzlich lassen sich die von den Datenquellen bezogenen Informationen in Momentaufnahmen und Events untergliedern.

Momentaufnahmen werden meist durch zyklische Abfragen von Überwachungsschnittstellen gewonnen (z. B. über eine CIM- oder SNMP-Schnittstelle) und repräsentieren den Systemzustand zum Abfragezeitpunkt. Die Daten liegen im Informationsmodell des Management-Standards vor, dem die Überwachungsschnittstelle zugrunde liegt. Um die Systemrepräsentation in der ontologischen Wissensbasis mit der Momentaufnahme anzugleichen, bedarf es spezieller Mechanismen. Sie vergleichen beide Sichten und leiten daraus eine Menge von Operationen ab, die Individuen und Beziehungen zur Wissensbasis hinzufügen oder daraus entfernen.

Events beschreiben in der Regel relative Zustandsänderungen oder Ereignisse, ohne den Gesamtkontext widerzuspiegeln. Bei der Event-Abbildung ist es daher nötig, dass das zugrundeliegende Systemmodell initial vollständig ist. Events werden dann im Kontext des aktuellen Systemzustands interpretiert und die von ihnen repräsentierten Änderungen in Form von Wissensbasisoperationen umgesetzt.

Existierende Ansätze

De Vergara et al. nutzen in [36] Technologie-Gateways, um eine bijektive Abbildung zwischen den Modellwelten zu erzielen. Die Gateways führen eine zweistufige Abbildung durch. Für jede angebundene Technologie existiert eine eigenständige Domänenontologie, die syntaktisch aus dem technologiespezifischen Informationsmodell abgeleitet wurde. Das Gateway liest die Management-Informationen der betrachteten Komponenten aus und bildet sie syntaktisch auf die Ontologie ab. Im zweiten Schritt werden die im Domänenmodell enthaltenen Informationen nach in einer Mapping-Ontologie festgelegten Regeln auf das globale Ontologiemodell (hier „Common Model“) abgebildet, sodass eine ganzheitliche Management-Sicht entsteht. Die Mapping-Ontologie wird nicht im Detail beschrieben, es wird jedoch auf [76] verwiesen, in dem semi-automatische Mapping-Verfahren eingesetzt werden.

Der von Xiao et al. vorgestellte Ansatz [126] bezieht sich ausschließlich auf das Management von SNMP-basierten Infrastrukturen. Für die Transformation der SNMP MIBs in Ontologien wird ein Prozess beschrieben, bei dem die Ontologie händisch modelliert wird. Wie die Management-Informationen zur Laufzeit auf die Ontologie abgebildet werden, wird nicht beschrieben.

Strassner et al. setzen in [108] ein universelles Lexikon ein, um die betrachteten Informationsmodellfragmente automatisiert in Ontologien zu übersetzen. Die beschriebene Abbildung der Laufzeitinformationen auf das Informationsmodell geschieht über sogenannte Mediatoren. Deren Umsetzung sei anwendungsfallspezifisch und wird nicht beschrieben.

Cuppens-Boulahia et al. nutzen in [32] einen Ansatz, bei dem Events als Instanzen der syntaktisch aus dem IDMEF-Informationsmodell übersetzten IDMEF-Ontologie transformiert werden. Die Policy Instantiation Engine empfängt dazu die von der Alert Correlation Engine erzeugten IDMEF-Nachrichten und führt eine syntaktische Abbildung durch. Dabei wird jede Nachricht zu einem Individuum und deren Attributwerte zu Axiomen, die in die Wissensbasis eingefügt werden. Bei der Analyse können dann die Nachrichten mit Domänenentitäten verknüpft werden.

Fallon et al. transformieren in [40] die von Terminals empfangenen, XML-basierten Service Reports mittels XSL-Transformation in Individuen und Axiome. Dabei werden am Schema angebrachte Annotationen im Semantic Annotations for WSDL and XML Schema (SAWSDL)-Format [44] verarbeitet, die beschreiben, wie Elemente und Attribute auf RDF abzubilden sind. Die transformierten Nachrichten werden dann in die Wissensbasis eingefügt, wo sie bei der Analyse mit den Domänenentitäten verknüpft werden und so indirekt auf sie Einfluss nehmen können.

Rana et al. beschreiben in [98] ihr Verfahren als Semantic Uplift von Überwachungsdaten. Der Semantic Manager empfängt die vom Data Processor vorverarbeiteten Events, transformiert sie in RDF und fügt sie in die Wissensbasis ein. Die Transformation wird dabei nicht explizit beschrieben, sondern durch einen lexikalischen Abgleich von Event-Attributsnamen und im Domänenmodell definierten konkreten Rollen abgeleitet. Kann eine vollständige Abbildung gefunden werden, wird das Event auf seine ontologische Repräsentation abgebildet und in die Wissensbasis eingefügt.

Der von Keeney et al. in [75] vorgestellte Ansatz nutzt das SARA-Framework, um ein Semantic Lifting umzusetzen. Darin ist ein sogenannter Homogenisierer dafür verantwortlich, Informationen der heterogenen Datenquellen auf RDF abzubilden. Dazu führt er eine datenquellenspezifische Modelltransformation durch, die eine syntaktische und strukturelle Abbildung auf das Zielmodell vornimmt. Die auf RDF abgebildeten Informationen können anschließend semantisch weiterverarbeitet werden.

Zusammenfassung

De Vergara et al. beschreiben als einzige ein vollständiges Verfahren, um Momentaufnahmen auf das ontologische Informationsmodell abzubilden. Sie heben die strukturierten Daten auf eine semantische Ebene und gleichen anschließend die Wissensbasis daran an. Bei der Event-Verarbeitung existiert kein Ansatz, der eine echte Event-Interpretation vornimmt, also aus den Events konkrete Zustandsänderungen der überwachten Komponenten ableitet. Die verarbeiteten Monitoring-Events stellen selbst eher Domänenentitäten dar, die auf die semantische Ebene gehoben und in die Wissensbasis eingefügt werden, wo sie dann zwar im Kontext des Systemmodells verarbeitet werden, jedoch keinen direkten Einfluss auf dessen Zustand nehmen können.

3.3.3 Modellanalyse

In der Analysephase werden beim automatisierten Management die aus den unterschiedlichen Datenquellen gewonnenen und auf eine semantische Ebene gehobenen Informationen miteinander verknüpft und aus der resultierenden, domänenübergreifenden Sicht neues Wissen abgeleitet. So können beispielsweise technische Kenngrößen genutzt werden, um sich anbahnende Fehler zu erkennen oder über mehrere Abstraktionsebenen hinweg auf Geschäftsebene die Konformität zu definierten SLA zu prüfen. Die Modellanalyse geht Hand in Hand mit der Rekonfigurationsplanung, da sie dafür verantwortlich ist, eine hinreichende Informationsbasis bereitzustellen, die es erlaubt, adäquate Problemlösungen zu finden.

Beim ontologiebasierten IT-Operations-Management wird zur Analyse häufig eine Kombination aus Ontologie-Reasoning und domänenspezifischen Regeln eingesetzt. Beim Reasoning wird die OWL-Standardsemantik eingesetzt, um die Wissensbasiskonsistenz zu prüfen und auf Basis der Taxonomie neue Fakten zu schließen. Die domänenspezifischen Regeln sind Ableitungsvorschriften, die über die OWL-Ausdrucksmächtigkeit hinausgehen und anwendungsfallspezifische Logik darstellen. Häufig kommen dabei SWRL-Regeln zum Einsatz, mit denen auf Taxonomieebene funktionale Abhängigkeiten und Berechnungsvorschriften ausdrückt werden können, die domänen- und abstraktionsebenenübergreifend agieren. Sie werden als Teil der Ontologie beim Reasoning-Prozess verarbeitet.

Existierende Ansätze

De Vergara et al. setzen in [58] zur Analyse SWRL-Regeln ein, die sie in Constraint-Regeln und Objektverhaltensregeln unterscheiden. Constraint-Regeln beschreiben Konsistenzprüfungen, die über die regulären Mechanismen des OWL-Reasonings hinausgehen. Objektverhaltensregeln definieren, wie die in der Ontologie repräsentierten Domänenobjekte auf bestimmte Ereignisse und Bedingungen reagieren. In ihren Anwendungsfällen setzen sie eine Kombination aus domänenspezifischen Regeln und domänenübergreifenden Regeln ein. Die domänenspezifischen Regeln beschreiben das isolierte Komponentenverhalten einer Domäne. Sie werden manuell aus textuellen Beschreibungen der abgebildeten Management-Standards gewonnen. Die domänenübergreifenden Regeln beschreiben, wie sich Komponenten über mehrere Abstraktionsebenen hinweg beeinflussen. Sie werden anwendungsfallspezifisch von Domänenexperten formuliert. Verknüpft man beide Regeltypen, ist ein Management-System in der Lage, auf Geschäftsebene SLA-Verletzungen zu erkennen,

die durch eine auf technischer Ebene angesiedelte Netzwerkkonfiguration bedingt sind.

Xiao et al. setzten in [126] ein Modellanalyseverfahren ein, das dem von de Vergara et al. sehr ähnlich ist. Das bei ihnen Verhaltensmodell genannte Analysewissen besteht jedoch ausschließlich aus domänenspezifischen Regeln und betrachtet nur technische Daten.

Cuppens-Boulahia et al. nutzen in [32] SWRL-Regeln, um die abgebildeten IDMF-Nachrichten mit dem Systemmodell zu verknüpfen und daraus einen domänenübergreifenden Or-BAC-Bedrohungskontext abzuleiten, indem beispielsweise aus Ursprungs- und Zieladresse einer IDMF-Nachricht die entsprechenden Host-Individuen im Systemmodell identifiziert und in Beziehung gesetzt werden. Der abgeleitete Kontext dient als Grundlage für die Auswahl geeigneter Gegenmaßnahmen bei einer Netzwerkattacke.

Rana et al. nutzen in [98] ebenfalls SWRL-Regeln, um die abgebildeten Datenpakete mit dem Systemmodell in Kontext zu setzen. Dadurch werden beispielsweise hinterlegte Gerätecharakteristiken abgeglichen, um zu identifizieren, ob ein gesendetes Paket von einem wohlbekannten Gerät stammt.

Fallon et al. nutzen in [40] während der Abbildungsphase SWRL, um unter anderem die Priorität neuer Service Sessions zu bestimmen, indem sie einem Nutzer zugeordnet und dessen globale Priorität verwendet wird. Während der semantischen Analyse setzen sie jedoch auf ausdrucksmächtigere SPARQL-Queries, mit denen sie prüfen, ob die Service-Sessions konform zu den vereinbarten SLAs sind.

Keeney et al. setzen in [75] ebenfalls Analyseregeln ein, verwenden jedoch kein SWRL. Sie nutzen von Domänenexperten formulierte Anreicherungsregeln, die in der Semantic Attribute Reconciliation Architecture (SARA) umgesetzt werden. Sie statten damit verarbeitete Datensätze anhand ihrer Nutzdaten mit sogenannten Semantic Attributes aus.

Strassner et al. beschreiben die Informationsmodellanalyse in [107] konzeptionell als Kombination aus Ontologie-Reasoning, endlichen Automaten und Machine-Learning-Algorithmen, bleiben jedoch Details schuldig.

Zusammenfassung

Die verwandten Arbeiten haben gezeigt, dass die Verknüpfung von OWL-Reasoning und domänenspezifischen Inferenzregeln einen effektiven Analyseansatz darstellt.

Die Flexibilität und Vielseitigkeit einer Regelsprache erlauben es, komplexe Zusammenhänge in modulare Teilaspekte aufzubrechen, die von einer Regel-Engine automatisch verknüpft werden, um funktionale Abhängigkeiten domänen- und abstraktionsebenenübergreifend aufzulösen. Im ontologiebasierten IT-Operations-Management hat SWRL als Regelsprache weite Verbreitung. Die darin formulierten Regeln lassen sich in die Domänenmodelle einbetten und werden zur Laufzeit vom Reasoner ausgewertet.

3.3.4 Planung und Steuerung

Ein zentraler Aspekt des automatisierten Managements ist die Fähigkeit des autonomen Managers, selbstständig Entscheidungen zu treffen und umzusetzen. Dazu muss er über Mechanismen verfügen, die es ihm erlauben, Zustandsabweichungen im Systemmodell zu erkennen und daraus eine Abfolge von Management-Aktionen abzuleiten, die das System in einen Soll-Zustand zurückführen. Die dabei verfolgte Strategie kann entweder von einem Domänenexperten fest vorgegeben, aus vergangenen Entscheidungen erlernt, oder implizit aus einem Wirkmodell abgeleitet werden.

Um eine automatisierte Planung und Steuerung umzusetzen, muss ein Management-Framework Formalismen bereitstellen, über die Management-Aktionen, deren Vorbedingungen und deren Effekte beschrieben werden können. Zur Laufzeit muss das Framework das resultierende Modell interpretieren und umsetzen. Ein verbreitetes Vorgehen ist dabei das Policy-basierte Management (PBM) [123]. Policies legen Kriterien fest, unter denen eine bestimmte Management-Aktion ausgeführt werden soll. Sie werden meist als Wenn-Dann-Regel kodiert, die ein Management-System automatisiert gegen das Informationsmodell prüft und ggf. assoziierte Management-Aktionen ausführt.

Existierende Ansätze

Bei de Vergara et al. [59] und Xiao et al. [126] werden Management-Aktionen als parametrisierbare OWL-S-Prozesse [81] modelliert. Die Policies werden als SWRL-Regeln formuliert, die das in der Analysephase abgeleitete Wissen nutzen, um die OWL-S-Prozesse während des Reasonings zu parametrisieren und zu aktivieren. Nach Abschluss des Reasonings extrahiert das Management-System die Parameter aller aktivierten Prozesse aus der Wissensbasis und setzt die entsprechenden Dienstaufrufe um.

Cuppens-Boulahia et al. verfolgen in [32] ein ähnliches Prinzip, dort werden Management-Aktionen jedoch nicht von OWL-S-Prozessen sondern von Reaction Security Rules abstrahiert. Sie repräsentieren Rechte, die einem Nutzer unter bestimmten Bedingungen eingeräumt werden. Die in SWRL formulierten Policies nutzen den in der Analysephase abgeleiteten Bedrohungskontext, um Rechte zu vergeben oder zu widerrufen, indem entsprechende Beziehungen eingefügt oder entfernt werden. Eine hart-kodierte Komponente ist nach Abschluss des Reasoning dafür zuständig, die resultierende Rechtesituation in reale Netzwerk-Policies umzusetzen, die auf die Netzwerkkomponenten verteilt werden.

Bei Rana et al. werden in [98] ebenfalls SWRL-Regeln eingesetzt, um die Policies auszudrücken, dabei führen die Regeln jedoch keine direkten Management-Aktionen durch, sondern sind selbst auf einer höheren Abstraktionsebene angesiedelte Repräsentationen von Netzwerk-Policies. Während des Reasonings wird bei jeder Regelaktivierung die zugrundeliegende Variablenbelegung erfasst und im Anschluss von einer Transformationskomponente genutzt, um aus einem mit der Regel verknüpften Template konkrete Netzwerk-Policies zu generieren und im System zu etablieren.

Keeney et al. setzen in [75] ein Policy-basiertes Management in ihrer Semantic-based Service Control Engine (2SCE) um. Darin repräsentieren sogenannte Policy Condition Clauses (PCCs) die Policies. In ihnen dienen die aggregierten RTCE-Events als Trigger, die dazu führen, dass die von SARA semantisch angereicherten Daten evaluiert und ggf. Service-Aktionen ausgelöst werden, die in einer Datenbank hinterlegt sind.

In der Arbeit von Fallon et al. wird in [40] ebenfalls eine Art Policy-basiertes Management durchgeführt, dabei sind die Policies jedoch nicht als Regeln formuliert, sondern als fest-kodierte Prozesse. Jeder Prozess definiert eine Kette von SPARQL-Abfragen, die nach dem Reasoning auf dem Informationsmodell durchgeführt werden und die aktuelle Situation überprüfen und bewerten. Verlaufen alle Abfragen eines Prozesses positiv, wird die damit verknüpfte Management-Aktion parametrisiert und ausgeführt. Im vorgestellten Anwendungsfall werden dabei Service-Sessions entweder gedrosselt oder entdrosselt, um die Service-Qualität hochpriorer Sessions zu verbessern.

Strassner et al. stellen in [107] als einzige einen Ansatz vor, der nicht auf Policies basiert. Sie setzen ein autonomes Management um, indem sie im autonomen Manager einen Mechanismus vorsehen, der in der Lage ist, auf Basis des Ist- und des Soll-Zustands des Systems mit Hilfe von Ontologie-Reasoning, endlichen Automaten und Machine-Learning-Algorithmen eine Menge von Operationen auszuwählen, die

das System in den Soll-Zustand überführen. Wie die Operationen beschrieben und ausgewählt werden, bleibt jedoch offen.

Zusammenfassung

Der im ontologiebasierten Management verbreitetste Ansatz ist das Policy-basierte Management. Dabei wird die Management-Logik fest in Regeln kodiert, die das Informationsmodell auf eine Menge relevanter Einflussfaktoren prüfen und bei Erfolg eine definierte Management-Aktion durchführen. Mit einem solchen Verfahren ist ein autonomer Manager in der Lage, in nahezu Echtzeit Management-Entscheidungen zu treffen und umzusetzen. Auch hier ist der Einsatz von SWRL zur Regeldefinition verbreitet. Ein echtes autonomes Management, bei dem der Manager in der Lage ist, selbständig Aktionen zu verketten, um ein höherwertiges Ziel zu erreichen, existiert nur konzeptionell.

4 Konzeption und Realisierung

Inhalt dieses Kapitels sind Konzeption und Realisierung eines neuartigen IT-Operations-Management-Frameworks, mit dem ein automatisiertes Management arbiträrer Anwendungsfälle umgesetzt werden kann. Dazu wird in Abschnitt 4.1 anhand der in Kapitel 3 durchgeführten Analyse zunächst für jedes Problemfeld geprüft, ob ein bereits existierendes Verfahren eingesetzt werden kann. Für die resultierenden weißen Flecken, also die Problemfelder, für die kein geeignetes Verfahren bekannt ist, werden in Abschnitt 4.2 neue Lösungen entwickelt. In Abschnitt 4.3 werden dann die ausgewählten und entwickelten Teillösungen zum neuartigen IT-Operations-Management-Framework verknüpft, dessen prototypische Realisierung in Abschnitt 4.4 beschrieben ist.

4.1 Anwendbarkeit existierender Verfahren

In diesem Abschnitt wird anhand der in Kapitel 3 durchgeführten Analyse geprüft, welche existierenden Verfahren im Framework eingesetzt werden können. Dazu wird für jedes Problemfeld untersucht, ob ein Verfahren existiert, das mit geringen Anpassungen eingesetzt werden kann, ein Verfahren existiert, das konzeptionell vielversprechend ist, jedoch erst auf das Framework zugeschnitten werden muss oder kein geeignetes Verfahren bekannt ist. Resultat ist eine partielle Problemfeldabbildung, die die weißen Flecken aufzeigt, die vor der Framework-Konzeption abgedeckt werden müssen. Die Problemfeldanalyse ist logisch in die Bereiche Wissensbasis und autonomer Regelkreis unterteilt; alle Entscheidungen sind abschließend in Abschnitt 4.1.3 zusammengefasst.

4.1.1 Wissensbasis

In Abschnitt 3.2 (siehe Seite 49) wurde bereits gezeigt, welchen hohen Stellenwert die semantische Wissensbasis im automatisierten IT-Operations-Management einnimmt und welche Tragkraft die Auswahl geeigneter Vokabulare, Entwurfsmuster und Technologien hat.

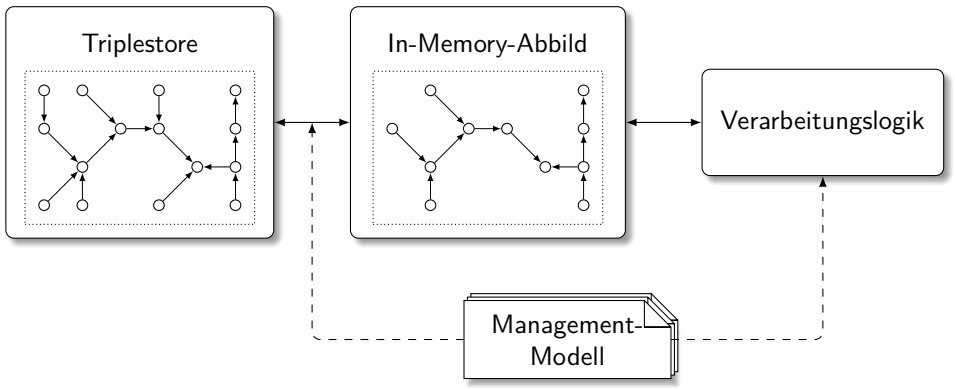


Abbildung 4.1.: Hybrider Datenhaltungsansatz, bei dem die Management-relevante Untermenge für die Verarbeitungslogik effizient zugreifbar In-Memory gehalten wird.

Die in Abschnitt 3.2.1 (siehe Seite 49) durchgeführte Analyse offenbarte, dass das **Informationsmodell** nicht statisch sein darf, sondern sowohl syntaktisch als auch semantisch um anwendungsfallspezifische Konzepte ergänzt werden können muss. OWL verfügt bereits über ein Modularisierungskonzept, mit dem Modelle nachträglich erweitert und verknüpft werden können. Das Framework muss daher lediglich Mechanismen bereitstellen, über die neue Domänenmodelle in das Informationsmodell der Wissensbasis integriert werden können, sodass deren Semantik bei der Verarbeitung berücksichtigt wird.

Zur Laufzeit muss das Framework in der Lage sein, die den aktuellen Zustand des überwachten Systems beschreibenden und potenziell sehr umfassenden Instanzdaten effizient zu verwalten, um den Verarbeitungskomponenten des Regelkreises eine reibungslose Interaktion zu ermöglichen. Dabei existieren grundsätzlich die Ansätze einer In-Memory-Datenhaltung und der Auslagerung in einen Triplestore. Der bei den verwandten Arbeiten verbreitetste Ansatz zur **Datenhaltung** ist In-Memory, der durch die enge Kopplung mit den anderen Komponenten schnelle Lese- und Schreibzugriffe ermöglicht. Bei kleinen Systemmodellen ist dieser Ansatz praktikabel, komplexe Anwendungsfälle stoßen auf Grund der schieren Datenmenge allerdings schnell an ihre Grenzen. Um große Datensätze effizient zu verwalten, ist ein Triplestore besser geeignet, da er Informationen in nichtflüchtigem Speicher abgelegt, indiziert und nur bei Bedarf lädt. Zudem verfügt er über Transaktionsmechanismen, die ACID-Eigenschaften garantieren. Dies geht jedoch auch mit den Nachteilen einher, dass Lese- und Schreiboperationen ineffizienter sind und durch die lose Kopplung in der Regel keine Schnittstellen für eine reaktive Verarbeitung existieren.

Ideal wäre eine Kombination beider Ansätze, bei der Management-relevante Daten für die Verarbeitungskomponenten effizient zugreifbar In-Memory gehalten werden und das vollständige Systemmodell in einem Triplestore abgelegt wird, der bei Bedarf (z. B. externen Queries) abgefragt werden kann. Die Auswahl der Management-relevanten Daten spielt dabei eine wichtige Rolle. Zwar präsentieren Fallon et al. in [41] einen Ansatz, bei dem Daten anhand ihrer Charakteristiken in unterschiedlichen Partitionen abgelegt und nur bei der zyklischen Analyse verknüpft werden, als Auswahlkriterium relevanter Informationen wird dabei allerdings ausschließlich ein Zeitstempel berücksichtigt. Daher soll im Zuge dieser Arbeit ein neuartiger, hybrider Datenhaltungsansatz (siehe Abb. 4.1) entwickelt werden, bei dem Management-relevante Daten auf Basis des Management-Modells automatisch klassifiziert und effizient zugreifbar In-Memory gehalten werden.

Um Informationen in der Wissensbasis nicht nur universell gültig, sondern mit zeitlichen Abhängigkeiten modellieren zu können, wurde in Abschnitt 3.2.2 (siehe Seite 52) untersucht, welche Ansätze zur **temporalen Darstellung** in OWL existieren. Keine der verwandten Arbeiten verfügt über ein entsprechendes Konzept, in [77] unterzieht Krieger jedoch unterschiedliche Ansätze einer Detailanalyse. Dabei kommt er zu dem Schluss, dass Quintupel, N-äre-Relationen und 4D-Fluents am vielversprechendsten und davon die Quintupel auf Grund ihres niedrigen Speicherbedarfs und ihrer guten Abfrage-Performance den anderen überlegen seien.

Der mit den Quintupeln einhergehende Verlust der Standardkonformität, durch den existierende Ontologien, Modellierungswerkzeuge, Bibliotheken und Reasoner nicht mehr einsetzbar sind, wird für den Framework-Einsatz als Ausschlusskriterium angesehen. Als Alternativen verbleiben die N-ären-Relationen und die 4D-Fluents. Da es sich bei beiden Ansätzen um standardkonforme Entwurfsmuster handelt, kann bei der Modellierung und Verarbeitung auf existierende Technologien zurückgegriffen werden. Der Fluents-Ansatz hat allerdings gegenüber des N-ären-Ansatzes einen entscheidenden Vorteil: er lässt sich auf existierende Ontologien anwenden, ohne dass eine Anpassung des Ursprungsmodells nötig ist. Bei den N-ären-Relationen ist dies nicht möglich, da die Modellsemantik verändert werden muss (konkrete werden zu abstrakten Rollen).

Der Fluents-Ansatz erlaubt es, einschränkungsfrei aus der Vielfalt vorhandener Ontologien zu schöpfen und Standardtechnologien bei der Modellierung und Verarbeitung einzusetzen und ist daher der beste Framework-Kandidat. Zwar kommt diese Flexibilität mit einem erhöhten Speicherbedarf und Verarbeitungsaufwand einher, die Vorteile überwiegen jedoch. Durch seine Generizität lässt sich der Fluents-Ansatz nahtlos in existierende Domänenmodelle integrieren. Lediglich zur Laufzeit muss

dafür Sorge getragen werden, dass das Entwurfsmuster beim Einfügen, Entfernen und Abfragen berücksichtigt wird.

Die in Abschnitt 3.2.3 (siehe Seite 54) durchgeführte Analyse zum Einsatz von **Reasoning** hat gezeigt, dass ein Instanzdaten-Reasoning zur Laufzeit unabdingbar, auf Grund der Komplexität allerdings weitgehend ungeeignet für ein Echtzeit-Management ist. Zur Verbesserung der Reasoning Performance haben de Vergara et al. in [59] einen Ansatz vorgestellt, der eine OWL-Sprachuntermenge einsetzt, die die Ausdrucksmächtigkeit und somit auch die Reasoning-Komplexität reduziert. Die eingesetzte Sprachuntermenge ist jedoch nicht formal beschrieben und kann daher nicht adaptiert werden.

Seit OWL 2 existieren als Teil des Standards mit den Sprachprofilen (siehe Abschnitt 2.3.2, Seite 28) formal unterlagerte Sprachuntermengen, die auf unterschiedliche Anwendungsfälle gemünzt sind und in deren Kontext die Reasoning-Komplexität deutlich reduzieren. Die in Abschnitt 3.2.3 (siehe Seite 56) durchgeführte Untersuchung zeigte allerdings, dass keines der Profile mächtig genug ist, alle betrachteten IT-Management-Modelle vollständig zu repräsentieren. Stellt man dies wiederum der Tatsache gegenüber, dass das RL-Profil lediglich beim Ausdruck von Kardinalitätseinschränkungen Defizite aufweist und das Common Information Model (CIM) diese als einziges IT-Management-Informationsmodell unterstützt, erscheint es als guter Kompromiss, Kardinalitätseinschränkungen größer Eins der Domänenmodelle beim Reasoning zu vernachlässigen und im Gegenzug eine deutliche Performance-Steigerung erzielen zu können. Die Reasoning-Komplexität wird gegenüber OWL-DL-basierten Reasonern in den meisten Fällen von `NEXPTIME`-vollständig auf `PTIME`-vollständig reduziert und kann in einer Regel-Engine umgesetzt werden.

Durch die Beschränkung des Informationsmodells auf OWL-RL ist die Framework-Verarbeitungslogik in der Lage, die Reaktionszeit zu reduzieren bzw. mehr Instanzen zu verarbeiten. Darüber hinaus soll der Datenhaltungsansatz dahingehend erweitert werden, dass neben den unmittelbar Management-relevanten Fakten auch all die Fakten In-Memory gehalten werden, aus denen durch OWL-RL-Reasoning Management-relevante Fakten abgeleitet werden können. Dadurch reicht es aus, beim der Verarbeitungslogik vorgeschalteten Reasoning ausschließlich den reduzierten In-Memory-Datensatz zu berücksichtigen. Zur weiteren Optimierung sollen dabei Spezialisierungen der generischen OWL-RL-Regeln eingesetzt werden, die aus der zur Laufzeit statischen Informationsmodelltaxonomie generiert werden.

Die Untersuchung von **Management-Modellen** in Abschnitt 3.2.4 (siehe Seite 58) hat gezeigt, dass der von de Vergara et al. vorgestellte Ansatz [59] der einzige ist,

bei dem große Teile der Management-Logik in einheitlicher Form vorliegen. Sie ist in OWL und SWRL serialisiert und Teil der Wissensbasis. Wichtige Aspekte wie die Event-Verarbeitung werden vom Modell jedoch nicht repräsentiert. Darüber hinaus ziehen die Autoren nach der Umsetzung mehrerer Anwendungsfälle in [36] das Fazit, dass das von ihnen eingesetzte SWRL nicht ausdrucksmächtig genug sei, da keine Aggregationen und Negationen verwendet werden können.

Grundsätzlich erscheint die Idee, die Management-Logik in OWL zu serialisieren und ebenfalls in der Wissensbasis zu halten, dennoch vielversprechend. Dadurch wird ein direkter Bezug zwischen Logik und Domänenentitäten hergestellt, sodass Zusammenhänge über alle Verarbeitungsstufen hinweg sichtbar und validierbar werden. Daher soll dieser Ansatz adaptiert und eine Taxonomie für ein umfassendes Management-Modell entwickelt werden, das es erlaubt, alle relevanten Management-Aspekte in ontologischer Form zu modellieren, dabei direkten Bezug auf die Domänenentitäten nehmen zu können und so eine transparente Datenintegration zu schaffen. Das Framework soll dann zur Laufzeit die modellierte Management-Logik interpretieren und in den einzelnen MAPE-K-Phasen umsetzen.

4.1.2 Autonomer Regelkreis

Auf Grund der schieren Menge an Überwachungsdaten, die von einem IT-Operations-Management-System zur Laufzeit verarbeitet werden müssen, ist es unabdingbar, eine **Vorverarbeitung** durchzuführen. Keine der in Abschnitt 3.3.1 (siehe Seite 59) untersuchten Arbeiten setzt ein Verfahren um, mit dem die in der semantischen Wissensbasis bereitgestellten Kontextinformationen in den Vorverarbeitungsprozess integriert werden können. Es existieren jedoch vom IT-Management unabhängige Ansätze, die eine solche Vorverarbeitung ermöglichen. Die Idee der SPARQL-basierten Ansätze besteht darin, einen Tripeldatenstrom zu verarbeiten, in dem jedes Tripel mit einem Zeitstempel versehen ist und als Event aufgefasst wird. Dieser Gedanke widerspricht allerdings dem Event-Verständnis des IT-Operations-Managements, bei dem es einer strukturierten Nachricht gleichgesetzt wird, dessen Attributsvereinigung eine atomare Zustandsänderung oder ein atomares Ereignis im System beschreibt.

Die semantische Event-Verarbeitung von Teymourian et al. kommt dem, was man sich als Vorverarbeitung eines automatisierten IT-Operations-Managements wünscht, am nächsten. Die regelbasierte Verarbeitung von als RDF-Dokumente repräsentierten Events, bei der über eine standardisierte Abfragesprache Kontextwissen herangezogen werden kann, bildet eine gute Grundlage, an die die Umsetzung im zu

entwickelnden Management-Framework angelehnt werden soll. Monitoring-Events sollen dabei ebenfalls in RDF-Dokumente überführt und deren Struktur als Teil der Domänenontologie beschrieben werden. Im Management-Modell sollen dann Verarbeitungsregeln definiert werden können, die den Event-Datenstrom mit dem in der Wissensbasis vorliegenden Systemmodell verknüpfen und daraus höherwertige Events ableiten. Zur Laufzeit soll das Framework die Regeln interpretieren und umsetzen.

Bei der in Abschnitt 3.3.2 (siehe Seite 63) durchgeführten Analyse zur **Informationsabbildung** existierten mit der Momentaufnahme- und Event-basierten Abbildung zwei unterschiedliche Vorgehensmodelle. Beim IT-Operations-Management sieht man sich in der Regel mit beiden Informationsarten konfrontiert. Gerade bei der Verknüpfung mehrerer Domänen ist die Momentaufnahmeverarbeitung jedoch sehr komplex und aufwendig, da ein stetiger, semantischer Modellabgleich durchgeführt werden muss. Ein solches Vorgehen steht zudem im Widerspruch zum reaktiven IT-Operations-Management, bei dem Informationen nicht zyklisch, sondern kontinuierlich verarbeitet werden. Aus diesem Grund soll im Framework eine Event-basierte Verarbeitung umgesetzt werden. Um dennoch Technologien anbinden zu können, die ausschließlich Momentaufnahmen bereitstellen, können je nach Anwendungsfall technologiespezifische Adapter entwickelt werden, die aus den in der Regel bereits in strukturierter Form vorliegenden Modellen Zustandsänderungen ableiten und als Events materialisieren.

Damit wäre es für ein Framework ausreichend, initial die Sichten der unterschiedlichen Domänen in die Wissensbasis einzupflegen und anschließend durch Event-Interpretation relative Anpassungen daran vorzunehmen. Die in Abschnitt 3.3.2 (siehe Seite 63) vorgestellten Event-basierten Abbildungsansätze sind allerdings nicht darauf ausgelegt, Events zu interpretieren und auf Zustandsänderungen abzubilden, sie führen lediglich eine syntaktische Transformation der Events durch und fügen deren RDF-Repräsentation in die Wissensbasis ein. Betrachtet man die Anforderungen an einen solchen Interpretationsmechanismus, zeigen sich jedoch viele Parallelen zur Event-Vorverarbeitung: Events werden mit dem Systemmodell verknüpft, daraus allerdings keine neuen Events, sondern Wissensbasisoperationen abgeleitet, die das Systemmodell an die Realität angleichen. Daher können im Management-Modell ähnliche Beschreibungsverfahren eingesetzt werden.

Der verbreitetste Ansatz bei der **Modellanalyse** der in Abschnitt 3.3.3 (siehe Seite 66) betrachteten Arbeiten besteht darin, Ableitungsregeln einzusetzen, die auf Konzeptebene des Domänenmodells funktionale Abhängigkeiten definieren. Die Regeln werden zur Laufzeit auf die Wissensbasis angewandt, um Schlussfolgerungen

über nicht beobachtbare Zustände des Systems zu ziehen, Kenngrößen zu aggregieren und die Konformität zu SLAs zu prüfen. Dabei werden meist SWRL-Regeln eingesetzt, die jedoch durch ihre direkte Integration in den Reasoning-Prozess einigen Einschränkungen unterliegen. Um die Entscheidbarkeit zu gewährleisten, müssen SWRL-Regeln DL-Safe [90] sein, dürfen also nicht die Existenz neuer Individuen schlussfolgern. Zudem unterliegen sie der Open-World-Assumption, was die Ausdrucksmächtigkeit in der Form beschränkt, dass in den Regeln weder negiert noch aggregiert werden darf. Besonders die fehlende Wertaggregation stellt für die Modellanalyse ein Ausschlusskriterium dar, da im IT-Operations-Management für fast alle relevanten Kenngrößen Werte aufsummiert, Minima, Maxima oder Mittelwerte gebildet oder Vorkommnisse gezählt werden müssen. Die fehlende Ausdrucksmächtigkeit stellt auch einen der Hauptkritikpunkte der von de Vergara et al. veröffentlichten Fallstudie [36] dar, in der die Autoren ihre mehrjährige Erfahrungen im ontologiebasierten IT-Operations-Managements zusammentragen.

Ein alternativer Ansatz besteht darin, die Analysefunktionalität oberhalb des Reasoning-Prozesses anzusiedeln. So nutzen Fallon et al. nach dem Reasoning SPARQL-Queries für komplexe Analysen. SPARQL verfügt über eine hinreichende Ausdrucksmächtigkeit, ist jedoch als Abfrage- und nicht als Regelsprache angelegt. Daher können Ableitungsschritte zwar in einzelne Queries modularisiert werden, es muss jedoch manuell eine Kompositionsstrategie entwickelt werden, die festlegt, wann und in welcher Reihenfolge die Queries ausgeführt und in welcher Form Zwischenergebnisse propagiert werden, um eine konsistente Auswertung sicherzustellen. Eine datengetriebene Verarbeitung ist damit ebenfalls schwer umsetzbar, da die Query-Ausführung aktiv getrieben werden muss.

Die regelbasierte Modellanalyse stellt durch ihre Modularität und Reaktivität eine effiziente Vorgehensweise dar. SWRL ist allerdings nicht ausdrucksmächtig genug, um allen, für das IT-Operations-Management relevanten, Aspekten gerecht zu werden. Deshalb soll als Teil des Management-Modells eine Beschreibungssprache entwickelt werden, mit der Domänenexperten mächtigere Ableitungsregeln definieren können. Die reaktive Regelauswertung soll zur Laufzeit unter einer Closed-World-Annahme oberhalb Reasonings angesiedelt sein und die Wissensbasis ausschließlich durch explizite, Fakten materialisierende Operationen beeinflussen (siehe Abb. 4.2). Dies erlaubt, dass in den Regeln sowohl Negationen, als auch Aggregationen genutzt werden können.

Für die **Planung und Steuerung** hat sich in Abschnitt 3.3.4 (siehe Seite 68) das Policy-basierte Management als De-facto-Standard des automatisierten, wissensbasierten IT-Operations-Managements erwiesen. Als Regeln formulierte Policies beschreiben

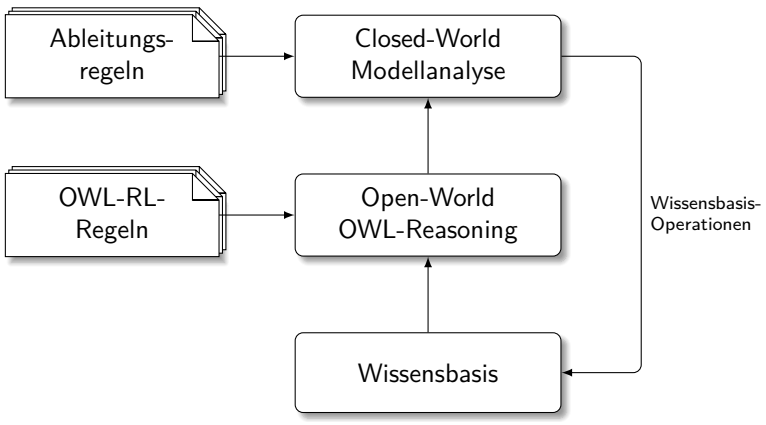


Abbildung 4.2.: Dem Open-World-OWL-Reasoning nachgeschaltete Closed-World-Modellanalyse mit ausdrucksmächtigeren Ableitungsregeln und Faktenmaterialisierung.

explizit, auf welche Systemzustände der autonome Manager mit welchen Management-Aktionen reagieren soll. Dies erlaubt eine effiziente Verarbeitung, bei der Management-Entscheidungen in kürzester Zeit nach Ursacheneintritt getroffen und umgesetzt werden können. Wie schon bei der Modellanalyse kommt bei den meisten der verwandten Arbeiten SWRL zum Einsatz. Die beschränkte Ausdrucksmächtigkeit macht es jedoch auch hier für komplexere Anwendungsfälle weitestgehend ungeeignet. Daher soll es das hier zu entwickelnde Management-Modell ermöglichen, den Analyseregeln an Ausdrucksmächtigkeit äquivalente Policies zu formulieren, die Domänenentitäten und Events verknüpfen und parametrisierbare Management-Aktionen auslösen. Das Framework soll dazu einen Mechanismus bereitstellen, der die Policies zur Laufzeit auswertet und die Management-Aktionen auf dem überwachten System umsetzt.

Zwar stellt das Policy-basierte Management eine effiziente Lösung für isolierte Probleme dar, betrachtet man allerdings komplexere Systeme und die vertikale Integration über mehrere Abstraktionsebenen hinweg, ist es schwer sicherzustellen, dass durch auf niedrigen Ebenen angesiedelte Policies keine auf höheren Ebenen angesiedelten Service Level Agreements verletzt werden. So mag es beispielsweise in den Augen eines Administrators durchaus Sinn machen, eine Policy zu definieren, die eine virtuelle Maschine bei akuter Überlast von einem physikalischen Host auf einen anderen umzieht, dass dadurch jedoch ein wichtiger Geschäftsprozess negativ beeinflusst wird, dessen virtuelle Maschine auf dem Ziel-Host liegt, wird nicht berücksichtigt. Daher soll das Framework über die Umsetzung starrer Policies hinaus in der Lage sein, eigenständig Management-Entscheidungen zu treffen, die den Systemzustand global optimieren. Keiner der in Abschnitt 3.3.4 (siehe Seite 68) vorgestellten Ansätze setzt eine solche autonome Optimierung um. Daher soll ein

neuer Ansatz entwickelt werden, der existierende Optimierungsverfahren auf das ontologiebasierte Management überträgt und in das einheitliche Management-Modell integriert.

4.1.3 Zusammenfassung

Dieser Abschnitt fasst die in Abschnitt 4.1.1 und Abschnitt 4.1.2 getroffenen Design-Entscheidungen bezüglich der Wissensbasis und des Regelkreises zusammen. Tabelle 4.1 zeigt dazu eine Übersicht der ausgewählten Verfahren.

Die Wissensbasis soll auf einem modularen Informationsmodell basieren, das um anwendungsfallspezifische Konzepte und Semantik erweitert werden kann. Die Datenhaltung soll zur Laufzeit zweistufig geschehen: Im Triplestore wird ein vollständiges Systemmodell gehalten, gegen das Anfragen externer Komponenten evaluiert werden, die davon Management-relevante Informationsuntermenge wird redundant In-Memory gehalten, um der Verarbeitungslogik des autonomen Regelkreises einen effizienten Zugriff und eine reaktive Verarbeitung zu ermöglichen. Die Untermenge soll von einem Analyseverfahren automatisch aus dem Management-Modell und der Informationsmodelltaxonomie abgeleitet werden. Für die Modellierung temporalen Wissens sollen die standardkonformen 4D-Fluents eingesetzt werden, die als Entwurfsmuster auf existierenden Ontologien angewandt werden können, jedoch mit einem Daten- und Abfragemehraufwand einher gehen. Zur Verbesserung der Reasoning Performance wird die Ausdrucksmächtigkeit des Informationsmodells auf OWL-RL beschränkt. Dadurch können zwar keine vom Common Information Model unterstützten Kardinalitätseinschränkungen größer Eins berücksichtigt werden, es wird jedoch ein deutlich effizienteres, in einer Regel-Engine umsetzbares Reasoning möglich. Darüber hinaus soll der Datenhaltungsansatz so angepasst werden, dass alle implizit Management-relevanten Fakten ebenfalls In-Memory gehalten werden und so das der Management-Logik vorgeschaltete Reasoning ausschließlich auf dem reduzierten Datensatz durchgeführt werden muss. Dabei sollen taxonomiespezifische Spezialisierungen der generischen OWL-RL-Regeln eingesetzt werden, die automatisch aus der zur Laufzeit statischen Informationsmodelltaxonomie abgeleitet werden. Die Management-Logik soll in einem zu entwickelnden ontologiebasierten Management-Modell ausdrückbar sein, dessen Taxonomie und Semantik eine direkte Verknüpfung zwischen Domänenentitäten und Management-Logik erlaubt und so eine transparente und validierbare Sicht des gesamten Verarbeitungsprozesses bietet. Das Framework soll einen Mechanismus bereitstellen, der die darin beschriebene Logik selbstständig interpretiert und die Verarbeitungskomponenten auf den beschriebenen Anwendungsfall adaptiert.

Problemfeld	Gewählter Ansatz	
Informationsmodell	Modular und erweiterbar als OWL-Ontologien	●
Datenhaltung	Hybrider Ansatz	○
Temporale Darstellung	4D-Fluents	●
Reasoning	Domänenspezifische OWL-RL-Regeln	○
Management-Modell	Ontologiebasiertes Management-Modell	○
Vorverarbeitung	Semantisches CEP	◐
Informationsabbildung	Event-Abbildung auf Wissensbasisoperationen	◐
Modellanalyse	Ableitungsregeln	◐
Planung und Steuerung		
• Automatisches Mgmt.	Policies	◐
• Autonomes Mgmt.	Autonome Optimierung	○

Tabelle 4.1.: Auswahl geeigneter Lösungsansätze für die Problemfelder.
● Der Ansatz wird so im Framework eingesetzt
◐ Das Konzept wird im Mgmt.-Modell und der Laufzeitumgebung adaptiert
○ Der Ansatz muss neu entwickelt werden

Im Regelkreis soll die Vorverarbeitung durch semantisches Complex Event Processing umgesetzt werden, bei dem als RDF-Dokumente serialisierte Monitoring-Events mit dem in der Wissensbasis liegenden Systemmodell verknüpft und daraus höherwertige Events abgeleitet werden. Die Informationsabbildung ist innerhalb des Frameworks auf die Interpretation von Events beschränkt. Momentaufnahmen des überwachten Systems können ausschließlich initial in die Wissensbasis geladen werden, danach liegt es in der Verantwortung technologiespezifischer Adapter, Ereignisse Event-basiert bereitzustellen. Die Event-Interpretation lehnt sich an die Vorverarbeitung an und verknüpft Events mit Kontextinformationen, um daraus Wissensbasisoperationen abzuleiten, die das Systemmodell an die Realität angleichen. Die Modellanalyse soll mit Ableitungsregeln umgesetzt werden, die Aggregation und Negation erlauben, oberhalb des Reasoning-Prozesses angesiedelt sind und automatisch komponiert und reaktiv verarbeitet werden. Bei der Planung und Umsetzung wird ein dualer Ansatz verfolgt: zum einen soll anhand von als Regeln formulierten Policies ein Policy-basiertes Management umgesetzt werden, das in nahezu Echtzeit lokale Management-Entscheidungen treffen und umsetzen kann, zum anderen soll ein autonomes Optimierungsverfahren bereitgestellt werden, das anhand von anwendungsfallspezifischen Aktionseffekten und Bewertungskriterien global optimiert.

4.2 Entwicklung fehlender Verfahren

Der vorherige Abschnitt hat gezeigt, dass zwar für einige der betrachteten Problemfelder Verfahren existieren, die ins Framework integriert werden können, für andere Problemfelder hingegen neue Verfahren entwickelt und existierende Konzepte angepasst werden müssen, um das Ziel eines adaptierbaren Management-Frameworks zu erreichen. Daher wird in Abschnitt 4.2.1 ein Management-Modell entwickelt, das die ausgewählten Vorverarbeitungs-, Informationsabbildungs-, Modellanalyse- und Steuerungskonzepte adaptiert und in einheitlicher Form unter direktem Bezug auf die Domänenentitäten beschreibbar macht. In Abschnitt 4.2.2 wird das Konzept zur hybriden Datenhaltung verfeinert und im Detail ausgearbeitet. Zur Verbesserung der Reasoning Performance wird in Abschnitt 4.2.3 gezeigt, wie der Datenhaltungsansatz auf die implizit Management-relevanten Daten ausgeweitet werden kann und wie aus der zur Laufzeit statischen Informationsmodelltaxonomie spezialisierte OWL-RL-Regeln abgeleitet werden können. Abschließend wird in Abschnitt 4.2.4 ein Ansatz zur autonomen Optimierung entwickelt und das Management-Modell um entsprechende Aspekte erweitert. Die in diesem Abschnitt entwickelten Ansätze stellen einen wesentlichen Beitrag dieser Arbeit dar.

4.2.1 Management-Modell

Bei den in Abschnitt 3.2.4 (siehe Seite 58) vorgestellten Management-Ansätzen ist das Verarbeitungswissen über unterschiedliche Quellen (CEP- und SPARQL-Queries, programmatische Abbildungs-, Analyse und Steuerungskomponenten, etc.) verteilt und stellt in der Regel lediglich syntaktischen Bezug zum Domänenmodell her. Diese Wissensfragmentierung verschleiert den Gesamtzusammenhang, und isolierte Anpassungen am Informations- oder Management-Modell führen schnell zu Inkonsistenzen, die erst zur Laufzeit als Fehler in Erscheinung treten.

Die größte Stärke des ontologiebasierten IT-Operations-Managements liegt in der Informationsharmonisierung, bei der Daten unterschiedlicher Domänen in einem einheitlichen Modell semantisch verknüpft werden. Weitet man dieses Konzept auf das Management-Modell aus und beschreibt alle verarbeitungslogischen Aspekte in einheitlicher, ontologischer und semantisch auf die Domänenentitäten bezugnehmender Form, entsteht eine umfassende und validierbare Gesamtsicht, die den vollständigen Management-Prozess von der Datenerfassung bis hin zur Management-Entscheidung transparent darstellt.

In diesem Abschnitt wird ein ontologisches Management-Modell entwickelt, mit dem ein automatisiertes Management arbiträrer Anwendungsfälle beschrieben werden kann. Darin soll die Vorverarbeitung, Informationsabbildung, Modellanalyse und Policy-basierte Rekonfiguration beschreibbar sein. Das Modell stellt einen elementaren Bestandteil des Management-Frameworks dar.

Anforderungen

Um alle für den Regelkreis eines automatischen Managements relevanten Aspekte beschreiben zu können, muss das Management-Modell die folgenden Anforderungen erfüllen:

A 1 Mit dem Management-Modell muss die Struktur von Monitoring-Events und Management-Aktionen überwachter Komponenten in ontologischer Form beschrieben werden können, sodass sie bei der Definition der Management-Logik referenzierbar sind (Schnittstellenmodell).

A 2 Im Management-Modell muss beschrieben werden können, wie Monitoring-Events miteinander und mit Kontextinformationen der Wissensbasis verknüpft werden können, um daraus höherwertige Events abzuleiten (Event-Vorverarbeitung).

A 3 Im Management-Modell muss beschrieben werden können, wie Events unter Berücksichtigung von Kontextinformationen der Wissensbasis interpretiert und auf Wissensbasisänderungen abgebildet werden können (Event-Interpretation).

A 4 Im Management-Modell müssen komplexe funktionale Abhängigkeiten zwischen Domänenentitäten beschrieben werden können, sodass nicht unmittelbar beobachtbare Komponentenzustände, Leistungskennzahlen, die Service-Qualität und die SLA-Konformität abgeleitet werden können (funktionale Abhängigkeiten).

A 5 Im Management-Modell müssen Policies definiert werden können, die in Abhängigkeit des Systemmodells und des Event-Datenstroms parametrisierbare Management-Aktionen auf dem überwachten System auslösen (Policies).

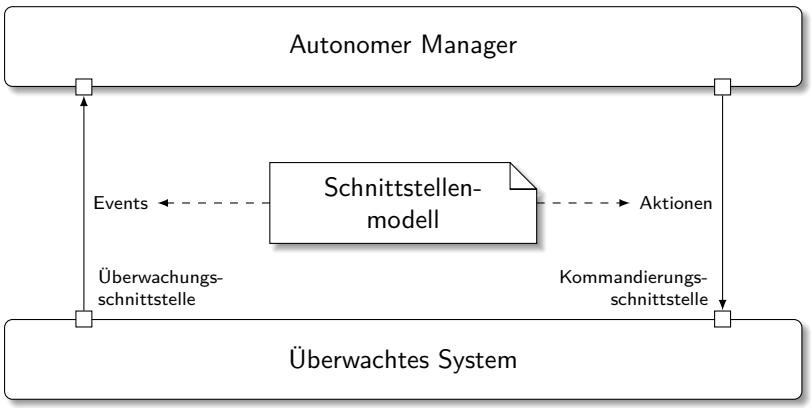


Abbildung 4.3.: Schnittstellenmodell als Schema der zwischen überwachtem System und autonomen Manager ausgetauschten Informationen.

Mit einem diese Anforderungen erfüllenden Management-Modell ist es möglich, die Verarbeitungslogik eines automatisierten Regelkreises zu beschreiben, der auf Basis eines ontologiebasierten Informationsmodells ein Policy-basiertes Management durchführt. Monitoring-Events des überwachten Systems werden unter Bezugnahme auf das Informationsmodell kontextsensitiv miteinander verknüpft, und die resultierenden, höherwertigen Daten interpretiert und auf Zustandsänderungen der Wissensbasis abgebildet. Die Effekte auf das Gesamtmodell werden aus den funktionalen Abhängigkeiten abgeleitet und Leistungskennzahlen, Service-Qualität und SLA-Konformität des überwachten Systems bestimmt. Gegen das resultierende Modell werden Policies evaluiert und abgeleitete Management-Aktionen auf das überwachte System umgesetzt.

Schnittstellenmodell

Zur Laufzeit werden stetig Informationen zwischen einem überwachten System und dem Management-System ausgetauscht (siehe Abb. 4.3). Über eine Überwachungsschnittstelle werden als Events materialisierte Systemereignisse bereitgestellt und in das Framework eingespeist. Der autonome Manager verarbeitet die Events und generiert in Abhängigkeit des Systemzustands Management-Aktionen, die über eine Kommandierungsschnittstelle auf dem überwachten System umgesetzt werden.

Um die domänenspezifischen Events und Aktionen im Management-Modell referenzieren und in die Verarbeitungslogik integrieren zu können, bedarf es einer formalen Beschreibung der domänenspezifischen Schnittstellentypen (Events und Aktionen). In Abschnitt 4.1 wurde bereits festgelegt, Events innerhalb des Frameworks als RDF-Dokumente zu repräsentieren. Domänenspezifische Adapter agieren als Mediatoren,

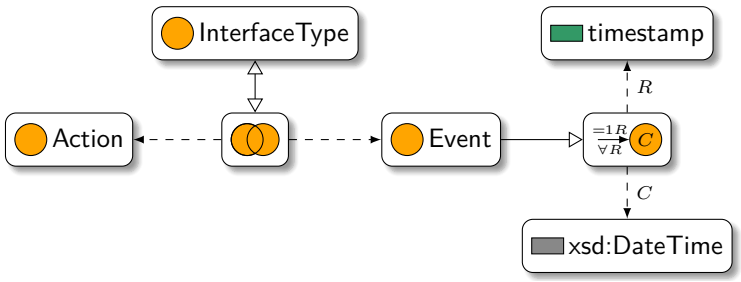


Abbildung 4.4.: Domain Interface Ontology (DIO) mit der Schnittstellentypenoberklasse `InterfaceType` als disjunkte Vereinigung aus `Action` und `Event`.

die zwischen dem System und dem autonomen Manager vermitteln. Dieses Konzept wird auf Management-Aktionen ausgeweitet, die innerhalb des Frameworks und an dessen Schnittstelle ebenfalls als RDF-Dokumente repräsentiert werden.

Um die Struktur der in den Dokumenten serialisierten Daten zu beschreiben, wird OWL eingesetzt. Die Domain Interface Ontology (DIO, Abbildung 4.4) stellt eine leichtgewichtige Taxonomie (siehe Abb. 4.4) bereit, die OWL-Basisklassen und -rollen für Events und Aktionen definiert. Die `Event`-Klasse bildet die Oberklasse aller Events, die `Action`-Klasse die Oberklasse aller Management-Aktionen. Eine Einschränkung legt fest, dass jedes `Event` über die konkrete Rolle `timestamp` mit genau einem `xsd:DateTime`-Literal in Beziehung gesetzt werden muss, das dessen Erzeugungszeitstempel repräsentiert. Für Aktionen gibt es keine Grundeigenschaften. Um die Schnittstellentypen einer Domäne im Management-Modell referenzierbar zu machen, muss eine Schnittstellenontologie modelliert werden, die Aktions- und Event-Unterklassen beschreibt. Die vollständige DIO-Ontologie ist in Abschnitt A.2.2 (siehe Seite 205) zu finden.

Verarbeitungsmodell

Auf Basis des entwickelten Schnittstellenmodells wird in diesem Abschnitt das Verarbeitungsmodell entworfen. Betrachtet man die für den Regelkreis ausgewählten Verfahren, zeigt sich ein gleichartiges Muster: Informationen aus der Wissensbasis und dem Event-Datenstrom werden miteinander verknüpft und daraus neues Wissen über den Systemzustand abgeleitet oder Events und Aktionen materialisiert. Sie folgen damit dem Prinzip einer regelbasierten Verarbeitung, bei der die einzelnen Verarbeitungsfragmente als logische Implikationen (Ableitungsregeln) oder Produktionsregeln (Erzeugungsregeln) beschrieben werden. Im Folgenden werden die einzelnen Verarbeitungsschritte klassifiziert und auf Spezialisierungen logischer Implikationen und Produktionsregeln abgebildet.

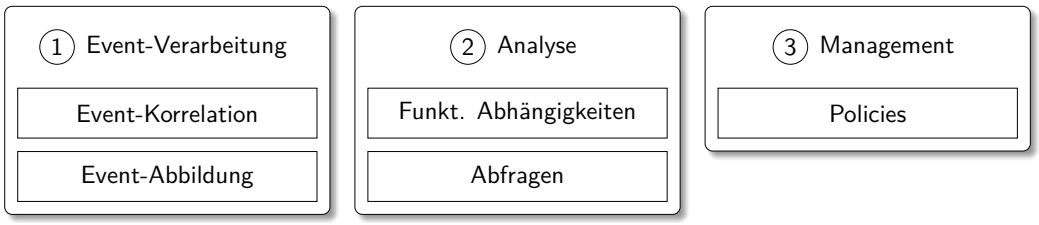


Abbildung 4.5.: Bestandteile des Verarbeitungsmodells.

Logisch lässt sich die Verarbeitungslogik in drei Gruppen unterteilen (siehe Abbildung 4.5):

- Die *Event-Verarbeitungsgruppe* umfasst Verfahren, mit denen Monitoring-Informationen korreliert und auf die Wissensbasis abgebildet werden.
- Die *Analysegruppe* umfasst Verfahren, mit denen innerhalb der Wissensbasis neues Wissen geschlussfolgert wird.
- Die *Management-Gruppe* umfasst Verfahren, mit denen Management-Aktionen ausgeführt werden.

Die in der Analysegruppe angesiedelten Abfragen sind zwar nicht Teil der Anforderungen, sie sollen jedoch dazu dienen, Teilverarbeitungsschritte in Form von abstrakten Zielen zu modularisieren und so die Wiederverwendbarkeit zu erhöhen.

Die in der *Event-Verarbeitungsgruppe* angesiedelten Korrelationen und Abbildungen dienen als Schnittstelle zwischen den Monitoring-Events und dem Informationsmodell. Event-Korrelationen sind Produktionsregeln, die die vom überwachten System bereitgestellten Monitoring-Events verdichten und filtern. Sie verknüpfen die Events des Datenstroms mit Fakten der Wissensbasis und schlussfolgern Event-Aktionen, die Events zum Datenstrom hinzufügen oder daraus entfernen.

Sei E die Menge aller Events, K die Menge aller Wissensbasistripel, T die Menge aller Zeitstempel und $A_E = E \times \{Assert, Retract\}$ die Menge aller Event-Aktionen, dann sind Event-Korrelationsregeln definiert als

$$P_{corr} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(A_E)\}. \tag{4.1}$$

Event-Abbildungsregeln sind Produktionsregeln, die Events des Datenstroms mit Kontextinformationen der Wissensbasis verknüpfen und daraus Faktenaktionen ableiten, die Tripel zur Wissensbasis hinzufügen oder daraus entfernen. Sei $A_K =$

$K \times \{Assert, Retract\}$ die Menge aller Wissensbasisaktionen, dann sind die Event-Abbildungsregeln definiert als

$$P_{map} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(A_K)\}. \tag{4.2}$$

Die *Analysegruppe* umfasst die Verarbeitungslogik, die die Wissensbasis und den Datenstrom analysiert und daraus neues Wissen ableitet, das als Grundlage von Management-Entscheidungen genutzt werden kann. Funktionale Abhängigkeiten sind logische Implikationen, die aus der Wissensbasis neue Tripel schlussfolgern. Ein geschlossenes Tripel ist nur solange gültig, wie auch dessen Grundlage gültig ist. Funktionale Abhängigkeiten sind definiert als

$$I_{dep} = \{f \mid f : \mathcal{P}(K) \to \mathcal{P}(K)\}. \tag{4.3}$$

Abfragen sind logische Implikationen, die sowohl auf dem Informationsmodell, als auch auf dem Event-Datenstrom arbeiten. Anders als funktionale Abhängigkeiten implizieren sie keine Wissensbasistripel, sondern abstrakte Ziele, die von anderen Regeln referenziert oder von externen Komponenten beobachtet werden können. Sei G die Menge aller abstrakten Ziele, dann sind Abfragen definiert als

$$I_{query} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(G)\}. \tag{4.4}$$

Die *Management-Gruppe* umfasst die Verarbeitungslogik, die das reale System durch Management-Aktionen beeinflusst. Policies sind Produktionsregeln, deren Funktionsweise der von Event-Korrelationsregeln ähnelt. Sie verknüpfen Datenstrom-Events und Wissensbasistripel, schlussfolgern daraus jedoch keine Event-Aktionen, sondern lösen Management-Aktionen auf dem überwachten System aus. Sei M die Menge aller Management-Aktionen, dann sind Policies definiert als

$$P_{policy} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(M)\}. \tag{4.5}$$

Die Struktur von Abfragen erlaubt es, den Definitionsbereich aller Implikationen und Produktionsregeln der Form $\mathcal{P}(E) \times \mathcal{P}(K) \times T$ auf $\mathcal{P}(E) \times \mathcal{P}(K) \times \mathcal{P}(G) \times T$ zu erweitern, wodurch alle Regeln (mit Ausnahme von funktionalen Abhängigkeiten) durch Abfragen implizierte abstrakte Ziele verwenden können.

Modelltaxonomie

Im vorangegangenen Abschnitt wurde gezeigt, dass sich die gesamte Verarbeitungslogik des automatischen Regelkreises mit Implikations- und Produktionsregeln

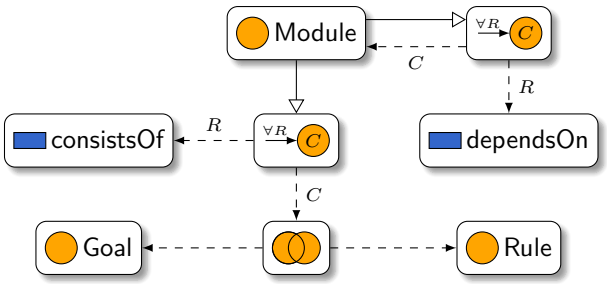


Abbildung 4.6.: Taxonomie von Management-Modulen.

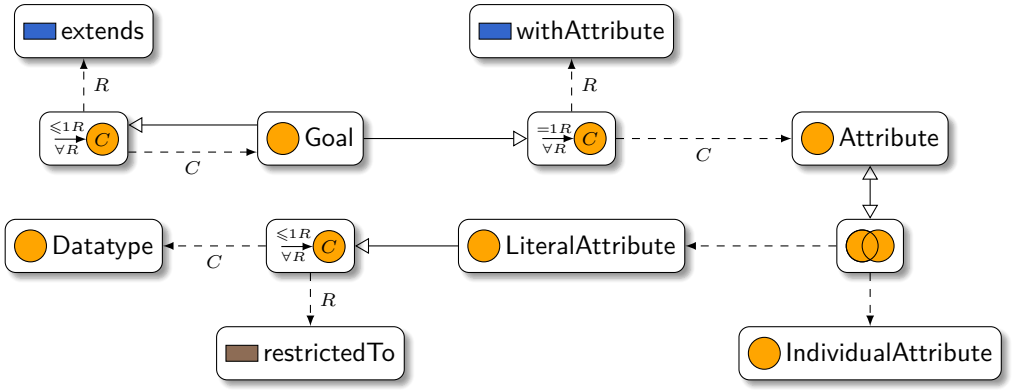


Abbildung 4.7.: Taxonomie abstrakter Ziele als Individuen- und Literalattributstupel.

beschreiben lässt. Zusätzlich wurden abstrakte Ziele eingeführt, die zur Modularisierung der Management-Logik genutzt werden können. In diesem Abschnitt wird die ontologische Taxonomie der Ontology-based Management Language (OntML) entwickelt, mit der sich die Management-Logik unter direktem, semantischem Bezug auf die Entitäten des Domänen- und Schnittstellenmodells als Ontologie beschreiben lässt. Die Abbildungen nutzen die in Abschnitt A.1 (siehe Seite 203) dargestellte, grafische Notation; Beispiele der vorgestellten Konzepte können in Abschnitt A.2.5 (siehe Seite 227) gefunden werden.

Die oberste Gliederungsebene von OntML bilden sogenannte Management-Module (Module, Abbildung 4.6). Sie fassen verwandte Verarbeitungsvorschriften und Teilziele zu referenzierbaren Fragmenten zusammen. Ein Modul ist ein benanntes OWL-Individuum, das durch seinen IRI eindeutig identifizierbar ist. Es setzt sich aus beliebig vielen Elementen der disjunkten Vereinigung von abstrakten Zielen (Goal) und Verarbeitungsregeln (Rule) zusammen. Module können voneinander abhängen und explizite Abhängigkeiten an die abstrakten Ziele und Regeln der referenzierten Module haben.

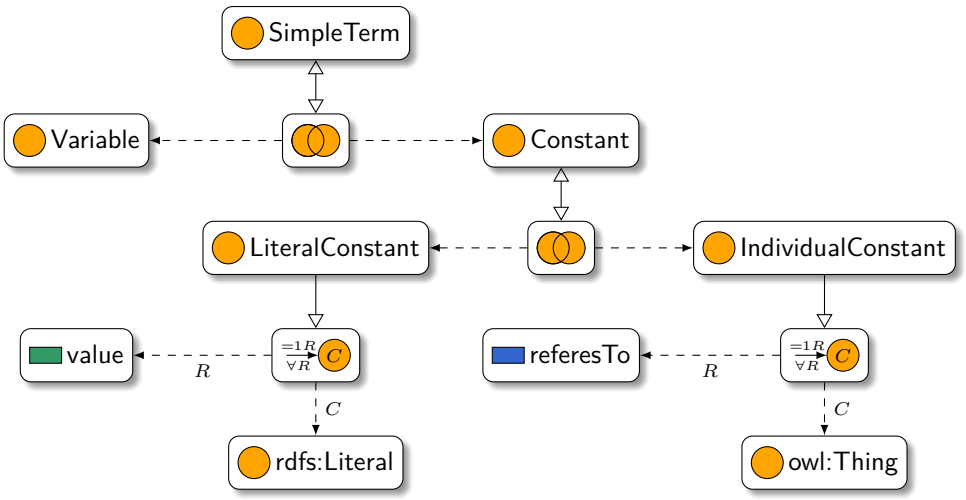


Abbildung 4.8.: Taxonomie einfacher Terme als disjunkte Vereinigung von Variablen und Konstanten.

Abstrakte Ziele (siehe Abb. 4.7) sind rekursiv definierte Tupel, deren Komponenten Individuen oder Literale sein können. Ein abstraktes Ziel kann von maximal einem abstrakten Ziel erben und es um genau ein Attribut erweitern. Attribute sind die disjunkte Vereinigung von Individuen- und Literalattributen (Individual und Literal Attribute). Eine Zielinstanz hält in einer mit einem Individuenattribut typisierten Komponente ein Individuum der Problemdomäne und in einer mit einem Literalattribut typisierten Komponente einen literalen Wert. Der von einem Literalattribut repräsentierte Wert kann auf einen RDF-Datentyp (Datatype) eingeschränkt werden. Da in OWL das Punning von Datentypen verboten ist, kann dieser nicht über eine abstrakte Rolle, sondern nur über eine Annotation referenziert werden. Daher kann die in der Abbildung dargestellte Einschränkung in dieser Form nicht ausgedrückt werden, sie ist rein informativ und hat keine Modellrepräsentation.

Regeln werden in OntML mit Individuen und Beziehungen modelliert, sodass auf Modellebene semantischer Bezug auf importierte Entitäten der betrachteten Domäne genommen werden kann. Die Regelstruktur und -semantik wird dabei nicht „auf der grünen Wiese“ entwickelt, sondern als RIF-Dialekt (siehe Abschnitt 2.4.2, Seite 36) aus den Fragmenten des RIF Framework for Logic Dialects (RIF-FLD) konstruiert.

Den Grundbaustein von Regeln bilden Terme. Als einfache Terme (Simple Term, Abbildung 4.8) bezeichnet man Variablen (Variable) und Konstanten (Constant), aus denen komplexere Terme zusammengesetzt werden können. OntML adaptiert dieses Konzept, der in RIF sehr allgemein gehaltene Konstantenbegriff wird jedoch für die ontologische Domäne eingeschränkt. Konstanten sind in OntML die disjunkte Vereinigung von Individuenkonstanten (Individual Constant) und Literalkonstanten

(Literal Constant). Über Individuenkonstanten können Individuen der Problemdomäne referenziert und so in Regeln eingebunden werden. Mit Literalkonstanten lassen sich in Regeln skalare Werte ausdrücken, die keine direkte Repräsentation im Domänenmodell haben (z. B. der Schwellenwert einer Antwortzeit). Variablen sind Stellvertreter, die an Werte gebunden und über ihren IRI eindeutig identifiziert und referenziert werden können.

Der Begriff Basisterme steht in RIF stellvertretend für die disjunkte Vereinigung aus einfachen Termen, Positionstermen und externen Termen. Positionsterme werden verwendet, um in der Wissensbasis Instanzen eines bestimmten Prädikatensymbols zu binden (z. B. eine Person, deren erste Komponente der Vor- und zweite Komponente den Nachnamen ist). Externe Terme repräsentieren Positionsterme, die außerhalb der Wissensbasis leben.

Auf Grund der Open-World-Assumption lassen sich Positionsterme im direkten Ontologiekontext nur sehr eingeschränkt nutzen; die Zugehörigkeit eines Individuums zu einer Klasse sagt nur wenig über dessen Struktur aus. So kann für die OWL-Klasse Person zwar die Einschränkung definiert werden, dass deren Instanzen einen Vor- und einen Nachnamen haben müssen, dies schließt jedoch weder aus, dass noch weitere Attribute wie ein Alter existieren, noch legt es fest, dass der Vorname das erste und der Nachname das zweite Attribut einer Klasseninstanz sind. Die einzige sinnvolle Einsatzmöglichkeit von Positionstermen im direkten Ontologiekontext besteht darin, einzelne RDF-Tripel zu binden (`rdf:type(?x, :Person)`, `ex:name(?x, ?y)`, etc.), dies wird jedoch auf Grund der später vorgestellten und kompakteren Frames in OntML nicht verfolgt.

Abseits der ontologischen Wissensbasis finden Positionsterme (Positional Term) in OntML jedoch sehr wohl Anwendung. Über sie werden abstrakte Ziele, Events (Event) und Aktionen (Action) gebunden und externe Funktionen (Function) genutzt (siehe Abb. 4.9). Abstrakte Ziele sind ohnehin als Tupel mit abgeschlossener und geordneter Attributsmenge definiert, Events und Aktionen haben genau zwei Attribute: die RDF-Ressource, die sie repräsentieren und das die Ressource beschreibende RDF-Dokument. Funktionsaufrufe werden genutzt, um vom Framework bereitgestellte Funktionen aufzurufen. Als Basis stehen dabei die in RIF-Build-Ins (RIF-DTB) [97] beschriebenen Funktionen (mathematische Operationen, Vergleichsoperationen, etc.) bereit. Über eine spezielle Blank-Node-Funktion kann ein deterministischer Ressourcenname abgeleitet werden, indem ein Hash-Wert über die übergebenen Parameter gebildet wird. Dies ist besonders beim Hinzufügen von Ressourcen hilfreich, die noch keine Modellrepräsentation haben (z. B. Discovery einer neuen Komponente zur Laufzeit). Zusätzlich sollen später anwendungsfallspezifische Funktionen

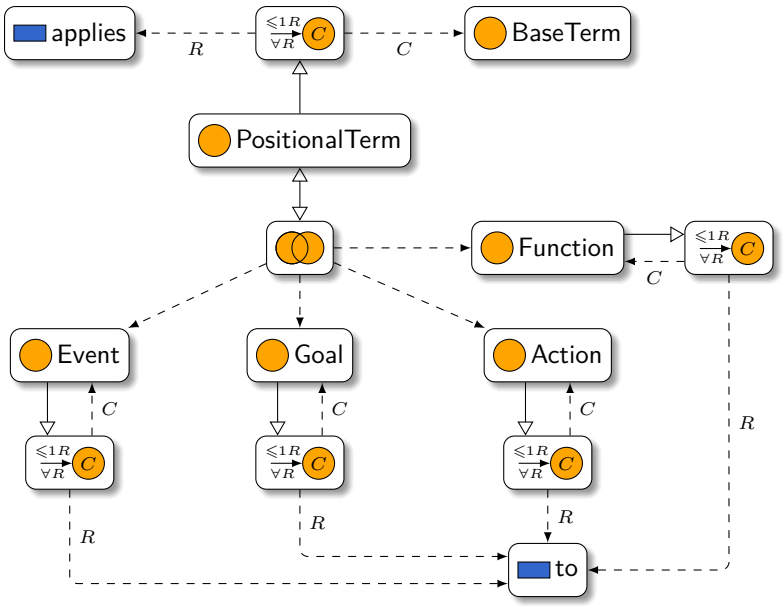


Abbildung 4.9.: Taxonomie von Positionstermen, die als wiederholte partielle Anwendung von Basistermen auf Events, Aktionen, Ziele oder Funktionen modelliert werden.

eingebunden werden können, indem sie als Teil des Domänenmodells ontologisch beschrieben und über eine Framework-Schnittstelle implementiert oder als aufrufbare OWL-S-Dienste [81] modelliert werden. Dieses Konzept ist in dieser Arbeit jedoch nicht umgesetzt.

Bei der Positionstermmodellierung werden in OntML Ziele, Events, Aktionen und Funktionen als gecurryt [11] aufgefasst und deren Stelligkeit durch wiederholte partielle Anwendung eines Basisterms auf Null reduziert. Events und Aktionen müssen dementsprechend partiell genau zweimal (die repräsentierte Ressource und dessen Dokument), Ziele und Funktionen n-mal, für n ist die Stelligkeit des Ziels/der Funktion, angewandt werden, um einen gültigen Positionsterm zu erzeugen. Ziele, Events, Aktionen und Funktionen sind disjunkt.

Aggregationsterme (siehe Abb. 4.10) sind eine spezielle Form externer Terme. Sie wenden eine einstellige Aggregationsfunktion (Summe, Mittelwert, Maximum, etc.) auf eine Wertmenge an und bilden sie so auf einen skalaren Wert ab. Aggregationen bilden die Grundlage vieler IT-Management-Kennzahlen; beispielsweise die mittlere Auslastung aller Knoten eines Clusters oder die maximale Antwortzeit. Sie sind unter der Open-World-Assumption unentscheidbar und dürfen daher in direkt in den Reasoning-Prozess integriert Sprachen wie SWRL nicht verwendet werden. Da die Management-Logik des Frameworks jedoch oberhalb des Reasonings

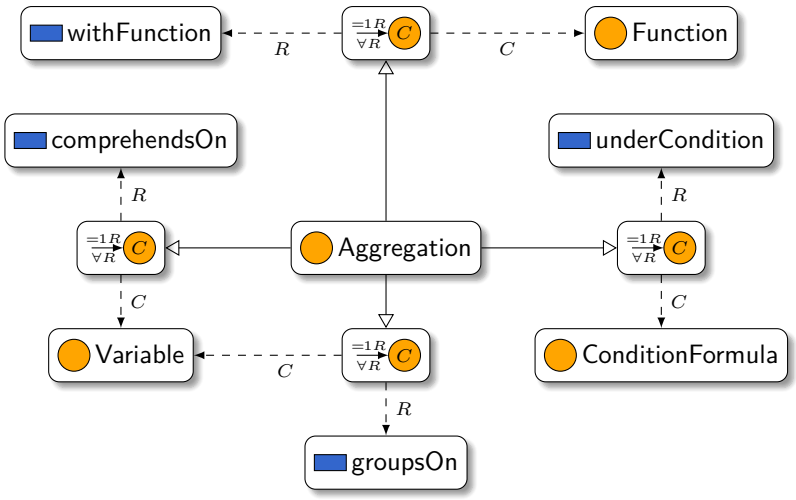


Abbildung 4.10.: Taxonomie von Aggregationstermen, bestehend aus Aggregationsfunktion, einer Variablen und einer Bedingung.

angesiedelt wird und alle abgeleiteten Fakten explizit materialisiert werden, können Aggregationen Teil von OntML sein.

Ein Aggregationsterm (Aggregation Term) setzt sich aus einer Aggregationsfunktion, einer Aggregationsvariablen und mehreren Gruppierungsvariablen zusammen. Die zu aggregierenden Werte werden von einer Bedingungsformel (Condition Formula, später erläutert) beschrieben, die die Aggregations- und die Gruppierungsvariablen bindet. Aus allen für die Formel gültigen Aggregationsvariablenbelegungen wird die Wertmenge gebildet und die Aggregationsfunktion darauf angewandt.

Mit Frames (siehe Abb. 4.11) lassen sich in RIF Objekte über eine ungeordnete Menge von Charakteristiken beschreiben. Sie eigenen sich damit sehr gut, OWL-Individuen zu charakterisieren (vgl. [25]) und werden in OntML ausschließlich zu diesem Zweck verwendet. Ein Frame besteht aus einem das Kontextindividuum repräsentierenden einfachen Term und einer Menge von Charakteristiken (Characteristic). Charakteristiken beschreiben das Individuum anhand von Klassenzugehörigkeiten (Class Characteristic) und dessen Beziehung zu Individuen und Literalen (Role Characteristic).

Eine Klassencharakteristik referenziert eine Domänenklasse und drückt aus, dass das Kontextindividuum Element der Klasse ist. Musste bei der Beschreibung der abstrakten Ziele noch eine Annotation genutzt werden, um den Literalattributsdatentyp zu referenzieren, kann hier von der OWL-2-Metamodellierungsfähigkeit profitiert werden. Durch Punning darf eine RDF-Ressource gleichzeitig OWL-Klasse (oder -Rolle) und -Individuum sein kann. Daher kann eine abstrakte Rolle genutzt

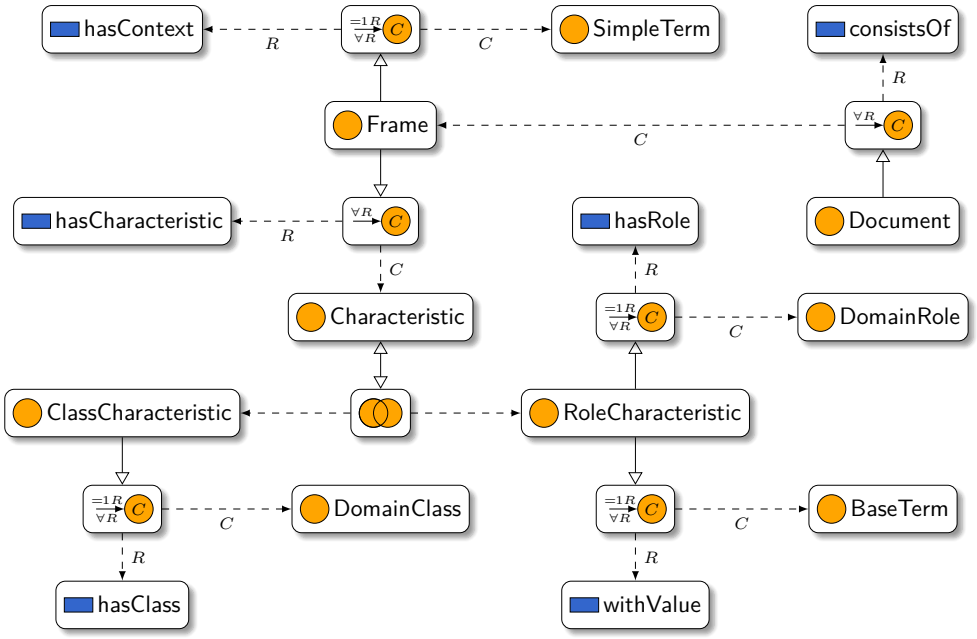


Abbildung 4.11.: Taxonomie von Frames, die Individuen anhand ihrer Klassen und Rollen beschreiben.

und deren Wertebereich auf die synthetische Domänenklasse (Domain Class) beschränkt werden, wenn die Klasse referenziert wird. Dies führt wiederum dazu, dass beim Reasoning des Management-Modells jede referenzierte Klasse zur Instanz der Domänenklasse wird. Analog zur Klassencharakteristik kann mit einer Rollencharakteristik ausgedrückt werden, dass das Kontextindividuum über eine konkrete oder abstrakte Rolle mit einem von einem Basisterm repräsentierten Individuum oder Wert in Beziehung steht. Dabei wird ebenfalls Punning eingesetzt, um die die Rolle referenzierende abstrakte Rolle auf die synthetische Domänenrollenklasse (Domain Role) einzuschränken.

Frames können in OntML in Dokumentterme eingebettet werden, die als Parameter von Event- oder Aktionstermen genutzt werden. Dadurch wird ausgedrückt, dass der Frame nicht im zentralen Informationsmodell, sondern im das Event oder die Aktion repräsentierenden Dokument gebunden werden soll.

Den letzten in OntML eingesetzten Termtyp stellen die Äquivalenzterme (Equality Term, Abbildung 4.12) dar. In RIF können damit beliebige Basisterme gleichgesetzt und eine Unifikation höherer Ordnung erzielt werden. OntML beschränkt Äquivalenzen auf Unifikation erster Stufe, sodass weiterhin Einschränkungen (Guard) und Zuweisungen (Assignment) ausdrückbar sind. Bei einer Einschränkungen wird die linke Hand implizit als *wahr* angenommen ($true = greaterThan(x, y)$). Eine

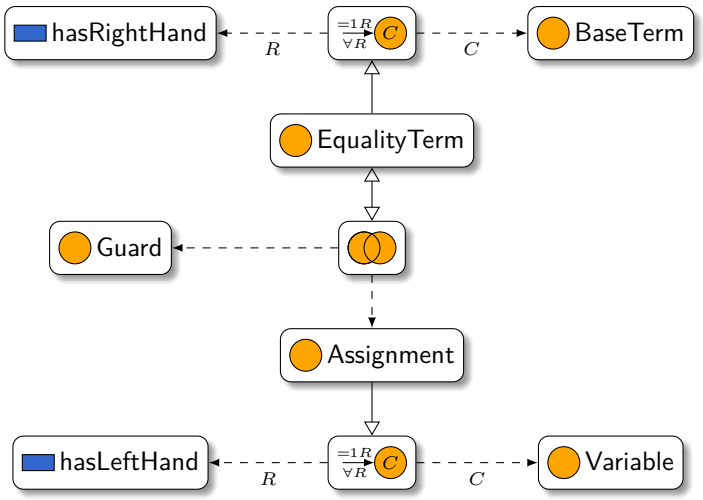


Abbildung 4.12.: Taxonomie von Äquivalenztermen, deren linke Hand ein einfacher Term und rechte Hand ein Basisterm sein kann.

Zuweisung hat als linke Hand eine Variable, die dem Basisterm der rechten Hand gleichgesetzt wird ($x = add(y, 5)$).

Die *temporale Verarbeitung* stellt einen integralen Bestandteil der Management-Logik dar. In RIF können Terme nur indirekt zeitlich zueinander in Relation gesetzt werden, indem explizit über Äquivalenzterme Einschränkungen auf Termattributen festgelegt werden. In OntML soll die temporale Verarbeitung hingegen integraler Sprachbestandteil sein. Dafür existieren sogenannte temporale Einschränkungen, die an Instanzen bestimmter OntML-Klassen angebracht werden können.

Eine temporale Einschränkung (Temporal Restriction, Abbildung 4.13) ist die disjunkte Vereinigung von Zeitfenstern (Time Window, z. B. in den letzten 20 Minuten), relativen Zeitpunkten (Relative Instant, z. B. vor 3 Tagen) und Allen-Relationen zu anderen temporalen Termen (Allen Relation, z. B. 10 Sekunden *nach* Event x). Die Klasse der dabei eingesetzten Allen-Operatoren ist äquivalent zu einer Aufzählungsklasse, die Individuen für alle in Abbildung 2.4 gezeigten Operatoren umfasst, die jedoch aus Platzgründen in der Abbildung nicht dargestellt sind.

Temporale Einschränkungen können in Frames an Klassifikations- und Rollencharakteristiken angebracht werden, wenn es sich bei der referenzierten Klasse um eine temporale Klasse (Unterklasse von `fluent:Class`) oder eine temporale Rolle (Unterrolle von `fluent:ObjectProperty` oder `fluent:DataProperty`) handelt. Bei der Regelverarbeitung werden dann in der Fluent-Ressource-Struktur die definierten zeitlichen Einschränkungen berücksichtigt. Darüber hinaus können Event-Positi-

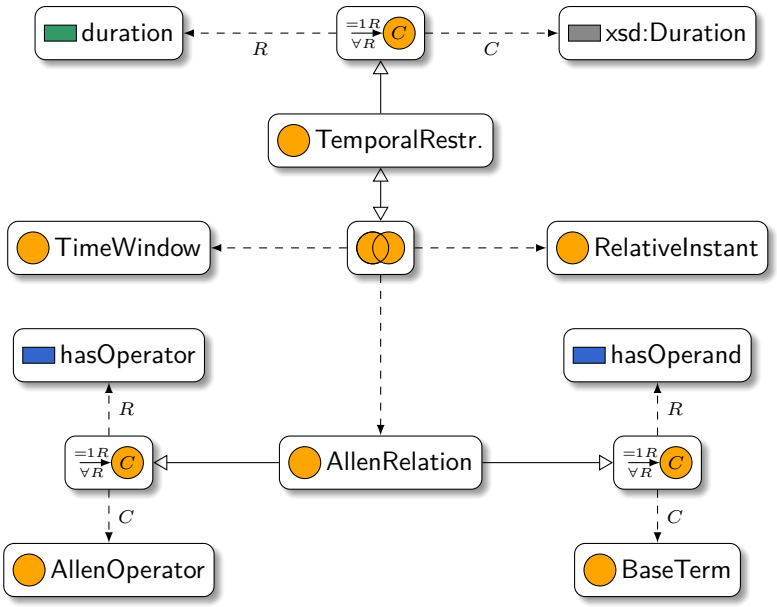


Abbildung 4.13.: Taxonomie von für Events und Charakteristiken verwendbaren temporalen Einschränkungen als disjunkte Vereinigung von relativen Zeitfenstern, relativen Zeitpunkten und Allen-Einschränkungen.

onsterme temporal eingeschränkt und so eine Complex-Event-Processing-Semantik ausgedrückt werden.

Das Bindeglied zwischen Termen und Regeln bilden Formeln (siehe Abb. 4.14). Sogenannte Bedingungsformeln (Condition Formula) setzen sich aus atomaren und komplexen Formeln zusammen. Atomare Formeln (Atomic Formula) sind die Schnittstelle zur Termebene und in OntML äquivalent zur Vereinigung von Positionstermen, Frames und Äquivalenztermen. Komplexe Formeln (Complex Formula) sind die disjunkte Vereinigung aus eingebetteten Formeln (Nested Formula) und Verknüpfungsformeln (Connection Formula). Eingebettete Formeln haben genau eine Unterformel und sind die disjunkte Vereinigung aus existenzquantifizierten Formeln (Existentials) und Negationen (Negation). Existenzquantifizierte Formeln definieren freie Variablen, die in der Unterformel gebunden werden können; Negationen drücken aus, dass für eine eingebettete Unterformel keine Resolution möglich sein darf. Verbindungsformeln verknüpfen genau zwei Unterformeln und sind die disjunkte Vereinigung aus Konjunktionen (Conjunction, Und-Verknüpfungen) und Disjunktionen (Disjunction, Oder-Verknüpfungen).

In OntML (siehe Abb. 4.15) existieren Implikations- (Implication Rule) und Produktionsregeln (Production Rule). Beide Regelarten setzen sich aus einer Menge universalquantifizierter Variablen, einer Bedingung und einer Konsequenz zusammen. Die Bedingung ist eine Bedingungsformel, in der die Regelvariablen gebunden

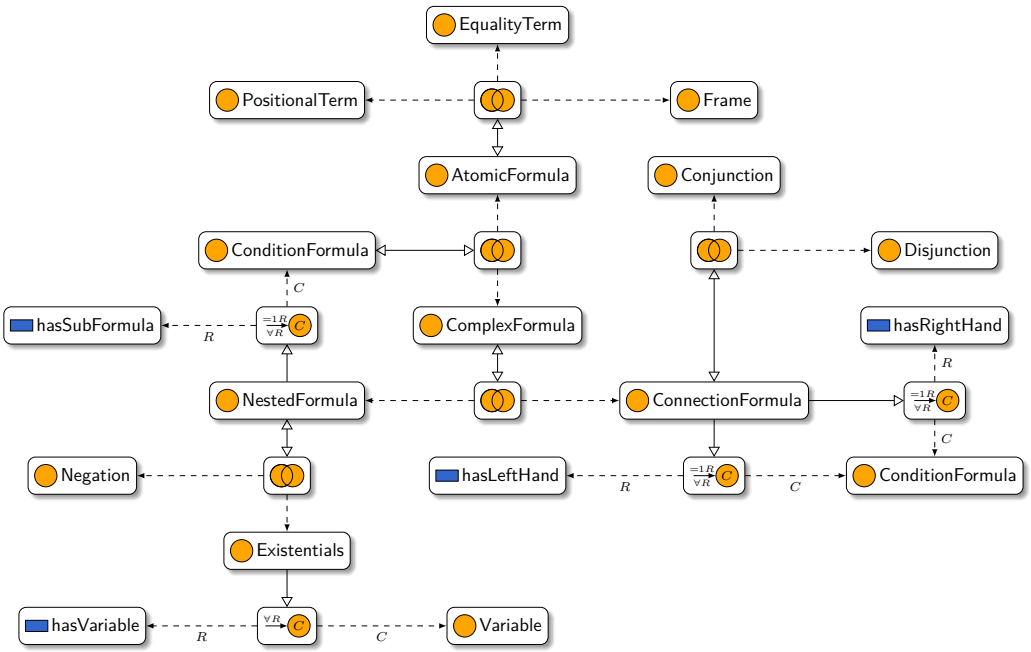


Abbildung 4.14.: Taxonomie von Bedingungsformeln als disjunkte Vereinigung von atomaren und komplexen Formeln.

werden. Für jede gültige Variablenbelegung wird die Regelkonsequenz aktiviert. Die Implikationskonsequenz setzt sich aus Frames und Positionstermen zusammen, die solange gültig sind, wie auch die Bedingung gültig ist. Die Produktionskonsequenz umfasst verschiedene Aktionen (Action), die einmalig ausgeführt werden, wenn die Regel aktiviert wird. Eine Aktion ist die disjunkte Vereinigung (in der Abbildung nicht dargestellt) aus atomaren Aktionen (Atomic Action) und zusammengesetzten Aktionen (Compound Action). Atomare Aktionen können einen Positionsterm erzeugen (Assert Atomic), entfernen (Retract Atomic) oder ausführen (Execute Atomic), zusammengesetzte Aktionen einen Frame erzeugen (Assert Compound) oder entfernen (Retract Compound).

Für jeden aus den Anforderungen abgeleiteten Regeltyp existiert in OntML eine Implikations- oder Produktionsregelspezialisierung. Event-Korrelationen, Event-Abbildungen und Policies sind Produktionsregelspezialisierungen. Sie alle binden in ihrer Bedingung Wissen aus der Wissensbasis und dem Event-Datenstrom und leiten daraus Aktionen ab. Eine Event-Korrelation darf als Konsequenz ausschließlich Event-Positionsterme zum Datenstrom hinzuzufügen oder daraus entfernen. Eine Event-Abbildung darf als Konsequenz ausschließlich Frames zur Wissensbasis hinzufügen und so die im gebundenen Event beschriebenen Änderungen auf die Wissensbasis abbilden. Policies dürfen als Konsequenz ausschließlich Aktionspositionsterme ausführen, um das überwachte System aktiv zu beeinflussen.

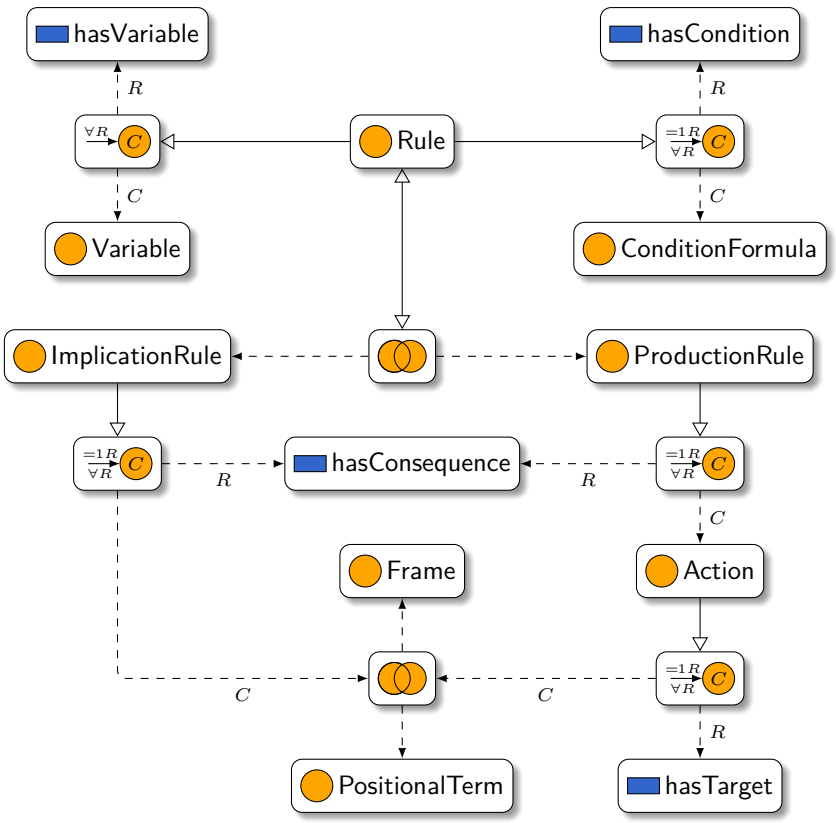


Abbildung 4.15.: Taxonomie von Regeln als disjunkte Vereinigung von Implikations- und Produktionsregeln.

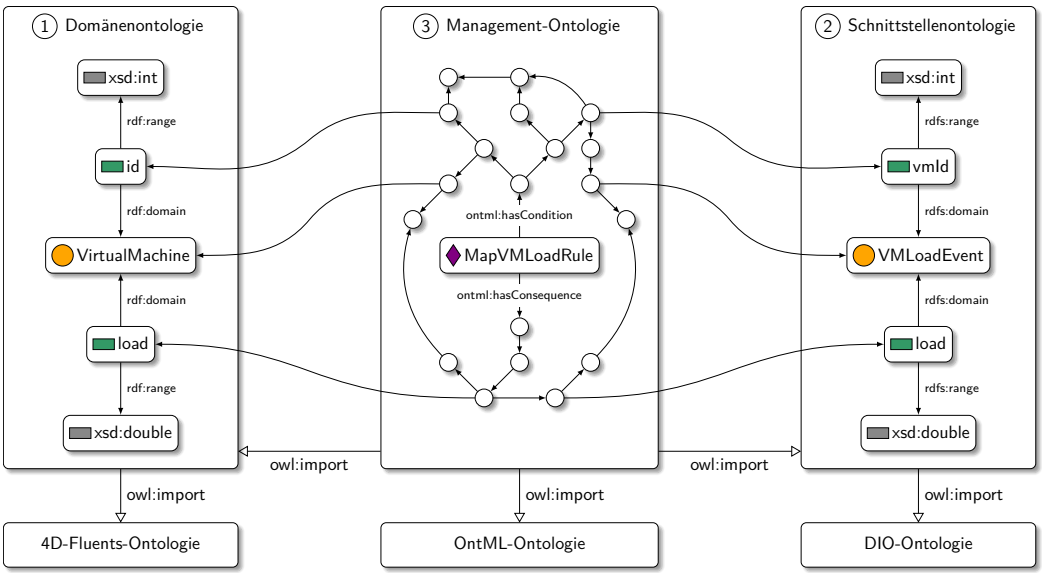


Abbildung 4.16.: Zusammenspiel der unterschiedlichen Ontologien am Beispiel von Virtual Machine Management.

Funktionale Abhängigkeiten und Abfragen sind Implikationsspezialisierungen. Funktionale Abhängigkeiten binden als Bedingung ausschließlich Wissensbasisfakten und implizieren daraus Frames. Abfragen implizieren Ziel-Positionsterme und beschreiben so, wie als abstrakte Ziele formulierte Teilberechnungen durchgeführt werden. Eine Abfrage darf in der Bedingung Wissen aus der Wissensbasis und dem Event-Datenstrom binden.

OntML nutzt ausschließlich Konzepte aus dem RIF-FLD [20] und unterliegt der darin definierten Semantik. Die temporalen Erweiterungen sind reine Syntaxerweiterungen, die sich für Events auf Äquivalenzterme und für Fluents auf die Fluents-Strukturen bindende Frames und Äquivalenzterme umschreiben lassen.

Beispielszenario

Abbildung 4.16 zeigt beispielhaft, wie die Ontologiestruktur eines Virtual Machine Managements aussehen könnte. In der ① Domänenontologie sind die Entitäten der betrachteten Domäne modelliert. `VirtualMachine` ist die Klasse der Virtuellen Maschinen, die konkrete Rolle `id` ordnet jeder Virtuellen Maschine einen eindeutigen Bezeichner zu, die `load`-Datenrelation gibt die aktuelle Auslastung der Virtuellen Maschine an. Die `load`-Relation ist temporal, erbt also von der entsprechenden 4D-Fluents-Relation (in der Abbildung nicht dargestellt).

Die ② Schnittstellenontologie definiert das vom überwachten System zur Laufzeit empfangene Event, das die aktuelle Auslastung einer Virtuellen Maschine enthält. Das `VMLoadEvent` erbt von `ido:Event` und hat die Attribute `vmId`, in dem der Bezeichner der repräsentierten Virtuellen Maschine gehalten wird, und `load`, in dem die Auslastung der Maschine gehalten wird.

Die Verknüpfung der beiden bisher noch unabhängigen Ontologien findet in der ③ Management-Ontologie (siehe Abschnitt A.2.5, Seite 227) statt. Sie importiert die Domänen-, Schnittstellen- und OntML-Ontologie und nutzt die OntML-Entitäten, um eine Event-Abbildungsregel zu definieren. Die Bedingung und Konsequenz (beide in der Abbildung auf Grund der Komplexität nur abstrakt dargestellt) der Regel enthalten jeweils Terme, die die Entitäten der importierten Domänen- und Schnittstellenontologie referenzieren.

Die einheitliche, ontologiebasierte Definition der Domäne, der Schnittstelle und der Management-Logik ermöglicht eine übergreifende Sicht auf den Gesamtzusammenhang der unterschiedlichen Entitäten. Das entstehende, transparente Management-Modell wirkt der Wissensfragmentierung, die in anderen ontologiebasierten Management-Ansätzen vorherrscht, entgegen und ermöglicht es Administratoren und Domänenexperten, bereits zum Modellierungszeitpunkt komplexe Zusammenhänge zu verstehen und aus Änderungen resultierende Seiteneffekte besser abzuschätzen. So können Fragestellungen wie „Von welcher Regel werden Instanzen einer bestimmten Klasse erzeugt?“, „Welches Event ist Grundlage für den Wert einer konkreten Rolle im Informationsmodell?“, „In welchem funktionalen Zusammenhang stehen zwei Relationen?“ mit Hilfe von Modellabfragen (z. B. in SPARQL) beantwortet werden und die direkte Bezugnahme der Domänenentitäten in den Management-Regeln wirkt möglichen Inkonsistenzen zwischen den Modellwelten entgegen.

4.2.2 Datenhaltung

Um zur Laufzeit in der Lage zu sein, alle Informationen des überwachten Systems in der Wissensbasis vorzuhalten und dennoch reaktiv und in nahezu Echtzeit verarbeiten zu können, wird für das Framework ein Hybridansatz entwickelt, der die Vorteile eines dedizierten Triplestores und einer In-Memory-Datenhaltung vereint. Dabei übernimmt der Triplestore die Rolle der allumfassenden Wissensbasis, die eine vollständige Sicht auf das überwachte System bereitstellt. Die davon Management-relevante (die von der Verarbeitungslogik als Eingabe benötigte) Untermenge wird in den Hauptspeicher repliziert, sodass die Verarbeitungskomponenten des Regelkreises effizient darauf zugreifen können. Das In-Memory-Abbild entspricht dabei

prinzipiell der Idee eines Caches, setzt jedoch keine typischen Cache-Strategien wie Verdrängung, Invalidierung, etc. um. Die In-Memory zu haltenden Daten müssen dabei nicht explizit definiert werden, sondern leiten sich implizit aus dem Management-Modell und der Informationsmodelltaxonomie ab. Beide sind zur Laufzeit statisch, sodass vor Eintritt in die Laufzeitphase, jedoch nachdem alle für einen Anwendungsfall benötigten Ontologien in die Wissensbasis geladen wurden, eine zweistufige Modellanalyse durchgeführt werden kann, die alle relevanten Entitäten aufdeckt.

In der ersten Analysestufe werden alle explizit vom Management-Modell referenzierten Entitäten bestimmt. Dazu wird die Wissensbasis einem Ontologie-Reasoning unterzogen, bei dem die OWL-RL-Standardregeln zum Einsatz kommen. Die auf dem Management-Modell definierten Einschränkungen führen dazu, dass alle von Frames über Rollencharakteristiken referenzierten Rollen als Domänenrolle und alle über Klassencharakteristiken referenzierte Klassen als Domänenkonzepte klassifiziert werden. Sie werden extrahiert und bilden die Menge der explizit Management-relevanten Domänenentitäten. Diese wird um die Konzepte und Rollen der 4D-Fluents-Ontologie erweitert, da sie die Grundlage der temporalen Verarbeitung des Frameworks bilden.

Neben den explizit Management-relevanten Informationen werden im In-Memory-Abbild auch all die Informationen gehalten, aus denen implizit, mittels OWL-Reasoning, Management-relevante Daten abgeleitet werden können. Dies ermöglicht es, dass der der Verarbeitung vorgeschaltete Reasoning-Prozess ausschließlich auf dem reduzierten In-Memory-Abbild arbeiten muss. Dazu wird in der zweiten Analysestufe eine transitive Hülle gebildet, die alle benötigten Entitäten umfasst. Dabei werden die im OWL-RL-Standard definierten Ableitungsregeln eingesetzt. Aufgrund der statischen Informationsmodelltaxonomie und dem bereits durchgeführten Reasoning müssen bei der Analyse nur die Regelsätze zur *Semantik der Rollenaxiome* (The Semantics of Axioms about Properties), *Semantik der Klassenaxiome* (The Semantics of Class Axioms) und *Semantik der Klassen* (The Semantics of Classes) aus [89] berücksichtigt werden.

Um die Klassenhülle zu ermitteln, wird ein gerichteter Graph aller Klassen des geschlussfolgerten Informationsmodells aufgebaut. Darin existiert eine Kante von c_1 nach c_2, gdw.

- c_2 eine Unterklasse von c_1 ist (Regel *cax-sco*),
- c_2 äquivalent zu c_1 ist (*cax-eqc*),
- c_2 eine Schnittmenge über c_1 ist (*cls-int*),

- c_1 eine Vereinigung über c_2 ist (*cls-uni*),
- c_1 eine existenzeingeschränkte Klasse mit Zielklasse c_2 ist (*cls-svf*) oder
- c_2 eine universaleingeschränkte Klasse mit Zielklasse c_1 ist (*cls-avf*).

Nachdem der Klassengraph aufgebaut wurde, kann ausgehend von den explizit Management-relevanten Klassen mit einer Breitensuche die transitive Hülle TC gebildet werden.

Für die Rollenhülle wird ebenfalls ein gerichteter Graph aufgebaut, der alle Rollen des Informationsmodells umfasst. In ihm existiert eine Kante von r_1 nach r_2 gdw.

- r_2 eine Unterrolle von r_1 ist (*prp-spo1*),
- r_2 Kettenglied der Rollenkette r_1 ist (*prp-spo2*),
- r_2 äquivalent zu r_1 ist (*prp-eqp*) oder
- r_2 invers zu r_1 ist (*prp-inv*).

Nachdem der Rollengraph aufgebaut wurde, kann auch hier die transitive Hülle TR gebildet werden. Die Eingabemenge umfasst über die explizit Management-relevanten Rollen hinaus auch all die Rollen, aus denen Individuenäquivalenz oder die Zugehörigkeit zu einer relevanten Klasse geschlussfolgert werden kann. Eine Rolle r ist Teil der Eingabemenge, gdw.

- r eine explizit Management-relevante Rolle ist,
- r eine funktionale Rolle ist (*prp-fp*),
- r eine invers-funktionale Rolle ist (*prp-ifp*),
- r eine Schlüsselrolle ist (*prp-key*),
- r Prädikat einer maximalen Kardinalitätseinschränkung von eins ist (*cls-maxc*),
- r als Definitionsbereich eine Klasse $c \in TC$ hat (*prp-dom*),
- r als Wertebereich eine Klasse $c \in TC$ hat (*prp-rng*),
- r Prädikat einer existenzeingeschränkten Klasse $c \in TC$ ist (*cls-svf*),
- r Prädikat einer Universaleinschränkung auf $c \in TC$ ist (*cls-avf*) oder
- r Prädikat einer Werteingeschränkten Klasse $c \in TC$ ist (*cls-hv*).

Aus den resultierenden Rollen- und Klassenhüllen wird ein Filter gebildet, der zur Laufzeit entscheidet, ob ein Tripel In-Memory gehalten werden soll. Ein Tripel $(s\ p\ o)$ ist dann Element der Management-relevante Untermenge gdw. $p \in TR$ oder $o \in TC \wedge p = \texttt{rdf:type}$. Zwar existiert so ein potentieller Informations-Overhead, da auch Tripel In-Memory gehalten werden, die auf Grund ihres Kontextes niemals von

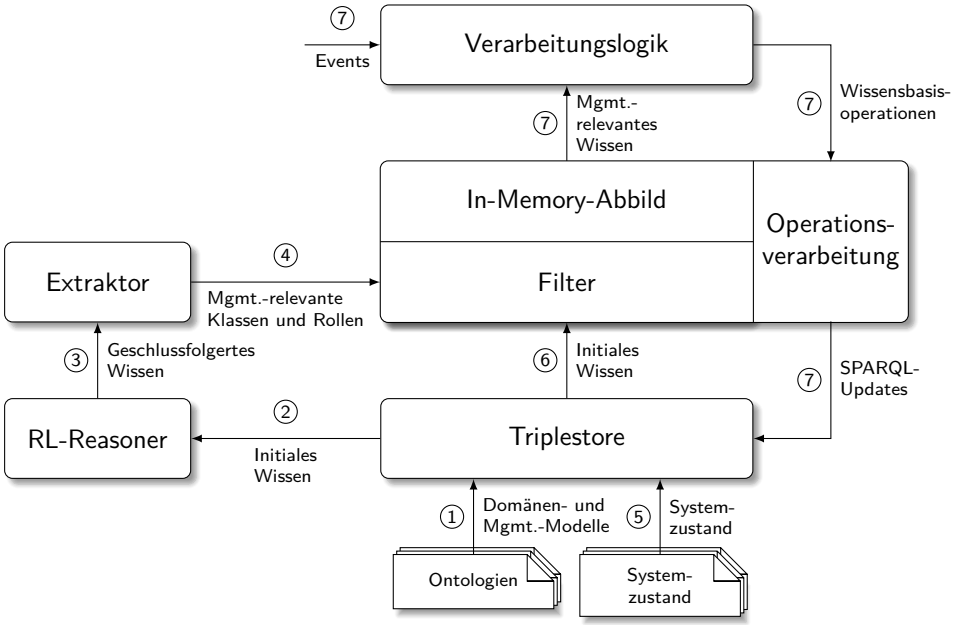


Abbildung 4.17.: Komponenten, Ablauf und Datenfluss der hybriden Wissensbasis.

der Management-Logik verarbeitet werden, eine Datenunvollständigkeit kann jedoch ausgeschlossen und die Filterung zur Laufzeit sehr effizient umgesetzt werden.

Abbildung 4.17 zeigt die Komponenten, den Ablauf und den Datenfluss der hybriden Wissensbasis. Beim Start wird der Triplestore mit den anwendungsfallspezifischen Domänen- und Management-Modellen befüllt ①. Anschließend werden die geladenen Daten von einem Reasoner verarbeitet ②, der ein vollständiges OWL-RL-Reasoning durchführt. Das geschlussfolgerte Wissen wird einem Extraktor bereitgestellt ③, der das beschriebene Verfahren nutzt, um die explizit Management-relevanten Klassen und Rollen zu extrahieren und deren transitive Hülle zu bilden. Die resultierende Entitätsmenge nutzt das In-Memory-Abbild ④, um Filterregeln zu erzeugen.

Zu Beginn der Laufzeitphase wird der initiale Systemzustand in den Triplestore importiert ⑤ und das In-Memory-Abbild mit denen dem Filter genügenden Tripeln befüllt ⑥. Zur Laufzeit ⑦ werden Wissensbasisänderungen ausschließlich von der Management-Logik getrieben, indem Events interpretiert und funktionale Abhängigkeiten aufgelöst werden. Die daraus resultierenden Wissensbasisoperationen beschreiben, welche Tripel zur Wissensbasis hinzugefügt oder daraus entfernt werden müssen. Für jede Operation werden das In-Memory-Abbild und der Triplestore aktualisiert. Der Triplestore setzt alle Wissensbasisoperationen, das Abbild nur Management-relevante Daten anpassende Operationen um.

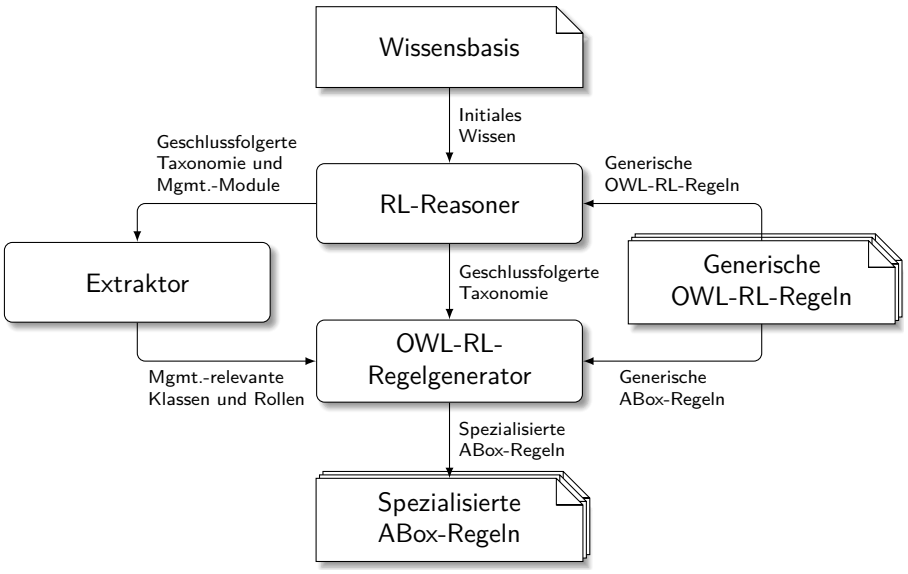


Abbildung 4.18.: Generierung domänenspezifischer ABox-Regeln durch Spezialisierung der OWL-RL-ABox-Regeln.

4.2.3 Reasoning

Der in Abschnitt 4.2.2 entwickelte hybride Datenhaltungsansatz, bei dem über die explizit Management-relevanten Tripel hinaus auch all die Tripel im In-Memory-Abbild gehalten werden, aus denen beim OWL-RL-Reasoning implizit Management-relevante Daten abgeleitet werden können, erlaubt es, dass das kontinuierliche Reasoning ausschließlich auf dem In-Memory-Abbild durchgeführt werden muss. Da die Informationsmodelltaxonomie zur Laufzeit des Frameworks statisch und das TBox-Reasoning unabhängig von den Instanzdaten ist, reicht es aus, initial ein einmaliges TBox-Reasoning durchzuführen, bei dem alle Fakten über die Taxonomie abgeleitet werden. Das der Verarbeitungslogik vorgeschaltete Reasoning muss dann zur Laufzeit ausschließlich die ABox-Semantik umsetzen. Der OWL-RL-Standard definiert dafür eine Menge generischer Regeln, die Taxonomie und Instanzdaten verknüpfen und daraus neue Instanzdaten ableiten.

In diesem Abschnitt wird ein neues Verfahren (siehe Abb. 4.18) entwickelt, das die Einschränkung der zur Laufzeit statischen Taxonomie dazu nutzt, die generischen ABox-Regeln anwendungsfallspezifisch zu konkretisieren, indem der taxonomische Regelanteil bereits vorverarbeitet wird. Dies führt zum einen dazu, dass die Taxonomie zur Laufzeit nicht im In-Memory-Abbild gehalten werden muss, zum anderen, dass die Regeln die Taxonomietripel nicht dynamisch binden müssen und so der Verarbeitungsaufwand reduziert wird.

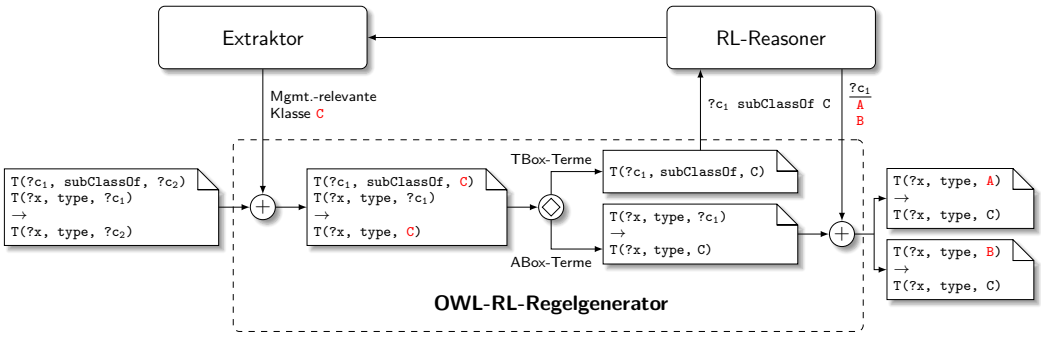


Abbildung 4.19.: Beispielhafte Regelspezialisierung durch Verknüpfung der Unterklassenregel (cax-sco) mit der Management-relevanten Klasse `C`.

Das Verfahren setzt auf dem in Abschnitt 4.2.2 beschriebenen Extraktionsmechanismus auf. Es verknüpft die darin im Reasoning-Schritt abgeleitete Taxonomie und die extrahierten Klassen und Rollen mit den OWL-RL-Regeln zur *Semantik von Rollenaxiomen* (The Semantics of Axioms about Properties), *Semantik von Klassenaxiomen* (The Semantics of Class Axioms), *Semantik von Klassen* (The Semantics of Classes) und *Semantik der Äquivalenz* (The Semantics of Equality) aus [89] und generiert daraus spezialisierte ABox-Regeln.

Die OWL-RL-Regeln sind logische Implikationen, deren Bedingung und Konsequenz Konjunktionen von Tripeltermen sind. Für jede Regel, die ein Klassifikationstripel (`rdf:type`) mit variabler Objektkomponente impliziert, wird für jede Management-relevante Klasse eine spezialisierte Regel erzeugt, in der die Klassenvariable durch die Klasse ersetzt ist. Für jede Regel, die ein Tripel mit variabler Prädikatenkomponente impliziert, wird für jede Management-relevante Rolle eine spezialisierte Regel erzeugt, in der die Rollenvariable durch die Rolle ersetzt ist. Jede spezialisierte Regel wird anschließend in ihre ABox- und TBox-Bestandteile zerlegt; die Konsequenz von ABox-Regeln enthält grundsätzlich nur ABox-Terme. Für die konjugierten TBox-Terme werden auf der geschlussfolgerten Wissensbasis alle Belegungskombinationen freier Variablen gesucht. Aus den ABox-Bestandteilen der Regel wird dann für jede gefundene Kombination eine Spezialisierung abgeleitet, in der die Variablen durch die Belegungen ersetzt sind.

Abbildung 4.19 zeigt die beispielhafte Anwendung des Verfahrens. Der Extraktionsalgorithmus schließt aus der abgeleiteten Wissensbasis die Management-Relevanz der Klasse `C`. Der Regelgenerator findet in den ABox-Regeln die Unterklassenregeln (cax-sco), die die Klassenzugehörigkeit (`rdf:type`) zu der freien Objektvariable $?c_2$ impliziert. Die Regel wird spezialisiert, indem $?c_2$ in allen Termen durch die Konstante `C` ersetzt wird. Anschließend wird die Spezialisierung in ihre TBox- und ABox-Bestandteile zerlegt und die TBox-Terme werden genutzt, um alle Belegungskombi-

nationen darin vorkommender freier Variablen zu finden. Als gültige Belegungen für $?c_1$ werden die Klassen `A` und `B` gefunden und durch Ersetzung der entsprechenden Variablen in den ABox-Bestandteilen der Regel zwei neue Regeln generiert, die jeweils die Klassenzugehörigkeit eines freien Individuums zu `C` implizieren, wenn das Individuum Instanz von `A` bzw. `B` ist.

4.2.4 Autonomes Management

Die in Abschnitt 3.1 (siehe Seite 44) vorgestellten verwandten IT-Operations-Management-Ansätze setzen ausschließlich ein automatisches Management um, bei dem der autonome Manager anhand fest definierter Regeln Entscheidungen trifft und Rekonfigurationen durchführt. Gerade bei komplexeren Systemen ist es jedoch schwierig, dabei alle globalen Effekte abzuschätzen und zu berücksichtigen. Daher soll in diesem Abschnitt ein autonomer Ansatz entwickelt werden, bei dem Management-Aktionen nicht von statisch formulierten Policies ausgelöst, sondern mit ihren Vorbedingungen und Effekten deklarativ beschrieben werden. Dies erlaubt es, sie unter Berücksichtigung der globalen Auswirkungen automatisch zu kombinieren und so den Gesamtsystemzustand zu optimieren. Die autonome Optimierung stellt eine optionale Erweiterung des Frameworks dar, ein automatisches Management kann auch ohne sie umgesetzt werden.

Optimierung ist ein eigenständiges Forschungsgebiet der angewandten Mathematik. Dabei werden in der Regel eine Zielfunktion und Nebenbedingungen definiert und versucht, die Systemparameter so anzupassen, dass die Zielfunktion unter Einhaltung aller Nebenbedingungen maximiert (oder minimiert) wird. Man unterscheidet in globale und lokale Optimierung; bei der globalen Optimierung wird das Optimum des gesamten Suchraums, bei der lokalen das relative Optimum in der Nachbarschaft des Ausgangszustands gesucht. Viele der existierenden Optimierungsverfahren sind auf ein konkretes Problemfeld beschränkt oder müssen mit tiefgreifendem Kontextwissen auf einen Anwendungsfall adaptiert werden. Ein IT-Operations-Management-Framework muss jedoch in der Lage sein, unterschiedlichste Anwendungsfälle umzusetzen, ohne einem Administrator eine komplexe Algorithmenparametrisierung abzuverlangen. Dabei ist es in der Regel auch nicht das Ziel, eine optimale Systemkonfiguration zu finden, vielmehr soll ausgehend von einem unerwünschten Systemzustand eine Operationskette gefunden werden, die mit möglichst wenigen Anpassungen Fehler beseitigt oder proaktiv unterbindet, ohne andere Systemaspekte negativ zu beeinflussen.

Für diese Art von Problem haben sich metaheuristische Optimierungsverfahren als effektiver Lösungsansatz erwiesen. Sie sind in der Lage, auch für unvollständige und unvollkommene Informationen mit eingeschränkten Berechnungsressourcen eine hinreichend gute Lösung zu finden [16]. Sie kombinieren abstrakte Schritte und wenden sie auf das Problem an, bis ein Terminierungskriterium erreicht wird. Für die lokale Suche haben sich dabei Hill-Climbing-Algorithmen als effizientes Mittel erwiesen, die ausgehend von einer initialen Lösung ein lokales Optimum annähern [17]. Dabei werden inkrementell die aktuell möglichen Schritte untersucht, der bestmögliche ausgewählt und von dort aus weiter optimiert.

Im zu entwickelnden Framework sollen Hill-Climbing-basierte Metaheuristiken eingesetzt werden, um den Systemzustand ohne tiefgreifendes Kontextwissen zu optimieren. Sie benötigen als Eingabe eine Aktionsquelle, die in Abhängigkeit des aktuellen Systemzustands ausführbare Schritte bereitstellt, Aktionsimplementierungen, die den Effekt von Aktionen auf das System beschreiben und Fitnessfunktionen, die den Systemzustand auf ein Bewertungskriterium abbilden. Um diese Aspekte beschreibbar zu machen, werden die Anforderungen an das Management-Modell um die folgenden Punkte erweitert:

A 6 Im Management-Modell muss beschrieben werden können, welche Management-Aktionen in Abhängigkeit des aktuellen Systemzustands durchführbar sind (Aktionsquelle).

A 7 Im Management-Modell muss beschrieben werden können, wie sich Management-Aktionen auf die Wissensbasis und den Event-Datenstrom auswirken (Aktionsimplementierung).

A 8 Im Management-Modell muss beschrieben werden können, wie der aktuelle Systemzustand zu bewerten und auf einen Qualitätsindikator abzubilden ist (Fitnessfunktion).

Dabei sollen bei der Aktionsimplementierung und Bewertung die anderen im Framework eingesetzten Verarbeitungsmechanismen berücksichtigt werden, sodass auch implizite Auswirkungen sichtbar sind. Dadurch wird es möglich, dass bei der autonomen Optimierung zunächst der initiale Systemzustand anhand der im Management-Modell definierten Fitnessfunktionen bewertet wird. Anschließend werden über die Aktionsbezugsquellen alle aktuell möglichen Aktionen extrahiert, über die Aktionsimplementierungen auf die Wissensbasis und den Event-Datenstrom angewandt, die impliziten Auswirkungen von der Event-Vorverarbeitung, Event-Interpretation

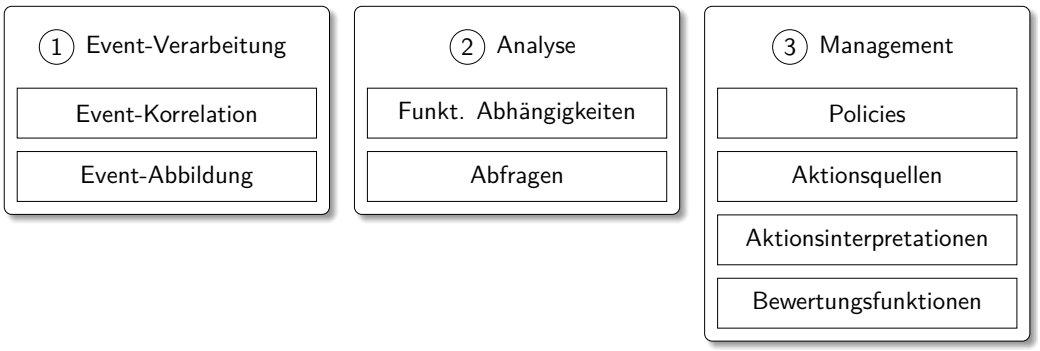


Abbildung 4.20.: Bestandteile des erweiterten Verarbeitungsmodells.

und den funktionalen Abhängigkeiten abgeleitet und der resultierende Zustand von den Fitnessfunktionen bewertet. Anschließend kann die beste Aktion ausgewählt und iterativ weiter optimiert werden. Erfüllt das Management-Modell all diese Anforderungen, kann die gesamte Verarbeitungslogik in einheitlicher Form und unter semantischer Bezugnahme auf das Informationsmodell beschrieben werden.

Die Management-Gruppe des Modells (siehe Abb. 4.5, Seite 85) wird um Aktionsquellen, Aktionsimplementierungen und Fitnessfunktionen erweitert (siehe Abb. 4.20), die das Kontextwissen für die Optimierung bereitstellen. Eine Aktionsquelle ist eine logische Implikation, die in Abhängigkeit des Event-Datenstroms und der Wissensbasis parametrisierte Aktionen beschreibt, die zum aktuellen Zeitpunkt auf dem überwachten System ausführbar sind. Die vorgeschlagenen Aktionen werden nicht auf dem verwalteten System ausgeführt, sondern als Eingabe des Optimierungsalgorithmus genutzt. Aktionsquellen sind definiert als

$$I_{source} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(M)\}. \tag{4.6}$$

Aktionsimplementierungen sind Analogien zu Event-Abbildungsregeln, beschreiben jedoch, wie sich eine Aktion auf die Wissensbasis auswirkt. Sie sind definiert als

$$P_{impl} = \{f \mid f : \mathcal{P}(M) \times \mathcal{P}(E) \times \mathcal{P}(K) \times T \to \mathcal{P}(A_F)\}. \tag{4.7}$$

Fitnessfunktionen sind Abfragespezialisierungen, die den Zustand des Event-Datenstroms und der Wissensbasis beurteilen und auf eine Bewertung abbilden. Eine Bewertung ist ein abstraktes Ziel in Form eines Paares ganzzahliger Skalare, die ein hartes und weiches Bewertungskriterium repräsentieren. Sei S die Menge aller Bewertungen, dann sind Bewertungsfunktionen definiert als

$$I_{score} = \{f \mid f : \mathcal{P}(E) \times \mathcal{P}(K) \times T \to S\}. \tag{4.8}$$

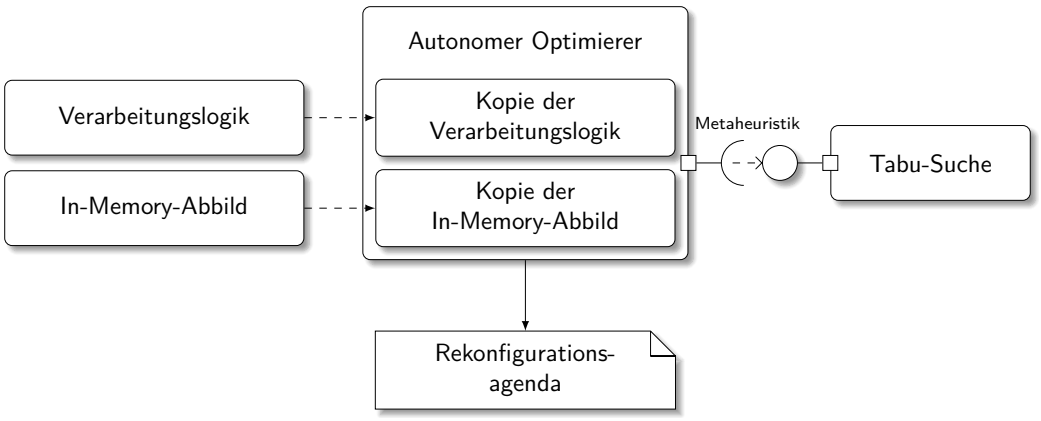


Abbildung 4.21.: Autonome Optimierung einer Wissensbasiskopie unter Einsatz einer lose gekoppelten Metaheuristik.

Um diese Aspekte im ontologischen Management-Modell ausdrücken zu können, wird dessen Taxonomie wie folgt erweitert: Es wird ein anwendungsfallunabhängiges abstraktes Ziel Bewertung (Score) eingeführt, das sich aus zwei ganzzahligen Literalattributen für den Hard- und Soft-Score zusammensetzt. Die Produktionsregeln werden um eine Aktionsinterpretationsspezialisierung (Action Interpretation) erweitert, die als Konsequenz Event- und Wissensbasisoperationen durchführen kann. Die Implikationsregeln werden um Aktionsquellen (Action Source) und Bewertungsfunktionen (Score Function) erweitert. Aktionsvorschläge dürfen als Konsequenz ausschließlich Aktionspositionsterme implizieren, Bewertungsfunktionen sind spezialisierte Abfragen, die als Konsequenz ausschließlich das abstrakte Bewertungsziel implizieren dürfen.

Die eingesetzte Metaheuristik ist über eine Schnittstelle vom Framework entkoppelt (siehe Abb. 4.21, Seite 107). Dadurch ist es möglich, die für den konkreten Anwendungsfall geeignetste Heuristik auszuwählen. Als Standardheuristik wird eine Tabu-Suche [53, 54] eingesetzt, die als Hill-Climbing-Spezialisierung auch leichte Verschlechterungen akzeptiert und Zustände als verboten (tabu) deklariert, um suboptimale Regionen und Plateaus zu überwinden. Die Optimierung wird parallel zum Policy-basierten Management durchgeführt. Dabei wird auf einer Kopie des In-Memory-Abbilds optimiert, bis eine zufriedenstellende Lösung gefunden oder eine vom Administrator definierbare Suchtiefe oder Laufzeit erreicht wurde. Nach Optimierungsabschluss wird das Ergebnis gegen die sich während der Optimierung weiterentwickelnde Ursprungswissensbasis validiert und anschließend dem Administrator zur Freigabe vorgelegt. Dieser entscheidet dann, ob die Agenda umgesetzt werden soll.

4.3 Management-Framework

Nach [73] ist ein Framework eine semi-vollständige Anwendung, die wiederverwendbare und geläufige Strukturen bereitstellt, die erweitert und auf einen spezifischen Anwendungsfall zugeschnitten werden können. Das Framework übernimmt dabei die Rolle des Koordinators und steuert den Datenfluss und die Abläufe auf Basis eines generischen Algorithmus. Es wird auf einen konkreten Anwendungsfall adaptiert, indem über wohldefinierte Schnittstellen Komponenten angebunden werden und so dessen Verhalten beeinflusst wird.

In diesem Abschnitt wird auf Basis der in Abschnitt 4.1 ausgewählten und der in Abschnitt 4.2 entwickelten Verfahren ein adaptierbares Framework entwickelt, mit dem ein automatisiertes, ontologiebasiertes Management arbiträrer Anwendungsfälle umgesetzt werden kann. Dazu werden in Abschnitt 4.3.1 zunächst die einzelnen Phasen und Verfahren vorgestellt, die bei der Integration eines Anwendungsfalls bis hin zum autonomen Management durchlaufen werden. Abschnitt 4.3.2 präsentiert dann die in Subsysteme untergliederte Architektur des Laufzeitsystems.

4.3.1 Phasen

Beim Einsatz des Frameworks werden verschiedene Phasen durchlaufen: In der *Adaptionsphase* wird das Framework auf einen Anwendungsfall zugeschnitten, indem ontologische Modelle und Adapter entwickelt werden. In der *Initialisierungsphase* passt das Framework sein Laufzeitsystem auf den in den Modellen beschriebenen Anwendungsfall an und bindet die bereitgestellten Adapter in die Infrastruktur ein. In der *Laufzeitphase* verarbeitet das Laufzeitsystem kontinuierlich die von den Adaptern bereitgestellten Monitoring-Daten des überwachten Systems, in dem es die darin beschriebenen Informationen mit dem Systemmodell verknüpft, auswertet und automatisiert Management-Entscheidungen trifft und umsetzt. In der parallel zur *Laufzeitphase* stattfindenden *Optimierungsphase* wird das überwachte System anhand definierter Ziele und Einschränkungen autonom optimiert. Die folgenden Abschnitte erläutern die einzelnen Phasen im Detail.

Adaptionsphase

In der Adaptionsphase wird das Framework auf einen konkreten Anwendungsfall adaptiert. Sie untergliedert sich in die Schritte Modellierung, Anbindung und Konfiguration. Der Modellierungschritt umfasst den Entwurf und die Umsetzung von

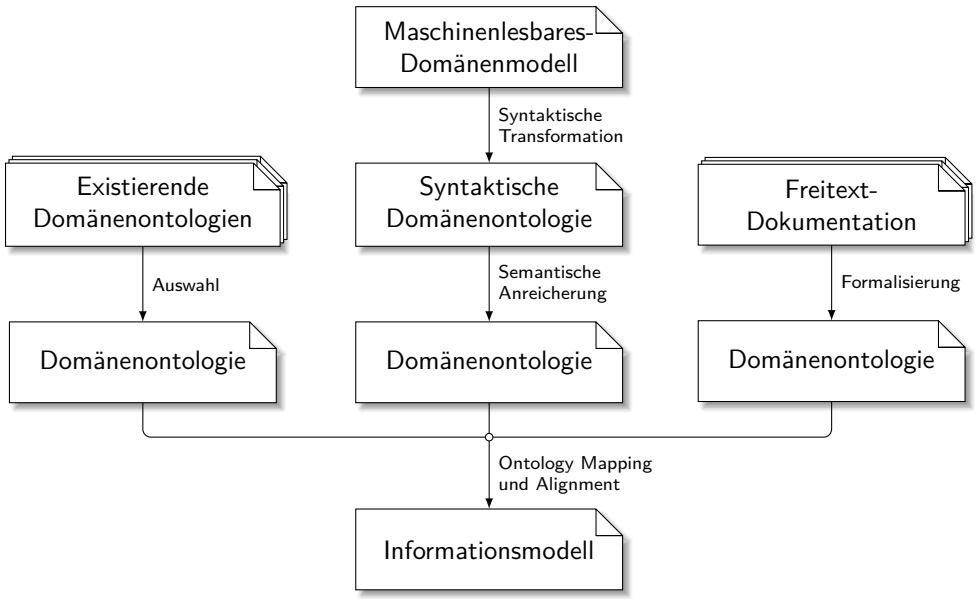


Abbildung 4.22.: Entwicklung des ontologischen Informationsmodells.

Domänen-, Schnittstellen- und Management-Modellen, der Anbindungsschritt den Entwurf und die Umsetzung von Technologieadaptern, der Konfigurationsschritt die konkrete Modell- und Adapterauswahl und -konfiguration.

Modellierung Bei der Adaption des Frameworks auf einen Anwendungsfall besteht der erste Schritt im Entwurf des ontologischen Informationsmodells (siehe Abschnitt 4.1.1, Seite 71). Dazu müssen Ontologien aller für das Management relevanten Teildomänen entwickelt bzw. ausgewählt werden (siehe Abb. 4.22). Im idealen Fall existiert bereits eine von Domänenexperten entwickelte Ontologie, die alle für den Anwendungsfall relevanten Aspekte abdeckt. Das W3C pflegt auf seiner Webseite[1] eine Liste von Ontologien und Ontologiequellen, die für diesen Zweck herangezogen werden können.

Existiert kein hinreichendes Modell, muss eine neue Ontologie entwickelt werden. Dies kann auf unterschiedliche Arten geschehen: Ist ein maschinenlesbares, nicht-ontologisches Domänenmodell (z. B. eine SNMP-MIB oder ein XSD-Schema) vorhanden, kann eine syntaktische Transformation durchgeführt und das resultierende Modell von Domänenexperten semantisch angereichert werden (vgl. [35]). Existiert ein solches Modell nicht, muss die Ontologie von einem Domänenexperten manuell entwickelt werden, indem nicht-formale Ressourcen wie Freitextdokumentation unter Einsatz gängiger Ontologieentwurfsmuster [51] formalisiert werden.

[1] `https://www.w3.org/wiki/Lists_of_ontologies`

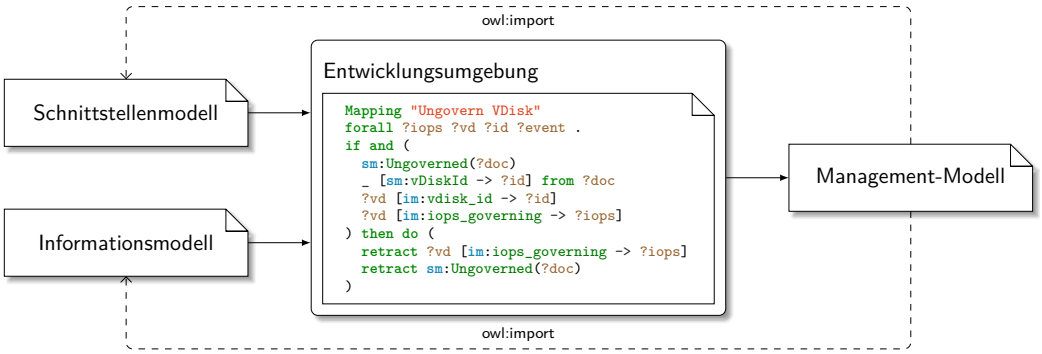


Abbildung 4.23.: Tool-gestützte Entwicklung des Management-Modells in domänenspezifischer Sprache (DSL).

Als Modellierungswerkzeug hat sich dafür Protégé [52] der Stanford University etabliert.

Die Modelle der Teildomänen werden anschließend miteinander verknüpft, indem domänenübergreifend vorkommende Konzepte und Rollen mit Verfahren des Ontology Mapping und Alignment [27] aufeinander abgebildet werden. Dadurch entsteht ein einheitliches Informationsmodell der Gesamtdomäne.

Um zur Laufzeit temporale Zusammenhänge in der Wissensbasis repräsentieren zu können, wird das in Abschnitt 4.1.1 (siehe Seite 71) ausgewählte 4D-Fluents Entwurfsmuster genutzt. Dazu wird das Ursprungsmodell von einer Erweiterungsontologie importiert und temporale Rollen und Klassen als Unterklassen bzw. -rollen von `fluents:Class`, `fluents:ObjectProperty` oder `fluents:DataProperty` ausgewiesen. Ob die Auszeichnung vor der Verknüpfung auf den Teilmodellen oder nach der Verknüpfung auf dem Gesamtmodell durchgeführt wird, spielt keine Rolle.

Nachdem das ontologische Informationsmodell der Domäne entwickelt wurde, besteht der nächste Schritt darin, die Schnittstellen des überwachten Systems zu beschreiben. Dazu wird ein Schnittstellenmodell entwickelt, das die Struktur der von RDF-Dokumenten repräsentierten Monitoring-Events und Management-Aktionen beschreibt, die zur Laufzeit zwischen dem überwachten System und dem Laufzeitsystem des Frameworks ausgetauscht werden. Die Events und Aktionen werden auf Basis der in Abschnitt 4.2.1 (siehe Seite 83) vorgestellten Domain Interface Ontology (DIO) modelliert. Analog zum Informationsmodell können mehrere Schnittstellenmodelle für Teildomänen entwickelt werden. Da die Event- und Aktionsdokumente innerhalb des Frameworks isoliert betrachtet werden, findet keine Modellverknüpfung statt.

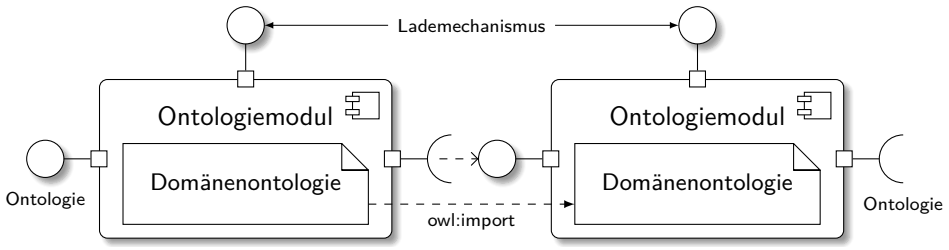


Abbildung 4.24.: Ontologiemodule kapseln Domänenontologien, definieren Abhängigkeiten und stellen Lademechanismen bereit.

Auf Basis des Informations- und Schnittstellenmodells kann mit der in Abschnitt 4.2.1 (siehe Seite 86) entwickelten Taxonomie der Ontology-based Management Language (OntML) die anwendungsfallspezifische Management-Logik beschrieben werden. Die Management-Module können über mehrere Ontologien verteilt sein. Innerhalb eines Moduls werden abstrakte Ziele und Management-Regeln definiert, die semantisch auf Klassen, Rollen und Individuen importierter Informations- und Schnittstellenontologien Bezug nehmen.

Betrachtet man die komplexe OntML-Taxonomie, zeigt sich, dass selbst bei kleinen Anwendungsfällen die manuelle Modulentwicklung sehr aufwendig ist. Deshalb bietet das Framework eine domänenspezifische Sprache (DSL) mit zugehöriger Entwicklungsumgebung (siehe Abb. 4.23), in der Ontologiemodelle importiert und auf Basis der darin definierten Entitäten aus abstrakten Zielen und Regeln bestehende Management-Module kodiert werden können. Die in der DSL eingesetzte Regelsprache ist stark an die RIF-Präsentationssyntax (siehe Abschnitt 2.4.2, Seite 36) angelehnt. Damit beschriebene Management-Module werden entsprechend der OntML-Taxonomie in RDF serialisiert, sodass sie vom Framework adaptiert werden können.

Ontologien sind innerhalb des Frameworks in sogenannte Ontologiemodule (siehe Abb. 4.24) gekapselt, die eine Abstraktion auf die physikalische Repräsentation des Ontologiedokuments darstellen. Dadurch sollen zur Laufzeit die folgenden Probleme vermieden werden:

- Der IRI einer im Framework einzusetzenden Ontologie repräsentiert nur eine abstrakte Ressource und verweist auf kein ladbares Dokument, das zur Wissensbasis hinzugefügt werden kann.
- Hinter der IRI steht zwar ein ladbares Dokument, aus Sicherheits-, Konsistenz- oder Performance-Gründen soll jedoch ein lokales Replikat geladen werden.

- Die in einer Ontologie definierten Abhängigkeiten an andere Ontologien sollen im Framework-Kontext vernachlässigt oder zusätzliche Abhängigkeiten injiziert werden.

Ein Ontologiemodul ist eine Software-Komponente mit Fähigkeiten (Capabilities) und Anforderungen (Requirements). In den Fähigkeiten legt das Modul fest, welche über den Versions-IRI identifizierten Ontologien es bereitstellt, über Anforderung, welche Ontologien es benötigt. Sind alle Anforderungen eines Moduls erfüllt, stellt es einen Lademechanismus bereit, über den das gekapselte Ontologiedokument als RDF-Tripelmenge geladen werden kann. Dabei ist es dem Modul überlassen, ob die Daten aus einer lokalen Datei stammen, entfernt abgefragt, oder bedarfsorientiert generiert werden. So können neben einfachen Ontologiemodulen, die physikalische Dokumente kapseln, auch komplexe Module umgesetzt werden, die in erster Instanz nur Metadaten des von ihnen abstrahierten Modells bereitstellen (Fähigkeiten und Anforderungen) und die realen Tripel zum Ladezeitpunkt dynamisch generieren (z. B. aus einem anderen Modellformat).

Das Framework stellt ein Werkzeug bereit, mit dem existierende Ontologiedokumente automatisch in Ontologiemodule konvertiert werden können. Dabei wird anhand der im Dokument enthaltenen Ontologiemetadaten eine Software-Komponente generiert, die entsprechende Fähigkeiten (aus `owl:versionIRI` extrahiert) und Anforderungen (aus `owl:import` extrahiert) definiert und einen Lademechanismus bereitstellt, über den das gekapselte Dokument geladen werden kann. Für die 4D-Fluents-, DIO- und OntML-Ontologien stellt das Framework fertige Ontologiemodule bereit.

Anbindung Um das Laufzeitsystem des Frameworks an ein überwachtes System anzubinden, müssen Technologieadapter bereitgestellt werden. Adapter sind Software-Komponenten, die Modelle und Protokolle beider Systeme kennen und zwischen ihnen vermitteln können. Prinzipiell sind Adapter anwendungsfallunabhängig und können in unterschiedlichen Szenarien die Brücke zu standardisierten Technologien (z. B. SNMP oder CIM) bilden. Häufig erfordern es proprietäre Technologien und Komponenten jedoch, szenariospezifische Adapter zu entwickeln. Das Framework sieht drei Adaptertypen vor, über die es mit einem überwachten System verknüpft werden kann:

(1) Wie in Abschnitt 4.1 beschrieben, kann die Wissensbasis zur Laufzeit ausschließlich durch Event-Interpretationen beeinflusst werden. In der Initialisierungsphase soll es jedoch möglich sein, eine Momentaufnahme des überwachten Systems zu importieren und so einen Datengrundstock zu schaffen. Dabei kommen sogenannte

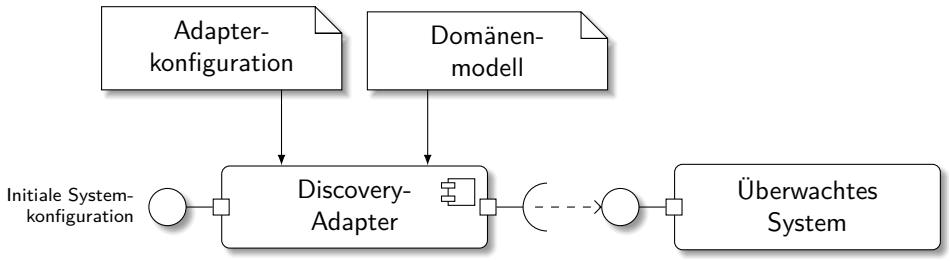


Abbildung 4.25.: Discovery-Adapter stellen initiale, dem Domänenmodell entsprechende Systemkonfigurationen bereit.

Discovery-Adapter (siehe Abb. 4.25) zum Einsatz. Sie bilden in einem Discovery-Vorgang anhand von Konfigurationsparametern die konkrete Systemkonfiguration auf eine dem Domänenmodell entsprechende Tripelmenge ab. Die Bezugsquelle kann dabei mannigfaltig sein; ein Discovery-Adapter kann sich beispielsweise zu einem konfigurierbaren SNMP-Daemon verbinden, die relevanten Informationen auslesen und auf das Domänenmodell abbilden, Stammdaten aus einer Datenbank lesen und auf das Domänenmodell abbilden oder eine bereits als Ontologiedokument vorliegende Systemrepräsentation laden.

(2) Um Monitoring-Events vom überwachten System empfangen zu können, müssen dem Laufzeitsystem Überwachungsadapter (siehe Abb. 4.26) bereitgestellt werden. Sie beziehen Monitoring-Daten vom überwachten System, konvertieren sie in dem Schnittstellenmodell entsprechende RDF-Dokumente und speisen diese als Events in das Laufzeitsystem ein. Typischerweise werden Events von den Adaptern gewonnen, indem aktiv Management-Schnittstellen abgefragt, Topics von der dem System unterlagerten nachrichtenorientierten Middleware abonniert oder Log-Datenströme gelesen werden. Das Gegenstück zu Überwachungsadaptern bilden Umsetzungsadapter. Sie nehmen die als RDF-Dokumente kodierten Management-Aktionen des Laufzeitsystems entgegen und führen das davon beschriebene Rekonfigurationskommando auf dem überwachten System aus (z. B. durch Dienstaufrufe).

(3) Neben den Adapterschnittstellen bietet das Laufzeitsystem eine Management-Schnittstelle, über die externe Komponenten mit ihm interagieren können. Über sie können der in der Wissensbasis gehaltene Systemzustand abgefragt und abstrakte Ziele abonniert werden und so beispielsweise Visualisierungswerkzeuge angebunden werden. Darüber hinaus kann über die Schnittstelle der Management-Prozess aktiv beeinflusst werden, indem die autonome Optimierung angestoßen und über die Umsetzung der resultierenden Rekonfigurationsagenda entschieden wird oder manuell Management-Aktionen ausgelöst werden.

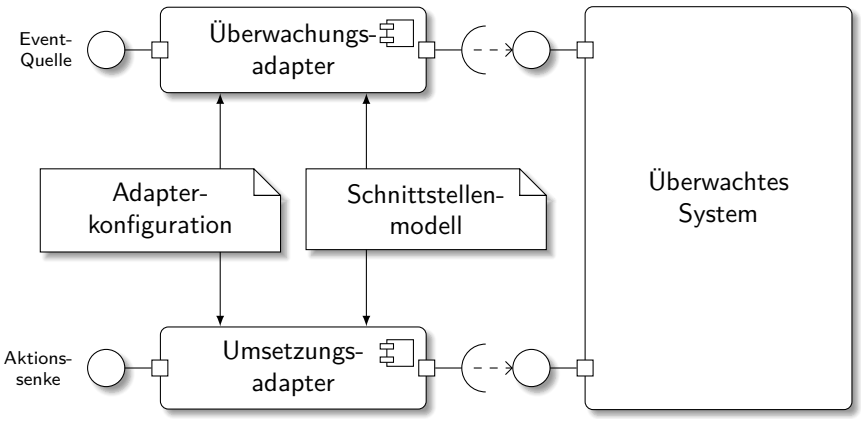


Abbildung 4.26.: Überwachungsadapter stellen Überwachungsdaten als Events bereit, Umsetzungsadapter führen Aktionen als Kommandos auf dem überwachten System aus.

Konfiguration Der abschließende Adaptionsschritt besteht in der Definition der anwendungsfallspezifischen Framework-Konfiguration. Darin wird festgelegt, welche Ontologien in die Wissensbasis geladen und welche Adapter angebunden werden müssen, um das konkrete Szenario umzusetzen. Für eine Adapterimplementierung können dabei durchaus mehrere Konfigurationen definiert werden.

Initialisierungsphase

In der Initialisierungsphase passt das Framework sein Laufzeitsystem an die für den Anwendungsfall in der Konfiguration beschriebenen Modelle und Adapter an. Dabei werden die Schritte Wissensbasisinitialisierung, Regelgenerierung, Komponenteninitialisierung und Discovery durchlaufen, die im Folgenden im Detail erläutert werden.

Wissenbasisinitialisierung Der erste Initialisierungsschritt besteht darin, die für den Anwendungsfall benötigen Ontologien in die Wissensbasis des Laufzeitsystems einzufügen. Welche Modelle geladen werden sollen, wird explizit in der Framework-Konfiguration festgelegt. Der Modul-Manager ist dafür zuständig, aus den bereitgestellten Ontologiemodulen eine Modulkombination zu bilden, die allen konfigurierten Ontologien und deren Abhängigkeiten genügt. Dazu bildet er einen Abhängigkeitsgraph, auf dem er eine Tiefensuche durchführt. Wird eine valide Modulkombination gefunden, stößt der Modul-Manager die Lademechanismen aller benötigten Ontologiemodule an und fügt die resultierenden RDF-Statements

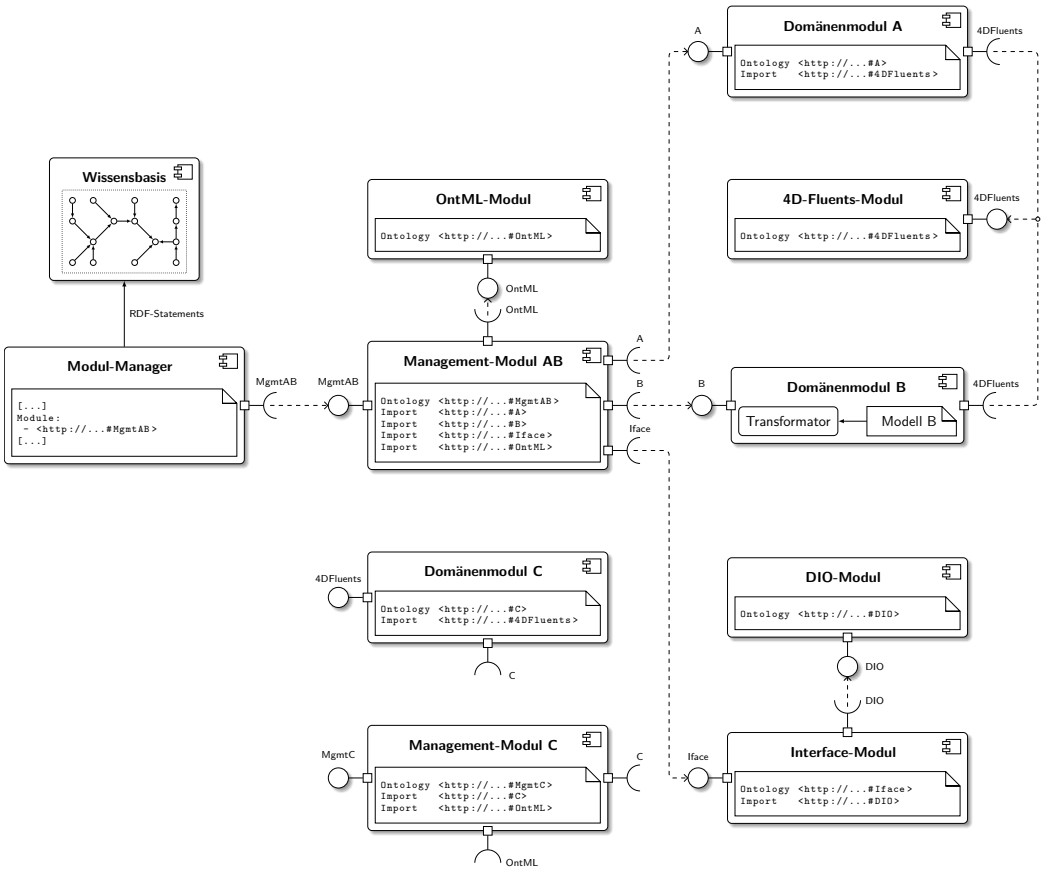


Abbildung 4.27.: Beispielhafte Modulresolution des Modul-Managers.

zur Wissensbasis hinzu. Wird keine valide Modulkombination gefunden, wird der Ladevorgang abgebrochen.

Abbildung 4.27 zeigt beispielhaft die Funktionsweise des Modul-Managers. In der Konfiguration wird die *Mgmt-AB*-Ontologie als benötigtes Modell festgelegt und muss vom Modul-Manager aufgelöst und geladen werden. Dafür wird eine Modulkombination gesucht, die alle transitiven Anforderungen erfüllt. Die Tiefensuche ergibt die Kombination *Management-Modul AB*, *Domänenmodul A*, *Domänenmodul B*, *4D-Fluents-Modul*, *Interface-Modul* und *DIO-Modul*. Über die von den Modulen bereitgestellten Mechanismen lädt der Modul-Manager die gekapselten Ontologien und fügt sie in die Wissensbasis ein. Das *Domänenmodul B* transformiert während des Ladevorgangs ein nicht-ontologisches Modell nach RDF und stellt die resultierende Tripelmenge bereit.

Der Ladevorgang findet einmalig statt; Ontologiemodule können zur Laufzeit weder hinzugefügt noch entfernt werden. Nachdem die Wissensbasis mit den benötig-

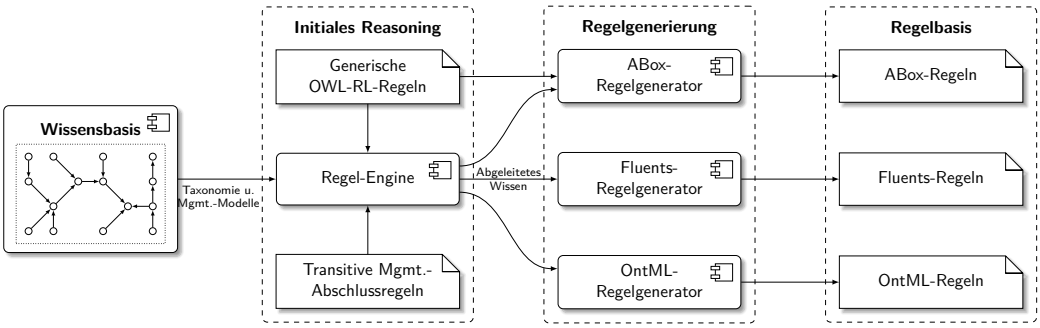


Abbildung 4.28.: Prozess zur Gewinnung anwendungsfallspezifischer Regeln aus der geladenen Informationsmodelltaxonomie und den Management-Modellen.

ten Ontologien und deren Abhängigkeiten initialisiert wurde, werden der Modul-Manager und die Ontologiemodule nicht mehr benötigt.

Regelgenerierung Nach der Wissensbasisinitialisierung ist das Informations- und Management-Modell während der nachfolgenden Phasen statisch. Um die durch die Taxonomie definierte OWL-Semantik und die im Management-Modell definierte Management-Logik zur Laufzeit umsetzen zu können, werden im Regelgenerierungsschritt (siehe Abb. 4.28) anwendungsfallspezifische Regeln erzeugt.

Im ersten Generierungsteilschritt werden die in die Wissensbasis geladenen Daten einem Reasoning unterzogen. Dabei werden die generischen OWL-RL-Regeln auf einen Abzug der Wissensbasis angewandt, um mit den TBox-Regeln Schlussfolgerungen über die Informationsmodelltaxonomie und mit den ABox-Regeln Schlussfolgerungen über das Management-Modell zu ziehen (siehe Abschnitt 4.2.1). Das ABox-Reasoning führt dazu, dass alle vom Management-Modell referenzierten Domänenentitäten als solche klassifiziert werden. Um dabei neben den explizit Management-relevanten Entitäten auch die implizit Management-relevanten Entitäten aufzudecken, kommt beim Reasoning ein zusätzlicher Regelsatz zum Einsatz, der die in Abschnitt 4.2.2 beschriebenen Ableitungsvorschriften umsetzt.

Nach Abschluss des Reasonings findet die eigentliche Regelgenerierung statt, bei der für den Anwendungsfall Engine-spezifische Regeln generiert werden. Voraussetzung dafür ist, dass die Ziel-Engine RIF-BLD [19], RIF-PRD [19] und Aggregationsterme aus RIF-FLD [20] unterstützt und über einen kontinuierlichen Event-Verarbeitungsmechanismus verfügt, der gleitende Zeitfenster und Allen-Relationen unterstützt (siehe Abschnitt 4.2.1 und Abschnitt 4.2.3). Der Generierungsprozess unterteilt sich in OWL-RL-, Fluent- und OntML-Regelgenerierung.

Bei der OWL-RL-Regelgenerierung wird der in Abschnitt 4.2.3 entwickelte Ansatz auf alle beim Reasoning als Management-relevant klassifizierten Entitäten angewandt und so aus den generischen OWL-RL-ABox-Regeln unter Berücksichtigung der geschlussfolgerten Informationsmodelltaxonomie spezialisierte ABox-Regeln generiert. Der Prozess resultiert in einer Menge von Implikationsregeln, die im In-Memory-Abbild zum Reasoning der Management-relevanten Datenuntermenge eingesetzt werden können.

Zur effizienten Verarbeitung der Fluent-Strukturen zur Laufzeit werden Regeln generiert, die die davon repräsentierten temporalen Beziehungen auf die Event-Ebene heben, sodass der native Event-Verarbeitungsmechanismus der Regel-Engine zur Verarbeitung eingesetzt werden kann. Aus jeder Fluent-Struktur wird ein Event abgleitet, das ein Quadrupel aus Subjekt, Prädikat, Objekt und Gültigkeitsintervall ist. Subjekt, Prädikat und Objekt repräsentieren die universell gültigen Ressourcen der Beziehung, das Gültigkeitsintervall das Intervall der beteiligten temporalen Ausprägung des Subjekts.

Sei tp die abstrakte Rolle `fluents:hasTemporalPart` und ext die abstrakte Rolle `fluents:temporalExtent`, dann wird für jede Management-relevante und temporale abstrakte Rolle p eine logische Implikation der Form

$$\frac{\text{s tp } s_p\ .\qquad s_p\ \text{p}\ o_p\ .\qquad \text{o tp } o_p\ .\qquad s_p\ \text{ext i}\ .}{\text{s p o i}\ .}\ \textbf{abstract fluent role}$$

für jede Management-relevante und temporale konkrete Rolle p eine logische Implikation der Form

$$\frac{\text{s tp } s_p\ .\qquad s_p\ \text{p o}\ .\qquad s_p\ \text{ext i}\ .}{\text{s p o i}\ .}\ \textbf{concrete fluent role}$$

und für jede Management-relevante und temporale Klasse c eine logische Implikation der Form

$$\frac{\text{s tp } s_p\ .\qquad s_p\ \text{rdf:type c}\ .\qquad s_p\ \text{ext i}\ .}{\text{s rdf:type c i}\ .}\ \textbf{fluent class}$$

generiert.

Parallel zur OWL- und Fluent-Semantik wird die als OntML-Module in der Wissensbasis vorliegende Management-Logik in Engine-spezifische-Regeln übersetzt. Dieser Prozess ist größtenteils trivial; viele OntML-Konstrukte können auf syntaktischer

Ebene direkt auf ein RIF-artiges Zielformat abgebildet werden. Dabei werden funktionale Abhängigkeiten als Produktionsregeln umgesetzt, sodass die implizierten Tripel über Wissensbasisoperationen materialisiert werden. Hält die funktionale Abhängigkeit nicht mehr, wird das Tripel über eine weitere Operation aktiv aus der Wissensbasis entfernt.

Temporale Beziehungen werden bei der Regelgenerierung gesondert behandelt. Ist die von einer Rollen- bzw. Klassencharakteristik einer Regel referenzierte Entität im Informationsmodell als temporal annotiert, wird in der generierten Regel nicht das RDF-Tripel sondern das aus den Fluents abgeleitete Quadrupel gebunden. Sei t_{now} das „Jetzt“ in der kontinuierlichen Verarbeitung, dann wird ein Gültigkeitsintervall i mit Startzeitpunkt t_0 und ohne Endzeitpunkt interpretiert als $i = [t_0; t_{now})$. Ein Zeitfenster der Länge l ist definiert als $w_l = [t_{now} - l; t_{now})$. Ein Zeitfenster w *umfasst* ein Intervall i gdw. w *überschneidet sich mit* i oder i *überschneidet sich mit* w (vgl. Abbildung 2.4). Hat eine Charakteristik keine temporale Einschränkung, bindet die Regel alle Quadrupel, die das Zeitfenster der Länge Null *umfasst* (alle im „Jetzt“ gültigen Quadrupel). Hat die Charakteristik eine Zeitfenstereinschränkung w_l, dann bindet die Regel alle Quadrupel mit Intervall i, für die w_l *umfasst* i gilt. Ist die Charakteristik auf einen relativen Zeitpunkt t eingeschränkt, bindet die Regel alle Quadrupel mit Intervall i, für die i *überschneidet sich mit* w_l gilt. Hat die Charakteristik eine Einschränkung über eine Allen-Relation a auf den Operand o, dann bindet die Regel alle Quadrupel q, für die a zwischen q und o hält. Monitoring-Events unterliegen derselben Semantik; ein Event zum Zeitpunkt t hat das Gültigkeitsintervall $[t; t)$.

Bei den Regelkonsequenzen müssen temporale Beziehungen ebenfalls gesondert behandelt werden. Bei Produktionsregeln, die einem Kontextindividuum als Konsequenz eine temporale Rolle oder Klasse zuweisen, wird eine entsprechende Fluent-Struktur mit einem rechts-offenen Gültigkeitsintervall zur Wissensbasis hinzugefügt. Bei Produktionsregeln, die als Konsequenz eine temporale Rolle oder Klasse eines Kontextindividuums löschen, wird der Fluent nicht gelöscht, sondern dessen Gültigkeitsintervall der Endzeitpunkt t_{now} hinzugefügt. Funktionale Abhängigkeiten unterliegen auf Grund ihrer Umsetzung als Produktionsregeln (s. o.) ebenfalls dieser Semantik, der Fluent wird jedoch automatisch geschlossen, sobald die Abhängigkeit nicht mehr hält.

Komponenteninitialisierung Nachdem die Ontologiemodule in die Wissensbasis geladen und die Regelbasis generiert wurde, werden im nächsten Schritt die für die Laufzeit- und Optimierungsphase benötigten Komponenten initialisiert. Kern des Management-Prozesses bildet eine Regel-Engine, deren Regelbasis aus denen

im Generierungsprozess erzeugten ABox-, Fluent- und OntML-Regeln besteht. Die lokale Wissensbasis der Engine hält gleichzeitig das In-Memory-Abbild des in Abschnitt 4.2.2 entwickelten hybriden Datenhaltungsansatzes. Für den autonomen Optimierungsprozess wird eine zweite Regel-Engine-Instanz mit derselben Regelbasis erzeugt, die parallel zur ersten Engine genutzt werden kann.

Nachdem beide Regel-Engines mit den generierten Regelsätzen erzeugt wurden, werden die anwendungsfallspezifischen Adapter (siehe Abschnitt 4.3.1) geladen. Über die Framework-Konfiguration können für jede Adapterimplementierung mehrere Instanzen konfiguriert werden; die Konfigurationsparameter eines Adapters sind implementierungsspezifisch: ein SNMP-Überwachungsadapter wird beispielsweise mit einer IP-Adresse, einer Port-Nummer und Benutzerdaten, ein Konfigurations-Discovery-Adapter, der Daten aus einer Excel-Tabelle liest, hingegen mit einem Dateipfad konfiguriert. Für jeden Adaptertyp (Discovery, Überwachung und Umsetzung) existiert eine separate Verwaltungskomponente, der sogenannte Adapter-Manager. Jeder Adapter-Manager liest die Instanzkonfigurationen seines Adaptertyps aus der Framework-Konfiguration, erzeugt die entsprechenden Komponenteninstanzen und konfiguriert sie.

Discovery Vor dem Übergang in die Laufzeitphase wird die Wissensbasis initial mit einer Momentaufnahme des überwachten Systems befüllt. Dazu wird ein Discovery-Prozess angestoßen, bei dem die konfigurierten und vom Discovery-Manager erzeugten Adapter ihre spezifischen Discovery-Mechanismen einsetzen, um aus unterschiedlichen Datenquellen Informationen über das überwachte System zu gewinnen. Jeder Adapter bildet die gewonnenen Informationen auf seinem Domänenmodell entsprechende RDF-Ressourcen und Beziehungen ab und stellt dem Discovery-Manager die resultierenden Tripel bereit. Der Manager sammelt die Daten aller Discovery-Adapter ein und fügt sie zur Wissensbasis hinzu. In den nachfolgenden Phasen findet kein weiteres Discovery statt, sodass der Manager und die Adapterinstanzen nicht mehr benötigt werden.

Abbildung 4.29 zeigt beispielhaft, wie der Discovery-Manager anhand der Framework-Konfiguration Discovery-Adapter-Instanzen erzeugt, die über ihre technologiespezifischen Protokolle Informationen von den konfigurierten Datenquellen sammeln, in RDF umwandeln und dem Manager bereitstellen, der das Resultat in die Wissensbasis einfügt.

Nach Abschluss des Discovery-Prozesses, wird das in der Regel-Engine angesiedelte In-Memory-Abbild mit der Wissensbasis synchronisiert. Dazu wird ein SPARQL-Query erzeugt, das auf Basis der im Reasoning geschlussfolgerten Entitäten alle

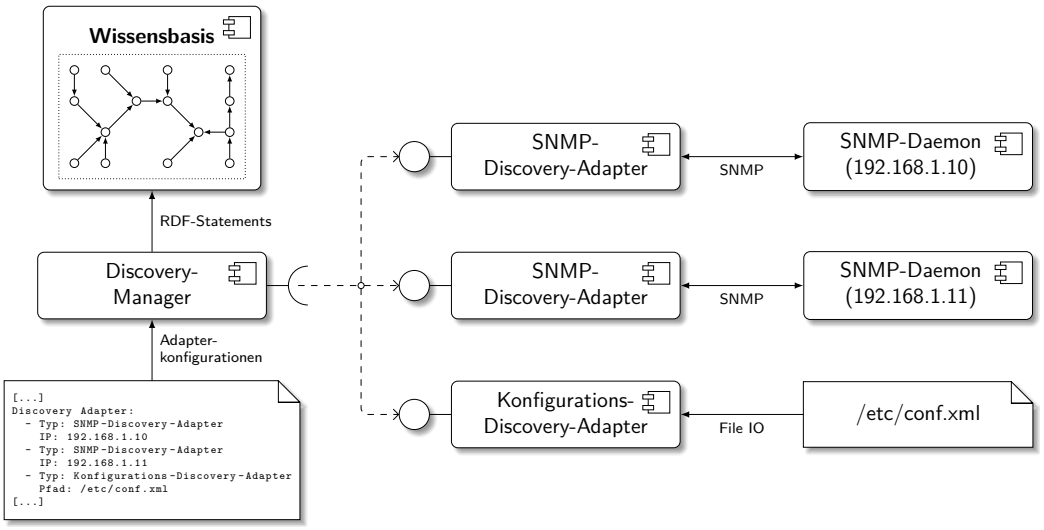


Abbildung 4.29.: Discovery-Prozess zur initialen Befüllung der Wissensbasis mit einer Momentaufnahme des überwachten Systems.

Management-relevanten Tripel aus der Wissensbasis extrahiert und in die lokale Wissensbasis der Regel-Engine einfügt.

Laufzeitphase

In der Laufzeitphase setzt das Laufzeitsystem des Frameworks eine kontinuierliche Verarbeitung um, bei der stetig Events vom überwachten System empfangen, anhand der im Management-Modell definierten Verarbeitungslogik ausgewertet und die abgeleiteten Aktionen auf dem überwachten System umgesetzt werden.

Zu Beginn der Laufzeitphase bekommt jeder in der Initialisierungsphase instanziierte Überwachungsadapter vom Überwachungs-Manager einen Event-Kanal zugeteilt und wird anschließend in einem separaten Thread gestartet. Die Adapter interagieren dann über die gesamte Laufzeitphase über ihre spezifische Schnittstelle mit den überwachten Komponenten, transformieren die gewonnenen Informationen in RDF-Dokumente und fügen sie in ihren Kanal ein. Alle Kanalenden laufen beim Überwachungs-Manager zusammen, dessen Aufgabe darin besteht, die eingehenden Events anhand des Schnittstellenmodells zu validieren und anschließend in die lokale Wissensbasis der Regel-Engine einzufügen.

Die Regel-Engine wird zu Beginn der Laufzeitphase in einen kontinuierlichen Verarbeitungsmodus versetzt, in dem sie stetig ihre lokale Wissensbasis und die Zeit überwacht. Ändert sich die Wissensbasis (wenn z. B. ein neues Event hinzugefügt wurde) oder schreitet die Zeit um ein gewisses Maß fort (sodass z. B. ein Event

ein Zeitfenster verlässt), revalidiert sie ihre Regeln und leitet neue Aktivierungen ab, deren Konsequenzen unverzüglich umgesetzt werden. Daraus resultierende Event-Operationen werden ausschließlich auf der lokalen Wissensbasis umgesetzt, indem Events hinzugefügt oder entfernt werden. Wissensbasisoperationen werden vollständig auf dem Triplestore umgesetzt, auf dem In-Memory-Abbild nur die Management-relevante Daten betreffenden Operationen. Wird ein über die Management-Schnittstelle des Frameworks abonniertes abstraktes Ziel erreicht, wird es an den entsprechenden Management-Adapter weitergeleitet. Von Policies abgeleitete Aktionsausführungen werden zur weiteren Verarbeitung an den Umsetzungs-Manager übergeben.

Der Umsetzungs-Manager und dessen Adapter sind zur Laufzeit passiv. Erst wenn von der Regel-Engine eine auszuführende Management-Aktion abgeleitet wurde, wird der Manager aktiv und sucht unter den in der Initialisierungsphase instanziierten Umsetzungsadaptern nach einem geeigneten Umsetzungskandidaten. Dabei fragt er alle für den Aktionstyp registrierten Adapter, ob sie die parametrisierte Aktion umsetzen können. Jeder Adapter entscheidet dann anhand der Aktionsparameter, ob er dazu in der Lage ist (z. B. ob er der SNMP-Adapter für den in der Aktion definierten Ziel-Host ist). Aus allen umsetzungsfähigen Adaptern wählt der Umsetzungs-Manager einen Adapter aus und beauftragt ihn mit der Aktionsumsetzung. Der Adapter transformiert dann die in RDF beschriebene Aktion in ein spezifisches Steuerungskommando und führt es über die technologiespezifische Schnittstelle auf dem überwachten System aus.

Abbildung 4.30 zeigt das Zusammenspiel und den Datenfluss der unterschiedlichen Komponenten. Die Überwachungsadapter beziehen über die technologiespezifische Überwachungsschnittstelle Informationen, transformieren sie in RDF- Dokumente und geben sie über ihren Event-Kanal an den Überwachungs-Manager weiter. Dieser validiert die Events anhand des Schnittstellenmodells und fügt sie in die lokale Wissensbasis der Regel-Engine ein. Die Regel-Engine revalidiert ihre Regeln und leitet daraus Operationen ab, die unmittelbar auf ihre lokale Wissensbasis umgesetzt werden. Wissensbasisoperationen werden zudem auf die globale Wissensbasis angewandt, Management-Aktionen werden an den Umsetzungs-Manager weitergereicht, der einen geeigneten Umsetzungsadapter sucht, der die Aktion über eine technologiespezifische Kommandierungsschnittstelle ausführt. Abstrakte Ziele und der Systemzustand können über die Management-Schnittstelle abonniert bzw. abgefragt werden.

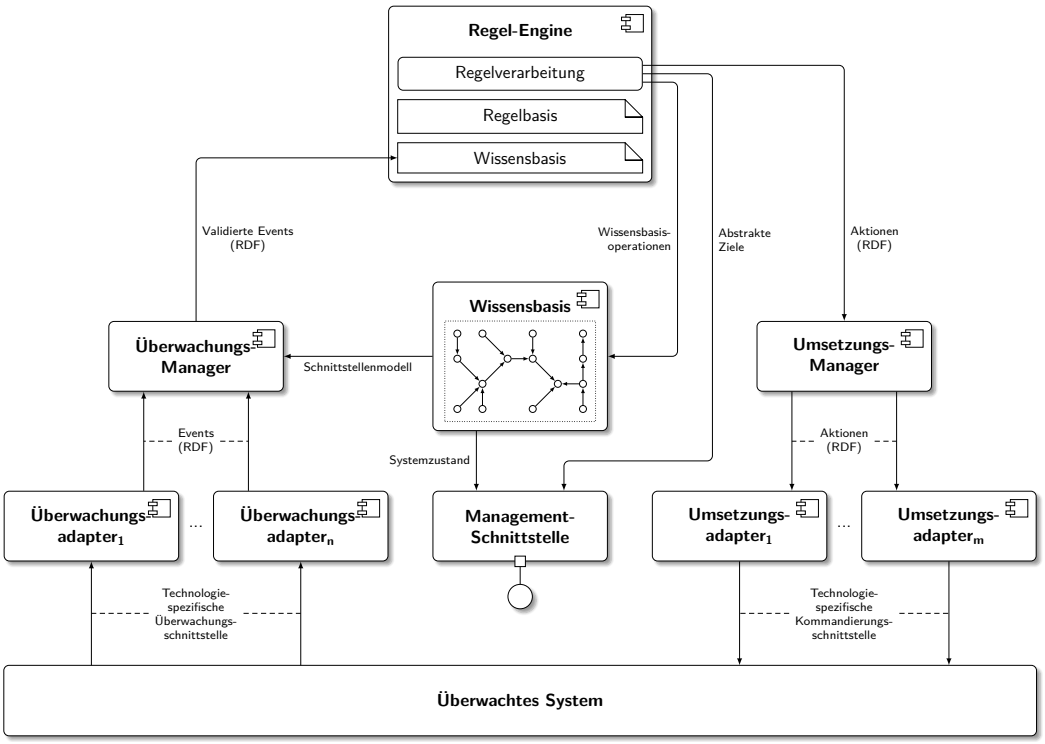


Abbildung 4.30.: Datenfluss und Zusammenspiel der Komponenten in der Laufzeitphase.

Optimierungsphase

In der Optimierungsphase wird parallel zu dem in der Laufzeitphase umgesetzten automatisierten Management der in Abschnitt 4.2.4 vorgestellte, autonome Optimierungsansatz umgesetzt. Der Optimierungsprozess wird über die Management-Schnittstelle des Laufzeitsystems angestoßen. Ob der Prozess manuell von einem Systemadministrator oder automatisiert von einer Überwachtungskomponente (die bspw. auf bestimmte Events, abstrakte Ziele oder Systemzustände reagiert) angestoßen wird, hängt von der spezifischen Szenarioadaption (siehe Abschnitt 4.3.1) ab.

Die Optimierung findet auf einer Kopie der Wissensbasis und der Verarbeitungslogik statt, sodass der automatisierte Management-Prozess nicht beeinflusst wird. Die in der Initialisierungsphase erzeugte Regel-Engine-Kopie (siehe Abschnitt 4.3.1) ist zur Laufzeit prinzipiell passiv. In ihrer Regelbasis befinden sich die anwendungsfallspezifischen OWL-RL-, Fluents- und OntML-Regeln, ihre lokale Wissensbasis ist leer. Zu Beginn des Optimierungsprozesses wird die lokale Wissensbasis der Optimierungs-Engine mit der lokalen Wissensbasis der Laufzeit-Engine synchronisiert, indem die darin befindlichen Fakten (RDF-Tripel und Events) kopiert werden. Anders als beim automatisierten Management findet bei der Optimierung keine kontinuierliche Re-

gelauswertung statt, sie wird bedarfsorientiert von der Optimierungskomponente durchgeführt.

Die autonome Optimierung findet Schrittweise (inkrementell) statt. In jedem Schritt werden drei Phasen durchlaufen:

- In der *Erhebungsphase* werden die in Abhängigkeit des aktuellen Systemzustands möglichen Management-Aktionen aufgedeckt. Dazu werden die im Management-Modell hinterlegten Aktionsquellen (logische Implikationen) abgefragt und mögliche Aktionen in einer Liste zusammengetragen.
- In der *Bewertungsphase* werden die in der Erhebungsphase aufgedeckten Aktionen bewertet. Dazu wird jede Aktion auf eine Bewertung abgebildet, indem (1) eine Transaktion auf der lokalen Wissensbasis der Optimierungs-Engine begonnen wird, (2) die Aktion in die Wissensbasis eingefügt wird, (3) eine Regelauswertung durchgeführt wird, bei der anhand der im Modell hinterlegten Aktionsimplementierung und der weiteren Management-Logik der Aktionseffekt auf das Systemmodell angewandt wird, (4) über die Fitnessfunktionen eine Bewertung durchgeführt wird und (5) die Transaktion abgebrochen (Rollback) wird. Über die Bewertung hinaus, wird für jede Aktion die Menge der hinzugefügten und entfernten Tripel gebildet (ein Log geführt), sodass aus dem Ausgangsmodell das resultierende Systemmodell rekonstruiert werden kann.
- In der *Steuerungsphase* wählt die eingesetzte Metaheuristik auf Basis der bereits umgesetzten Aktionen und der Bewertungsergebnisse (Aktionen, Bewertungen und Logs) die nächste auszuführende Aktion aus oder entscheidet über die Optimierungsprozessterminierung. Wird eine Aktion ausgewählt, wird ihr Bewertungsergebnis zur Rekonfigurationsagenda hinzugefügt, die Aktion in die Wissensbasis eingefügt, eine Regelauswertung durchgeführt, die Transaktion abgeschlossen (Commit) und auf dem resultierenden Zustand der nächste Optimierungsschritt durchgeführt.

Die im Laufzeitsystem als Standard-Metaheuristik eingesetzte Tabu-Suche kann über die Framework-Konfiguration mit einer maximalen Schrittzahl, maximalen Optimierungsdauer und Schlüsselmerkmalen konfiguriert werden. Die Schlüsselmerkmale sind Domänenklassen und -rollen, die bei der Tabu-Einstufung eines Systemzustands herangezogen werden: entsprechen die Klassen- und Rolleninstanzen aller Schlüsselmerkmale eines Systemzustands denen eines in der Agenda bereits durchlaufenen Zustands, ist die Aktion Tabu und darf nicht durchgeführt werden.

Vor der inkrementellen Optimierung wird eine initiale Ausgangszustandsbewertung durchgeführt, die als Vergleichsgrundlage des ersten Iterationsschritts dient. Nach

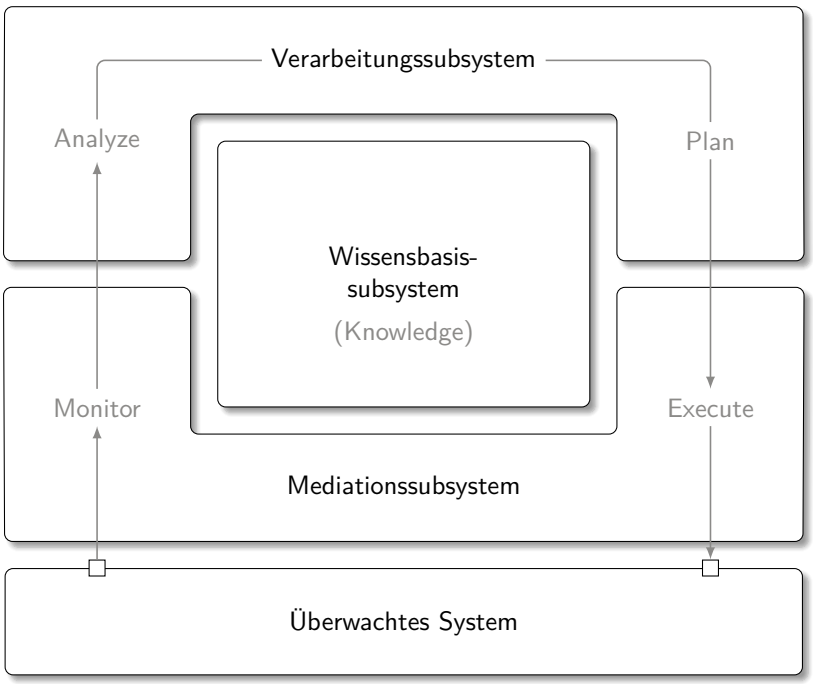


Abbildung 4.31.: Grobarchitektur und Einordnung der Subsysteme des Laufzeitsystems ins MAPE-K-Modell.

Abschluss des Optimierungsprozesses (Terminierung durch die Metaheuristik), wird die erarbeitete Rekonfigurationsagenda über die Management-Schnittstelle an den Aufrufer zurückgegeben. Dieser entscheidet dann, ob die Agenda angenommen oder abgelehnt werden soll. Wird die Agenda angenommen, wird die darin enthaltene Aktionskette über den Umsetzungs-Manager auf das überwachte System angewandt.

4.3.2 Architektur

Die im vorherigen Abschnitt beschriebenen Phasen sind von unterschiedlichen Software-Komponenten realisiert, die in diesem Abschnitt beschrieben und zur Komponentenarchitektur des Laufzeitsystems zusammengeführt werden. Die Architektur besteht aus mehreren Subsystemen (siehe Abb. 4.31), die jeweils einen konkreten Aufgabenbereich des MAPE-K-basierten Gesamtsystems umsetzen. Das Wissensbasissubsystem verwaltet das semantische Informationsmodell, das Management-Modell und die Instanzdaten und macht sie für die anderen Subsysteme zugreifbar. Das Mediationssubsystem bildet die Schnittstelle zum überwachten System, über die ein konkreter Anwendungsfall adaptiert wird. Das Verarbeitungssubsystem setzt die im Management-Modell beschriebene Logik um, indem die darin angesiedelten Kompo-

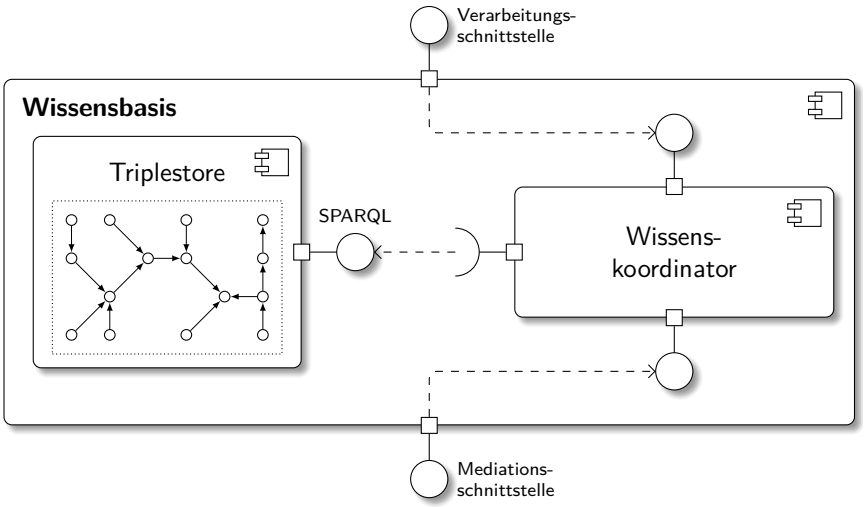


Abbildung 4.32.: Aufbau des Wissensbasissubsystems, bestehend aus dem passiven Triplestore und dem Wissenskoordinator.

nenten Daten analysieren und Rekonfigurationen planen, die auf dem überwachten System umgesetzt werden. In den folgenden Abschnitten sind die Komponenten der einzelnen Subsysteme beschrieben.

Wissensbasissubsystem

Aufgabe des Wissensbasissubsystems (siehe Abb. 4.32) ist es, das semantische Wissen zu verwalten und den anderen Komponenten bereitzustellen. Kern des Subsystems bildet der *Triplestore*, in dem das vollständige, ontologische Wissen des Anwendungsfalls abgelegt wird. Zur Laufzeit umfasst er die Framework-Basisontologien (4D-Fluents, DIO, OntML), die anwendungsfallspezifischen Domänen- und Management-Ontologien und die das überwachte System repräsentierenden Instanzdaten. Alle Informationen werden physisch gemeinsam in einem RDF-Graph gehalten; eine Unterteilung in die aufgeführten Kategorien existiert nur logisch. Nach außen bietet der Triplestore eine SPARQL-Schnittstelle, über die Wissen hinzugefügt, entfernt oder abgefragt werden kann. Er ist grundsätzlich passiv und wird von den anderen Framework-Komponenten getrieben.

Die SPARQL-Schnittstelle des Triplestores wird vom sogenannten *Wissenskoordinator* gekapselt, der eine einfache Zugriffskontrolle umsetzt. Er bietet nach außen die SPARQL-basierten Mediations- und Verarbeitungsschnittstellen an. Über die Mediationsschnittstelle dürfen während der Initialisierungsphase Daten geschrieben, während der Laufzeitphase ausschließlich gelesen werden. Sie bildet die Verknüpfung zum Mediationssubsystem und erlaubt es den darin angesiedelten Komponenten,

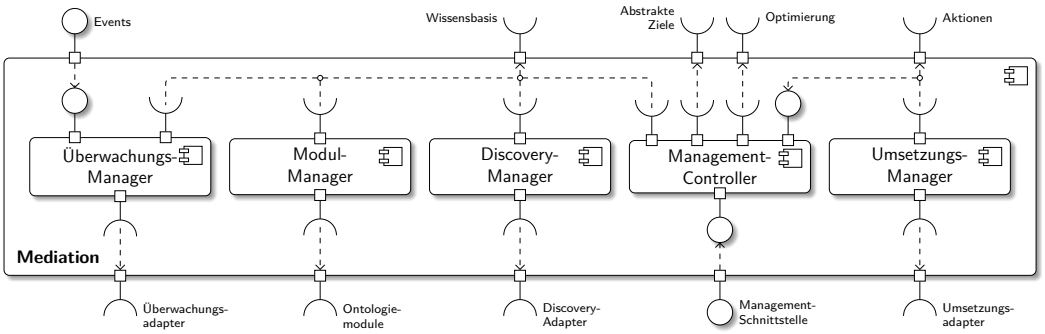


Abbildung 4.33.: Aufbau des Mediationssubsystems als Schnittstelle zum überwachten System.

initial Modelle und Instanzdaten in den Triplestore zu laden, zur Laufzeit jedoch ausschließlich lesend zuzugreifen. Von der Verarbeitungsschnittstelle darf während der Initialisierungsphase gelesen, während der Laufzeitphase gelesen und geschrieben werden. Sie bildet die Verknüpfung zum Verarbeitungssubsystem. Die darin angesiedelten Komponenten können sich so initial auf das Management-Modell und die Domänentaxonomie adaptieren und zur Laufzeit Wissensbasisoperationen auf das Systemmodell abbilden.

Mediationssubsystem

Das Mediationssubsystem (siehe Abb. 4.33) ist das Bindeglied zwischen dem Management-Framework und dem überwachten System. Es stellt Schnittstellen und Verwaltungskomponenten bereit, über die Modelle und Adapter eingebunden werden können.

Im Kern besteht das Subsystem aus den in Abschnitt 4.3.1 beschriebenen Modul- und Adapter-Managern. Jeder Manager ist für die Initialisierung und Verwaltung seines Komponententyps zuständig, deren Instanziierung und Konfiguration von der anwendungsfallspezifischen Framework-Konfiguration abhängt. Für jeden Komponententyp sieht das Framework eine zu implementierende Schnittstelle vor.

Ontologiemodule müssen Methoden zur Abfrage ihrer Fähigkeiten (bereitgestellte Ontologien), Abfrage ihrer Anforderungen (benötigte Ontologien) und einen Lademechanismus bereitstellen. Discovery-Adapter müssen eine Methode bereitstellen, über die der technologiespezifische Discovery-Mechanismus durchgeführt werden kann. Überwachungsadapter müssen eine blockierende Startmethode implementieren, die einen Event-Kanal erwartet und anschließend aktiv die Kommunikation mit

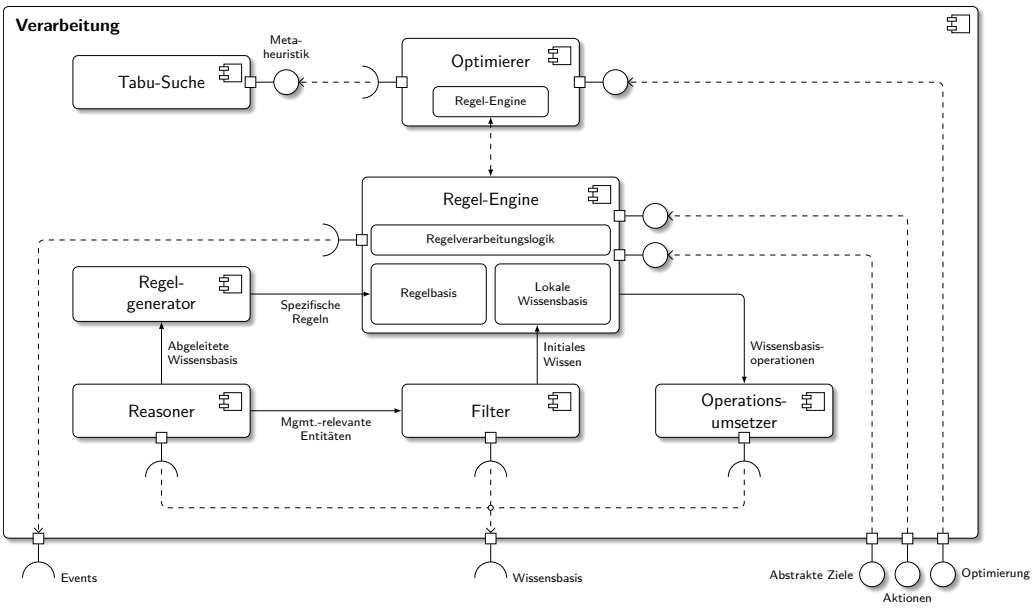


Abbildung 4.34.: Aufbau des Verarbeitungssubsystems.

der überwachten Komponente treibt. Umsetzungsadapter müssen Methoden bereitstellen, über die geprüft werden kann, ob sie eine Aktion umsetzen können, und die Aktionsumsetzung kommandiert werden kann. Der Management-Controller bietet nach außen eine Schnittstelle an, über die per SPARQL das Systemmodell abgefragt, abstrakte Ziele abonniert, der Optimierungsprozess angestoßen und Management-Aktionen kommandiert werden können.

Den anderen Subsystemen gegenüber bietet das Mediationssubsystem eine Event-Schnittstelle an, über die die von den Überwachungsadaptern erzeugten Events bereitgestellt werden. Selbst benötigt es Dienste, über die die Wissensbasis zugegriffen werden kann, sodass der Überwachungs-Manager das Schnittstellenmodell abfragen und die Events dagegen validieren, der Modul-Manager die Tripel der geladenen Ontologiemodule einfügen, der Discovery-Manager den initialen Systemzustand einpflegen und der Management-Controller SPARQL-Abfragen gegen das Systemmodell stellen kann. Zudem werden Dienste benötigt, über die abstrakte Ziele abonniert, mit der Optimierungskomponente interagiert und auszuführende Aktionen bezogen werden können.

Verarbeitungsubsystem

Im Verarbeitungssubsystem (siehe Abb. 4.34) wird im Laufzeitsystem die Management-Logik umgesetzt. Neben den eigentlichen Verarbeitungskomponenten umfasst

das Subsystem auch die Komponenten, die die Verarbeitungslogik in der Initialisierungsphase auf den Anwendungsfall adaptieren.

Kern des Subsystems bildet die Regel-Engine, deren lokale Wissensbasis gleichzeitig das in Abschnitt 4.2.2 beschriebene In-Memory-Abbild umfasst. Die von der Engine umzusetzenden Regeln werden in der Initialisierungsphase dynamisch aus den anwendungsfallspezifischen Modellen generiert.

Der Reasoner nutzt intern eine Regel-Engine, deren Regelbasis sich aus den generischen OWL-RL-Regeln und den Klassifikationsregeln für die transitive Hülle der Management-relevanten Klassen und Rollen zusammensetzt. Nachdem alle Ontologiemodule in die Wissensbasis geladen wurden, extrahiert der Reasoner über die entsprechende Wissensbasisschnittstelle alle Tripel, fügt sie in die lokale Wissensbasis seiner Regel-Engine ein und wendet die Regeln darauf an, um unbekannte Fakten über das Informations- und Management-Modell abzuleiten.

Der Regelgenerator nutzt das vom Reasoner bereitgestellte Wissen, um mit den in Abschnitt 4.3.1 beschriebenen Verfahren die OWL-RL-, Fluents- und OntML-Regeln zu generieren und in die Regelbasis der Engine einzufügen. Der Filter bildet die vom Reasoning abgeleiteten, Management-relevanten Entitäten auf ein SPARQL-Query ab, mit dem er nach Abschluss des Discovery-Prozesses einmalig alle im In-Memory-Abbild zu haltenden Fakten aus der Wissensbasis selektiert und in die lokale Wissensbasis der Regel-Engine einfügt.

Die Regel-Engine bezieht über eine Schnittstelle kontinuierlich Monitoring-Events, fügt sie in ihre Wissensbasis ein und verarbeitet sie. Daraus abgeleitete Wissensbasisoperationen werden (falls Management-relevant) auf die lokale Wissensbasis angewandt und an den Operationsumsetzer weitergegeben, der die Operationen als SPARQL-Queries auf die Wissensbasis anwendet. Über zwei Schnittstellen stellt die Regel-Engine nach außen die auszuführenden Aktionen und die abstrakten Ziele bereit.

Der Optimierer hält eine Kopie der Regel-Engine und bietet eine Schnittstelle, über die der Optimierungsprozess gestartet werden kann. Beim Optimierungsprozess nutzt er die bereitgestellte Metaheuristik, um eine Kopie der lokalen Wissensbasis zu optimieren. Die resultierende Rekonfigurationsagenda gibt er an den Aufrufer zurück. Als Standardmetaheuristik setzt das Laufzeitsystem die Tabu-Suche um.

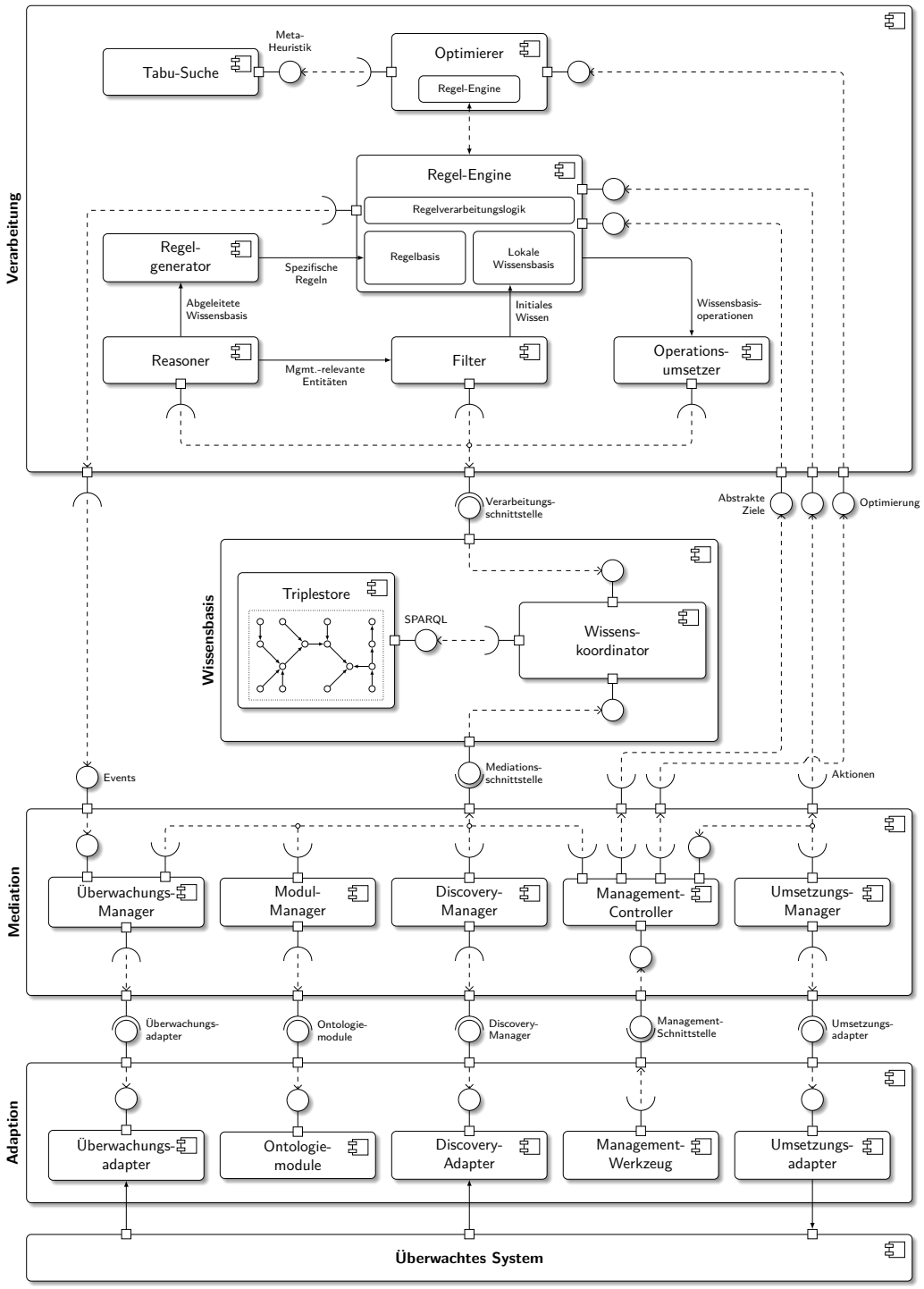


Abbildung 4.35.: Gesamtarchitektur bestehend aus Steuerungs-, Wissensbasis- und Mediationssubsystem.

Gesamtarchitektur

Durch Verknüpfung der Subsysteme entsteht die in Abbildung 4.35 dargestellte Gesamtarchitektur des Laufzeitsystems. Das Wissensbasissubsystem sitzt zentral und ermöglicht es den Komponenten des Verarbeitungs- und Mediationssubsystems, über den Wissenskoordinator auf die im Triplestore abgelegten Informationen zuzugreifen. Das Mediationssubsystem bindet die für einen Anwendungsfall benötigten Modelle und Adapter in das Laufzeitsystem ein. Der Überwachungs-Manager bezieht zur Laufzeit Events der Überwachungsadapter und stellt sie der Regel-Engine bereit. Die Engine fügt die Events in ihre lokale Wissensbasis ein, verarbeitet sie und erzeugt Wissensbasisoperationen, auszuführende Aktionen und abstrakte Ziele. Wissensbasisoperationen werden vom Operationsumsetzer auf dem Triplestore umgesetzt, Aktionen an den Umsetzungs-Manager und abstrakte Ziele an den Management-Controller weitergereicht. Der Umsetzungs-Manager führt Aktionen über Umsetzungsadapter auf dem überwachten System aus. Über den Management-Controller werden externe Management-Werkzeuge angebunden, die das Systemmodell abfragen, die Optimierung starten oder abstrakte Ziele abonnieren können. Das Laufzeitsystem wird auf konkrete Anwendungsfälle adaptiert, indem Ontologiemodule, Discovery-, Überwachungs- und Umsetzungsadapter bereitgestellt und konfiguriert werden.

4.4 Umsetzung

Die Software-Komponenten des in Abschnitt 4.3 entwickelten Frameworks wurden als Teil der vorliegenden Arbeit prototypisch umgesetzt. Dieser Abschnitt gibt eine kurze Übersicht der Laufzeitsystem- und Werkzeugimplementierung. Der Fokus liegt dabei auf den ausgewählten Technologien und den Schnittstellen, die zur Integration neuer Anwendungsfälle genutzt werden können.

4.4.1 Laufzeitsystem

Als Sprachplattform des Laufzeitsystems (siehe Abb. 4.35, Seite 129) wurde die Java Virtual Maschine (JVM) eingesetzt. Sie ist plattformunabhängig, verfügt über ein umfangreiches Ökosystem (Bibliotheken, Werkzeuge, etc.) und hat einen hohen Verbreitungsgrad. Die Komponenten wurden in Scala[2] bzw. Java[3] implementiert.

[2] http://scala-lang.org/
[3] http://www.oracle.com/technetwork/java/index.html

Für eine bessere Modularisierbarkeit, Erweiterbarkeit und Dienstabstraktion wurde OSGi[4] (ehemals *Open Services Gateway initiative*, heute eigenständiges Synonym) als Komponenten-Framework eingesetzt. OSGi's mehrschichtige Architektur setzt auf der JVM auf und besteht aus einer Ausführungsumgebung, einem Modulsystem, Lebenszyklen und Diensten, die vertikal von einem einheitlichen Sicherheitskonzept durchzogen sind. Komponenten sind darin in sogenannten Bundles organisiert, die bereitgestellte bzw. benötigte Ressourcen und Dienste deklarativ beschreiben. Der Lebenszyklus der Bundles (Starten, Stoppen, Dienstverknüpfung, etc.) wird dynamisch von der Ausführungsumgebung gesteuert. Als OSGi-Implementierung wurde Apache Felix[5], zur RDF-Verarbeitung Eclipse RDF4J[6] und als Build-Werkzeug Gradle[7] eingesetzt.

Das Laufzeitsystem setzt sich aus mehreren, lose gekoppelten Komponenten zusammen, die Subsystemfunktionalitäten realisieren. Die nach außen zur Integration neuer Anwendungsfälle bereitgestellten Schnittstellen sind im sogenannten *API*-Bundle zusammengefasst. Die *Ontology-Module*-Schnittstelle abstrahiert Ontologie-Module und deren Lademechanismus, die *Discovery-Adapter*-Schnittstelle Discovery-Adapter, die *Monitor-Adapter*-Schnittstelle Überwachungsadapter, die *Execute-Adapter*-Schnittstelle Ausführungsadapter und die *Management-Adapter*-Schnittstelle Management-Adapter. Die von Ontologiemodulen bereitgestellten und benötigten Ontologien werden über OSGi-Capabilities und -Requirements auf Komponentenebene ausgedrückt. RDF-Dokumente (inkl. Events und Aktionen) werden innerhalb des Laufzeitsystems als Mengen von RDF4J-Statements ausgetauscht. Management-Adaptern wird zur Laufzeit ein sogenannter *Management-Context* bereitgestellt, über den sie auf die Wissensbasis zugreifen, Wissensbasis-Updates und abstrakte Ziele abonnieren und die Optimierung anstoßen können. Um neue Ontologien und Adapter in das Framework zu integrieren, müssen die entsprechenden Schnittstellen implementiert und als OSGi-Komponenten in Bundles bereitgestellt werden.

Die *Knowledge*-Komponente setzt das Wissensbasissubsystem (siehe Abschnitt 4.3.2) um und nutzt dafür den von RDF4J bereitgestellten Triplestore. Der den Triplestore kapselnde Wissens-Manager setzt die Benachrichtigungsschnittstelle um, über die Management-Adapter reaktiv über Wissensbasisaktualisierungen informiert werden.

Die *Mediation*-Komponente setzt das Mediationssubsystem (siehe Abschnitt 4.3.2) um. Sie stellt den Module-, Discovery-Adapter-, Monitor-Adapter-, Execute-Adapter-

[4] `https://www.osgi.org/`
[5] `http://felix.apache.org/`
[6] `http://rdf4j.org/`
[7] `https://gradle.org/`

und Management-Adapter-Manager bereit. Jedem Manager werden beim Start des Laufzeitsystems die für den Anwendungsfall benötigten Adapterkonfigurationen seines Typs übergeben, für die er über den OSGi Configuration Admin Komponentenkonfigurationen anlegt. Der Configuration Admin erzeugt dann die entsprechenden Komponenteninstanzen, die der Manager anschließend ins Laufzeitsystem einbindet.

Die *Logic*-Komponente setzt die Management-Logik des Verarbeitungssubsystems (siehe Abschnitt 4.3.2) um. Dabei wird die Regel-Engine Drools[8] eingesetzt. Drools ist eine Business Rules Engine (BRE), die Forward-Chaining-basierte Produktionsregeln und Backward-Chaining-basierte Queries unterstützt. Sie kann in einem Streaming-Modus betrieben werden, der über Complex-Event-Processing-Mechanismen (Zeitfenster, Allen-Relationen, etc.) verfügt. Regeln werden in der Drools Rule Language (DRL) formuliert, die zur Laufzeit vom sogenannten PHREAK-Algorithmus umgesetzt werden. Der PHREAK-Algorithmus ist eine um Lazy-Evaluation erweiterte Version des Rete-OO-Algorithmus, der die partielle Auswertung in den Beta-Knoten maximal verzögert.

Die in der Drools-Instanz der Verarbeitungslogik eingesetzte Regelbasis wird dynamisch von dem in Abschnitt 4.3.1 beschriebenen Regelgenerierungsprozess erzeugt. Dabei wird zunächst von der *Reasoner*-Komponente ein einmaliges OWL-RL-Reasoning der geladenen Ontologiemodule durchgeführt, indem der manuell von RIF nach DRL transformierte OWL-RL-Regelsatz angewandt und etwaig unbekanntes Wissen materialisiert wird. Das resultierende Modell wird von der *Generator*-Komponente genutzt, um die zur Laufzeit einzusetzenden Regeln zu generieren. Die *OWL-Generator*-Komponente generiert Regeln, die *OntML-Generator*-Komponente transformiert die im Modell enthaltenen OntML-Module nach DRL. Die Vereinigung der Regelsätze bildet die Grundlage der Laufzeitverarbeitung.

Zur Laufzeit werden die als Management-relevant klassifizierten RDF-Statements der globalen Wissensbasis und die von den Überwachungsadaptern empfangenen Monitoring-Events in die lokale Wissensbasis (Working Memory) der Drools Session eingefügt und kontinuierlich verarbeitet. Von Produktionsregeln abgeleitete Operationen werden auf der lokalen Wissensbasis umgesetzt und von einer Synchronisationskomponente auf die globale Wissensbasis projiziert. Fluents werden im Working Memory der Regel-Engine als Qintupel gehalten und erst bei der Abbildung auf die Wissensbasis als Fluent-Strukturen materialisiert.

Die Repräsentation der RDF-Tripel und der Schnittstellentypen in der lokalen Wissensbasis kann über Konfigurationsparameter beeinflusst werden. Über den State-

[8] `http://drools.org/`

ment-Typ-Parameter kann festgelegt werden, wie die das überwachte System beschreibenden RDF-Tripel zur Laufzeit im Working-Memory der Regel-Engine gehalten werden. Beim generischen Ansatz werden die RDF4J-Statements unverändert eingefügt und verarbeitet. Beim spezifischen Ansatz wird für jede Rolle und Klasse ein eigener Drools-Datentyp generiert und die Statements beim Einfügen darauf abgebildet. Dies hat den Vorteil, dass die Regel-Engine bereits an den Blättern des Auswertungsnetzwerks auf den Typ filtern kann und für konkrete Rollen, für die im Domänenmodell ein Wertebereich festgelegt wurde, ein primitiver Datentyp für die Objektkomponente genutzt werden kann. So wird beispielsweise die Aussage

```
{ subject   = <#John>            :: IRI,
  predicate = <#age>             :: IRI,
  object    = "\"42\"^^xsd:int"  :: String
} :: Statement
```

in der Wissensbasis der Regel-Engine als

```
{ subject = <#John> :: IRI,
  object  = 42      :: Int
} :: Age
```

repräsentiert, wenn die Rolle `<#age>` den Wertebereich `xsd:int` hat. Dies erlaubt eine effizientere Indizierung und macht das kontinuierliche Entpacken des als RDF-Literal gekapselten Wertes bei der Regelauswertung unnötig. Mit der spezifischen Darstellungsform muss jedoch auch jedes zwischen Verarbeitungslogik und der globalen Wissensbasis ausgetauschte Tripel explizit konvertiert werden, was einen Mehraufwand bedeutet.

Über den Inlining-Parameter kann beeinflusst werden, wie die Schnittstellentypen im Working-Memory repräsentiert werden. Ohne Inlining setzt sich ein Event bzw. eine Aktion aus einer Kontextressource und einem RDF-Dokument zusammen. Soll in einer Regel ein Frame gegen ein solches Dokument ausgewertet werden, müssen alle darin enthaltenen Statements extrahiert und betrachtet werden. Bei aktiviertem Inlining wird während des Generierungsprozesses eine Analyse der Schnittstellentypenklasse durchgeführt. Dabei werden alle Rollen extrahiert, für die die Klasse auf eine maximale Kardinalität von Eins beschränkt ist. Für diese Rollen kann der Kontextressource im mitgeführten Dokument maximal ein Wert zugewiesen sein. Der generierte Schnittstellentyp wird für jede dieser Rollen um ein Attribut erweitert, in dem zur Laufzeit der im Dokument für die Kontextressource definierte Wert redundant abgelegt wird. Beim Hinzufügen eines Events bzw. einer Aktion werden

die entsprechenden Werte von einer Abbildungskomponente aus dem Dokument extrahiert und falls vorhanden die Attribute gesetzt. So wird beispielsweise das Event

```
{ context  = <#ev> :: IRI,
  document = [
    (<#ev>, rdf:type, <#ObservationEvent>),
    (<#ev>, <#value>, "\"17\"^^xsd:Int"  ),
    ...
  ] :: [Statement]
} :: Event
```

in der Wissensbasis der Regel-Engine als

```
{ context  = <#ev>              :: IRI,
  value    = "\"17\"^^xsd:Int" :: String,
  document = [
    (<#ev>, rdf:type, <#ObservationEvent>),
    (<#ev>, <#value>, "\"17\"^^xsd:Int"  ),
    ...
  ] :: [Statement]
} :: ObservationEvent
```

repräsentiert, wenn die Rolle `<#value>` auf eine maximale Kardinalität von Eins beschränkt ist. Wird nun in einer Regel eine solche Rolle für das Kontextindividuum referenziert, kann direkt auf das Attribut zugegriffen werden, ohne das Dokument durchsuchen zu müssen. Dies erlaubt eine effizientere Event- und Aktionsindizierung, kommt jedoch ebenfalls mit einem erhöhten Abbildungsaufwand einher.

Die spezifische Statement-Repräsentation wirkt sich auch auf die Event-Repräsentation aus, indem die in den Dokumenten gehaltenen Statements ebenfalls auf die spezialisierten Typen abgebildet und die für ein Inlining ausgewählten Attribute konkretisiert werden. In der Standardkonfiguration des Laufzeitsystems werden spezifische Statements und ein aktives Inlining genutzt. Ein Performance-Vergleich der unterschiedlichen Konfigurationen wird in Kapitel 5 durchgeführt.

Die *Optimization*-Komponente setzt die Optimierungslogik des Verarbeitungssubsystems (siehe Abschnitt 4.3.2) um. Sie nutzt eine konfigurierbare Anzahl von Kopien der Verarbeitungskomponente, mit deren Hilfe sie das Systemmodell unter Einsatz des in (siehe Abschnitt 4.3.1) beschriebenen Verfahrens optimiert. Dazu evaluiert sie parallel alle für den aktuellen Systemzustand vorgeschlagenen Management-Aktionen, indem sie sie nacheinander in die Drools-Session einfügt, alle Regeln auswertet, den resultierenden Score und Systemzustand ausliest und die Session anschließend

Komponente	Scala + Java SLOC
API (extern und intern)	767
Manager (inkl. Knowledge Base)	485
Mediation	318
Logic	1488
Reasoner	61
Generator	439
OntML-Generator	1485
OWL-Generator	236
Optimierung	143
Utility	423
Gesamt	5896

Tabelle 4.4.: Umfang der entwickelten Komponenten des Laufzeitsystems.

zurücksetzt. Sind alle Aktionen evaluiert, entscheidet die Optimierungsstrategie über die Auswahl einer durchzuführenden Aktion bzw. die Terminierung des Optimierungsprozesses. Als Standardstrategie wird die Tabu-Suche eingesetzt und mit einer einzigen Verarbeitungskomponentenkopie gearbeitet.

Die Komponenten des Laufzeitsystems werden von der *Manager*-Komponente verwaltet. Sie konfiguriert und verknüpft die von den Subsystemen bereitgestellten Dienste und koordiniert den Datenfluss. Tabelle 4.4 zeigt den Umfang der entwickelten Framework-Komponenten.

4.4.2 Werkzeuge

Damit eine Ontologie im Laufzeitsystem eingesetzt werden kann, muss eine OSGi-Komponente entwickelt werden, die die *Ontology-Module*-Schnittstelle implementiert und deren Fähigkeiten und Abhängigkeiten deklarativ beschreibt. Da Ontologien in der Regel jedoch nicht dynamisch generiert werden, sondern bereits als RDF-Dokumente vorliegen, bedeutet diese zusätzliche Abstraktion unnötigen manuellen Aufwand. Daher wurde mit dem *Module-Generator* ein Gradle-Plugin entwickelt, mit dem dieser Prozess automatisiert werden kann. Das Plugin liest die Ontologiemetadaten (IRI, Version und Imports) eines RDF-Dokuments aus, generiert daraus eine die Ontologiemodulschnittstelle implementierende OSGi-Komponente (Java-Klasse mit OSGi-Service-Annotationen) und verpackt sie gemeinsam mit dem Dokument in ein OSGi Bundle, das vom Laufzeitsystem geladen werden kann.

Werkzeug	Scala + Java SLOC
Module-Generator	152
OntML-IDE	1664
Gesamt	1816

Tabelle 4.5.: Umfang der entwickelten Werkzeuge des Frameworks.

Das in Abschnitt 4.3.1 beschriebene Modellierungswerkzeug wurde auf Basis des auf dem Eclipse Modeling Framework (EMF) aufsetzenden Xtext-Frameworks[9] realisiert. Xtext generiert aus einer Grammatik und einem Workflow eine vollständige integrierte Entwicklungsumgebung (IDE), bestehend aus Parser, Linker, Type Checker, Compiler Stub und Editor-Integration (Syntaxhervorhebung, Autovervollständigung, etc.) für Eclipse, IntelliJ IDEA oder den Web Browser. Es wurde eine OntML-Grammatik (siehe Abschnitt A.2.4, Seite 221) entwickelt, mit der sich Management-Module mit Hilfe einer stark an die RIF-Präsentationssyntax angelehnten DSL beschreiben lassen. Um die Entitäten importierter Domänenontologien in OntML referenzieren zu können, wurde ein EMF-Metamodell mit zugehörigem Modelladapter entwickelt. Der OntML-Compiler wandelt in der DSL beschriebene Management-Module in Ontologiedokumente um, die vom *Module-Generator* in Ontologiemodule verpackt und anschließend vom Laufzeitsystem geladen werden können. Abschnitt A.2.6 (siehe Seite 232) zeigt, wie das in Abschnitt A.2.5 (siehe Seite 227) dargestellte Beispiel in der OntML-DSL ausgedrückt werden kann. Tabelle 4.5 zeigt den Umfang der entwickelten Werkzeuge.

[9] `https://eclipse.org/Xtext/`

5 Evaluation

In diesem Kapitel werden die in Kapitel 4 vorgestellten Konzepte und das daraus entwickelte Framework und Laufzeitsystem evaluiert. Dazu wird anhand zweier Fallstudien die allgemeine Anwendbarkeit des Frameworks gezeigt, der damit verbundene Integrationsaufwand untersucht und das resultierende Laufzeitverhalten betrachtet. Anschließend werden unter Bezug auf die Fallstudien die zu Beginn dieser Arbeit in Abschnitt 1.4 (siehe Seite 5) definierten Ziele gegen den entwickelten Ansatz geprüft.

Abschnitt 5.1 präsentiert einen Anwendungsfall aus dem Bereich des Storage Managements, bei dem virtuelle Festplatten anhand definierter Service Level Agreements automatisch überwacht und reguliert werden. Abschnitt 5.2 präsentiert einen Anwendungsfall aus dem Flugsicherungskontext, bei dem Radare überwacht und deren Zuteilung zu Observationsbereichen bei Ausfällen mit resultierenden SLA-Verletzungen autonom rekonfiguriert werden. Beide Anwendungsfälle wurden gemeinsam mit Industriepartnern anhand realer Problemstellungen entwickelt. Die anschließend in Abschnitt 5.3 durchgeführte Anforderungsüberprüfung findet auf Basis der Fallstudien statt.

Für alle durchgeführten Messungen wurde ein Desktop-PC mit einer Intel® Core™ i7-3770 CPU @ 3.40GHz und 16GB Arbeitsspeicher eingesetzt, auf dem ein Ubuntu 16.04 (64 Bit) mit Java 8 (1.8.0_111, 64 Bit) betrieben wurde.

5.1 Storage-Management-Fallstudie

Unter Speichervirtualisierung versteht man eine Technik, bei der die Eigenschaften physischer Speichermedien (Festplatten, Bandlaufwerke, etc.) durch eine Abstraktionsschicht aufgebrochen werden. Dem Benutzer wird virtueller Blockspeicher bereitgestellt, dessen Verwaltung und Abbildung auf ein oder mehrere physische Medien (Virtualization und Translation) von einem Storage Virtualization Manager (SVM) [92] realisiert wird. Der Storage Virtualization Manager ist eine Software, die eine als Storage Area Network (SAN, Menge von über Netzwerk-Switches zusammengefassten Storage Devices) angelegte, verteilte Speicherinfrastruktur organisiert.

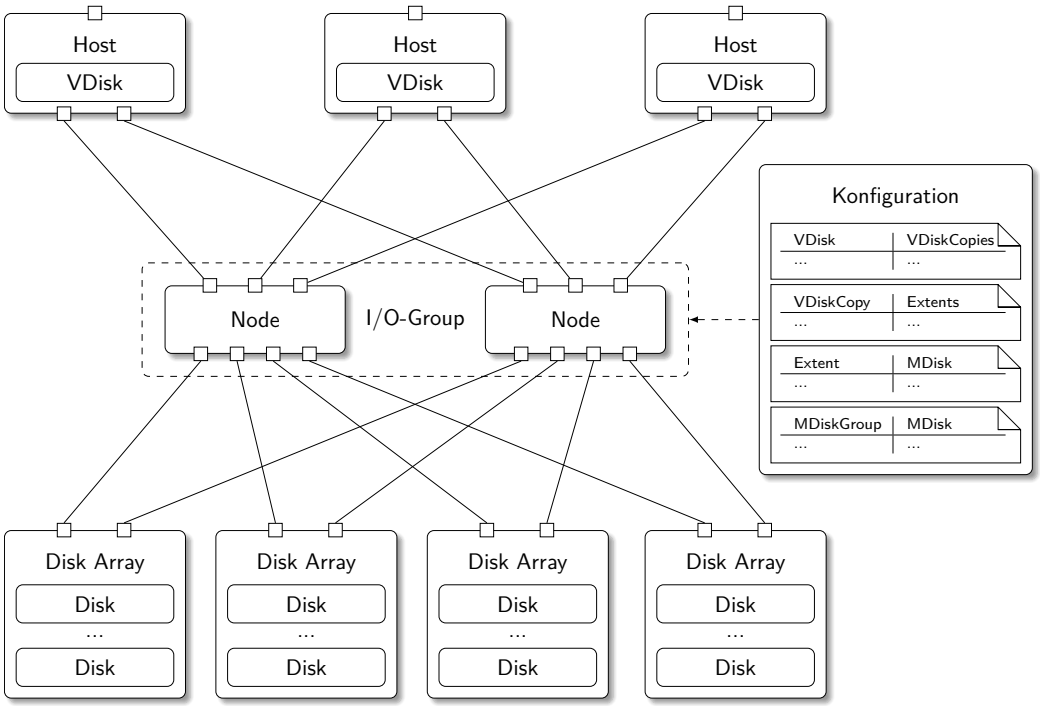


Abbildung 5.1.: Vereinfachtes Modell der IBM SVC Speicherinfrastruktur.

Der IBM SAN Volume Controller[1] (SVC, Abbildung 5.1) ist eine Verknüpfung von Hard- und Software-Komponenten, die gemeinsam eine SVM-basierte Speichervirtualisierung umsetzen. Physische, in Disk Arrays organisierte Platten werden über RAID-Mechanismen logisch zu sogenannten Managed Disks (MDisks) zusammengefasst, die Blockspeicher darstellen. Aus einer oder mehreren solcher MDisks werden Speicher-Pools (MDisk Group) gebildet, die jeweils in Speichereinheiten (Extents) gleicher Größe unterteilt werden. Aus den Extents eines Pools wird virtueller Speicher (VDisks oder Volumes) geformt, der ein oder zwei reale Speicherabbilder (VDisk Copies) hat, die zwischen Pools gleicher Extent-Größe verschoben werden können. Der Zugriff auf den virtuellen Speicher geschieht über sogenannte SVC-Nodes, die eine Kombination aus Netzwerk-Switch und SVM sind. Sie kennen die Storage-Konfiguration und setzen Lese- und Schreiboperationen von VDisks auf MDisks um. Zwei Nodes bilden eine I/O-Gruppe (I/O-Group) und duplizieren gegenseitig ihre Schreiboperationen. Bis zu vier I/O-Gruppen bilden ein Cluster, dessen Gesamtkonfiguration von einem Master Node verwaltet wird. Virtueller Speicher ist immer genau einer I/O-Gruppe zugeordnet und kann über deren Nodes zugegriffen werden. Hosts nutzen den virtuellen Blockspeicher, um nach außen Storage-Dienste wie Dateisysteme anzubieten.

[1] `http://www-03.ibm.com/systems/de/storage/software/virtualization/svc/`

Die Firma System Vertrieb Alexander GmbH (SVA)[2] entwickelt und vertreibt mit Business Volume Qualicision (BVQ)[3] eine „Monitoring-, Performance-Analyse- und Reporting-Lösung für die Speichersysteme IBM SAN Volume Controller (SVC) und die IBM Storwize Familie“. BVQ verknüpft die Entitäten und Performance-Messwerte einer SVC-Konfiguration auf einer höheren Abstraktionsebene mit anderen Domänen. So können beispielsweise virtuelle Disks mit denen sie nutzenden virtuellen Maschinen verknüpft werden oder VDisk-Gruppen gebildet und mit Speicherklassen und SLAs versehen werden. In BVQ eingebettete Analyseverfahren berechnen offline für einen Konfigurationsschnappschuss KPIs, die dem Benutzer mit den Basisdaten verknüpft, interaktiv dargestellt werden. Darüber hinaus können statische Reports erzeugt werden.

Im gemeinsamen Forschungsprojekt *Ontologiebasiertes Storage Management* (OntoStorM)[4] wurde untersucht, inwiefern sich das bisher hart-kodierte Informations- und Analysemodell von BVQ durch semantische Technologien ersetzen lässt. Darin wurden (ohne Beteiligung des Autors) die existierenden, strukturierten BVQ-Modelle und die Analyselogik automatisiert in OWL-Ontologien und SPARQL-Queries transformiert und ein Modelladapter entwickelt, der zwischen den Repräsentationen vermittelt [115]. Auf Basis dieser Vorarbeiten wurde im Folgeprojekt *TOMATO Ontology Management Toolkit* (TOMATO)[5] das in der vorliegenden Arbeit entwickelte Management-Framework vom Autor angewandt, um ein automatisiertes Echtzeit-Management einer SVC-Infrastruktur umzusetzen. Anders als bei der bisher in BVQ umgesetzten offline-Analyse zu diskreten Zeitpunkten (typischerweise ein Report pro Tag) sollte dabei eine kontinuierliche Systemüberwachung stattfinden, bei der stetig die SLA-Konformität geprüft und ggf. automatisiert steuernd ins System eingegriffen wird.

5.1.1 Anwendungsfall

Als Anwendungsfall wurde das automatisierte QoS-Management von High-Cost-Speicher ausgewählt. Virtueller Speicher wird in BVQ in Speicherklassen eingeteilt; typischerweise Gold, Silber und Bronze. Gold ist die teuerste Speicherklasse mit den höchsten Verfügbarkeits- und Antwortzeitzusicherungen. Virtueller Speicher dieser Klasse wird in der Regel redundant auf Solid State Disks (SSDs) abgelegt und von Performance-kritischen Anwendungen (z. B. mit direkter Benutzerinteraktion)

[2] http://www.sva.de/
[3] http://www.sva.de/solutions/storage/bvq_storage.html
[4] https://www.hs-rm.de/de/fachbereiche/design-informatik-medien/forschungsprofil/ontostorm/
[5] https://www.hs-rm.de/de/fachbereiche/design-informatik-medien/forschungsprofil/tomato/

genutzt. Silber ist die mittlere Speicherklasse, die in der Regel SSDs und Hard Disk Drives (HDDs) mischt, um eine ausgewogene Performance zu erzielen. Bronze ist die günstigste Speicherklasse mit den schwächsten QoS-Zusicherungen. Ihre virtuellen Disks werden in der Regel auf HDDs abgebildet und von Performance-unkritischen Anwendungen (z. B. einem Batch Job oder zur Archivierung) genutzt. Die mit den Speicherklassen verknüpften SLAs legen maximale Antwortzeiten (z. B. 15 ms bei Gold) und Anzahl zulässiger Antwortzeitüberschreitungen (z. B. 5 pro Tag) fest. Um Vertragsstrafen zu vermeiden, muss der Storage-Anbieter die vereinbarte Service-Qualität zu jedem Zeitpunkt sicherstellen. Häufig wird deshalb überprovisioniert, sodass selbst unter Spitzenlasten keine Engpässe entstehen können. Kostendruck und der Trend hin zur Green-IT [91] zwingen die Storage-Betreiber jedoch immer stärker zu einer Unterprovisionierung, bei der die Verteilung der virtuellen Disks anhand ihres typischen I/O-Verhaltens optimiert wird. So wird beispielsweise der Speicher einer nur tagsüber eingesetzten Benutzeranwendung mit dem Speicher eines nur nachts ausgeführten Batch Jobs im gleichen Speicher-Pool gehalten, obwohl die Summe beider Spitzenlasten über dessen Kapazität liegt. Wird der Batch Job nun außerplanmäßig tagsüber von einem Nutzer angestoßen, kommt es zu Engpässen. Die maximale Bandbreite der Fibre Channel Ports wird überschritten, die Puffer laufen voll, die Disks stehlen sich gegenseitig den Cache, etc. Als Folge steigt die Antwortzeit der virtuellen Disks rapide an, die maximale Antwortzeit des High-Cost-Speichers wird überschritten und die SLAs werden gebrochen.

Als probates Gegenmittel hat sich bei den SVA-Domänenexperten das sogenannte I/O-Governing des Low-Cost-Speichers erwiesen. Dabei wird auf den SVC-Nodes für eine virtuelle Disk eine maximale I/O-Rate festgelegt, die sie nicht überschreiten kann. Eine generalisierte Definition statischer I/O-Grenzen für Low-Cost-Speicher führt jedoch dazu, dass diese unter Niedriglast nicht ihr volles Potential entfalten können und verfügbare Ressourcen ungenutzt bleiben. Daher sollte mit Hilfe des entwickelten Frameworks ein automatisiertes Management umgesetzt werden, bei dem die Performance-Werte des Storage-Systems kontinuierlich überwacht und Low-Cost-Speicher eines Pools bei akuter Überlast temporär eingeschränkt wird, um High-Cost-Speicher des Pools zu schützen.

5.1.2 Umsetzung

Zur Umsetzung des Anwendungsfalls wurden gemäß der Adaptionsphase des Frameworks (siehe Abschnitt 4.3.1, Seite 108) zunächst im Modellierungsschritt die ontologischen Modelle ausgewählt oder entwickelt. Für das Modell des Storage-Systems

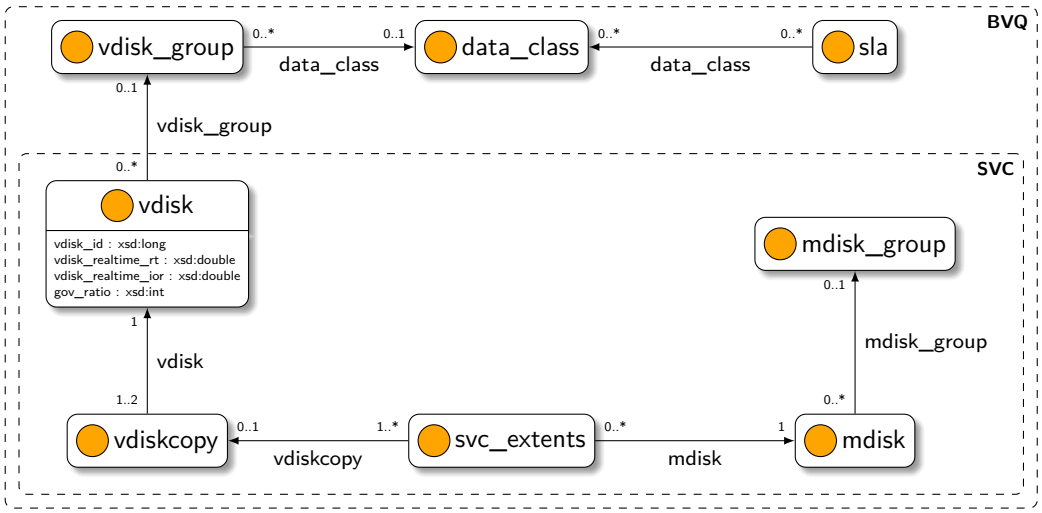


Abbildung 5.2.: Für den Anwendungsfall relevante Teilausschnitte des SVC- und BVQ-Ontologiemodells.

wurden die im Vorprojekt entwickelten Ontologien eingesetzt. Zwar wird im Folgenden in eine SVC- und BVQ-Ontologie unterschieden, in der Realität handelt es sich jedoch um ein in einem einzelnen Dokument verbundenes Modell. Abbildung 5.2 zeigt die für den Anwendungsfall relevanten Teilausschnitte, die für eine kompaktere Darstellung in einer UML-artigen Notation gehalten sind.

Die SVC-Ontologie definiert auf der technischen Ebene die Speichervirtualisierungsentitäten. Eine virtuelle Disk (`vdisk`) ist über eine ID (`vdisk_id`) eindeutig identifizierbar und hat eine aktuelle Antwortzeit (`vdisk_realtime_rt`), I/O-Rate (`vdisk_realtime_ior`) und I/O-Governing-Rate (`gov_ratio`). Von ihr existieren ein oder zwei Speicherabbilder (`vdiskcopy`), die sich aus Speichereinheiten (`svc_extents`) zusammensetzen, die aus Managed Disks (`mdisks`) eines Pools (`mdisk_group`) geschnitten wurden.

Die BVQ-Ontologie definiert auf der Management-Ebene logische Entitäten, mit denen die technischen Entitäten gruppiert und mit QoS-Eigenschaften verknüpft werden. Eine virtuelle Speichergruppe (`vdisk_group`) fasst mehrere virtuelle Disks zusammen und verknüpft sie mit einer Speicherklasse (`data_class`). Die Speicherklasse kann wiederum mit SLAs (`sla`) versehen werden, die für alle virtuellen Disks der Gruppen der Speicherklasse gelten.

Anschließend wurde manuell die in Abbildung 5.3 dargestellte Anwendungsfallontologie entwickelt. Sie importiert die Storage-Modelle und erweitert sie um für die Fallstudie benötigte Aspekte. Die konkreten Rollen `vdisk_realtime_rt`, `vdisk_realtime_ior` und `gov_ratio` wurden als temporal deklariert, da sich die

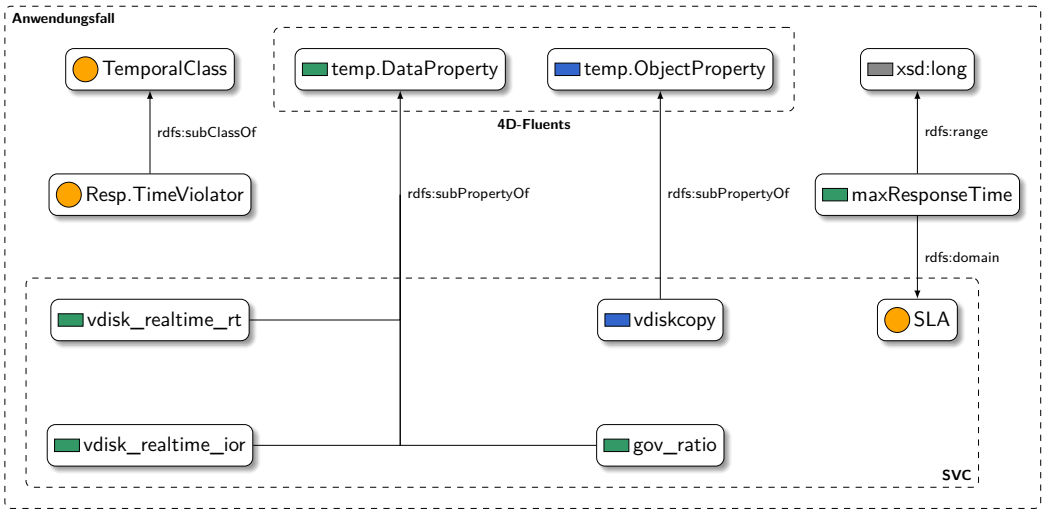


Abbildung 5.3.: Anwendungsfallontologie zur temporalen Auszeichnung von SVC-Rollen und Definition neuer Konzepte und Rollen.

Antwortzeit und die I/O-Rate bei jeder Messung und das Governing bei jedem Management-Eingriff verändert. Die abstrakte Rolle `vdiskcopy` wurde ebenfalls als temporal deklariert, um das Verschieben virtueller Disks zwischen Pools repräsentieren zu können. Zusätzlich wurde die temporale Klasse `ResponseTimeViolator` eingeführt, die eine Überschreitung der zulässigen Antwortzeit einer VDisk signalisiert. Da die existierenden SLA-Ausdrucksmöglichkeiten des BVQ-Modells sehr eingeschränkt sind, wurde die konkrete Rolle (`maxResponseTime`) eingeführt, die ein SLA mit einer maximalen Antwortzeit verknüpft.

Im nächsten Schritt wurde die in Abbildung 5.4 dargestellte Schnittstellenontologie entworfen. Auf Basis der Domain Interface Ontology (siehe Abschnitt 4.2.1, Seite 83) wurden darin die für den Anwendungsfall relevanten Monitoring-Events und Management-Aktionen modelliert. Ein `VDiskMetric`-Event repräsentiert die QoS-Messung einer virtuellen Disk. Es umfasst den eindeutigen Identifier (`vDiskId`) und die durchschnittliche Antwortzeit (`responseTime`) und I/O-Rate (`iops`) des Messintervalls. Das `Governed`-Event drückt aus, dass ein I/O-Governing einer VDisk vorgenommen wurde. Es umfasst den eindeutigen Identifier (`vDiskId`) und die festgelegte I/O-Grenze (`iops`) der VDisk. Analog dazu existiert das `Ungoverned`-Event, das aussagt, dass für eine VDisk mit dem eindeutigen Identifier (`vDiskId`) die I/O-Grenze aufgehoben wurde. Mit den Management-Aktionen `Govern` bzw. `Ungovern` kann dem Storage-System die I/O-Grenze einer VDisk kommandiert werden. Beide Aktionen umfassen den eindeutigen Identifier einer VDisk (`vDiskId`), die `Govern`-Aktion zusätzlich die kommandierte I/O-Grenze (`iops`). Mit dem `GoverenedHint` wurde zusätzlich ein synthetisches Event eingeführt, das innerhalb der Management-

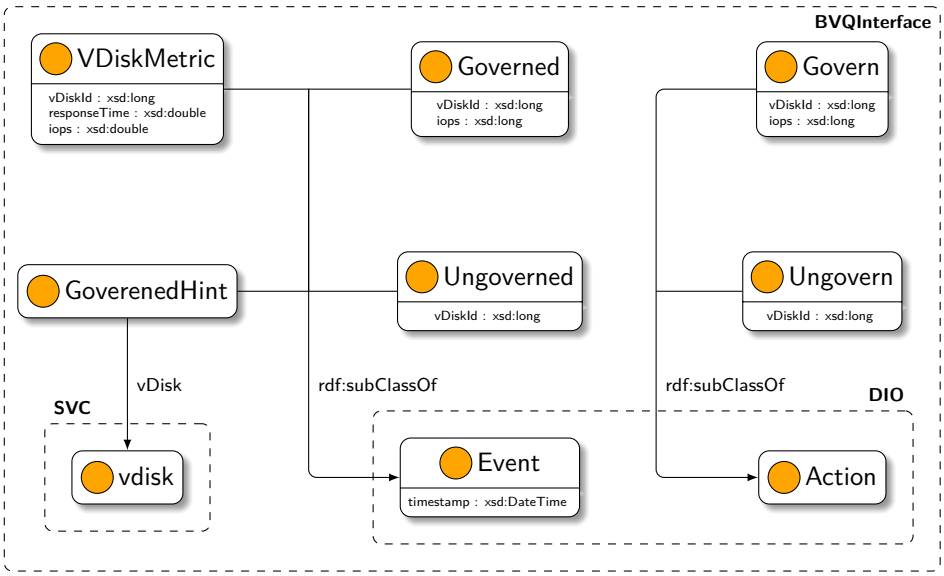


Abbildung 5.4.: Schnittstellenontologie zur Definition der Monitoring-Events und Management-Aktionen von BVQ.

Ontologie	Klassen	abstrakte Rollen	konkrete Rollen	Individuen	Axiome
BVQ und SVC	68	67	899	64	12139
Anwendungsfall	1	0	2	3	21
Schnittstelle	7	1	3	0	29
Gesamt	76	68	904	67	12189

Tabelle 5.1.: Umfang der entwickelten Domänen- und Schnittstellenontologien des Storage-Management-Anwendungsfalls.

Logik signalisiert, dass für eine VDisk (`vDisk`) zu einem bestimmten Zeitpunkt ein Governing kommandiert wurde. Tabelle 5.1 zeigt den Umfang der entwickelten Domänen- und Schnittstellenontologien.

Auf Basis der ausgewählten und entwickelten Ontologien wurde anschließend mit der OntML-DSL in der zugehörigen IDE (siehe Abschnitt 4.4, Seite 130) die Management-Logik des Anwendungsfalls definiert. Sie ist in die vier Management-Module Paths, Events, SLA und Governing unterteilt:

Das Paths-Modul (siehe Abschnitt A.3.1, Seite 233) definiert abstrakte Ziele, Abfragen und funktionale Abhängigkeiten, die Beziehungen zwischen den Entitäten des Domänenmodells herstellen. Das abstrakte Ziel *mdg_vdisks* und das dazugehörige Query „MDisk Group VDisks“ stellen über die MDisks, Extents und VDisk Copies eine Verknüpfung zwischen Pools und den darin liegenden virtuellen Disks her. Das abstrakte Ziel *mdg_ior* und das dazugehörige Query „MDisk Group Aggregated I/Os“

summieren die I/O-Rate aller VDisks eines Pools auf. Die funktionale Abhängigkeit „VDisk Data Class“ weist jeder VDisk die Speicherklasse ihrer VDisk-Gruppe zu. Die funktionale Abhängigkeit „VDisk Response Time Limit“ weist jeder VDisk die maximale Antwortzeit zu, die im SLA ihrer Speicherklasse definiert ist.

Das Event-Modul (siehe Abschnitt A.3.2, Seite 235) definiert Abbildungsregeln, die Events auf Wissensbasisoperationen abbilden. Die Abbildungsregel „Map VDisk Metric“ extrahiert aus einem Metrik-Event die VDisk-ID, die Antwortzeit und die I/O-Rate, sucht die VDisk mit der entsprechenden ID, löscht die alte Antwortzeit und die alte I/O-Rate und setzt die neuen Werte. Da es sich bei Antwortzeit und I/O-Rate um temporale Rollen handelt, werden die alten Werte nicht wirklich gelöscht, sondern nur deren Fluent-Intervalle geschlossen. Die Abbildungsregeln „Map Governed“ bzw. „Map Ungoverned“ aktualisieren bzw. entfernen die I/O-Einschränkung der über die ID referenzierten VDisk.

Das SLA-Modul (siehe Abschnitt A.3.3, Seite 236) legt fest, wann eine VDisk gegen ihr SLA verstößt. Die funktionale Abhängigkeit „Response Time Violator“ inferiert die Zugehörigkeit einer VDisk zur temporalen Klasse `ResponseTimeViolator`, wenn ihre aktuelle I/O-Rate die maximale I/O-Rate überschreitet.

Das Governing-Modul (siehe Abschnitt A.3.4, Seite 237) definiert die Steuerungslogik des Anwendungsfalls. Es umfasst zum einen abstrakte Ziele und Abfragen, mit denen VDisks bestimmter Eigenschaften gefunden werden können, zum anderen Policies, die Management-Aktionen auslösen. Das abstrakte Ziel *mdg_bronze_vdisk* mit dem zugehörigen Query „Bronze VDisk in MDisk Group“ findet für einen Pool alle VDisks mit der Speicherklasse *Bronze*. Das abstrakte Ziel *mdg_bronze_vdisk_with_max_iops* mit dem zugehörigen Query „Bronze VDisk with max. IO/s in MDisk Group“ filtert daraus die VDisk mit der höchsten I/O-Rate. Das abstrakte Ziel *mdg_goverened_vdisk* mit dem zugehörigen Query „Governed VDisk in MDisk Group“ findet für einen Pool alle VDisks mit I/O-Einschränkung. Das abstrakte Ziel *mdg_strongest_govered_vdisk* mit dem zugehörigen Query „Strongest Governed VDisk in MDisk Group“ filtert daraus die VDisk mit der stärksten Einschränkung (niedrigste zugelassene I/O-Rate).

Die Policy „Govern Bronze VDisk“ prüft, ob die Auslastung eines Pools über 90% liegt. Ist dies der Fall, sucht sie die bronzene VDisk mit der höchsten I/O-Rate des Pools und erzeugt eine Governing-Aktion, die die I/O-Rate auf 50% der aktuellen I/O-Rate beschränkt. Die Policy „Relax Governing“ prüft, ob es in einem Pool mit einer Auslastung unter 90% eine eingeschränkte VDisk gibt. Ist dies der Fall, wird geprüft, ob eine Verdopplung der I/O-Rate der am stärksten eingeschränkten VDisk die Auslastung überschreiten würde und falls nicht, wird die entsprechende Governing-

OntML-DSL				RDF-Serialisierung		
Modul	abstrakte Ziele	Regeln	SLOC	Individuen	Klassen-zuweisungen	Rollen-zuweisungen
Paths	2	4	48	81	199	140
Events	0	3	50	98	260	153
SLA	0	1	15	19	44	28
Governing	4	7	116	372	1114	664
Gesamt	6	15	229	570	1617	985

Tabelle 5.2.: Umfang der entwickelten Management-Module des Storage-Management-Anwendungsfalls.

Aktion kommandiert. Die Policy „Ungovern VDisk“ entfernt die I/O-Einschränkung einer VDisk vollständig, wenn der Pool nicht überlastet ist und die VDisk nicht an ihr Limit kommt. Alle Policies stellen zusätzlich sicher, dass Governing-Aktionen nur in bestimmten Zeitabständen ausgelöst werden, indem sie das Governing-Hint-Event binden und erzeugen. Tabelle 5.2 zeigt den Umfang der entwickelten Management-Module.

Für die Interaktion zwischen dem überwachten System und dem Laufzeitsystem wurden im Anbindungsschritt der Adaptionsphase entsprechende Adapter entwickelt. Der Anwendungsfall wurde dabei als Simulation angelegt, bei der das überwachte System von einer Softwarekomponente emuliert wird, die durch die Generierung synthetischer Daten gezielt Engpässe hervorrufen kann. Dabei wurde eng mit den Domänenexperten von SVA zusammengearbeitet, um möglichst realistische Szenarien zu erzeugen.

Der entwickelte BVQ-Discovery-Adapter ist dafür zuständig, initial eine Storage-Konfiguration bereitzustellen, die in die Wissensbasis importiert werden kann. Bei einer realen Anbindung könnte der im Vorprojekt entwickelte Modelladapter genutzt werden, um eine ontologische Repräsentation der Systemkonfiguration zu erzeugen. Bei der Simulation werden in der Adapterkonfiguration mehrere Pools definiert, für die der Discovery-Adapter VDisks-, VDisk-Copy-, Extent-, MDisk-, MDisk-Group-, VDisk-Group-, Data-Class- und SLA-Instanzen erzeugt und miteinander verknüpft. Zusätzlich initialisiert er ein Simulationsmodell, das die gemeinsame Informationsbasis aller Adapter bildet.

Der BVQ-Monitor-Adapter erzeugt VDisk-Metriken, Governing- und Ungoverning-Events. Bei einer realen Anbindung würde er die Überwachungsschnittstelle von BVQ nutzen, um die Performance-Werte zyklisch auszulesen, in dem Schnittstellenmodell entsprechende RDF-Dokumente umzuwandeln und diese in das System einzuspeisen.

Komponente	Java SLOC
BVQ-Monitor-Adapter	223
BVQ-Discovery-Adapter	94
BVQ-Management-Adapter	139
BVQ-Execute-Adapter	52
BVQ-Management-User-Interface	910
Gesamt	1415

Tabelle 5.3.: Umfang der entwickelten Anbindungskomponenten des Storage-Management-Anwendungsfalls.

Für die Simulation wird er mit einer erwarteten Antwortzeit μ_r, Antwortzeitvarianz σ_r^2 und I/O-Varianz σ_{io}^2 konfiguriert. Die erwartete I/O-Rate μ_{io} der VDisks kann über eine grafische Benutzeroberfläche dynamisch oder über die Konfiguration statisch beeinflusst werden. In einem festgelegten Intervall erzeugt der Adapter für jede virtuelle Disk eine Metrik, die ihre simulierte I/O-Rate und Antwortzeit enthält. Zur Berechnung der I/O-Rate wird aus der Normalverteilung $\mathcal{N}(\mu_{io}, \sigma_{io}^2)$ eine aktuell geforderte I/O-Rate gezogen. Die geforderte I/O-Rate jeder VDisk wird mit der jeweiligen Governing-Rate auf eine zulässige I/O-Rate beschränkt, wie es beim realen System der angefragte SVC-Node umsetzen würde. Anschließend wird die zulässige I/O-Rate mit der Gesamtauslastung des Pools skaliert, falls dieser überlastet ist. Der skalierte Wert repräsentiert die gemessene I/O-Rate der VDisk. Für die Antwortzeit wird aus der Normalverteilung $\mathcal{N}(\mu_r, \sigma_r^2)$ eine erwartete Antwortzeit gezogen und mit dem Verhältnis zwischen gemessener I/O-Rate und geforderter I/O-Rate potenziert. Damit wird der von den Domänenexperten beschriebene, exponentielle Antwortzeitanstieg bei Überlast angenähert. Der resultierende Wert repräsentiert die gemessene Antwortzeit der VDisk. Zusätzlich überwacht der Monitor-Adapter das Simulationsmodell und erzeugt Events, wenn das I/O-Governing einer VDisk angepasst wurde. Für jede virtuelle Disk wird ein separater, mit der ID initialisierter Zufallszahlengenerator genutzt, um simulationsübergreifend deterministische Werte zu erhalten.

Der BVQ-Execute-Adapter empfängt die von der Verarbeitungslogik erzeugten Governing-Aktionen, extrahiert aus dem RDF-Dokument die VDisk-ID und die I/O-Rate und passt das Simulationsmodell an. Der BVQ-Management-Adapter visualisiert den von der Wissensbasis repräsentierten Systemzustand und erlaubt es, die erwartete I/O-Rate der VDisks anzupassen. Tabelle 5.3 zeigt den Umfang der entwickelten Anbindungskomponenten.

Für die Modelle und Adapter wurde ein Build-Prozess definiert, der die Ontologien mit dem *Module-Generator* automatisch in Ontologiemodule transformiert und für alle anwendungsfallspezifischen Komponenten OSGi-Bundles erzeugt. Zudem wurde eine Framework-Konfiguration erstellt, die die zu ladenden Ontologie-Module und die Adapter-Konfigurationen beschreibt.

Nach Abschluss der Adaptionsphase wurden die entwickelten Ontologie-Module und Adapter in das Laufzeitsystem integriert, indem die Bundles und die Konfiguration zur OSGi-Laufzeitumgebung hinzugefügt wurden, die aus den in Abschnitt 4.4 (siehe Seite 130) vorgestellten Komponenten und deren Abhängigkeiten besteht.

5.1.3 Laufzeitverhalten

Um das Laufzeitverhalten des Storage-Management-Anwendungsfalls zu untersuchen, wurde ein Storage-System aus zehn Pools mit jeweils 15 VDisks (5 goldenen, 5 silbernen und 5 bronzenen) simuliert, bei dem jede VDisk pro Sekunde eine VDisk-Metrik erzeugt, die auf die Wissensbasis abgebildet und analysiert wird. Zusätzlich wurde das I/O-Verhalten der VDisks so konfiguriert, dass alle 60 Sekunden eine Überlastsituation entsteht, die ein Eingreifen des autonomen Managers erforderlich macht. Dieser kommandiert dann ein Governing von Low-Cost-Disks, um die High-Cost-Disks zu schützen. Ist der finale Governing-Zustand erreicht, wird die I/O-Last auf Normalniveau zurückgesetzt, was dazu führt, dass der Manager zunächst das Governing lockert und schließlich vollständig entfernt. Das Storage-System kehrt so vor der nächsten Überlastsituation in seinen Ausgangszustand zurück.

Beim Start des Laufzeitsystems wurden für den simulierten Anwendungsfall im Wissensbasisinitialisierungsschritt der Initialisierungsphase (siehe Abschnitt 4.3.1, Seite 114) aus den relevanten Ontologie-Modulen 38 980 RDF-Tripel in die Wissensbasis importiert. Im nachfolgenden Regelgenerierungsschritt wurde die Wissensbasis beim generischen OWL-RL-Reasoning auf 152 686 Tripel erweitert. Aus den daraus extrahierten Management-relevanten Entitäten und OntML-Modulen wurden in der Standardkonfiguration 890 Zeilen DRL-Regeln erzeugt, die die Regelbasis der Laufzeitverarbeitung bilden. Im Komponenteninitialisierungsschritt wurden die entwickelten BVQ-Adapter geladen und ins Laufzeitsystem integriert. Beim Discovery-Schritt wurden vom BVQ-Discovery-Adapter 2526 Tripel zur Wissensbasis hinzugefügt.

Zu Beginn der Laufzeitphase (siehe Abschnitt 4.3.1, Seite 120) wurden anhand der als Management-relevant klassifizierten Entitäten 826 reguläre und 150 temporale

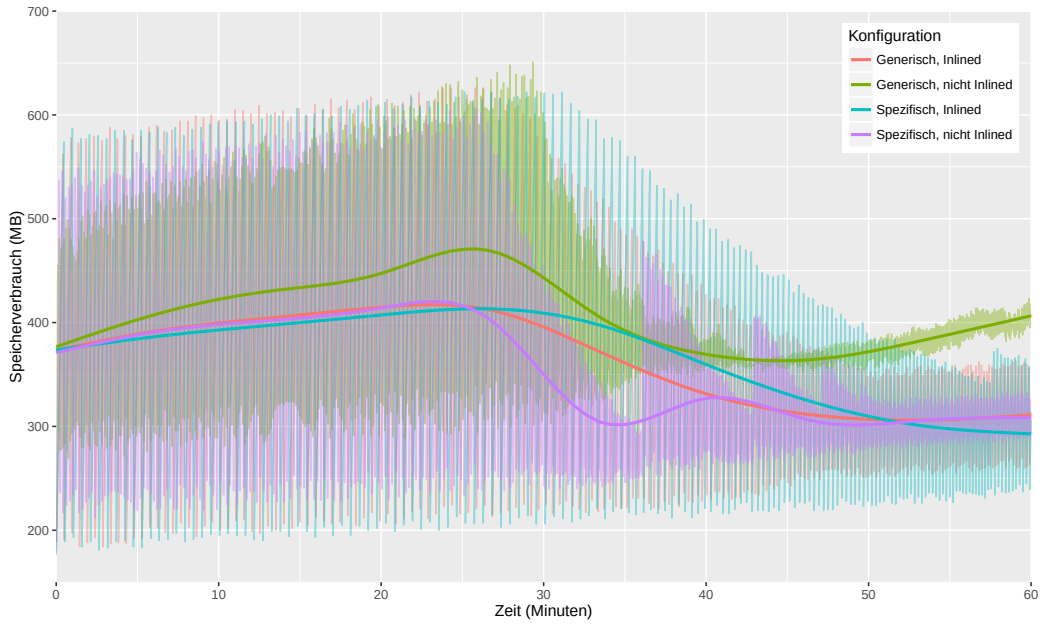


Abbildung 5.5.: Gemessener Speicherverbrauch des Laufzeitsystems für den Storage-Management-Anwendungsfall. Messungen überlagert mit mit Generalized Additive Models (GAM) geglätteten Kurven.

Tripel in die lokale Wissensbasis der Regel-Engine importiert. Anschließend wurde der BVQ-Monitoring-Adapter gestartet, der kontinuierlich VDisk-Metriken erzeugt.

Abbildung 5.5 zeigt den gemessenen Speicherverbrauch aller Konfigurationskombinationen des Anwendungsfalls über eine Messdauer von einer Stunde. Da zur Laufzeit kontinuierlich sehr viele Objekte (Events, RDF-Tripel, Regel-Engine-interne Objekte) erzeugt werden, die zyklisch vom Java-Garbage-Collector eingesammelt werden, unterliegt der Speicherverbrauch starken Schwankungen. Daher wurden zur besseren Visualisierung die realen Speicherverbräuche mit geglätteten Kurven überlagert, die mit Generalized Additive Models (GAM) erzeugt wurden. Der durchschnittliche Speicherverbrauch aller Konfigurationen liegt bei ca. 400 MB. Im Durchschnitt weist die Konfiguration mit spezifischen Statements ohne Inlining mit 355 MB den niedrigsten Speicherverbrauch auf. Ihr folgen der generische Ansatz mit Inlining mit 365 MB und der spezifische Ansatz mit Inlining mit 369 MB. Den höchsten Speicherverbrauch hat der generische Ansatz ohne Inlining, der im Durchschnitt 408 MB benötigt. Tendenziell zeigt der spezifische Ansatz mit Inlining zum Ende der Messung hin den besten Speicherverbrauch. Bei allen Konfigurationen sind nach ca. 30 Minuten deutlich die automatischen Optimierungen der JVM zu erkennen, die dazu führen, dass bestimmte Objekttypen länger in der Young Generation des Heaps gehalten und so bereits bei einer Minor Garbage Collection eingesammelt werden.

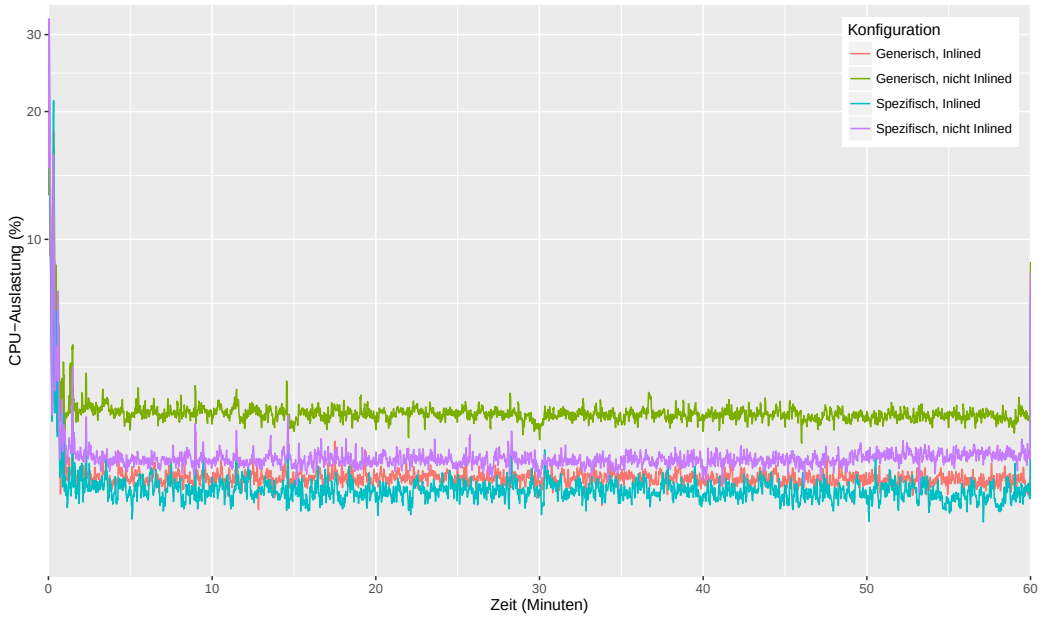


Abbildung 5.6.: Gemessene CPU-Auslastung des Laufzeitsystems für den Storage-Management-Anwendungsfall (Y-Achse log1p logarithmiert).

Abbildung 5.5 zeigt analog zum Speicherverbrauch die gemessene CPU-Auslastung aller Konfigurationskombinationen. Nachdem sich das Laufzeitsystem nach ca. einer Minute eingeschwungen hat, erzeugen alle Konfigurationen bei der kontinuierlichen Verarbeitung eine geringe CPU-Auslastung. Die niedrigste durchschnittliche Auslastung hat der spezifische Ansatz mit aktiviertem Inlining mit 2,17%. Ihm folgt der generische Ansatz mit Inlining mit 2,37% und der spezifische Ansatz ohne Inlining mit 2,70%. Die höchste Auslastung erzeugt der generische Ansatz ohne Inlining mit durchschnittlich 3,62%. Während der simulierten Überlastsituationen zeigten sich keine auffälligen Auslastungspitzen.

Neben dem Ressourcenverbrauch wurden für den Storage-Management-Anwendungsfall die Antwort- und Reaktionszeiten des Laufzeitsystems gemessen. Die Antwortzeit wurde ermittelt, indem die Zeitdifferenz zwischen dem Absetzen einer VDisk-Metrik im BVQ-Monitor-Adapter und dem Eintreffen der entsprechenden Wissensbasisaktualisierung im BVQ-Management-Adapter gemessen wurde. Die Reaktionszeit wurde ermittelt, indem die Zeitdifferenz zwischen dem Absetzen der zur Überlast führenden VDisk-Metrik und dem Eintreffen der ersten Governing-Aktion im Execute-Adapter gemessen wurde.

Abbildung 5.7 zeigt die ermittelten Antwortzeiten der unterschiedlichen Konfigurationskombinationen. Das beste Antwortzeitverhalten zeigt der spezifische Ansatz mit aktiviertem Inlining, bei dem das System im Durchschnitt nach 3,70 ms antwortete.

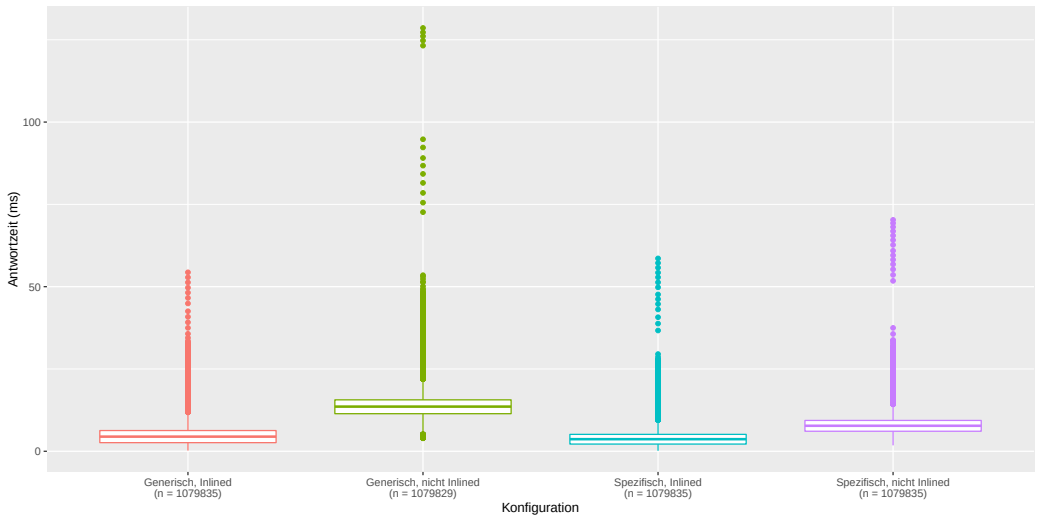


Abbildung 5.7.: Gemessene Antwortzeit des Laufzeitsystems für den Storage-Management-Anwendungsfall.

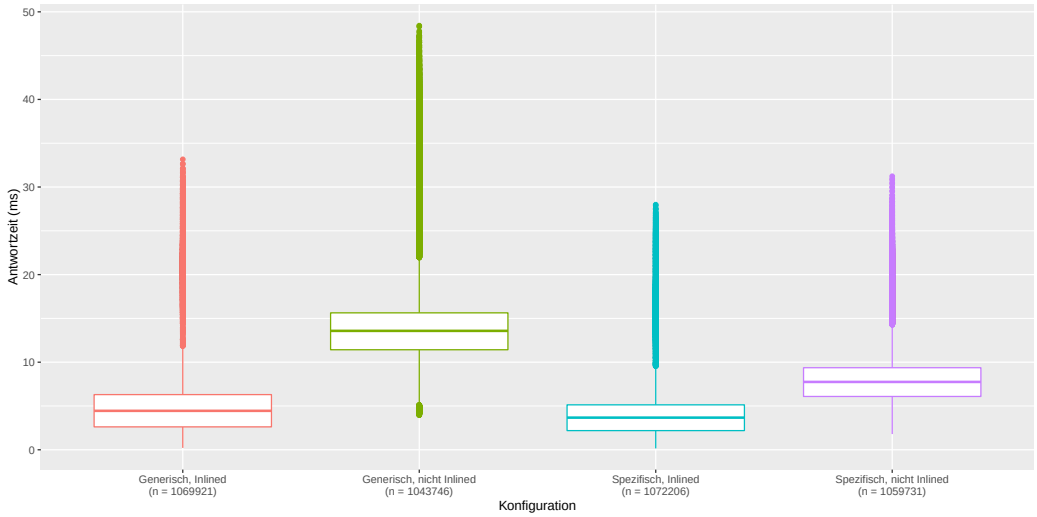


Abbildung 5.8.: Gemessene Antwortzeit des Laufzeitsystems für den Storage-Management-Anwendungsfall mit herausgefilterten Messungen, die während einer Garbage-Collection stattfanden.

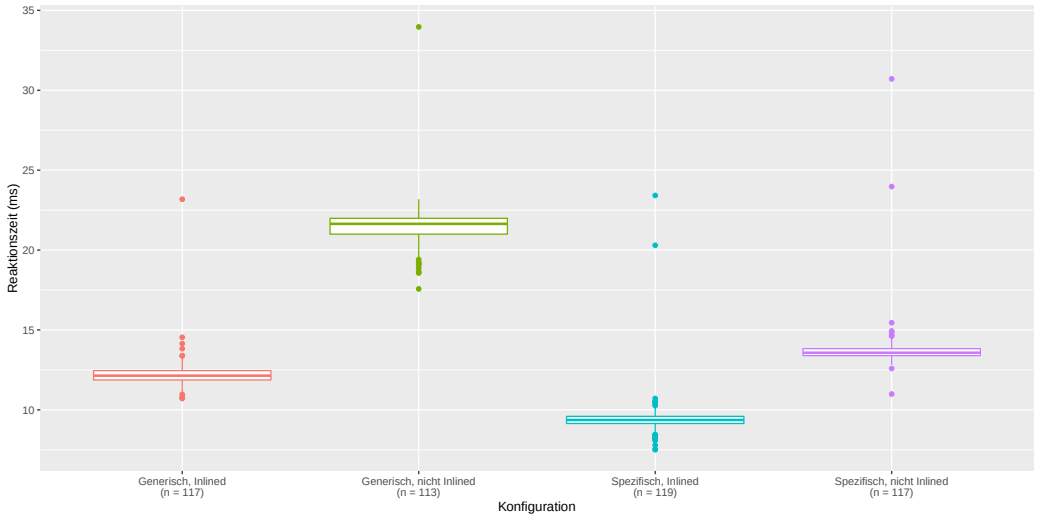


Abbildung 5.9.: Gemessene Reaktionszeit des Laufzeitsystems für den Storage-Management-Anwendungsfall, mit herausgefilterten Messungen, die während einer Garbage-Collection stattfanden.

Die durchschnittliche Antwortzeit des generischen Ansatzes mit Inlining lag bei 4,50 ms, beim spezifischen Ansatz ohne Inlining bei 7,63 ms und beim generischen Ansatz ohne Inlining bei 13,82 ms. Pro Sekunde wurden dabei ca. 1050 Regeln ausgelöst, was sieben Aktivierungen pro verarbeiteter Metrik entspricht. Die Ausreißer lassen sich durch eine parallel laufende Garbage Collection erklären, die die Verarbeitung für einige Millisekunden blockiert. Abbildung 5.8 zeigt die bereinigten Antwortzeiten, bei denen alle Messungen herausgefiltert wurden, die für eine Garbage Collection zum Zeitpunkt t mit der Dauer d im Intervall $[t-d; t+3d]$ lagen. Dadurch sollen die Effekte, die von der Speicherknappheit vor der Garbage Collection und der angestauten Arbeit während der Garbage Collection ausgelöst wurden, reduziert werden.

Abbildung 5.9 zeigt die ermittelten Reaktionszeiten der unterschiedlichen Konfigurationskombinationen. Die beste Reaktionszeit hat der spezifische Ansatz mit aktivem Inlining, der ein Governing im Mittel nach 9,52 ms kommandiert. Ihm folgen die generische Konfiguration mit Inlining mit 12,32 ms und die spezifische Konfiguration ohne Inlining mit 14,56 ms. Mit 21,51 ms weist der generische Ansatz ohne Inlining eine deutlich höhere Reaktionszeit auf. Wie bei der Antwortzeit wurden auch hier die während einer Garbage Collection gemessenen Werte in der Darstellung herausgefiltert.

Damit zeigt sich, dass das auf Basis der entwickelten Konzepte umgesetzte Laufzeitsystem in der Lage ist, ein automatisiertes Management mittelgroßer Storage-Systeme durchzuführen. Die dafür benötigten Ressourcen können von einem her-

kömmlichen Desktop-PC bereitgestellt werden. Die gemessenen Antwort- und Reaktionszeiten zeigen für alle Konfigurationen eine Management-Reaktion im niedrigen, zweistelligen Millisekundenbereich.

Für die unterschiedlichen Konfigurationskombinationen zeigt sich, dass der spezifische Ansatz mit aktiviertem Inlining, sowohl was den Ressourcenverbrauch als auch die Antwort- und Reaktionszeiten angeht, am besten für diese Art von Anwendungsfall geeignet ist. Dabei haben sowohl die Statement-Repräsentation als auch das Inlining signifikanten Einfluss auf beide Messgrößen. Die spezifischen Statements wirken sich besonders positiv auf Regeln aus, die verstärkt auf das ontologische Informationsmodell zugreifen, das Inlining auf Regeln, die verstärkt auf den Event-Payload zugreifen.

5.2 Radar-Management-Fallstudie

Als Air Traffic Control (ATC)-Systeme bezeichnet man im Flugsicherungskontext[6] komplexe IT-Systeme, die einen Datenverarbeitungsprozess realisieren, bei dem Informationen unterschiedlichster Datenquellen miteinander verknüpft und daraus Boden- und Luftlagedarstellungen abgeleitet werden. Die gewonnenen und visuell aufbereiteten Daten werden Fluglotsen in Kontrollzentren (Center) und Kontrolltürmen (Tower) bereitgestellt, die auf deren Basis Überflüge, Vorfeldbewegungen, Starts und Landeanflüge koordinieren.

Bei der Lokalisierung und Identifizierung von Objekten im Luftraum sind Radare nach wie vor die primäre Datenquelle. Sie lassen sich prinzipiell in Primär- und Sekundärradare unterscheiden. Ein Primärradar (Primary Surveillance Radar, PSR) ist ein klassisches Pulsradar, das elektromagnetische Wellen im Radiofrequenzbereich aussendet, die von getroffenen Objekten als Echo zurückgeworfen werden. Das Radar fängt das reflektierte Signal auf und nähert über die Signallaufzeit, den Winkel und die Radialgeschwindigkeit die relative Position und Geschwindigkeit des Objekts an. Im Gegensatz zu Primärradaren sind Sekundärradare (Secondary Surveillance Radar, SSR) auf die Kooperation des Ziels angewiesen. Sie senden ein Datensignal ab, das von einem Transponder des Luftfahrzeugs empfangen und mit einem Antwortsignal erwidert wird. Dadurch erhöht sich zum einen die Signalreichweite, zum anderen können Zusatzinformationen mitgesandt werden. Der verbindliche Mark-X-Standard legt fest, dass der Transponder eines zivilen Ziels mit einer vom Fluglotsen beim Luftraumeintritt zugeteilten Identifikationsnummer und

[6] https://www.dfs.de/dfs_homepage/de/Flugsicherung/Glossar%20Flugsicherung/

der barometrischen Flughöhe antworten muss (Mode-A). Da die für die Identifikation vorgesehenen 4-Byte durch das steigende Verkehrsaufkommen nicht mehr ausreichen, wurde die Antwort im Mark-XII-Standard um eine individuelle, fest einprogrammierte Transponderadresse erweitert (Mode-S).

Um Luftfahrzeuge verlässlich lokalisieren und identifizieren zu können, werden beim sogenannten Tracking die Messungen (Plots) mehrerer Radare fusioniert und mit zusätzlichen Daten wie Flugplänen verknüpft. In Flughafennähe kommt zudem verstärkt Multilateration (MLAT) zum Einsatz, bei der das vom Transponder ausgestrahlte Sekundärsignal von mehreren passiven, zeitsynchronisierten Empfängern erfasst und über deren Bodenposition und die Signalempfangszeit die Zielposition trianguliert wird. Neuere Luftfahrzeuge haben zudem einen GPS-Empfänger an Board, mit dem sie selbst ihre Position bestimmen und als Automatic Dependent Surveillance Broadcast (ADS-B) verbreiten. Dadurch können Ziele auch in nicht von Radaren abgedeckten Bereichen (z. B. dem offenen Ozean) verfolgt werden; die Vertrauenswürdigkeit der Daten ist auf Grund der Manipulationsmöglichkeiten jedoch gering.

5.2.1 Anwendungsfall

Um die vom Gesetzgeber geforderte Hochverfügbarkeit gewährleisten zu können, betreibt die DFS Deutsche Flugsicherung GmbH (DFS)[7] mehrere Hard- und Softwareredundante ATC-Systeme. Für den betrachteten Anwendungsfall sind das hauseigene PHOENIX [65] und das extern eingekaufte Very Advanced Air Traffic Control Automation System (VAFORIT) relevant. Beide Systeme nutzen bei der Radardatenverarbeitung ein kartesisches Koordinatensystem, dessen Ursprung durch eine geographische Koordinate auf Nullniveau definiert ist. Alle Radarmessungen werden mittels stereografischer Projektion auf das Koordinatensystem abgebildet. PHOENIX wendet dann bei einer echten Multiradardatenverarbeitung auf alle Messungen einen Kalman-Filter [74] an, um die Positionen und Charakteristiken der Ziele zu bestimmen. VAFORIT hingegen unterteilt den Luftraum in Observationsbereiche, die isoliert betrachtet werden. Dabei wird auf der XY-Ebene des Koordinatensystems ein an den Achsen ausgerichtetes Gitter (Tile Set) aufgespannt, das aus einer festen Anzahl quadratischer Gitterzellen (Tiles) besteht. Der Gitterursprung wird relativ zum Ebenenursprung in Seemeilen (M) definiert. Der Raum über jeder Gitterzelle wird anhand von Höhenbändern in Quader unterteilt, die die Observationsbereiche bilden. Das oberste Höhenband (oberer Luftraum) ist nach oben unbegrenzt. Jedem

[7] `https://www.dfs.de/`

Observationsbereich wird eine begrenzte Anzahl Radare zugeteilt, deren Messungen beim darin durchgeführten Tracking berücksichtigt werden.

Abbildung 5.10 zeigt eine typische Kachelkonfiguration für Deutschland. Der Ursprung des kartesischen Koordinatensystems liegt bei $51°\mathrm{N}\ 10°\mathrm{E}$, der Gitterursprung $(-512\,\mathrm{M}, -512\,\mathrm{M})$ relativ dazu. Es werden 64×64 Zellen mit einer Ausdehnung von jeweils $16\,\mathrm{M}$ aufgespannt. Fünf Höhenbänder á $5000\,\mathrm{ft}$ unterteilen den Luftraum vertikal in Observationsbereiche, denen jeweils bis zu 6 Radare zugeteilt sind.

Um ein Luftfahrzeug in einem Observationsbereich zuverlässig lokalisieren und charakterisieren zu können, ist eine Mindestabdeckung von Radaren bestimmter Eigenschaften nötig. Ein Service Level Agreement legt deshalb fest, dass ein Bereich von mindestens einem Primärradar, zwei Sekundärradaren und zwei Sekundärradaren mit Mode-S-Unterstützung überwacht werden muss. Radare können durchaus mehrere dieser Eigenschaften aufweisen und entsprechend zu mehreren SLIs beitragen. Eine Tile-Set-Konfiguration wird grundsätzlich so ausgelegt, dass für jeden relevanten Überwachungsbereich die SLA-Konformität gegeben ist. Ein Überwachungsbereich ist irrelevant, wenn ihm kein Radar zugeteilt ist. Kommt es während des Betriebs zu vollständigen oder partiellen Radarausfällen, kann es zu SLA-Brüchen kommen. Um diese zu erkennen und aufzuzeigen, existiert in PHOENIX eine Komponente, die die SLA-Konformität der VAFORIT-Konfiguration überwacht. Der typischerweise einmal pro Tag manuell angestoßene Prozess nutzt als Eingabe die Tile-Set-Konfiguration und eine von PHOENIX bereitgestellte Liste der während des Zeitraums durchlaufenen Radarstatus und generiert daraus einen SLA-Report, der Verstöße aufführt. Bei persistenten Ausfällen wird anhand aufgezeichneter Radarmessungen (typischerweise der letzten 24 Stunden) eine Coverage-Detection durchgeführt und die Tile-Set-Konfiguration angepasst; transiente Ausfälle bleiben weitestgehend unbeachtet.

Im gemeinsamen Forschungsprojekt *Wissensbasiertes IT-Management in Surveillance- und Flugsicherungssystemen* (WiBaITM)[8], in dessen Kontext ein Großteil der in dieser Arbeit entwickelten Konzepte entstanden ist, sollte untersucht werden, inwiefern sich das ontologiebasierte Framework für ein Online-Radar-Management eignet. Die zu entwickelnde Lösung sollte dabei die Rolle eines Assistenzsystems einnehmen, das eine stetige SLA-Überwachung und Coverage-Detection durchführt, bei Verstößen autonom eine Rekonfigurationsagenda ableitet und einem Administrator zur Anwendung vorschlägt.

[8] `https://www.hs-rm.de/de/fachbereiche/design-informatik-medien/forschungsprofil/dfs/`

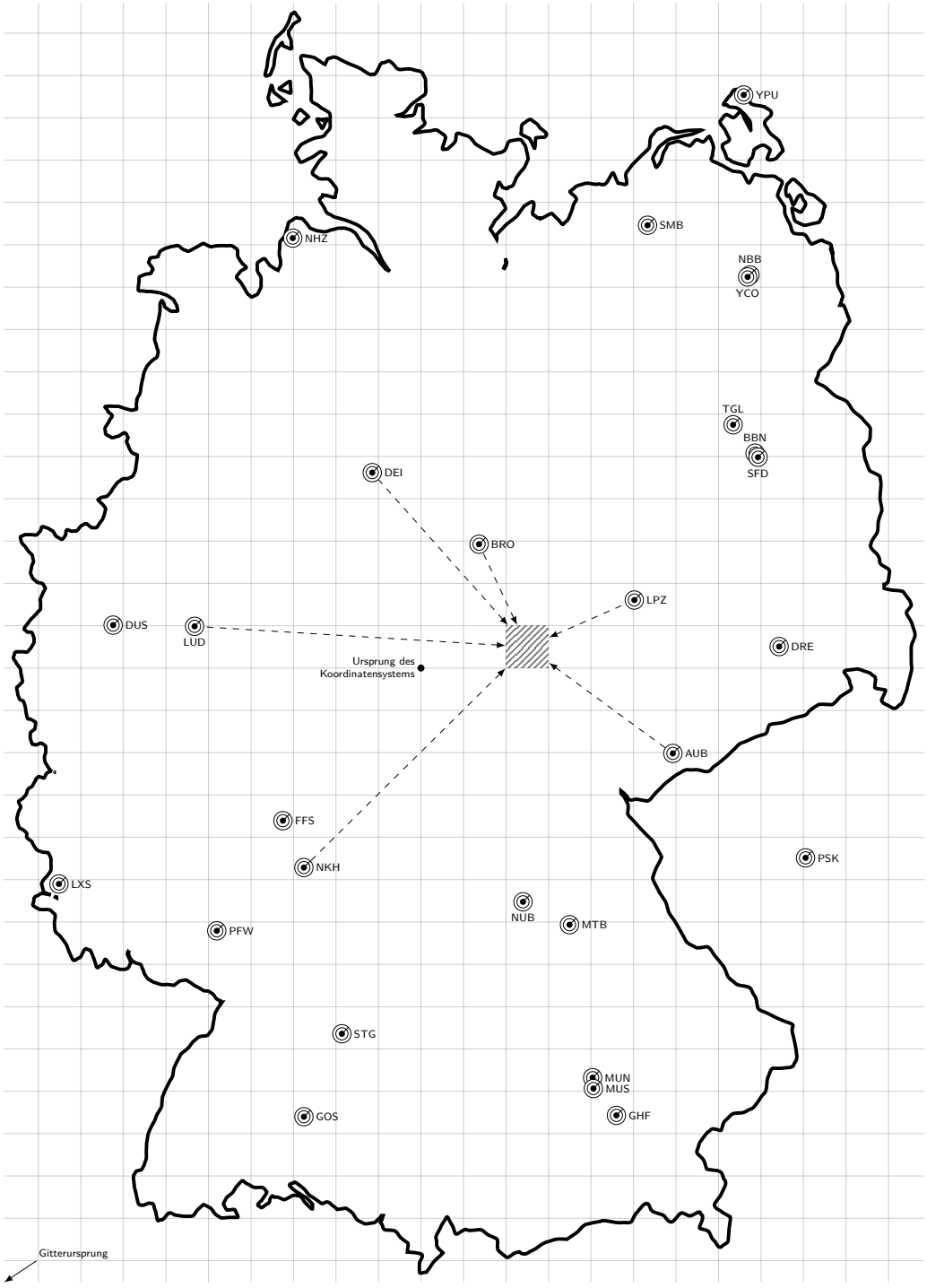


Abbildung 5.10.: Beispiel einer Tile-Set-Konfiguration mit Radarzuordnung eines Tiles für Deutschland.

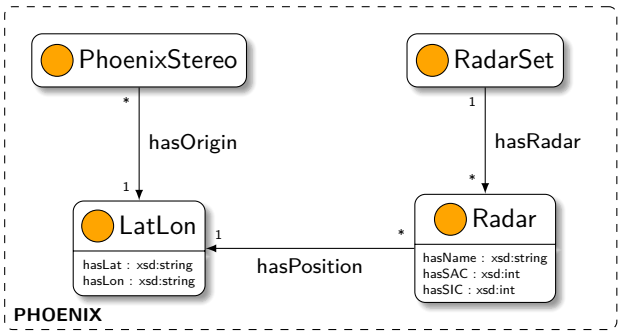


Abbildung 5.11.: Für den Anwendungsfall relevanter Teilausschnitt des PHOENIX-Konfigurationsmodells.

5.2.2 Umsetzung

Zur Umsetzung des Anwendungsfalls wurden zunächst gemäß des Framework-Vorgehensmodells im Modellierungsschritt der Adaptionsphase (siehe Abschnitt 4.3.1, Seite 108) Ontologien der betrachteten Domänen entwickelt. In PHOENIX existiert ein Konfigurationsschema, mit dem eine PHOENIX-Installation beschrieben wird. Das XML-basierte Modell (kein XML-Schema) definiert Knoten (Nodes) und Felder (Items). Ein Knoten repräsentiert einen benannten Typ, der von anderen Typen erben kann und sich aus mehreren Feldern zusammensetzt. Ein Feld hat einen Namen, einen Typ (primitiver Datentyp oder referenzierter Knoten) und legt Metadaten wie Wertebereich und Standardwert fest. Um die im Konfigurationsschema definierten Konzepte nutzbar zu machen, wurde ein Modelltransformator entwickelt, der das Modell in eine OWL-Ontologie überführt. Der Transformator bildet jeden Knoten auf eine OWL-Klasse und jedes Feld auf eine OWL-Rolle ab. Dabei werden etwaige Hierarchien und Einschränkungen berücksichtigt und auf Klassen- und Rolleneinschränkungen abgebildet. Beim Transformationsprozess können relevante Modellausschnitte ausgewählt und deren transitive Hülle abgebildet werden.

Beim betrachteten Anwendungsfall wurde der Transformationssauschnitt auf die Radarkonfiguration eingeschränkt. Abbildung 5.11 zeigt die relevanten Entitäten der PHOENIX-Ontologie. Die PHOENIX-Stereo-Projektion (`PhoenixStereo`) hat einen Ursprung in geographischen Koordinaten (`LatLon`). Eine Radarkonfiguration (`RadarSet`) besteht aus mehreren Radaren (`Radar`), die jeweils einen Namen (`hasName`), einen System Area Code (`hasSAC`), einen System Identification Code (`hasSIC`) und eine Position (`hasPosition`) haben.

Da für die Tile-Set-Konfiguration von VAFORIT kein transformierbares Modell vorlag, wurde die in Abbildung 5.12 dargestellte Tile-Set-Ontologie manuell entwickelt. Ein Tile-Set (`TileSet`) hat einen Ursprung in geographischen Koordinaten (`LatLon`),

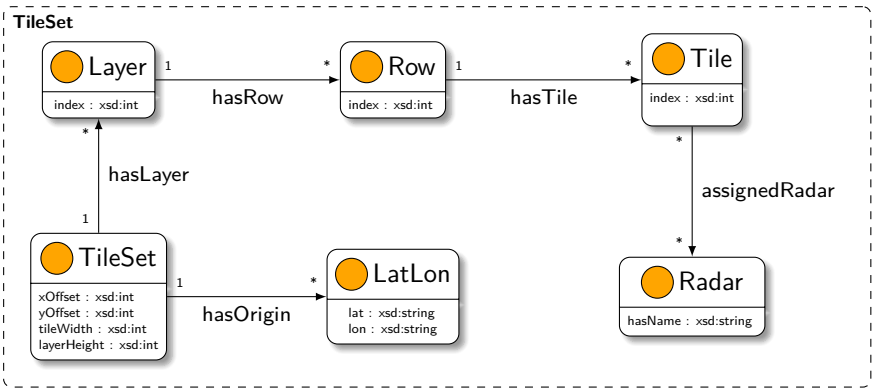


Abbildung 5.12.: Für den Anwendungsfall entwickelte Tile-Set-Ontologie.

einen relativen Gitterursprung in nautischen Meilen (X- und Y-Offset), eine Kachelausdehnung in nautischen Meilen und eine Höhenbandhöhe in Fuß. Jedes Höhenband (`Layer`) ist in Zeilen (`Rows`) und jede Zeile in Observationsbereiche (`Tiles`) unterteilt, denen Radare (`Radar`) zugewiesen sind. Höhenbänder, Zeilen und Kacheln werden über ihren Index, Radare über ihren Namen identifiziert.

Zur Verknüpfung der beiden Domänen und zur Modellierung der von ihnen nicht erfassten Aspekte wurde anschließend die in Abbildung 5.13 dargestellte Online Radar Service Level Management (ORSLM)-Ontologie entwickelt. Die Klasse `SensorStatus` repräsentiert die unterschiedlichen Status, die eine Radaranlage annehmen kann. Der Radarstatus kann aktiv (`UP`), degradiert (`DEGRADED`) oder inaktiv (`DOWN`) sein. Die Klasse `SubSensorStatus` repräsentiert die unterschiedlichen Status, die ein Radarsensor annehmen kann. Der Subsensorstatus kann aktiv (`ACTIVE`) oder inaktiv (`NOT_PRESENT`) sein. Einem PHOENIX-Radar kann über die temporale Rolle `sensorStatus` ein Radarstatus und über die temporalen Rollen `psrStatus`, `ssrStatus` und `modeSStatus` ein Subsensorstatus für den entsprechenden Sensor zugewiesen werden. Die temporalen Klassen `PSRViolator`, `SSRViolator` und `ModeSViolator` dienen dazu, Observationsbereiche als SLA-brüchig zu markieren.

Die Struktur der zur Laufzeit zwischen PHOENIX und dem Laufzeitsystem ausgetauschten oder in der Management-Logik verwendeten Events und Aktionen wurden in der in Abbildung 5.14 dargestellten Messages-Ontologie modelliert. Eine Radarmessung (`PlotMsg`) setzt sich aus den Radaridentifizierungsmerkmalen (SAC und SIC) und der zum Ursprung des Koordinatensystems relativen Raumposition (Höhe, X- und Y-Koordinate) zusammen. Eine abgebildete Messung (`MappedPlot`) repräsentiert eine Radarmessung, die anhand ihrer Merkmale mit dem entsprechenden Radar- und Überwachungsbereichsindividuum verknüpft wurde. Eine Radarstatusnachricht (`RadarStatusMsg`) setzt sich aus Radaridentifizierungsmerkmalen, einem

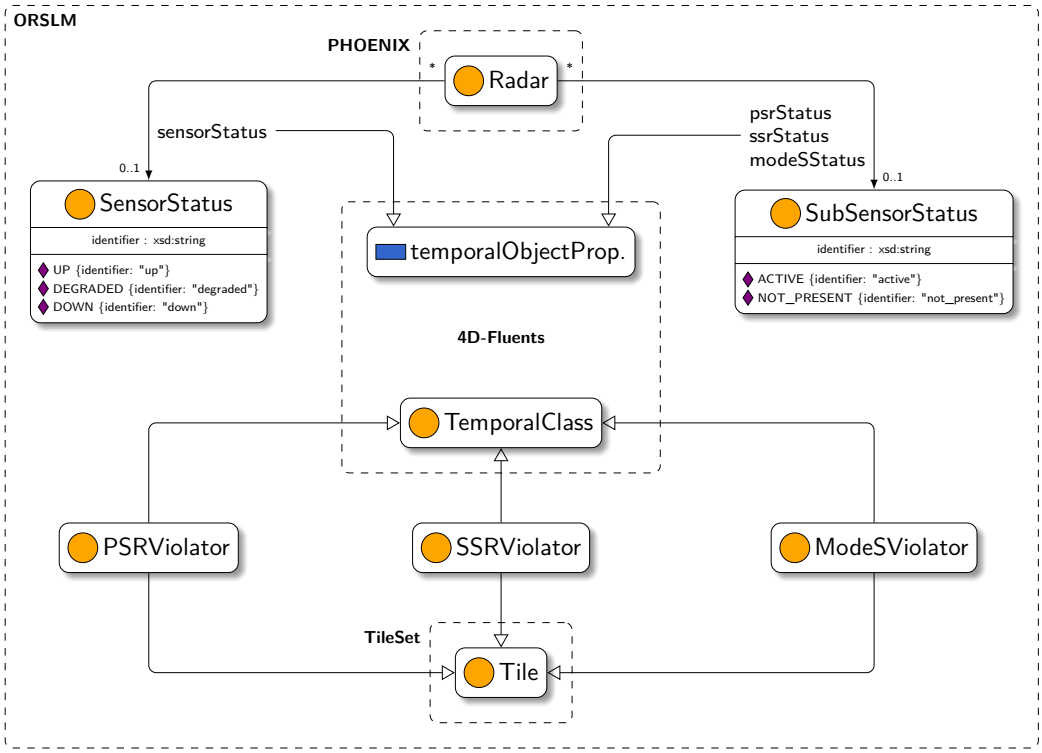


Abbildung 5.13.: Online Radar Service Level Management (ORSLM)-Ontologie.

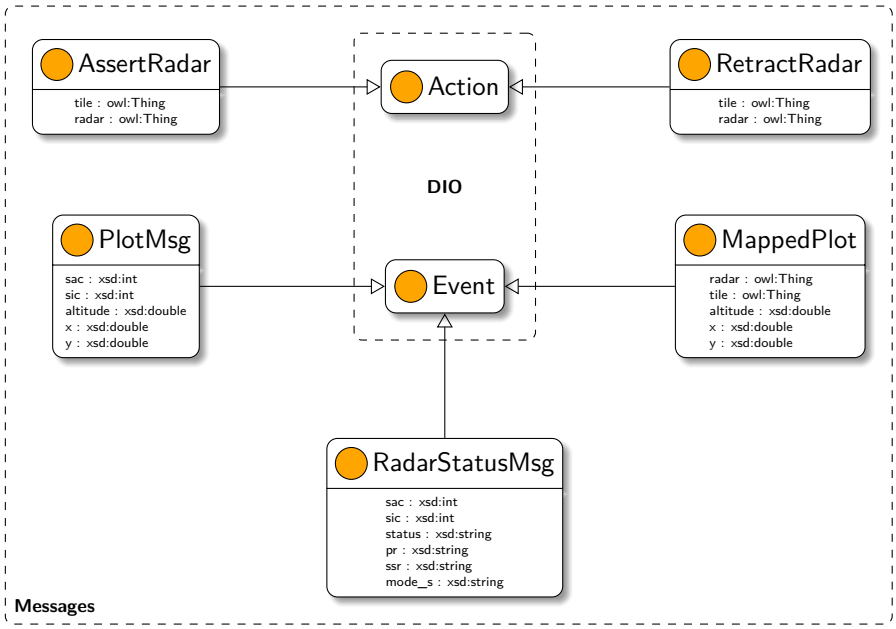


Abbildung 5.14.: Die Messages-Ontologie definiert die Schnittstellentypen des Anwendungsfalls.

Ontologie	Klassen	abstrakte Rollen	konkrete Rollen	Individuen	Axiome
PHOENIX	28	26	115	19	615
TileSet	7	8	8	1	61
ORSLM	6	4	1	5	49
Messages	6	2	10	0	51
Gesamt	46	38	134	25	776

Tabelle 5.4.: Umfang der entwickelten Domänen- und Schnittstellenontologien des Radar-Management-Anwendungsfalls.

allgemeinen Radarstatus und den Status der einzelnen Sensoren (PSR, SSR und Mode-S) zusammen. Eine Radarzuweisung (`AssertRadar`) ist eine Management-Aktion, die einem Überwachungsbereich ein neues Radar zuweist, eine Radarentfernung (`RetractRadar`) entfernt ein einem Überwachungsbereich zugeteiltes Radar. Tabelle 5.4 zeigt den Umfang der entwickelten Domänen- und Schnittstellenontologien.

Auf Basis der entwickelten Ontologien wurde mit der OntML-DSL in der zugehörigen IDE (siehe Abschnitt 4.4, Seite 130) die Management-Logik des Anwendungsfalls definiert. Sie ist in die fünf Management-Module Queries, Plot-Mapping, Radar-Status-Mapping, SLA und Reconfiguration unterteilt.

Das Queries-Modul (siehe Abschnitt A.4.1, Seite 240) definiert abstrakte Ziele und Abfragen, die von den anderen Modulen bei der Umsetzung der eigentlichen Management-Logik verwendet werden. Das abstrakte Ziel *ActiveRadar* und die dazugehörige Abfrage „Active Radar“ suchen alle Radare, deren Status aktiv (*UP*) ist. Die abstrakten Ziele *ModeSRadar*, *PSRRadar* und *SSRRadar* mit den gleichnamigen Abfragen suchen alle aktiven Radare, bei denen der entsprechende Subsensorstatus ebenfalls aktiv (*ACTIVE*) ist. Das abstrakte Ziel *ConfiguredTile* und die zugehörige Abfrage „Configured Tile“ suchen die Observationsbereiche, deren zugewiesene Radare alle einen Status haben.

Das Plot-Mapping-Modul (siehe Abschnitt A.4.2, Seite 241) enthält die Event-Korrelationsregel „Create Mapped Plot“. Sie ist dafür zuständig, jede vom überwachten System empfangene Radarmessung in eine abgebildete Messung zu überführen. Dazu bindet sie in der Wissensbasis die Kachelbreite, die Höhenbandhöhe, den Kachelversatz und das oberste Höhenband. Für jede gefundene Radarmessung sucht sie dann anhand der Kachelattribute und der Koordinate das passende Observationsbereichsindividuum und anhand der Radaridentifizierungsmerkmale das passende Radarindividuum, erzeugt dafür eine abgebildete Messung und entfernt

die Ursprungsmessung. Alle Plots, für die kein Observationsbereich oder kein Radar gefunden wird, werden verworfen. Die Event-Korrelationsregel „Filter Mapped Plots“ kondensiert die Event-Menge, indem sie redundante Messungen entfernt.

Das Radar-Status-Mapping-Modul (siehe Abschnitt A.4.3, Seite 243) enthält die Event-Mapping-Regel „Map Radar Status Messages“. Sie ist dafür zuständig, die in Radarstatusnachrichten beschriebenen Statusänderungen auf die Radarindividuen abzubilden. Für jede Nachricht sucht sie anhand der Radaridentifizierungsmerkmale das Radarindividuum und anhand der Statuszeichenketten die Statusindividuen. Anschließend entfernt sie die alten Statusindividuen, weist die neuen Statusindividuen zu und entfernt die Nachricht.

Das SLA-Modul (siehe Abschnitt A.4.4, Seite 245) definiert für jeden Sensortyp (PSR, SSR, Mode-S) eine funktionale Abhängigkeit, die für jeden Observationsbereich zählt, wie viele der ihm zugewiesenen Radare den entsprechenden Sensortyp unterstützen. Liegt der Wert unterhalb der SLA-Schranke, wird dem Observationsbereich die zum Sensortyp passende `Violator`-Klasse zugewiesen. Zusätzlich existiert für jeden Sensortyp eine Bewertungsfunktion, die für jeden SLA-Violator die fehlende Radaranzahl als Hard-Score impliziert.

Das Reconfiguration-Modul (siehe Abschnitt A.4.5, Seite 247) definiert das abstrakte Ziel `candidate` mit der zugehörigen Abfrage „Radar Candidate“. Sie leiten ein Radar als Observationsbereichskandidat ab, wenn es innerhalb der letzten 24 Stunden Messungen dafür erzeugt hat und nicht bereits zugewiesen ist. Für jeden Sensortyp existieren zwei Aktionsvorschlagsregeln. Die „Suggest Radar Assertion“-Regeln schlagen die Zuweisung eines Radars zu einem Observationsbereich vor, wenn der Observationsbereich gegen ein SLA verstößt und weniger als sechs Radare zugewiesen hat, und das Radar ein Kandidat für den Bereich ist und den fehlenden Sensor aufweist. Die „Suggest Radar Retraction“-Regeln schlagen vor, ein Radar von einem Observationsbereich zu entfernen, wenn der Observationsbereich gegen ein SLA verstößt und bereits sechs Radare zugewiesen hat, und der benötigte Sensor beim Radar nicht aktiv ist. Tabelle 5.5 zeigt den Umfang der entwickelten Management-Module.

Im Anbindungsschritt der Adaptionsphase wurden zunächst Discovery-Adapter entwickelt, die die Systemkonfiguration initial in die Wissensbasis importieren. Der PHOENIX-Discovery-Adapter liest eine PHOENIX-Konfiguration ein und bildet sie auf Instanzen und Beziehungen der aus dem Schema abgeleiteten OWL-Klassen und Rollen ab. Bei der Konfiguration handelt es sich um eine hierarchische Struktur aus Name-Wert-Paaren, die der Adapter gemäß der auf das Konfigurationsschema

OntML-DSL				RDF-Serialisierung		
Modul	abstrakte Ziele	Regeln	SLOC	Individuen	Klassen-zu-weisungen	Rollen-zu-weisungen
Queries	5	5	45	65	170	110
Plot-Mapping	0	2	69	190	499	311
Radar-Status-Mapping	0	1	46	97	242	148
SLA	0	6	70	96	293	179
Reconfiguration	1	9	119	284	856	516
Gesamt	6	23	346	732	2060	1264

Tabelle 5.5.: Umfang der entwickelten Management-Module des Radar-Management-Anwendungsfalls.

angewandten Konvertierungsvorschriften übersetzt und in die Wissensbasis einfügt. Mit einem Filterausdruck lassen sich die zu importierenden Daten einschränken.

Der Tile-Set-Discovery-Adapter importiert eine Tile-Set-Konfiguration in die Wissensbasis. Die Konfigurationsdatei besteht aus einem Kopf, der die Radarnamen und Gittereigenschaften (Ursprung, Versatz, Kachelbreite, etc.) definiert und einem Körper, in dem für jedes Indextripel (Höhenband, X, Y) Radare festgelegt sind. Der Adapter erzeugt daraus Individuen und Beziehungen, die der modellierten Tile-Set-Ontologie entsprechen, und fügt sie in die Wissensbasis ein. Dabei ignoriert er Observationsbereiche, denen keine Radare zugeteilt sind. In der Adapterkonfiguration können die zu importierenden Observationsbereiche eingeschränkt werden (z. B. nur oberer Luftraum).

Zur Kommunikation zwischen PHOENIX und dem Laufzeitsystem wurden auf beiden Seiten Adapter entwickelt. Auf PHOENIX-Seite entstand der *d-man*-Prozess, der als Teil des PHOENIX-Systems die dort eingesetzte Nachrichten-Middleware und Broadcasting-Infrastruktur nutzt, um Radarmessungen und Radarstatus zu beziehen. Im Hinblick auf zukünftige Erweiterungen bindet er zudem Schnittstellen ein, über die Komponenteninformationen und Sensorstatistiken abgefragt werden können. Nach außen bietet der *d-man*-Prozess eine TCP-Verbindung an, über die Laufzeitsystemadapter Überwachungsdaten beziehen oder Management-Informationen bereitstellen können. Die grafische Benutzeroberfläche des Prozesses stellt die aktuelle SLA-Situation dar und ermöglicht es, den Optimierungsprozess anzustoßen.

Auf Seite des Laufzeitsystems wurden Monitor- und Management-Adapter entwickelt. Der PHOENIX-Monitor-Adapter bezieht vom *d-man* Radarmessungen und Status-

Komponente	Java SLOC	C++ SLOC
TileSet-Discovery-Adapter	334	0
PHOENIX-Schematransformator	195	0
PHOENIX-Discovery-Adapter	393	0
PHOENIX-Monitor-Adapter	473	0
PHOENIX-Management-Adapter	331	0
ORSLM-Management-Adapter	672	0
d-man	0	2214
Gesamt	2398	2214

Tabelle 5.6.: Umfang der entwickelten Anbindungskomponenten des Radar-Management-Anwendungsfalls.

nachrichten, bildet sie auf die in der Messages-Ontologie definierten Konzepte ab und fügt sie in die Wissensbasis ein. Der PHOENIX-Management-Adapter stößt eine vom *d-man* kommandierte Optimierung an und antwortet mit der erarbeiteten Rekonfigurationsagenda. Zudem lauscht er auf Wissensbasisänderungen und teilt dem *d-man* SLA-Verstöße mit.

Der Kommunikationskanal wird von einer Verbindungskomponente abstrahiert, die je nach Konfiguration eine reale TCP-Verbindung zum *d-man*-Prozess aufbaut oder eine Aufzeichnung abspielt. Um im Replay-Modus die vom Laufzeitsystem geschlossenen Management-Informationen visualisieren und aktiv beeinflussen zu können, existiert der ORSLM-Adapter. Er bietet eine Kommandierungsoberfläche, über die Radar- und Sensorausfälle simuliert werden können, indem entsprechende Statusnachrichten erzeugt und ins Laufzeitsystem eingespeist werden. Tabelle 5.6 zeigt den Umfang der entwickelten Anbindungskomponenten.

Für die entwickelten Modelle und Adapter wurde ein Build-Prozess definiert, der die Ontologien mit dem *Module-Generator* automatisch in Ontologiemodule transformiert und für alle anwendungsfallspezifischen Komponenten OSGi-Bundles erzeugt. Zudem wurden zwei Framework-Konfigurationen erstellt, die die zu ladenden Ontologiemodule und Adapterkonfigurationen beschreiben. Beide Konfigurationen führen ein Management des oberen Luftraums durch; eine Konfiguration kommuniziert mit dem *d-man*-Prozess einer realen PHOENIX-Installation; eine Konfiguration liest aufgezeichnete Daten aus einer Log-Datei. Es wurde eine Laufzeitumgebung erzeugt, die neben den in Abschnitt 4.4 (siehe Seite 130) entwickelten generischen Laufzeitkomponenten die anwendungsfallspezifischen Ontologiemodule und Adapter umfasst.

5.2.3 Laufzeitverhalten

Um das Laufzeitverhalten des Radar-Management-Anwendungsfalls zu untersuchen, wurden vom ATC-System reale Plot- und Radarstatusnachrichten aufgezeichnet und die zugrundeliegende PHOENIX- und Tile-Set-Konfiguration gesichert. Der Event-Datenstrom diente bei den durchgeführten Messungen als Datenquelle des PHOENIX-Monitor-Adapters, die PHOENIX-Konfiguration als Datenquelle des PHOENIX-Discovery-Adapters und die Tile-Set-Konfiguration als Datenquelle des TileSet-Discovery-Adapters. Zusätzlich wurde der ORSLM-Management-Adapter so konfiguriert, dass er nach einer Minute drei zufällige Radare auswählt (deterministischer Zufallszahlengenerator), deren Ausfall simuliert und eine autonome Optimierung anstößt. Nach Abschluss der Optimierung setzt er den Radarstatus zurück und stellt so den Ausgangzustand wieder her, bevor er nach zwei weiteren Minuten einen erneuten Ausfall simuliert. Schon der Ausfall einer Radaranlage ist nach Aussage der DFS-Domänenexperten selten, es kann jedoch erschwerend hinzukommen, dass Anlagen durch geplante Wartungsarbeiten nicht zur Verfügung stehen. Daher wird die Simulation von drei parallelen Ausfällen als realistischer Worst-Case eingeschätzt.

Beim Start des Laufzeitsystems wurden für den simulierten Anwendungsfall im Wissensbasisinitialisierungsschritt der Initialisierungsphase (siehe Abschnitt 4.3.1, Seite 114) aus den eingebundenen Ontologiemodulen 5020 RDF-Tripel in die Wissensbasis importiert, die im Reasoning-Schritt der Regelgenerierung auf 8769 Tripel erweitert wurden. In den nachfolgenden Generierungsschritten wurden aus den extrahierten Management-relevanten Entitäten und OntML-Modulen in der Standardkonfiguration 1452 Zeilen DRL-Regeln erzeugt und als Regelbasis für die Laufzeitverarbeitung genutzt. Im Komponenteninitialisierungsschritt wurden die entwickelten Adapter ins Laufzeitsystem eingebunden. Der PHOENIX-Adapter importierte im Discovery-Schritt 26 Radare und deren Konfigurationskontext, der Tile-Set-Adapter 568 Observationsbereiche. Insgesamt wurden dabei 29 632 Tripel zur Wissensbasis hinzugefügt.

Zu Beginn der Laufzeitphase (siehe Abschnitt 4.3.1, Seite 120) wurden anhand der als Management-relevant klassifizierten Entitäten 1948 reguläre und 3108 temporale Tripel in die lokale Wissensbasis der Regel-Engine importiert. Anschließend wurden vom PHOENIX-Monitor-Adapter zwischen 500 und 1000 Events pro Sekunde ins System eingefügt, für die eine kontinuierliche Coverage-Detection und SLA-Überwachung durchgeführt wurde.

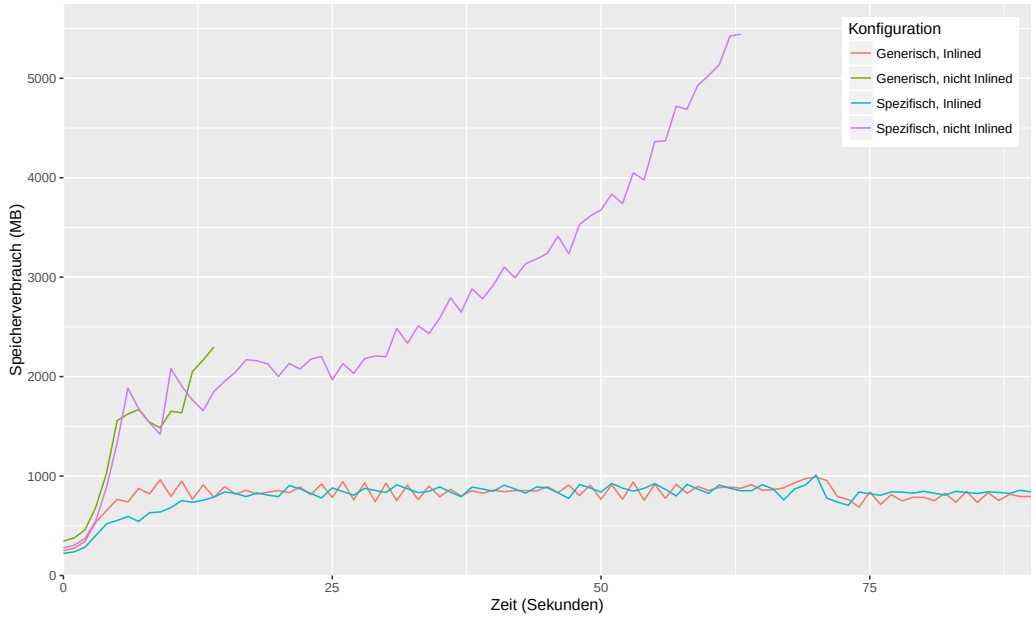


Abbildung 5.15.: Gemessener Speicherverbrauch des Laufzeitsystems für den Radar-Management-Anwendungsfall mit allen Konfigurationen.

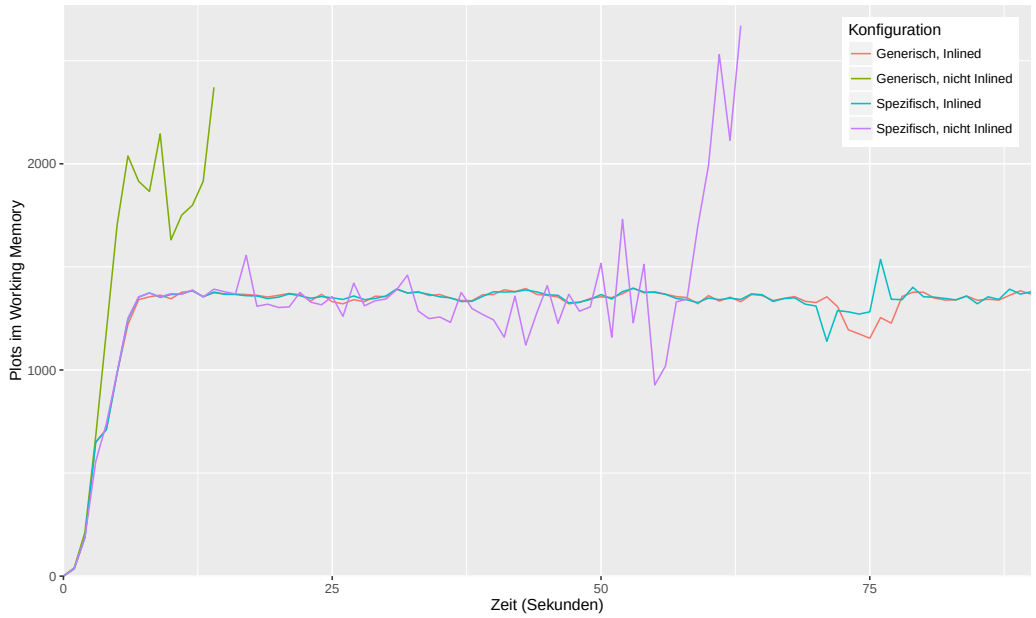


Abbildung 5.16.: Gemessene Plot-Event-Anzahl im Working Memory des Laufzeitsystems für den Radar-Management-Anwendungsfall.

Wie auch beim Storage-Management-Anwendungsfall wurden die Messungen für alle vier Konfigurationskombinationen durchgeführt. Dabei zeichnete sich allerdings ab, dass der Anwendungsfall ohne aktiviertes Inlining nicht umsetzbar ist.

Abbildung 5.15 zeigt den Speicherverbrauch, Abbildung 5.16 die Anzahl der aktuell im Working Memory der Regel-Engine gehaltenen Plot-Events. Beide Konfigurationen mit aktiviertem Inlining schwingen sich nach kurzer Zeit bei ca. 800 MB Speicherverbrauch ein und sind in der Lage, die eingehenden Events kontinuierlich abzuarbeiten. Der Speicherverbrauch des generischen Ansatzes ohne aktiviertes Inlining steigt bereits nach 15 Sekunden auf 2299 MB, die Regel-Engine ist nicht mehr in der Lage, das Event-Aufkommen zu verarbeiten und stürzt mit einem internen Fehler ab. Der spezifische Ansatz ohne aktiviertes Inlining benötigt zwar initial weniger Speicher als der generische Ansatz, der Bedarf wächst jedoch innerhalb von 64 Sekunden auf 5443 MB an. Auch hier zeigt sich, dass die Regel-Engine nach ca. 60 Sekunden nicht mehr in der Lage ist, das Event-Aufkommen zu bewältigen, bevor sie nach 66 Sekunden ebenfalls mit einem internen Fehler abstürzt.

Die Ursache liegt dabei in der „Filter-Mapped-Plots“-Regel, die gleichartige abgebildete Plots erkennt und aggregiert. Dazu muss die Regel-Engine für jeden neu abgebildeten Plot prüfen, ob bereits ein abgebildeter Plot mit identischem Radar und Observationsbereich existiert. Ohne aktiviertes Inlining müssen dabei für jedes Event die relevanten Statements aus dem Event-Dokument extrahiert werden, was einen enormen Verarbeitungsaufwand bedeutet. Bei aktiviertem Inlining werden das Radar und der Observationsbereich redundant als Event-Attribute mitgeführt und können so von der Regel-Engine indiziert werden. Um gleichartige Plots zu finden, reicht dann ein effizienter Index-Lookup. Würde man den Anwendungsfall ohne die Filterregel umsetzen, müssten alle abgebildeten Plots des 24-stündigen Zeitfensters im Working Memory gehalten und bei der Optimierung verarbeitet werden. So würden sich bei 500 Plots pro Sekunde über 43 Millionen Events ansammeln. Im Weiteren werden daher nur die generische und spezifische Konfiguration mit aktiviertem Inlining betrachtet.

Abbildung 5.17 zeigt den gemessenen Speicherverbrauch der betrachteten Konfigurationen des Anwendungsfalls. Die Konfiguration mit spezifischen Statements benötigt im Durchschnitt 1050 MB (967 MB außerhalb der Optimierung), die Konfiguration mit generischen Statements 1213 MB (1003 MB). Bei der autonomen Optimierung zeigen beide Konfigurationen deutliche Speicherspitzen; der maximale Verbrauch steigt bei der spezifischen Konfiguration auf bis zu 2552 MB, bei der generischen Konfiguration auf bis zu 3312 MB.

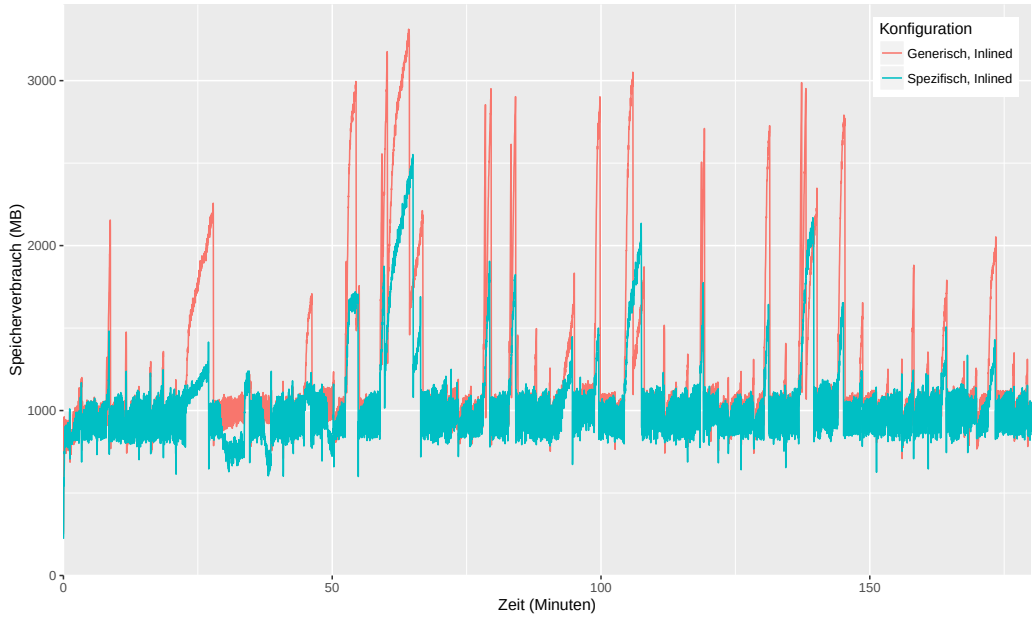


Abbildung 5.17.: Gemessener Speicherverbrauch des Laufzeitsystems für den Radar-Management-Anwendungsfall.

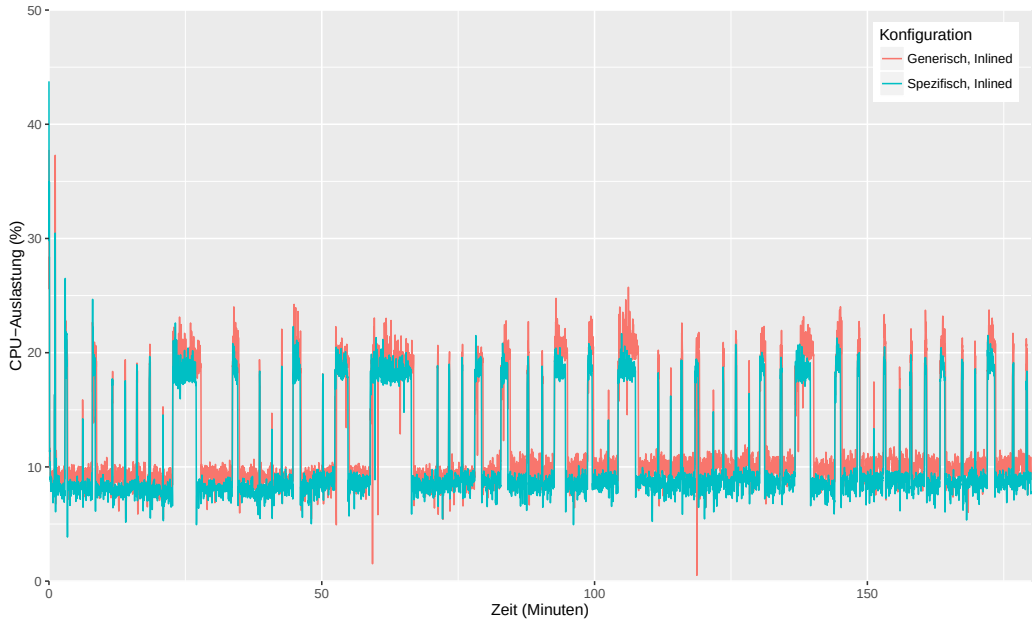


Abbildung 5.18.: Gemessene CPU-Auslastung des Laufzeitsystems für den Radar-Management-Anwendungsfall.

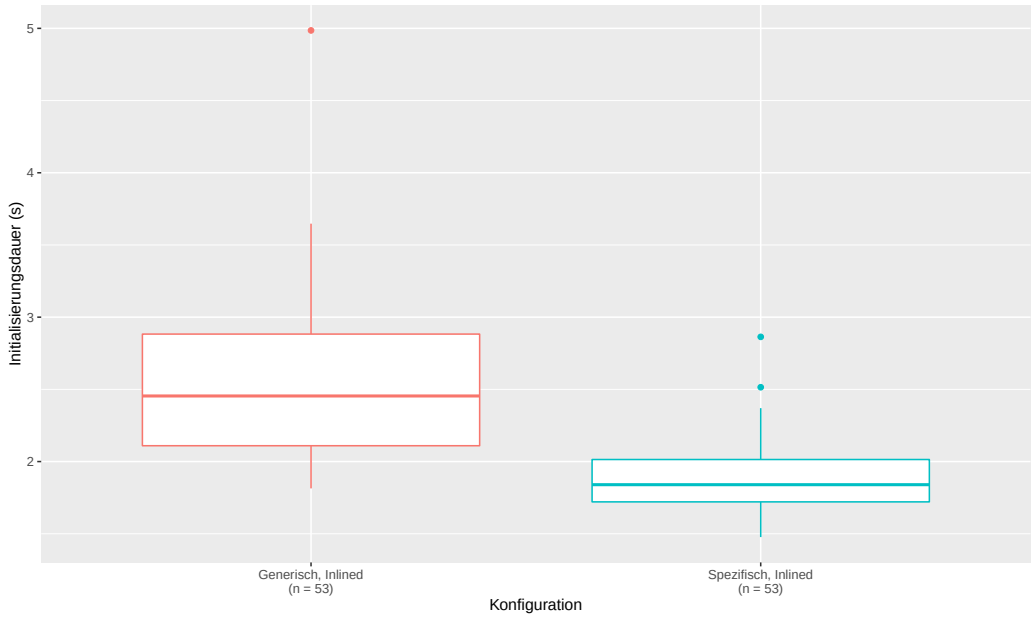


Abbildung 5.19.: Gemessene Initialisierungsdauer der autonomen Optimierung für den Radar-Management-Anwendungsfall.

Analog dazu zeigt die in Abbildung 5.18 dargestellte gemessene CPU-Auslastung beider Konfigurationskombinationen ein ähnliches Verhalten. Die spezifische Konfiguration erzeugt im Durchschnitt eine Auslastung von 10,84% (8,44% außerhalb der Optimierung), die generische Konfiguration von 12,13% (9,49%). Auch hier zeigen sich deutliche Auslastungspitzen bei der autonomen Optimierung. Dort steigt die CPU-Auslastung für die spezifische Konfiguration nach den ersten fünf Minuten auf maximal 24,67%, bei der generischen Konfiguration auf maximal 25,73%.

Über den Ressourcenverbrauch hinaus wurde auf Basis des Radar-Management-Anwendungsfalls die in der Optimierungsphase des Frameworks (siehe Abschnitt 4.3.1, Seite 122) durchgeführte autonome Optimierung vermessen, die sich aus dem Initialisierungs- und dem Optimierungsschritt zusammensetzt.

Abbildung 5.19 zeigt die Optimierungsinitialisierungsdauer beider Konfigurationstypen. Dabei wird eine Kopie der Verarbeitungslogik erzeugt, indem eine neue Drools Session mit einer der Laufzeit-Session identischen Regelbasis instanziiert wird, alle Fakten der Laufzeit-Session kopiert werden und anschließend eine initiale Regelauswertung durchgeführt wird. Dieser Vorgang dauert bei der spezifischen Konfiguration im Durchschnitt 1,90 Sekunden, bei der generischen Konfiguration 2,56 Sekunden. Der Erzeugungs- und Kopiervorgang dauert bei beiden Konfigurationen etwa gleich lang, da die Fakten der spezifischen Konfiguration bereits in der Laufzeit-Session in abgebildeter Form vorliegen. Die Zeitdifferenz resultiert fast ausschließlich aus der

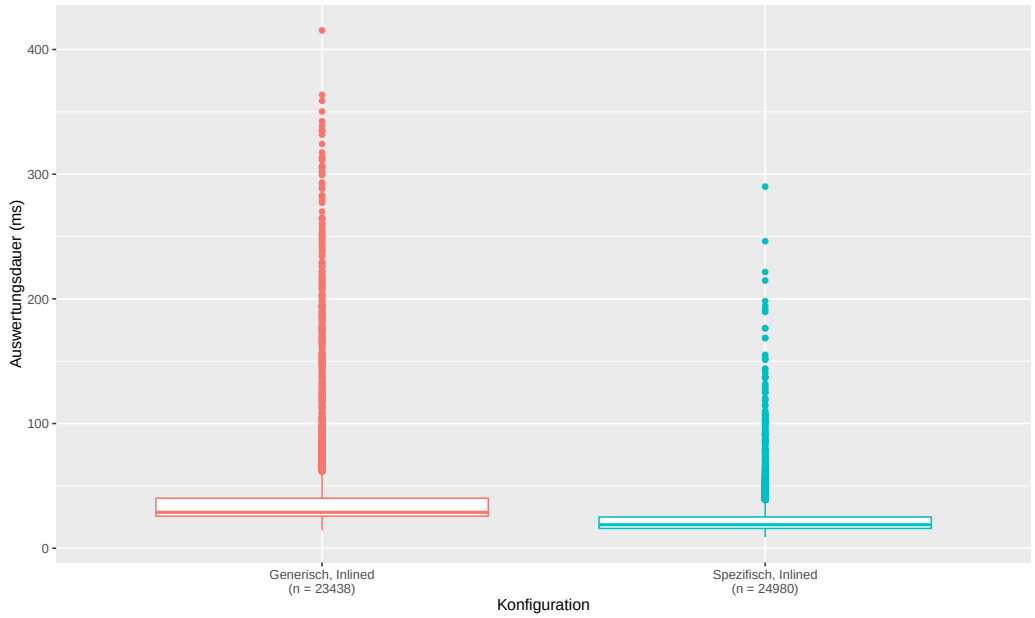


Abbildung 5.20.: Gemessene Auswertungsdauer einer Aktion bei der autonomen Optimierung des Radar-Management-Anwendungsfalls, mit herausgefilterten Messungen, die während einer Garbage-Collection stattfanden.

initialen Regelauswertung, bei der die spezifische Konfiguration durch die bessere Statement-Indizierung eine effizientere Regelauswertung erlaubt.

Nach Abschluss der Initialisierung findet die eigentliche Optimierung statt, bei der schrittweise alle aktuell möglichen Aktionen ausgewertet werden, die beste nicht-Tabu-Aktion ausgewählt und auf deren Basis weiteroptimiert wird, bis das Terminierungskriterium erreicht ist.

Abbildung 5.20 zeigt die Auswertungsdauern einzelner Aktionen. Die spezifische Konfiguration benötigt im Durchschnitt 22,12 ms, die generische Konfiguration 37,29 ms, um eine Aktion auszuwerten. Die Auswertung umfasst das Einfügen der Aktion in den Working Memory, das Feuern aller aktivierbaren Regeln, die Extraktion des Scores und der Wissensbasisänderungen und das Zurücksetzen der Regel-Engine auf den Ausgangszustand. Wie schon bei der Antwort- und Reaktionszeit des Storage-Anwendungsfalls sind die parallel zu einer Garbage Collection stattfindenden Messungen in der Abbildung herausgefiltert.

Abbildung 5.21 zeigt die Auswahl- und Umsetzungsdauer eines einzelnen Schrittes. Dabei werden die evaluierten Aktionen nach ihrem Score sortiert und die beste nicht-Tabu Aktion ausgewählt. Die Tabu-Prüfung geschieht dabei träge, es wird also nur der Tabu-Zustand einer Aktion bestimmt, wenn sie den besten Score hat oder alle besseren Aktionen als Tabu klassifiziert wurden. Im Durchschnitt dauert

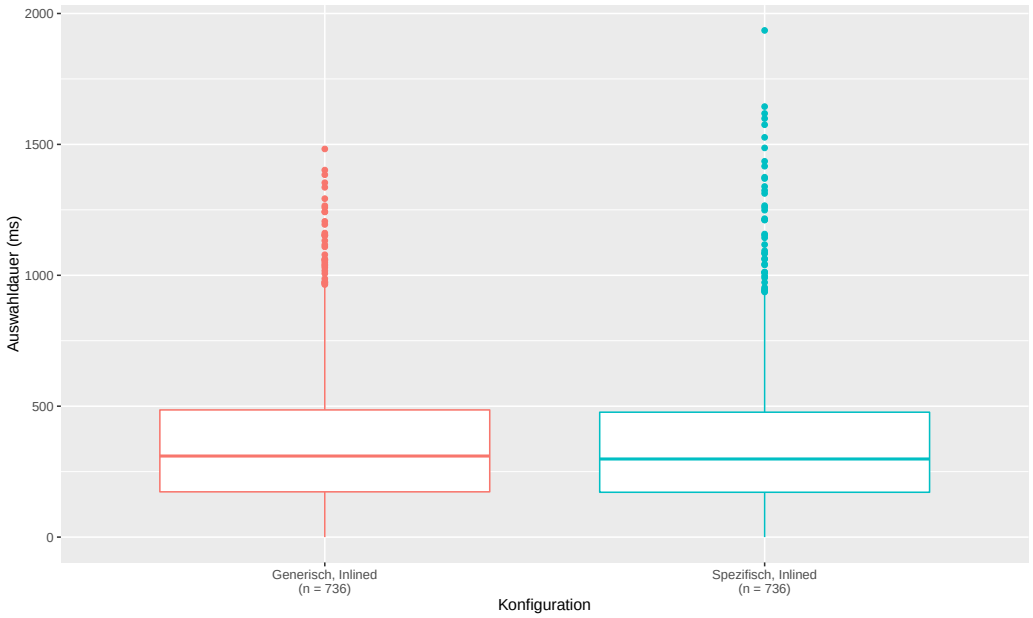


Abbildung 5.21.: Gemessene Auswahl- und Umsetzungsdauer eines Schrittes bei der autonomen Optimierung des Radar-Management-Anwendungsfalls.

dieser Vorgang bei der spezifischen Konfiguration 370 ms und bei der generischen Konfiguration 370 ms. Der Auswahlmechanismus ist bei beiden Konfiguration gleich, sie unterscheiden sich lediglich in der Aktionsumsetzung.

Die Dauer eines Optimierungsschrittes setzt sich aus der Summe der Auswertungsdauern der im Schritt evaluierten Aktionen und der Auswahl- und Umsetzungsdauer zusammen. Die Optimierungsdauer ist die Summe der Schrittdauern der durchlaufenen Schritte. Die Dauer der sich aus Initialisierungs- und Optimierungsschritt zusammensetzende Gesamtoptimierung sollte daher linear abhängig von der Schritt- und Aktionsevaluationsanzahl sein. Die in Abschnitt 5.7 dargestellten, aus den

Konfiguration	Koeffizient	Schätzung (s)	Standardfehler (s)	t-Wert	$\Pr(>\|t\|)$
Generisch	(Konstant)	$-16,718591$	$4,058462$	$-4,119$	$0,000142$
	Evaluationen	$0,086946$	$0,005274$	$16,487$	$< 2e^{-16}$
	Schritte	$1,853982$	$0,372397$	$4,979$	$8,01e^{-06}$
Spezifisch	(Konstant)	$-18,419542$	$4,698811$	$-3,920$	$0,00027$
	Evaluationen	$0,074610$	$0,006106$	$12,220$	$< 2e^{-16}$
	Schritte	$1,859263$	$0,431154$	$4,312$	$7,59e^{-05}$

Tabelle 5.7.: Lineares Regressionsmodell der Gesamtoptimierungsdauer in Abhängigkeit der Evaluationen und Schritte.

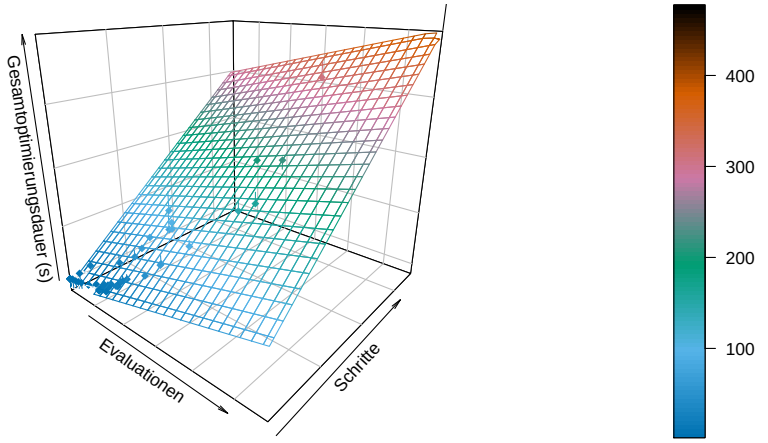


Abbildung 5.22.: Gesamtoptimierungsdauer in Abhängigkeit der Schritt- und Evaluationsanzahl für die spezifische Konfiguration. Regressionsebenenmodell aus Abschnitt 5.7.

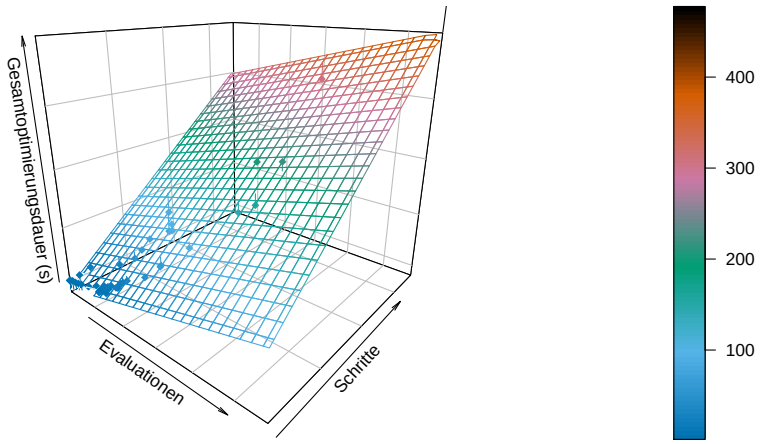


Abbildung 5.23.: Gesamtoptimierungsdauer in Abhängigkeit der Schritt- und Evaluationsanzahl für die generische Konfiguration. Regressionsebenenmodell aus Abschnitt 5.7.

Messdaten abgeleiteten linearen Regressionsmodelle bestätigen dies. Die spezifische Konfiguration (siehe Abb. 5.22) erzielt dabei ein Bestimmtheitsmaß (R^2) von 94,13%, die generische Konfiguration (siehe Abb. 5.23) von 96,45%. Testläufe mit mehreren, parallel Aktionen auswertenden Management-Logik-Kopien zeigten eine deutliche Reduktion der Optimierungsdauer, gingen jedoch auch mit einem deutlich erhöhten Ressourcenbedarf einher. Daher würde es Sinn machen, das Laufzeitsystem um ein verteiltes Auswertungskonzept zu erweitern, bei dem die Aktionen auf mehreren Knoten parallel ausgewertet werden.

Bei den durchgeführten Messungen wurden im Durchschnitt in 13,89 Schritten 495,83 Aktionen ausgewertet und dabei pro Schritt die Anzahl der SLA-Verletzungen um 0,69 reduziert. Die Gesamtoptimierungsdauer lag dabei bei der spezifischen Konfiguration bei durchschnittlich 44,39 Sekunden, bei der generischen Konfiguration bei 52,14 Sekunden. Maximal wurden in 57 Schritten 3596 Aktionen evaluiert, was bei der spezifischen Konfiguration 453,79 Sekunden, bei der generischen Konfiguration 477,57 Sekunden dauerte.

Insgesamt zeigt sich, dass das entwickelte Laufzeitsystem in der Lage ist, ein automatisiertes Radar-Management eines ATC-Systems durchzuführen und dabei kontinuierlich eine Event-Rate von 500 bis 1000 Events pro Sekunde zu verarbeiten und die SLA-Konformität von 568 Observationsbereichen zu prüfen. Bei der parallel zur regulären Verarbeitung durchgeführten Optimierung war das Laufzeitsystem in der Lage, das überwachte System ausschließlich auf Basis der vom Informations- und Management-Modell bereitgestellten Informationen autonom zu optimieren und so die Anzahl der SLA-Verstöße deutlich zu reduzieren. Mit aktivem Inlining konnte der Anwendungsfall auf einem herkömmlichen Desktop-System umgesetzt werden. Dabei zeigte die Konfiguration mit spezifischen Statements während der regulären Verarbeitung ein besseres Speicher- und Auslastungsverhalten und bei der autonomen Optimierung ein deutlich besseres Speicherverhalten, ein besseres Auslastungsverhalten und eine kürzere Optimierungsdauer als die Konfiguration mit generischen Statements.

5.3 Anforderungsüberprüfung

Auf Basis der in den vorherigen Abschnitten durchgeführten Fallstudien wird in diesem Abschnitt überprüft, ob der entwickelte Ansatz alle definierten Anforderungen erfüllt. Als Kernanforderungen für ein automatisiertes, wissensbasiertes IT-

Operations-Management-Framework wurden in Abschnitt 1.4 die folgenden Ziele definiert:

- Es muss eine kontextsensitive Event-Vorverarbeitung umgesetzt sein, mit der Überwachungsdaten des gemanagten Systems vor der komplexen Detailanalyse gefiltert, angereichert und aggregiert werden können. Kontextsensitiv bedeutet dabei, dass die Verarbeitung auf einer semantischen Ebene stattfindet, bei der das in der Wissensbasis hinterlegte Systemmodell einbezogen werden kann.
- Es muss ein kontextsensitiver Event-Abbildungsmechanismus bereitgestellt werden, mit dem die in den Überwachungsdaten enthaltenen Informationen auf das Systemmodell abgebildet werden können.
- Es müssen Analyseverfahren existieren, mit denen nicht beobachtbare Zustände und Qualitätsindikatoren des überwachten Systems innerhalb der Wissensbasis automatisch abgeleitet werden können.
- Es müssen Planungsverfahren bereitgestellt werden, die anhand des in der Wissensbasis repräsentierten Systemzustands entscheiden, welche Management-Aktionen auf dem realen System umgesetzt werden sollen.
- Die betrachteten Management-Aspekte müssen sich in einem einheitlichen Management-Modell beschreiben lassen, um einer Wissensfragmentierung entgegenzuwirken.
- Der Ansatz muss so generisch sein, dass er sich mit geringem Aufwand auf arbiträre Anwendungsfälle anwenden lässt.

Neben den Kernanforderungen galt es, Lösungen für allgemeine Probleme zu finden, die sich aus dem Einsatz von Ontologien als zentrales IT-Operations-Management-Informationsmodell ergeben:

- Die für die Ontologieverarbeitung grundlegenden Validierungs- und Schlussfolgerungsalgorithmen skalieren mit steigender Instanzfülle schlecht, was zu langen Management-Zyklen bis hin zu nicht verarbeitbaren Modellen führt.
- Da OWL über kein designiertes Zeitkonzept verfügt, sondern alle darin modellierten Aussagen universell gültig sind, können keine zeitlichen Verläufe und vergangene Zustände dargestellt und somit auch bei der Analyse und Planung nicht einbezogen werden.

Im Folgenden werden die dafür in dieser Arbeit entwickelten Lösungsansätze vorgestellt und anhand ihres Einsatzes bei den durchgeführten Fallstudien überprüft.

Zur Umsetzung der **kontextsensitiven Event-Vorverarbeitung** wurde zunächst auf konzeptueller Ebene entschieden, Events innerhalb des Laufzeitsystems als RDF-

Dokumente zu repräsentieren und auszutauschen. Ihre Struktur wird mit der Taxonomie einer Schnittstellenontologie beschrieben, die als Teil des Management-Modells entwickelt wurde. Überwachungs-Events müssen von den anwendungsfallspezifischen Technologieadaptern in das entsprechende Format überführt werden, bevor sie in das Laufzeitsystem eingespeist werden können. Die Events können dann mit den als Teil des Management-Modells entwickelten Event-Processing-Regeln unter Einsatz von temporalen Vergleichsoperatoren und Zeitfenstern gebunden, aus den RDF-Dokumenten Ressourcen und Literale extrahiert und diese mit den Systemmodellentitäten verknüpft werden. Als Konsequenz können Event-Processing-Regeln Events aus dem Datenstrom entfernen oder neue Events hinzufügen, sodass Filterung, Aggregation und Anreicherung möglich sind. Zur Laufzeit werden sowohl die Events, als auch das Systemmodell gemeinsam im Working Memory einer CEP-fähigen Regel-Engine gehalten, die die Regeln durch Verknüpfung beider Datenquellen umsetzt.

In der Storage-Management-Fallstudie wurde keine Event-Vorverarbeitung eingesetzt, da die Metriken direkt auf das Informationsmodell abgebildet werden konnten. In der Radar-Management-Fallstudie konnte gezeigt werden, dass die Event-Processing-Regeln in der Lage sind, die vom ATC-System empfangenen Radarmessungen mit dem Systemmodell zu verknüpfen, mit Kontextinformationen anzureichern, den Datenstrom auszudünnen und eine kontinuierliche Coverage-Detection durchzuführen. Dazu wurde aus jeder empfangenen Radarmessung der SAC/SIC und die Messposition extrahiert, anhand derer das Radar- und Observationsbereichsindividuum im Systemmodell gesucht und damit eine um die Individuen angereicherte Messung erzeugt. Messungen, für die kein Kontext gefunden wurde, wurden von der Regel-Engine automatisch gelöscht, wenn sie das Zeitfenster verließen. Die angereicherten Events wurden zudem aggregiert, indem bei zwei Events mit gleichem Payload das ältere verworfen wurde. Die Verarbeitung der vom Anwendungsfall erzeugten 500 - 1000 Events pro Sekunde zeigte, dass der Ansatz durchaus in der Lage ist, das Event-Aufkommen mittelgroßer IT-Systeme zu bewältigen.

Zur Umsetzung der **kontextsensitiven Event-Abbildung** wurden im entwickelten Management-Modell sogenannte Event-Mapping-Regeln vorgesehen. Analog zu den Event-Processing-Regeln können sie in ihrer Bedingung Events aus dem Überwachungsdatenstrom und Instanzen aus dem Systemmodell verknüpfen. In der Regelkonsequenz kann das Systemmodell beeinflusst werden, indem Aussagen hinzugefügt oder daraus entfernt werden. Ein Blank-Node-Operator erlaubt es, deterministische Ressourcennamen zu generieren, sodass auch neue Instanzen hinzugefügt werden können.

Im vorgestellten Storage-Management-Anwendungsfall wurde gezeigt, wie Abbildungsregeln genutzt werden können, um als Events vorliegende Performance-Metriken auf das Informationsmodell abzubilden. Dabei wurde sekündlich aus 150 Metriken die ID, die aktuelle Antwortzeit und die I/O-Rate einer virtuellen Disk extrahiert, das entsprechende VDisk-Individuum im Systemmodell gesucht und dessen aktuelle Antwortzeit und I/O-Rate aktualisiert. In der Radar-Management-Fallstudie wurde gezeigt, dass Abbildungsregeln in der Lage sind, aktualisierte Radarzustände auf das Systemmodell abzubilden. Über den SAC/SIC und die als Zeichenketten kodierten Sensorstatus wurden das Radarindividuum und die Statusindividuen in der Wissensbasis gesucht und die Statusrelationen aktualisiert.

Zur Umsetzung der **Analyseverfahren** wurden im entwickelten Management-Modell funktionale Abhängigkeiten vorgesehen, mit denen Zusammenhänge zwischen Modellinstanzen beschrieben werden können. Anders als bei existierenden Ansätzen sind die Analysevorschriften nicht als SWRL-Regeln formuliert, die unter der Open-World-Annahme als Teil des OWL-Reasonings ausgewertet werden, sondern oberhalb des Reasoning auf der Anwendungsschicht angesiedelt, wo sie unter der Closed-World-Annahme ausgewertet und die abgeleiteten Fakten explizit im Systemmodell materialisiert werden. Dadurch ist es möglich, Negation und Aggregation auszudrücken, die zur Bestimmung vieler IT-Management-relevanter Qualitätsindikatoren benötigt werden. Dies führt jedoch auch dazu, dass im Analysemodell unentscheidbare Zusammenhänge definiert werden können und es dem Domänenexperten obliegt, die Entscheidbarkeit sicherzustellen.

Im Storage-Management-Anwendungsfall wurden funktionale Abhängigkeiten genutzt, um auszudrücken, dass eine virtuelle Disk die Speicherklasse ihrer Gruppe hat und die Auslastung eines Pools die Summe der Auslastungen seiner Platten ist. In der Radar-Management-Fallstudie wurde mit funktionalen Abhängigkeiten beschrieben, dass ein Observationsbereich gegen sein SLA verstößt und einer entsprechenden Violator-Klasse angehört, wenn die Anzahl der ihm zugewiesenen Radare eines Features unter der im SLA festgelegten Mindestanzahl liegt.

Zur Umsetzung der **Planungsverfahren** wurden zwei Ansätze verfolgt: ein auf Policies basierendes, automatisiertes Management und ein auf Optimierungsalgorithmen basierendes, autonomes Management. Beim Policy-basierten Management werden im Management-Modell den Complex-Event-Processing-Regeln ähnliche Policies definiert. Als Konsequenz beeinflussen sie jedoch nicht den Event-Datenstrom, sondern leiten aus dem Systemmodell Management-Aktionen ab, die unverzüglich auf dem überwachten System umgesetzt werden. Bei der autonomen Optimierung werden im Management-Modell Aktionsvorschlagsregeln, Aktionsimplementierungen und

Bewertungsfunktionen definiert, die von einem Optimierungsalgorithmus genutzt werden, um parallel zur Laufzeit eine Kopie des Systemmodells zu optimieren. Als Optimierungsverfahren wurden Hill-Climbing-basierte, metaheuristische Optimierungsverfahren ausgewählt, da sie auch für unvollständige und unvollkommene Informationen mit eingeschränkten Berechnungsressourcen eine hinreichend gute Lösung finden können und so für den Einsatz bei arbiträren Anwendungsfällen geeignet sind. Mit ihnen wird schrittweise eine optimierte Lösung angenähert und die dazu führende Optimierungsagenda einem Administrator bereitgestellt, der über deren Anwendung auf das reale System entscheidet.

In der Storage-Management-Fallstudie wurde gezeigt, dass die im Management-Modell ausdrückbaren Policies in der Lage sind, ein überwachtes System bei SLA-Verstößen automatisiert in einen konformen Zustand zurückzuführen. Dabei wurde eine Kombination aus drei Policies eingesetzt, die auf Basis der aktuellen Antwortzeit und I/O-Rate virtueller Disks angepasste I/O-Grenzen an das überwachte System kommandiert, um High-Cost-Speicher temporär zu schützen.

In der Radar-Management-Fallstudie wurde gezeigt, dass die autonome Optimierung in der Lage ist, bei Radarausfällen eine Rekonfigurationsagenda abzuleiten und so die Anzahl an SLA-Brüchen zu senken. Dabei mussten die Seiteneffekte der Management-Aktionen (anders als bei Policies) nicht explizit berücksichtigt werden, sondern wurden implizit durch Verknüpfung der Aktionsimplementierung und dem Analysemodell abgeleitet.

Das **einheitliche Management-Modell** wurde umgesetzt, indem eine ontologische Taxonomie entwickelt wurde, mit der alle oben aufgeführten Verarbeitungsschritte beschrieben und dabei direkter, semantischer Bezug auf die Domänenentitäten genommen werden konnte. Bei der Umsetzung der Anwendungsfälle zeigte sich, dass eine manuelle Modellierung der Management-Logik mit Standardontologiewerkzeugen nicht praktikabel ist. Daher wurde eine domänenspezifische Sprache entwickelt, die die Zugänglichkeit und den Entwicklungskomfort erhöht, indem die Management-Regeln in einer RIF-artigen Syntax beschrieben und dann automatisch auf die ontologische Taxonomie des Management-Modells abgebildet werden. Mit dem Modell ließen sich alle Management-Aspekte der betrachteten Anwendungsfälle in einheitlicher Form beschreiben. Die resultierende Verknüpfung aus Domänen- und Management-Modellen bietet so eine zusammenhängende Gesamtsicht des Anwendungsfalls und der Management-Logik.

Um den entwickelten Ansatz mit **geringem Aufwand auf arbiträre Anwendungsfälle anwenden zu können**, wurde das in Abschnitt 4.3 (siehe Seite 108) vorgestellte Management-Framework entwickelt, das sich aus Vorgehensmodell und Lauf-

zeitsystem zusammensetzt. Das Vorgehensmodell beschreibt, wie Domänenmodelle, Management-Modelle und Technologieadapter entwickelt und in das Laufzeitsystem integriert werden. Das Laufzeitsystem ist eine Software-Komponente, die sich automatisch auf die bereitgestellten Modelle und Adapter adaptiert und die im Management-Modell beschriebene Logik mit Hilfe einer Regel-Engine umsetzt. In Abschnitt 5.1.2 und Abschnitt 5.2.2 konnte bereits gezeigt werden, dass der entwickelte Ansatz in der Lage ist, unterschiedliche Anwendungsfälle umzusetzen. Im Folgenden wird der dafür benötigte Aufwand betrachtet.

Bei der Storage-Management-Fallstudie wurde das SVC- und BVQ-Domänenmodell bereits in einem Vorprojekt automatisiert aus einem existierenden Modell abgeleitet. Die zur Umsetzung des Anwendungsfalls vorgenommenen Erweiterungen und die modellierte Schnittstellenontologie beliefen sich auf acht Klassen, sechs Rollen und drei Individuen, die von 50 Axiomen beschrieben wurden. Die Management-Logik wurde mit sechs abstrakten Zielen und 15 Regeln in 229 Zeilen der OntML-DSL ausgedrückt. Der Umfang der entwickelten Adapter belief sich auf 505 Zeilen Java-Code (1415 inkl. Benutzeroberfläche) und 52 Zeilen Konfigurations-XML. Der Gesamtintegrationsaufwand mit 836 Zeilen Code wird als gering angesehen.

Bei der Radar-Management-Fallstudie wurde eine 195 Zeilen Java-Code umfassende Transformationskomponente entwickelt, die das existierende PHOENIX-Konfigurationsmodell auf eine Ontologie abbildet. Anschließend wurden manuell 19 Klassen, 33 Rollen und sechs Individuen modelliert, die mit 161 Axiomen die Tile-Set-Konfiguration und die Schnittstelle beschreiben und die betrachteten Domänen miteinander verknüpfen. Die Management-Logik wurde mit sechs abstrakten Zielen und 23 Regeln in 346 Zeilen der OntML-DSL ausgedrückt. Der Umfang der entwickelten Adapter belief sich auf 1726 Zeilen Java-Code (2398 inkl. Benutzeroberfläche) und 58 Zeilen Konfigurations-XML. Die Integration in das Flugsicherungssystem belief sich auf 1013 Zeilen C/C++-Code (2214 inkl. Benutzeroberfläche und Erweiterungen). Der Gesamtintegrationsaufwand war mit 3304 Zeilen Code zwar deutlich höher als beim Storage-Management-Anwendungsfall, die entwickelten Adapter wurden jedoch so generisch angelegt, dass weitere Anwendungsfälle mit deutlich geringerem Aufwand umgesetzt werden können.

Um die **Skalierbarkeit des Reasonings** zu verbessern, wurde in Abschnitt 3.2.3 (siehe Seite 56) eine Untersuchung existierender IT-Management-Modelle durchgeführt und im Zuge dessen das OWL-RL-Profil als geeignete Sprachuntermenge ausgewählt. Dadurch konnte die Reasoning-Komplexität von `NEXPTIME`-vollständig (für OWL-DL) auf `PTIME`-vollständig reduziert und das Reasoning in einer Regel-Engine umgesetzt werden. Durch die zur Laufzeit statische Domänentaxonomie können dabei speziali-

sierte Regeln eingesetzt werden, die sich auf den für das Management relevanten Modellausschnitt beschränken. Bei den Anwendungsfällen zeigte sich jedoch, dass die existierenden proprietären Modelle an einigen Stellen Kardinalitätseinschränkungen verwenden, die von OWL-RL nicht berücksichtigt werden können. So darf im Storage-Modell beispielsweise eine virtuelle Disk nur maximal zwei Speicherabbilder haben und ein Cluster nur aus maximal sechs I/O-Gruppen bestehen, die sich jeweils aus genau zwei Knoten zusammensetzen. Beim Radar-Management sieht das transformierte Konfigurationsmodell ebenfalls für einige Knoten und Attribute Kardinalitätseinschränkungen größer eins vor. Die fehlende Ausdrucksmöglichkeit dieser Aspekte führt dazu, dass gewisse Inkonsistenzen in den Instanzdaten nicht erkannt werden können, beispielsweise wenn eine virtuelle Disk mehr als zwei disjunkte Abbilder hat. Solche Inkonsistenzen entstehen jedoch in der Regel nur, wenn bereits das Domänenmodell, die Datenquelle oder das Abbildungsverfahren inkonsistent sind. Durch Qualitätssicherungsmaßnahmen im Entwicklungsprozess kann daher die Fehlerwahrscheinlichkeit bereits vorab minimiert werden. Zudem können SPARQL-Queries genutzt werden, um die Einhaltung der Constraints zyklisch zu prüfen. In keinem der betrachteten Anwendungsfälle trat eine entsprechende Inkonsistenz auf.

Für die **temporale Darstellung** wurden mehrere existierende Ansätze untersucht und der 4D-Fluents-Ansatz ausgewählt, da er sich durch seine Adaptivität und Standardkonformität als Entwurfsmuster auf existierende Ontologien anwenden lässt. In beiden Fallstudien konnten dadurch die Domänenmodelle zunächst unabhängig vom Zeitkonzept entwickelt bzw. transformiert und anschließend durch Anwendung des Fluent-Entwurfsmusters in Erweiterungsontologien temporal ausgezeichnet werden, ohne die Ursprungsmodelle anzupassen. Dabei konnten sowohl bei der Modellierung als auch bei der Laufzeitverarbeitung Standardwerkzeuge und -bibliotheken eingesetzt werden. Lediglich bei der Interaktion mit den Instanzdaten muss sich der Benutzer bzw. dessen Werkzeuge der temporalen Rollen und Klassen bewusst sein und die entsprechenden Fluent-Strukturen berücksichtigen. Die für das Management-Modell entwickelte Adaption ist „Fluent-Aware“: der Benutzer muss Fluent-Strukturen bei der Regelformulierung nicht explizit berücksichtigen, sie werden bei der Adaption automatisch erkannt und entsprechend abgebildet.

Die Fluent-Darstellung ist bei der Laufzeitdatenhaltung mit einem deutlichen Mehraufwand verbunden. So werden bei einer temporalen Beziehung drei Tripel für das Intervall, vier Tripel für jede temporale Ausprägung und ein Tripel für die eigentliche Beziehung benötigt. Damit kostet eine temporale abstrakte Beziehung zwölf Tripel und temporale konkrete Beziehungen und Klassenzuweisungen acht Tripel. Besonders beim Storage-Management-Anwendungsfall, bei dem die aktuelle Antwortzeit

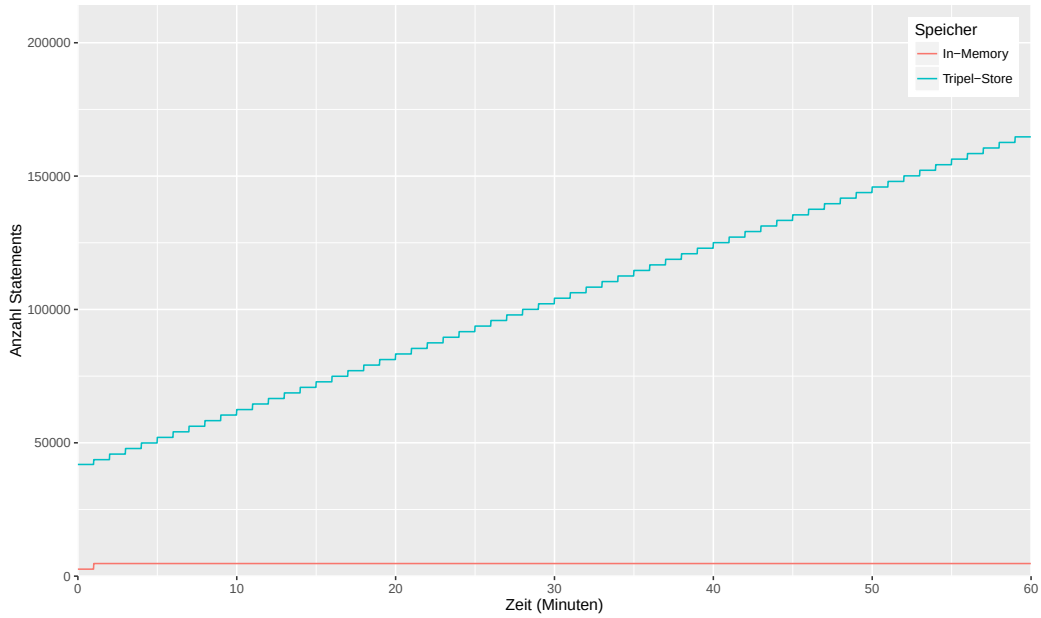


Abbildung 5.24.: Anzahl der RDF-Tripel im Triplestore und im In-Memory-Abbild.

und die I/O-Rate stetig aktualisiert werden, wächst die Wissensbasis dadurch schnell an. Für die in der Fallstudie betrachtete Storage-Konfiguration bedeutet dies, dass in jedem Metrik-Zyklus 3360 Tripel zur Wissensbasis hinzugefügt werden (150 VDisks, zwei Kenngrößen pro VDisk, 14 Tripel pro Kenngröße).

Um die Instanzdaten zur Laufzeit effizient zu verwalten, wurde daher ein hybrider Datenhaltungsansatz entwickelt, der die Vorteile eines Triplestores und einer In-Memory-Datenhaltung vereint. Dabei wird das vollständige Systemmodell persistent und abfragbar in einem Triplestore, die davon für die Verarbeitungslogik und den Reasoner relevante Untermenge in einem In-Memory-Abbild gehalten. Die Management-Logik und das OWL-Reasoning arbeiten ausschließlich auf dem In-Memory-Abbild. Alle abgeleiteten Informationen werden auf den Triplestore repliziert, um ein vollständiges Systemmodell zu erhalten. Für die Verarbeitung irrelevante Daten verwirft das In-Memory-Abbild und reduziert so die im Speicher zu haltende und von der Logik zu verarbeitende Datenmenge deutlich.

Der Reduktionseffekt zeigt sich besonders beim Ausphasen historischer Daten, die die von den Management-Regeln betrachteten Zeitfenster verlassen. Abbildung 5.24 zeigt das Verhältnis zwischen den im Triplestore gehaltenen RDF-Tripeln und der davon im In-Memory-Abbild gehaltenen Untermenge für die kontinuierlich Daten hinzufügende Storage-Management-Fallstudie. Bei der Messung wurde der reale Metrikzyklus von einer Metrik pro VDisk pro Minute simuliert. Zu Beginn der Messung sieht man, dass vom System 2626 der 41 883 Tripel als Management-relevant klassi-

fiziert und in das In-Memory-Abbild aufgenommen werden. Bei der Abbildung der ersten Metriken nach ca. einer Minute steigt die Tripelanzahl im In-Memory-Abbild auf 4726 und im Triplestore auf 43 683. Bei allen nachfolgenden Metrikabbildungen werden die Fluents der alten Werte geschlossen. Die Management-Logik erkennt über die in den Regeln definierten Zeitfenster, dass die alten Daten nicht mehr benötigt werden und entfernt sie aus dem In-Memory-Abbild. Der Triplestore hingegen legt die historischen Werte persistent ab. Bei einem produktiven Einsatz sollten im Triplestore Aggregationsmechanismen zur Komprimierung der historischen Daten eingesetzt werden. Dabei bieten sich temporale Aggregationssverfahren an, die Daten in mehreren Stufen unterschiedlicher Granularität verdichten. Zhang et al. stellen dazu in [128] zwei Ansätze vor, mit denen Daten entweder auf Basis eines begrenzten Speicherplatzes oder eines festen Zeitfensters aggregiert werden können. VMware vSphere[9] setzt beispielsweise ein solches Verfahren ein, um Echtzeitstatistiken alle fünf Minuten zu einem Datenpunkt zusammenzufassen, der einen Tag vorgehalten wird, sechs solcher Datenpunkte nach 30 Minuten zu einem weiteren Datenpunkt zusammenzufassen, der 1 Woche vorgehalten wird, usw., sodass die Datenmenge deutlich reduziert wird.

Insgesamt zeigt sich damit, dass der entwickelte Ansatz allen in den Zielen an ein automatisiertes, wissensbasiertes IT-Operations-Management-Framework gestellten Anforderungen gerecht wird.

[9] `http://www.vmware.com/products/vsphere.html`

6 Zusammenfassung und Ausblick

6.1 Zusammenfassung

Die Komplexität und Dynamik moderner IT-Systeme erfordert ein Umdenken im IT-Operations-Management. Die Verknüpfung eines ontologiebasierten Informationsmodells, das die bisher isoliert betrachteten Domänen zu einer abstraktionsebenenübergreifenden Gesamtsicht vereint, und eines autonomen Managers, der selbstständig Daten akquiriert, analysiert, Entscheidungen trifft und umsetzt, wurde als richtungsweisend eingestuft. In dieser Arbeit wurde dazu untersucht, inwiefern sich die im Kontext des Semantic Webs entstandene Web Ontology Language (OWL) und damit verbundene Standards mit der im automatisierten IT-Operations-Management als Standardarchitektur etablierten MAPE-K Loop verknüpfen lässt, um ein allgemeingültiges, wissensbasiertes IT-Operations-Management-Framework zu konzipieren, umzusetzen und zu evaluieren. Dazu wurden die definierten Anforderungen und identifizierten Problemfelder zunächst dediziert untersucht und existierende Ansätze betrachtet. Für Bereiche, in denen kein adäquater Ansatz gefunden werden konnte, wurden neue Ansätze entwickelt.

Für die ontologiebasierte Wissensbasis wurde untersucht, wie deren adaptierbares Informationsmodell aufgebaut sein muss, wie darin temporales Wissen repräsentiert werden kann, wie die Effizienz des Reasonings gesteigert werden kann, wie das Management-Wissen darin in einheitlicher Form ausgedrückt werden kann und wie Instanzdaten zur Laufzeit effizient verwaltet werden können. Dabei zeigte sich, dass das Modularisierungskonzept von OWL, mit dem relevante Domänen dediziert modelliert und nachträglich miteinander verknüpft werden können, hinreichend ist, um ein adaptierbares Informationsmodell zu schaffen. Für die Darstellung von temporalem Wissen wurde der 4D-Fluents-Ansatz ausgewählt, da er als standardkonformes Entwurfsmuster auf existierende Ontologien angewandt werden kann, ohne deren Ursprungsmodell anpassen zu müssen, und sich mit existierenden Werkzeugen und Bibliotheken verarbeiten lässt.

Da zur einheitlichen Modellierung der Management-Logik kein hinreichendes Modell gefunden werden konnte, wurde mit der Ontology-based Management Language (OntML) ein neuartiges Management-Modell entwickelt, mit dessen ontologischer Taxonomie alle als relevant identifizierten Management-Aspekte als Ontologieinstanzen beschrieben werden können. Dadurch ist es möglich, in der als Dialekt des Rule Interchange Formats (RIF) angelegten Regelsprache direkten, semantischen Bezug auf die Konzepte, Rollen und Instanzen der betrachteten Problemdomäne zu nehmen, sodass eine zusammenhängende und validierbare Gesamtsicht der Domäne und des Verarbeitungsprozesses entsteht.

Da keiner der existierenden Ansätze ein generisches und adaptierbares Verfahren zur Verbesserung der Reasoning Performance beim ontologiebasierten IT-Operations-Management bot, wurden die OWL-Sprachprofile auf ihre Eignung als Informationsmodellgrundlage hin untersucht. Dabei stellte sich das OWL-RL-Profil als geeignete Sprachgrundlage für die im Framework einzusetzenden Domänenmodelle heraus. Zwar lassen sich damit keine Kardinalitätseinschränkungen größer Eins ausdrücken, dafür kann es effizient und skalierbar in einer Regel-Engine umgesetzt werden.

Zur effizienten Laufzeitdatenhaltung und weiteren Verbesserung der Reasoning Performance wurde ein hybrider Datenhaltungsansatz entwickelt, der die Vorteile eines Triplestores und einer In-Memory-Ablage kombiniert. Dabei wird ein vollständiges Systemmodell im Triplestore, der davon Management-relevante Teil für die Verarbeitungskomponenten effizient zugreifbar in einem In-Memory-Abbild gehalten. Die relevanten Aussagen werden automatisch aus dem Management-Modell abgeleitet, indem für die darin referenzierten Domänenkonzepte und -rollen durch statische Analyse der Informationsmodelltaxonomie die transitive Ableitungshülle gebildet wird. Aus den generischen OWL-RL-Regeln werden dann spezialisierte ABox-Regeln abgeleitet, die zur Laufzeit das kontinuierliche Reasoning effizient auf dem In-Memory-Abbild umsetzen.

Die ausgewählten und entwickelten Konzepte erlauben es so, in der Wissensbasis eine einheitliche und abstraktionsebenenübergreifende Repräsentation des überwachten Systems zu halten, dessen Zustand über die Zeit nachvollzogen werden kann. Die im Regelkreis umzusetzende Management-Logik ist nicht über unterschiedliche Quellen und Formate fragmentiert, sondern ebenfalls in einheitlicher Form, unter direkter Bezugnahme der Domänenentitäten definiert, sodass ein nachvollziehbarer Gesamtverarbeitungsprozess entsteht. Die hybride Datenhaltung bietet im Triplestore eine vollständige Systemsicht, die für externe Analysen und Datenvisualisierungen herangezogen werden kann. Die davon In-Memory gehaltene, Management-relevante Untermenge mit angebundenem OWL-RL-Reasoning bietet

den Verarbeitungskomponenten des Regelkreises einen effizienten Zugriff und eine reaktive Schnittstelle, die kurze Reaktionszeiten ermöglichen.

Für den Regelkreis wurde untersucht, wie eine kontextsensitive Event-Vorverarbeitung umgesetzt werden kann, wie der Zustand des überwachten Systems auf die Wissensbasis abgebildet werden kann, wie das in der Wissensbasis gehaltene Systemmodell analysiert und daraus nicht beobachtbare Zustände und Qualitätsindikatoren abgeleitet werden können und wie automatisiert Management-Entscheidungen getroffen und umgesetzt werden können. Für die kontextsensitive Event-Vorverarbeitung wurde ein Ansatz aus dem Bereich des Semantic Complex Event Processings adaptiert, bei dem Events als RDF-Dokumente repräsentiert, die darin enthaltenen Tripel in Regeln mit dem Systemmodell verknüpft und daraus Datenstromoperationen abgeleitet werden. Bei der Informationsabbildung wurde ein kombinierter Ansatz entwickelt, bei dem initial eine vollständige Momentaufnahme des überwachten Systems in die Wissensbasis importiert wird, Anpassungen daran zur Laufzeit jedoch ausschließlich Event-getrieben geschehen, indem bei einer kontextsensitiven Verarbeitung aus Überwachungs-Events relative Wissensbasisänderungen abgeleitet werden.

Zur Modellanalyse wurden funktionale Abhängigkeiten eingesetzt, die als logische Implikationen auf Anwendungsebene unter Closed-World-Annahme ausgewertet und deren Schlussfolgerungen explizit im Modell materialisiert werden. Dadurch sind, anders als bei den in verwandten Arbeiten eingesetzten SWRL-Regeln, die für das IT-Operations-Management wichtigen Aggregationen und Negationen ausdrückbar, es erlaubt jedoch auch die Formulierung unentscheidbarer Abhängigkeiten.

Für das automatisierte Management wurde ein duales Konzept aus Policy-basiertem Management und autonomer Optimierung entwickelt. Als Produktionsregeln formulierte Policies werden kontinuierlich gegen die Wissensbasis ausgewertet und abgeleitete Management-Aktionen unverzüglich auf dem überwachten System umgesetzt. Bei der autonomen Optimierung wurden metaheuristische Verfahren eingesetzt, die auf Basis von im Management-Modell definierten Aktionsquellen, Aktionsimplementierungen und Bewertungsfunktionen selbstständig Management-Aktionen zu einer mehrstufigen Rekonfigurationsagenda verknüpfen. Dabei werden im Analysemodell beschriebene globale Effekte implizit berücksichtigt.

Die für den Regelkreis ausgewählten und entwickelten Konzepte erlauben es einem autonomen Manager so, kontinuierlich Monitoring-Events des überwachten Systems zu empfangen, kontextsensitiv zu aggregieren und zu filtern und auf Zustandsänderungen des initial importierten und in der Wissensbasis gehaltenen Systemmodells abzubilden. Durch funktionale Abhängigkeiten werden bei der Modellanalyse nicht

beobachtbare Zustände und Qualitätsindikatoren abgeleitet, die als Eingabe für Policies dienen, die anhand explizit definierter Kriterien in nahezu Echtzeit Management-Entscheidungen treffen. Die autonome Optimierung erlaubt es, das System unter Berücksichtigung impliziter Abhängigkeiten abstraktionsebenenübergreifend zu optimieren.

Für alle ausgewählten und entwickelten Methoden wurde anschließend ein neuartiges Verknüpfungskonzept entworfen, das die Teillösungen zu einem vollständigen und adaptierbaren Management-Framework zusammenfügt. Dessen mehrstufiges Vorgehensmodell beschreibt, wie Domänenontologien modelliert, verknüpft und temporal ausgezeichnet werden, wie das Management-Modell definiert wird, wie Technologieadapter entwickelt werden, wie der automatisierte Regelkreis aufgebaut ist und auf einen Anwendungsfall adaptiert wird, und wie die autonome Optimierung umgesetzt wird. Die entwickelte Systemarchitektur beschreibt den Aufbau eines generischen Laufzeitsystems, mit dem ein wissensbasiertes, automatisiertes Management arbiträrer Anwendungsfälle umgesetzt werden kann.

Das Laufzeitsystem wurde auf Basis des OSGi-Komponenten-Frameworks vollständig implementiert. Seine Generizität und Modularität erlauben es, neue Anwendungsfälle mit geringem Aufwand zu adaptieren. Dabei sind keine Anpassungen der Kernkomponenten nötig: neue Ontologien und Technologieadapter werden als Erweiterungen hinzugeladen. Zusätzlich wurde für das Management-Modell eine domänenspezifische Sprache mit integrierter Entwicklungsumgebung umgesetzt, die es erlaubt, die Management-Logik in einer kompakten, RIF-artigen Syntax zu definieren. Das Framework steht damit an der Schwelle, eine vollständige Management-Plattform zu werden.

Das Framework wurde in zwei Fallstudien auf unterschiedliche Anwendungsfälle angewandt. Bei der Storage-Management-Fallstudie wurde gezeigt, dass das Framework in der Lage ist, in einer virtualisierten Storage-Umgebung auf betrieblicher Ebene definierte SLAs zu schützen, indem bei drohender Verletzung auf technischer Ebene automatisch virtuelle Disks gedrosselt wurden. Das Laufzeitsystem erzielte dabei für ein mittelgroßes Storage-System bei 150 Events pro Sekunde eine Antwort- und Reaktionszeit im niedrigen, zweistelligen Millisekundenbereich. Bei der Radar-Management-Fallstudie wurde gezeigt, dass das Framework in der Lage ist, kontinuierlich die SLA-Compliance von 568 Observationsbereichen zu prüfen und auf Basis einer stetigen Coverage-Detection, bei der zwischen 500 und 1000 Event pro Sekunde verarbeitet wurden, im Fehlerfall autonom in durchschnittlich einer Minute eine Rekonfigurationsagenda abzuleiten, die die Anzahl der Compliance-Verstöße deutlich reduziert.

Anhand der Fallstudien wurde nachgewiesen, dass das Framework alle in den Zielen definierten Anforderungen erfüllt. Mit ihm können unterschiedliche Domänen abstraktionsübergreifend zu einem einheitlichen Informationsmodell verknüpft werden, das zur Laufzeit die Grundlage einer in der Wissensbasis vorgehaltenen Systemgesamtsicht bildet. Die umzusetzende Management-Logik wird in einem einheitlichen Managment-Modell beschrieben, das direkten Bezug auf die Informationsmodellentitäten nimmt. Zur Laufzeit adaptiert der autonome Manager die Management-Logik, um eine kontextsensitive Event-Vorverarbeitung und -Abbildung umzusetzen, aus dem Systemmodell unbeobachtbare Zustände und Qualitätsindikatoren abzuleiten und auf Basis von Policies und Optimierungsverfahren ein automatisiertes Management umzusetzen. Die Generizität des Frameworks erlaubt es Administratoren, arbiträre Management-Anwendungsfälle in einem bisher nicht möglichen Maß zu automatisieren.

Somit konnte in dieser Arbeit erfolgreich gezeigt werden, wie man durch Verknüpfung von automatisierten IT-Management-Verfahren und Semantic-Web-Technologien den Anforderungen des modernen IT-Operations-Managements gerecht werden kann.

6.2 Ausblick

Die in dieser Arbeit entwickelten Methoden und Konzepte bieten eine Vielzahl von Ansatzpunkten für mögliche Erweiterungen, die in zukünftigen Arbeiten umgesetzt werden können.

Durch die Entwicklung neuer Domänenontologien und generischer Technologieadapter kann der Aufwand für die Integration mit neuen Anwendungsfällen weiter reduziert werden. Dabei profitiert der Ansatz besonders davon, dass Ontologien keinem bestimmten Modellierungsschema folgen müssen, sondern mit dem Fluents-Ansatz nachträglich erweitert und integriert werden können.

Die Management-Sprache könnte, wie in Abschnitt 4.2.1 (siehe Seite 86) beschrieben, um ein Anbindungskonzept für externe Dienste erweitert werden, sodass außerhalb der Wissensbasis liegende Datenquellen in den Verarbeitungsprozess einbezogen oder spezialisierte, hart-kodierte Algorithmen eingebunden werden können. Zudem könnte die Ausdrucksmächtigkeit der Sprache in Bezug auf temporale Zusammenhänge erweitert werden. So müssen beispielsweise momentan Event- und Fluent-Muster als komplexe Verknüpfung von Konjunktionen und Negationen ausgedrückt werden.

Einen großen Gewinn für das Framework würde die Integration von Machine-Learning-Verfahren darstellen. Sie könnten den Datenstrom und die Wissensbasis kontinuierlich überwachen, das Systemverhalten lernen und Komponentenzustände vorhersagen, die für ein proaktives Management genutzt werden können. Zudem könnten typische Muster in den manuellen Eingriffen der Administratoren gelernt und daraus neue Policies abgeleitet werden. Darüber hinaus könnte das Framework, wie in Abschnitt 5.2.3 (siehe Seite 163) beschrieben, um ein verteiltes Auswertungskonzept erweitert werden, mit dem insbesondere die Laufzeit der autonomen Optimierung durch eine parallele Auswertung auf mehreren Knoten reduziert werden könnte.

Literaturverzeichnis

[1] James F. Allen. „Maintaining Knowledge About Temporal Intervals“. In: *Commun. ACM* 26.11 (Nov. 1983), S. 832–843. ISSN: 0001-0782 (siehe S. 39, 40).

[2] Darko Anicic, Paul Fodor, Sebastian Rudolph und Nenad Stojanovic. „EP-SPARQL: a unified language for event processing and stream reasoning“. In: *Proceedings of the 20th international conference on World wide web*. ACM. 2011, S. 635–644 (siehe S. 62).

[3] Arvind Arasu, Shivnath Babu und Jennifer Widom. *The CQL Continuous Query Language: Semantic Foundations and Query Execution*. Technical Report 2003-67. Stanford InfoLab, 2003 (siehe S. 62).

[4] Arvind Arasu, Shivnath Babu und Jennifer Widom. „The CQL continuous query language: semantic foundations and query execution“. In: *The VLDB Journal—The International Journal on Very Large Data Bases* 15.2 (2006), S. 121–142 (siehe S. 40, 41).

[5] AXELOS. *ITIL service lifecycle publication suite 2nd ed.* TSO (The Stationery Office), Juni 2011. ISBN: 9780113313235 (siehe S. 11).

[6] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003 (siehe S. 15, 22).

[7] Franz Baader, Sebastian Brandt und Carsten Lutz. „Pushing the EL envelope further“. In: *Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*. 2008 (siehe S. 29).

[8] Franz Baader und Ulrike Sattler. „An Overview of Tableau Algorithms for Description Logics“. English. In: *Studia Logica* 69.1 (2001), S. 5–40. ISSN: 0039-3215 (siehe S. 28, 55).

[9] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri und Michael Grossniklaus. „An execution environment for C-SPARQL queries“. In: *Proceedings of the 13th International Conference on Extending Database Technology*. ACM. 2010, S. 441–452 (siehe S. 62).

[10] DavideFrancesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle und Michael Grossniklaus. „Incremental Reasoning on Streams and Rich Background Knowledge“. English. In: *The Semantic Web: Research and Applications*. Hrsg. von Lora Aroyo, Grigoris Antoniou, Eero Hyvönen u. a. Bd. 6088. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, S. 1–15. ISBN: 978-3-642-13485-2 (siehe S. 62).

[11] Henk Barendregt, Wil Dekkers und Richard Statman. *Lambda calculus with types*. Cambridge University Press, 2013 (siehe S. 90).

[12] David Beckett, Tim Berners-Lee, Eric Prud'hommeaux und Gavin Carothers. *RDF 1.1 Turtle*. Feb. 2014. URL: http://www.w3.org/TR/turtle/ (siehe S. 21).

[13] Jan Benjamin, Pim Borst, Hans Akkermans und Bob Wielinga. „Ontology construction for technical domains". In: *Advances in Knowledge Acquisition*. Springer, 1996, S. 98–114 (siehe S. 15).

[14] Tim Berners-Lee, Roy T. Fielding und Larry Masinter. *Uniform Resource Identifier (URI): Generic Syntax*. STD 66. http://www.rfc-editor.org/rfc/rfc3986.txt. RFC Editor, Jan. 2005 (siehe S. 16).

[15] Tim Berners-Lee, Larry Masinter und Mark McCahill. *Uniform Resource Locators (URL)*. RFC 1738. http://www.rfc-editor.org/rfc/rfc1738.txt. RFC Editor, Dez. 1994 (siehe S. 16).

[16] Leonora Bianchi, Marco Dorigo, Luca Maria Gambardella und Walter J Gutjahr. „A survey on metaheuristics for stochastic combinatorial optimization". In: *Natural Computing: an international journal* 8.2 (2009), S. 239–287 (siehe S. 105).

[17] Christian Blum und Andrea Roli. „Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison". In: *ACM Comput. Surv.* 35.3 (Sep. 2003), S. 268–308. ISSN: 0360-0300 (siehe S. 105).

[18] Harold Boley, Gary Hallmark, Michael Kifer u. a. *RIF Core Dialect (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/2013/REC-rif-core-20130205/ (siehe S. 36).

[19] Harold Boley und Michael Kifer. *RIF Basic Logic Dialect (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/2013/REC-rif-bld-20130205/ (siehe S. 37, 116).

[20] Harold Boley und Michael Kifer. *RIF Framework for Logic Dialects (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/2013/REC-rif-fld-20130205/ (siehe S. 38, 97, 116).

[21] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt und M. Scott Marshall. „Graph Drawing: 9th International Symposium, GD 2001 Vienna, Austria, September 23–26, 2001 Revised Papers". In: Hrsg. von Petra Mutzel, Michael Jünger und Sebastian Leipert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002. Kap. GraphML Progress Report Structural Layer Proposal, S. 501–512. ISBN: 978-3-540-45848-7 (siehe S. 15).

[22] Dan Brickley. *W3C Semantic Web Interest Group: Basic Geo (WGS84 lat/long) Vocabulary*. Jan. 2003. URL: https://www.w3.org/2003/01/geo/ (siehe S. 50).

[23] Dan Brickley und R.V. Guha. *RDF Schema 1.1*. Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf-schema-20140225 (siehe S. 19, 52).

[24] Dan Brickley und Libby Miller. *FOAF Vocabulary Specification 0.99*. Jan. 2014. URL: http://xmlns.com/foaf/spec/20140114.html (siehe S. 50).

[25] Jos de Bruijn und Chris Welty. *RIF RDF and OWL Compatibility (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/rif-rdf-owl/ (siehe S. 91).

[26] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini und Riccardo Rosati. „Tractable reasoning and efficient query answering in description logics: The DL-Lite family“. In: *Journal of Automated reasoning* 39.3 (2007), S. 385–429 (siehe S. 29).

[27] Namyoun Choi, Il-Yeol Song und Hyoil Han. „A survey on ontology mapping“. In: *ACM Sigmod Record* 35.3 (2006), S. 34–41 (siehe S. 110).

[28] Kendall Grant Clark. *RDF Data Access Use Cases and Requirements*. März 2005. URL: http://www.w3.org/TR/2005/WD-rdf-dawg-uc-20050325/ (siehe S. 30).

[29] Owen Conlan, John Keeney, Cormac Hampson und Fionán Peter Williams. „Towards non-expert users monitoring networks and services through semantically enhanced visualizations“. In: *Network and Service Management (CNSM), 2010 International Conference on*. IEEE. 2010, S. 406–409 (siehe S. 46).

[30] Frédéric Cuppens, Nora Cuppens-Boulahia und Alexandre Miège. „Inheritance hierarchies in the Or-BAC Model and application in a network environment“. In: *Proc. Foundations of Computer Security (FCS04)* (2004), S. 41–60 (siehe S. 45, 50).

[31] Frédéric Cuppens und Alexandre Miège. „Modelling contexts in the Or-BAC model“. In: *Computer Security Applications Conference, 2003. Proceedings. 19th Annual*. IEEE. 2003, S. 416–425 (siehe S. 45, 50).

[32] Nora Cuppens-Boulahia, Frederic Cuppens, Fabien Autrel und Herve Debar. „An ontology-based approach to react to network attacks“. In: *International Journal of Information and Computer Security* 3.3 (2009), S. 280–305 (siehe S. 44, 50, 60, 64, 67, 69).

[33] Richard Cyganiak, David Wood und Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. Feb. 2014. URL: http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/ (siehe S. 4, 15).

[34] JE López De Vergara, Víctor A Villagrá, Juan I Asensio und Julio Berrocal. „Ontologies: Giving semantics to network management models“. In: *Network, IEEE* 17.3 (2003), S. 15–21 (siehe S. 56).

[35] JE López De Vergara, Víctor A Villagrá und Julio Berrocal. „Applying the Web Ontology Language to management information definitions“. In: *IEEE Communications Magazine* 42.7 (2004), S. 68–74 (siehe S. 109).

[36] Jorge E López De Vergara, Antonio Guerrero, Víctor A Villagrá und Julio Berrocal. „Ontology-based network management: study cases and lessons learned“. In: *Journal of Network and Systems Management* 17.3 (2009), S. 234–254 (siehe S. 44, 55, 59, 64, 75, 77).

[37] H. Debar, D. Curry und B. Feinstein. *The Intrusion Detection Message Exchange Format (IDMEF)*. RFC 4765. http://www.rfc-editor.org/rfc/rfc4765.txt. RFC Editor, März 2007 (siehe S. 45).

[38] M. Duerst und M. Suignard. *Internationalized Resource Identifiers (IRIs)*. RFC 3987. http://www.rfc-editor.org/rfc/rfc3987.txt. RFC Editor, Jan. 2005 (siehe S. 16).

[39] Opher Etzion und Peter Niblett. *Event Processing in Action*. 1st. Greenwich, CT, USA: Manning Publications Co., 2010. ISBN: 9781935182214 (siehe S. 39).

[40] Liam Fallon und Declan O'Sullivan. „Aesop: A semantic system for autonomic management of end-user service quality". In: *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*. IEEE. 2013, S. 716–719 (siehe S. 45, 65, 67, 69).

[41] Liam Fallon und Declan O'Sullivan. „The Aesop Approach for Semantic-based End-User Service Optimization". In: *Network and Service Management, IEEE Transactions on* 11.2 (2014), S. 220–234 (siehe S. 45, 50, 52, 55, 73).

[42] Liam Fallon und Declan O'Sullivan. „Using a semantic knowledge base for communication service quality management in home area networks". In: *Network Operations and Management Symposium (NOMS), 2012 IEEE*. IEEE. 2012, S. 43–51 (siehe S. 45, 50).

[43] Adam Farquhar, Richard Fikes und James Rice. „The ontolingua server: A tool for collaborative ontology construction". In: *International journal of human-computer studies* 46.6 (1997), S. 707–727 (siehe S. 15).

[44] Joel Farrell und Holger Lausen. *Semantic Annotations for WSDL and XML Schema*. Aug. 2007. URL: `https://www.w3.org/TR/sawsdl/` (siehe S. 65).

[45] Dieter Fensel, Frank Van Harmelen, Ian Horrocks, Deborah L McGuinness und Peter F Patel-Schneider. „OIL: An ontology infrastructure for the semantic web". In: *IEEE intelligent systems* 16.2 (2001), S. 38–45 (siehe S. 22).

[46] Charles L Forgy. „Rete: A fast algorithm for the many pattern/many object pattern match problem". In: *Artificial intelligence* 19.1 (1982), S. 17–37 (siehe S. 34, 35).

[47] Anthony Philip French. *Special relativity*. CRC Press, 1968. ISBN: 0-7487-6422-4 (siehe S. 38).

[48] Herve Gallaire, Jack Minker und Jean-Marie Nicolas. „Logic and databases: A deductive approach". In: *ACM Computing Surveys (CSUR)* 16.2 (1984), S. 153–185 (siehe S. 36).

[49] Jean Gallier. „SLD-Resolution and Logic Programming". In: *Logic for Computer Science: Foundations of Automatic Theorem Proving*. 2003 (siehe S. 35).

[50] Fabien Gandon und Guus Schreiber. *RDF 1.1 XML Syntax*. Feb. 2014. URL: `http://www.w3.org/TR/rdf-syntax-grammar/` (siehe S. 21).

[51] Aldo Gangemi und Valentina Presutti. „Ontology design patterns". In: *Handbook on ontologies*. Springer, 2009, S. 221–243 (siehe S. 109).

[52] John H Gennari, Mark A Musen, Ray W Fergerson u. a. „The evolution of Protégé: an environment for knowledge-based systems development". In: *International Journal of Human-computer studies* 58.1 (2003), S. 89–123 (siehe S. 110).

[53] Fred Glover. „Tabu search-part I". In: *ORSA Journal on computing* 1.3 (1989), S. 190–206 (siehe S. 107).

[54] Fred Glover. „Tabu search—part II“. In: *ORSA Journal on computing* 2.1 (1990), S. 4–32 (siehe S. 107).

[55] Asunción Gómez-Pérez und Oscar Corcho. „Ontology languages for the semantic web“. In: *Intelligent Systems, IEEE* 17.1 (2002), S. 54–60 (siehe S. 56).

[56] Benjamin N Grosof, Ian Horrocks, Raphael Volz und Stefan Decker. „Description logic programs: Combining logic programs with description logic“. In: *Proceedings of the 12th international conference on World Wide Web*. ACM. 2003, S. 48–57 (siehe S. 29).

[57] W3C OWL Working Group. *OWL 2 Web Ontology Language Document Overview (Second Edition)*. Dez. 2012. URL: https://www.w3.org/TR/owl2-overview/ (siehe S. 4, 22, 56).

[58] Antonio Guerrero, VíctorA. Villagrá, JorgeE.López de Vergara und Julio Berrocal. „Ontology-Based Integration of Management Behaviour and Information Definitions Using SWRL and OWL“. English. In: *Ambient Networks*. Hrsg. von Jürgen Schönwälder und Joan Serrat. Bd. 3775. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2005, S. 12–23. ISBN: 978-3-540-29388-0 (siehe S. 4, 44, 66).

[59] Antonio Guerrero, VíctorA. Villagrá, JorgeE.López de Vergara, Alfonso Sánchez-Macián und Julio Berrocal. „Ontology-Based Policy Refinement Using SWRL Rules for Management Information Definitions in OWL“. English. In: *Large Scale Management of Distributed Systems*. Hrsg. von Radu State, Sven van der Meer, Declan O'Sullivan und Tom Pfeifer. Bd. 4269. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, S. 227–232. ISBN: 978-3-540-47659-7 (siehe S. 44, 50, 55, 58, 68, 74).

[60] Cormac Hampson und Owen Conlan. „Supporting personalized information exploration through subjective expert-created semantic attributes“. In: *Semantic Computing, 2009. ICSC'09. IEEE International Conference on*. IEEE. 2009, S. 384–389 (siehe S. 46).

[61] Steve Harris und Andy Seaborne. *SPARQL 1.1 Query Language*. März 2013. URL: http://www.w3.org/TR/owl-features/ (siehe S. 30, 31).

[62] Souleiman Hasan, Sean O'Riain und Edward Curry. „Towards unified and native enrichment in event processing systems“. In: *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM. 2013, S. 171–182 (siehe S. 61).

[63] Tom Heath und Christian Bizer. „Linked data: Evolving the web into a global data space“. In: *Synthesis lectures on the semantic web: theory and technology* 1.1 (2011), S. 1–136 (siehe S. 18).

[64] Heinz-Gerd Hegering, Sebastian Abeck und Bernhard Neumair. „Integriertes Management vernetzter Systeme“. In: *Auflage Heidelberg: dpunkt-Verlag für digitale Technologie, Hüttig GmbH* (1999) (siehe S. 9, 12).

[65] Ralf Heidger. „The Phoenix White Paper“. 2010 (siehe S. 153).

[66] James Hendler und Deborah L McGuinness. „The DARPA agent markup language“. In: *IEEE Intelligent systems* 15.6 (2000), S. 67–73 (siehe S. 22).

[67] Pascal Hitzler, Markus Krotzsch und Sebastian Rudolph. *Foundations of semantic web technologies*. CRC Press, 2009. ISBN: 978-1-420-09050-5 (siehe S. 25).

[68] Ian Horrocks u. a. „DAML+OIL: A Description Logic for the Semantic Web“. In: *IEEE Data Eng. Bull.* 25.1 (2002), S. 4–9 (siehe S. 22).

[69] Ian Horrocks und Peter F Patel-Schneider. „Reducing OWL entailment to description logic satisfiability“. In: *The semantic web-ISWC 2003*. Springer, 2003, S. 17–29 (siehe S. 24, 26).

[70] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley u. a. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Mai 2004. URL: `https://www.w3.org/Submission/SWRL/` (siehe S. 44).

[71] Ian Horrocks, Ulrike Sattler und Stephan Tobies. „Practical reasoning for expressive description logics“. In: *Logic for Programming and Automated Reasoning*. Springer. 1999, S. 161–180 (siehe S. 27).

[72] IBM. *An Architectural Blueprint for Autonomic Computing*. Techn. Ber. IBM, Juni 2005 (siehe S. 5, 13).

[73] Ralph E Johnson und Brian Foote. „Designing reusable classes“. In: *Journal of object-oriented programming* 1.2 (1988), S. 22–35 (siehe S. 108).

[74] Rudolph Emil Kalman. „A new approach to linear filtering and prediction problems“. In: *Journal of basic Engineering* 82.1 (1960), S. 35–45 (siehe S. 153).

[75] John Keeney, Owen Conlan, Viliam Holub u. a. „A semantic monitoring and management framework for end-to-end services“. In: *Integrated Network Management (IM), 2011 IFIP/IEEE International Symposium on*. IEEE. 2011, S. 658–661 (siehe S. 46, 51, 60, 65, 67, 69).

[76] John Keeney, David Lewis, Declan O'Sullivan u. a. „Runtime semantic interoperability for gathering ontology-based network context“. In: *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*. IEEE. 2006, S. 56–65 (siehe S. 64).

[77] Hans-Ulrich Krieger. „A detailed comparison of seven approaches for the annotation of time-dependent factual knowledge in RDF and OWL“. In: *Proceedings 10th Joint ISO-ACL SIGSEM Workshop on Interoperable Semantic Annotation*. 2014, S. 1 (siehe S. 53, 73).

[78] Hans-Ulrich Krieger. „A General Methodology for Equipping Ontologies with Time.“ In: *LREC*. Citeseer. 2010 (siehe S. 53).

[79] Carsten Lutz. *An improved NExpTime-hardness result for description logic $\mathcal{ALC}$ extended with inverse roles, nominals, and counting*. LTCS-Report 05-05. Dresden, Germany: Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, 2005 (siehe S. 28, 55).

[80] Ashok Malhotra, Jim Melton, Norman Walsh und Michael Kay. *XQuery 1.0 and XPath 2.0 Functions and Operators (Second Edition)*. Apr. 2010. URL: `https://www.w3.org/TR/xpath-functions/` (siehe S. 31).

[81] David Martin, Mark Burstein, Jerry Hobbs u. a. *OWL-S: Semantic Markup for Web Services*. Nov. 2004. URL: https://www.w3.org/Submission/OWL-S/ (siehe S. 44, 68, 90).

[82] Deborah L. McGuinness und Frank van Harmelen. *OWL Web Ontology Language Overview*. Feb. 2004. URL: http://www.w3.org/TR/owl-features/ (siehe S. 22).

[83] Fabian Meyer. „Analyse von Event Streams im Kontext semantischer Systemmodelle“. In: *Informatiktage 2011*. Bd. S-10. Gesellschaft für Informatik e.V. (GI). Ahrstr. 45, 53175 Bonn, März 2011, S. 159–162. ISBN: 978-3-88579-444-8 (siehe S. 252).

[84] Fabian Meyer. „Kombination von Modellen zur Systemanalyse im Selbstmanagement-Kontext“. In: Electronic Communications of the EASST 37 (März 2011). ISSN: 1863-2122 (siehe S. 252).

[85] Fabian Meyer. „Ontologie-basiertes Monitoring von IT-Systemen“. In: *44. Jahrestagung der Gesellschaft für Informatik, Workshop Management komplexer IT-Systeme und Anwendungen*. Hrsg. von Erhard Plödereder, Lars Grunske, Erik Schneider und Dominik Ull. Bd. P-232. Lecture Notes in Informatics (LNI) - Proceedings. Bonn: Gesellschaft für Informatik, Sep. 2014, S. 861–872. ISBN: 978-3-88579-626-8 (siehe S. 252).

[86] Fabian Meyer und Reinhold Kroeger. „A framework for autonomic, ontology-based IT management“. In: *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, Nov. 2015, S. 78–84. ISBN: 978-3-9018-8277-7 (siehe S. 252).

[87] Fabian Meyer, Reinhold Kröger und Morris Milekovic. „An approach for knowledge-based IT management of air traffic control systems“. In: *2013 9th International Conference on Network and Service Management (CNSM)*. IEEE, Okt. 2013, S. 345–349. ISBN: 978-3-901882-53-1 (siehe S. 252).

[88] R. Moats. *URN Syntax*. RFC 2141. http://www.rfc-editor.org/rfc/rfc2141.txt. RFC Editor, Mai 1997 (siehe S. 16).

[89] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks u. a. *OWL 2 Web Ontology Language Profiles (Second Edition)*. Dez. 2012. URL: https://www.w3.org/TR/owl2-profiles/ (siehe S. 57, 99, 103).

[90] Boris Motik, Ulrike Sattler und Rudi Studer. „Query answering for OWL-DL with rules“. In: *Web Semantics: Science, Services and Agents on the World Wide Web* 3.1 (2005), S. 41–60 (siehe S. 77).

[91] San Murugesan. „Harnessing green IT: Principles and practices“. In: *IT professional* 10.1 (2008), S. 24–33 (siehe S. 140).

[92] N. Nahum. „Storage virtualization in a storage area network“. US Patent 6,898,670. Mai 2005 (siehe S. 137).

[93] Natasha Noy, Alan Rector, Pat Hayes und Chris Welty. „Defining n-ary relations on the semantic web“. In: *W3C Working Group Note* 12 (2006), S. 4 (siehe S. 53).

[94] Peter F. Patel-Schneider, Ian Horrocks und Bernardo Cuenca Grau. *OWL 1.1 Web Ontology Language Overview*. Dez. 2006. URL: http://www.w3.org/Submission/owl11-overview/ (siehe S. 22).

[95] David Peterson, Shudi Gao, Ashok Malhotra, C. M. Sperberg-McQueen und Henry S. Thompson. *W3C XML Schema Definition Language (XSD) 1.1 Part 2: Datatypes*. Apr. 2012. URL: http://www.w3.org/TR/xmlschema11-2/ (siehe S. 16).

[96] A. Phillips und M. Davis. *Tags for Identifying Languages*. BCP 47. RFC Editor, Sep. 2009 (siehe S. 16).

[97] Axel Polleres, Harold Boley und Michael Kifer. *RIF Datatypes and Built-Ins 1.0 (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/2013/REC-rif-dtb-20130205/ (siehe S. 36, 89).

[98] Annie Ibrahim Rana und Brendan Jennings. „Semantic uplift of monitoring data to select policies to manage home area networks“. In: *Advanced Information Networking and Applications (AINA), 2012 IEEE 26th International Conference on*. IEEE. 2012, S. 368–375 (siehe S. 45, 51, 60, 65, 67, 69).

[99] Annie Ibrahim Rana, Brendan Jennings, MO Foghlu und Sven van der Meer. „Autonomic policy-based HAN traffic classification using augmented meta model for policy translation“. In: *Wireless and Optical Communications Networks (WOCN), 2011 Eighth International Conference on*. IEEE. 2011, S. 1–8 (siehe S. 45).

[100] Annie Ibrahim Rana und Mícheál O Foghlu. „New role of policy-based management in home area networks-concepts, constraints and challenges“. In: *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*. IEEE. 2009, S. 1–6 (siehe S. 45).

[101] Olaf Resch. *Einführung in das IT-Management: Grundlagen, Umsetzung, Best Practice*. Erich Schmidt Verlag GmbH und Co, Sep. 2009. ISBN: 978-3503120604 (siehe S. 9, 10).

[102] Christian de Sainte Marie, Gary Hallmark und Adrian Paschke. *RIF Production Rule Dialect (Second Edition)*. Feb. 2013. URL: https://www.w3.org/TR/2013/REC-rif-prd-20130205/ (siehe S. 37, 38).

[103] Andrea Schaerf. „Reasoning with individuals in concept languages“. In: *Data and Knowledge Engineering* 13.2 (1994), S. 141–176 (siehe S. 28, 55).

[104] Pavel Shvaiko und Jérôme Euzenat. „Ontology matching: state of the art and future challenges“. In: *Knowledge and Data Engineering, IEEE Transactions on* 25.1 (2013), S. 158–176 (siehe S. 49).

[105] International Organization for Standardization. *ISO/IEC 20000*. Apr. 2011. URL: http://www.iso.org/iso/catalogue_detail?csnumber=51986 (siehe S. 11).

[106] J. Strassner. „DEN-ng: achieving business-driven network management“. In: *Network Operations and Management Symposium, 2002. NOMS 2002. 2002 IEEE/IFIP*. 2002, S. 753–766 (siehe S. 51).

[107] John Strassner, Nazim Agoulmine und Elyes Lehtihet. „Focale: A novel autonomic networking architecture“. In: *Latin American Autonomic Computing Symposium (LAACS)* (2006) (siehe S. 46, 67, 69).

[108] John Strassner, Sven van der Meer, Declan O'Sullivan und Simon Dobson. „The use of context-aware policies and ontologies to facilitate business-aware network management“. In: *Journal of Network and Systems Management* 17.3 (2009), S. 255–284 (siehe S. 47, 51, 64).

[109] John Strassner, Declan O'Sullivan und David Lewis. „Ontologies in the Engineering of Management and Autonomic Systems: A Reality Check“. English. In: *Journal of Network and Systems Management* 15.1 (2007), S. 5–11. ISSN: 1064-7570 (siehe S. 47).

[110] Rudi Studer, V Richard Benjamins und Dieter Fensel. „Knowledge engineering: principles and methods“. In: *Data & knowledge engineering* 25.1 (1998), S. 161–197 (siehe S. 4, 14, 15).

[111] Andreas Textor. „A CIM-based Ontology for Semantic IT-Management“. In: *5th International DMTF Academic Alliance Workshop on Systems and Virtualization Management: Standards and the Cloud*. Paris, France, Okt. 2011 (siehe S. 56).

[112] Andreas Textor, Fabian Meyer und Reinhold Kröger. „Automated IT Management using Ontologies“. In: *International Journal on Advances in Intelligent Systems* 5.3/4 (Dez. 2012), S. 291–301. ISSN: 1942-2679 (siehe S. 252).

[113] Andreas Textor, Fabian Meyer und Reinhold Kröger. „Semantic Processing in IT Management“. In: *Proceedings of the Fifth International Conference on Advances in Semantic Processing (SEMAPRO)*. Hrsg. von Pascal Lorenz und Eckhard Ammann. Lisbon, Portugal: IARIA, Nov. 2011. ISBN: 978-1-61208-013-0 (siehe S. 252).

[114] Andreas Textor, Fabian Meyer, Marcus Thoss u. a. „An Architecture for Semantically Enriched Data Stream Mining“. In: *Proceedings of the First International Conference on Data Analytics*. Hrsg. von Sandjai Bhulai, Joseph Zernik und Petre Dini. Barcelona, Spain: IARIA, Sep. 2012. ISBN: 978-1-61208-242-4 (siehe S. 252).

[115] Andreas Textor und Thomas Sikora. „An Ontology-Based Architecture for Storage Management“. In: *Proceedings of the 7th International Workshop on Formal Ontologies Meet Industry*. Hrsg. von Roberta Cuel und Robert Young. Bd. 225. Lecture Notes in Business Information Processing. Berlin: Springer, Aug. 2015, S. 63–74. ISBN: 978-3-319-21544-0 (siehe S. 139).

[116] Kia Teymourian und Adrian Paschke. „Plan-Based Semantic Enrichment of Event Streams“. In: *The Semantic Web: Trends and Challenges*. Springer, 2014, S. 21–35 (siehe S. 61).

[117] Kia Teymourian, Malte Rohde und Adrian Paschke. „Fusion of background knowledge and streams of events“. In: *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*. ACM. 2012, S. 302–313 (siehe S. 61).

[118] Kia Teymourian, Olga Streibel, Adrian Paschke, Rehab Alnemr und Christoph Meinel. „Towards semantic event-driven systems“. In: *New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on*. IEEE. 2009, S. 1–6 (siehe S. 61).

[119] Stephan Tobies. „Complexity results and practical algorithms for logics in knowledge representation“. In: *arXiv preprint cs/0106031* (2001) (siehe S. 28).

[120] Stephan Tobies. „The complexity of reasoning with cardinality restrictions and nominals in expressive Description Logics“. In: *Journal of Artificial Intelligence Research* 12 (2000), S. 199–217 (siehe S. 55).

[121] Jorge E. López De Vergara, Víctor A. Villagrá und Julio Berrocal. „Applying the Web ontology language to management information definitions“. In: *IEEE Communications Magazine* 42 (2004), S. 68–74 (siehe S. 4, 56).

[122] Jorge E López de Vergara, Víctor A Villagrá, Carlos Fadón u. a. „An autonomic approach to offer services in OSGi-based home gateways“. In: *Computer Communications* 31.13 (2008), S. 3049–3058 (siehe S. 44).

[123] Dinesh C Verma. „Simplifying network administration using policy-based management“. In: *Network, IEEE* 16.2 (2002), S. 20–26 (siehe S. 44, 68).

[124] Chris Welty, Richard Fikes und Selene Makarios. „A reusable ontology for fluents in OWL“. In: *FOIS*. Bd. 150. 2006, S. 226–236 (siehe S. 53).

[125] Andreas Winter. „Exchanging graphs with GXL“. In: *Graph drawing*. Springer. 2001, S. 485–500 (siehe S. 15).

[126] Debao Xiao und Hui Xu. „An integration of ontology-based and policy-based network management for automation“. In: *Computational Intelligence for Modelling, Control and Automation, 2006 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*. IEEE. 2006, S. 27–27 (siehe S. 45, 51, 59, 64, 67, 68).

[127] Hui Xu und Debao Xiao. „A common ontology-based intelligent configuration management model for ip network devices“. In: *Innovative Computing, Information and Control, 2006. ICICIC'06. First International Conference on*. Bd. 1. IEEE. 2006, S. 385–388 (siehe S. 46).

[128] Donghui Zhang, Dimitrios Gunopulos, Vassilis J Tsotras und Bernhard Seeger. „Temporal aggregation over data streams using multiple granularities“. In: *International Conference on Extending Database Technology*. Springer. 2002, S. 646–663 (siehe S. 179).

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang

A

A.1 Grafische OWL-Notation

Klasse

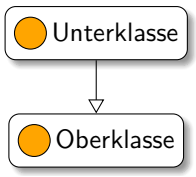
Individuum

Abstrakte Rolle

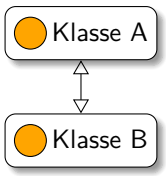
Konkrete Rolle

Datentyp

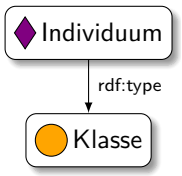
Literal



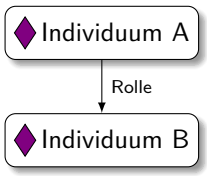
Klassen- oder Rollenhierarchie



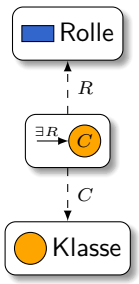
Klassen- oder Rollenäquivalenz



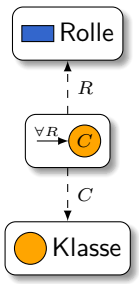
Klassenzuweisung



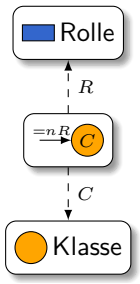
Rollenzuweisung



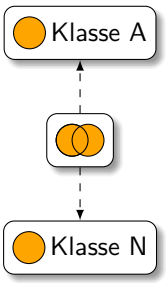
Existenz-einschränkung über Rolle R auf Klasse C



Universal-einschränkung über Rolle R auf Klasse C



Kardinalitäts-einschränkung über Rolle R auf Klasse C mit Kardinalität n und Operator $=$, $<$, $>$, $\leqslant$ oder $\geqslant$



Klassenvereinigung

A.2 Framework

A.2.1 4D-Fluents-Ontologie

```
@prefix : <http://www.ibm.com/4dFluents#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.ibm.com/4dFluents> a owl:Ontology .

# http://www.ibm.com/4dFluents#hasTemporalPart
:hasTemporalPart a owl:ObjectProperty ;
  a owl:InverseFunctionalProperty ;
  owl:inverseOf :temporalPartOf ;
  rdfs:domain [
    a owl:Class ;
    owl:complementOf :TimeInterval
  ] ;
  rdfs:range :TemporalPart .

# http://www.ibm.com/4dFluents#temporalExtent
:temporalExtent a owl:ObjectProperty , owl:FunctionalProperty ;
  rdfs:domain :TemporalPart ;
  rdfs:range :TimeInterval .

# http://www.ibm.com/4dFluents#temporalObjectProperty
:temporalObjectProperty a owl:ObjectProperty ;
  rdfs:domain :TemporalPart ;
  rdfs:range :TemporalPart .

# http://www.ibm.com/4dFluents#temporalPartOf
:temporalPartOf a owl:ObjectProperty , owl:FunctionalProperty ;
```

```
    rdfs:domain :TemporalPart ;
    rdfs:range [
      a owl:Class ;
      owl:complementOf :TimeInterval
    ] .

# http://www.ibm.com/4dFluents#hasEnd
:hasEnd a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :TimeInterval ;
    rdfs:range xsd:dateTime .

# http://www.ibm.com/4dFluents#hasStart
:hasStart a owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:domain :TimeInterval ;
    rdfs:range xsd:dateTime .

# http://www.ibm.com/4dFluents#temporalDataProperty
:temporalDataProperty a owl:DatatypeProperty ;
    rdfs:domain :TemporalPart .

# http://www.ibm.com/4dFluents#TemporalClass
:TemporalClass a owl:Class .

# http://www.ibm.com/4dFluents#TemporalPart
:TemporalPart a owl:Class ;
    owl:disjointWith :TimeInterval .

# http://www.ibm.com/4dFluents#TimeInterval
:TimeInterval a owl:Class .
```

A.2.2 Domain Interface Ontology (DIO)-Ontologie

```
@prefix : <http://wwwvs.cs.hs-rm.de/obmf/DIO#> .
@prefix dio: <http://wwwvs.cs.hs-rm.de/obmf/dio#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://wwwvs.cs.hs-rm.de/obmf/DIO> a owl:Ontology .

# http://wwwvs.cs.hs-rm.de/obmf/dio#timestamp
dio:timestamp a owl:DatatypeProperty .

# http://wwwvs.cs.hs-rm.de/obmf/dio#Action
```

```
dio:Action a owl:Class ;
  rdfs:subClassOf dio:InterfaceType .

# http://wwwvs.cs.hs-rm.de/obmf/dio#Event
dio:Event a owl:Class ;
  rdfs:subClassOf dio:InterfaceType , [
    a owl:Restriction ;
    owl:onProperty dio:timestamp ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onDataRange xsd:dateTime
  ] .

# http://wwwvs.cs.hs-rm.de/obmf/dio#InterfaceType
dio:InterfaceType a owl:Class .
```

A.2.3 Ontology-based Management Language (OntML)-Ontologie

```
@prefix : <http://www.hs_rm.de/cs/vs/ontml#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xml: <http://www.w3.org/XML/1998/namespace/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

<http://www.hs_rm.de/cs/vs/OntML> a owl:Ontology ;
  owl:imports <http://wwwvs.cs.hs-rm.de/obmf/DIO> .

# http://www.hs_rm.de/cs/vs/ontml#restrictedTo
:restrictedTo a owl:AnnotationProperty .

# http://www.w3.org/2001/XMLSchema#duration
xsd:duration a rdfs:Datatype .

# http://www.hs_rm.de/cs/vs/ontml#applies
:applies a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#comprehendsOn
:comprehendsOn a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#consistsOf
:consistsOf a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#dependsOn
:dependsOn a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#extends
```

```
:extends a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#groupsOn
:groupsOn a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasCharacteristic
:hasCharacteristic a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasClass
:hasClass a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasCondition
:hasCondition a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasConsequence
:hasConsequence a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasContext
:hasContext a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasLeftHand
:hasLeftHand a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasOperand
:hasOperand a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasOperator
:hasOperator a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasRightHand
:hasRightHand a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasRole
:hasRole a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#hasSubFormula
:hasSubFormula a owl:ObjectProperty ;
  rdfs:range :ConditionFormula .

# http://www.hs_rm.de/cs/vs/ontml#hasVariable
:hasVariable a owl:ObjectProperty ;
  rdfs:subPropertyOf owl:topObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#referesTo
:referesTo a owl:ObjectProperty .

# http://www.hs_rm.de/cs/vs/ontml#target
:target a owl:ObjectProperty ;
```

```
78    rdfs:subPropertyOf owl:topObjectProperty .
79
80  # http://www.hs_rm.de/cs/vs/ontml#to
81  :to a owl:ObjectProperty .
82
83  # http://www.hs_rm.de/cs/vs/ontml#underCondition
84  :underCondition a owl:ObjectProperty .
85
86  # http://www.hs_rm.de/cs/vs/ontml#underRestriction
87  :underRestriction a owl:ObjectProperty .
88
89  # http://www.hs_rm.de/cs/vs/ontml#withAttribute
90  :withAttribute a owl:ObjectProperty .
91
92  # http://www.hs_rm.de/cs/vs/ontml#withFunction
93  :withFunction a owl:ObjectProperty .
94
95  # http://www.hs_rm.de/cs/vs/ontml#withValue
96  :withValue a owl:ObjectProperty .
97
98  # http://www.hs_rm.de/cs/vs/ontml#duration
99  :duration a owl:DatatypeProperty .
100
101 # http://www.hs_rm.de/cs/vs/ontml#name
102 :name a owl:DatatypeProperty .
103
104 # http://www.hs_rm.de/cs/vs/ontml#value
105 :value a owl:DatatypeProperty .
106
107 # http://www.hs_rm.de/cs/vs/ontml#Action
108 :Action a owl:Class ;
109   owl:disjointUnionOf (:ActionBlock :AtomicAction :CompoundAction ) .
110
111 # http://www.hs_rm.de/cs/vs/ontml#ActionBlock
112 :ActionBlock a owl:Class ;
113   rdfs:subClassOf :Action , [
114     a owl:Restriction ;
115           owl:onProperty :hasLeftHand ;
116           owl:allValuesFrom :Action
117   ] , [
118     a owl:Restriction ;
119     owl:onProperty :hasRightHand ;
120     owl:allValuesFrom :Action
121   ], [
122     a owl:Restriction ;
123     owl:onProperty :hasLeftHand ;
124     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
125     owl:onClass :Action
```

```
126     ], [
127       a owl:Restriction ;
128       owl:onProperty :hasRightHand ;
129       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
130       owl:onClass :Action
131     ].
132
133   # http://www.hs_rm.de/cs/vs/ontml#ActionInterpretation
134   :ActionInterpretation a owl:Class ;
135     rdfs:subClassOf :ProductionRule .
136
137   # http://www.hs_rm.de/cs/vs/ontml#ActionSuggestion
138   :ActionSuggestion a owl:Class ;
139     rdfs:subClassOf :ImplicationRule .
140
141   # http://www.hs_rm.de/cs/vs/ontml#AggregationTerm
142   :AggregationTerm a owl:Class ;
143     rdfs:subClassOf :BaseTerm , [
144       a owl:Restriction ;
145       owl:onProperty :comprehendsOn ;
146       owl:allValuesFrom :Variable
147     ] , [
148       a owl:Restriction ;
149       owl:onProperty :groupsOn ;
150       owl:allValuesFrom :Variable
151     ] , [
152       a owl:Restriction ;
153       owl:onProperty :underCondition ;
154       owl:allValuesFrom :ConditionFormula
155     ] , [
156       a owl:Restriction ;
157       owl:onProperty :withFunction ;
158       owl:allValuesFrom :Function
159     ] , [
160       a owl:Restriction ;
161       owl:onProperty :comprehendsOn ;
162       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
163       owl:onClass :Variable
164     ] , [
165       a owl:Restriction ;
166       owl:onProperty :underCondition ;
167       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
168       owl:onClass :ConditionFormula
169     ] , [
170       a owl:Restriction ;
171       owl:onProperty :withFunction ;
172       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
173       owl:onClass :Function
```

```
174     ] .
175
176   # http://www.hs_rm.de/cs/vs/ontml#AllenOperator
177   :AllenOperator a owl:Class ;
178     rdfs:subClassOf [
179       owl:oneOf (:After :Before)
180     ] .
181
182   # http://www.hs_rm.de/cs/vs/ontml#AllenRelation
183   :AllenRelation a owl:Class ;
184     rdfs:subClassOf :TemporalRestriction , [
185       a owl:Restriction ;
186       owl:onProperty :hasOperand ;
187       owl:allValuesFrom :BaseTerm
188     ] , [
189       a owl:Restriction ;
190       owl:onProperty :hasOperator ;
191       owl:allValuesFrom :AllenOperator
192     ] , [
193       a owl:Restriction ;
194       owl:onProperty :hasOperand ;
195       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
196       owl:onClass :BaseTerm
197     ] , [
198       a owl:Restriction ;
199       owl:onProperty :hasOperator ;
200       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
201       owl:onClass :AllenOperator
202     ].
203
204   # http://www.hs_rm.de/cs/vs/ontml#Assert
205   :Assert a owl:Class ;
206     rdfs:subClassOf :AtomicAction .
207
208   # http://www.hs_rm.de/cs/vs/ontml#AssertCompound
209   :AssertCompound a owl:Class ;
210     rdfs:subClassOf :CompoundAction .
211
212   # http://www.hs_rm.de/cs/vs/ontml#Assignment
213   :Assignment a owl:Class ;
214     rdfs:subClassOf :EqualityTerm , [
215       a owl:Restriction ;
216       owl:onProperty :hasLeftHand ;
217       owl:allValuesFrom :Variable
218     ] , [
219       a owl:Restriction ;
220       owl:onProperty :hasLeftHand ;
221       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
```

```
222       owl:onClass :Variable
223     ] .
224
225   # http://www.hs_rm.de/cs/vs/ontml#AtomicAction
226   :AtomicAction a owl:Class ;
227     rdfs:subClassOf :Action , [
228       a owl:Restriction ;
229       owl:onProperty :target ;
230       owl:allValuesFrom :PositionalTerm
231     ] , [
232       a owl:Restriction ;
233       owl:onProperty :target ;
234       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
235       owl:onClass :PositionalTerm
236     ] ;
237     owl:disjointUnionOf (:Assert :Execute :Retract) .
238
239   # http://www.hs_rm.de/cs/vs/ontml#AtomicFormula
240   :AtomicFormula a owl:Class ;
241     rdfs:subClassOf :ConditionFormula ;
242     owl:disjointUnionOf (:EqualityTerm :Frame :PositionalTerm ) .
243
244   # http://www.hs_rm.de/cs/vs/ontml#Attribute
245   :Attribute a owl:Class ;
246     owl:disjointUnionOf (:IndividualAttribute :LiteralAttribute) .
247
248   # http://www.hs_rm.de/cs/vs/ontml#BaseTerm
249   :BaseTerm a owl:Class ;
250     rdfs:subClassOf :Term ;
251     owl:disjointUnionOf (:AggregationTerm :DocumentTerm :PositionalTerm :SimpleTerm) .
252
253   # http://www.hs_rm.de/cs/vs/ontml#Characteristic
254   :Characteristic a owl:Class ;
255     owl:disjointUnionOf (:CharacteristicConjunction :SimpleCharacteristic) .
256
257   # http://www.hs_rm.de/cs/vs/ontml#CharacteristicConjunction
258   :CharacteristicConjunction a owl:Class ;
259     rdfs:subClassOf :Characteristic , [
260       a owl:Restriction ;
261       owl:onProperty :hasLeftHand ;
262       owl:allValuesFrom :Characteristic
263     ], [
264       a owl:Restriction ;
265       owl:onProperty :hasRightHand ;
266       owl:allValuesFrom :Characteristic
267     ], [
268       a owl:Restriction ;
269       owl:onProperty :hasLeftHand ;
```

```
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Characteristic
  ], [
    a owl:Restriction ;
    owl:onProperty :hasRightHand ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Characteristic
  ] .

# http://www.hs_rm.de/cs/vs/ontml#ClassCharacteristic
:ClassCharacteristic a owl:Class ;
  rdfs:subClassOf :SimpleCharacteristic , [
    a owl:Restriction ;
    owl:onProperty :hasClass ;
    owl:allValuesFrom :DomainClass
  ], [
    a owl:Restriction ;
    owl:onProperty :hasClass ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :DomainClass
  ] .

# http://www.hs_rm.de/cs/vs/ontml#ComplexFormula
:ComplexFormula a owl:Class ;
  rdfs:subClassOf :ConditionFormula ;
  owl:disjointUnionOf (:ConnectionFormula :NestedFormula ) .

# http://www.hs_rm.de/cs/vs/ontml#CompoundAction
:CompoundAction a owl:Class ;
  rdfs:subClassOf :Action , [
      a owl:Restriction ;
     owl:onProperty :target ;
     owl:allValuesFrom :Frame
  ], [
    a owl:Restriction ;
    owl:onProperty :target ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Frame
  ] ;
  owl:disjointUnionOf (:AssertCompound :RetractCompound) .

# http://www.hs_rm.de/cs/vs/ontml#ConditionFormula
:ConditionFormula a owl:Class ;
  owl:disjointUnionOf (:ComplexFormula :EqualityTerm :Frame ) .

# http://www.hs_rm.de/cs/vs/ontml#Conjunction
:Conjunction a owl:Class ;
  rdfs:subClassOf :ConnectionFormula , [
```

```
318       a owl:Restriction ;
319       owl:onProperty :hasLeftHand ;
320       owl:allValuesFrom :ConditionFormula
321     ] .
322
323   # http://www.hs_rm.de/cs/vs/ontml#ConnectionFormula
324   :ConnectionFormula a owl:Class ;
325     rdfs:subClassOf :ComplexFormula , [
326       a owl:Restriction ;
327       owl:onProperty :hasLeftHand ;
328       owl:allValuesFrom :ConditionFormula
329     ] , [
330       a owl:Restriction ;
331       owl:onProperty :hasRightHand ;
332       owl:allValuesFrom :ConditionFormula
333     ] , [
334       a owl:Restriction ;
335       owl:onProperty :hasLeftHand ;
336       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
337       owl:onClass :ConditionFormula
338     ] , [
339       a owl:Restriction ;
340       owl:onProperty :hasRightHand ;
341       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
342       owl:onClass :ConditionFormula
343     ] ;
344     owl:disjointUnionOf (:Conjunction :Disjunction) .
345
346   # http://www.hs_rm.de/cs/vs/ontml#Constant
347   :Constant a owl:Class ;
348     rdfs:subClassOf :SimpleTerm ;
349     owl:disjointUnionOf (:IndividualConstant :LiteralConstant) .
350
351   # http://www.hs_rm.de/cs/vs/ontml#Dependency
352   :Dependency a owl:Class ;
353     rdfs:subClassOf :ImplicationRule .
354
355   # http://www.hs_rm.de/cs/vs/ontml#Disjunction
356   :Disjunction a owl:Class ;
357     rdfs:subClassOf :ConnectionFormula .
358
359   # http://www.hs_rm.de/cs/vs/ontml#DocumentTerm
360   :DocumentTerm a owl:Class ;
361     rdfs:subClassOf :BaseTerm , [
362       a owl:Restriction ;
363       owl:onProperty :consistsOf ;
364       owl:allValuesFrom :Frame
365     ] .
```

```
366
367 # http://www.hs_rm.de/cs/vs/ontml#DomainClass
368 :DomainClass a owl:Class ;
369   rdfs:subClassOf :DomainEntity .
370
371 # http://www.hs_rm.de/cs/vs/ontml#DomainEntity
372 :DomainEntity a owl:Class ;
373   owl:disjointUnionOf (:DomainClass :DomainRole) .
374
375 # http://www.hs_rm.de/cs/vs/ontml#DomainRole
376 :DomainRole a owl:Class ;
377   rdfs:subClassOf :DomainEntity .
378
379 # http://www.hs_rm.de/cs/vs/ontml#EqualityTerm
380 :EqualityTerm a owl:Class ;
381   rdfs:subClassOf :AtomicFormula , :Term , [
382     a owl:Restriction ;
383     owl:onProperty :hasRightHand ;
384     owl:allValuesFrom :BaseTerm
385   ] , [
386     a owl:Restriction ;
387     owl:onProperty :hasRightHand ;
388     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
389     owl:onClass :BaseTerm
390   ] .
391
392 :EqualityTerm owl:disjointUnionOf (:Assignment :Guard ) .
393
394 # http://www.hs_rm.de/cs/vs/ontml#EventCorrelation
395 :EventCorrelation a owl:Class ;
396   rdfs:subClassOf :ProductionRule .
397
398 # http://www.hs_rm.de/cs/vs/ontml#EventMapping
399 :EventMapping a owl:Class ;
400   rdfs:subClassOf :ProductionRule .
401
402 # http://www.hs_rm.de/cs/vs/ontml#Execute
403 :Execute a owl:Class ;
404   rdfs:subClassOf :AtomicAction .
405
406 # http://www.hs_rm.de/cs/vs/ontml#Existentials
407 :Existentials a owl:Class ;
408   rdfs:subClassOf :NestedFormula , [
409     a owl:Restriction ;
410     owl:onProperty :hasVariable ;
411     owl:allValuesFrom :Variable
412   ] .
413
```

```
# http://www.hs_rm.de/cs/vs/ontml#Frame
:Frame a owl:Class ;
  rdfs:subClassOf :AtomicFormula , :Term , [
    a owl:Restriction ;
    owl:onProperty :hasCcharacteristic ;
    owl:allValuesFrom :Characteristic
  ] , [
    a owl:Restriction ;
    owl:onProperty :hasContext ;
    owl:allValuesFrom :SimpleTerm
  ] , [
    a owl:Restriction ;
    owl:onProperty :hasCharacteristic ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Characteristic
  ] , [
    a owl:Restriction ;
    owl:onProperty :hasContext ;
    owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :SimpleTerm
  ] .

# http://www.hs_rm.de/cs/vs/ontml#Function
:Function a owl:Class ;
  rdfs:subClassOf :PositionalTerm .

# http://www.hs_rm.de/cs/vs/ontml#Goal
:Goal a owl:Class ;
  rdfs:subClassOf :PositionalTerm , [
    a owl:Restriction ;
    owl:onProperty :extends ;
    owl:allValuesFrom :Goal
  ] , [
    a owl:Restriction ;
    owl:onProperty :withAttribute ;
    owl:allValuesFrom :Attribute
  ] , [
    a owl:Restriction ;
    owl:onProperty :withAttribute ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Attribute
  ] , [
    a owl:Restriction ;
    owl:onProperty :extends ;
    owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :Goal
  ] .

```

```
# http://www.hs_rm.de/cs/vs/ontml#Guard
:Guard a owl:Class ;
  rdfs:subClassOf :EqualityTerm .

# http://www.hs_rm.de/cs/vs/ontml#ImplicationRule
:ImplicationRule a owl:Class ;
  rdfs:subClassOf :Rule , [
    a owl:Restriction ;
    owl:onProperty :hasConsequence ;
    owl:allValuesFrom [ owl:unionOf (:Frame :PositionalTerm) ]
  ] ;
  owl:disjointUnionOf (:ActionSuggestion :Dependency :Query :Score ) .


# http://www.hs_rm.de/cs/vs/ontml#IndividualAttribute
:IndividualAttribute a owl:Class ;
  rdfs:subClassOf :Attribute .

# http://www.hs_rm.de/cs/vs/ontml#IndividualConstant
:IndividualConstant a owl:Class ;
  rdfs:subClassOf :Constant , [
    a owl:Restriction ;
    owl:onProperty :referesTo ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
  ] .

# http://www.hs_rm.de/cs/vs/ontml#LiteralAttribute
:LiteralAttribute a owl:Class ;
  rdfs:subClassOf :Attribute .

# http://www.hs_rm.de/cs/vs/ontml#LiteralConstant
:LiteralConstant a owl:Class ;
  rdfs:subClassOf :Constant , [
    a owl:Restriction ;
    owl:onProperty :value ;
    owl:cardinality "1"^^xsd:nonNegativeInteger
  ] .

# http://www.hs_rm.de/cs/vs/ontml#Module
:Module a owl:Class ;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :consistsOf ;
    owl:allValuesFrom [ owl:unionOf (:Goal :Rule) ]
  ] , [
    a owl:Restriction ;
    owl:onProperty :dependsOn ;
    owl:allValuesFrom :Module
```

```
  ] .

# http://www.hs_rm.de/cs/vs/ontml#Negation
:Negation a owl:Class ;
  rdfs:subClassOf :NestedFormula .

# http://www.hs_rm.de/cs/vs/ontml#NestedFormula

:NestedFormula a owl:Class ;
  rdfs:subClassOf :ComplexFormula , [
    a owl:Restriction ;
    owl:onProperty :hasSubFormula ;
    owl:allValuesFrom :ConditionFormula
  ] , [
    a owl:Restriction ;
    owl:onProperty :hasSubFormula ;
    owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :ConditionFormula
  ] ;
  owl:disjointUnionOf (:Existentials :Negation :Universals) .


# http://www.hs_rm.de/cs/vs/ontml#Policy
:Policy a owl:Class ;
  rdfs:subClassOf :ProductionRule .

# http://www.hs_rm.de/cs/vs/ontml#PositionalTerm
:PositionalTerm a owl:Class ;
  rdfs:subClassOf :AtomicFormula , :BaseTerm , [
    a owl:Restriction ;
    owl:onProperty :applies ;
    owl:allValuesFrom :BaseTerm
  ] , [
    a owl:Restriction ;
    owl:onProperty :to ;
    owl:allValuesFrom :PositionalTerm
  ] , [
    a owl:Restriction ;
    owl:onProperty :applies ;
    owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :BaseTerm
  ] , [
    a owl:Restriction ;
    owl:onProperty :to ;
    owl:maxQualifiedCardinality "1"^^xsd:nonNegativeInteger ;
    owl:onClass :PositionalTerm
  ] ;
  owl:disjointUnionOf (:Function :Goal <http://wwwvs.cs.hs-rm.de/obmf/dio#InterfaceType>) .
```

```
558
559 # http://www.hs_rm.de/cs/vs/ontml#ProductionRule
560 :ProductionRule a owl:Class ;
561   rdfs:subClassOf :Rule , [
562     a owl:Restriction ;
563     owl:onProperty :hasConsequence ;
564     owl:allValuesFrom <http://wwwvs.cs.hs-rm.de/obmf/dio#Action>
565   ] ,[
566     a owl:Restriction ;
567     owl:onProperty :hasConsequence ;
568     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
569     owl:onClass <http://wwwvs.cs.hs-rm.de/obmf/dio#Action>
570   ] ;
571   owl:disjointUnionOf (:ActionInterpretation :EventCorrelation :EventMapping :Policy) .
572
573
574 # http://www.hs_rm.de/cs/vs/ontml#Query
575 :Query a owl:Class ;
576   rdfs:subClassOf :ImplicationRule .
577
578 # http://www.hs_rm.de/cs/vs/ontml#RelativeInstant
579 :RelativeInstant a owl:Class ;
580   rdfs:subClassOf :TemporalRestriction .
581
582 # http://www.hs_rm.de/cs/vs/ontml#Retract
583 :Retract a owl:Class ;
584   rdfs:subClassOf :AtomicAction .
585
586 # http://www.hs_rm.de/cs/vs/ontml#RetractCompound
587 :RetractCompound a owl:Class ;
588   rdfs:subClassOf :CompoundAction .
589
590 # http://www.hs_rm.de/cs/vs/ontml#RoleCharacteristic
591 :RoleCharacteristic a owl:Class ;
592   rdfs:subClassOf :SimpleCharacteristic , [
593     a owl:Restriction ;
594     owl:onProperty :hasRole ;
595     owl:allValuesFrom :DomainRole
596   ] , [
597     a owl:Restriction ;
598     owl:onProperty :withValue ;
599     owl:allValuesFrom :SimpleTerm
600   ] , [
601     a owl:Restriction ;
602     owl:onProperty :hasRole ;
603     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
604     owl:onClass :DomainRole
605   ] , [
```

```
606       a owl:Restriction ;
607       owl:onProperty :withValue ;
608       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
609       owl:onClass :SimpleTerm
610     ] .
611
612   # http://www.hs_rm.de/cs/vs/ontml#Rule
613   :Rule a owl:Class ;
614     rdfs:subClassOf [
615       a owl:Restriction ;
616       owl:onProperty :hasCondition ;
617       owl:allValuesFrom :ConditionFormula
618     ] , [
619       a owl:Restriction ;
620       owl:onProperty :hasVariable ;
621       owl:allValuesFrom :Variable
622     ] , [
623       a owl:Restriction ;
624       owl:onProperty :hasCondition ;
625       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
626       owl:onClass :ConditionFormula
627     ] , [
628       a owl:Restriction ;
629       owl:onProperty :name ;
630       owl:allValuesFrom xsd:string
631     ] , [
632       a owl:Restriction ;
633       owl:onProperty :name ;
634       owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
635       owl:onDataRange xsd:string
636     ] ;
637     owl:disjointUnionOf (:ImplicationRule :ProductionRule).
638
639   # http://www.hs_rm.de/cs/vs/ontml#Score
640   :Score a owl:Class ;
641     rdfs:subClassOf :ImplicationRule .
642
643   # http://www.hs_rm.de/cs/vs/ontml#SimpleCharacteristic
644   :SimpleCharacteristic a owl:Class ;
645     rdfs:subClassOf :Characteristic , [
646       a owl:Restriction ;
647       owl:onProperty :underRestriction ;
648       owl:allValuesFrom :TemporalRestriction
649     ] ;
650     owl:disjointUnionOf (:ClassCharacteristic :RoleCharacteristic) .
651
652   # http://www.hs_rm.de/cs/vs/ontml#SimpleTerm
653   :SimpleTerm a owl:Class ;
```

```
654   rdfs:subClassOf :BaseTerm .
655
656 # http://www.hs_rm.de/cs/vs/ontml#TemporalRestriction
657 :TemporalRestriction a owl:Class ;
658   rdfs:subClassOf [
659     a owl:Restriction ;
660     owl:onProperty :duration ;
661     owl:allValuesFrom xsd:duration
662   ] , [
663     a owl:Restriction ;
664     owl:onProperty :duration ;
665     owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
666     owl:onDataRange xsd:duration
667   ] ;
668   owl:disjointUnionOf (:AllenRelation :RelativeInstant :TimeWindow) .
669
670 # http://www.hs_rm.de/cs/vs/ontml#Term
671 :Term a owl:Class .
672
673 # http://www.hs_rm.de/cs/vs/ontml#TimeWindow
674 :TimeWindow a owl:Class ;
675   rdfs:subClassOf :TemporalRestriction .
676
677 # http://www.hs_rm.de/cs/vs/ontml#Universals
678 :Universals a owl:Class ;
679   rdfs:subClassOf :NestedFormula , [
680     a owl:Restriction ;
681     owl:onProperty :hasVariable ;
682     owl:allValuesFrom :Variable
683   ] .
684
685 # http://www.hs_rm.de/cs/vs/ontml#Variable
686 :Variable a owl:Class ;
687   rdfs:subClassOf :SimpleTerm , [
688     a owl:Restriction ;
689     owl:onProperty :name ;
690     owl:allValuesFrom xsd:string
691   ] .
692
693 # http://www.hs_rm.de/cs/vs/ontml#Wildcard
694 :Wildcard a owl:Class ;
695   rdfs:subClassOf :SimpleTerm .
696
697 # http://wwwvs.cs.hs-rm.de/obmf/dio#InterfaceType
698 <http://wwwvs.cs.hs-rm.de/obmf/dio#InterfaceType> rdfs:subClassOf :PositionalTerm .
699
700 # http://www.hs_rm.de/cs/vs/ontml#After
701 :After a owl:NamedIndividual , :AllenOperator .
```

```
# http://www.hs_rm.de/cs/vs/ontml#Before
:Before a owl:NamedIndividual , :AllenOperator .
```

A.2.4 Ontology-based Management Language (OntML)-Grammatik

```
grammar de.hs_rm.cs.vs.OntML hidden(WS, ML_COMMENT, SL_COMMENT)

generate ontml "http://www.hs_rm.de/cs/vs/OntML"
import "http://www.eclipse.org/emf/2002/Ecore" as ecore
import "http://wwwvs.cs.hs-rm.de/ontml/dio" as dio

Module:
  'module' iri=FULL_IRI '('
  (imports+=Import)*
  (prefixes+=Prefix)*
  (rules+=Rule | goals+=Goal)*
  ')';

Import:
  'import' importURI=STRING;

/* Meta */
Prefix:
  'prefix' name=PREFIX_NAME iri=PREFIX_IRI;

Goal returns dio::Goal:
  'goal' {dio::Goal} iri=FULL_IRI '(' attributes+=Attribute
                                  (',' attributes+=Attribute)* ')';

Attribute returns dio::Attribute:
  name=ID '::' type=Types;

enum Types returns dio::Primitive:
  String='String'
  | Integer='Integer'
  | Float='Float'
  | Double='Double'
  | Boolean='Boolean'
  | Long='Long'
  | DateTime='DateTime'
  | Individual='Individual';

enum AggregateBuiltIn returns FunctionBuiltIn:
  Sum='sum' | Average='average' | Minimum='minimum'
  | Maximum='maximum' | Count='count';
```

```
41
42  enum CastBuiltIn returns FunctionBuiltIn:
43    ToInteger='toInteger' | ToLong='toLong' | ToDouble='toDouble';
44
45  enum ProductionRuleType:
46    Correlation | Mapping | Policy | Interpretation;
47
48  enum ImplicationRuleType:
49    Dependency | Query | Score | Suggestion;
50
51  /* Rules */
52  Variable:
53    name=VARIABLE_ID;
54
55  Rule returns Rule:
56    ProductionRule | ImplicationRule;
57
58  ProductionRule returns ProductionRule:
59    type=ProductionRuleType name=STRING
60    ('forall' variables+=Variable+ '.')?
61    'if' condition=ConditionFormula 'then' consequence=ActionBlock;
62
63  ImplicationRule returns ImplicationRule:
64    type=ImplicationRuleType name=STRING
65    ('forall' variables+=Variable+ '.')?
66    (consequences+=(Frame | PositionalTerm)
67    | 'and' '(' consequences+=(Frame | PositionalTerm)* ')')
68    ':-' condition=ConditionFormula;
69
70  /* Actions */
71  ActionBlock:
72    'do' {ActionBlock} '(' actions+=ProductionAction* ')';
73
74  ProductionAction returns ProductionAction:
75    AtomicAction | CompoundAction;
76
77  AtomicAction returns AtomicAction:
78    ('assert' {Assert} | 'retract' {Retract}
79    | 'execute' {Execute}) target=PositionalTerm;
80
81  CompoundAction returns CompoundAction:
82    ('assert' {AssertCompound} | 'retract' {RetractCompound}) target=Frame;
83
84  /* ConditionFormula */
85  ConditionFormula returns ConditionFormula:
86    AtomicFormula | ComplexFormula;
87
88  ComplexFormula returns ComplexFormula:
```

```
89    ConnectionFormula | NestedFormula;
90
91  ConnectionFormula returns ConnectionFormula:
92    Conjunction | Disjunction;
93
94  NestedFormula returns NestedFormula:
95    Existentials | Universals | Negation;
96
97  Conjunction returns Conjunction:
98    'and' '(' subFormulas+=ConditionFormula+ ')';
99
100 Disjunction returns Disjunction:
101   'or' '(' subFormulas+=ConditionFormula+ ')';
102
103 Existentials:
104   'exists' variables+=Variable+ '.' subformula=ConditionFormula;
105
106 Universals:
107   'forall' variables+=Variable+ '.' subformula=ConditionFormula;
108
109 Negation:
110   'not' subformula=ConditionFormula;
111
112 AtomicFormula returns AtomicFormula:
113   EqualityTerm | Frame | BaseTerm;
114
115 /* Terms */
116 EqualityTerm returns EqualityTerm:
117   Assignment | Guard;
118
119 Assignment:
120   'let' left=VariableReference '=' right=BaseTerm;
121
122 Guard:
123   'guard' '(' right=BaseTerm ')';
124
125 /* Frame */
126 Frame:
127   context=BaseTerm -> '[' characteristics+=Characteristic* ']';
128
129 Characteristic returns Characteristic:
130   (ClassCharacteristic | RoleCharacteristic)
131   temporalRestriction=TemporalExpression?;
132
133 ClassCharacteristic:
134   -> 'a' '->' type=[dio::OClass|IRI];
135
136 RoleCharacteristic:
```

```
137   predicate=[dio::Property|IRI] '->' object=BaseTerm;
138
139 /* BaseTerm */
140 BaseTerm returns Term:
141   PositionalTerm | ExternalTerm | DocumentTerm;
142
143 /* Positional Terms */
144 PositionalTerm returns BaseTerm:
145   {PositionalTerm} type=[dio::InterfaceType|IRI] '(' arguments+=BaseTerm
146   (',' arguments+=BaseTerm)* ')' temporalRestriction=TemporalExpression?;
147
148 /* DocumentTerm */
149 DocumentTerm returns BaseTerm:
150   {DocumentTerm} 'Document' '(' frames+=Frame* ')';
151
152 /* Simple Terms */
153 SimpleTerm returns BaseTerm:
154   VariableReference | Constant | Wildcard | BNode;
155
156 VariableReference returns SimpleTerm:
157   {VariableReference} variable=[Variable|VARIABLE_ID];
158
159 Constant returns SimpleTerm:
160   {StringConstant} value=STRING | {LongConstant} value=Long
161   | {DoubleConstant} value=Double | {IntegerConstant} value=INT
162   | {TrueConstant} 'true' | {FalseConstant} 'false' | IndividualReference;
163
164 IndividualReference returns SimpleTerm:
165   {IndividualReference} Individual=[dio::Resource|IRI];
166
167 Wildcard returns SimpleTerm:
168   {Wildcard} '_';
169
170 BNode returns SimpleTerm:
171   {BNode} 'BNode' '(' idTerms+=SimpleTerm (',' idTerms+=SimpleTerm)* ')';
172
173 /* External Term */
174 ExternalTerm returns Term:
175   Expression | AggregateTerm;
176
177 AggregateTerm returns AggregateTerm:
178   function=AggregateBuiltIn comprehensionVariable=Variable
179   ('[' variables+=Variable+ ']')? '.' subformula=ConditionFormula;
180
181 Expression returns Term:
182   LogicalDisjunction;
183
184 LogicalDisjunction returns Term:
```

```
185   LogicalConjunction
186   ('&&' {LogicalDisjunction.left=current} right=LogicalConjunction)*;
187
188 LogicalConjunction returns Term:
189   Relation
190   ('||' {LogicalConjunction.left=current} right=Relation)*;
191
192 Relation returns Term:
193   AdditionSubtraction ((-> '==' {Equal.left=current}
194                         | -> '!=' {NotEqual.left=current}
195                         | -> '>' {GreaterThan.left=current}
196                         | -> '>=' {GreaterEqual.left=current}
197                         | -> '<' {LessThan.left=current}
198                         | -> '<=' {LessEqual.left=current})
199   right=AdditionSubtraction)?;
200
201 AdditionSubtraction returns Term:
202   MultiplicationDivision ((-> '+' {Addition.left=current}
203                           | -> '-' {Subtraction.left=current})
204   right=MultiplicationDivision)*;
205
206 MultiplicationDivision returns Term:
207   ExpressionPrimary ((-> '*' {Multiplication.left=current}
208                      | -> '/' {Division.left=current}
209                      | -> '%' {Modulo.left=current})
210   right=ExpressionPrimary)*;
211
212 ExpressionPrimary returns Term:
213   SimpleTerm | Application | '(' BaseTerm ')';
214
215 Application returns Term:
216   {Application} function=(AggregateBuiltIn | CastBuiltIn)
217   '(' (arguments+=BaseTerm (',' arguments+=BaseTerm)*)? ')';
218
219 /* Temporal */
220 TemporalExpression:
221   InstantExpression | WindowExpression | AllenRestriction;
222
223 InstantExpression:
224   duration=Duration
225   'ago';
226
227 WindowExpression:
228   'within' duration=Duration;
229
230 AllenRestriction:
231   duration=Duration? operator=AllenOperator filler=VariableReference;
232
```

```
Duration:
  MilliSeconds;

MilliSeconds returns Duration:
  Seconds (milliSeconds=INT 'ms')? | (milliSeconds=INT 'ms');

Seconds returns Duration:
  Minutes (seconds=INT 's')?
  | (seconds=INT 's');

Minutes returns Duration:
  Hours (minutes=INT 'm')? | minutes=INT 'm';

Hours returns Duration:
  hours=INT 'h';

enum AllenOperator:
  Before='before' | After='after';

IRI:
  FULL_IRI | ABBREV_IRI;

FULL_IRI returns ecore::EString:
  '<' PATH_ELEMENT '://' PATH_ELEMENT ('.' PATH_ELEMENT)* (':' INT)?
  ('/' PATH_ELEMENT)+ ('#' PATH_ELEMENT)? '>';

PREFIX_IRI returns ecore::EString:
  '<' PATH_ELEMENT '://' PATH_ELEMENT
  ('.' PATH_ELEMENT)* (':' INT)? ('/' PATH_ELEMENT)+ ('#' | "/") '>';

ABBREV_IRI:
  PREFIX_NAME PATH_ELEMENT;

PREFIX_NAME:
  PATH_ELEMENT? ':';

PATH_ELEMENT:
  SIMPLE_PATH_ELEMENT ('-' SIMPLE_PATH_ELEMENT)*;

SIMPLE_PATH_ELEMENT:
  INT (ID?) | ID;

VARIABLE_ID:
  '?' ID;

Double returns ecore::EDouble:
  INT '.' INT;
```

```
Long returns ecore::ELong:
  INT 'L';

terminal ID:
  ('a'..'z' | 'A'..'Z' | '_') ('a'..'z' | 'A'..'Z' | '_' |
  '0'..'9')*;

terminal INT returns ecore::EInt:
  ('0'..'9')+;

terminal STRING:
  '"' ('\\' .
  /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\' | '"'))* '"' | "'" ('\\' .
  /* 'b'|'t'|'n'|'f'|'r'|'u'|'"'|"'"|'\\' */ | !('\\' | "'"))* "'";

terminal ML_COMMENT:
  '(*'->'*)';

terminal SL_COMMENT:
  '--' !('\n' | '\r')* ('\r'? '\n')?;

terminal WS:
  (' ' | '\t' | '\r' | '\n')+;

terminal ANY_OTHER:
  .;
```

A.2.5 VM-Beispiel (RDF)

```
@prefix : <http://www.hs_rm.de/cs/vs/ontml#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix vm: <http://wwwvs.cs.hs-rm.de/vm#> .
@prefix inf: <http://wwwvs.cs.hs-rm.de/vm/interface#> .
@prefix rules: <http://wwwvs.cs.hs-rm.de/vm/VMRules#> .
@prefix mvm: <http://wwwvs.cs.hs-rm.de/vm/VMRules/EventMapping#> .
@prefix ml: <http://wwwvs.cs.hs-rm.de/vm/VMRules/HostMaxVMLoad#> .

# OntML-Modul, bestehend aus MapVMLoad-Regel, HostMaxVMLoad-Implikation und maxLoad-Ziel
<http://wwwvs.cs.hs-rm.de/vm/Rules> a owl:Ontology , :Module ;
  owl:imports <http://wwwvs.cs.hs-rm.de/vm/interface> ,
              <http://wwwvs.cs.hs-rm.de/vm> ,
              <http://www.ibm.com/4dFluents> ,
              <http://wwwvs.cs.hs-rm.de/obmf/DIO> ,
```

```
18                <http://www.hs_rm.de/cs/vs/OntML> ;
19    :consistsOf rules:MapVMLoad, rules:HostMaxVMLoad, rules:maxLoad .
20
21  # Variablen
22  mvm:vmVar a :Variable ;
23    :name "vm" .
24
25  mvm:idVar a :Variable ;
26    :name "id" .
27
28  mvm:evVar a :Variable ;
29    :name "ev" .
30
31  mvm:loadVar a :Variable ;
32    :name "load" .
33
34  # Event-Mapping-Regel "Map VM Load"
35  rules:MapVMLoad a :EventMapping ;
36    :name "Map VM Load" ;
37    # Referenzierte Variablen
38    :hasVariable mvm:evVar , mvm:loadVar , mvm:idVar , mvm:vmVar ;
39    # Bedingung als Konjunktion von Formeln
40    :hasCondition [
41      a :Conjunction ;
42      :hasLeftHand [
43        # Frame in der Wissensbasis
44        a :Frame ;
45        # Frame hat Kontext ?vm
46        :hasContext mvm:vmVar ;
47        :hasCharacteristic [
48         # Rollencharakteristik bindet das Tripel (?vm vm:id ?id)
49          a :RoleCharacteristic ;
50          :withValue mvm:idVar ;
51          :hasRole vm:id ;
52        ] ;
53      ] ;
54      :hasRightHand [
55        # Positionsterm
56        a :PositionalTerm ;
57        :to [
58          a :PositionalTerm ;
59          # VMLoadEvent als Symbol
60          :to inf:VMLoadEvent ;
61          # Erstes Argument
62          :applies mvm:evVar ;
63        ] ;
64        # Zweites Argument
65        :applies [
```

```
        # Event-Dokument
        a :DocumentTerm ;
        :consistsOf [
          # Event-Dokument Frame
          a :Frame ;
          # Frame hat Kontext ?ev
          :hasContext mvm:evVar ;
          :hasCharacteristic [
            # Frame bestehend aus zwei Charakteristiken
            a :CharacteristicConjunction ;
            :hasLeftHand [
              # Rollencharakteristik bindet das Tripel (?ev inf:vmId ?id)
              a :RoleCharacteristic ;
              :withValue mvm:idVar ;
              :hasRole inf:vmId ;
            ] ;
            :hasRightHand [
              # Rollencharakteristik bindet das Tripel (?ev inf:load ?load)
              a :RoleCharacteristic ;
              :withValue mvm:loadVar ;
              :hasRole inf:load ;
            ] ;
          ] ;
        ] ;
      ] ;
    ] ;
  ] ;
  # Konsequenz
  :hasConsequence [
    # Geschachtelte Aktionsblöcke
    a :ActionBlock ;
    :hasLeftHand [
      a :ActionBlock ;
      :hasLeftHand [
        # Löschen aller Tripel (?vm vm:load _)
        a :RetractCompound ;
        :target [
          a :Frame ;
          :hasContext mvm:vmVar ;
          :hasCharacteristic [
            a :RoleCharacteristic ;
            :hasRole vm:load ;
            :withValue [ a :Wildcard ] ;
          ] ;
        ] ;
      ] ;
      :hasRightHand [
        # Erzeugen des Tripels (?vm vm:load ?load)
```

```
        a :AssertCompound ;
        :target [
           a :Frame ;
          :hasContext mvm:vmVar ;
          :hasCharacteristic [
            a :RoleCharacteristic ;
            :hasRole inf:load ;
            :withValue mvm:loadVar ;
          ] ;
        ] ;
      ] ;
    ] ;
    :hasRightHand [
      # Löschen des verarbeiteten VMLoadEvents
      a :Retract ;
      :target [
        a :PositionalTerm ;
        :to [
          a :PositionalTerm ;
          :applies mvm:evVar ;
          :to inf:VMLoadEvent ;
        ] ;
        :applies [ a :Wildcard ] ;
      ]
    ] ;
  ] .

# Abstraktes Ziel
rules:maxLoad a :Goal ;
  :extends [
    a :Goal ;
    # Mit Attribut "load" : Double
    :withAttribute [
      a :LiteralAttribute ;
      :restrictedTo xsd:double ;
      :name "load" ;
    ] ;
  ] ;
  :withAttribute [
    # Mit Attribut "host" : Individual
    a :IndividualAttribute ;
    :name "host" ;
  ] .

# Variablen
ml:maxLoadVar a :Variable ;
  :name "maxLoad" .
```

```
ml:hostVar a :Variable ;
  :name "host" .

ml:vmVar a :Variable ;
  :name "vm" .

ml:loadVar a :Variable ;
  :name "load" .

# Query
rules:HostMaxVMLoad a :ImplicationRule ;
  :name "Host Max VM Load" ;
  :hasVariable ml:maxLoadVar , ml:hostVar ;
  # Bedingung
  :hasCondition [
    a :Conjunction ;
    :hasLeftHand [
      a :Frame ;
      :hasContext ml:hostVar ;
      :hasCharacteristic [
        a :ClassCharacteristic ;
        :hasClass vm:Host ;
      ] ;
    ] ;
    :hasRightHand [
      # Zuweisung
      a :Assignment ;
      :hasLeftHand ml:maxLoadVar ;
      :hasRightHand [
        # Aggregation
        a :AggregationTerm ;
        :groupsOn ml:vmVar ;
        :comprehendsOn ml:loadVar ;
        :withFunction <http://www.w3.org/2005/xpath-functions#max> ;
        :underCondition [
          a :Conjunction ;
          :hasLeftHand [
            a :Frame ;
            :hasContext ml:hostVar ;
            :hasCharacteristic [
              a :RoleCharacteristic ;
              :hasRole vm:deploys ;
              :withValue ml:vmVar ;
            ] ;
          ] ;
          :hasRightHand [
            a :Frame ;
            :hasContext ml:vmVar ;
```

```
            :hasCharacteristic [
              a :RoleCharacteristic ;
              :hasRole vm:load ;
              :withValue ml:loadVar ;
            ] ;
          ] ;
        ] ;
      ] ;
    ] ;
  ] ;
  # Implikation des abstrakten Ziels
  :hasConsequence [
    a :PositionalTerm ;
    :applies ml:maxLoadVar ;
    :to [
      a :PositionalTerm ;
      :applies ml:hostVar ;
      :to rules:maxLoad ;
    ] ;
  ] .
```

A.2.6 VM-Beispiel (OntML)

```
-- Modul
module <http://wwwvs.cs.hs-rm.de/vm#VMRules> (

  -- Imports
  import "VM.ttl"
  import "VMInterface.ttl"

    -- Präfixe
  prefix : <http://wwwvs.cs.hs-rm.de/vm#>
  prefix ev: <http://wwwvs.cs.hs-rm.de/vm/interface#>
  prefix rules: <http://wwwvs.cs.hs-rm.de/vm/vmrules#>

  -- Abstraktes Ziel
  goal <http://wwwvs.cs.hs-rm.de/vm/vmrules#maxLoad> ( host :: Individual, load :: Double )

    -- Mapping Regel
  Mapping "Map VM Load"
  forall ?vm ?id ?ev ?load .
  -- Bedingung
  if and (
    -- Frame in Wissensbasis
    ?vm [ :id -> ?id ]
    -- Positionsterm
```

```
      ev:VMLoadEvent ( ?ev,
        -- Dokumentterm
        Document (
          -- Frame im Dokument
          ?ev [
          ev:vmId -> ?id
          ev:load -> ?load
          ]
        )
      )
    )
    -- Konsequenz
    then do (
      -- Lösche Frame
      retract ?vm [ :load -> _ ]
      -- Erzeuge Frame
      assert ?vm [ ev:load -> ?load ]
      -- Lösche Event
      retract ev:VMLoadEvent(?ev, _)
    )

    -- Query
    Query "Host Max VM Load"
    forall ?host ?maxLoad .
    -- Impliziertes Ziel
    rules:maxLoad(?host, ?maxLoad) :- and (
      ?host [ a -> :Host ]
      -- Aggregation
      let ?maxLoad = maximum ?load [?vm] . and (
        ?host [ :deploys -> ?vm ]
        ?vm [:load -> ?load]
      )
    )
)
```

A.3 Fallstudie Storage-Management

A.3.1 Paths-Modul

```
module <http://wwwvs.cs.hs-rm.de/bvqmanagement/Paths> (

  import "../resources/ontologies/bvq_cloud_ext.ttl"
  import "../resources/ontologies/bvq_cloud.ttl"

  prefix dp: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/dataproperties#>
```

```
prefix op: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/objectproperties#>
prefix cl: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/classes#>
prefix ext: <http://www.vs.cs.hs-rm.de/ontostorm/BVQCloudExt#>
prefix paths: <http://wwwvs.cs.hs-rm.de/bvqmanagement/Paths#>

goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/Paths#mdg_vdisks>
     ( mdg :: Individual, vd :: Individual)
goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/Paths#mdg_ior>
     ( mdg :: Individual, ior :: Double)

Dependency "VDisk Data Class"
forall ?vd ?vdg ?dc .
?vd [ op:data_class -> ?dc ] :- and (
  ?vd [
    a -> cl:vdisk
    op:vdisk_group -> ?vdg
  ]
  ?vdg [ op:data_class -> ?dc ]
)

Dependency "VDisk Response Time Limit"
forall ?vd ?dataCl ?limit .
?vd [ ext:maxResponseTime -> ?limit ] :- and (
  ?vd [
    a -> cl:vdisk
    op:data_class -> ?dataCl
  ]
  ?dataCl [ ext:maxResponseTime -> ?limit ]
)

Query "MDisk Group VDisks"
forall ?mdg ?vd ?vdc ?md ?ext .
paths:mdg_vdisks(?mdg, ?vd) :- and (
  ?md [ op:mdisk_group -> ?mdg ]
  ?ext [
    op:mdisk -> ?md
    op:vdiskcopy -> ?vdc
  ]
  ?vdc [ op:vdisk -> ?vd ]
)

Query "MDisk Group Aggregated I/Os"
forall ?iops ?mdg .
paths:mdg_ior(?mdg, ?iops) :- and (
  ?mdg [ a -> cl:mdisk_group ]
  let ?iops = sum ?vd_iops [ ?vd ] . and (
    paths:mdg_vdisks(?mdg, ?vd)
    ?vd [ dp:vdisk_realtime_ior -> ?vd_iops ]
```

```
    )
  )

)
```

A.3.2 Events-Modul

```
module <http://wwwvs.cs.hs-rm.de/bvqmanagement/Events> (

  import "../resources/ontologies/bvq_cloud_ext.ttl"
  import "../resources/ontologies/bvq_cloud.ttl"
  import "../resources/ontologies/bvq_mt.ttl"

  prefix ev: <http://wwwvs.cs.hs-rm.de/bvqmanagement/types#>
  prefix dp: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/dataproperties#>
  prefix ext: <http://www.vs.cs.hs-rm.de/ontostorm/BVQCloudExt#>

  Mapping "Map VDisk Metric"
  forall ?iops ?respTime ?vd ?id ?stat .
  if and (
    ?vd [ dp:vdisk_id -> ?id ]
    ev:VDiskMetric ( ?stat, Document ( ?stat [
        ev:vDiskId -> ?id
        ev:iops -> ?iops
        ev:responseTime -> ?respTime
      ] ) )

  ) then do (
    -- Retract old I/Os and response time
    retract ?vd [
      dp:vdisk_realtime_ior -> _
      dp:vdisk_realtime_rt -> _
    ]
    -- Assert new I/Os and response time
    assert ?vd [
      dp:vdisk_realtime_ior -> ?iops
      dp:vdisk_realtime_rt -> ?respTime
    ]
    -- Delete event
    retract ev:VDiskMetric(?stat, _)
  )

  Mapping "Map Governed"
  forall ?iops ?vd ?id ?event .
  if and (
    ev:Governed ( ?event, Document ( ?event [
```

```
          ev:vDiskId -> ?id
          ev:iops -> ?iops
        ] ) )

    ?vd [ dp:vdisk_id -> ?id ]
  ) then do (
    retract ?vd [ dp:gov_ratio -> _ ]
    assert ?vd [ dp:gov_ratio -> toInteger(?iops) ]
    retract ev:Governed ( ?event, _ )
  )

  Mapping "Map Ungoverned"
  forall ?vd ?id ?event .
  if and (
    ev:Ungoverned ( ?event, Document ( ?event [ ev:vDiskId -> ?id ] ) )

    ?vd [ dp:vdisk_id -> ?id ]
  ) then do (
    retract ?vd [ dp:gov_ratio -> _ ]
    retract ev:Ungoverned ( ?event, _ )
  )

)
```

A.3.3 SLA-Modul

```
module <http://wwwvs.cs.hs-rm.de/bvqmanagement/SLA> (

  import "../resources/ontologies/bvq_cloud.ttl"
  import "../resources/ontologies/bvq_cloud_ext.ttl"

  prefix ext: <http://www.vs.cs.hs-rm.de/ontostorm/BVQCloudExt#>
  prefix dp: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/dataproperties#>

  Dependency "Response Time Violator"
  forall ?vd ?respTime ?respTime ?limit .
  ?vd [ a -> ext:ResponseTimeViolator ] :- and (
    ?vd [
      ext:maxResponseTime -> ?limit
      dp:vdisk_realtime_rt -> ?respTime
    ]
    guard ( ?respTime > ?limit )
  )
)
```

A.3.4 Governing-Modul

```
module <http://wwwvs.cs.hs-rm.de/bvqmanagement/Governing> (

  import "../resources/ontologies/bvq_cloud_ext.ttl"
  import "../resources/ontologies/bvq_cloud.ttl"
  import "../resources/ontologies/bvq_mt.ttl"
  import "Paths.ontml"

  prefix : <http://wwwvs.cs.hs-rm.de/bvqmanagement/types#>
  prefix gov: <http://wwwvs.cs.hs-rm.de/bvqmanagement/governing#>
  prefix dp: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/dataproperties#>
  prefix op: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/objectproperties#>
  prefix cl: <http://www.vs.cs.hs-rm.de/ontostorm/BVQ/classes#>
  prefix ext: <http://www.vs.cs.hs-rm.de/ontostorm/BVQCloudExt#>
  prefix paths: <http://wwwvs.cs.hs-rm.de/bvqmanagement/Paths#>

  goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/governing#mdg_bronze_vdisk_with_max_iops>
       ( mdg :: Individual, vd :: Individual, iops :: Long )
  goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/governing#mdg_bronze_vdisk>
       ( mdg :: Individual, vd :: Individual, iops :: Long )
  goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/governing#mdg_goverened_vdisk>
       ( mdg :: Individual, vd :: Individual, gov :: Long )
  goal <http://wwwvs.cs.hs-rm.de/bvqmanagement/governing#mdg_strongest_govered_vdisk>
       ( mdg :: Individual, vd :: Individual, gov :: Long )

  Policy "Govern VDisk"
  forall ?factor ?stepping ?mdg ?vdIops ?gov ?vd ?vid ?limit ?mdgIops ?treshold ?ac ?gh .
  if and (
    let ?factor = 0.75
    let ?stepping = 1000
    let ?treshold = 0.95

    -- Find a pool above capacity
    ?mdg [ ext:maxIops -> ?limit ]
    paths:mdg_ior(?mdg, ?mdgIops)
    guard ( ?mdgIops > ?treshold * ?limit )

    -- Find the bronze vdisk with maximum I/Os
    gov:mdg_bronze_vdisk_with_max_iops ( ?mdg, ?vd, ?vdIops )
    ?vd [ dp:vdisk_id -> ?vid ]
    guard ( 0 < ?vdIops )

    -- Govern the IO/s to ?factor of the current IO/s
    let ?gov = toLong((?vdIops * ?factor) - (?vdIops * ?factor) % ?stepping)

    -- If the pool has not been governed within the last second
    not exists ?hint ?govvd . and (
```

```
47        paths:mdg_vdisks(?mdg, ?govvd)
48        :GoverenedHint ( ?hint, Document ( ?hint [ :vDisk -> ?govvd ] ) ) within 1 s
49      )
50
51      let ?ac = BNode(?vid, ?gov)
52      let ?gh = BNode(?vd)
53
54    ) then do (
55      execute :Govern (?ac, Document ( ?ac [
56          :vDiskId -> ?vid
57          :iops -> ?gov
58        ] ) )
59      assert :GoverenedHint (?gh,  Document ( ?gh [ :vDisk -> ?vd ] ) )
60    )
61
62    Policy "Relax Governing"
63    forall ?stepping ?mdg ?gov ?vd ?id ?factor ?newGov ?mdgIops ?limit ?treshold ?ac ?gh .
64    if and (
65      let ?factor = 1 / 0.75
66      let ?stepping = 1000
67      let ?treshold = 0.95
68
69      -- Find a pool below capacity
70      ?mdg [ ext:maxIops -> ?limit ]
71      paths:mdg_ior(?mdg, ?mdgIops)
72      guard ( ?mdgIops < ?treshold * ?limit )
73
74      -- With a governed VDisk
75      gov:mdg_strongest_govered_vdisk ( ?mdg, ?vd, ?gov )
76      ?vd [ dp:vdisk_id -> ?id ]
77
78      -- Calculate governing offset based on ?factor
79      let ?newGov = toLong((?gov * ?factor) - (?gov * ?factor) % ?stepping + ?stepping)
80
81      -- Check if MDiskGroup can handle the new governing rate
82      guard ( ?mdgIops - ?gov + ?newGov < ?treshold * ?limit )
83
84      -- If the pool has not been governed within the last second
85      not exists ?hint ?govvd . and (
86        paths:mdg_vdisks(?mdg, ?govvd)
87        :GoverenedHint ( ?hint, Document ( ?hint [ :vDisk -> ?govvd ] ) ) within 1 s
88      )
89
90      let ?ac = BNode(?id, ?newGov)
91      let ?gh = BNode(?vd)
92    ) then do (
93      -- Relax governing to ?newGov IO/s
94      execute :Govern (?ac, Document ( ?ac [
```

```
95          :vDiskId -> ?id
96          :iops -> ?newGov
97        ] ) )
98      assert :GoverenedHint (?gh, Document ( ?gh [ :vDisk -> ?vd ] ) )
99    )
100
101   Policy "Ungovern VDisk" forall ?gov ?vd ?vdiops ?vdid ?ac .
102   if and (
103     -- Find a governed VDisk below capacity
104     gov:mdg_goverened_vdisk ( _, ?vd, ?gov )
105     ?vd [ dp:vdisk_realtime_ior -> ?vdiops ]
106     guard ( ?vdiops < ?gov - 1000 )
107
108     -- Which has not been governed within 5s
109     not exists ?hint . and (
110         :GoverenedHint ( ?hint, Document ( ?hint [ :vDisk -> ?vd ] ) ) within 5 s
111     )
112
113     ?vd [ dp:vdisk_id -> ?vdid ]
114
115     let ?ac = BNode(?vdid)
116   ) then do (
117     -- Ungovern VDisk
118     execute :Ungovern (?ac, Document ( ?ac [ :vDiskId -> ?vdid ] ) )
119   )
120
121   Query "Bronze VDisk in MDisk Group"
122   forall ?mdg ?vd ?iops .
123   gov:mdg_bronze_vdisk ( ?mdg, ?vd, ?iops ) :- and (
124     paths:mdg_vdisks(?mdg, ?vd)
125     ?vd [
126       op:data_class -> ext:Bronze
127       dp:vdisk_realtime_ior -> ?iops
128     ]
129   )
130
131   Query "Bronze VDisk with max. IO/s in MDisk Group"
132   forall ?mdg ?vd ?iops .
133   gov:mdg_bronze_vdisk_with_max_iops ( ?mdg, ?vd, ?iops ) :- and (
134     -- Get VDisk on MDiskGroup
135     gov:mdg_bronze_vdisk ( ?mdg, ?vd, ?iops )
136     -- Ensure it's the VDisk with most IO/s
137     not exists ?otherIops . and (
138       gov:mdg_bronze_vdisk ( ?mdg, _, ?otherIops )
139       guard ( ?otherIops > ?iops )
140     )
141   )
142
```

```
  Query "Governed VDisk in MDisk Group"
  forall ?mdg ?gov ?vd .
  gov:mdg_goverened_vdisk ( ?mdg, ?vd, ?gov ) :- and (
    paths:mdg_vdisks(?mdg, ?vd)
    ?vd [ dp:gov_ratio -> ?gov ]
  )

  Query "Strongest Governed VDisk in MDisk Group"
  forall ?mdg ?gov ?vd ?vdiops .
  gov:mdg_strongest_govered_vdisk ( ?mdg, ?vd, ?gov ) :- and (
    -- Get governed vdisk above treshold
    gov:mdg_goverened_vdisk ( ?mdg, ?vd, ?gov )
    ?vd [ dp:vdisk_realtime_ior -> ?vdiops ]
    guard ( ?gov - 1000 < ?vdiops )
    -- Ensure it's the VDisk with most governing
    not exists ?othervd ?otherGov ?otheriops . and (
      gov:mdg_goverened_vdisk ( ?mdg, ?othervd, ?otherGov )
      ?othervd [ dp:vdisk_realtime_ior -> ?otheriops ]
      guard ( ?otherGov - 1000 < ?otheriops )
      guard ( ?otherGov < ?gov )
    )
  )

)
```

A.4 Fallstudie Radar-Management

A.4.1 Queries-Modul

```
module <http://www.dfs.de/orslm/Queries> (

  import "../resources/ontologies/TileSet.ttl"
  import "../resources/ontologies/ORSLM.ttl"

  prefix orslm: <http://www.dfs.de/ORSLM#>
  prefix ts: <http://www.dfs.de/TileSet#>
  prefix qs: <http://www.dfs.de/orslm/Queries#>

  goal <http://www.dfs.de/orslm/Queries#ActiveRadar> ( radar :: Individual)
  goal <http://www.dfs.de/orslm/Queries#ModeSRadar> ( radar :: Individual)
  goal <http://www.dfs.de/orslm/Queries#PSRRadar> ( radar :: Individual)
  goal <http://www.dfs.de/orslm/Queries#SSRRadar> ( radar :: Individual)

  goal <http://www.dfs.de/orslm/Queries#configuredTile> ( tile :: Individual )

```

```
  Query "Active Radar"
  forall ?rad .
  qs:ActiveRadar(?rad) :- ?rad [
    a -> ts:Radar
    orslm:sensorStatus -> orslm:SENSOR_STATUS_UP
  ]

  Query "Mode-S Radar"
  forall ?rad .
  qs:ModeSRadar(?rad) :- and (
    qs:ActiveRadar(?rad)
    ?rad [ orslm:modeSStatus -> orslm:SUBSENSOR_STATUS_ACTIVE ]
  )

  Query "PSR Radar"
  forall ?rad .
  qs:PSRRadar(?rad) :- and (
    qs:ActiveRadar(?rad)
    ?rad [ orslm:psrStatus -> orslm:SUBSENSOR_STATUS_ACTIVE ]
  )

  Query "SSR Radar"
  forall ?rad .
  qs:SSRRadar(?rad) :- and (
    qs:ActiveRadar(?rad)
    ?rad [ orslm:psrStatus -> orslm:SUBSENSOR_STATUS_ACTIVE ]
  )

  Query "Configured Tile"
  forall ?tile .
  qs:configuredTile(?tile) :- and (
    ?tile [ a -> ts:Tile ]

    not exists ?rad . and (
      ?tile [ ts:assignedRadar -> ?rad ]
      not ?rad [ orslm:sensorStatus -> _ ]
    )
  )

)
```

A.4.2 Plot-Mapping-Modul

```
module <http://www.dfs.de/orslm/Plots> (

  import "../resources/ontologies/Phoenix.ttl"
```

```
4   import "../resources/ontologies/TileSet.ttl"
5   import "../resources/ontologies/PhoenixMessages.ttl"
6
7   prefix rad: <http://www.dfs.de/Phoenix/Component/config/Tracker/RadarSet/Radar#>
8   prefix ts: <http://www.dfs.de/TileSet#>
9   prefix msg: <http://www.dfs.de/Phoenix/Messages#>
10
11  Correlation "Create Mapped Plot"
12  forall ?height ?width ?xOff ?yOff ?maxLayer ?phxRad ?tsRad ?msg ?mapped
13         ?sac ?sic ?name ?x ?y ?alt ?ix ?iy ?ialt ?tile ?layer ?column  .
14  if and (
15    _ [
16      ts:tileWidth -> ?width
17      ts:layerHeight -> ?height
18      ts:xOffset -> ?xOff
19      ts:yOffset -> ?yOff
20    ]
21
22    let ?maxLayer = maximum ?idx [?lay] . ?lay [
23      a -> ts:Layer
24      ts:index -> ?idx
25    ]
26
27    msg:PlotMsg (?msg, Document ( ?msg [
28        msg:sac -> ?sac
29        msg:sic -> ?sic
30        msg:x -> ?x
31        msg:y -> ?y
32        msg:altitude -> ?alt
33      ] ) ) within 5 s
34
35    let ?ix = toInteger((?x - ?xOff) / ?width) + 1
36    let ?iy = toInteger((?y - ?yOff) / ?width) + 1
37    let ?ialt = minimum(toInteger(?alt / ?height) + 1, toInteger(?maxLayer))
38
39    ?layer [
40      ts:index -> ?ialt
41      ts:hasColumn -> ?column
42    ]
43
44    ?column [
45      ts:index -> ?ix
46      ts:hasTile -> ?tile
47    ]
48
49    ?tile [ ts:index -> ?iy ]
50
51    ?phxRad [
```

```
        rad:hasSAC -> ?sac
        rad:hasSIC -> ?sic
        rad:hasName -> ?name
      ]

      ?tsRad [ ts:name -> ?name ]

      let ?mapped = BNode(?tsRad, ?tile, ?x, ?y, ?alt)
    ) then do (
      assert msg:MappedPlot(?mapped, Document ( ?mapped [
          msg:radar -> ?tsRad
          msg:tile -> ?tile
          msg:x -> ?x
          msg:y -> ?y
          msg:altitude -> ?alt
        ] ) )
      retract msg:PlotMsg (?msg, _)
    )

    Correlation "Only keep latest mapped plot"
    forall ?newPlot ?oldPlot ?radar ?tile .
    if and (
      msg:MappedPlot(?oldPlot, Document ( ?oldPlot [
          msg:radar -> ?radar
          msg:tile -> ?tile
        ] ) ) within 24h

      msg:MappedPlot(?newPlot, Document ( ?newPlot [
          msg:radar -> ?radar
          msg:tile -> ?tile
        ] ) ) after ?oldPlot

    ) then do ( retract msg:MappedPlot(?oldPlot, _) )

)
```

A.4.3 Radar-Status-Mapping-Modul

```
module <http://www.dfs.de/ORSLM/RadarStatus> (

  import "../resources/ontologies/Phoenix.ttl"
  import "../resources/ontologies/TileSet.ttl"
  import "../resources/ontologies/PhoenixMessages.ttl"
  import "../resources/ontologies/ORSLM.ttl"

  prefix rad: <http://www.dfs.de/Phoenix/Component/config/Tracker/RadarSet/Radar#>
```

```
9   prefix orslm: <http://www.dfs.de/ORSLM#>
10  prefix ts: <http://www.dfs.de/TileSet#>
11  prefix msg: <http://www.dfs.de/Phoenix/Messages#>
12
13  Mapping "Map Radar Status Messages"
14  forall ?phxRad ?tsRad ?sac ?sic ?name ?msg ?stat ?mode_s ?pr ?ssr
15         ?status ?modeSStatus ?psrStatus ?ssrStatus .
16  if and (
17    ?phxRad [
18      rad:hasSAC -> ?sac
19      rad:hasSIC -> ?sic
20      rad:hasName -> ?name
21    ]
22
23    ?tsRad [ ts:name -> ?name ]
24
25    msg:RadarStatusMsg(?msg, Document ( ?msg [
26        msg:sac -> ?sac
27        msg:sic -> ?sic
28        msg:status -> ?stat
29        msg:mode_s -> ?mode_s
30        msg:pr -> ?pr
31        msg:ssr -> ?ssr
32      ] ) )
33
34    ?status [ orslm:identifier -> ?stat ]
35    ?modeSStatus [ orslm:identifier -> ?mode_s ]
36    ?psrStatus [ orslm:identifier -> ?pr ]
37    ?ssrStatus [ orslm:identifier -> ?ssr ]
38  ) then do (
39    retract ?tsRad [
40      orslm:sensorStatus -> _
41      orslm:modeSStatus -> _
42      orslm:psrStatus -> _
43      orslm:ssrStatus -> _
44    ]
45    assert ?tsRad [
46      orslm:sensorStatus -> ?status
47      orslm:modeSStatus -> ?modeSStatus
48      orslm:psrStatus -> ?psrStatus
49      orslm:ssrStatus -> ?ssrStatus
50    ]
51    retract msg:RadarStatusMsg(?msg, _)
52  )
53
54 )
```

A.4.4 SLA-Modul

```
module <http://www.dfs.de/ORSLM/SLA> (

  import "../resources/ontologies/TileSet.ttl"
  import "../resources/ontologies/ORSLM.ttl"
  import "Queries.ontml"

  prefix dio: <http://wwwvs.cs.hs-rm.de/obmf/dio#>
  prefix orslm: <http://www.dfs.de/ORSLM#>
  prefix ts: <http://www.dfs.de/TileSet#>
  prefix qs: <http://www.dfs.de/orslm/Queries#>

  Score "PSR Violator Score"
  forall ?tile ?cnt ?hard .
  dio:HardSoftScore(?hard, 0) :- and (
    ?tile [ a -> orslm:PSRViolator ]

    -- Count Mode-S radars
    let ?cnt = count ?rad . and (
      ?tile [ ts:assignedRadar -> ?rad ]
      qs:PSRRadar(?rad)
    )

    -- Hard score is the difference between target and actual count
    let ?hard = 1 - ?cnt
  )

  Score "SSR Violator Score"
  forall ?tile ?cnt ?hard .
  dio:HardSoftScore(?hard, 0) :- and (
    ?tile [ a -> orslm:SSRViolator ]

    -- Count Mode-S radars
    let ?cnt = count ?rad .  and (
      ?tile [ ts:assignedRadar -> ?rad ]
      qs:SSRRadar(?rad)
    )

    -- Hard score is the difference between target and actual count
    let ?hard = 2 - ?cnt
  )

  Score "Mode-S Violator Score"
  forall ?tile ?cnt ?hard .
  dio:HardSoftScore(?hard, 0) :- and (
    ?tile [ a -> orslm:ModeSViolator ]

```

```
47      -- Count Mode-S radars
48      let ?cnt = count ?rad .  and (
49        ?tile [ ts:assignedRadar -> ?rad ]
50        qs:ModeSRadar(?rad)
51      )
52
53      -- Hard score is the difference between target and actual count
54      let ?hard = 2 - ?cnt
55    )
56
57    Dependency "PSR Violator"
58    forall ?tile ?cnt .
59    ?tile [ a -> orslm:PSRViolator ] :- and (
60      qs:configuredTile(?tile)
61
62      -- Count PSR radars
63      let ?cnt = count ?rad .   and (
64        ?tile [ ts:assignedRadar -> ?rad ]
65        qs:PSRRadar(?rad)
66      )
67
68      -- Check if less than 1 PSR radar is assigned
69      guard ( ?cnt < 1 )
70    )
71
72    Dependency "SSR Violator"
73    forall ?tile ?cnt .
74    ?tile [ a -> orslm:SSRViolator ] :- and (
75      qs:configuredTile(?tile)
76
77      -- Count SSR radars
78      let ?cnt = count ?rad . and (
79        ?tile [ ts:assignedRadar -> ?rad ]
80        qs:SSRRadar(?rad)
81      )
82
83      -- Check if less than 2 SSR radars are assigned
84      guard ( ?cnt < 2 )
85    )
86
87    Dependency "Mode-S Violator"
88    forall ?tile ?cnt .
89    ?tile [ a -> orslm:ModeSViolator ] :- and (
90      qs:configuredTile(?tile)
91
92      -- Count Mode-S radars
93      let ?cnt = count ?rad .  and (
94        ?tile [ ts:assignedRadar -> ?rad ]
```

```
        qs:ModeSRadar(?rad)
      )

      -- Check if less than 2 Mode-S radars are assigned
      guard ( ?cnt < 2 )
    )

)
```

A.4.5 Reconfiguration-Modul

```
module <http://www.dfs.de/ORSLM/Reconfiguration> (

  import "../resources/ontologies/TileSet.ttl"
  import "../resources/ontologies/PhoenixMessages.ttl"
  import "../resources/ontologies/ORSLM.ttl"
  import "Queries.ontml"

  prefix orslm: <http://www.dfs.de/ORSLM#>
  prefix msg: <http://www.dfs.de/Phoenix/Messages#>
  prefix ts: <http://www.dfs.de/TileSet#>
  prefix qs: <http://www.dfs.de/orslm/Queries#>
  prefix rc: <http://www.dfs.de/ORSLM/Reconfiguration#>

  goal <http://www.dfs.de/ORSLM/Reconfiguration#candidate>
       ( tile :: Individual, radar :: Individual )

  Suggestion "Suggest Radar Assertion for Mode-S-Violator"
  forall ?ac ?tile ?rad  ?cnt .
  msg:AssertRadar(?ac, Document ( ?ac [
      msg:radar -> ?rad
      msg:tile -> ?tile
    ] ) ) :- and (
    -- Find Mode-S-Violator
    ?tile [ a -> orslm:ModeSViolator ]

    -- Check that the tile has less than 6 assigned radars
    let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
    guard ( ?cnt < 6 )

    -- Find a Mode-S radar
    rc:candidate(?tile, ?rad)
    qs:ModeSRadar(?rad)

    let ?ac = BNode(?rad, ?tile)
  )
```

```
36
37  Suggestion "Suggest Radar Assertion for PSR-Violator"
38  forall ?ac ?tile ?rad ?cnt .
39  msg:AssertRadar(?ac, Document ( ?ac [
40      msg:radar -> ?rad
41      msg:tile -> ?tile
42    ] ) ) :- and (
43    -- Find PSR-Violator
44    ?tile [ a -> orslm:PSRViolator ]
45
46    -- Check that the tile has less than 6 assigned radars
47    let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
48    guard ( ?cnt < 6 )
49
50    -- Find a PSRRadar
51    rc:candidate(?tile, ?rad)
52    qs:PSRRadar(?rad)
53
54    let ?ac = BNode(?rad, ?tile)
55  )
56
57  Suggestion "Suggest Radar Assertion for SSR-Violator "
58  forall ?ac ?tile ?rad ?cnt .
59  msg:AssertRadar(?ac, Document ( ?ac [
60      msg:radar -> ?rad
61      msg:tile -> ?tile
62    ] ) ) :- and (
63    -- Find SSR-Violator
64    ?tile [ a -> orslm:SSRViolator ]
65
66    -- Check that the tile has less than 6 assigned radars
67    let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
68    guard ( ?cnt < 6 )
69
70    -- Find a SSRRadar
71    rc:candidate(?tile, ?rad)
72    qs:SSRRadar(?rad)
73
74    let ?ac = BNode(?rad, ?tile)
75  )
76
77  Interpretation "Interpret Radar Assertion"
78  forall ?ev ?tile ?radar .
79  if and ( msg:AssertRadar(?ev,
80      Document ( ?ev [
81        msg:radar -> ?radar
82        msg:tile -> ?tile
83      ] ) ) ) then do (
```

```
    assert ?tile [ ts:assignedRadar -> ?radar ]
    retract msg:AssertRadar(?ev, _)
)

Suggestion "Suggest Radar Retraction for Mode-S-Violator"
forall ?ac ?tile ?rad ?cnt .
msg:RetractRadar(?ac, Document ( ?ac [
    msg:radar -> ?rad
    msg:tile -> ?tile
  ] ) ) :- and (
  -- Find Mode-S-Violator
  ?tile [ a -> orslm:ModeSViolator ]

  -- Check that the tile has at least 6 assigned radars
  let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
  guard ( ?cnt >= 6 )

  -- Find an assigned non-Mode-S radar
  ?tile [ ts:assignedRadar -> ?rad ]
  not qs:ModeSRadar(?rad)

  let ?ac = BNode(?rad, ?tile)
)

Suggestion "Suggest Radar Retraction for PSR-Violator"
forall ?ac ?tile  ?rad ?cnt .
msg:RetractRadar(?ac, Document ( ?ac [
    msg:radar -> ?rad
    msg:tile -> ?tile
  ] ) ) :- and (
  -- Find PSR-Violator
  ?tile [ a -> orslm:PSRViolator ]

  -- Check that the tile has at least 6 assigned radars
  let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
  guard ( ?cnt >= 6 )

  -- Find an assigned non-PSR radar
  ?tile [ ts:assignedRadar -> ?rad ]
  not qs:PSRRadar(?rad)

  let ?ac = BNode(?rad, ?tile)
)

Suggestion "Suggest Radar Retraction for SSR-Violator"
forall ?ac ?tile  ?rad ?cnt .
msg:RetractRadar(?ac, Document ( ?ac [
    msg:radar -> ?rad
```

```
132         msg:tile -> ?tile
133       ] ) ) :- and (
134       -- Find SSR-Violator
135       ?tile [ a -> orslm:SSRViolator ]
136
137       -- Check that the tile has at least 6 assigned radars
138       let ?cnt = count ?ass . ?tile [ ts:assignedRadar -> ?ass ]
139       guard ( ?cnt >= 6 )
140
141       -- Find an assigned non-SSR radar
142       ?tile [ ts:assignedRadar -> ?rad ]
143       not qs:SSRRadar(?rad)
144
145       let ?ac = BNode(?rad, ?tile)
146     )
147
148     Interpretation "Interpret Retraction Replacement"
149     forall ?ev ?tile ?rad .
150     if and ( msg:RetractRadar(?ev, Document ( ?ev [
151           msg:radar -> ?rad
152           msg:tile -> ?tile
153         ] ) ) ) then do (
154       retract ?tile [ ts:assignedRadar -> ?rad ]
155       retract msg:RetractRadar(?ev, _)
156     )
157
158     Query "Radar Candidate"
159     forall ?tile ?radar ?plot .
160     rc:candidate(?tile, ?radar) :- and (
161       ?tile [ a -> ts:Tile ]
162       -- That has created plots for the radar
163       msg:MappedPlot(?plot, Document ( _ [
164           msg:radar -> ?radar
165           msg:tile -> ?tile
166         ] ) ) within 24h
167
168       not ?tile [ ts:assignedRadar -> ?radar ]
169     )
170
171 )
```

Lebenslauf

Person

Name	Fabian Meyer
Geboren	22.02.1986 in Erlangen
Staatsangehörigkeit	Deutsch

Bildung

07/2012 – 03/2017	**Promotionsstudium**, Johannes Gutenberg-Universität Mainz, Mainz. Institut für Informatik Fachbereich Physik, Mathematik und Informatik
09/2008 – 09/2010	**Master of Science**, Hochschule RheinMain, Wiesbaden. Studiengang Informatik Fachbereich Design Informatik Medien
09/2005 – 09/2008	**Bachelor of Science**, Fachhochschule Wiesbaden, Wiesbaden. Studiengang Angewandte Informatik Fachbereich Design Informatik Medien

Forschung

10/2010 – 03/2017	**Wissenschaftlicher Mitarbeiter**, Hochschule RheinMain, Wiesbaden. Labor für Verteilte Systeme
12/2008 – 03/2010	**Wissenschaftliche Hilfskraft**, Hochschule RheinMain, Wiesbaden. Labor für Verteilte Systeme

Lehre

03/2011 – 03/2015	**Lehrbeauftragter**, Hochschule RheinMain, Wiesbaden. Praktika Verteilte Systeme und Computernetze
04/2011 – 09/2013	**Lehrbeauftragter**, Johannes Gutenberg-Universität Mainz, Mainz. Praktika Verteilte Systeme und Betriebssysteme
04/2012 – 04/2012	**Lehrbeauftragter**, Universität Brasov, Brasov (Rumänien). Blockpraktikum Verteilte Systeme
09/2007 – 03/2008	**Tutor**, Fachhochschule Wiesbaden, Wiesbaden. Praktikum Digitaltechnik

Veröffentlichungen

- Fabian Meyer und Reinhold Kroeger. „A framework for autonomic, ontology-based IT management“. In: *2015 11th International Conference on Network and Service Management (CNSM)*. IEEE, Nov. 2015, S. 78–84. ISBN: 978-3-9018-8277-7.
- Fabian Meyer. „Ontologie-basiertes Monitoring von IT-Systemen“. In: *44. Jahrestagung der Gesellschaft für Informatik, Workshop Management komplexer IT-Systeme und Anwendungen*. Hrsg. von Erhard Plödereder, Lars Grunske, Erik Schneider und Dominik Ull. Bd. P-232. Lecture Notes in Informatics (LNI) - Proceedings. Bonn: Gesellschaft für Informatik, Sep. 2014, S. 861–872. ISBN: 978-3-88579-626-8.
- Fabian Meyer, Reinhold Kröger und Morris Milekovic. „An approach for knowledge-based IT management of air traffic control systems“. In: *2013 9th International Conference on Network and Service Management (CNSM)*. IEEE, Okt. 2013, S. 345–349. ISBN: 978-3-901882-53-1.
- Andreas Textor, Fabian Meyer, Marcus Thoss, Jan Schäfer, Reinhold Kröger und Michael Frey. „An Architecture for Semantically Enriched Data Stream Mining“. In: *Proceedings of the First International Conference on Data Analytics*. Hrsg. von Sandjai Bhulai, Joseph Zernik und Petre Dini. Barcelona, Spain: IARIA, Sep. 2012. ISBN: 978-1-61208-242-4.
- Andreas Textor, Fabian Meyer und Reinhold Kröger. „Automated IT Management using Ontologies“. In: *International Journal on Advances in Intelligent Systems* 5.3/4 (Dez. 2012), S. 291–301. ISSN: 1942-2679.
- Fabian Meyer. „Kombination von Modellen zur Systemanalyse im Selbstmanagement-Kontext“. In: Electronic Communications of the EASST 37 (März 2011). ISSN: 1863-2122.
- Fabian Meyer. „Analyse von Event Streams im Kontext semantischer Systemmodelle“. In: *Informatiktage 2011*. Bd. S-10. Gesellschaft für Informatik e.V. (GI). Ahrstr. 45, 53175 Bonn, März 2011, S. 159–162. ISBN: 978-3-88579-444-8.
- Andreas Textor, Fabian Meyer und Reinhold Kröger. „Semantic Processing in IT Management“. In: *Proceedings of the Fifth International Conference on Advances in Semantic Processing (SEMAPRO)*. Hrsg. von Pascal Lorenz und Eckhard Ammann. Lisbon, Portugal: IARIA, Nov. 2011. ISBN: 978-1-61208-013-0.