Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1258

Dipl.-Ing. Lars Evertz,
Stuttgart

Entwicklung einer Systemstruktur zur einheitlichen Verwaltung von entitätsbezogenen Lebenszyklusdaten

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

„Entwicklung einer Systemstruktur zur einheitlichen Verwaltung von entitätsbezogenen Lebenszyklusdaten“

Von der Fakultät für Georessourcen und Materialtechnik
der Rheinisch-Westfälischen Technischen Hochschule Aachen

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

genehmigte Dissertation

vorgelegt von **Dipl.-Ing.**

Lars Evertz

aus Leverkusen.

Berichter: Univ.-Prof. Dr.-Ing. Ulrich Epple
Univ-Prof. Dr.-Ing. habil. Martin Wollschlaeger

Tag der mündlichen Prüfung: 01. Dezember 2017

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

Dipl.-Ing. Lars Evertz,
Stuttgart

Nr. 1258

Entwicklung einer Systemstruktur zur einheitlichen Verwaltung von entitätsbezogenen Lebenszyklusdaten

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Evertz, Lars
Entwicklung einer Systemstruktur zur einheitlichen Verwaltung von entitätsbezogenen Lebenszyklusdaten
Fortschr.-Ber. VDI Reihe 8 Nr. 1258. Düsseldorf: VDI Verlag 2018.
166 Seiten, 54 Bilder, 10 Tabellen.
ISBN 978-3-18-525808-4, ISSN 0178-9546,
€ 62,00/VDI-Mitgliederpreis € 55,80.

Für die Dokumentation: Systemstruktur – Entität – Lifecycle Management – Industrie 4.0 – Merkmale – Webservices – Wertschöpfungsketten – Datenverwaltung

Die Übergabe gedanklicher oder physischer Einheiten bildet die Basis für unternehmensübergreifende Wertschöpfung. Um diese gemeinsame Wertschöpfung optimieren zu können, müssen alle Beteiligten in der Lage sein, jederzeit auf alle diese Einheiten betreffenden Informationen zugreifen zu können; unabhängig davon, in welchem Verantwortungsbereich sich die Einheit gerade befindet. Dies ist ein Kernaspekt von Industrie 4.0. Zu seiner Umsetzung wird in diesem Beitrag eine Systemstruktur entwickelt, die ausgehend von Merkmalen und einer dienstbasierten Schnittstelle die globale Verfügbarkeit und Adressierbarkeit der Informationen bei einfacher Erweiterbarkeit sicherstellt. Ferner können Zusammenhänge zwischen verschiedenen Informationen modelliert werden, wobei die Modelle selbst mit den Informationen verknüpft werden. Die Entwicklungen werden anhand mehrerer Anwendungsfälle evaluiert.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter www.dnb.de abrufbar.

Bibliographic information published by the Deutsche Bibliothek
(German National Library)
The Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie (German National Bibliography); detailed bibliographic data is available via Internet at www.dnb.de.

D82 (Diss. RWTH Aachen University, 2017)

ISSN 0178-9546

ISBN 978-3-18-525808-4

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit am Lehrstuhl für Prozessleittechnik der RWTH-Aachen. An dieser Stelle möchte ich denen danken, die mich während dieses Unterfangens begleitet und die Zeit in Aachen zu einem sehr positiven Abschnitt meines Lebens gemacht haben.

An erster Stelle bedanke ich mich bei Herrn Professor Dr.-Ing. Ulrich Epple dafür, dass er diese Dissertation ermöglicht hat und mir in vielen, teils kontroversen Diskussionen andere Sichtweisen aufgezeigt und dabei so manches „Aha"-Erlebnis ausgelöst hat. Seine Führung und die dabei gegebenen Freiräume bildeten weiterhin die Grundlage für den Zusammenhalt und die gute Atmosphäre unter allen Mitarbeitern.
Ebenso bedanke ich mich bei Professor Dr.-Ing. habil. Martin Wollschlaeger, Inhaber der Professur für Prozesskommunikation der Technischen Universität Dresden, für die Übernahme der Rolle des Zweitgutachters.

Ferner bedanke ich mich bei meine Kollegen vom Lehrstuhl für den fachlichen und weniger fachlichen Austausch. Es hat immer Freude gemacht mit euch zu arbeiten. Besonderer Dank gebührt Andreas, Christian, Constantin, David und Kai für interessante Diskussionen und Feierabendrunden.
Außerdem gilt mein Dank Martina Uecker für die Unterstützung bei allen Fragen rund um Labor und Anlagen sowie Frau Bey und Frau Milescu-Huber für die Unterstützung bei den kleinen und großen organisatorischen Dingen.

Schlussendlich bedanke ich mich bei meinen Eltern Jutta und Willi und meinen Schwestern Nina und Sarah-Julia. Ihr habt mir diesen Weg ermöglicht und mich begleitet.

Stuttgart, im September 2017

Lars Evertz

„Je mehr ich lerne,
desto mehr merke ich,
wie wenig ich kann.“
ein weiser Trainingspartner - frei nach
Sokrates

Inhaltsverzeichnis

Abkürzungen und Symbole

AM	Asset Management.
API	Application programming interface (Schnittstelle zur Anwendungsprogrammierung).
AT	Automatisierungstechnik.
BLOB	Binary Large OBject (Datenobjekt unbekannter Struktur).
bspw.	beispielsweise.
bzw.	beziehungsweise.
CAD	Computer Aided Design.
CDD	Common Data Dictionary.
CPM	Core Product Model.
CRUD	Create, Read, Update, Delete.
d. h.	das heißt.
DIN	Deutsches Institut für Normung e. V.
DNS	Domain Name System.
DTD	Document type definition.
eOTD	ECCMA Open Technical Dictionary.
EPC	Electronic Product Code.
et al.	et altera - und weitere.
etc.	et cetera - und so weiter.
GUID	Globally unique identifier (global einzigartiger Identifikator).
HTTP	Hypertext Transfer Protocol.
HTTPS	Hypertext Transfer Protocol (HTTP) Secure (Übertragung von HTTP unter Nutzung des Transport Layer Security Protokolls).
I40	Industrie 4.0.
IANA	Internet Assigned Numbers Authority.
IEC	Internationale Elektrotechnische Kommission.
IEEE	The Institute of Electrical and Electronics Engineers.
IEV	International electrotechnical vocabulary (Internationales Elektrotechnisches Wörterbuch).

ISO	Internationale Organisation für Normung.
IT	Informationstechnologie.
MIME	Multipurpose Internet Mail Extensions.
NIST	National Institute of Standards and Technology.
OASIS	Organization for the Advancement of Structured Information Standards.
OLE	Objects Linking and Embedding.
ONC	Open Network Computing.
ONS	Object Naming Service.
OPC	OLE for Process Control.
OPC-UA	OPC Unified Architechture.
PAC	PEID Access Container.
PDKM	Product Data and Knowledge Management.
PDM	Product Data Management.
PEID	Product Embedded Information Device.
PLM	Product Lifecycle Management.
PMI	PROMISE Messaging Interface.
RAMI 4.0	Referenzarchitekturmodell für Industrie 4.0.
RFC	Request for Comments.
RNTD	RosettaNet Technical Dictionary.
RPC	Remote Procedure Call.
SOA	Service oriented Architecture (dienstorientierte Architektur).
SOAP	SOAP - Ein Übertragungsprotokoll für Nachrichten (ehemals Simple Object Acces Protocol).
SOM	Semantic Object Model.
STEP	Standard for the Exchange of Product Model Data (Standard für den Austausch von Produktdaten).
SVN	Subversion.
u. a.	unter anderem.
UML	Unified Modeling Language.
UMTS	Universal Mobile Telecommunications System.
UNSPSC	United Nations Standard Products and Services Code.
UPnP	Universal Plug And Play.
URL	Uniform Resource Locator.
VPN	Virtual Private Network.

W3C	World Wide Web Consortium.
WSDL	Web Service Description Language.
XDR	External Data Representation.
XML	Extensible Markup Language.
z. B.	zum Beispiel.

Kurzfassung

Wertschöpfung in der Industrie findet heutzutage in komplexen Netzwerken aus vielfältigen Unternehmen statt. Der Übergang von Waren und Dienstleistungen zwischen diesen Unternehmen ist daher alltägliche Praxis und stellt den Kernbereich der übergreifenden Interaktion dar. Durch gedankliche Verlängerung der Wertschöpfungsketten, wie sie im Rahmen der Initiative Industrie 4.0 identifiziert wurden, über ein einzelnes Unternehmen hinaus, werden die Komplexität der Verzahnung und der Austausch von (physischen oder gedanklichen) Einheiten als Knotenpunkte leicht erkennbar. Die Verzahnung ist jedoch was den Austausch von Informationen angeht wesentlich weniger weit fortgeschritten als es beim Austausch von Produkten der Fall ist. Der Informationsaustausch macht aber eine Optimierung der gemeinsamen Wertschöpfung erst möglich. Folglich bietet es sich für die Unternehmen an, Informationen, die mit den ausgetauschten Einheiten zusammenhängen, gegenseitig zugänglich zu machen. Ziel ist die Nutzbarmachung aller Informationen, die während der Existenz einer Einheit erfasst werden, für alle Beteiligten, da dies jedem einzelnen von ihnen die Optimierung seiner Wertschöpfungsprozesse erlaubt. Beispiele sind Informationen zur Nutzung einer Maschine, die ihrem Hersteller die Weiterentwicklung vereinfachen oder von Wartungsdienstleistern für die Erzeugung eines optimalen Wartungsplans genutzt werden. Auch für den Nutzer ergeben sich neue Möglichkeiten, wenn er mit Hersteller und Dienstleister gemeinsam seine eigenen Prozesse optimieren kann. Ein derartiger unternehmensübergreifender Informationsaustausch ist daher eine Zielvorstellung, gerade auch im Bereich Industrie 4.0.

Für diesen Austausch bedarf es gemeinsamer Modelle und Schnittstellen. Solche sind jedoch bislang nicht übergreifend vorhanden. Zwar gibt es vielversprechende Asset Management und Product Lifecycle Management Ansätze, doch sind diese entweder auf bestimmte Domänen oder gar Gegenstandstypen beschränkt, oder sind so abstrakt gehalten, dass ihre Umsetzungen möglicherweise nicht interoperabel sind. Daher wird in dieser Arbeit ein System für das Lifecycle Management von Einheiten entwickelt, das

1. In der Lage ist, Einheiten und ihre Eigenschaften über die gesamte Existenzphase zu verfolgen;

2. Die dynamische Anpassung der Modellierung der Einheiten erlaubt;

3. Durch ein Metamodell ein gemeinsames Verständnis aller modellierten Einheiten und Eigenschaften schafft und daher

4. Über Unternehmensgrenzen hinweg aufgebaut werden kann.

Dabei wird die Interoperabilität durch die Nutzung einer ebenfalls in dieser Arbeit spezifizierten Dienstschnittstelle erzeugt.

Die Abbildung des Lebenszyklus einer Einheit erfolgt durch die Erfassung ihrer inhärenten Eigenschaften zu diskreten Zeitpunkten. Diese sind im Rahmen von Erfassungs-

genauigkeiten widerspruchsfrei, da sie der Realität entsprechen. Scheinbar widersprüchliche Aussagen können aufgelöst werden, wenn die Erfassungsbedingungen bekannt sind.

Daher wird für die Verarbeitung der Lebenszyklusinformationen ein Merkmal-Ansatz gewählt, der diese Metainformationen aufnimmt und gleichzeitig die aufgenommenen Informationen ohne Vorwissen erkundbar macht. Die einfache Erweiterung der abgebildeten Eigenschaften einer Einheit ist ebenso möglich, wie die Abfrage der Bedeutung jedes gespeicherten Werts. Neben den Eigenschaften werden auch Zusammenhänge abgebildet.

Zur Adressierung jedes Abbildes und jedes Datums werden URL-basierte Identifikatoren genutzt. Diese erlauben weltweit eindeutige Identifikation und sind gleichzeitig aufgrund der vorhandenen Infrastruktur dezentral verwaltbar. Die Schnittstelle zur Abfrage aller Informationen basiert auf Web-Technologien und stellt bis zur Übertragungstechnologie spezifizierte Dienste zur Verfügung. Dadurch ist die Interaktion in heterogenen Systemlandschaften möglich.

Ausgehend hiervon werden die Komponenten eines Gesamtsystems und ihre Verkoppelung vorgestellt. Es gibt Komponenten zur Haltung von Semantikinformationen und zur Haltung der Lebenszyklusabbilder sowie aktive Komponenten, die die Informationen über die angesprochene Schnittstelle abfragen und weiter verarbeiten. Letztere können auch so gestaltet sein, dass sie die angesprochenen Zusammenhänge zwischen Eigenschaften oder Einheiten automatisch auswerten. Zwei solche Komponenten wurden für die Evaluierung des Ansatzes implementiert. Dabei wurden die entwickelten Informationsmodelle vollständig umgesetzt und die Schnittstelle so weit, wie es für die Verwaltung der Lebenszyklusinformationen notwendig ist. Die Schnittstellenteile, die der Erzeugung der gemeinsamen Semantikbasis dienen, wurden nicht umgesetzt, da für die Verwaltung der Lebenszyklusinformationen die Dienste ausreichen, die diese Basis abfragen. Die Beschreibungselemente für die semantische Basis wurden vor der Evaluierung manuell angelegt.

Anhand vierer Anwendungsfälle wurden Konzept und Implementierung evaluiert. Die Anwendungsfälle bilden die oben aufgelisteten Anforderungen ab und gehen in so weit darüber hinaus, als das auch auf die automatische Abbildung und Auswertung von Zusammenhängen zwischen Einheiten und ihren Eigenschaften eingegangen wird. Dies zeigt die Fähigkeit des Konzepts, die aufgenommenen Daten direkt und überall nutzbar zu machen. Die formale Abbildung von Zusammenhängen ist zwar systemspezifisch und daher nicht im Konzept festgelegt, die Art, wie die Informationen für solche Abbildungen zugänglich gemacht werden, ist jedoch spezifiziert. So können für verschiedene Anwendungsfälle die passenden Beschreibungssprachen gewählt werden. Der Zugriff auf die Einheitsinformationen erfolgt einheitlich und erlaubt die Erkundung der Eigenschaften der Einheiten zu beliebigen Zeitpunkten ihrer Existenz unabhängig vom Ort des Zugriffs, sofern das Vertrauen zwischen Anbieter und Nutzer der Informationen den Zugriff erlaubt. Hinsichtlich der Informationssicherheit können bewährte Methoden aus dem Bereich der Webtechnologien Anwendung finden. Insgesamt zeichnet sich der Ansatz durch hohe Flexibilität und Interoperabilität aus.

Abstract

Nowadays, industrial enterprises generate value within complex and heterogeneous networks. A core point of interaction between these enterprises is the exchange of goods and services. In the course of the initiative of „Industrie 4.0“ a number of core value chains within an enterprise were identified. Continuing all of these value chains over several enterprises, the aforementioned complex web becomes visible. Therein, the exchange of material and immaterial units marks the points of interaction. This interaction is far less advanced concerning the exchange of (meta-) information than concerning exchange of the products themselves. This poses a problem as metainformation is needed to optimize interdependent value generation. Hence, it seems promising for enterprises to share information concerning their exchanged things. The ultimate goal is to put every bit of information gathered throughout the lifecycle of each thing at the disposal of each party concerned with this thing. This enables the entrprises to optimize their value generation processes. For example manufacturers of machines or service contractors can take on information concerning the usage of a machine to improve it or to develop better maintenance plans respectively. Moreover the user of the machine gains the ability to improve his processes and scheduling based on additional information from his partners. Consequently, this cross enterprise sharing of (meta-) information is a major goal in „Industrie 4.0“.

To enable this exchange of information common models and interfaces are needed. There are several promising asset management and product lifecycle management concepts. However, these are either targeted at specific domains or even types of things, or they use such an abstract level of specification that bringing together their implementations will probably need huge efforts or even be impossible at all. Therefore in this Dissertation, the author develops a lifecycle management system that

1. can describe arbitrary things and their properties throughout their whole time of existence;

2. allows to dynamically change how these things are modelled (structure and properties);

3. uses a meta-model to create a common understanding how things are modelled and what meaning each property has and therefore

4. can be implemented and used across enterprises.

To ensure interoperability the system uses a service interface that is specified down to the technological level in this document.

The lifecycle of a unit is modelled by measuring the properties inherent to the unit at discrete points in time. The gathered information is - inaccuracies of measurement aside - without inconsistencies, as it depicts reality. Seemingly inconsistent data can be resolved if the environment and conditions of the measurements are known.

Hence, the concept to manage lifecycle data takes this meta-information into account. At the same time it allows for browsing the gathered information without previous knowledge of the unit. Furthermore it is possible to add or remove properties to be measured whithout interruption of the system. Querying a measured value's semantics is possible as well. In addition to properties, relationships can be modelled.

Each modelled unit and each datapoint is addressed with an URL-base identificator. This allows for globally unique identification. Moreover, existent infrastructure is used to manage these identificators efficiently and decentrally. The interface to all gathered information is web-based. It uses HTTP as the transfer protocol and specifies services in WSDL. Hence interaction between heterogeneous systems can be achieved easily.

Starting out with this concept, components and their interaction within a system are introduced. There are components to store and provide semantic information or life cycle data. Then, there are proactive components, which use the aforementioned interface to request information and work with it. The latter includes modelling or simulating the relationships between units and properties. Two of these components were implemented in order to evaluate the approach. The information model was implemented completely and the interface as far as necessarry to handle lifecycle information. Operations to build up the common semantic base were not implemented as it was sufficient to use the requesting side of this. The common semantic base was build up manually before the evaluation started.

Concept and implementation were evaluated with four use cases. These mirror the listed requirements and extent them to modelling and interpretation of relationships between units and properties. This shows how the gathered information can be used instantaneously and everywhere. The concept defines no formal way to describe relationships. However, it specifies how the data needed to interpret relationships can be acquired. Hence, it is possible to choose the right formalisms for the use cases at hand. Access to information works in a unified manner. It allows to browse units and their properties and query information about arbitrary points in their existence phase. It is only restricted by the parties holding the information and their mutual trust. To enforce this security of information, well-proven and widespread methods from the web-domain can be used. Alltogether, the concept shows a high grade of flexibility and interoperability.

1 Einführung

Die Optimierung der Wertschöpfung ist Ziel jedes Unternehmens. Im Zuge der Globalisierung und der immer engeren Vernetzung und Verzahnung der Wertschöpfungsprozesse zwischen verschiedenen Unternehmen kommt damit der Optimierung dieses Zusammenspiels eine besondere Bedeutung zu. Um eine solche Optimierung zu erreichen, müssen die wertschöpfungstechnisch verzahnten Unternehmen auch informationstechnisch vernetzt werden.

Bereits im Jahr 2003 wurden die Vorteile des unternehmensübergreifenden Datenaustauschs, der als „Schließung der Informationsschleifen im Produktlebenszyklus“ (übersetzt aus [71]) bezeichnet wurde, dargestellt. Es wurde festgestellt, dass ein solcher Informationsaustausch

- Herstellern alle Daten über die Nutzung und die Zustände seiner Produkte zur Verfügung stellt, so dass sie besagte Produkte optimieren können;
- Wartungstechnikern und Entsorgungstechnikern die Arbeit durch genaue Zustandsinformationen der Geräte erleichtert;
- Entwicklern die Möglichkeit gibt, die Erfahrungen der Anwender in die Entwicklung neuer Produkte einfließen zu lassen und
- Entsorgungsunternehmen Informationen über wertvolle Bestandteile ausgemusterter Einheiten liefert[71].

Diese Liste ist keinesfalls vollständig; insbesondere, da die Vorteile der Nutzer durch die Verfolgung der Einheiten nicht analysiert wurden. Sie gibt aber einen guten Eindruck der großen Menge der Nutznießer des unternehmensübergreifenden Austauschs von gegenstandsbezogenen Daten.

Im Rahmen der Initiative Industrie 4.0 wurden verschiedene Wertschöpfungsketten in Unternehmen identifiziert und beschrieben. Neben der eigentlichen Produktion werden der Anlagenbau, Anlagen- und Verfahrensentwicklung, Produkt(linien)entwicklung und weitere Aufgaben wie Vertrieb und Marketing angeführt. Abbildung 1.1 zeigt diese Ketten [87]. Es ist erkennbar, dass Wertschöpfung an ganz verschiedenen Stellen in einem Unternehmen stattfindet. Unter gedanklicher Verlängerung der Wertschöpfungsketten hin zu kooperierenden Unternehmen ergibt sich das in Abbildung 1.2 angeführte (unvollständige) Bild. Dabei stellen die dickeren Pfeile Übergänge von Dingen zwischen den Unternehmen dar. Aus Gründen der Übersichtlichkeit wurden hier nur Vorprodukte und andere physische Gegenstände betrachtet, die zwischen den Unternehmen wechseln. Nichtsdestoweniger beziehen sich die weiteren Ausführungen auch auf Planungsdokumente und weitere nichtmaterielle Dinge, die zwischen den einzelnen Gewerken der Unternehmen ausgetauscht werden.

Die ausgetauschten Dinge stellen die Kernobjekte der Interaktion zwischen den Unternehmen dar. Um die gemeinsame Wertschöpfung zu optimieren, bietet es sich daher

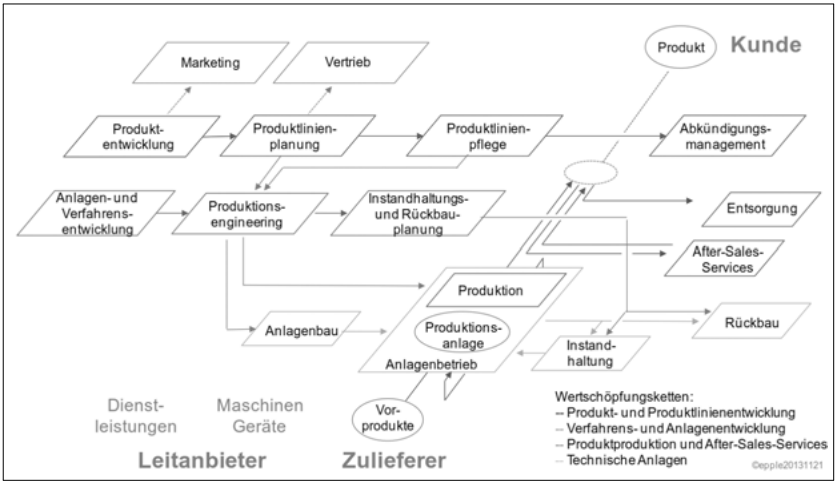


Abbildung 1.1: Wertschöpfungsketten in einem Unternehmen. [87]

an, diese Objekte zu verfolgen und zu verwalten. Derartig verwaltete Objekte werden als Entitäten bezeichnet. Sie befinden sich während ihrer Existenzphase in verschiedenen Verantwortungsbereichen und generieren oder tragen Informationen. Diese sind häufig nicht nur für denjenigen von Interesse, in deren Verantwortungsbereich sich die Entität befindet. Die Umgebungsbedingungen, die ein in einer Anlage eingebauter Sensor sieht, sind bspw. nicht nur für den verantwortlichen Anwender, sondern auch für den Hersteller interessant, um angepasste Wartungsanweisungen zu erzeugen. Aus diesem Grund ist es sinnvoll, Informationen während der gesamten Existenzphase einer Einheit mit den daran interessierten Parteien zu teilen, sofern die Vertraulichkeit der Informationen und das Vertrauen in die anderen Parteien dies erlauben. Die Entitäten sollen demnach dezentral den gesamten Lebenszyklus entlang verfolgt werden.

Mit einer derartigen dezentralen Verfolgung lassen sich große Mengen an Daten erheben, entitätsbezogen ablegen und unternehmensübergreifend verfügbar machen. Diese Daten erlauben die spätere Analyse, wobei zum Zeitpunkt der Datenaufnahme noch nicht klar sein muss, wie sie später eingesetzt werden. Dies ist letztlich eine Big-Data-Herangehensweise an die über den Lebenszyklus der Entitäten erfassten Daten. Um dies zu ermöglichen, bedarf es erstens einer einheitlichen Schnittstelle, die diese Daten verfügbar macht, und zweitens eines gemeinsamen Verständnisses der Semantik der erfassten Daten. Für letzteres muss zumindest die semantische Gleichheit zweier Daten maschinell ermittelbar sein. Dies muss ebenfalls über eine einheitliche Schnittstelle möglich sein. Um den Implementierungsaufwand gering zu halten, sollten für beide Schnittstellen die gleichen Prinzipien angewendet werden. Mit derartigen Schnittstellen können alle Objekte, die für eine Organisation interessant sind, unabhängig von ihrer Art auch über Unternehmensgrenzen hinaus verfolgt und verwaltet werden.

Bei der unternehmensübergreifenden Verwaltung von Entitäten treten einige Herausforderungen zu Tage. Die erste ist die Wahrung der Konsistenz der an verschiedenen

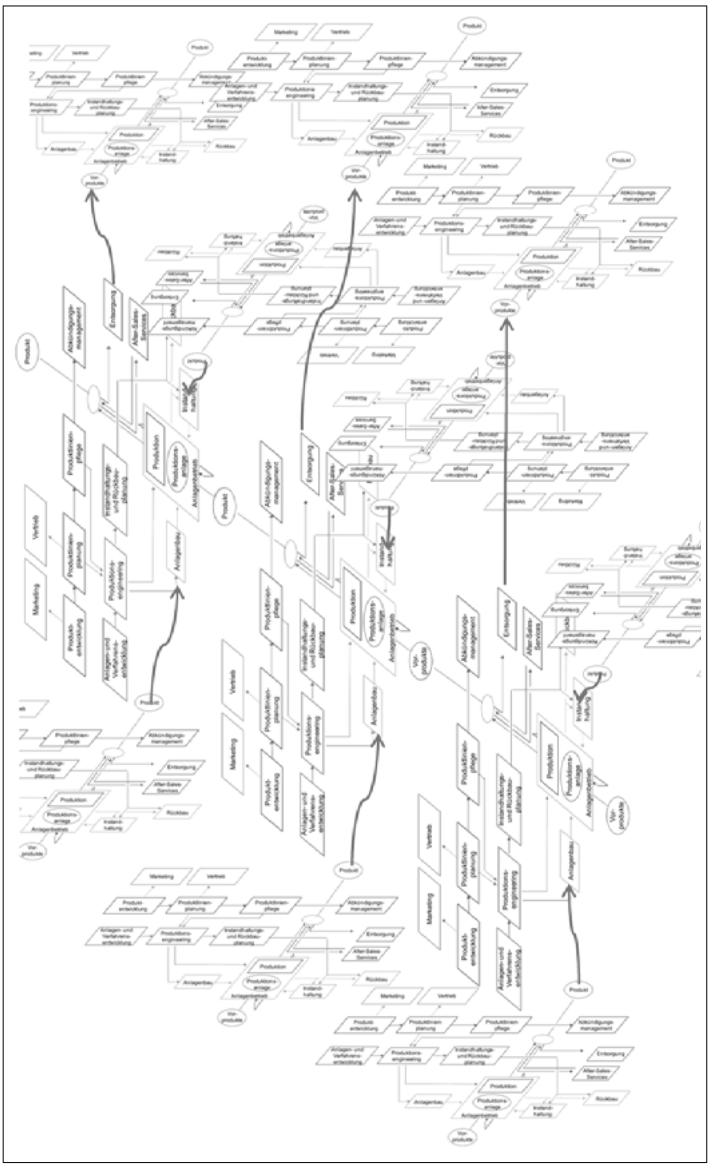


Abbildung 1.2: Gedanklich über das Unternehmen hinaus verlängerte Wertschöpfungsketten ergeben ein komplexes Netzwerk. Die Pfeile sind dabei Übergänge von Dingen zwischen den Unternehmen. Aus Gründen der Übersichtlichkeit wurde an jedem Punkt nur ein externes Unternehmen angeschlossen. Auch sind bei der Hinzunahme gedanklicher Dinge noch weitere Anknüpfungspunkte gegeben. Die Teilbilder der Wertschöpfungsketten entstammen [87].

Stellen verwalteten Informationen. Des Weiteren müssen auch Datenquellen angebunden werden können, die nicht zum Entitätenverwaltungssystem gehören. Dies ist notwendig, um einerseits doppelte Datenhaltung innerhalb eines Verantwortungsbereichs zu reduzie-

ren und andererseits, weil es je nach Art der Daten zusätzliche Anforderungen an ihre Übertragung gibt (bspw. sehr hohe Übertragungsraten und enge Zeitfenster), die spezialisierte Protokolle erforderlich machen, die die Entitätenverwaltung aufgrund eben dieser Spezialisierung selbst nicht anbieten kann. Zuletzt muss der Engineering Aufwand für eine Entitätenverwaltung in einem vertretbaren Rahmen bleiben. Daher ist die Möglichkeit der Automatisierung der Verwaltungsschritte vorzusehen. Dies betrifft insbesondere die Verkoppelung von verwalteten Einheiten und die Veränderung einer Einheit (z. B. eines Halbzeugs) durch eine andere Einheit (z. B. eine Werkzeugmaschine). Solche Zusammenhänge sollten abbildbar sein und automatisch zu festgelegten Verwaltungsvorgängen führen.

Die Implementierung eines System zur einheitlichen, unternehmensübergreifenden Verwaltung der entitätsbezogenen Daten unter Adressierung der genannten Herausforderungen dient als Grundlage zur Anwendung von Industrie 4.0 Paradigmen. Dabei ist das Hauptziel die Verbesserung der ökonomischen Leistung und Flexibilität durch erleichterte Entscheidungsfindung aufgrund der besseren Verfügbarkeit der notwendigen Informationen [99].

1.1 Aufbau der Dissertation

Die weitere Dissertation ist wie folgt strukturiert: In Kapitel **??** werden grundlegende Modelle und Technologien, die im Zusammenhang mit der Entitätenverwaltung stehen, vorgestellt. Anschließend werden in Kapitel 3 bestehende Lebenszyklus-Verwaltungsansätze dargestellt und im Hinblick auf die Anwendbarkeit für eine unternehmensübergreifende Entitätenverwaltung bewertet. Kapitel 4 setzt Lebenszyklen und ihre Modellierung in Bezug worauf aufbauend in Kapitel 5 ein Entitätenverwaltungssystem entwickelt wird. Dabei werden die grundlegenden Modelle und Interaktionsschnittstellen definiert und das Zusammenwirken der vorgeschlagenen Komponenten erläutert. Kapitel 6 stellt eine prototypische Implementierung der entwickelten Struktur vor. Mit diesem Prototyp wird das vorgeschlagene System anhand von Anwendungsbeispielen evaluiert. Kapitel 7 schließt die Dissertation mit einer Diskussion der Evaluation ab.

2 Grundlagen

Im folgenden Kapitel werden grundlegende Technologien und Ansätze vorgestellt, die im Rahmen der lebenszyklusbezogenen Verwaltung von Entitätsdaten interessant sein können. Zunächst werden Merkmalansätze als Form der Modellierung von Einheitseigenschaften vorgestellt. Daran schließen sich die Ansätze zur Modellierung von Einheiten an. Im nächsten Abschnitt wird auf die dienstbasierte Interaktion als Kommunikationsparadigma in lose gekoppelten Systemen eingegangen, um anschließend das Thema Produktdatenaustausch aufzugreifen. Schlussendlich werden die verschiedenen Ansätze zur Abbildung von Lebenszyklen vorgestellt.

2.1 Merkmale

Merkmale sind die Träger von Informationen zu charakteristischen Eigenschaften einer Entität. Ihre Ausprägungen lassen sich durch einfache Werte ausdrücken [49]. Sie eignen sich demnach zur Beschreibung von Systemen, wobei keine explizite Modellierung der Systemstruktur erfolgt, wie in Abbildung 2.1 dargestellt. Als klassifizierte Eigenschaften können Merkmale jedoch einen Überblick über wichtige Systemeigenschaften bieten [32]. Im Folgenden werden die relevanten Normen im Bereich der Merkmale kurz umrissen[1]. Anschließend wird auf die Anwendung von Merkmalen im Umfeld aktueller wissenschaftlicher Entwicklungen in der Automatisierungstechnik eingegangen.

Die Norm DIN 4000-1 [22] wurde in der ersten Version 1975 veröffentlicht und stellt die Grundlagen zur Gestaltung von Merkmal-Listen und Merkmalen zusammen. Sie definiert die Inhalte einer Merkmalbeschreibung, deren Zusammensetzung zu Listen und die Darstellung letzterer. Die DIN 4000-1 ist der Beginn einer umfassenden Normenreihe „Sachmerkmal-Listen“[2], die derartige Listen für verschiedene Gegenstandstypen zusammenstellt. Weitere Normen, wie bspw. ISO 11179-3 [63] und DIN EN 61360-1 [19], definieren die Bestandteile von Merkmalsdefinitionen und Klassifikationsschemata für Merkmale. Generell lässt sich festhalten, dass ein Merkmal immer eine Definition des Merkmals und einen eindeutigen Identifikator beinhaltet, der nach einem Klassifikationsschema zusammengesetzt sein kann. Dies ermöglicht die automatisierte Auswertung von Merkmalen.

Durch diese Eigenschaft können Merkmale zum elektronischen Austausch von Produktinformationen und deren automatisierter Auswertung eingesetzt werden. In der Normenreihe ISO 13584 [59] bspw. werden Grundlagen zur Beschreibung von Automatisierungskomponenten durch Merkmale festgelegt. Dabei werden die Informationsmodelle der DIN EN 61360 verwendet. Ebenso wird der Austausch der Produktdaten beschrieben. Auch die Normenreihe DIN 4002 [20] nutzt die in der DIN EN 61360 spezifizierten Modelle zum Produktdatenaustausch. Dabei werden Merkmale in eine Referenzhierarchie eingeordnet, die auch die Verwaltung in einem Merkmallexikon unterstützt.

[1] für eine detailliertere Zusammenfassung sei auf Heeg [49] verwiesen
[2] ehemals „Sachmerkmal-Leisten“

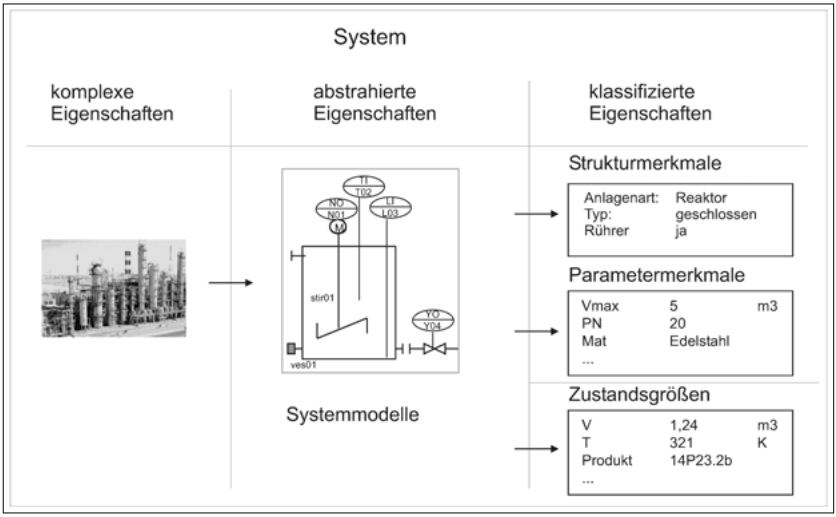


Abbildung 2.1: Abstraktion eines Systems als Strukturmodell und mit Hilfe klassifizierter Eigenschaften. [32]

Mertens stellt in seiner Dissertation [78] ein Metamodell zur Verwaltung merkmalbasierter Informationen vor. Dieses Modell ist in Abbildung 2.2 dargestellt. Es legt fest, dass der reale Merkmalträger außerhalb des Modells liegt und genau wie seine Repräsentation keine formale Beschreibung von Struktur oder Merkmalen besitzt. Diese wird auf der Ebene des Merkmalträgertyps gebildet. Er legt die allgemeinen Merkmale fest, die einen Merkmalträger beschreiben.

Ein Merkmaltyp ist eine neutrale Merkmaldefinition, wie sie bspw. in Merkmallexika zu finden ist. Um einen Merkmalträger beschreiben zu können, muss ein Merkmaltyp mit einer dem Merkmalträgertypen zugeordneten Semantik belegt werden. Dies wird auch als *Sachbezug* bezeichnet. Ist diese Semantik definiert, so entsteht ein allgemeines Merkmal[3]. Diese allgemeinen Merkmale sind nach einmaliger Definition fix und dürfen auf weiteren Vererbungsebenen von Merkmalträgertypen nicht spezialisiert werden. Grund hierfür ist, dass Mehrfach-Vererbung bei Merkmalträgertypen erlaubt ist und die semantische Definition der allgemeinen Merkmale eindeutig bleiben muss, um im Umgang mit abgeleiteten Merkmalträgertypen Konsistenz zu sichern.

Um mit Ausprägungen von Merkmalen umzugehen, werden in der Verwaltung Merkmalaussagen verwendet. Mit diesen können Informationen wie Istwerte, Anforderungen, Zusicherungen etc. festgehalten werden. Solche Aussagen können sich entweder auf einen Merkmalträger oder auf einen Merkmalträgertyp beziehen. Sie beziehen sich immer auf genau ein allgemeines Merkmal, wodurch die Semantik festgelegt ist. Aussagen sind typisiert. Es wird davon ausgegangen, dass innerhalb einer Domäne ein überschaubarer Satz solcher Aussagetypen ausreicht.

[3]Zu unterscheiden vom speziellen Merkmal, das dem realen Merkmalträger (außerhalb der Merkmalverwaltung) zugeordnet ist

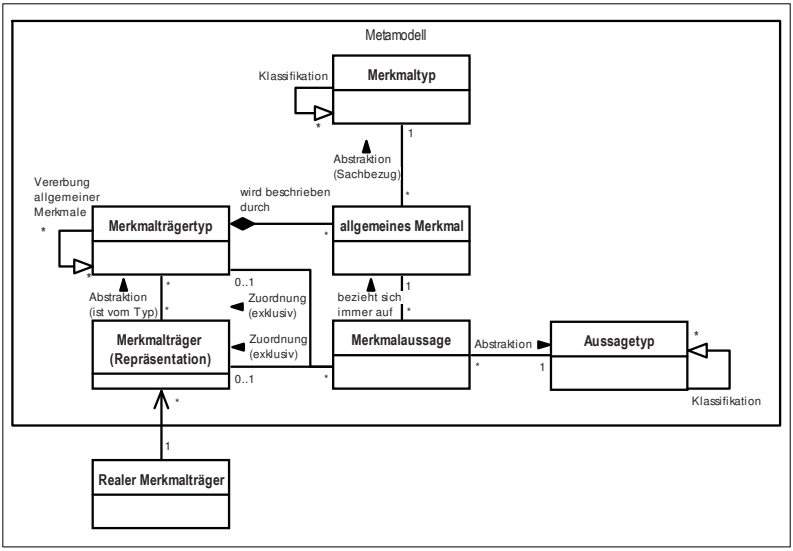


Abbildung 2.2: Metamodell zur Merkmalverwaltung nach [78]. Es ist zu beachten, dass der reale Merkmalträger außerhalb des Modells liegt.

Abbildung 2.3 zeigt am Beispiel des Merkmalträgertyps Kfz den Unterschied zwischen allgemeinen und speziellen Merkmalen. Spezielle Merkmale sind die Merkmale eines realen Systems gepaart mit ihren Ausprägungen. Dazu sei angemerkt, dass diese Ausprägungen in der Informationswelt nicht bekannt sind. Sie können nur durch Messung ermittelt werden. Anders herum kann dem System eine Instanz in der Informationswelt zugeordnet werden, die ihrerseits Informationen hält, die dem physischen System nicht bekannt sind, ihm aber zugeordnet werden. Dabei haben die besagte Beschreibungsinstanz und das physische System unabhängige Lebenszyklen [32].

Nach Epple können „klassifizierte Eigenschaften auf Grund ihrer Dynamik in Merkmale und Zustände“ eingeteilt werden [32]. Das bedeutet, dass die Eigenschaften, die sich im Betrachtungsintervall nicht ändern als Merkmale bezeichnet werden, während solche, die sich ändern, Zustände genannt werden. Diese Einteilung kann sich demnach je nach Betrachtungsintervall unterscheiden. Die Entscheidung, welche klassifizierte Eigenschaft als Merkmal und welche als Zustand angesehen wird und ob diese Unterscheidung im speziellen Fall gemacht werden muss, wird damit zur Designentscheidung.

Ein Konzept zum merkmalbasierten Austausch von Asset-Informationen wurde 2012 von Kampert vorgestellt [69]. Davon ausgehend, dass die verschiedenen Normen (wie [19, 59]) domänenspezifische Informationsmodelle bereitstellen und nicht auf ein generisches Modell für Assets ausgerichtet sind, schlägt er eine Kombination aus einem Informations- und einem Datenmodell vor. Dabei ist das Informationsmodell dem in [32] genannten kompatibel. Das Datenmodell basiert auf Objekten und attribuierten Relationen. Des Weiteren

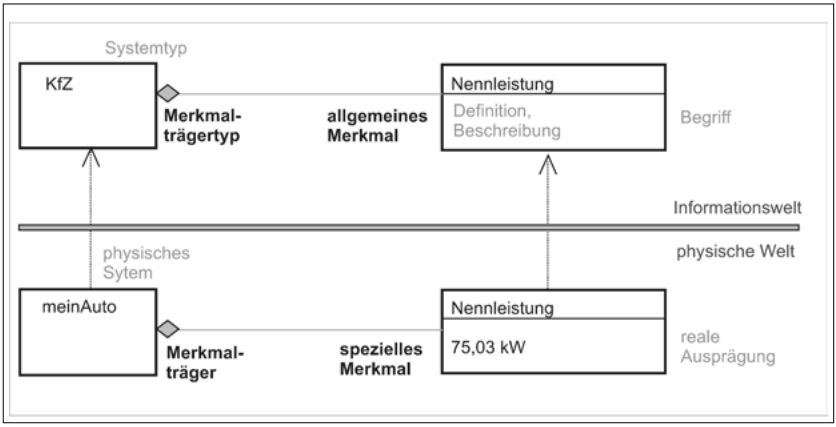


Abbildung 2.3: Zuordnung von Merkmalen, Merkmalträgern und deren Beschreibungen [32]

schlägt er den dienstbasierten Umgang mit derartigen Merkmalsystemen vor. Die daraus hervorgehenden Dienste stellt er in [70] vor.

Höme et al. schlagen vor, Merkmale zur semantischen Beschreibung von Diensttypen und Ressourcen zu nutzen, um diese von Maschinen interpretierbar zu machen. Sie beabsichtigen „eine linguistisch basierte Begriffsbildung auf das IEC 61360-Datenmodell aufzusetzen, um die Semantik der Dienstdaten maschinenlesbar zu machen." Diese Art der Modellierung wenden sie auch auf die Elemente des Produkt-, Prozess- und Ressourcen-Modells an, um eine semantisch eindeutige Beschreibung von Industrie 4.0-Komponenten zu erreichen. [51]

Eine ähnliche Idee verfolgt der Autor in [34]. Hier werden Merkmale zur semantisch eindeutigen Beschreibung von Dienste-Parametern benutzt. Dadurch kann die semantische Kompatibilität verschiedener Dienste automatisiert geprüft werden. Des Weiteren werden auf syntaktischer Ebene automatisch Adapter zwischen verschiedenen, semantisch kompatiblen Schnittstellen erzeugt. Dabei wird davon ausgegangen, dass die unterschiedlichen Syntaxen ihrerseits wohl definiert sind. Ist semantische Kompatibilität nicht gegeben, so wird der Prozess der Dienstintegration halbautomatisch durchgeführt.

In [35] beschreiben Fay et al. wie mit Hilfe von Merkmalen und weiteren Standards (IEC 62264 [55], Edifact [18], AutomationML [26], Formalisierte Prozessbeschreibung [100]) Anwendungsfälle aus dem Industrie 4.0 Bereich angegangen werden können. Dabei werden Merkmale als semantisch eindeutig definierte Basis zur Beschreibung von Produkteigenschaften verwendet, um in Verbindung mit den anderen semantischen Beschreibungen ad hoc Kommunikation in einer gemeinsamen Sprache[4] zwischen verschiedenen Systemen zu ermöglichen.

Hadlich geht in seiner Dissertation [47] auf die Verwendung von Merkmalen im Engineering von Anlagen ein. Dabei geht er davon aus, dass Systeme mit Merkmalen charakteri-

[4] im Duktus der Autoren von [35]

siert werden können, wie in Abbildung 2.4 dargestellt. Er verknüpft ein Merkmalmodell mit Strukturmodellen, um Komponenten, Prozesse und Funktionen einer Anlage und des Engineerings hinsichtlich ihrer Eigenschaften und ihrer Beziehungen darzustellen. Daraus entwickelt er eine erweiterte CAEX Beschreibung „CAEX++“, die den Engineeringprozess durch Bereitstellung der nötigen Daten unterstützt. Dabei werden zusätzliche Beziehungen innerhalb der Modelle und zwischen verschiedenen Modellen über besondere Merkmale ausgedrückt um bspw. Entwurfsentscheidungen zu dokumentieren. Dies erlaubt eine Verfolgung des Lebenszyklus des Entwurfs. Außerdem werden Referenzen auf externe Merkmaldatenbanken einbezogen um Komponenten etc. näher zu beschreiben.
Wird dasselbe System unter verschiedenen Aspekten betrachtet, so wird über *<RefSemanticClass>* Elemente die semantische Zusammengehörigkeit verschiedener Klassen abgebildet.

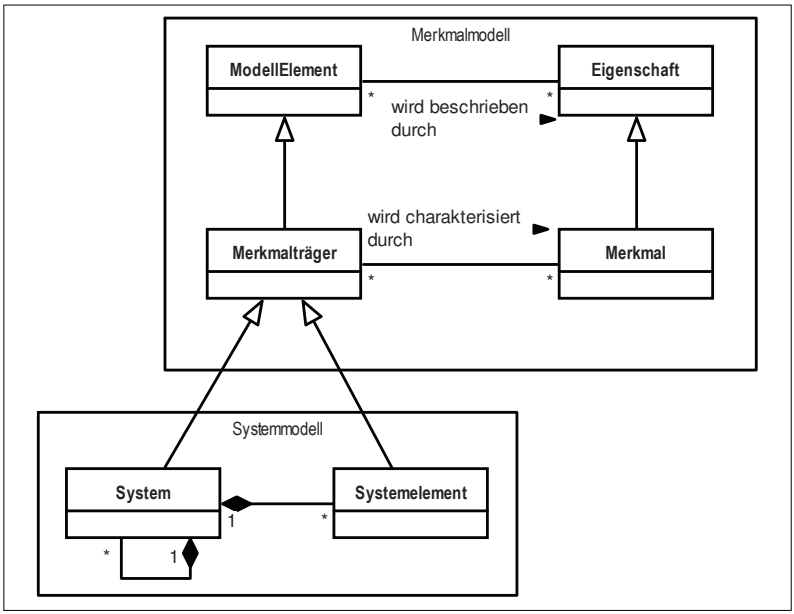


Abbildung 2.4: Merkmale charakterisieren Systeme und Systemelemente [47]

2.1.1 Merkmaldatenbanken und Klassifikationssysteme

Die Normenreihe IEC 60050[44][5] beinhaltet das International electrotechnical vocabulary (Internationales Elektrotechnisches Wörterbuch) (IEV). Sie führt Begriffe und ihre Definitionen mit eindeutigen, maschineninterpretierbaren Identifikatoren zusammen und kann so als Vorstufe einer Merkmaldatenbank angesehen werden. Es gibt eine online-Version des

[5]Der zitierte Teil 351 befasst sich mit den Begriffen der Leittechnik.

IEV[6], die URL-basierte Eingaben der Identifikatoren zu den Definitionen auflöst. Da im IEV Begriffe und Definitionen verwaltet werden, kann es bei der automatischen Semantikinterpretation helfen. Es stellt jedoch keine echte Merkmaldatenbank dar, da

1. nicht alle Begriffe klassifizierte Eigenschaften beschreiben,
2. die Begriffe nicht zwingend mit Werten assoziiert werden können,
3. die Interpretation von assoziierten Werten nicht definiert wird und
4. die definierten Merkmale nicht zwingend einen Sachbezug haben.

Nichtsdestotrotz stellt es durch die Web-basierte Zugriffsmöglichkeit eine wichtige Informationsquelle dar.

Hepp et al. untersuchten in [50] verschiedene Produktklassifizierungsstandards. Einige der untersuchten Standards (eCl@ss, ECCMA Open Technical Dictionary (eOTD) und RosettaNet Technical Dictionary (RNTD)) definieren dabei Merkmale der klassifizierten Produkte. In der Evaluation fällt dabei auf, dass die Definitionen oft nicht vollständig sind. Außerdem wird ein hohes Maß an Doppelungen vermutet. Trotzdem bieten die genannten Systeme eine Basis zur semantischen Beschreibung der Merkmale. Die Möglichkeit, diese Merkmale automatisch abzufragen, stellt dabei die Grundlage für die Verwendung der Systeme als semantische Datenbanken für verteilte Systeme dar.

In der aktuellen Version 9.0 beinhaltet eCl@ss 40.800 Produktklassen und 16.800 Merkmale [27]. Wie Fay et al. in [35] beschreiben, stellt es Merkmalbeschreibungen und Produktklassifikationen bereit, die nicht nur in der Beschaffungs-, sondern auch in der Engineering-Phase eines Systems hilfreich sein können. Dabei merken sie jedoch an, dass einige für Industrie 4.0 relevante Merkmale in eCl@ss noch fehlen.

Das Common Data Dictionary (CDD)[7][52] ist ein online verfügbares Repository der in der IEC 61360, IEC 61987 [21] und weiteren Normen spezifizierten Merkmale und Merkmalträger. Dabei werden Merkmale semantisch eindeutig definiert und den Merkmalträgertypen zugeordnet. Merkmale und Merkmalträgertypen werden natürlich sprachlich definiert und mit einem maschineninterpretierbaren Identifikator versehen. Das CDD ist nach den betreffenden Normen unterteilt und bildet diese ab. Doppelungen werden dabei derzeit innerhalb des CDD nicht aufgelöst. Auch die Güte und Vollständigkeit der den Merkmalträgertypen zugeordneten Merkmale ist je nach abgebildeter Norm verschieden.

Merkmalträgertypen und Entitätsdefinitionen

Eine Entitätsdefinition umfasst die sprachliche Definition einer Entität sowie die Darstellung der für einen Entitätstyp relevanten Merkmale. Wird eine Instanz eines Merkmalträgertyps in ihrem Lebenszyklus verfolgt, so wird der Merkmalträgertyp in diesem speziellen Fall zur Entitätsdefinition. Im Folgenden ist der einfacheren Lesbarkeit halber ausschließlich von Entitätsdefinitionen die Rede.

Die im vorigen Kapitel 2.1.1 genannte Evaluation von Hepp et al. ([50]) zeigt, dass bereits eine große Anzahl an Entitätstypen in Datenbanken definiert ist. Die evaluierten Datenbanken (eCl@ss, eOTD, RNTD und UNSPSC) decken jedoch jeweils nur einen

[6] zu finden unter erter http://www.electropedia.org

[7] Abrufbar unter http://std.iec.ch/cdd/iec61360/iec61360.nsf/TreeFrameset bzw. http://std.iec.ch/cdd/iec61987/iec61987.nsf/TreeFrameset für die entsprechenden Normen

domänenspezifischen Ausschnitt der Menge an Entitätstypen ab. Gleiches gilt für die im CDD abgebildeten Entitätstypen.

Wie erwähnt liefern die genannten Systeme Beschreibungen für Entitätstypen. Eine Beschreibung des Ist-Zustandes einer Entität ist damit also nicht gegeben. Diese erfolgt durch Ausprägungsaussagen zu den in den verschiedenen Systemen definierten Merkmalen der Entitäten. Das bedeutet, die in eCl@ss, eOTD, RNTD etc. bereitgestellten Merkmale können in einem weiteren Verwaltungssystem aufgenommen und für jede Instanz eines Entitätstyps mit Ausprägungsaussagen versehen werden. Damit stellen die Klassifikationssysteme eine gute semantische Basis für eine Verwaltung von Entitätsdaten bereit.

2.2 Entitätenmodelle

Das generische Modell (Kernmodell) einer Entität ist in Abbildung 2.5 dargestellt. Grundsätzlich ist eine Entität eine physische oder gedankliche Einheit (ein Gegenstand oder „Ding“), das individuell bekannt ist und in seinem Lebenszyklus verfolgt wird (vergleiche [25]). Diese Verfolgung des Lebenszyklus wird laut Kernmodell von einer Verwaltungseinheit in der Informationswelt durchgeführt. Es ist eine Designentscheidung, ob eine Einheit als Entität verwaltet wird, oder nicht. Lediglich die Möglichkeit der eindeutigen Identifikation der Einheit muss gegeben sein.

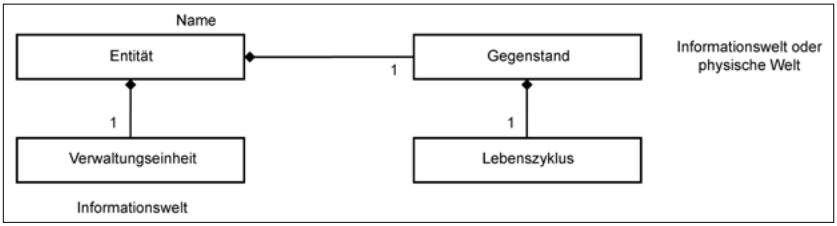


Abbildung 2.5: Modell einer Entität [25]

Laut Kernmodell müssen Entitäten nicht disjunkt sein. Eine Entität kann also (zeitweilig) Teil einer anderen Entität sein. Sie muss lediglich zu jedem Zeitpunkt eindeutig identifizierbar und ihrer Verwaltungseinheit zuordenbar sein. Es ist erkennbar, dass grundsätzlich jede Einheit zur Entität werden kann. Folglich ist es eine Designentscheidung, welche Gegenstände verwaltet und somit zu Entitäten werden.

Das Kernmodell kann als gemeinsamer Nenner der im Folgenden beschriebenen Entitätenmodelle angesehen werden. Es eignet sich als Beschreibung, um dem Leser ein allgemeines Verständnis von Entitäten zu vermitteln. Für eine Anwendung im Bereich des automatisierten Datenaustauschs ist es jedoch zu allgemein, da die einzelnen Inhalte der Verwaltungseinheit und ihr Bezug zum Gegenstand nicht definiert werden. Einige spezifischere Modelle werden im Folgenden vorgestellt.

2.2.1 Entitäten im Industrie 4.0 Verständnis

In seinem Statusreport „Gegenstände, Entitäten, Komponenten“ hat der VDI / VDE-GMA-Fachausschuss 7.21 eine Definition von Entitäten im Umfeld von Industrie 4.0 darge-

legt [86]. Dieses Verständnis orientiert sich stark an dem im letzten Abschnitt vorgestellten Kernmodell. Es wird zusätzlich explizit darauf hingewiesen, dass sowohl Gegenstände der physischen Welt, wie auch solche der Informationswelt gleichwertig als Entitäten verwaltet werden können, wobei sich die dafür zuständige Verwaltungseinheit (der „Ressourcenmanager“ [86]) immer in der Informationswelt befindet. Abbildung 2.6 zeigt dies.
Es werden vier Stufen der Bekanntheit eines Gegenstands in der Informationswelt definiert, wobei die Verwaltung als Entität die höchste Stufe darstellt. Diese geht über die individuelle Identifizierbarkeit hinaus und beinhaltet u. a. „Funktionen zur Gegenstandsverfolgung, zur Aufnahme von Lebenszyklusdaten zur operativen Steuerung des eigenen Produktionsprozesses und zur automatisierten Überwachung und Qualitätssicherung.“ Des Weiteren werden Gegenstände hinsichtlich ihrer Kommunikationsfähigkeit klassifiziert und angemerkt, dass aktive Kommunikationsfähigkeit die Verwaltung eines Gegenstands vereinfacht. Diesbezüglich wird im Referenzarchitekturmodell für Industrie 4.0 (RAMI 4.0) die Aussage getroffen, dass aktive Kommunikationsfähigkeit und die Bereitstellung spezifischer Dienste (Industrie 4.0-konforme Kommunikationsfähigkeit) es ermöglichen, die Verwaltungseinheit einer Entität im Gegenstand selbst unter zu bringen [88].

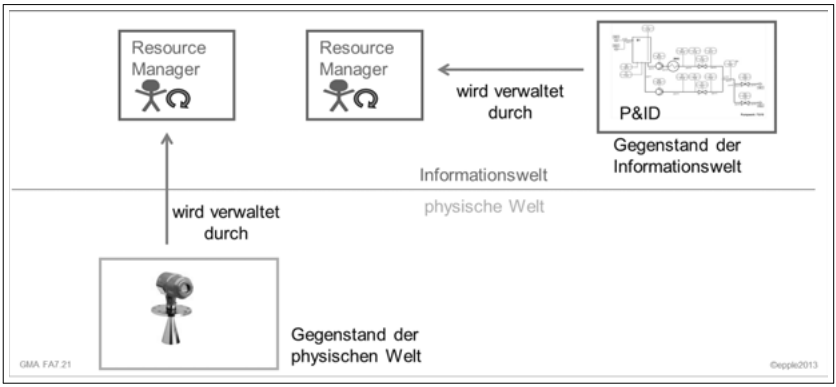


Abbildung 2.6: Im Umfeld von Industrie 4.0 werden Gegenstände der physischen Welt und solche der Informationswelt gleichwertig als Entitäten verwaltet.[86]

2.2.2 Core Product Model

Das Core Product Model (CPM) wurde vom National Institute of Standards and Technology (NIST) als generisches Datenmodell für die Verwendung in Product Lifecycle Management (PLM) Applikationen (vergleiche auch Abschnitt 3.1) entwickelt. Es ist als generisches Modell mit ebenso generischer Semantik konzipiert. Folglich wird domänenspezifische Semantik erst bei der Implementierung des Modells integriert. Das CPM wurde in [39] beschrieben und in [38] erweitert. Im Folgenden wird die erweiterte Version aus [38] näher erläutert.

Das CPM ist als Rahmenwerk zur Modellierung von Produktinformationen gedacht und soll sämtliche von der Entwicklung bis zur Abkündigung des Produkts anfallenden Informationen modellieren können. Abbildung 2.7 zeigt eine Unified Modeling Language

(UML)-Darstellung des CPM. Durch die Klassifikation von Produktinformationen hinsichtlich (geometrischer) Form, Funktion (im Sinne von spezifiziertem Verhalten) und Verhalten und deren Verschachtelungsmöglichkeiten weist das Modell eine nicht unerhebliche Komplexität auf.

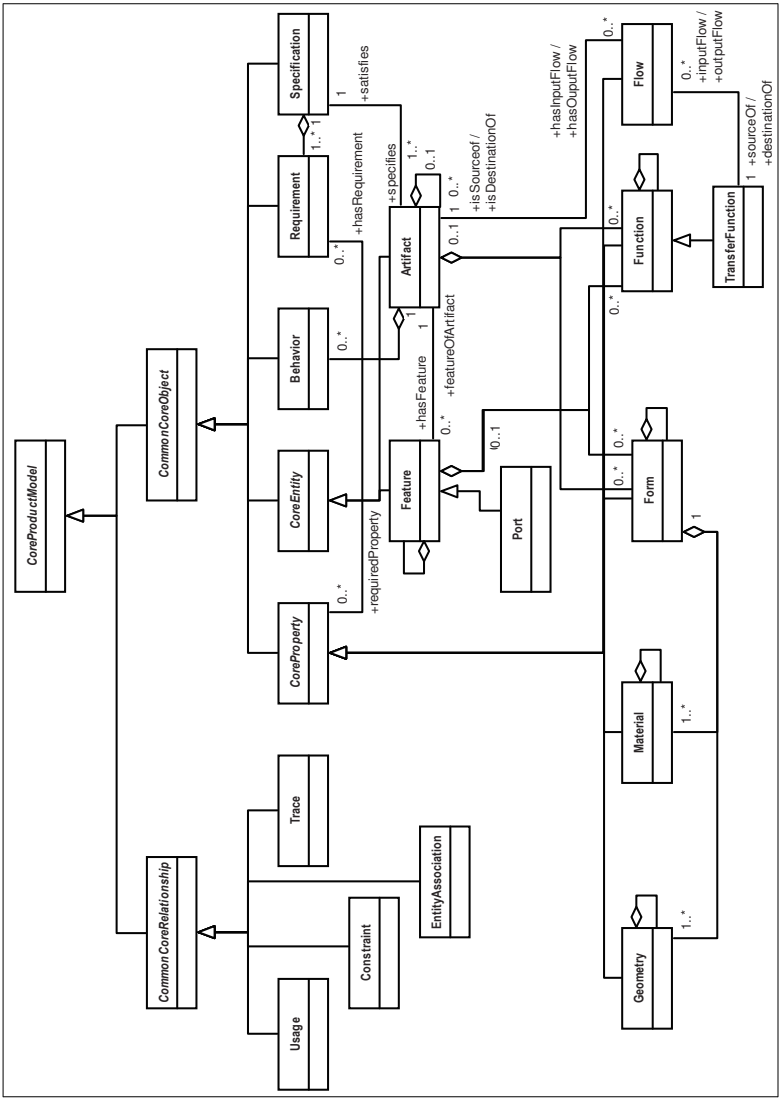


Abbildung 2.7: UML-Diagramm des CPM nach [38]. Auf die Bezeichnung von besonderen Aggregationen und die Feststellung, welche Assoziationen auf welche Klassen angewendet werden dürfen, wurde aus Gründen der Lesbarkeit verzichtet.

Das CPM nutzt die Prinzipien der Entity Relationship Modellierung. Dies führt zu

den Basisklassen *CommonCoreObject* und *CommonCoreRelationship* (vergleiche Abbildung 2.7). Da das CPM mit wenigen generischen Klassen auskommen soll, wurden diesen Klassen Container für domänen- und objektspezifische Informationen gegeben. Dies ist zum einen ein *type* genanntes Attribut für die Klassen-Objekte, das eine textuelle Klassifizierung erlaubt und zum anderen ein *information* genanntes komplexes Attribut der Instanzen, das eine textuelle Beschreibung, eine textuelle Dokumentation (bspw. eine Uniform Resource Locator (URL) oder andere Referenz auf ein Beschreibungsdokument) und eine Liste von Name-Wert-Paaren für weitere Attribute enthält.

Ein Ziel der Entwicklung des CPM war die Vereinfachung des Austauschs von Produktinformationen während der Entwicklungsphase eines Produkts. Um dies zu erreichen, veröffentlichte das NIST ebenfalls in [38] eine Abbildung auf Extensible Markup Language (XML) und eine Implementierung in Java. Außerdem wurden verschiedene Erweiterungen dieses Modells entwickelt (z. B. [6, 37, 92, 101–103]). Diese beziehen sich alle auf die Handhabung von Produkten im Sinne von Produkttypen oder Produktfamilien und nicht auf die individuelle Instanz eines Produkts.

2.2.3 Modell der ISO 15926

Die Normenreihe ISO 15926 befasst sich mit der Integration von Lebenszyklusdaten für verfahrenstechnische Anlagen, insbesondere der Öl- und Gasindustrie. Im ersten Teil [61] wird ein Überblick über die Konzepte der Norm gegeben. Sie soll ein generisches Datenmodell der Dinge, die im Zusammenhang mit einer Anlage stehen können, bereitstellen, um alle Lebenszyklus-bezogenen Aspekte abzubilden.

Das generische Datenmodell wird in [60] definiert. Dabei wird zwischen Klassen und Instanzen unterschieden. Letztere werden nochmals in tatsächliche Instanzen (*actual_individuals*) und mögliche Instanzen (*possible_individuals*) eingeteilt. Ein *possible_individual* drückt aus, dass besagte Instanz möglicherweise existierte, existiert oder existieren wird. Der letzte Fall ist im Speziellen für die Planung interessant, da hier die Absicht angezeigt werden kann, eine bestimmte Instanz einzusetzen. Jede Instanz wird durch einen systemweit einzigartigen Identifikator, die *thing.id* identifiziert. Dabei bezieht sich die Eindeutigkeit auf ein Verwaltungssystem. Demzufolge kann dieselbe Instanz in verschiedenen Systemen unterschiedlich benannt sein oder verschiedene Instanzen denselben Identifikator in verschiedenen Systemen tragen [60].

Das generische Modell besteht aus einer Vielzahl einzelner Definitionen und bewegt sich auf einer sehr abstrakten Ebene. Um es für einen realen Fall zu nutzen, wird es mit Referenzdaten verknüpft, die die konkreten Gegenstände und Klassen mit ihren Verknüpfungen darstellen. Solche Referenzdaten werden in Bibliotheken angelegt. Diese werden in ISO-Standards festgelegt. Dabei legt die Reihe ISO 15926 fest, wie solche Bibliotheken entwickelt und validiert werden. Durch die geforderte Normung der Referenzdatenbibliotheken ist deren Erzeugung zeitaufwändig.

Verwaltung und Austausch von Daten kann entweder über STEP-Dateien (vergleiche ISO 10303 [58]) oder in einem Informationssystem erfolgen. Wird letzteres verwendet, so muss es eine API bereitstellen, die die Erzeugung, Löschung und Bearbeitung von Instanzen ermöglicht. Dabei müssen Schnittstellen nach den in ISO 10303 beschriebenen Methoden geprüft werden.

Um die Implementierung eines Systems nach ISO 15926 zu erleichtern, wurde in Teil 7 [62] eine Vorlagen-Methodik spezifiziert. Das generische Modell wird mit Ausdrücken in

Prädikatenlogik abgebildet, um es einfacher in Ontologien überführen zu können. Dies bedingt jedoch eine hohe Anzahl derartiger Ausdrücke, da Objekte mit variabler Menge an Kindobjekten nicht dargestellt werden können und statt dessen für verschiedene Möglichkeiten eigene Ausdrücke gebildet werden.

2.2.4 Modell der ISO 13584 [59]

Die Normenreihe ISO 13584 stellt eine Teilebibliothek für industrielle Automatisierungssysteme und Integration bereit. Teil 501 [59] befasst sich im Speziellen mit Messinstrumenten. Sie legt die Merkmale fest, die für die Beschreibung von Automatisierungskomponenten nötig sind. Die Modellierung der Merkmale erfolgt dabei mit der Sprache EXPRESS (definiert in ISO 10303-11).

Neben den klassenspezifischen Merkmalen wir jede Klasse durch die Elemente

- Code (Identifikator)
- Superclass (Basisklasse)
- Preferred Name
- Synonymous Name
- Visible Types
- Applicable Types
- Visible Properties
- Applicable Properties
- Definition
- Source Document of Definition
- Note
- Simplified Drawing
- Date of Original Definition
- Date of Current version
- Date of Current Revision
- Version Number
- Revision Number und
- Remark

beschrieben (Definitionen der Elemente in ISO 13584-42:1998). Außerdem sind die Klassen nach vorgegebenen Richtlinien hierarchisiert. Die ersten vier Ebenen sind dabei festgelegt. Falls die Merkmale der Klassen auf der vierten Ebene nicht übereinstimmen, werden Unterklassen gebildet. Die konkreten Teilebibliotheken werden über die ISO verwaltet, wobei Änderungen aufwändig sind.

2.3 Dienstbasierte Interaktion

Um den steigenden Anforderungen hinsichtlich Flexibilität in Unternehmen nachzukommen, werden, unter anderem in [93], Dienstsysteme vorgeschlagen. Diese ermöglichen eine lose Kopplung von Funktionalitäten und erleichtern somit die Rekonfiguration. Auch im Rahmen der Initiative Industrie 4.0 ist dienstbasierte Interaktion eine wichtige technologische und organisatorische Grundlage [88].

Der Grundsatz der dienstbasierten Interaktion ist der Zugriff auf eine (fremde) Fähigkeit über eine definierte Schnittstelle zur Erreichung eines Ziels [74]. Dieses Prinzip lässt sich in weiten Bereichen vom Pizza-Bringdienst über unternehmensübergreifende Dienstleistungen bis zum Angebot von IT-Fähigkeiten anwenden. Im Kern lassen sich alle dienstbasierten Systeme auf das in Abbildung 2.8 dargestellte Modell abbilden. Dieses Modell sagt aus, dass ein Dienst einen Diensttyp realisiert, der durch ein Dokument, die Dienstbeschreibung, definiert wird. Der Dienst selbst wird dabei in eine oder mehrere Operationen aufgespalten, die die eigentliche Funktionalität darstellen. Ein Benutzer kann, wenn er die Definition des Diensttyps kennt, den Dienst in Anspruch nehmen („Dienstaufruf als diskretes Ereignis“ in Abbildung 2.8). Ob der Nutzer dieses will, kann er anhand der Zusicherungen zu den Qualitätsmerkmalen des Dienstes entscheiden. Diese Qualitätsmerkmale werden von der Organisation, die den Dienst anbietet, garantiert und umgesetzt [25].

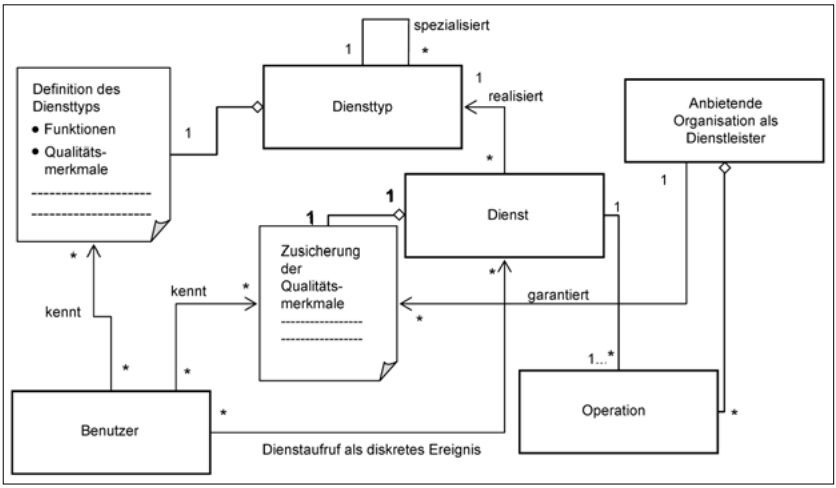


Abbildung 2.8: Kernmodell von Diensten. [25]

Im Zusammenhang mit der Verteilung und auftragsbasierten Nutzung von Fähigkeiten in einem Unternehmen wurde der Begriff der Service oriented Architecture (dienstorientierte Architektur) (SOA) geprägt [89]. Um derartige Architekturkonzepte auch informationstechnisch zu unterstützen, hat die Organization for the Advancement of Structured Information Standards (OASIS) 2006 ein Referenzmodell veröffentlicht [74]. Dieses ist bewusst abstrakt gehalten und stellt die übergreifenden Zusammenhänge bezüglich beschreibender Dokumente, Interaktionen, Dienst-Management (Richtlinien und Verträge) etc. dar.

Ein zentraler Punkt ist dabei die Dienstbeschreibung, die einen Dienst hinsichtlich aller relevanter Punkte beschreibt. Abbildung 2.9 zeigt die Verbindungen der Aspekte in einer dienstorientierten Architektur mit der Dienstbeschreibung. Die dargestellten Aspekte werden im Folgenden kurz erläutert. Die Ausführungen[8] basieren dabei soweit nicht anderweitig angegeben auf dem OASIS-Referenzmodell [74].

Die Interaktion zwischen Dienstnutzern und Dienstanbietern erfordert ihre gegenseitige Sichtbarkeit. Diese gliedert sich in die Aspekte der Kenntnis möglicher Interaktionspartner, der Bereitschaft zur Interaktion und der gegenseitigen Erreichbarkeit auf, wobei alle drei Aspekte für eine erfolgreiche Interaktion berücksichtigt werden müssen.

Die Auswertung der Beschreibung eines Dienstes stellt bei seinen potentiellen Nutzern Kenntnis her. Auf welchem Wege die Dienstbeschreibung dabei zu den Nutzern findet (bspw. durch direkt Übermittlung, als Teil einer Norm oder durch Erkundung der Struktur eines potentiellen Dienstanbieters) ist unerheblich. Die Kenntnis bildet die Grundlage, auf der ein Dienstnutzer über seine Bereitschaft zur Interaktion entscheidet. Dabei gehen in die Entscheidung sowohl die funktionalen Eigenschaften, als auch der weitere Nutzungskontext ein. Letzterer wird durch die in der Dienstbeschreibung dargestellten Richtlinien und Metriken festgelegt. Auch der Dienstanbieter trifft beim Aufruf eine Entscheidung über die Bereitschaft zur Interaktion aufgrund der ihm zur Verfügung stehenden Informationen zum Nutzer und aufgrund seines eigenen, aktuellen Zustands.

Die Möglichkeit der Interaktion mit einem Partner wird in diesem Kontext als Erreichbarkeit bezeichnet. Sie setzt das Wissen der Adresse des Interaktionspartners und des Interaktionsablaufs mit ihm voraus. Außerdem muss es einen Kommunikationskanal zwischen den Partnern geben.

Die Funktionalität eines Dienstes hat eine Auswirkung auf die reale Welt (*real world effect*). Dies kann sowohl eine Änderung des physischen Zustands bspw. in einer Anlage sein, als auch die Änderung eines Datenbankeintrags[9]. Funktionalität und Auswirkungen in der realen Welt sind in der Dienstbeschreibung dargestellt.

Die Interaktion mit einem Dienst besteht aus einer Abfolge von Kommunikationsaktionen und hat das Ziel, den angesprochenen Effekt in der realen Welt zu erzielen. Kommunikation erfolgt dabei über eine in der Dienstbeschreibung festgelegte Schnittstelle und besteht aus dem Austausch von Nachrichten. Die Beschreibung der Syntax und Semantik der Nachrichten bildet das Informationsmodell der Schnittstelle. Das Verhaltensmodell beschreibt, welche Aktionen durch die Schnittstelle ausgelöst werden können, welchen Effekt diese haben und was ihre Vorbedingungen sind.

Um das Verhalten eines dienstbasierten Gesamtsystems in den gewünschten Bahnen zu halten, werden von den Systemverantwortlichen anwendungsbezogene Einschränkungen getroffen. Dies geschieht durch Richtlinien und Verträge. Dabei sind Richtlinien einseitig festgelegt, während Verträge auf einer Übereinkunft zwischen den Interaktionspartnern basieren. Die gemachten Einschränkungen werden durch die Dienstbeschreibung bekannt gemacht. Richtlinien können neben der Interaktion mit dem Dienst selbst auch den physischen Kontext seiner Ausführung betreffen. Dabei legen sie „den Vertrag fest, den ein Dienstnutzer beim Aufruf einer Dienstfunktionalität mit dem Dienstanbieter eingeht“ [33], übersetzt aus [74]. Um die Einhaltung der Richtlinien beurteilen und wenn nötig durch-

[8] Eine ähnliche Zusammenfassung wurde vom Autor bereits in [17] veröffentlicht

[9] Die im OASIS-Referenzmodell genannte *reale Welt* ist die Summe der Informationswelt und der physischen Welt im I40-Verständnis [86]

setzen zu können, sind Metriken erforderlich, die die „besagte Einhaltung quantifizieren können. Diese Metriken können einerseits die Leistungsfähigkeit betreffen und andererseits die Voraussetzungen für die Interaktion bewerten. Welche Metriken für einen Dienst Anwendung finden wird in der Dienstbeschreibung dargestellt“ [33], übersetzt aus [74].

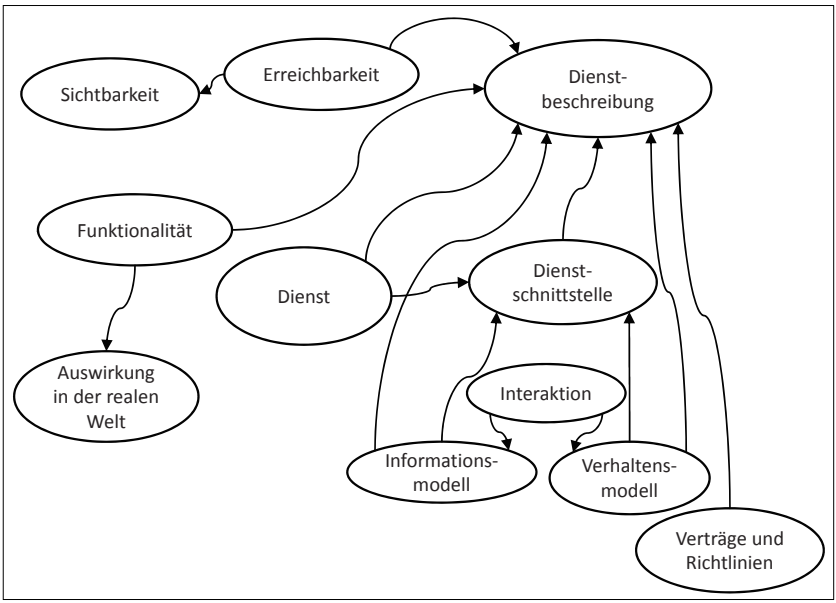


Abbildung 2.9: Elemente und ihre Verbindungen im Kontext der Dienstbeschreibung nach [74].

Dienste können auf vielfältige Weise implementiert werden. Im Informationstechnologie (IT)-Umfeld haben sich vornehmlich Webservices und Remote Procedure Calls (RPCs) etabliert. Im Automatisierungstechnik (AT)-Umfeld gewinnt derzeit OPC-UA [53] stetig an Bedeutung. In den Ausarbeitungen zum Thema Industrie 4.0 werden Dienste an vielen Stellen als Schnittstelle definiert [17, 86, 88]. Im Folgenden werden die angesprochenen Implementierungen und anschließend die Nutzung im Bereich Industrie 4.0 näher beschrieben.

2.3.1 Webservices

Webservice-Technologien werden von einer Arbeitsgruppe des World Wide Web Consortium (W3C) spezifiziert. Sie dienen der Interoperabilität in heterogenen Systemlandschaften. Webservices sind abstrakte Darstellungen einer bestimmten Funktionalität. Sie müssen von konkreten Implementierungen umgesetzt werden. Dabei können verschiedenste Implementierungen denselben Service darstellen [75].

Das technologische Fundament der Webservices bildet die WSDL [7, 14]. Ein WSDL-Dokument beschreibt einen Webservice. Dafür stellt es eine Reihe von Definitionen bereit.

Abbildung 2.10 zeigt die Elemente einer WSDL 1.1 Beschreibung.[14]

Sie beginnt mit einer *types*-Sektion. Diese definiert die für die weiteren Definitionen benötigten Typen. Um hier möglichst flexibel zu sein, werden XML-Schemata verwendet. An die *types*-Sektion schließen sich *message*-Sektionen an. Diese definieren den syntaktischen Aufbau jeweils einer Nachricht. Dabei wird auf die vorher bereitgestellten Typen zurückgegriffen.[14]

Darauf aufbauend definieren *portType*-Sektionen Sätze von Operationen. Diese werden als *operation*-Elemente dargestellt. Jeder Operation können maximal je eine *input* und *output* sowie mehrere *fault*-Nachrichten zugewiesen werden. Dabei werden die vorab definierten Nachrichten referenziert.[14]

Anschließend werden in *binding*-Sektionen die konkreten Datenformate und genutzten Protokolle festgelegt. Dabei werden so genannte *extensibility elements* verwendet, da hier auch der WSDL-Spezifikation unbekannte Technologien eingesetzt werden können. Innerhalb der WSDL-Spezifikation sind Bindings für SOAP[10] (siehe auch [46]), HTTP GET und POST (siehe auch [5]) und Multipurpose Internet Mail Extensions (MIME) (siehe auch [42]) definiert. Auch diese sind als *extensibility elements* umgesetzt.

In den abschließenden *service*-Sektionen werden Elemente des Typs *port* aggregiert, die die vorab definierten *bindings* mit konkreten Adressen verknüpfen. [14]

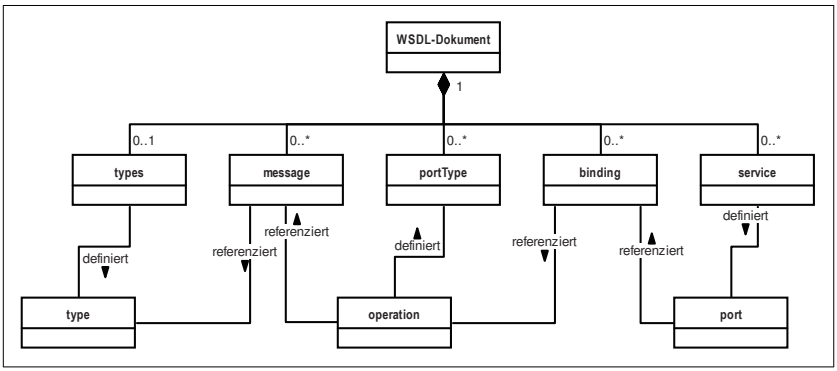


Abbildung 2.10: Elemente in einer WSDL-Beschreibung (Version 1.1).

Der Aufbau einer WSDL-Beschreibung lässt die Aufteilung in verschiedene Dateien zu und begünstigt somit die Wiederverwendung der einzelnen Definitionen durch Referenzierung. Alle Sektionen und Elemente können mit *documentation*-Elementen beschrieben werden. Auch *extensibility elements* können überall eingesetzt werden, um die Beschreibung über die Möglichkeiten, die WSDL allein bietet, hinaus zu erweitern. [14]

Es ist zu beachten, dass die WSDL-Spezifikation selbst nur syntaktische Elemente beschreiben kann. Um Semantik zu beschreiben sind verschiedene Erweiterungen wie bspw. WSMO-Lite [36] und WSDL-S [1] entwickelt worden.

In 2007 hat das W3C die neue Version 2.0 von WSDL als Empfehlung veröffentlicht [7]. In dieser Version wurde ein Komponentenmodell der Beschreibung eingeführt, dass das XML-Format der WSDL-Datei unterfüttert. Außerdem wurde in Zusatzdokumenten spezifiziert,

[10]Ehemals Akronym für „Simple Object Access Protocol“ seit Version 1.2 jedoch ein eigenständiger Begriff

wie die Abbildung der in WSDL beschriebenen Nachrichten auf ein Übertragungsmedium zu beschreiben ist. Diese Spezifikationen wurden anschließend genutzt um diese Abbildung, das so genannte „binding“, auf SOAP und HTTP zu spezifizieren. Bei letzterem werden alle HTTP-Operationen unterstützt [13]. WSDL 1.1 Dateien können nach Version 2.0 konvertiert werden. Der umgekehrte Weg ist aufgrund neuer Fähigkeiten nicht zwingend möglich. Die syntaktisch auffälligste Veränderung beim Versionssprung ist der Verzicht auf *message* Elemente. Statt dessen referenzieren Operationen nun direkt bestimmte Typen. Das *portType* Element wurde in *interface* und das *port* Element in *endpoint* umbenannt.

Die Nutzung von Webservices ist im Internet weit verbreitet. Im Bereich der Unternehmens-IT finden sie in vielen verbreiteten Softwareumgebungen Anwendung bei der Vernetzung der oberen Leitebenen (Betriebs- und Unternehmensleitung) [33]. Im Rahmen von Industrie 4.0 werden sie selten erwähnt, jedoch stellen sie eine gute Möglichkeit dar, heterogene Systeme lose zu koppeln. Ihre verbreitete Verwendung in der Unternehmens-IT macht sie außerdem zu einem gut in bestehenden Systemen einsetzbaren Werkzeug.

2.3.2 Remote Procedure Call (RPC)

RPCs werden zur Interprozesskommunikation eingesetzt. Die meist genutzte Implementierung, der Open Network Computing (ONC) RPC, ist in RFC 5531 [98] spezifiziert. Die Spezifikation betrifft dabei das Nachrichtenprotokoll.

ONC RPC-Nachrichten basieren auf dem External Data Representation (XDR) Format. Dieses ist in RFC 4506 spezifiziert [29]. Das XDR Format legt fest, wie bestimmte Datentypen zu übertragen sind. Aus derartigen Datenstrukturen werden ONC RPC-Nachrichten zusammengesetzt. RFC 5531 gibt vor, wie eine Aufrufnachricht grundlegend auszusehen hat. Es werden jedoch weder Dienste, noch Beschreibungsmethoden für sie definiert. Es wird nicht festgelegt, ob ONC RPCs synchron oder asynchron erfolgen sollen. Trotzdem wird das klassische, synchrone RPC-Modell als Beispiel angeführt.

Durch ONC RPC ansprechbare Dienste benötigen eine bei der Internet Assigned Numbers Authority (IANA) registrierte Programmnummer. Diese stellt sicher, dass der Aufruf dem Diensttyp eindeutig zugeordnet werden kann. Die auszuführende Operation wird durch eine in der Spezifikation des Dienstes definierte weitere Nummer festgelegt. Um Veränderungen der Schnittstelle eines Dienstes zu ermöglichen, wird weiterhin eine Versionsnummer übertragen. Alle drei Nummern werden als 32bit lange Zahlen im Kopf einer ONC RPC-Nachricht übertragen. [98]

Da es außer dem Protokoll keine weiteren Festlegungen für ONC RPCs gibt, können alle damit umgesetzten Dienste in einem beliebigen Format spezifiziert werden. Häufig erfolgt dies unter Verwendung der in RFC 4506 dargestellten Beschreibungssprache für die Nachrichtensyntax und natürlich sprachlich für weitere Beschreibungen.

Da es sich bei RPC um ein reines Übertragungsformat handelt, das zudem eher statisch ist, findet es sich explizit in den Diskussionen zu Industrie 4.0 kaum wieder. Gleichwohl kann es auch in diesem Bereich als Übertragungsmedium dienen.

2.3.3 ACPLT/KS

Das Kommunikationssystem ACPLT/KS wurde entwickelt, um eine generische Zugriffsschnittstelle auf die verschiedenen in der Prozessleittechnik verwendeten Modelle zu erzeugen. Dabei sollten Objektorientierung und die Möglichkeit der Selbstbeschreibung der

dargestellten Informationsquellen im Vordergrund stehen. Um dies zu erreichen, entwickelte Albrecht [2] ein Metamodell für die Elemente der Informationsübertragung. Dieses Metamodell besteht aus *Domain* genannten Containern, die weitere *Domains* und andere *Attributes* aggregieren können. *Attributes* können ihrerseits Variablen, Links, Historiendaten oder Strukturen sein. Dabei bilden Links Beziehungen zwischen *Domains* ab und erlauben so die Vernetzung der Container unter bestimmten Gesichtspunkten. Variablen sind einfache Attributwerte. Historiendaten bilden Werteabfolgen von Variablen ab. Strukturen sind aus mehreren Variablen komponierte Attribute.

Auf die genannten Modellelemente kann über definierte Dienste zugegriffen werden. Die Dienste sind als ONC RPCs spezifiziert und bilden die folgende Funktionalität ab:

- `KS_GETEP:` Erkundet die Kindelemente (über beliebige Links) eines Objekts. Es liefert alle Kinder mit ihren Typen, Kommentaren und semantischen Informationen.
- `KS_GETCANONICALPATH:` Gibt den kanonischen Pfad zu einem Objekt zurück. Dies ermöglicht die Identifikation eines Objekt, das über verschiedene Link-Beziehungen gefunden werden kann.
- `KS_GETVAR:` Gibt die aktuellen Werte einer oder mehrerer Variablen oder Strukturen zurück.
- `KS_SETVAR:` Setzt die Werte einer oder mehrerer Variablen oder Strukturen.
- `KS_EXGDATA:` Setzt die Werte einer oder mehrerer Variablen oder Strukturen und liest anschließend die aktuellen Werte einer oder mehrerer Variablen oder Strukturen aus
- `KS_GETHIST:` Liest ein Historienelement aus.
- `KS_SETHIST:` Setzt ein Historienelement.
- `KS_CREATEOBJECT:` Erzeugt ein Objekt (Domain, Variable, Historienelement, Struktur oder Link).
- `KS_DELETEOBJECT:` Löscht ein Objekt.
- `KS_RENAMEOBJECT:` Benennt ein Objekt um oder verschiebt es.
- `KS_LINK:` Erzeugt einen Link zwischen zwei Objekten.
- `KS_UNLINK:` Löscht einen Link zwischen zwei Objekten.

Außerdem gibt es noch die Dienste `KS_GETSERVER, KS_REGISTER` und `KS_UNREGISTER`. Diese dienen der Koexistenz mehrerer KS-Applikationen auf einer Plattform und haben keinen Bezug zum operativen Datenzugriff.

Mit den genannten Diensten können beliebige Daten-/Objektstrukturen verwaltet werden. Das ACPLT/KS Protokoll wird meist als Datenzugriffsschnittstelle zwischen der Prozessleitebene und der Betriebsleit- oder Unternehmensleitebene eingesetzt. Die dem Autor bekannten Implementierungen des KS-Protokolls sind synchron. Es selbst macht keine Festlegungen, wie weitere Dienste zu spezifizieren und zu implementieren sind. Nichtsdestotrotz gibt es eine Erweiterung, die die Möglichkeit bereitstellt, wahlfrei Nachrichten zu verschicken.

Die Nutzung von ACPLT/KS beschränkt sich im tatsächlichen Einsatz auf den Austausch von Betriebsdaten zwischen Prozessleit- und Betriebsleitebene. Die Fähigkeit Systeme mit ACPLT/KS strukturell zu erkunden und zur Laufzeit zu verändern, eröffnet weitere Möglichkeiten, die jedoch nach Information des Autors ausschließlich im wissenschaftlichen Umfeld Einsatz finden.

2.3.4 OPC-UA

OPC-UA ist ein Standard zum Vernetzen heterogener Systeme. Es ist in der Normenreihe IEC 62541 [53] spezifiziert. Die Kommunikation zwischen den Systemen kommt durch den Austausch von Nachrichten zustande, wobei definierte Dienste genutzt werden, die in einem Datenraum arbeiten, dessen Struktur ebenfalls in der Norm festgelegt wurde. Die folgenden Ausführungen basieren auf der OPC-UA-Spezifikation, im Speziellen auf [53, 54, 57].

OPC-UA definiert einen Adressraum aus Knoten, die sich im Verwaltungsbereich eines Servers befinden. Diese Knoten werden über eindeutige *NodeId*s adressiert. Dabei enthält eine *NodeId* einen Verweis auf einen Namensraum und einen *Identifier* innerhalb dieses Namensraums. Der *Identifier* kann dabei numerisch oder textbasiert sein. Außerdem ist eine Adressierung über einen Globally unique identifier (global einzigartiger Identifikator) (GUID) oder in einem namensraumspezifischen Format möglich.

Knoten können über Referenzen verknüpft werden. Dabei sind Referenzen in einer Typhierarchie definiert. Die Erkundung eines OPC-UA-Servers ist über alle Referenzen möglich, sodass verschiedene Sichten generiert werden können.

Jeder Knoten hat einen Typ. Es kann sich um Objekte, Variablen, Sichten, Methoden und Typdefinitionen von Variablen, Objekten, Daten und Referenzen handeln. Jeder dieser Knotentypen hat eine Reihe von Attributen, die in der Norm spezifiziert sind. Mit *ObjectType*-Knoten können neue Objekttypen definiert werden. Diesen können Variablen zugeordnet werden.

Der Zugriff auf einen OPC-UA-Server erfolgt über Dienste. Diese sind in Sätzen gruppiert. Mit den Daten in einem Server kann über die folgenden Dienste interagiert werden:

- *NodeManagement*-Satz
  - `AddNodes`: Fügt dem Adressraum einen oder mehrere Knoten hinzu.
  - `AddReferences`: Fügt Referenzen zwischen Knoten hinzu.
  - `DeleteNodes`: Entfernt einen oder mehrere Knoten aus dem Adressraum.
  - `DeleteReferences`: Entfernt eine oder mehrere Referenzen.
- *View*-Satz
  - `Browse`: Erkundet die Referenzen eines Knotens.
  - `BrowseNext`: Führt einen `Browse`- oder `BrowseNext`-Aufruf fort, falls die Antwort aufgrund ihrer Größe aufgeteilt werden muss.
  - `TranslateBrowsePathsToNodeIds`: Erfragt die *NodeId*s der durch einen Pfad angegebenen Knoten.
- *Query*-Satz

  - `QueryFirst`: Stellt eine generische Datenanfrage an einen Server (für detailliertere Informationen sei auf [54] verwiesen).
  - `QueryNext`: Führt eine `QueryFirst`- oder `QueryNext`-Anfrage weiter.

- *Attribute*-Satz
  - `Read`: Liest Attributwerte von Knoten.
  - `HistoryRead`: Liest Historiendaten oder Ereignisse von Knoten aus.
  - `Write`: Setzt Attributwerte von Knoten.
  - `HistoryUpdate`: Führt ein Update auf Historiendaten oder Ereignisse von Knoten aus.

- *Method*-Satz
  - `Call`: Ruft eine Methode eines Knotens auf.

- *MonitoredItem*-Satz
  - `CreateMonitoredItems`: Erzeugt neue *Items* für eine *Subscription*.
  - `ModifyMonitoredItems`: Verändert *Items* einer *Subscription*.
  - `SetMonitoringMode`: Setzt den Modus von *Items* einer *Subscription*.
  - `SetTriggering`: Setzt den Trigger eines *Items*.
  - `DeleteMonitoredItems`: Löscht *Items* einer *Subscription*.

- *Subscription*-Satz
  - `CreateSubscription`: Erzeugt eine *Subscription*.
  - `ModifySubscription`: Verändert eine *Subscription*.
  - `SetPublishingMode`: Aktiviert das Senden von Benachrichtigungen einer *Subscription*.
  - `Publish`: Bestätigt den Erhalt einer Benachrichtigung oder erfragt eine Benachrichtigung (unspezifisches Polling - der Server sendet nicht proaktiv).
  - `TransferSubscriptions`: Transferiert *Subscriptions* in eine andere Session.
  - `DeleteSubscriptions`: Löscht eine oder mehrere *Subscriptions*.

Neben den genannten Diensten existieren noch die *Discovery*-, *SecureChannel*- und *Session*-Sätze. Diese dienen der Verwaltung von Servern und der Kommunikation selbst. Sie arbeiten nicht direkt im Adressraum eines Servers. Außerdem können mit dem `call`-Dienst beliebige weitere Funktionalitäten aufgerufen werden. Dies macht auch sie zu Diensten. Diese letzteren Dienste sind nutzerspezifisch zu definieren, wobei es keine Vorgaben gibt, wie sie und ihre Schnittstellen zu beschreiben sind. Die Schnittstellen können zur Laufzeit im System erkundet werden.

Die große Menge an definierten Diensten unterstreicht die Komplexität von OPC-UA. Dies wird jedoch dadurch relativiert, dass nicht jeder Server alle Dienstsätze unterstützen muss. Es existieren so genannte Profile, die Subsets der Dienste festlegen, die ein Server unterstützen muss, um konform zu sein.

OPC-UA wird im Bereich Industrie 4.0 als Kerntechnologie des Datenaustauschs zwischen beliebigen Komponenten gesehen. Dies wird unterstützt durch die große Anzahl an Companion-Standards, die Methoden und Datenstrukturen von OPC-UA für spezifische Anwendungen definieren. Aufgrund des jungen Alters der meisten dieser Erweiterungen ist nicht davon auszugehen, dass sie sich bereits flächendeckend etabliert haben. Für das grundlegende System OPC-UA sieht dies anders aus. Es ist bereits in vielen Geräten zu finden, wobei auf anwendungsspezifische Modellierung der Daten gesetzt wird.

2.4 Produktdatenaustausch

Ein Großteil der vorhandenen Entitäten wird von Produkten ausgemacht. Daher ist der Austausch von Produktdaten ein zentraler Punkt eines Entity-Lifecycle Management Systems. Im Folgenden werden verschiedene Methoden zum Produktdatenaustausch beschrieben.

Eine wichtige Normenreihe für den Produktdatenaustausch ist die ISO 10303 oder Standard for the Exchange of Product Model Data (Standard für den Austausch von Produktdaten) (STEP) [58]. Diese dient der Spezifikation von Produktdatenmodellen, die den Datenaustausch erleichtern sollen. Besagte Modelle werden in der in Teil 11 der ISO 10303 spezifizierten Modellierungssprache EXPRESS ausgedrückt. Es werden eine Reihe von Basismodellen definiert, die als Grundlage für anwendungsspezifische Modelle dienen. Daraus werden mit Anwendungsprotokollen spezifische Sichten auf die Produktdaten generiert. Die Normenreihe besteht aus über 600 Teilen[11] (Vornormen mitgezählt). Diese beinhalten größtenteils anwendungsspezifische Modelle und Implementierungsmethoden. Weitere Teile der Reihe beschreiben die Beschreibungsmethoden und Konformitätstests.

Burkett hat sich zum Ziel gesetzt, autorisierten Nutzern den Zugriff auf Produktdaten von überall und ohne Zutun anderer Personen zu ermöglichen. Dabei setzt er auf STEP als Datenmodell und XML zur Strukturierung der Syntax. Die Daten werden mit Web-Technologien zugreifbar gemacht, wobei PDM-Systeme als Werkzeuge genutzt werden. Um die Übertragung im XML-Format mit Semantik zu unterlegen, wurden aus den EXPRESS-Modellen der Produkte XML-DTDs erzeugt. Des Weiteren werden explizit Nutzungskontexte angegeben, denen jeder Begriff zugeordnet wird. Dadurch wird ein semantisches Mapping zwischen verschiedenen Systemen möglich. Dies ist praktikabel, da die angesprochene Nutzergruppe (Personal des US-Verteidigungsministeriums) nur wenige verschiedene Nutzungskontexte verwendet. [10]

Yoo und Kim möchten das Wissensmanagement von Produktdaten in virtuellen Unternehmen erleichtern. Sie unterscheiden die Bereiche Metadaten zur Beschreibung der Datenquellen, Ontologien für semantische Informationen zu Daten und Mapping zur Transformation zwischen verschiedenen Formaten. Für diese Arbeit ist dabei der Teil der Metadaten am interessantesten. Metadaten werden für STEP-Dokumente klassifiziert. Die Kategorien sind

- Design: Informationen zu den STEP-Dateien: Dateiname, Größe, Adresse (URL), Beschreibung, genutzter Präprozessor und Schema;
- Registry: betrifft den Einsteller der Daten: Name, Datum, Mail, ID;

[11] Stand Mai 2016

- Part: Informationen zu Unterbaugruppen: Name, ID, Beschreibung, Anzahl, Hierarchie-Ebene, zugehörige Dokumente und Personen, umgebendes und eingeschlossene Objekte;
- Person: zugehöriges Personal: Name, ID, Rolle, Arbeitgeber;
- Approval: Informationen zu Genehmigungen: Status, Person, die approbiert hat, Typ der Approbation, Datum.

Mit diesen Feldern werden die Dokumente im System verwaltet. Um Kontext zwischen den Metadaten, insbesondere bei Teilebeziehungen zu erzeugen, werden Ontologien verwendet. Damit und mit zusätzlichen Synonymlisten kann in den Metadaten gesucht werden. Das Mapping zwischen verschiedenen Formaten wird durch DTDs umgesetzt.

Eine bereits in Kapitel 2.1 angesprochene Methode Produktdaten auszutauschen ist die Übertragung von Merkmalsinformationen. Solche Merkmale müssen semantisch definiert sein und eindeutige Identifikatoren haben. Die Normenreihe DIN 61987 bspw. legt Merkmale für Automatisierungskomponenten fest. Teil 11 geht dabei speziell auf Sensorik, Teil 12 auf Durchflussmesstechnik ein. Das CDD ist ein online verfügbares Repository der definierten Merkmale. Die Umsetzung des Austauschs der Merkmaldaten ist dabei vom Anwender zu spezifizieren und zu implementieren. Dies betrifft sowohl die kommunikationstechnische Basis, als auch die Zuordnung von Werten zu den definierten Merkmalen.[21, 23, 24, 52]

Leukel et al. hatten das Ziel, Klassifikationssysteme für Produktdaten zu modellieren, um ein generisches Modell dieser Systeme zu erzeugen. Damit sollen die Inhalte von Klassifikationshierarchien vergleichbar und übertragbar gemacht werden. Dazu wurde ein XML-Schema erzeugt, was die verschiedenen Elemente aller Systeme tragen kann. Es werden jedoch keine Merkmale o. ä. definiert. Lediglich die Darstellung einer solchen Definition wird festgelegt, damit sie aus den entsprechenden Klassifikationssystemen heraus übernommen werden kann. Keines der von Leukel et al. untersuchten Klassifikationssysteme füllt alle Felder des Modells aus. [73]

Braaksma et al. analysierten die Nutzung von Produktdatenaustauschstandards für die Zusammenarbeit in der Prozessindustrie. Im Gegensatz zur Flugzeug- oder Automobilindustrie hat sich von den vorhandenen Standards hier keiner durchgesetzt. In den ersten Industrien werden die Standards von den Marktführern oder von Überwachungsgremien gefordert und durchgesetzt. Dies ist in der Prozessindustrie nicht der Fall. Eine Begründung wird darin gesucht, dass in Flugzeug- und Automobilindustrie Serienproduktion herrscht, während prozesstechnische Anlagen Einzelanfertigungen sind. Dadurch kann ein Satz von Produktdatentypen nicht zwingend für mehrere Anlagen verwendet werden. [8]

2.5 Lebenszyklusmodelle

Die Betrachtung und Modellierung von vollständigen Lebenszyklen wird auf verschiedenen Ebenen verfolgt. Einerseits kann der Begriff des Lebenszyklus einheitsbezogen aufgefasst werden, wie es für die später vorgestellte Entitätenverwaltung getan wird. Dabei beschreibt er die Existenzphase der Einheit. Andererseits, kann er auf Produktlinien, Produkttypen oder Projekte übertragen werden. Zwar liegt der Fokus dieser Arbeit auf der einheitsbezogenen Betrachtungsweise, doch da der Begriff des Lebenszyklus vom Kontext abhängig unterschiedlich benutzt wird, werden im Folgenden auch andere Betrachtungsweisen erörtert,

um ein größeres Bild der aktuellen Entwicklungen zu zeigen und das spezielle Verständnis des Begriffs des Lebenszyklus in dieser Arbeit in Kontext zu setzen.

2.5.1 Referenzmodell

In [25] wird das Kernmodell des Lebenszyklus einer Einheit beschrieben. Grundsätzlich handelt es sich hierbei um „eine gerichtete und strikt azyklische Folge“ von Prozessen, die mit einem Entstehungsprozess beginnt und mit einem Vergehensprozess endet [25]. Während ihrer Lebenszeit befindet sich eine Einheit zu jedem Zeitpunkt in einem Lebenszyklusprozess. Das bedeutet die Abfolge dieser Prozesse ist dicht. An den Übergängen zwischen zwei Lebenszyklusprozessen befindet sich die Einheit in einem Übergangszustand. Dieser hat keine zeitliche Ausdehnung.
Der Lebenszyklus einer Einheit kann unter verschiedenen Aspekten modelliert werden. Beispiele aus der DIN SPEC 40912 [25] sind

- Planung des Lebenszyklus;
- Prognose des zukünftigen Lebenszyklusverlaufs und
- retrospektive Beschreibung des tatsächlichen Lebenszyklus.

Die dafür notwendigen Modelle können sich im Detail unterscheiden, gehen aber alle auf die zuvor beschriebene und in Abbildung 2.11 dargestellte Form zurück.

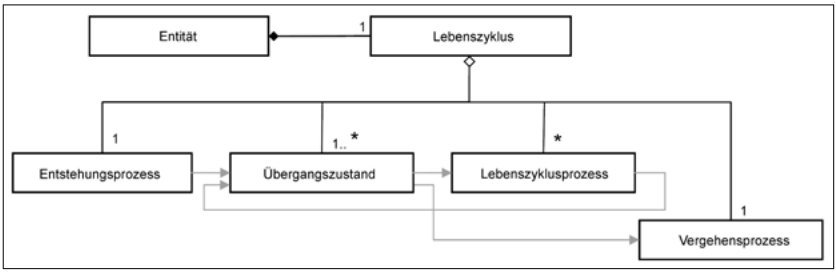


Abbildung 2.11: Referenzmodell eines Lebenszyklus [25]

2.5.2 Lebenszyklusmodell der IEC 62890

Die Norm IEC 62890 thematisiert das Lifecycle Management für Systeme und Produkte im Bereich der Automatisierungstechnik [56]. Die Norm befindet sich noch in Abstimmung. Während ihrer Entwicklung sind Vorveröffentlichungen entstanden, die in diesen Abschnitt mit einfließen.Im Folgenden soll das darin enthaltene Lebenszyklusmodell vorgestellt werden.
Die IEC 62890 unterscheidet die Lebenszyklen von Produkttypen und Produktinstanzen. Für beide werden verschiedene Phasen im jeweiligen Lebenszyklus beschrieben. Abbildung 2.12 zeigt die Phasen im Lebenszyklus eines Produkttyps und die Aktivitäten, die der Anbieter des Produkts zu den jeweiligen Zeitpunkten durchführt. Der Lebenszyklus

eines Produkttyps beginnt mit dessen Entwicklung durch den Anbieter. Am Ende der Entwicklung beginnt der Anbieter mit der Produktion. Kurz darauf beginnt die Vertriebsphase. Während dieser läuft die Produktion weiter. Außerdem wartet der Anbieter den Produkttyp, entwickelt ihn weiter und unterstützt den Anwender des Produktes durch Dienstleistungen. An die Vertriebsphase schließt sich die Unterstützungsphase an. In dieser Phase findet keine Produktion von Produktinstanzen mehr statt (ausgenommen für Wartungszwecke oder zur Deckung von Gewährleistungsansprüchen). Die Unterstützung der Anwender läuft jedoch weiter. Die Weiterentwicklung des Produkttyps wird eingeschränkt und bezieht sich nun auf nachfolgende Produkttypen.

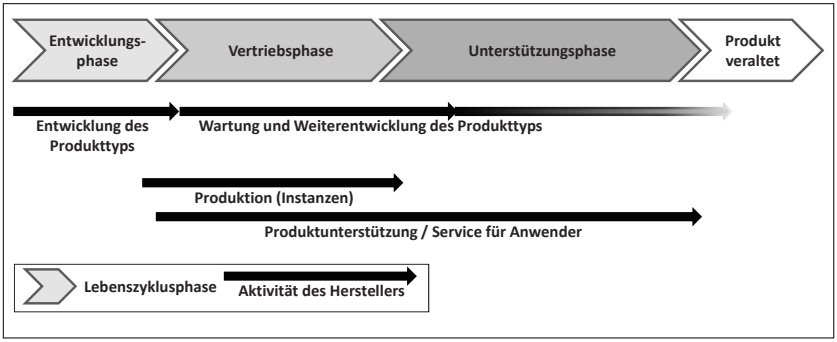


Abbildung 2.12: Lebenszyklusphasen für Produkttypen und Aktivitäten des Herstellers nach [56]

Der Lebenszyklus einer Produktinstanz weicht deutlich von dem eines Produkttyps ab. Nach IEC 62890 beginnt der Lebenszyklus einer Produktinstanz mit der Herstellung und endet bspw. mit ihrer Verschrottung. Während dieser Lebenszeit gibt es weitere Phasen von Interesse. Als solche werden beispielhaft genannt:

- die Nutzungszeit,
- die Ausfallzeit,
- die Garantiezeit,
- der Zeitraum der vom Hersteller vorgesehenen Unterstützungszeit (Standard Unterstützungszeit) und
- der Zeitraum der mit dem Hersteller individuell vereinbarten Unterstützungszeit.

Teilweise können diese Phasen parallel liegen (die Standardunterstützungszeit liegt meist parallel zur Garantiezeit und zur Nutzungs- und Ausfallzeit). Teilweise schließen sie sich gegenseitig aus, z. B. Nutzungszeit und Ausfallzeit. Lebenszyklusprozesse, die, wie im Kernmodell dargestellt, zu Zustandsänderungen der Produktinstanz führen, sind nicht modelliert [56].

Ein wichtiger Punkt ist die Verbindung der Lebenszyklen verschiedener Komponenten eines Gesamtsystems. Diese Herausforderung ergibt sich, wenn die in einer Anlage

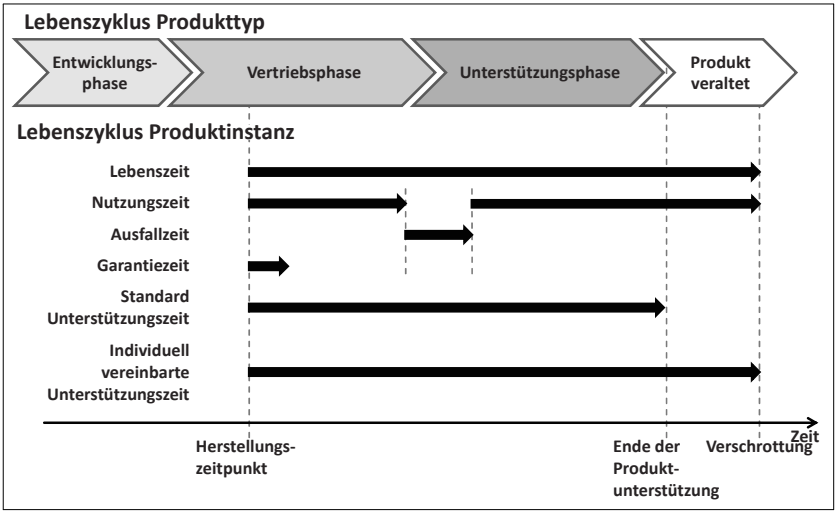


Abbildung 2.13: Phasen im Lebenszyklus einer Produktinstanz im zeitlichen Vergleich zum Lebenszyklus ihres Produkttyps nach [56, 94]

eingesetzten Komponenten eine kürzere Einsatzzeit haben, als die Gesamtanlage. In diesem Fall müssen Komponenten ersetzt oder instandgesetzt werden. Dies wird zum Problem, wenn nicht nur die Lebenszeit der eingesetzten Komponente, sondern auch ihre Unterstützungszeit endet. In diesem Fall ist kein genau gleiches Ersatzgerät verfügbar. Folglich muss ein kompatibler Ersatz gefunden werden. Dies soll durch eine Klassifikation von Kompatibilitätsstufen zwischen verschiedenen Geräten erleichtert werden. Außerdem werden verschiedene Strategien vorgestellt, wie proaktiv mit derartigen Ersatzproblemen umgegangen werden kann, wie bspw. die frühzeitige Beschaffung von Ersatzteilen, bevor eine Komponente abgekündigt wird. [85]

Wenn im Rahmen der IEC 62890 Lebenszyklusprozesse thematisiert werden, so sind die Aktivitäten der verschiedenen Akteure gemeint, die während des Lebenszyklus eines Produkttyps oder einer Produktinstanz durchgeführt werden. Diese wirken nicht unbedingt direkt auf die Instanz ein, sondern können auch organisatorischer oder planerischer Natur sein. Damit sind sie keine einheitsbezogenen Lebenszyklusprozesse im Sinne des Kernmodells.

2.5.3 Lebenszyklen im Referenzarchitekturmodell für Industrie 4.0 [88]

Im Referenzarchitekturmodell für Industrie 4.0 nimmt das Thema Lebenszyklus eine entscheidende Rolle ein. Es stellt zusammen mit den Wertschöpfungsketten eine Achse des drei-achsigen Referenzarchitekturmodells dar. Dies soll die ganzheitliche Betrachtung des Industrie 4.0-Ansatzes unterstreichen.

Das in diesem Zusammenhang verwendete Lebenszyklusmodell basiert auf dem im Ab-

schnitt 2.5.2 dargestellten Modell. Auch hier wird bei der Betrachtung von Gegenständen zwischen Typen und Instanzen unterschieden. Dabei beschreibt ein Typ einen potentiellen Gegenstand und eine Instanz einen spezifischen Gegenstand eines Typs. Dies ist nötig, da sich die Lebenszyklen von Typ und Instanz unterscheiden. Außerdem trägt es den unterschiedlichen Sichtweisen verschiedener Nutzer auf denselben Gegenstand Rechnung. In Abbildung 2.14 sind die Gegenstände *Fabrik*, *Maschine* und *Zulieferteil* mit ihren Lebenszyklen für Typ und Instanz dargestellt. In den Planungsphasen für *Fabrik*, *Maschine* und *Zulieferteil* wird mit den Typen gearbeitet und entsprechende Daten generiert. Sobald die einzelnen Gegenstände erzeugt werden, entstehen die zugehörigen Instanzen. Deren Lebenszyklus unterscheidet sich deutlich von dem der Typen. Trotzdem können im Lebenszyklus einer Instanz generierte Informationen wiederum zum Gegenstandstyp zurückfließen und damit dessen Lebenszyklus ändern.

Diese Erweiterung des Modells der IEC 62890 bezieht also die Instanzen selbst mit ein. Wie im Statusbericht zu I40-Komponenten dargestellt, ist die Verwaltung ihres Lebenszyklusses ein zentraler Punkt bei der Umsetzung der I40-Konzepte in die Praxis. Gleichzeitig wird der Bezug zur Entwicklung von Produkttypen und Produktlinien hergestellt [86, 87]. Der übergreifende Ansatz zeigt, welche Zusammenhänge bei der Betrachtung von Einheiten in ihrem Lebenszyklus eine Rolle spielen. Wie die Einheiten zu verwalten sind, wird nicht thematisiert.

2.5.4 Lebenszyklusmodell nach ISO/IEC/IEEE 15288 bzw. ISO/IEC 12207 [67]

Die Normen ISO/IEC/IEEE 15288 [67] und ISO/IEC 12207 [65] definieren Lebenszyklusprozesse im Engineering von Systemen bzw. Software. Dabei bezeichnet der Begriff Lebenszyklusprozess innerhalb dieser Normen einen Prozess, der von einer Organisation im Rahmen eines Engineeringprojekts durchgeführt wird.

Der Lebenszyklus eines Projekts erstreckt sich von der Feststellung der Notwendigkeit des Projekts, bis zum Verwerfen desselben. Die dabei darauf angewendeten Lebenszyklusprozesse werden in der ISO/IEC/IEEE 15288 in die Gruppen

- Einigungsprozesse (bspw. Akquiseprozesse)
- Organisatorische Hilfsprozesse (bspw. Infrastrukturverwaltung oder Qualitätsmanagement)
- Technische Verwaltungsprozesse (bspw. Projektplanung oder Risikomanagement) und
- Technische Prozesse (bspw. Architekturdefinition oder Implementierung)

eingeteilt. Dies soll Organisationen helfen, Modelle für die Durchführung ihrer Projekte zu erzeugen. Ein Bezug zu Entitäten, wie bspw. während eines Projekts erzeugte Produkte, ist nicht vorhanden. Die ISO/IEC 12207 führt die neuen Prozessgruppen Software Implementierung, Software Unterstützung und Software Wiederverwendung ein. Des Weiteren sind die in den Gruppen eingeordneten Prozesse an die Besonderheiten des Software Engineerings angepasst.

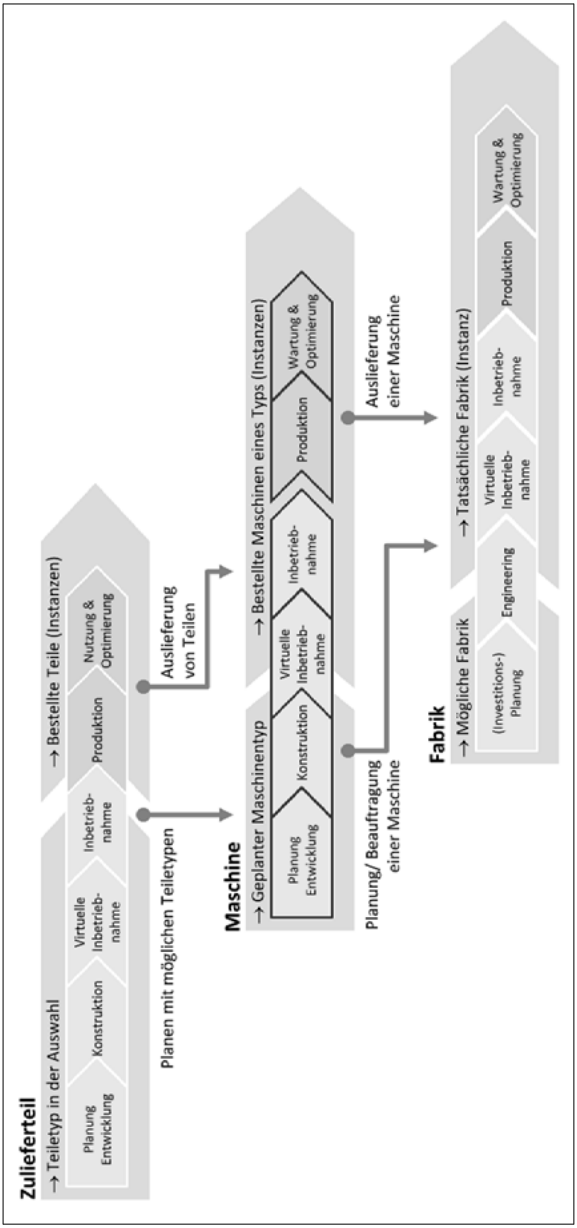


Abbildung 2.14: Die Unterscheidung von Typen und Instanzen trägt verschiedenen Sichtweisen der Beteiligten Rechnung. Erhält bspw. der Maschinenhersteller eine Rückmeldung aus dem Betrieb seiner Maschine, so fließen eventuelle Verbesserungen in die Planungsunterlagen der Maschine und damit in den Maschinentyp ein. [88]

Aufbauend auf den Prozessen aus ISO/IEC/IEEE 15288 und ISO/IEC 12207 spezifiziert ISO/IEC/IEEE 15289 [68] mit welchen Dokumenten eine Organisation diese Prozesse dokumentieren sollte und welche inhaltlichen Aspekte die einzelnen Dokumente abdecken müssen. Abgesehen von Anfragen werden sie ihrerseits mit den gemeinsamen Metadaten Datum, Scope, Urheber, Glossar und Änderungshistorie sowie weiteren dokumentspezifischen Metadaten verwaltet. Anfragen haben als gemeinsame Metadaten nur Datum und Scope.

In dieser Art der Lebenszyklusbetrachtung können sich die Phasen und Prozesse überlappen, wie in Abbildung 2.15 beispielhaft dargestellt. Dies ist möglich, weil sie innerhalb des Projekts auf ganz verschiedene Entitäten wirken und somit Prozessräume voneinander abgrenzbar sind. Es bedeutet aber auch, dass die einzelnen Prozesse ohne direkten Entitätsbezug verwaltet werden.

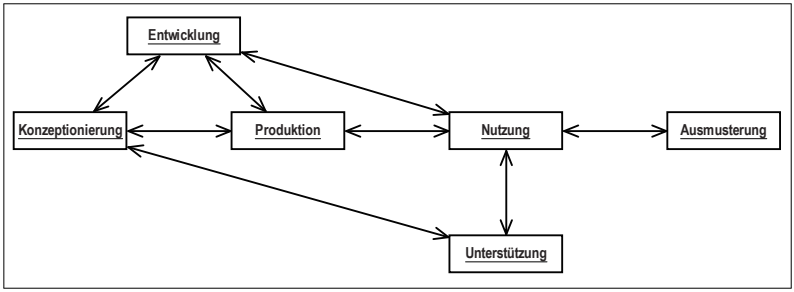


Abbildung 2.15: Phasen und Übergänge eines möglichen Lebenszyklus nach [66]. Auf allen Pfaden ist wiederholtes Durchwandern, wie auch Rekursion erlaubt.

3 Analyse bestehender Lifecycle-Management Ansätze

Lifecycle-Management wird auf verschiedenen Ebenen betrieben. Als erstes zu nennen ist das Product Lifecycle Management (PLM). Es bezieht sich auf die Verfolgung der Entwicklung von Produkttypen. Dabei können auch Daten von den Produktinstanzen wieder in das Management einfließen. Der Fokus liegt jedoch auf der Verwaltung der Typen. Die Instanzen liegen beim Asset Management (AM) im Fokus. Hier werden Geräte und andere Dinge individuell verwaltet, wobei die Einsatzphase meist im Fokus liegt. Schließlich werden auch Dateien und Dokumente in spezifischen Systemen verwaltet. Diese sind die Träger gedanklicher Einheiten. Daher sind auch solche Systeme hier von Interesse. Dieses Kapitel stellt im Folgenden verschiedene Ansätze der drei genannten Verwaltungsparadigmen dar.

3.1 Product Lifecycle Management

Das Product Lifecycle Management (PLM) ist aus dem Product Data Management (PDM) hervorgegangen. Letzteres „ist das Management des Produkt- und Prozessmodells mit der Zielsetzung, eindeutige und reproduzierbare Produktkonfigurationen zu erzeugen.“ [28] PDM war zunächst durch die Verwaltung von produktbezogenen Dokumenten (insbesondere CAD-Dokumente) geprägt. Wird dieser Ansatz, durch die Verteilung von Informationen und den netzwerkbasierten Zugriff erweitert, so kann er über den gesamten Lebenszyklus eines Produkts, wie auch über die Zulieferprozesse eingesetzt und integriert werden, wie Abbildung 3.1 darstellt. Dies definiert PLM [28]. Im Folgenden werden einige PLM-Ansätze aus der Forschung dargestellt.

Sudarsan et al. stellen in [97] ein Rahmenwerk vor, mit dem Produkte für das PLM modelliert werden können. Dabei sollen die anfallenden Daten jederzeit für alle Bereiche des produzierenden Unternehmens verfügbar sein und über den gesamten Produktlebenszyklus genutzt werden können. Um dies zu erreichen, definieren sie einen Modellkern aus dem CPM (siehe Abschnitt 2.2.2), dem open assembly model, einem design-analysis integration model und einem product family evolution model.
Das open assembly model beschreibt die Zusammensetzung von Systemen aus Subkomponenten. Dabei liegt der Fokus auf kinematischen Zusammenhängen und mechanischen Toleranzen.
Das design-analysis integration model soll die automatische Analyse eines Produkts hinsichtlich seiner Funktionalität ermöglichen. Dazu werden einem Produktmodel Zusammenhangsmodelle hinzugefügt, die die funktionale Analyse in verschiedenen Sichten (bspw. Belastung, Formänderung oder Kinematik) erlauben.
Das product family evolution model dient dazu, die Veränderungen über den Lebenszyklus einer Produktfamilie zu dokumentieren. Alle drei Modelle sind Erweiterungen des CPM. Sie beziehen sich auf Produkttypen. Die Einbindung und Verwaltung von Produktinstan-

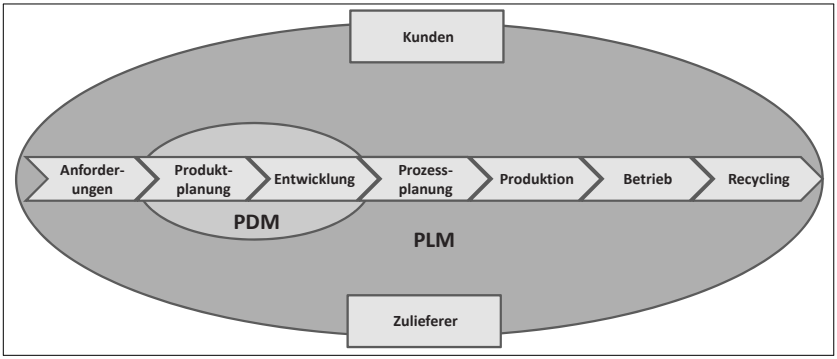


Abbildung 3.1: PLM erweitert das PDM auf den gesamten Produktlebenszyklus und über Zulieferketten hinweg. [28]

zen ist nicht vorgesehen. Auch der primäre Bezug zur Fertigungsindustrie ist erkennbar. Die Modellierung von Fluiden und Produktströmen ist nicht im Modellkern verankert. In ihren Ausführungen gehen Sudarsan et al. nur auf die Datenmodelle ein. Ein Application programming interface (Schnittstelle zur Anwendungsprogrammierung) (API) soll Teil weiterer Forschung sein.

Anderl et al. gehen bei der Entwicklung eines integrierten Bauteildatenmodells für Industrie 4.0 vom sogenannten Theta-Ansatz aus [3]. Dabei wird, wie Abbildung 3.2 zeigt, die Wertschöpfung eines Produkts mit zwei synchronisierten Sequenzen beschrieben, die jeweils der Informationswelt oder der physischen Welt zugeordnet sind. Dieses Modell verwaltet die Daten instanzbezogen. Aufgebaut ist das Bauteildatenmodell aus einem Kernmodell und daran angeschlossenen Partialmodellen. Das Kernmodell ist generisch und hält die Informationen für Identifikation, Adressierung, Lokalisierung, administrative und organisatorische Aufgaben sowie für die geometrische und materialbezogene Repräsentation. Die Partialmodelle fügen dann die real aus den Wertschöpfungsprozessen ermittelten Daten hinzu. Dabei soll die Datenerfassung bereits beim Rohstoff beginnen. Da angedacht ist, Produkte selbst zum Informationsträger zu machen, erhalten sie eine Art Gedächtnis. Dieses kann mit dem Produkt zum Anwender übergehen. Weil dieser aus dem Produktgedächtnis auch Informationen über den Produktionsprozess des Herstellers erhalten kann, müssen im Modell auch die Verantwortlichkeit und Zugriffsrechte auf die Daten abgebildet werden. Dies darf jedoch die Möglichkeit des Echtzeitzugriffs für autorisierte Nutzer nicht einschränken.

Oh et al. haben sich die erleichterte Integration von CAD-Daten in PDM-Systeme zum Ziel gesetzt [91]. Sie schlagen die Nutzung von STEP als Austauschformat vor. Um die Übersetzungen zwischen den verschiedenen Systemen zu modellieren, entwickelten sie das UML mapping diagram als Erweiterung von UML. Aus diesen können Übersetzer zwischen STEP und dem PDM-System erzeugt werden.

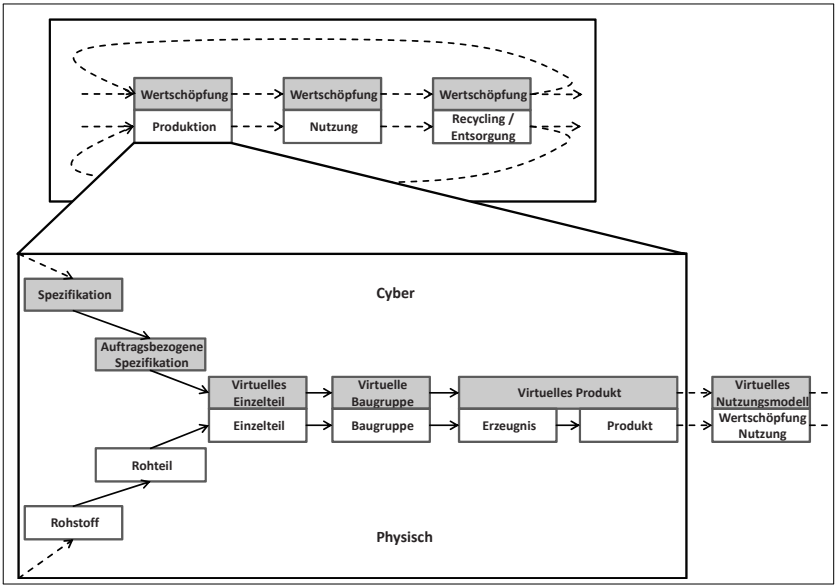


Abbildung 3.2: Das Theta-Wertschöpfungsmodell nach [3] zeigt die Kopplung von Dingen (Physisch) und deren Informationstechnischen Repräsentanten (Cyber).

3.1.1 PROMISE Projekt

Das von der EU geförderte Projekt PROMISE zielte darauf ab, die Informationsschleifen im Produktlebenszyklus zu schließen [72]. Es sollten also produktbezogene Daten aus der Nutzung wieder an den Hersteller zurückfließen können oder anderen Organisationen, die im Lebenszyklus einer Produktinstanz auftreten, bspw. Recycling-Unternehmen, verfügbar gemacht werden. Es wird dabei von intelligenten Produkten ausgegangen, die ein Product Embedded Information Device (PEID) tragen. Dieses PEID kann variieren zwischen einem Typschild mit einer GUID bis hin zu einem Embedded System, das kontinuierlich den Produktzustand überwacht.

Die Interaktion zwischen dem intelligenten Produkt und einem PLM-System erfolgt über einen PLM-Agenten. Dieser kann das PEID auslesen und die Daten über Internet mit dem PLM-System austauschen. Je nach PEID ist auch ein Zurückschreiben von Daten möglich. Dabei kann ein PLM-Agent eine Person, die ein mobiles Lesegerät nutzt, oder auch ein fest installiertes automatisches System sein. Die Kommunikation zwischen PLM-Agenten und PLM-System erfolgt dabei über das so genannte PROMISE Messaging Interface (PMI). Zur Identifikation der Produktinstanz wird der Electronic Product Code (EPC) [45, 76] genutzt. Dieser ermöglicht die global eindeutige Identifikation von Objekten. Um von besagtem Code an die Daten des konkreten Objekts zu kommen, soll der Object Naming Service (ONS) (erstmals in [40] vorgestellt) zum Einsatz kommen [71]. Der ONS ist ein Verzeichnisdienst, der auf Web-Technologien aufbaut und oberhalb des Domain Name Sy-

stem (DNS) angesiedelt ist. Er ordnet den EPCs die Internet-Adressen zuständiger Organisationen zu, von denen weitere Informationen abgerufen werden können. Durch eine Hierarchisierung des EPC ist dies zwar effizient möglich, trotzdem wird die Last auf einem solchen System sehr hoch sein, wie Kiritsis in [71] bemerkt.

Als gemeinsame Schnittstelle für den Datenaustausch mit PEIDs wurde der Core PEID Access Container (PAC) entwickelt. Dieser ist ein Universal Plug And Play (UPnP) Knoten, der definierte Dienste bereitstellt, um Daten aus den PEIDs auszulesen. Diese Dienste erlauben den lesenden und schreibenden Zugriff auf Attribute. Letztere sind als flache Liste von Name-Wert-Paaren ausgeführt [72, 82]. Die Implementierung des PAC ist anwendungsspezifisch.

Kern der Datenverarbeitung im PROMISE Projekt ist das Product Data and Knowledge Management (PDKM)-System. Es dient der Verwaltung aller anfallenden Informationen. Dazu wurde das PROMISE Semantic Object Model (SOM) entwickelt. Dieses ist in Abbildung 3.3 abgebildet [84]. Es stellt Klassen zur Speicherung und Verwaltung aller Daten bereit, die ab dem Beginn der Produktion einer Produktinstanz erhoben werden können. Dabei wird explizit von physischen Produkten ausgegangen, was sich auch in den gewählten Attributen ausdrückt. Es werden Zustände (CONDITION) der Produkte erfasst und mit Merkmalen und deren Werten verknüpft. Die Merkmalwerte können wiederum mit Dokumenten verknüpft werden. Aktivitäten und Ereignisse modellieren die Lebenszyklusphasen des Produkts, die ihrerseits explizit angegeben werden.

Das bereits erwähnte PMI ist die zentrale Schnittstelle zwischen den Systemen im Rahmen des PROMISE Projekts. Seine Spezifikation ist in [83] dargestellt und beinhaltet XML-Schemata für mögliche Nachrichten. Mit diesen Nachrichten können Ziele erkundet und ihre Datenobjekte ausgelesen und gesetzt werden. Dabei sind verschiedene Muster des Nachrichtenaustauschs (bspw. sofortige Antwort, verzögerte Antwort oder Subskriptionen) möglich. Zu beachten ist, dass die Spezifikation des PMI generisch ist und nicht mit dem SOM oder den Core PAC Diensten korreliert. Es sind ausschließlich generische Lese- und Schreib-Dienste vorhanden. Die Nachrichten sollen mit einem Webservice *notify* im SOAP-Format (definiert in [46]) übertragen werden. Andere Übertragungswege sind aber nicht ausgeschlossen [11, 83].

Die Konzepte des PROMISE Projekts wurden in verschiedenen Demonstrationsszenarien evaluiert. Darunter sind Wartung von Schwerlastfahrzeugen [4], prädiktive Wartung von Kühlschränken [12] und Verfolgung von Lieferungen und Gütern [41]. Ein Zusammenspiel der einzelnen Demonstratoren wurde nicht thematisiert. Auch die Implementierungen sind in weiten Teilen anwendungs- bzw. produktspezifisch.

3.2 Asset Management (AM)

Ein Asset ist ein gedankliches oder physisches Objekt, das möglicherweise oder tatsächlich einen Wert für eine Organisation hat. Dabei kann dieser Wert greifbarer oder finanzieller Natur sein, muss es aber nicht. Unter diesem Aspekt können auch Gruppen von Objekten unter einem Asset zusammengefasst werden. Des Weiteren kann ein Asset während seiner Lebensphase zu verschiedenen Organisationen gehören [64]. Auf dieser Definition aufbauend definiert die ISO 55000 Asset Management (AM) als die Menge koordinierter Aktivitäten einer Organisation, mit dem Ziel aus ihren Assets Wert zu erzeugen. Dies ist besonders unter dem finanziellen Aspekt zu sehen und beinhaltet meist die Betrachtung und

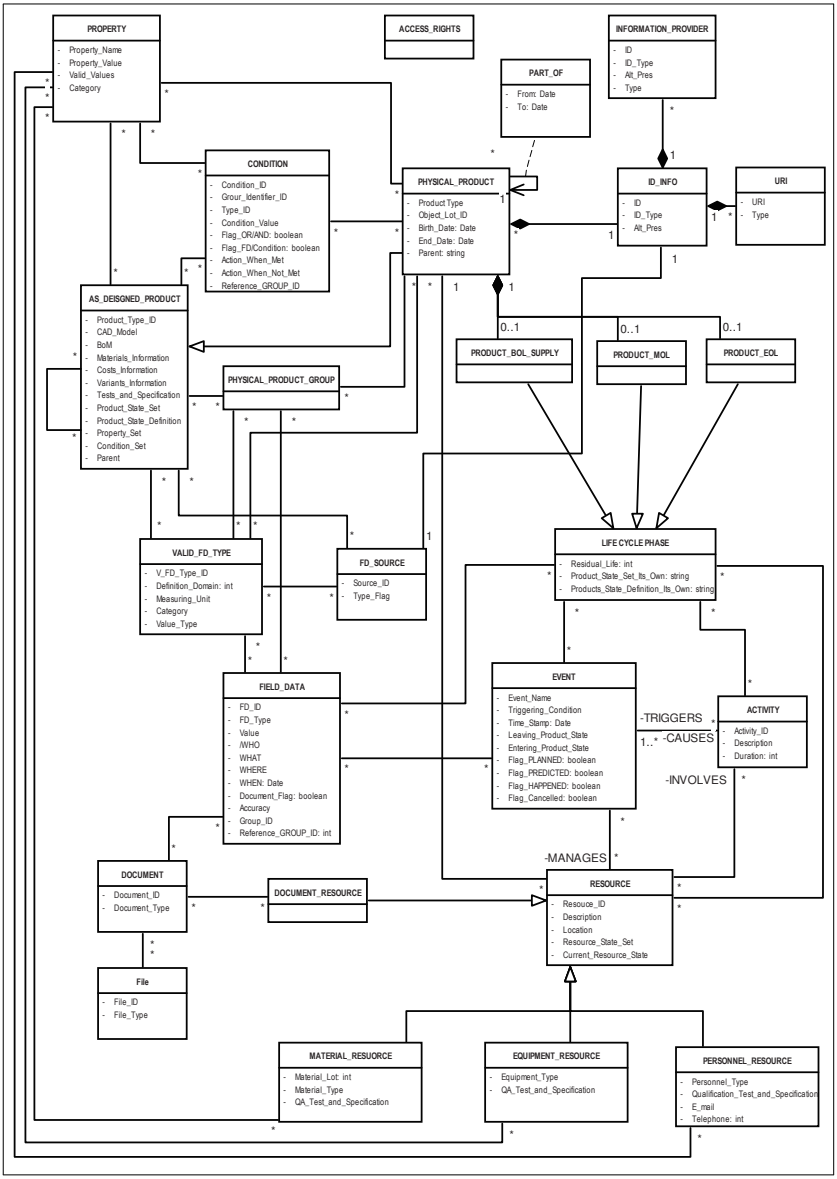


Abbildung 3.3: Das SOM des PROMISE Projekts nach [84].

Abwägung von Kosten, Risiken, Möglichkeiten und Leistung [64]. Während die ISO 55000 explizit auch Menschen in die Betrachtung einschließt und weiche Faktoren wie Kultur, Verhalten und Motivation als Aspekte des AM sieht, schränken Mitchell und Carlson ihre Sicht auf physische Objekte in einer Organisation ein. Gleichwohl verstehen sie unter dem Management der Assets alle Prozesse, die die Wertgeneration aus den Assets optimieren [79]. Auch El-Akruti sieht im Asset Management die Kombination von technischen und verwaltungsbezogenen Aktivitäten, die die Assets mittelbar oder unmittelbar betreffen. Er verortet die Herausforderung des AM in der Koordination, Integration und Optimierung des gesamten Asset Lebenszyklusses für Langzeitprofitabilität. Daher sieht er ein Problem darin, dass häufig nur die Nutzungsphase eines Assets betrachtet wird, die Kosten aber schon in der Planungsphase maßgeblich bestimmt werden [31]. Müller grenzt in seiner Dissertation die Betrachtung des Produktionsprozesses von den finanztechnischen Aktivitäten durch den Begriff des *anlagennahen Asset Management* bzw. *Plant Asset Management* ab. Dieser umfasst

1. Die Erzeugung und Bereitstellung aller Informationen über Assets (statisch / dynamisch, direkt / indirekt (durch Simulation) erzeugt, aus dem Prozess hergeleitet), besonders zur Unterstützung von Maßnahmen zur Sicherstellung von Funktionserfüllung und Werterhaltung;

2. Die Verwaltung der Assets über den gesamten Lebenszyklus (Ablage und Präsentation der Daten bspw. in den Sichten Historie, betriebswirtschaftliche Aspekte und technische Aspekte) und

3. Die Organisation des Einsatzes und Zustand-Erhaltens der Assets. (Dies ist eng mit Punkt 1 verknüpft, da es auf den dort gesammelten Informationen basiert)

Er merkt weiterhin an, dass anlagennahes AM meist nicht umfassend umgesetzt, sondern auf bestimmte Geräte fokussiert wird [80].

Schumann und Brent sehen die Notwendigkeit eines AM Ansatzes, der von Anfang an den gesamten Lebenszyklus der Assets und der umgebenden Systeme betrachtet. Sie nennen dies Asset Lifecycle Management. Der Grundgedanke ist hier, dass die Leistungsfähigkeit aber auch die zukünftig nötigen Wartungs- und Unterstützungsaktivitäten bereits in der Planungsphase eines Systems bei der Auswahl der Assets maßgeblich bestimmt werden. Das klassische Asset Management fokussiert jedoch auf bereits vorhandene Assets. Nach Ansicht der Autoren ist es kritisch die nicht-funktionalen Eigenschaften wie Zuverlässigkeit und Verfügbarkeit in der Planungsphase gleichwertig mit den funktionalen Eigenschaften anzusehen und in die Auswahl der eingesetzten Assets einzubeziehen. [96]

El-Akruti merkt an, dass AM meist als unterstützende Unternehmensfunktion angesehen wird, obwohl die Wertschöpfung in einem Unternehmen maßgeblich von der Leistung der eingesetzten Assets abhängt. Er führt an, dass AM in der Unternehmensstrategie verankert sein sollte und die vorher einzeln ablaufenden assetbezogenen Aktivitäten in einer Gesamtstrategie kanalisieren soll. Dazu entwickelte er das in Abbildung 3.4 dargestellte Rahmenwerk. Dieses zeigt, dass AM bezogene Aktivitäten auf allen Ebenen eines Unternehmens zu finden sind und stellt den Informationsfluss zwischen den Ebenen dar. Zustandsinformationen fließen dabei auf der linken Seite von unten nach oben, während Anweisungen rechts von oben nach unten weitergegeben werden. Letztlich gibt also die Unternehmensstrategie vor, wie mit Assets zu verfahren ist. [30, 31]

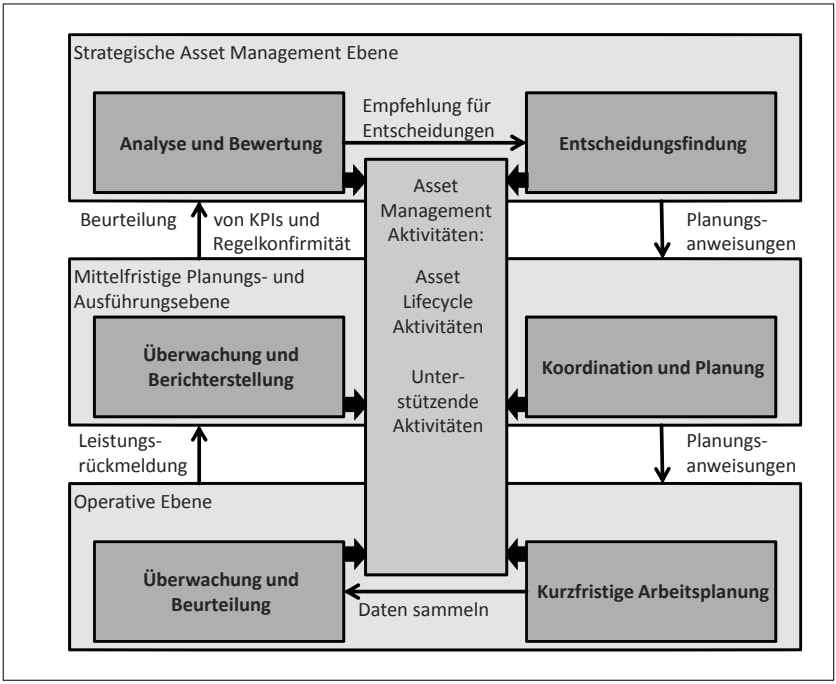


Abbildung 3.4: AM Rahmenwerk nach El-Akruti [31]. Informationen gehen die Ebenen hinauf, während Entscheidungen die Ebenen hinuntergehen. Auf allen Ebenen werden Asset Lifecycle Aktivitäten und unterstützende Aktivitäten genutzt.

Haffejee untersuchte AM Strategien in der Wasserversorgung. Er stellt heraus, dass Assets über ihren gesamten Lebenszyklus - von der Einsatzplanung bis zur Entsorgung - verwaltet werden müssen. Dabei definiert er strategische Assets. Diese haben einen großen Einfluss auf die Wertgeneration in einem Unternehmen. Daher werden primär diese Assets verwaltet. Er entwickelte ein Asset Lifecycle Management Modell, das die wesentlichen Aktivitäten in Beziehung setzt. Dieses Modell ist in Abbildung 3.5 dargestellt. Mit Treibern sind in diesem Zusammenhang die Treiber für Veränderung gemeint. Diese können unternehmensintern oder extern sein. Letzteres schließt dabei auch Faktoren wie Kultur ein. Die Datenhaltung soll für alle Assets in einem zentralen System erfolgen. [48]

Brown und Humphrey stellen die breite Aufstellung des AM heraus. Es basiert auf den drei Säulen der (Unternehmens-) Verwaltung, des Engineerings und der Informationssammlung und -verwaltung. Die Verknüpfung dieser Säulen in einem Gesamtsystem ist kompliziert, aber notwendig, wenn mehr als taktische Ziele erreicht werden sollen. Hier ist demnach eine unternehmensweite Strategie notwendig, wobei auch nicht-finanzielle Aspekte berücksichtigt werden müssen. [9]

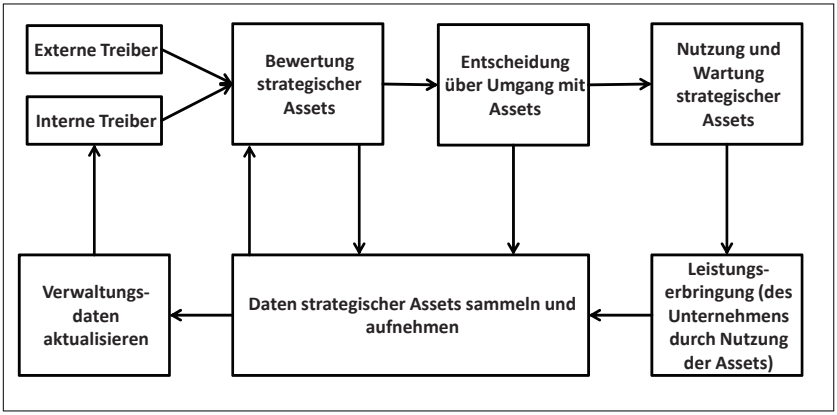


Abbildung 3.5: Asset Lifecycle Management Modell nach Haffejee [48].

Weitere Überlegungen im AM Umfeld zielen zumeist auf die operative Wartungs- und Instandhaltungsplanung für Geräte ab. Beispielhaft sei hier [90] erwähnt. Der Ansatz schlägt die automatische Modellierung von Produktionssystemen vor, um Anomalien automatisch und effizient erkennen und damit vorausschauend reagieren zu können.

3.3 Version Management

Produkttypen unterliegen einem Wandel im Laufe ihres Lebenszyklus. Insbesondere wenn es um solche Typen mit langer Herstellungs- und Unterstützungsphase geht, müssen Veränderungen nachverfolgt und hinsichtlich ihrer Kompatibilität evaluiert werden [16]. Um dies zu unterstützen, wurden eine Reihe von Methoden zur Versionsverwaltung entwickelt. Diese werden hier kurz vorgestellt.

Klassische Versionsverwaltungssysteme dienen besonders dazu, die Veränderungen in Programmquelltexten sichtbar und nachverfolgbar zu machen und dadurch die Entwicklungsarbeit in verteilten Teams zu erleichtern. Subversion (SVN) ist ein solches System. Es hält die Quelldateien eines Projekts mit ihrer Änderungshistorie nach und speichert für jede gemeldete Änderung eine Revisionsnummer, einen Zeitstempel, den Autor und einen Kommentar. Dabei arbeitet es dateibasiert. Ein Verschieben kommt also einem Löschen und neu Erzeugen gleich. Außerdem werden Änderungen nicht explizit gespeichert, sondern durch Vergleich zweier Revisionen ermittelt [15].

Ein weiteres Versionsverwaltungssystem ist git. Es fokussiert mehr auf verteiltes Arbeiten und arbeitet daher mit anderen Metadaten. Für jedes Objekt (dies kann eine Datei oder ein Dateibaum sein) wird ein Hash gespeichert. Dies ist eine aus den Dateiinhalten gebildete Prüfsumme, die in der Lage ist, den Dateiinhalt eindeutig zu identifizieren. Die eingesetzte Hash-Funktion ist kryptographischer Natur. Das bedeutet, es ist nicht mit vertretbarem Aufwand möglich, gezielt eine zweite Datei zu erzeugen, deren Inhalt denselben Hash erzeugt. Außerdem ist die Wahrscheinlichkeit, dass zwei Objekte im selben Projekt denselben Hash haben, vernachlässigbar klein. Dadurch kann git inhaltsorientiert arbeiten.

Eine Umbenennung einer Datei kann somit als solche erkannt werden. Außerdem kann die inhaltliche Gleichheit zweier Objekte eines Projekts beim verteilten Arbeiten durch Vergleich der Hashes erkannt werden. Neben dem Hash des Objekts selbst wird auch der Hash der vorigen Version des Objekts gespeichert. Dadurch ist die Nachverfolgung der Änderungen möglich. Außerdem werden Zugriffsrechte des Objekts, der Objekttyp (Datei, Baum, etc.), der Urheber der Änderungen, der Einbringer der Änderungen und ein Kommentar gespeichert. Die Unterscheidung zwischen Urheber und Einbringer ist dem verteilten Arbeiten geschuldet. Einbringer ist dabei derjenige, der die Änderungen in ein Projekt bringt, Urheber der ursprüngliche Autor der Änderungen [81].

Beiden genannten Systemen ist gemein, dass alle semantischen Informationen zu gemachten Veränderungen in natürlich sprachlichen Kommentaren vorliegen. Das System artshop des Lehrstuhls für Informatik 11[1] der RWTH-Aachen University umgeht dies durch die Nutzung klassifizierter Annotationen anstelle natürlichsprachlicher Kommentare. Dabei werden den Objekten Listen von Annotationstypen und zugehörigen Werten zugeordnet. Dies erleichtert die automatische Verarbeitung der Annotationen. Zu jeder Annotation werden Autor und ein Zeitstempel mitgeführt. Die Annotationstypen sind anwendungsspezifisch und werden in einem Repository verwaltet [77]. Das artshop-System dient nicht der Verwaltung von Quelltexten, sondern von Modellen und bringt diese mit Anforderungen zusammen. Beides wird im System in einem definierten Zwischenformat in einer Datenbank gespeichert. Durch die Umwandlung in das Zwischenformat ist eine automatische Analyse möglich. Die erfassten Metainformationen zu den Modellen sind jedoch nicht allgemein, sondern spezifisch für die eingesetzten Modelltypen [43].

3.4 Diskussion

Die vorigen Kapitel haben gezeigt, dass es kein System gibt, dass den Lebenszyklus einer Entität über Unternehmensgrenzen hinweg durchgängig verfolgen kann. Statt dessen werden verschiedene Systeme in verschiedenen Phasen des Lebenszyklus und von verschiedenen Organisationen eingesetzt. Zwar gibt es Bestrebungen, Systeme zu verkoppeln, dies scheint jedoch nicht über Individuallösungen hinauszugehen.

PLM-Systeme beziehen sich fast ausschließlich auf Produkttypen. Dabei kommen die Modelle weitgehend aus der Welt der Stückgutfertigung, was durch die Häufung von Geometriebeschreibungen (CAD-Daten) als verwaltete Informationen unterstrichen wird. Hiermit allein kann eine umfassende Beschreibung eines Entitätslebenszyklus nicht gelingen, da der Produkttyp zwar die Merkmale der Instanz festlegt, sich deren Ausprägungen aber unterscheiden können. Auch kann ein zentrales System nicht berücksichtigen, dass Hersteller und Anwender völlig verschiedene Sichten der Planungsphase einer Entität haben. Der Hersteller legt den Typ aus, während der Anwender die Anforderungen festlegt und sich daraus erst die Auswahl eines Entitätstyps ergibt. Gleichwohl können PLM-Systeme wertvolle Daten zum Ausgangszustand und zur Entstehung einer Instanz liefern.

Dieses Problem gilt auch für intelligente Produkte. Wenn das Produkt selbst zum Datenspeicher wird, müsste es für eine umfassende Informationsverwaltung alle Informationen für alle möglichen Sichtweisen bereit halten. Spätestens beim Wechsel der Produktinstanz in einen anderen Verantwortungsbereich ergeben sich damit zwei Probleme:

[1] Software für eingebettete Systeme

1. Mit dem Produkt können auch vertrauliche Informationen in einen anderen Verantwortungsbereich übergehen;

2. Der Zugriff auf die Daten wird für die Organisation, aus deren Verantwortungsbereich das Produkt herausgegangen ist, schwieriger.

Punkt eins kann durch Modellierung von Zugriffsrechten oder durch Löschung vertraulicher Informationen bei der Abgabe einer Instanz ausgeglichen werden. Die Modellierung von Zugriffsrechten ist jedoch kompliziert und bringt keine wirkliche Sicherheit, da die Instanz sich komplett in einem fremden Verantwortungsbereich befindet. Die Löschung vertraulicher Informationen bedeutet, dass diese, sofern noch von Interesse, vorab in ein anderes Informationssystem der Organisation übertragen werden müssen. Diese Verschiebung der Informationen bei der Übergabe bringt zusätzlichen Aufwand. Ob dieser kleiner ist als die Speicherung in einem Informationssystem mit Verweisen zur Einheit, ist fraglich. Die Speicherung aller Informationen in einem unternehmensbezogenen Informationssystem mit direkten Querverweisen zu den betreffenden Instanzen würde auch den Zugriff der abgebenden Organisation erleichtern (Punkt zwei). Daher scheint ein Ansatz über die eindeutige, automatische Identifikation der Instanz, z. B. durch ein erweitertes Typenschild (wie im PROMISE Projekt als PEID unterster Ebene bezeichnet), mit Abfrage aus den entsprechenden Informationssystemen der eigenen oder fremden Organisationen über einheitliche Schnittstellen, der praktikablere Weg zu sein. Selbstverständlich muss dafür nachverfolgt werden können, welche Organisationen vorab bereits Daten zur Instanz gesammelt haben.

Asset Management Systeme sind in der Lage, mit Instanzdaten zu arbeiten. Dies kann auch die Produkte einer Organisation mit einbeziehen. Der Bezug zum Engineering eines Produkts ist jedoch nicht generell gegeben. Auch die Einsatzplanung eines Assets wird nur selten in einem solchen System verwaltet, obwohl ganzheitliches Asset Management theoretisch alle Asset-bezogenen Aktivitäten umfassen müsste. Meist wird Asset Management anlagennah betrieben. Das bedeutet, dass es sich vorwiegend auf die Datenerfassung und Auswertung zur Nutzungsphase bezieht und Instandhaltungsaspekte betrachtet. In diesem Sinne werden Daten auch nicht über Unternehmensgrenzen weiter gegeben.

Dies kann auch damit zusammenhängen, dass es keinen flächendeckenden Standard für den Produktdatenaustausch gibt. Alle Standards sind auf einen bestimmten Anwendungsbereich oder eine Branche beschränkt. Zudem sind sie aufgrund der Pflege durch Normungsgremien relativ unflexibel. Das führt dazu, dass Einzelanfertigungen, wie bspw. Großanlagen oder andere vom Nutzer konfigurierte Produkte, schwer abzubilden sind, sobald sie über Variationen eines Standardprodukts hinausgehen. Daher ist für den Austausch von Produktdaten eine flexible Modellierung erforderlich, die in der Lage ist, im konkreten Produktdatenmodell nicht nur die Ausprägungen von Produktmerkmalen zu übertragen, sondern auch semantische Informationen und Metainformationen zur Produktinstanz.

Als Verwaltungsmethode für gedankliche Einheiten wurden Versionsverwaltungssysteme betrachtet. Diese verwalten jedoch streng genommen nicht die gedankliche Einheit selbst, sondern die Träger dieser Einheiten in Form von Dateien. Die dabei verwendeten Metadaten lassen sich gleichwohl auf die Einheiten selbst übertragen. Dabei ist zu beachten, dass besagte Metadaten keinen Bezug zum Informationsinhalt der verwalteten Einheiten haben. Auch der Hash im Verwaltungssystem git erlaubt lediglich die Prüfung zweier Objekte auf Gleichheit und keine Rückschlüsse auf den Inhalt. Inhaltsbeschreibende Daten werden fast ausschließlich in Form natürlich sprachlicher Kommentare abgelegt. Damit ist eine Analyse dieser Informationen mit dem Ziel eines einheitlichen Verwaltungssystems kaum möglich.

Die Herangehensweise des artshop-Systems zeigt die Möglichkeit auf, solche Informationen über definierte Annotationen (Merkmale) anzufügen. Die dafür genutzten Annotationstypen sind modellspezifisch. Der Ansatz ist aus Sicht der maschinenauswertbaren Semantik vielversprechend. Eine gemeinsame, unternehmensübergreifende Semantikbasis ist jedoch aufgrund der spezifischen Definitionen nicht gegeben. Sie kann aber über öffentliche Definitionen der Annotationstypen hergestellt werden. Letztlich ist dies also eine Anwendung der merkmalbasierten Informationsverwaltung auf die Träger gedanklicher Einheiten.

Ein Projekt, in dem Informationen für verschiedene Organisationen im Lebenszyklus einer Einheit zugreifbar gemacht werden sollten, war PROMISE. Der Fokus lag hier darauf, die Informationen aus der Nutzungsphase zum einen wieder an den Hersteller zurückfließen zu lassen, um ihm die Weiterentwicklung zu vereinfachen und zum anderen Informationen mit dem Produkt mitzuführen, um dessen Nutzung oder Recycling zu vereinfachen. Dabei wurde von intelligenten Produkten ausgegangen, die ein Product Embedded Information Device (PEID) tragen. PEIDs wurden in verschiedene Ebenen untergliedert. Die niedrigste Ebene war dabei ein Typenschild, dass die eindeutige Identifikation zulässt, die höchste Ebene ein eingebettetes System, das selbstständig Daten aufnimmt, verarbeitet und zusammen mit weiteren Produktdaten abrufbar macht. Das PEID der höchsten Ebene zielte darauf ab, zusätzliche Daten bspw. für prädiktive Wartung zu sammeln und aktiv Wartungsprozesse auszulösen.

Die eindeutige Identifizierung einer Einheit sollte über den EPC erfolgen. Dieser wird über eine Registratur einer Organisation - genauer gesagt der URL der Organisation - zugeordnet, die die Daten der Einheit hält. Die Registratur ist der Object Naming Service (ONS). Da der EPC hierarchisch aufgebaut ist, ist die Zuordnung zwar effizient implementierbar, trotzdem ist die durch die schiere Menge an Abfragen auf dem ONS liegende Last immens. Ein weiteres Problem ist, dass der EPC teilweise zentral vergeben wird. Dies macht diese Benamungsweise unflexibel.

Ebenfalls im Rahmen des PROMISE Projekts wurde ein Semantic Object Model (SOM) entwickelt. Dieses sollte als gemeinsame Basis für den Informationsaustausch dienen. Bei der Betrachtung des SOM (vergleiche Abbildung 3.3) fällt sofort die hohe Komplexität auf. Des Weiteren werden drei Lebenszyklusphasen (Beginn, Nutzung und Ende) explizit modelliert und jeder Vorgang exakt einer Phase zugewiesen. Damit kann verschiedenen Sichtweisen der Beteiligten auf die Lebenszyklusphasen nicht Rechnung getragen werden, da von einem einheitlichen Verständnis der Phasen ausgegangen wird. Diese sind jedoch nicht objektiv fassbar. Außerdem werden weder gedankliche Einheiten modelliert, noch gibt es eine Möglichkeit, Zusammenhänge zwischen verschiedenen Merkmalen auszudrücken.

Als Schnittstelle für den Datenaustausch wurde das PROMISE Messaging Interface (PMI) entwickelt. Es nutzt XML-basierte Nachrichten zur Informationsübertragung. Verschiedene Nachrichtentypen wurden festgelegt, die über einen Webservice *notify* übertragen werden sollen. Andere Übertragungswege sind auch möglich. Die Nachrichten ermöglichen die Erkundung und das Auslesen von Daten. Sie haben jedoch keinen Bezug zum SOM. Damit ist der semantische Inhalt der Nachrichtenschnittstelle auf derselben Ebene wie die Operationen von OPC-UA oder ACPLT/KS. Neben dem PMI wurde eine API zur Anbindung der PEIDs entwickelt. Auch diese hat keinen Bezug zum SOM. Sie überträgt die Daten als flache Liste von Name-Wert Paaren.

Zusammenfassend ist zum PROMISE Projekt zu sagen, dass es eine große Anzahl an Problemen lösen kann. Es ist jedoch hinsichtlich des allgemeinen Datenaustauschs nicht getestet. Die Prototypen wurden in isolierten Projekten eingesetzt und waren jeweils auf

spezifische Problemstellungen fokussiert. Eine Interaktion der Prototypen wurde nicht beschrieben.

Für die einheitliche Verwaltung von Entitätsdaten über Unternehmensgrenzen hinweg können Teilaspekte der gezeigten Systeme und Projekte genutzt werden. Es gibt jedoch kein System, das allein alle Anforderungen, besonders hinsichtlich der notwendigen Flexibilität, erfüllt. Daher ist es wichtig, die Stärken der verschiedenen Systeme zusammen zu bringen und über gemeinsame Elemente auf semantischer, wie auch auf Kommunikationsebene, zu verknüpfen.

4 Reale Lebenszyklen und ihre Modellierung

Jedes existierende Objekt hat, allein weil es existiert, einen Lebenszyklus. Dieser ist durch die zeitkontinuierliche Veränderung der dem Objekt inhärenten Eigenschaften und Strukturen bestimmt. Beispielhaft sei eine Kreiselpumpe erwähnt, die zum Zeitpunkt t_0 beim Hersteller entsteht und zum Zeitpunkt t_{end} verschrottet wird. Der reale Lebenszyklus dieses Objekts beginnt also zum Zeitpunkt t_0, endet bei t_{end} und umfasst alles, was in der Zwischenzeit mit der Pumpe passiert. Eine exakte und vollständige Abbildung des Lebenszyklus der Pumpe würde also die kontinuierliche Erfassung aller ihrer Eigenschaften im Zeitraum von t_0 bis t_{end} erfordern. Es ist leicht zu erkennen, dass dies unmöglich ist, da

1. die Eigenschaften mit digitalen Informationssystemen bestenfalls quasi-kontinuierlich erfasst werden können;
2. die Menge der Eigenschaften der Pumpe für eine vollständige Abbildung zu groß ist;
3. nicht sicher gestellt werden kann, dass die Menge der erfassten Eigenschaften überhaupt vollständig ist und
4. die Lage der Zeitpunkte t_0 und t_{end} nicht objektiv definierbar ist.

Vielmehr wird es dazu kommen, dass zu bestimmten diskreten Zeitpunkten einige Eigenschaften der Pumpe bestimmt werden. Abbildung 4.1 beschreibt dies grafisch. Weiterhin wechselt die Pumpe zwischen besagten Zeitpunkten möglicherweise den Verantwortungsbereich, sodass die erfassten Informationen von verschiedenen Beteiligten gehalten werden. Auch ohne einen solchen Wechsel werden unterschiedliche Beteiligte Informationen zum Zustand der Pumpe sammeln und damit Wissen über ihren Lebenszyklus aufbauen. Die erfassten Daten sind dabei unterschiedliche und die Zeitpunkte bezogen auf den realen Lebenszyklus der Pumpe willkürlich. Daraus ergibt sich das in Abbildung 4.2 dargestellte Bild: verschiedene Beteiligte haben zu verschiedenen Zeitpunkten Wissen zum Zustand der Pumpe.

Dies lässt sich an Abbildung 2.13 spiegeln. Die Lebenszeit der Pumpe lässt sich in verschiedene Phasen einteilen (Nutzzeit, Ausfallzeit, Gewährleistungszeit etc.) (vergleiche [94, 95]). Während der verschiedenen Phasen sammeln verschiedene beteiligte Unternehmen oder Gewerke mehr Informationen als andere. In der Ausfallzeit wird die Wartungsabteilung mehr über die Pumpe wissen als der Betrieb. Der Hersteller wird die meisten Informationen vor der Betriebsphase sammeln, während der Anwender hier meist ohne Information ist. Die über den Lebenszyklus der Pumpe anfallenden Informationen zu ihrem Zustand sind folglich weit gestreut.

Da es sich bei der angesprochenen Art der Datenerfassung um den Ist-Zustand der realen Pumpe handelt, verbindet alle aufgenommenen Daten eine besondere Eigenschaft: Sie sind,

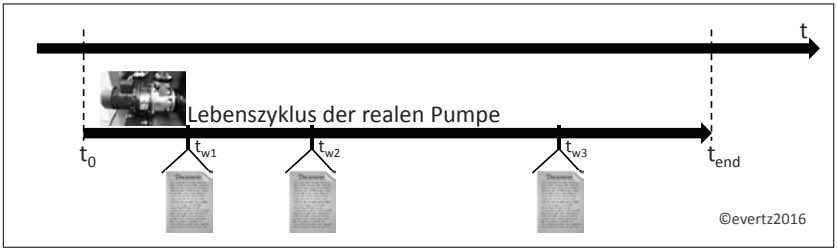


Abbildung 4.1: Der tatsächliche Lebenszyklus der Kreiselpumpe beginnt mit ihrer Entstehung zum Zeitpunkt t_0 und endet mit ihrem Vergehen zum Zeitpunkt t_{end}. Dazwischen kann der Zustand der Pumpe nur zu diskreten Zeitpunkten (hier beispielhaft t_{w1}, t_{w2} und t_{w3}) erfasst werden.

von einer fehlerfreien und objektiven Datenerfassung ausgehend, in jedem Fall konsistent. Selbst wenn verschiedene Beteiligte zum selben Zeitpunkt dieselbe Eigenschaft erfassen, kann sich das Ergebnis nur im Rahmen der Messunsicherheit unterscheiden, da es sich um einen tatsächlichen Zustand handelt.

Aus diesem Grund können alle derartigen erfassten Ist-Zustände auf der Zeitschiene geordnet und konsolidiert werden. Dabei muss nur in Sonderfällen, wie bei fehlerhaften Messungen und nicht objektiven Datenerfassungen (bspw. Klassifikation durch einen menschlichen Beobachter), Konsistenz wiederhergestellt werden. Es können also Beschreibungen beliebiger Abschnitte des realen Lebenszyklus der Pumpe zu einer Gesamtbeschreibung zusammengesetzt werden.

Diese Ausführungen zum Lebenszyklus gelten nicht nur für Pumpen und technische Einheiten. Abbildung 4.3 zeigt die Breite der möglichen Einheiten anhand einer beispielhaften Generalisierungshierarchie. Dabei ist auch diese nicht vollständig.

Einheiten können zunächst in physische und gedankliche Einheiten unterteilt werden. Dabei sind gedankliche Einheiten Ideen, Pläne etc., die in der Informationswelt angesiedelt sind. Sie bedürfen immer mindestens eines Trägers. Dieser kann auch eine Person sein. Alle anderen Einheiten sind als physische Einheiten zu sehen. Darunter fallen auch Dateien als Träger gedanklicher Einheiten.

Gedankliche Einheiten sind zumeist Beschreibungen, bspw. für Zustände, Vorgänge oder Gegenstände. Demgegenüber können physische Einheiten in gegenständliche und nichtgegenständliche Einheiten unterteilt werden. Letztere sind u. a. Dateien und Prozesse, da sie Ordnungsstrukturen und Abläufe in der physischen Welt sind, aber nicht selbst aus Materie bestehen. Gegenständliche Einheiten bestehen aus Materie. Sie sind entweder natürlich entstanden oder Mensch-gemacht. Die wichtigste natürliche Einheit ist die Person, jedoch gibt es selbstverständlich noch viele weitere Typen (Tiere, Pflanzen etc.). Die Mensch-gemachten Einheiten werden in dieser Klassifikation in Mengen (z. B. Fluide) und diskrete Einheiten (Stückgut) eingeteilt.

Wird der Lebenszyklus einer solchen Einheit verfolgt, also retrospektiv der Ist-Zustand der Einheit abgebildet, so wird diese Einheit zur Entität (vergleiche Kapitel 2.2). Ein Schlüsselelement im Rahmen der Initiative Industrie 4.0 ist die Verwaltungsschale. Diese setzt unter anderem genau die besagte Verfolgung des Lebenszyklus einer Einheit um. Da-

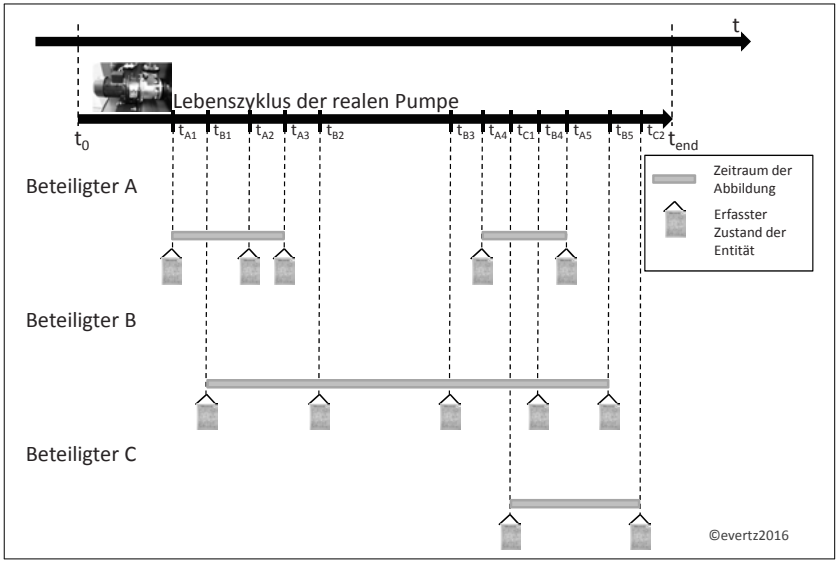


Abbildung 4.2: Verschiedene Beteiligte erfassen den Zustand einer Pumpe zu verschiedenen Zeitpunkten in ihrem Lebenszyklus. Daraus ergeben sich für die Beteiligten unterschiedliche Zeiträume, in denen sie Wissen über die Pumpe haben.

mit wird die Modellierung des Lebenszyklus selbst zu einer Kernforderung für Industrie 4.0, die für alle möglichen Arten von Dingen umgesetzt werden muss.

Objektiv wirkt zu jedem Zeitpunkt ihrer Existenz genau ein Prozess auf eine Einheit, der zumindest ihr Alter verändert (vergleiche [25]). Solche Prozesse sind wiederum nicht vollständig erfassbar, weshalb nur die für einen Betrachter interessanten Teilaspekte in seine Lebenszyklusmodellierung einer Einheit einfließen. Abbildung 4.4 zeigt beispielhaft verschiedene Prozessabfolgen im Lebenzyklus. Dabei ist mit jeder dargestellten Lebenszyklusabbildung eine bestimmte Betrachtungsweise einer Einheit und ein Betrachtungszeitraum verbunden. Die abgebildeten Lebenszyklen haben dabei keinen gemeinsamen Zeitbezug. Sie stellen die modellierten Abschnitte der Existenz der Einheiten dar.

Die Art der betrachteten Einheiten beeinflusst die möglichen Prozesse in ihrem Lebenszyklus. Gedankliche Einheiten bspw. haben keine Eigendynamik. Folglich sind sie nicht in der Lage, sich selbst zu verändern. Sie entstehen durch eine Idee oder gehen aus einer anderen gedanklichen Einheit durch Modifikation hervor und werden immer extern modifiziert. Sie vergehen beim Vergehen ihres letzten Trägers (bspw. der letzten Datei). Physische Einheiten hingegen können eine Eigendynamik haben. Dabei ist festzuhalten, dass physische Einheiten durchaus nicht-gegenständlich sein können, wie ein Prozess oder ein Programm. Diese sind zu unterscheiden von ihren Beschreibungen oder Modellen, die gedanklicher Natur sind und die Eigendynamik beschreiben, sie aber nicht selbst umsetzen[1].

[1] Auch „aktive Modelle“ beschreiben lediglich eine (Eigen-)Dynamik, die durch die Ausführung des Modells in einer Laufzeitumgebung (physisch) umgesetzt wird.

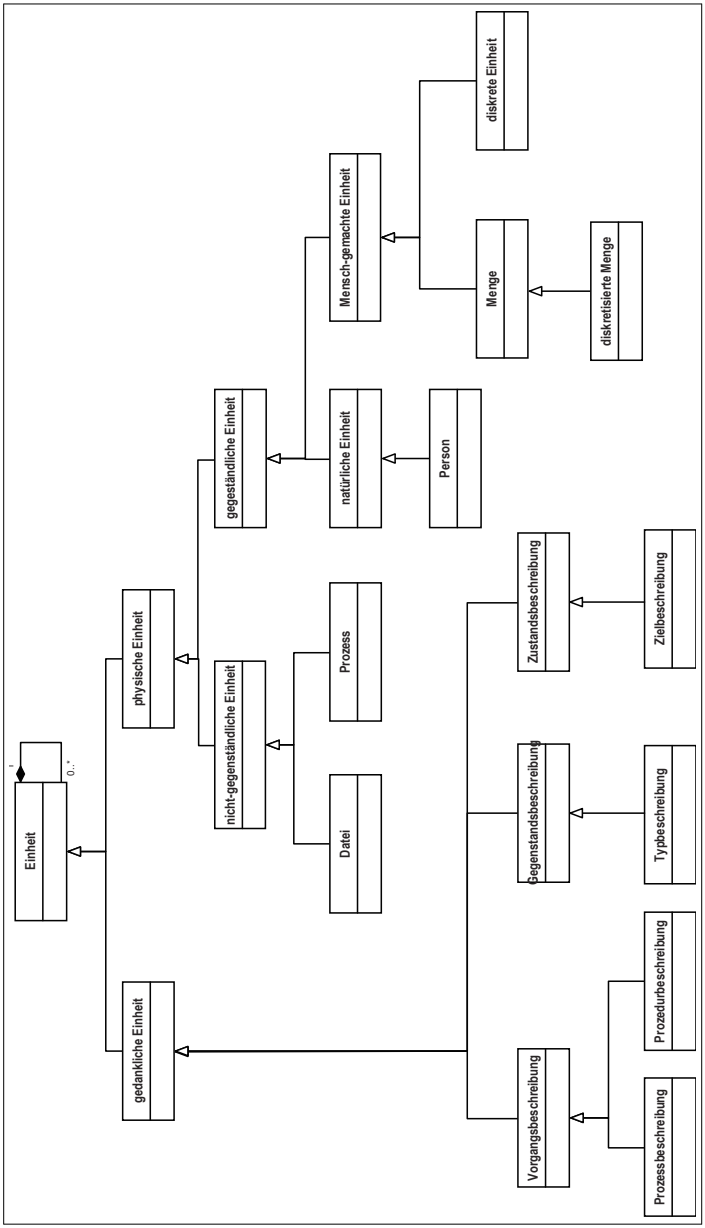


Abbildung 4.3: Beispielhafte Generalisierungshierarchie von Einheiten.

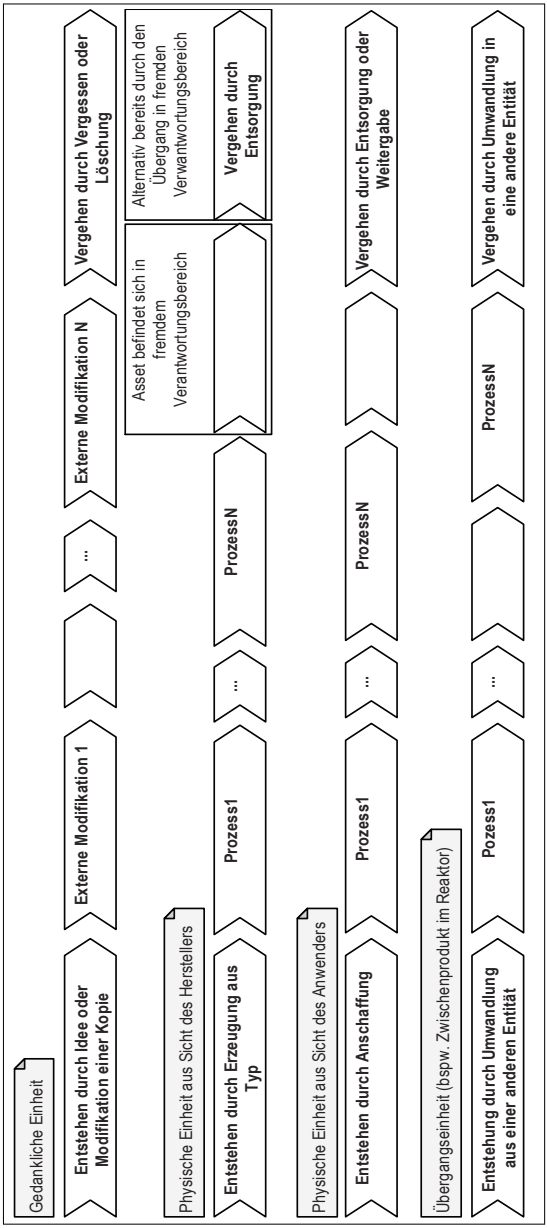


Abbildung 4.4: Aufbau verschiedener Sichtweisen auf den Lebenszyklus. Die Abfolge von Prozessen / Modifikationen ist nicht chronologisch. Ein leerer Prozess bedeutet, dass sich im Modell nur das Alter der Einheit ändert, aber keine weiteren Eigenschaften.

Je nach Perspektive ändert sich auch der Betrachtungszeitraum einer Einheit. Dies ist den beiden mittleren Beispiellebenszyklen in Abbildung 4.4 zu entnehmen. Die Entstehung einer (physischen) Einheit wird für den Hersteller mit der Erzeugung der Einheit aus ihrem Typ zusammen fallen. Die Festlegung des exakten Entstehungsprozesses ist dabei eine Designentscheidung, da erstens nicht objektiv festgelegt werden kann, wann die Einheit entsteht und zweitens nur der Hersteller selbst wissen kann, ab wann die Betrachtung der Einheit als solche für ihn Sinn macht. Für einen späteren Anwender der Einheit ist diese Form der Entstehung häufig nicht relevant, da sie sich seinem Verantwortungsbereich entzieht. Anders herum kann die Einheit in der Betrachtungsweise des Herstellers bei der Übergabe an den Anwender vergehen, da sie seinen Verantwortungsbereich verlässt.

Heutzutage ist dies kein wahrscheinliches Szenario, da der Hersteller meist weiterhin Serviceaufgaben übernimmt und Informationen der Einheit in seinen Entwicklungsprozess zurückfließen lassen möchte. Da er jedoch meist keinen direkten Zugriff auf die Einheit mehr hat, kann er ihren Lebenszyklus nur sporadisch verfolgen. Daher ist der ein seiner Betrachtung auf die Einheit wirkende Prozess leer, obwohl er weiß, dass sich Eigenschaften der Einheit ändern (vorletzter Prozess der zweiten Lebenszyklusabbildung in Abbildung 4.4).

Ein in Abbildung 4.4 dargestellter Sonderfall sind Übergangseinheiten. Damit sind Einheiten gemeint, die nur kurzzeitig existieren und ständig Veränderungen unterworfen sind. Möglicherweise sind sie nicht einmal konkret greifbar. Als Beispiel sei ein Zwischenprodukt innerhalb eines Prozesses genannt, das den Prozessraum nie verlässt. Derartige Einheiten können auch in der Nachverfolgung von Interesse sein, wenn bspw. die Eigenschaften anderer Einheiten direkt von der Lebensphase der Übergangseinheit abhängig sind.

Wird der Lebenszyklus einer Einheit mehrfach abgebildet, bspw. von verschiedenen Beteiligten mit unterschiedlichen Interessen, so können die betrachteten Zeiträume durchaus überlappen. Werden dabei nicht nur tatsächliche Zustände erfasst, wie oben beschrieben, sondern auch auf die Einheit wirkende Prozesse modelliert, so sind auch diese, da es sich um eine Abbildung der Realität handelt, im Rahmen der modellinhärenten Ungenauigkeiten konfliktfrei. Werden von den Beteiligten jeweils entkoppelte Facetten des objektiv auf die Einheit wirkenden Prozesses modelliert, so können diese einfach zusammengefasst werden. Überlappen die betrachteten Facetten, so ist die Zusammenfassung aufwendiger. Nichtsdestotrotz widersprechen sich die einzelnen Modelle nicht.

4.1 Zusammenführung von Lebenszyklusabbildungen einer Entität

Verschiedene Abbildungen des Lebenszyklus einer Einheit lassen sich wie bereits erwähnt konfliktfrei konsolidieren. Trotzdem müssen die zusammengeführten Informationen dabei geprüft werden, da die zugrunde liegenden Daten aufgrund von Fehlern oder Interpretationsspielräumen differieren können. Ebenso können sich die Modelle der auf die Einheit wirkenden Prozesse durch Vereinfachungen und Annahmen unterscheiden. Mit anderen Worten, die Beschreibungen der Realität differieren durch ihre Abstraktionen und müssen daher hinsichtlich der Grenzen ihrer Aussagekraft geprüft und zusammengeführt werden. Abbildung 4.5 zeigt die möglichen Widerspruchsfälle bei der Konsolidierung der Modelle.

Wenn bei der Konsolidierung der verschiedenen Modelle ein (scheinbarer) Widerspruch zu Tage tritt, muss dieser analysiert werden. Für erfasste Ist-Zustände gibt es hier zwei Möglichkeiten. Entweder ist eine Messung fehlerhaft, oder die angesetzten Skalen oder

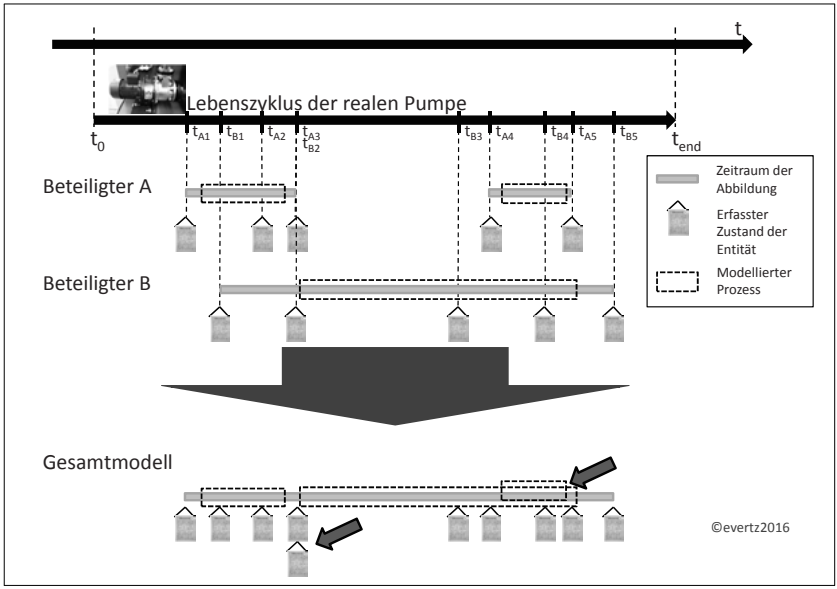


Abbildung 4.5: Die Lebenszyklusabbildungen verschiedener Beteiligter für eine Einheit lassen sich konfliktfrei konsolidieren. Bei überlappenden Beschreibungen auftretende Widersprüche bestehen zwischen den Abbildungen und nicht in der Realität. Sie können nur durch Messfehler und Modellungenauigkeiten auftreten. Diese müssen erkannt und bewertet werden.

Bewertungskriterien sind verschieden. Aus den reinen Daten lassen sich diese Fälle nicht unterscheiden. Es muss daher zu jedem Datum klar sein,

1. wie es ermittelt wurde,
2. welche Skala bzw. welches Bewertungskriterium dabei angewandt wurde,
3. wer für die Ermittlung verantwortlich ist und
4. wie groß die Messunsicherheiten sind.

Mit diesen Informationen können vermeintlich widersprüchliche Daten bewertet werden. Wenn zwei verschiedene Daten zum selben Zeitpunkt in allen Zusatzinformationen übereinstimmen und die Daten sich um mehr als die angegebene Messunsicherheit unterscheiden, so muss ein Messfehler bei mindestens einem Datum vorliegen. Um diesen genau zu lokalisieren, bedarf es weiterer Untersuchungen. Das Vorhandensein des Messfehlers kann jedoch sicher angezeigt werden.

Unterscheiden sich zwei Werte hinsichtlich der genannten Metainformationen, so liegt ein Widerspruch möglicherweise nur scheinbar vor. Sind bspw. die Bewertungskriterien nicht gleich, so mögen sich die Werte unterscheiden, die abgebildete Information kann

jedoch durchaus gleich sein. Die angegebenen Metainformationen ermöglichen damit die Erkennung und Bewertung der erfassten Daten.

Die Modellierung von Prozessen im Lebenszyklus einer Einheit erfordert ähnliche Metainformationen, um die Kongruenz verschiedener Modelle bewerten zu können. Es muss erkennbar sein

1. auf welche Aspekte der Einheit und ihrer Umgebung sich ein Prozessmodell bezieht;

2. welche Annahmen bei der Modellierung getroffen wurden und

3. in welchem Rahmen das Modell generalisierbar ist.

Wenn zwei Modelle unterschiedliche Aspekte einer Entität betreffen, so kann es nicht zu direkten Widersprüchen kommen. Modellieren sie dieselben Aspekte, so können die dabei ermittelten Daten durchaus voneinander abweichen, da Modelle die Realität grundsätzlich nicht vollständig beschreiben. Im Rahmen getroffener Annahmen und Vereinfachungen können sich ermittelte Werte folglich unterscheiden. Dies kann durch Auswertung der genannten Metainformationen erfasst und bewertet werden. Dasselbe gilt für die Modellierung verschiedener Aspekte, die in der Realität physikalisch verkoppelt sind. Unterschiedliche Modelle können widerspruchsfrei koexistieren. Wenn beide Modelle dabei exakt die gleichen Aspekte beschreiben, kann eine Zusammenfassung sinnvoll sein. Eine andere Möglichkeit zu einem Widerspruch zwischen Modellen zu kommen, ist die Anwendung eines Modells außerhalb seines Gültigkeitsbereichs. In diesem Fall ist das entsprechende Modell nicht auszuwerten, da es für den fraglichen Fall nicht zutrifft. Es handelt sich hier um keinen Konflikt der Modelle selbst. Auch dieser Fall ist mit den Metainformationen erfassbar.

4.2 Lebenszyklusbeschreibungen in der Verwaltungsschale

Die bisherigen Ausführungen gelten für die Beschreibung des Ist-Lebenszyklus einer realen Einheit. Eine solche Beschreibung ist ein Kernelement der Verwaltungsschale der Einheit, da die Erfassung von tatsächlichen Zuständen einer Einheit die grundlegendste Form ihrer Verwaltung ist. Selbst die einfachste Form abgelegter Ist-Zustände ergibt, sofern jedem Zustand ein Zeitstempel zugeordnet ist, eine grobe Lebenszyklusbeschreibung.

Zusätzlich kann eine Verwaltungsschale weitere Lebenszyklusbeschreibungen oder andere Modelle im Zusammenhang mit der Einheit beinhalten, bspw. einen Plan für ihre weitere Zukunft oder Simulationen zukünftiger Zustände. Solche Beschreibungen sind Teil des Modells der Einheit und bilden nicht zwangsläufig die Realität ab. Daher können derartige Beschreibungen als Modelle der Einheit gesehen werden, die ihrerseits einen eigenen Lebenszyklus haben. Die Lebenszyklen einer Einheit und ihrer Modelle sind dementsprechend entkoppelt. Abbildung 4.6 zeigt dies. Es ist zu erkennen, dass das Modell durchaus vor der Einheit vorhanden sein und weiterhin länger existieren kann. Dies ist bspw. bei Planungsmodellen für den Einsatzzweck einer Einheit der Fall. Sie legen vorab die Randbedingungen des Einsatzes fest und existieren auch weiter, wenn eine vormals eingesetzte Einheit durch eine andere ersetzt wurde. Trotzdem beziehen sie sich auf konkrete, reale Einheiten.

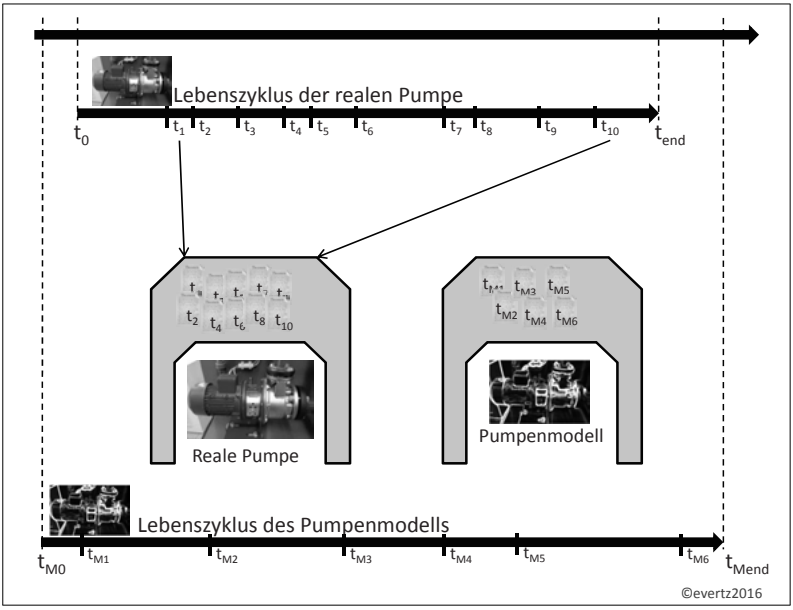


Abbildung 4.6: Die Einheit und ihr Modell haben entkoppelte Lebenszyklen, die ebenso entkoppelt verwaltet werden. Nichtsdestotrotz bezieht sich das Modell auf die Reale Einheit.

Auch Beschreibungen von Produkttypen sind solche Modelle. Diese Typen sind selbst gedankliche Einheiten. Sie können in ihrem Lebenszyklus gleichwertig wie physische Einheiten behandelt werden. Dabei ist zu beachten, dass es nach der Entstehung einer Einheit keine Verkoppelung mehr zu ihrem Typen gibt. Daher bietet sich die separierte Verwaltung, wie in Abbildung 4.6, in diesem Fall an.

Soll der Lebenszyklus eines Modells nicht verfolgt werden, so kann es in die Verwaltungsschale der zugehörigen Einheit integriert werden, wie in Abbildung 4.7 dargestellt. Wie im unteren Bereich der Abbildung dargestellt ändert eine solche Integration nichts am Lebenszyklus des Modells. Lediglich auf seine Nachverfolgung wird verzichtet. Eine solche Integration eines Modells in die Verwaltungsschale der zugehörigen Einheit ist sinnvoll, wenn das Modell selbst vergleichsweise statisch ist und sehr eng mit der Einheit verkoppelt. Das zum Erwerbszeitpunkt einer Pumpe für diese gültige Lastenheft wäre ein Beispiel für ein solches Modell. Dabei hätte wiederum das Lastenheft einen eigenen, völlig entkoppelten Lebenszyklus, aus dem ein Ausschnitt für die konkrete Pumpe gültig ist und als Planungsmodell für sie dient.

Eine dritte Möglichkeit ist, ein Modell einer Einheit zusammen mit seiner Verwaltungsschale in die Verwaltungsschale der Einheit zu integrieren. Abbildung 4.8 zeigt diesen Ansatz. In diesem Falle werden die Lebenszyklen von Modell und realer Einheit separat verfolgt, aber über einen gemeinsamen Zugriffspunkt verfügbar gemacht. Dies macht Sinn,

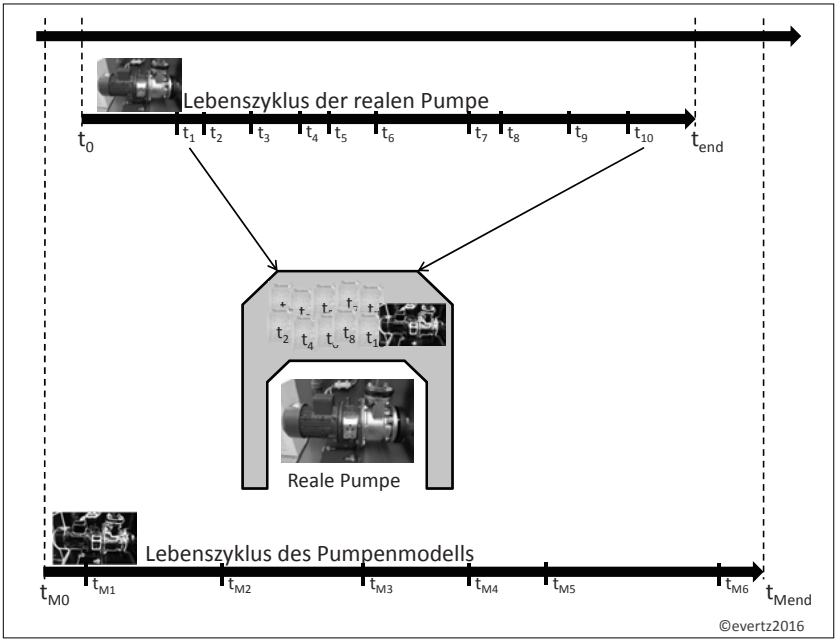


Abbildung 4.7: Die Einheit und ihr Modell haben entkoppelte Lebenszyklen. Der Lebenszyklus des Modells wird nicht verwaltet. Es wird als Teil der Verwaltungsschale der Einheit gehandhabt.

wenn Modell und Einheit eng verkoppelt sind, aber beide Lebenszyklen von Interesse sind, z. B. weil das Modell ständig aktualisiert wird.

Letztendlich ist es eine Designentscheidung, wie Modelle von Einheiten und andere zu den Einheiten gehörige Informationen in den Verwaltungsschalen gekapselt sind. Die Konstante ist, dass die Abbildungen des Ist-Lebenszyklus von Einheiten aller Art in ihren Verwaltungsschalen vorhanden sind. Dabei kann es vorkommen, dass dieselbe Einheit mehrere Verwaltungsschalen in unterschiedlichen Organisationen hat. In diesem Fall hält jede Organisation einen für sie interessanten Teil der Lebenszyklusabbildung der Einheit. Diese können unter den in Abschnitt 4.1 beschriebenen Bedingungen zusammengeführt werden. Daher müssen hier die angesprochenen Metainformationen Anwendung finden.

Realistisch betrachtet wird die voranschreitende Digitalisierung dazu führen, dass zu jeder Einheit eine Vielzahl unterschiedlicher Modelle existieren wird. Diese Modelle reichen von Strukturmodellen, Verhaltensmodellen und Einsatzplanung bis zu vorausschauenden Simulationen des Verhaltens der Einheit in einer bestimmten Situation. Dabei haben alle diese Modelle eigene Lebenszyklen, wie Abbildung 4.9 andeutet. Ferner gibt es Modelle, die nicht die konkrete Einheit, sondern ihren Typ beschreiben. In diesem Fall korreliert nur eine Version des Modells bzw. ein Zeitpunkt in seinem Lebenszyklus mit der konkreten Einheit (Abbildung 4.9 unten links). Wird der Typ einer konkreten Einheit weiterent-

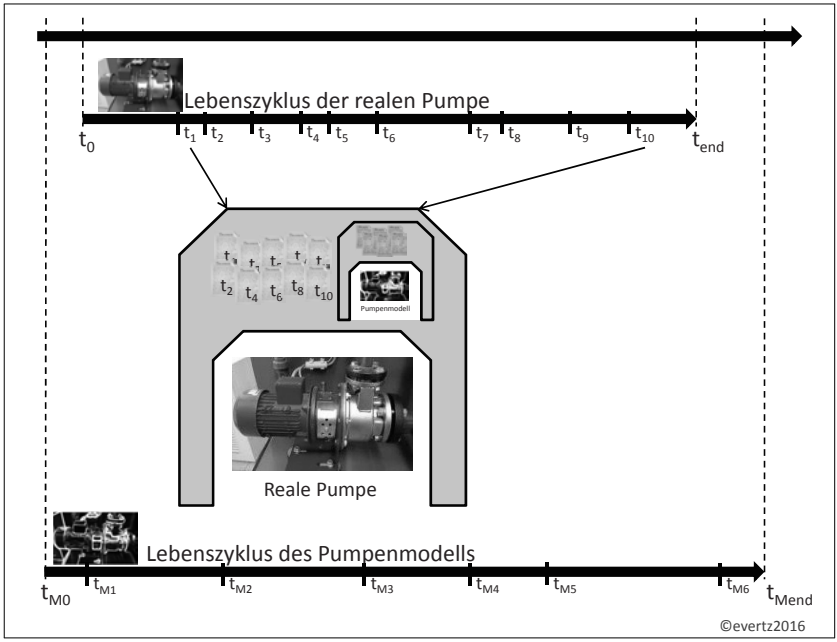


Abbildung 4.8: Die Einheit und ihr Modell haben entkoppelte Lebenszyklen. Die Verwaltungsschale des Modells wird in die Verwaltungsschale der Einheit integriert.

wickelt, so hat dies keine Auswirkungen auf sie. Der weiterentwickelte Typ beschreibt die Einheit folglich nicht mehr korrekt. Daher ist eine Korrelation von Typmodellen und konkreten Einheiten nur unter Bezugnahme auf die Version der Typbeschreibung, also den spezifischen Zeitpunkt in ihrem Lebenszyklus möglich.

Alle eine Einheit beschreibenden Modelle können miteinander in Beziehung gesetzt werden. Dies gilt sowohl für solche Modelle, die die konkrete Einheit beschreiben, als auch für solche, die ihren Typ beschreiben. Die Zusammenführung der Informationen kann über die Verwaltungsschale der Einheit passieren, wie in Abbildung 4.10 dargestellt. Dabei ist zu beachten, dass jedes Modell einen eigenen Lebenszyklus hat und eine eigene Verwaltungsschale haben kann. Nur die Beschreibung vergangener Ist-Zustände der konkreten Einheit selbst ist Teil ihrer Verwaltungsschale und hat demnach keine eigene.

Die zuletzt angesprochenen Modelle betreffen nicht mehr ausschließlich den Ist-Zustand einer Einheit. Es können bspw. auch Soll-Zustände modelliert werden. Solche verschiedenen Sichtweisen können unter bestimmten Gesichtspunkten verglichen und in neue Modelle überführt werden. Daher ist es wichtig, für jede Zustandsbeschreibung zu wissen, ob es sich um einen gemessenen Wert (wie vorab bei der retrospektiven Ist-Beschreibung der Einheit), um eine Zusicherung, eine Anforderung oder noch etwas anderes handelt. Durch die Verbindung der Semantik der gemachten Aussagen können weitere Schlüsse gezogen werden. Der Vergleich von Zusicherungen und Anforderungen bspw. ermöglicht die Planung

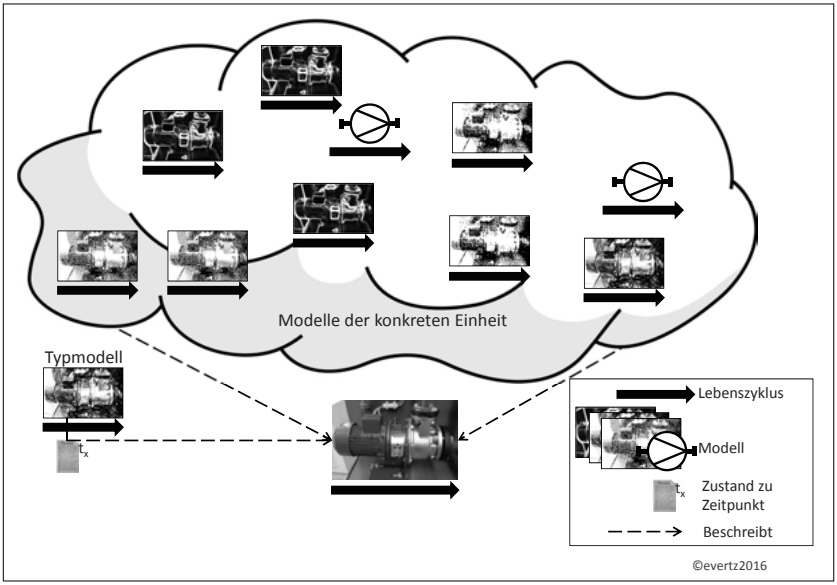


Abbildung 4.9: Zu jeder konkreten Einheit kann es eine Vielzahl an Modellen geben, die sie unter verschiedenen Betrachtungsweisen beschreiben. Jedes Modell hat seinen eigenen, von der Einheit entkoppelten Lebenszyklus. Trotzdem hat es zu jedem Zeitpunkt den Anspruch die Einheit zu beschreiben. Des Weiteren gibt es Modelle, die den Typ der Einheit beschreiben. Auch sie haben einen eigenen Lebenszyklus. Da der Typ unabhängig von der konkreten Einheit weiterentwickelt wird, bezieht sich immer nur ein bestimmter Zeitpunkt in seinem Lebenszyklus auf eine konkrete Einheit (links unten).

einer geeigneten Einheit. Die Abweichung von Anforderungen und Ist-Zuständen zeigt, dass eine Einheit für ihren Einsatz nicht (mehr) geeignet ist.

Bei der Zusammenführung verschiedener Modelle ist analog zur Zusammenführung der retrospektiven Ist-Beschreibung der Einheit Konsistenz sicherzustellen. Wie zuvor sind Widersprüche zwischen den Modellen auflösbar, wenn die Randbedingungen der Modelle und ihre Aussagegenauigkeit bekannt sind. Hinzu kommt bei der Betrachtung von Typbeschreibungen ihre von der konkreten Einheit unabhängige Entwicklung. Wie bereits erwähnt, trifft nur eine Version einer Typbeschreibung bzw. ihr Zustand zu einem bestimmten Zeitpunkt auf die konkrete Einheit zu. Daher muss bei der Bezugnahme auf ein Typmodell exakt dieser Zustand referenziert werden. Ob dies durch Angabe einer Versionsnummer oder durch einen Zeitpunkt passiert, ist unerheblich, sofern der Zustand exakt erfahrbar ist.

Die Versionierung und zeitliche Entwicklung von Modellen der konkreten Einheit hat einen geringeren Einfluss, da diese Modelle, wie bereits erwähnt, immer den Anspruch haben, die Einheit zu beschreiben. Trotzdem können Güte und Aussagekraft eines solchen Modells einer zeitlichen Entwicklung unterliegen. Besonders auffällig ist dies, wenn

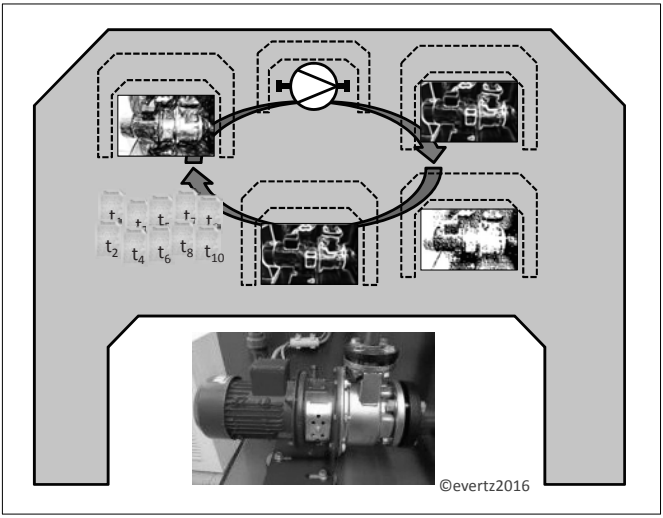


Abbildung 4.10: Die eine Einheit beschreibenden Modelle können miteinander in Beziehung gesetzt werden. Dies geschieht in der Verwaltungsschale als zentrales Verwaltungselement. Alle Modelle außer der retrospektiven Beschreibung der Ist-Zustände können dabei eigene Verwaltungsschalen besitzen. Die Retrospektive Ist-Beschreibung einer Einheit ist ist immer Teil der Verwaltungsschale der Einheit selbst.

grobe Modellfehler behoben werden. In diesem Fall ist es wichtig, dass die bereits in Abschnitt 4.1 angesprochenen Metainformationen nicht nur auf das entsprechende Modell, sondern zusätzlich bezogen auf die spezifischen, von ihm abgeleiteten Aussagen zu bestimmten Zeitpunkten bezogen werden können.

Grundlegend sei angemerkt, dass die angesprochenen Metainformationen nur die Grundlage bilden, um Aussagen von Modellen und tatsächliche Zustände einer Einheit in Bezug zu setzen. Inkonsistenzen, die innerhalb der angegebenen Genauigkeitsbereiche und Anwendungsgrenzen der Modelle liegen, stellen keine Widersprüche dar. Gehen die Inkonsistenzen darüber hinaus, muss ein Fehler in der Datenerfassung oder einem Modell vorliegen. Diesen zu finden kann durchaus komplex sein und ist nicht zwingend automatisierbar. Die Erkennung seiner Existenz ist jedoch ein wichtiger Schritt in der Verbindung der Daten.

5 Entwicklung einer Systemstruktur

In Kapitel 3 zeigte sich, dass die bestehenden Software-Systeme spezifische Fokusse haben und keine gemeinsam genutzten Datenmodelle verwendet werden. Daher wird in diesem Kapitel ausgehend von den generischen Referenzmodellen für Entitäten und Lebenszyklen eine Systemstruktur entwickelt, die eine Verwaltung der Ist-Lebenszyklen beliebiger Einheiten erlaubt. Dabei werden die in Kapitel 4 vorgestellten Zusammenhänge der verschiedenen Modelle umgesetzt. Nach einer Beschreibung der zugrunde liegenden Datenmodelle von Entitäten und Lebenszyklen werden die Schnittstellen zum Umgang mit diesen Modellen entwickelt. Darauf folgt eine Beschreibung der einzelnen Systemkomponenten. Anschließend wird die Verteilung dieser Komponenten erörtert, wobei ein Schwerpunkt auf die Verteilung über Unternehmensgrenzen hinweg gelegt wird. Als Nächstes wird die Interaktion zwischen einzelnen Komponenten erklärt um abschließend auf das Zusammenwirken aller Komponenten in einem unternehmensübergreifenden Netzwerk einzugehen.

5.1 Rekapitulation der Anforderungen

Ziel dieser Arbeit ist es, die über den Lebenszyklus einer Entität anfallenden Informationen für alle damit verbundenen Organisationen nutzbar zu machen. Dies impliziert eine Reihe von Anforderungen.

Zunächst muss für alle Beteiligten eine gemeinsame Semantikbasis geschaffen werden, die die Interpretation der zu einer Entität aufgenommenen Daten ermöglicht. Dabei muss die Semantikbasis flexibel genug sein, nicht vorausgesehene Eigenschaften abzubilden, bspw. wenn in verschiedenen Organisationen dieselbe Entität mit unterschiedlichen Eigenschaften verwaltet wird. Dies ist nur mit einem dynamischen Modell jeder Entität möglich. Um ein solches zu interpretieren, bedarf es eines gemeinsamen Metamodells der Entitätsinformationen.

Ein derart flexibles Datenmodell für Entitäten erlaubt es auch, verschiedene Sichtweisen zu verwalten. Dabei muss abgebildet werden, welche Informationen für welche Sichtweise von Interesse sind und ob diese mit anderen Organisationen geteilt werden dürfen. Dies muss jedoch nicht explizit im Datenmodell abgebildet werden, sondern kann separat verwaltet werden. Des weiteren müssen semantische Doppelungen von modellierten Entitätseigenschaften aufgelöst werden können.

Die eine Entität abbildenden Modelle müssen mit den in Kapitel 4 vorgestellten Metainformationen verbunden werden, damit die verschiedenen Modelle in Beziehung gesetzt werden können. Ebenso müssen Aussagen zu den Eigenschaften der Entität mit den Metainformationen versehen werden, um die verschiedenen Aussagen vergleichen und bewerten zu können. Teilweise kann diese Aufgabe durch die gemeinsame Semantikbasis übernommen werden.

Zusammenhänge zwischen den Eigenschaften einer Entität und denen anderer Entitäten oder Subkomponenten sollten abbildbar sein. Dies vereinfacht die Aggregation und Auf-

bereitung von Informationen. Zudem können so auch aktive Modelle der auf eine Einheit wirkenden Prozesse eingebunden werden, um daraus neue Information zu generieren. Um die abgebildeten Zusammenhänge automatisch auszuwerten oder für Berechnungen heranzuziehen, sollte ihre Modellierung nicht natürlich sprachlich, sondern formal erfolgen.

Die entwickelten Modelle müssen sowohl auf physische als auch auf gedankliche Einheiten anwendbar sein. Dabei sind Pläne und Beschreibungen gedankliche Einheiten, die von ihren Trägern separiert verwaltet werden sollen. Gedankliche Einheiten sind also abstrakte Informationsgebilde, die mehrere Träger haben können, die ihrerseits der physischen bzw. der Informationswelt zuzuordnen sind.

Um Informationen unternehmensübergreifend verfügbar zu machen, bedarf es außerdem einer definierten Schnittstelle für die angesprochenen Datenmodelle. Diese Schnittstelle sollte aus Gründen der Interoperabilität so weit wie möglich auf verbreitete und offen definierte Protokolle aufsetzen. Außerdem muss die Möglichkeit der Authentifizierung und Verschlüsselung bestehen, um die Rechtmäßigkeit (im Sinne des erlaubten Zugriffs) und Vertraulichkeit der Interaktionen abzusichern.

Besagte Schnittstelle sollte dabei so gewählt sein, dass vorhandene PLM-, PDM- und Asset Management-Systeme integriert werden können. Da diese bisher genutzt werden, um die anfallenden Daten zu sammeln und zu verarbeiten, ist dies nötig, um den Zugriff auf die Daten in einer für den Nutzer bekannten Art und Weise zu erhalten und gleichzeitig die darin vorhandenen Daten für andere Nutzer zugreifbar zu machen.

In einem Gesamtsystem muss jederzeit erfahrbar sein, welchem Kontext ein spezielles Datum zuzuordnen ist, um daraus Informationen zu generieren. Daher muss die Semantik aller Informationseinheiten genau wie die Informationen selbst über eine definierte Schnittstelle verfügbar sein. Für diese gelten die gleichen Anforderungen wie für die der Entitätsinformationen.

Bei aller Potenz des Gesamtsystems darf die intuitive Verständlichkeit nicht vernachlässigt werden. Dazu ist es sinnvoll, ein möglichst einfaches Metamodell zu verwenden und die Schnittstellen präzise darauf abzustimmen. Dies vergrößert zwar die jeweiligen nach dem Metamodell aufgebauten Modelle, da jedes einzelne Element nur wenig Semantik trägt, muss aber keine Einbußen an Verständlichkeit für sie nach sich ziehen, da durch geeignete Wahl der Sichtweise auf die Modelle die dargestellte Komplexität verringert werden kann.

Die in den folgenden Kapiteln entwickelten Modelle und Schnittstellen zielen auf die Erfüllung dieser Anforderungen ab. Darauf folgend wird unter Nutzung der Modelle und Schnittstellen ein Gesamtsystem mit verteilten Komponenten entwickelt.

5.2 Entitätsdatenmetamodell

Wie in Abschnitt 2.2 beschrieben wird eine physische oder gedankliche Einheit zur Entität, wenn sie in einem Informationssystem verwaltet und in ihrem Lebenszyklus verfolgt wird. Die Einheiten selbst können verschiedenster Natur sein, sollen aber einheitlich verwaltet werden. Die Einbeziehung gedanklicher Einheiten, also solcher, die der Informationswelt zuzuordnen sind, ist dabei ein besonderer Fall.

Gedankliche Einheiten sind u. a. Modelle und Beschreibungen. Sie haben zwar immer mindestens einen Träger in der physischen Welt, sind aber in ihrem Lebenszyklus von

diesem losgelöst zu betrachten[1]. Dies führt dazu, dass sämtliche Beschreibungen von Anlagen, Prozessen, Zuständen etc. per Designentscheidung selbst zu Entitäten gemacht werden können. Sogar die Verwaltungseinheiten von Entitäten könnten selbst als Entitäten verwaltet werden.

Demnach müssen verschiedenste Arten von Einheiten mit demselben System einheitlich verwaltet werden können. Hinzu kommt, dass, wie in Kapitel 4 beschrieben, unterschiedliche Beteiligte unterschiedliche Interessen an derselben Einheit haben und daher andere Eigenschaften der Einheit in ihren Modellen abbilden. Dies erfordert eine dynamisch veränderbare Beschreibung der Einheiten. Dabei müssen selbstverständlich die in Abschnitt 4.1 genannten Metainformationen einbezogen werden, um Konsolidierung und Vergleich der vorhandenen Daten zu ermöglichen. Daher wird im Folgenden ein Metamodell der Einheitenbeschreibung entwickelt, dass die beiden genannten Kriterien erfüllt und den ersten Schritt der Umsetzung eines Verwaltungssystems darstellt.

Eine Entität besteht immer aus einer Einheit (dem „Ding“) in der physischen oder der Informationswelt und mindestens einer Repräsentation in der Informationswelt, die den Zustand der Einheit zu einem bestimmten Zeitpunkt beschreibt. Da die Einheit selbst beiden Welten angehören kann, werden in Abbildung 5.1 die Begriffe „Einheitenebene“ (links) und „Verwaltungsebene“ (rechts) gewählt. Jede Einheit hat bestimmte inhärente Eigenschaften, die sich kontinuierlich verändern können. In der Verwaltungsebene gibt es zu jeder Einheit mindestens eine zugeordnete Verwaltungseinheit. Diese kapselt die einer Einheit zugeordneten Modelle. In Abbildung 5.1 ist dabei nur die zeitbezogene Modellierung der Eigenschaften der Einheit dargestellt. Diese ist in zeitbezogene Abbilder unterteilt, die per Designentscheidung festgelegte Zeiträume oder Zeitpunkte gliedern. Sie dienen ausschließlich der Datenorganisation. In ihnen sind semantische Beschreibungen der erfassten Eigenschaften gekapselt. Diese legen unter anderem die in Abschnitt 4.1 erörterten Metainformationen zu einer Eigenschaft fest. Die Aussagen zu den eigentlichen Werten der Eigenschaften werden durch Ausprägungsaussagen getroffen. Diese beschreiben den Wert der Eigenschaft zu einem bestimmten Zeitpunkt. Sie können weiterhin festlegen, welche Art von Aussage zu besagtem Wert getroffen wird (Messung, Zusicherung, Anforderung etc.), wobei in der retrospektiven Ist-Beschreibung nur Messungen der tatsächlichen Werte eine Rolle spielen.

Reale Einheiten können verschachtelt sein, wie ein Motor der Teil eines Kfz ist. Dies wird in der Verwaltungsebene durch die Referenzierung der Repräsentation einer übergeordneten Einheit abgebildet, wobei die Verschachtelung selbst als Eigenschaft angesehen wird. Die Beschreibung der einzelnen Entität ist damit unabhängig von eventuellen Organisationshierarchien, kann diese aber darstellen.

Zwischen den Eigenschaften von Einheiten können Zusammenhänge bestehen. Dies gilt sowohl innerhalb einer Einheit selbst (Die Masse eines Körpers steht im Zusammenhang mit seinem Volumen.), zwischen einer Einheit und ihren Untereinheiten (Solange ein Motor in einem Kfz eingebaut ist, steigt die Betriebsstundenzahl für beide gleich.) und zwischen anderen wechselwirkenden Einheiten (Die Innenraumtemperatur eines Kfz wird sich der Innenraumtemperatur der Garage, in der es abgestellt ist, anpassen.). Solche Zusammenhänge werden - sofern sie von Interesse sind - durch Zusammenhangsmodelle abgebildet. Diese sind immer unidirektional und sollten so weit wie möglich formalisiert und

[1] Die einzige Abhängigkeit ist, dass eine gedankliche Einheit mit dem Vergehen ihres letzten Trägers vergessen wird. Die Behandlung dieses Falles kann jedoch außerhalb des Modells erfolgen.

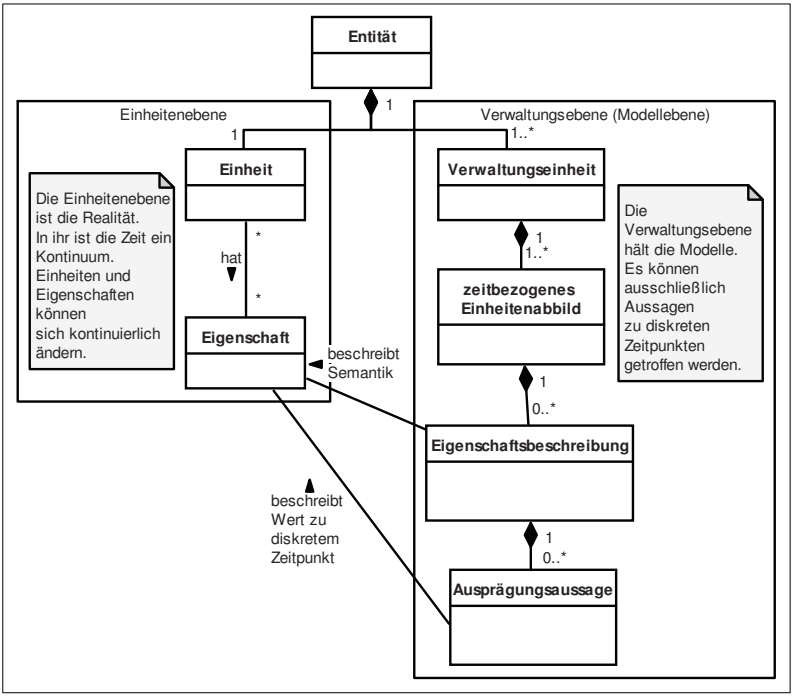


Abbildung 5.1: Eine Entität besteht aus der realen Einheit mit ihren Eigenschaften und mindestens einer Verwaltungseinheit, die diese Eigenschaften abbildet. Verwaltungseinheiten kapseln zeitbezogene Abbildungen der Einheit. In diesen wird die Semantik der erfassten Eigenschaften beschrieben. Die Semantikbeschreibungen wiederum kapseln Aussagen zu den Ausprägungen der Eigenschaften zu diskreten Zeitpunkten.

nicht natürlich sprachlich ausgedrückt werden. Sie werden in der Verwaltungsebene als Teil der zeitbezogenen Einheitenabbilder gehalten, da sie sich auf die darin abgebildeten Eigenschaften beziehen und nicht zwingend zeitlos gültig sind. Abbildung 5.2 zeigt dies.

Die dargestellten Klassen können direkt in die Implementierung eines Verwaltungssystems einfließen. Außerdem gibt es Referenzen, die

1. Zugehörigkeiten ausdrücken von Eigenschaftsbeschreibungen zu zeitbezogenen Einheitenabbildern, von Ausprägungsaussagen zu Eigenschaftsbeschreibungen und von Zusammenhangsmodellen zu zeitbezogenen Einheitenabbildern;

2. Zusammenhangsmodelle und zugehörige Eigenschaftsbeschreibungen verknüpfen und

3. Die verwaltete Einheit als Untereinheit einer anderen Einheit kennzeichnen.

Die erstgenannte Art von Referenzen ist in Abbildung 5.1 als Aggregation dargestellt. Damit wird ausgedrückt, dass bei der Übertragung von Entitätsdaten mit einem

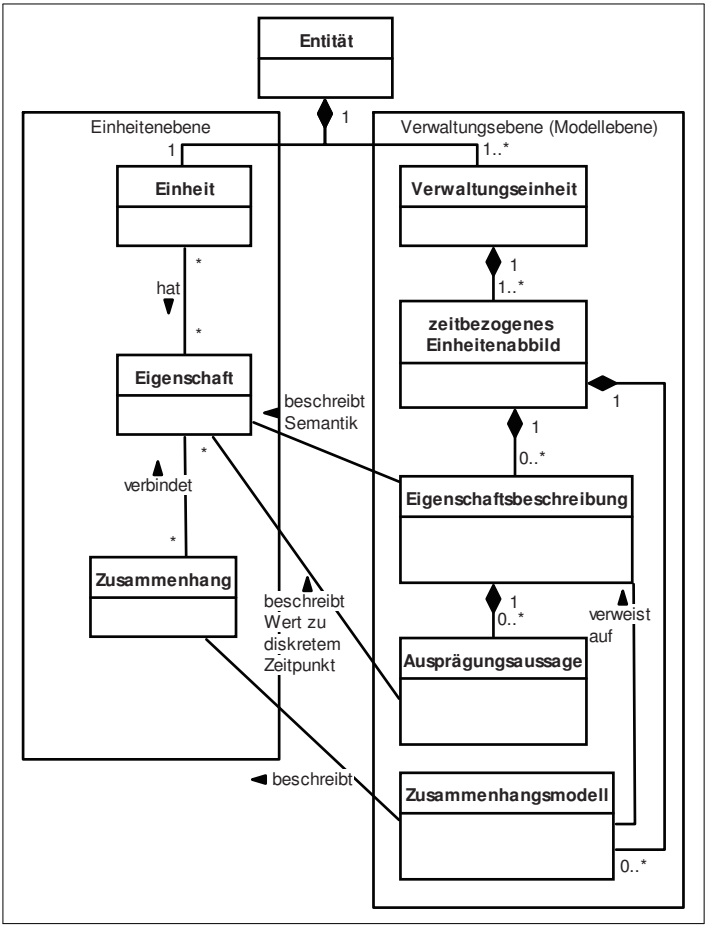


Abbildung 5.2: Eigenschaften der realen Einheiten können miteinander zusammenhängen. Dies kann durch entsprechende Modelle beschrieben werden, die den betreffenden Verwaltungseinheiten zugeordnet werden können.

Übertragungsprotokoll, das eine Baumstruktur nutzt (z. B. XML), diese als Darstellung genutzt werden soll.

Da in einem Verantwortungsbereich häufig gleichartige Einheiten verwaltet werden, ist es hinsichtlich der Nutzerfreundlichkeit sinnvoll, den zeitbezogenen Einheitenabbildern Prototypen zu geben. Diese folgen ebenfalls der in Abbildung 5.1 rechts dargestellten Struktur. Dabei enthält ein Prototyp keine Ausprägungsaussagen zu tatsächlichen Werten, da keine reale Einheit zugeordnet ist. Gleichwohl können mit Ausprägungsaussagen Wertebereiche

eingeschränkt oder Zusicherungen etc. getroffen werden. Durch Kopieren des Prototyps wird dann eine neue zeitbezogene Einheitenabbildung erzeugt.

5.3 Metamodell der Lebenszyklusabbildung

Der Lebenszyklus einer Einheit umfasst alle Prozesse, die während der Existenz der Einheit auf sie einwirken. Zu jedem Zeitpunkt befindet sich die Einheit in genau einem Prozess. An den Übergangszeitpunkten zwischen zwei Prozessen kann ein Übergangszustand gesehen werden, der keine zeitliche Ausdehnung hat. Jede Einheit durchläuft mindestens einen Entstehungs- und einen Vergehensprozess. Dies ist in Abbildung 5.3 dargestellt. Die Modellierung der verschiedenen Prozesse ist eine Designentscheidung des Betrachters. Nur dieser kann absehen, welche Vorgänge für ihn von Interesse sind, und welche Zeitabschnitte in sein Modell des Einheitenlebenszyklus einfließen sollen.

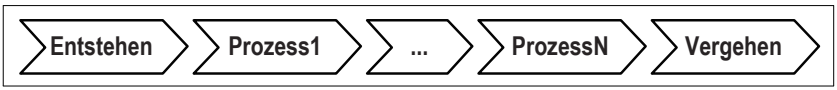


Abbildung 5.3: Aufbau eines allgemeinen Lebenszyklus nach [25]

Daher wird ein Betrachter Zeiträume im Lebenszyklus einer Einheit definieren, in denen er Daten der Einheit erhebt und auswertet. Dabei ist davon auszugehen, dass derartige Zeiträume mit auf die Einheit wirkenden Prozessen korrelieren, die der Betrachter zuvor als Grundlage seiner Abbildung des Einheitenlebenszyklus definiert hat. Innerhalb dieser Zeiträume werden dann zu diskreten Zeitpunkten die Eigenschaften der Einheit erfasst. Die modellierten Prozesse hingegen wirken über die Zeiträume kontinuierlich auf die Einheit. Die Konsolidierung der von verschiedenen Beteiligten erstellten Modelle erfolgt sowohl für erfasste Daten als auch für modellierte Prozesse und Zusammenhänge unter den in Abschnitt 4.1 dargestellten Bedingungen.

Die Beschreibung des Lebenszyklus einer Einheit kann nach dem in Abbildung 5.4 dargestellten Metamodell erfolgen. Die in Abbildung 5.4 oben eingerahmten Klassen dienen nur der Einordnung der weiteren Klassen. Sie sind nicht als im Informationssystem instantiierte Objekte zu verstehen. Vielmehr wird die Lebenszyklusbeschreibung einer Entität durch die Menge an zeitbezogenen Einheitenabbildern und deren Verbindungen zu modellierten Prozessen und Ereignissen umgesetzt.

Ereignisse sind dabei als in der Realität abgelaufene Vorgänge zu verstehen, die nicht semantisch auswertbar beschrieben sind, aber mit einer Einheit in Verbindung stehen und daher nicht außer Acht bleiben sollen. Sie sind sozusagen als nutzerspezifische Annotationen diskreter Zeitpunkte im Lebenszyklus der Einheit zu verstehen. Ereignisse können wiederum auf der Zeitschiene eingeordnet und betrachtet werden und sind als Abbildungen der Realität in sich konfliktfrei. Da ihnen jedoch keine global eindeutige Semantik beigestellt wird, können sie nur nutzerspezifisch ausgewertet werden.

Prozesse hingegen werden semantisch eindeutig beschrieben. Dabei kann diese Beschreibung aus einer Reihe von Beschreibungen einfacherer Zusammenhänge bestehen. Diese zusätzliche Ebene an Granularität fördert die Wiederverwendbarkeit der Beschreibungen. Ferner lassen sich einfache Zusammenhänge besser in formale Modelle gießen, die

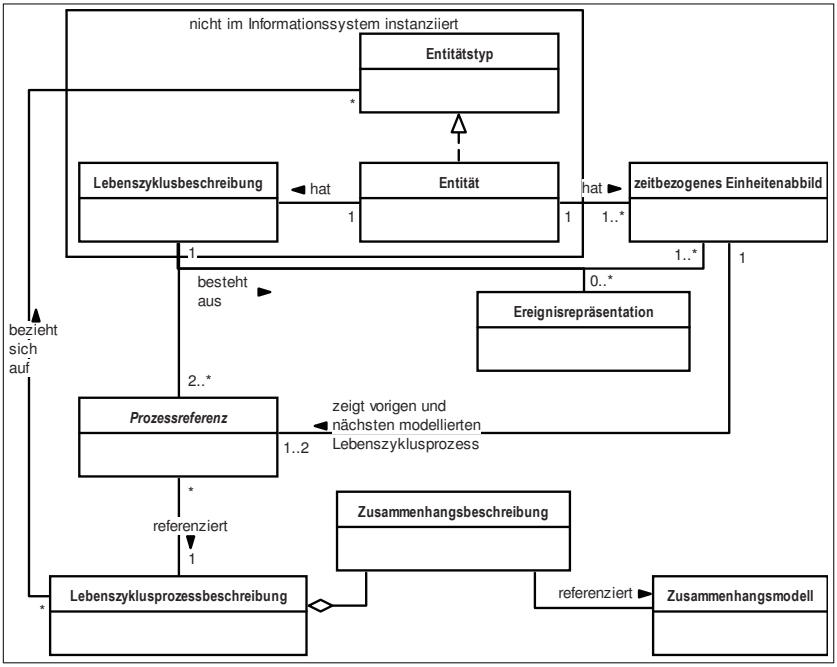


Abbildung 5.4: Metamodell für die Beschreibung von Lebenszyklen in einem Informationssystem.

zur Unterfütterung der Beschreibung referenziert werden können (unten rechts in Abbildung 5.4). Solche formalen Modelle können in einem entsprechenden Laufzeitsystem automatisch auf erfasste Entitätsinformationen angewandt werden. Dadurch ergeben sich weitere Möglichkeiten der Nutzung der erfassten Daten. Die Klasse *Prozessreferenz* ist in Abbildung 5.4 als abstrakt dargestellt. Dies bedeutet, dass sie nicht im Informationssystem instantiiert werden soll, sondern durch Elemente der *zeitbezogenen Einheitenabbilder* umgesetzt wird.

5.4 Datenmodell für die Umsetzung der Lebenszyklusverwaltung

Im Folgenden werden die beschriebenen Ideen zu einem konkreten Datenmodell weiterentwickelt. Dabei werden die in Abbildung 5.2 und 5.4 dargestellten Klassen direkt umgesetzt und mit den nötigen Attributen versehen. Eigenschaftsbeschreibungen und Ausprägungsaussagen sind durch diese Attribute vollständig definiert. Die Beschreibung einer Einheit, also die Menge der in einer zeitbezogenen Einheitenabbildung verwalteten Informationen, hingegen wird maßgeblich durch die untergeordneten Objekte umgesetzt.

Es ergeben sich die in Abbildung 5.5 dargestellten Verwaltungsobjekte, mit denen sich eine Entitätenverwaltung umsetzen lässt. Die abstrakte Klasse *Verwaltungsobjekt* dient als Basis aller Objekte im Verwaltungssystem. Jedes Verwaltungsobjekt hat daher einen menschenlesbaren Klarnamen und einen eindeutigen maschinenauswertbaren Identifikator. Dieser setzt sich aus einer (textuellen) ID und einer Namensraum-URL zusammen, wie in Abbildung 5.6 dargestellt. Die Namensraum-URL spannt einen von ihrem Besitzer verwalteten Namensraum auf. Daher muss sie auf ihn registriert sein und sollte sich auf Dauer nicht verändern. Innerhalb dieses Namensraumes ist die dazugehörige ID des Verwaltungsobjekts einzigartig. Dies ermöglicht die dezentrale und flexible Verwaltung der Identifikatoren. Da registrierte URLs global eindeutig sind und gleichzeitig als Adresse in einem webbasierten Informationssystem dienen, werden sie zugleich als Adresse des technischen Systems verwendet, das das durch den Identifikator identifizierte Objekt verwaltet. Die Wahl der ID innerhalb des Namensraums ist dessen Verwalter überlassen. Selbstverständlich kann er dabei auch auf global eindeutige Identifikatoren wie bspw. GUIDs zurückgreifen. Eine Besonderheit stellt der Eintrag „metaModel“ als Namensraum-URL dar. Dieser ist offensichtlich kein URL. Er besagt, dass das genannte Element im Metamodell definiert ist. Dies ist für Merkmal- und Aussagebeschreibungen der Fall, die die semantische Grundlage bilden. Die Beschreibung dieser speziellen Merkmal- und Aussagebeschreibungen sind in Anhang A zu finden.

Unterhalb der allgemeinen Basisklasse werden die Verwaltungsobjekte in *erweiterbare* und *nicht-erweiterbare Verwaltungsobjekte* eingeteilt. Nicht-erweiterbare Verwaltungsobjekte sind durch ihre Attribute vollständig definiert, wohingegen erweiterbare Verwaltungsobjekte den größten Teil ihrer Semantik durch ihre Erweiterungen (z. B. durch Merkmale) erhalten.

Zeitbezogene Einheitenabbilder sind die Kernelemente der Beschreibung von Einheiten. Da sie eine zeitliche Abfolge haben, tragen sie eine laufende Nummer und zusätzlich den Identifikator der ersten Abbildung (laufende Nummer 0). Zusätzlich werden der Beginn und der Endzeitpunkt des Bezugszeitraums, ein Verweis auf die zeitlich folgende Einheitenabbildung und eine Liste von Verweisen zeitlich vorangegangener Abbildungen mitgeführt. Die Verweise verwenden den in Abbildung 5.6 abgebildeten Typen *Abbildreferenz*. Damit ist der Zeitbezug der Abbilder hergestellt. Hinzu kommt der Einheitsbezug. Dieser dient der Zuordnung der verwalteten Einheit. Dies kann bspw. über die Daten eines Typenschilds mit Seriennummer für Geräte oder völlig andere, an der Einheit vorhandene, Identifikationsmerkmale erfolgen. Außerdem wird der vorige und der folgende für das verwaltete Objekt modellierte Lebenszyklusprozess mit einem *Identifikator* (vergleiche Abbildung 5.6) referenziert. Zusätzlich wird festgehalten, ob es sich um eine physische oder eine gedankliche Einheit handelt. Als *erweiterbare Verwaltungsobjekte* erhalten *zeitbezogene Einheitenabbilder* ihre spezifische Semantik durch die Aggregation von Merkmalsrepräsentationen. Durch die frei definierbare Semantik der Merkmalsrepräsentationen können mit ihnen auch Strukturen umschrieben werden, wie bspw. die Angabe von unterlagerten oder übergeordneten Einheiten im Bezug zur verwalteten Einheit. (Dies ist nicht in Abbildung 5.5 dargestellt.).

Da die einzelnen Einheitsabbilder flexibel modelliert werden können, ist es möglich, auch alle anderen Verwaltungsobjekte in ihrem Lebenszyklus zu verfolgen. Wenn bspw. Beschreibungen von Merkmalen, Aussagen oder Einheiten in unterschiedlichen Organisationen unterschiedliche Freigabeprozesse durchlaufen müssen, kann dies problemlos abgebildet werden. Außerdem ist die Verwaltungsmethodik damit einheitlich, d. h. auch der Lebenszyklus jedes Typen kann mit denselben Methoden nachvollzogen werden, wie jeder

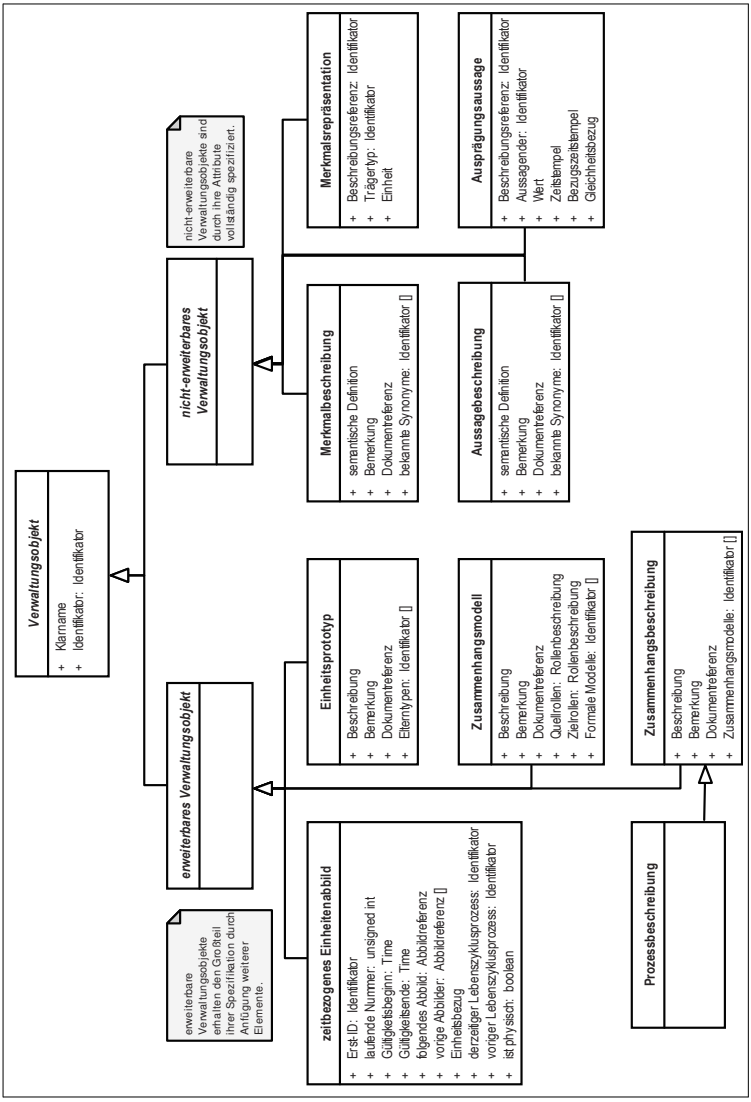


Abbildung 5.5: Informationstechnische Verwaltungsobjekte im Entitätenverwaltungssystem.

andere Lebenszyklus.

Einheitsprototypen dienen als Vorlagen für *zeitbezogene Einheitenabbilder*. Als Attribute tragen sie jeweils eine textuelle Beschreibung und eine Zusatzbemerkung. Diese sind als natürlich sprachliche Elemente gedacht. Außerdem haben sie eine Referenz (z. B. als URL) auf ein Dokument. Hier soll ein den Prototyp definierender Standard o. ä. eingetra-

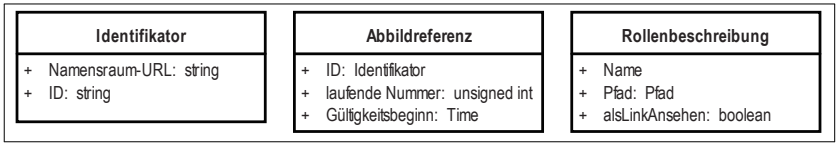


Abbildung 5.6: In den Modellelementen verwendete komplexe Datentypen.

gen werden. Schlussendlich halten sie eine Liste von Identifikatoren von Elterntypen. Es können mehrere Elterntypen sein, da diese nicht nur als Klassifikation, sondern auch zur Beschreibung unter verschiedenen Sichtweisen genutzt werden. So können z. B. Pumpen gleichzeitig als Feldgeräte (für die PLT) und als elektrischer Verbraucher (für die Energieverwaltung) und als Gegenstand des Anlagevermögens (für die Buchhaltung) verwaltet werden. Die typspezifische Semantik erhalten sie durch die Aggregation von Merkmalsrepräsentationen.

Merkmalbeschreibungen dienen der semantischen Definition von allgemeinen Merkmalen. Diese Definition wird in natürlicher Sprache in einem textuellen Attribut gehalten. Zusätzlich wird eine textuelle Bemerkung und eine Dokumentreferenz vergeben. Die Definition ist so zwar nicht maschinenlesbar, aber durch die eindeutige Identifikation der Merkmalbeschreibung kann automatisch die semantische Gleichheit zweier Merkmale durch Vergleich besagter Identifikatoren geprüft werden. Um dies weiter zu vereinfachen, wird noch eine Liste mit den Identifikatoren bekannter Synonyme, also semantisch gleicher Merkmalbeschreibungen, mitgeführt. Diese Liste wird manuell angepasst. Dies ermöglicht es einerseits, die semantische Gleichheit verschiedener Versionen einer Merkmalbeschreibung abzubilden und andererseits die semantische Gleichheit verschiedener Merkmalbeschreibungen, die von unterschiedlichen Organisationen definiert und verwaltet werden, darzustellen. Analog sind *Aussagebeschreibungen* als semantische Definitionen von *Ausprägungsaussagen* anzusehen. Sie haben die gleichen Attribute.

Merkmalsrepräsentationen beschreiben eine Eigenschaft eines Trägers in der Informationswelt. Sie tragen Attribute zur Referenzierung ihrer Beschreibung und des zugehörigen Trägertyps. Letzteres ist u. a. für die Verteilung der verschiedenen Verwaltungsobjekte hilfreich, da bei einer externen Beschreibung eines allgemeinen Merkmals der Bezug zum Trägertypen nur schwer feststellbar ist. Außerdem wird die Zuordnung des allgemeinen Merkmals zu einem abstrakten Einheitstypen vereinfacht. Beides ist notwendig, um beurteilen zu können, in welchen Sichtweisen ein Merkmal von Interesse ist. Ist eine Entität bspw. sowohl als Feldgerät, als auch als Vermögensasset modelliert, so ist durch die Referenzierung des letzteren Trägertyps am Merkmal *derzeitiger Gegenwert* nicht nur klar, wie dieses Merkmal zu verstehen ist, sondern auch, dass es für die wirtschaftliche und nicht für die prozesstechnische Sicht von Interesse ist. Des Weiteren wird die Einheit, in der die Eigenschaft gemessen wird festgelegt, sofern dies noch nicht in der Beschreibung des allgemeinen Merkmals erfolgt ist.

Ausprägungsaussagen versehen allgemeine Merkmale mit Werten spezifischer Bedeutung. So können gemessene, zugesicherte oder erwartete Werte festgelegt werden. Welche Semantik eine Ausprägungsaussage hat, wird in ihrer Beschreibung festgelegt, die im Verwaltungsobjekt referenziert wird. Außerdem wird derjenige, der die Aussage getroffen hat, der Wert und der Zeitpunkt des Treffens der Aussage abgelegt. Zusätzlich wird ange-

geben, auf welchen Zeitpunkt sich die Aussage bezieht. Dieser kann vom Zeitpunkt des Treffens der Aussage abweichen, da bspw. auch vorausschauende Simulationen Aussagen treffen können. Schlussendlich wird noch die Gleichheitsbeziehung (größer, kleiner, gleich, ungleich) der Aussage angegeben, sofern diese nicht bereits in der Aussagenbeschreibung definiert wurde.

Mit den bisher vorgestellten Elementen ist es möglich, die charakteristischen Eigenschaften einer Einheit zu beschreiben. Auch Strukturen, wie bspw. Kapselung von Einheiten, können durch Definition besonderer Merkmale abbildbar gemacht werden. Dabei kann mit Rollen gearbeitet werden. Der Einheitstyp Fahrzeug habe ein Merkmal *eingesetzter Motor*. Damit kann jede Einheit des Typs Fahrzeug den speziell von ihr verwendeten Motor per Referenz der Einheit des Typs Motor angeben. Damit ist es möglich, alle Einheiten separiert zu betrachten und gleichzeitig komplexe und mehrfache Bezüge zwischen den Einheiten abzubilden.

An diesem Punkt fehlt noch die Abbildung von Zusammenhängen zwischen Einheiten und / oder ihren Eigenschaften. Diese werden durch erweiterbare Verwaltungsobjekte des Typs *Zusammenhangsmodell* beschrieben. Dabei wird von gerichteten Zusammenhängen ausgegangen. Es gibt Quellen, die auf ein oder mehrere Ziele wirken. Die Zusammenhänge werden mit den bereits bekannten Attributen natürlich sprachlich beschrieben. Außerdem gibt es Listen mit Referenzen für Quell- und Zielrollen. Hier wird auf den Rollenbegriff zurückgegriffen, da in der Modellierung des Zusammenhangs noch nicht klar ist, welche Einheiten bzw. speziellen Merkmale bei der Anwendung des Modells zum Tragen kommen. Neben der textuellen Beschreibung kann ein Zusammenhangsmodell eine formale Beschreibung aufnehmen. Diese ist eine nutzerspezifische Erweiterung, die bspw. aus aggregierten Funktionsbausteinen oder einem mathematischen Modell bestehen kann. Dies ermöglicht gleichzeitig die Beschreibung und die Simulation des Zusammenhangs in einem Informationssystem. Ein derartiges Modell kann dabei die Beschreibung der Quell- und Zielrollen dazu nutzen, Ein- und Ausgangsgrößen im Informationssystem zu finden. Daher beinhalten Rollenbeschreibungen einen Pfad, der abwechselnd aus Identifikatoren von Merkmalbeschreibungen, Aussagebeschreibungen und Einheitstypen besteht. So können Beziehungen wie bspw. Kapselung in einer anderen Einheit aufgelöst werden. Alle Aussagebeschreibungen außer der letzten müssen vom Typ *metaModel:definedIdentificator* sein, um sicherzustellen, dass ein Identifikator folgt. Auf diese Art und Weise können Strukturen in einer Einheit verfolgt und die entsprechenden Eigenschaften gefunden werden.

Mehrere *Zusammenhangsmodelle* können mit einer *Zusammenhangsbeschreibung* aggregiert werden. Damit werden komplexe Zusammenhänge verbunden und können durch Angabe eines Prozessraumes zu einer *Prozessbeschreibung* erweitert werden.

Die beschriebenen Modellelemente erlauben die Beschreibung beliebiger Einheiten mit hoher Flexibilität hinsichtlich der abgebildeten Eigenschaften bei gleichzeitiger Unterfütterung mit semantischen Informationen. Die auf global eindeutigen Identifikatoren basierende Referenzierung ermöglicht die Verteilung der einzelnen Elemente über verschiedene Informationssysteme. Zu Beschreibung eines Zeitpunktes im Lebenszyklus einer Einheit reichen dabei die in Abbildung 5.2 dargestellten Elemente aus. Die weiteren Elemente dienen ihrer semantischen Beschreibung.

Alle Modellelemente sind so ausgelegt, dass sie in Form flacher Listen oder als Datenbank abgelegt werden können. Alle Referenzen können dabei über Identifikatoren und entsprechende Attribute umgesetzt werden, wenn das verwendete Informationssystem keine andere Referenzierungsmöglichkeit bietet. Außerdem ist der Austausch einzelner Datenstrukturen

systemübergreifend möglich, ohne Informationen zu verlieren. In der systemübergreifenden Lebenszyklusverwaltung von Entitäten ist die genaue Umsetzung der internen Datenhaltung nicht von Interesse. Wichtiger sind die Schnittstellen zwischen den Systemen und das ihnen zugrunde liegende Datenmodell. Die Schnittstellen werden im folgenden Kapitel beschrieben, wobei sie sich auf den vorgestellten Modellen für die Entitäts- und Lebenszyklusdatenhaltung abstützen.

Die Verteilung der einheitsbezogenen Informationen über verschiedene Informationssysteme bringt mit sich, dass die Erfassungszeitpunkte der Einheitseigenschaften und die modellierten Prozesse sich zwischen den Systemen unterscheiden. Nichtsdestotrotz kann der Informationsgehalt der aufgenommenen Daten übergreifend verwendet werden. Wenn in mehreren Informationssystemen dieselben Merkmale der Entität betrachtet werden, gestaltet sich ihre Nutzung besonders einfach. Da nicht alle Eigenschaften einer Einheit gleichzeitig in allen Informationssystemen erfasst werden können, sind die erfassten Informationen in den Systemen unterschiedlich. Wie bereits in Kapitel 4 angesprochen, lassen sich diese Daten direkt auf der Zeitschiene konsolidieren.

Des Weiteren können Ereignisse im Lebenszyklus einer Entität erfasst werden. Diese werden durch Ereignisrepräsentationen abgebildet. Sie sind nicht erweiterbare Verwaltungsobjekte und tragen als Attribute einen Klarnamen, einen Identifikator, den Zeitpunkt des Ereignisses, eine Liste betroffener Einheiten einen natürlich sprachlichen Betreff und eine natürlich sprachliche Beschreibung. In Abbildung 5.5 sind sie nicht dargestellt, da sie nicht der direkten Entitätenmodellierung dienen. Der Bezug eines Ereignisses zu einer Entität wird nur durch die Liste der betroffenen Einheiten abgebildet. Durch den Zeitpunkt kann auf den Zustand der Einheiten zu eben jenem Zeitpunkt geschlossen werden. Dies ist ausreichend, da das Ereignis selbst keinen direkten Bezug zu den Eigenschaften der Entität haben muss.

5.5 Schnittstellen zum Entitäts- und Lebenszyklusdatenaustausch

Um den Austausch der über den Lebenszyklus einer Entität gesammelten Informationen zu ermöglichen, bedarf es definierter Schnittstellen für alle beteiligten Systeme. Diese sind semantisch, wie auch syntaktisch und technologisch zu spezifizieren. Da es sich um lose gekoppelte und über Unternehmensgrenzen hinweg verteilte Systeme handeln soll, wird dienstbasierte Interaktion unter der Nutzung von Webtechnologien angestrebt. Die Nutzung von Webservices bietet sich an, da sie gut in die Unternehmens-IT integrierbar sind (vergleiche [33]) und es vielfältige Werkzeuge zur Implementierung sowohl der Dienstanbieter als auch der Nutzerseite gibt. Für den ebenenübergreifenden (unternehmensinternen) Datenaustausch kann die Schnittstelle zudem auf OPC-UA Datentypen und Methodenaufrufe abgebildet werden.

Die Schnittstelle soll auf der einen Seite möglichst schmal und einfach sein und auf der anderen Seite die metamodellspezifische Semantik spiegeln. Daher werden für verschiedene Modellelemente spezifische Create, Read, Update, Delete (CRUD)-Dienste definiert. Dies erhöht zwar die Anzahl der Dienste, aber da die verschiedenen Modellelementtypen über verschiedene Systeme verteilt sind, reicht in den Systemen meist ein Subset der Services aus. Da die Implementierungen der Komponenten eines Entitätenverwaltungssystems

durchaus die hier beschriebenen Fähigkeiten und Modellelemente um eigene ergänzen darf, gibt es zusätzliche Dienste, die den systemspezifischen Zugriff ermöglichen. Diese verwenden teilweise unstrukturierte Datentypen, die erst vom Zielsystem festgelegt werden. Die Nutzung solcher Dienste ist jedoch nicht zwingend notwendig um die Gesamtaufgabe einer Entitätenverwaltung abzubilden. Im folgenden wird die Dienstschnittstelle beschrieben, wobei eine Aufteilung in Sätze hinsichtlich der behandelten Modellelemente erfolgt.

5.5.1 Dienste für Semantikdefinitionen

Semantikdefinitionen werden von Merkmalbeschreibungen, Aussagebeschreibungen, Zusammenhangsmodellen und Zusammenhangsbeschreibungen gehalten. Für diese Modellelemente gibt es jeweils eine Operation zur Erzeugung, zum Auslesen und zur Modifikation von Attributwerten. Das Löschen von Semantikdefinitionen ist nicht per Dienst erlaubt, da es die Semantikbasis in einem verteilten System vernichten kann. Für Zusammenhangsmodelle gibt es zusätzlich eine Operation zur Ankopplung formaler Modelle und für Zusammenhangsbeschreibungen zur Zuweisung von Zusammenhangsmodellen. Diese Operationen sind zu Diensten gruppiert und werden im Folgenden beschrieben. Eine WSDL-Definition ist in Anhang B angegeben. Darin ist auch die Gruppierung der Operationen definiert.

createPropertyDefinition und createStatementDefinition

Die Operation `createPropertyDefinition` erzeugt eine neue Merkmalbeschreibung. Als Parameter nimmt sie einen Namen, die semantische Definition, eine Bemerkung, eine Dokumentreferenz und eine Liste von Identifikatoren bekannter Synonyme an. Dabei können alle Parameter außer Namen und Definition leer bleiben. Die aus den Parametern erzeugte Merkmalbeschreibung hat einen global eindeutigen Identifikator, der als Ergebnis der Operation zusammen mit einem Status übertragen wird. Im Fehlerfall ist der Identifikator leer. Die Operation `createStatementDefinition` ist analog zu `createPropertyDefinition` für die Erzeugung von Aussagebeschreibungen.

createRelationModel

Diese Operation erzeugt ein neues Zusammenhangsmodell. Sie übernimmt die Attribute Name, Beschreibung, Bemerkung, Dokumentreferenz, Quellrollen und Zielrollen als Parameter und liefert einen Status und den Identifikator des neuen Modells zurück. Auch hier dürfen abgesehen von Name und Beschreibung alle Parameter leer sein.

addFormalModelToRelationModel

Diese Operation verknüpft ein Zusammenhangsmodell mit einer formalen Beschreibung. Die formale Beschreibung ist dabei systemspezifisch. Sie befindet sich damit außerhalb der Beschreibungsmöglichkeiten des Metamodells und wird daher als Binary Large OBject (Datenobjekt unbekannter Struktur) (BLOB) übertragen. Als weiterer Parameter erhält diese Operation den Identifikator des Zusammenhangsmodells. Rückgabewerte sind ihr Status, ein Identifikator des angefügten Formalen Modells und der Identifikator des Zusammenhangsmodells. Letzterer kann sich vom vorher angegebenen unterscheiden, wenn der Dienstanbieter entscheidet, dass die Veränderung der Zusammenhangsbeschreibung,

bspw. aus Gründen der semantischen Konsistenzwahrung, eine neue Revision dieser erfordert.

createRelationDescription und createProcessDescription

Diese Operationen erzeugen eine Zusammenhangsbeschreibung bzw. eine Prozessbeschreibung. Die Parameter geben dabei Name, Beschreibung, Bemerkung und eine Dokumentreferenz an. Dabei dürfen Bemerkung und Dokumentreferenz leer sein. Die Operationen geben ihren Status und den Identifikator der angelegten Beschreibung zurück.

attachRelationModelToDescription

Diese Operation verknüpft ein Zusammenhangsmodell mit einer Zusammenhangsbeschreibung. Dabei können beide Seiten mehrfach verknüpft werden. Ein Zusammenhangsmodell kann für verschiedene Zusammenhangsbeschreibungen genutzt werden und umgekehrt. Parameter der Operation sind die Identifikatoren der beiden Elemente. Die Operation gibt ihren Status und den Identifikator der Zusammenhangsbeschreibung zurück. Bei letzterem verhält es sich wie für die Operation `addFormalModelToRelationModel`.

detachModel

Diese Operation entkoppelt ein formales Modell von einem Zusammenhangsmodell oder ein Zusammenhangsmodell von einer Zusammenhangsbeschreibung. Die Modelle werden dabei nicht verändert oder gelöscht. Die Operation gibt ihren Status und den Identifikator des Zusammenhangsmodells bzw. der Zusammenhangsbeschreibung zurück. Für den zurückgegebenen Identifikator gelten die gleichen Ausführungen wie bei `addFormalModelToRelationModel`.

setAttribute

Diese Operation setzt ein Attribut eines Beschreibungs- oder Definitionsobjekts. Sie erhält den Identifikator der Beschreibung, den Namen und den Wert des Attributs als Parameter. Dabei können auch Attribute gesetzt werden, die hier nicht beschrieben und folglich systemspezifisch sind. Diesen Fall erkennt der Dienstanbieter und interpretiert den Namensparameter entsprechend. Der Wertparameter wird dann als BLOB interpretiert. Ein weiterer Parameter ist ein boolesches Flag, dass angibt, ob ein neues Attribut angelegt werden soll, falls das genannte Attribut nicht existiert. Diese Funktionalität ist jedoch nicht obligatorisch für eine Implementierung der Operation. Rückgabewert der Operation ist ihr Status und der Identifikator der Beschreibung. Dieser kann sich, wie für `addFormalModelToRelationModel` angegeben, verändern.

getPropertyDefinition und getStatementDefinition

Diese Operationen dienen dem Auslesen der Definitionen. Sie übernehmen den Identifikator der Definition und ein boolesches Flag als Parameter. Das Flag gibt an, ob auch die systemspezifischen Attribute ausgelesen werden sollen. Die Operation gibt ihren Status und eine XML-Struktur zurück, die die Definition enthält. Die Werte systemspezifischer Attribute werden dabei als BLOB mit systemspezifischer Struktur übertragen.

readRelationModel, readRelationDescription und readProcessDescription

Diese Operationen fragen Zusammenhangsmodelle und Beschreibungen ab. Sie erhalten den Identifikator des abzufragenden Objekts und zwei boolesche Flags als Parameter. Ein Flag gibt an, ob unterlagerte Modelle als ganzes mit übertragen werden oder ausschließlich Referenzen in Form spezieller Merkmale (vergleiche Anhang A) zurückgegeben werden. Das zweite Flag gibt an, ob auch systemspezifische Attribute zurückgeliefert werden sollen. Die Operationen geben ihren Status und das Modell bzw. die Beschreibung als XML-Struktur zurück.

5.5.2 Dienste für Merkmaldatenaustausch

Merkmaldaten umfassen die Menge der zu einer Einheit gehörigen Merkmale und die ihnen zugehörigen Ausprägungsaussagen. Systemspezifische Attribute, wie sie in Abschnitt 5.5.1 erwähnt wurden, werden auch als Merkmale angesehen und mit den dafür spezifizierten Operationen übertragen. Da auch Referenzen zwischen Verwaltungsobjekten durch besondere Merkmale abgebildet werden, können auch sie mit den hier definierten Operationen abgefragt werden. Die Gruppierung der Operationen zu Diensten ist der WSDL-Beschreibung in Anhang B zu entnehmen.

getProperties

Diese Operation liest alle Merkmale einer Beschreibung aus. Sie gibt Aufschluss darüber, welche Merkmale eine Einheit beschreiben. Festgelegte Attribute von Beschreibungen zählen nicht dazu, da sie keinen eigenen Identifikator haben. Parameter sind der Identifikator des Verwaltungsobjekts, dessen Merkmale abgefragt werden sollen, ein Filter für Identifikatoren der Merkmaltypen und ein Filter für Identifikatoren der Merkmalträgertypen. Letzterer dient der Abfrage der unter einer bestimmten Sichtweise interessanten Merkmale. Die Filter können leer bleiben. Rückgabe ist neben dem Status eine Liste von Namen, Identifikatoren, Typen, assoziierten Trägertypen und Maßeinheiten der Merkmale.

getStatements

Diese Operation liest die Ausprägungsaussagen und Werte eines Merkmals aus. Sie erhält den Identifikator des Merkmals als Parameter. Hinzu kommen je ein Filter für Aussagenbeschreibungen und aussagende Instanzen. Leere Angaben für die Filter sind möglich. Die Operation liefert ihren Status und eine Liste mit Name, Identifikator, Beschreibungsreferenz, aussagender Instanz, Wert, Zeitstempel und Bezugszeitpunkt jeder Aussage mit. Die Werte werden dabei als BLOB übertragen, da ihre Struktur anwendungsspezifisch sein kann.

addProperty

Diese Operation fügt einem erweiterbaren Verwaltungsobjekt ein Merkmal hinzu. Dies wird genutzt, um verwaltete Objekte dynamisch zu erweitern. Da dies keine Änderung, sondern eine Erweiterung der semantischen Beschreibung des Verwaltungsobjekts darstellt, ist dies möglich ohne eine neue Revision des Verwaltungsobjekts zu erzeugen. Wird das Verwaltungsobjekt selbst als Entität verfolgt, so kann es sinnvoll sein, eine neue Revision

seiner eigenen Repräsentation anzulegen. Parameter der Operation `addProperty` sind der Name, der Identifikator der Merkmaldefinition und der des Trägertyps des neuen Merkmals sowie die Maßeinheit des Merkmals. Sein Identifikator wird zusammen mit dem Status der Operation zurückgegeben.

addStatement

Diese Operation fügt einem Merkmal eine Ausprägungsaussage hinzu. Sie erhält den Namen, Identifikatoren der Aussagenbeschreibung und des Aussagenden, den Wert, einen Zeitstempel und den Bezugszeitpunkt als Parameter. Die Zeitstempel dürfen leer sein. In diesem Falle wird der Aufrufzeitpunkt der Operation verwendet. Sie gibt den Identifikator der neuen Ausprägungsaussage und ihren Status zurück. Es können mehrere Ausprägungsaussagen gleichen Typs zu einem Merkmal getroffen werden. Diese sind anhand ihrer Identifikatoren unterscheidbar. Außerdem ist ihr Bezugszeitpunkt nur in Ausnahmefällen gleich. Damit kann eine Entwicklung des Merkmalwertes nachvollzogen werden. Beim Hinzufügen einer Aussage ist es für zeitbezogene Einheitenabbilder meist sinnvoll, eine neue Revision anzulegen. Die Entscheidung darüber liegt beim Verwaltungssystem.

5.5.3 Dienste für den Umgang mit zeitbezogenen Einheitenabbildern

Die meisten Einheitsinformationen werden durch Merkmale und dazugehörige Aussagen gehalten. Daher sind sie mit den in Abschnitt 5.5.2 aufgeführten Diensten bereits abruf- und veränderbar. Die Erzeugung, Löschung und Verknüpfung von Einheitenabbildern mit Zusammenhängen und Lebenszyklusprozessen wird mit den folgend beschriebenen Operationen umgesetzt. Da nicht alle Operationen für alle Anwendungsfälle notwendig sind, sind sie auf die drei Dienste `entityHandlingServiceData`, `entityHandlingServiceModels` und `entityHandlingServiceAdditional` aufgeteilt, damit nur die für einen Anwendungsfall notwendige Funktionalität implementiert werden muss. Die Operationen des Diensts `entityHandlingServiceData` werden als erstes beschrieben.

createEntityType

Diese Operation erzeugt einen Prototypen für ein Einheitenabbild. Sie erhält Namen, Beschreibung, Bemerkung, Dokumentreferenz und eine Liste von Elterntypen als Parameter. Bemerkung, Dokumentreferenz und Elterntypen dürfen leer übergeben werden. Bei der Übergabe von Elterntypen werden ihre Merkmale in die neue Typbeschreibung aufgenommen. Die Operation gibt ihren Status und den Identifikator des neuen Typen zurück.

createEntity

Diese Operation legt ein zeitbezogenes Einheitenabbild aus gegebenen Prototypen an. Dabei werden alle in diesen festgelegten Merkmale für die neue Abbildung übernommen. Kommt es dabei zu Doppelungen eines Merkmaltyps, so werden mehrere Merkmalsrepräsentationen angelegt, sofern sich die Trägertypen unterscheiden. Die Merkmale haben zunächst keine Ausprägungsaussagen. Parameter der Operation sind der Name der Einheit, Identifikatoren der Elterntypen, der Einheitsbezug, die Angabe, ob es sich um eine physische Einheit handelt, und ein Zeitpunkt der Entstehung der Entität. Der Zeitpunkt

darf leer übergeben werden. In diesem Fall wird der Aufrufzeitpunkt der Operation gesetzt. Die Operation gibt ihren Status und den Identifikator der neuen Einheitenabbildung zurück. Die Festlegung, ob es sich um eine physische Einheit handelt, erfolgt auf Instanz- und nicht vorab auf Typebene, weil es sich hier auch um eine Simulationseinheit o. ä. handeln könnte. Da es sich um die Erzeugung einer neuen Repräsentation handelt, ist der Identifikator gleichzeitig der Erst-Identifikator der Entität.

createNewEntitySnapshot

Diese Operation erzeugt eine neues zeitbezogenes Einheitenabbild aus einem vorigen Einheitenabbild. Ihre Parameter sind der Identifikator oder der Erst-Identifikator der bisherigen Einheitenabbildung, ein boolesches Flag, das angibt, ob die Aussagen der letzten Abbildung übernommen werden sollen, ein boolesches Flag, das angibt ob Zusammenhangsmodelle beibehalten werden sollen, und ein Zeitstempel der Entstehung der neuen Abbildung. Letztere hat keinen angehängten Lebenszyklusprozess. Je nach Wert der Flags werden Ausprägungsaussagen und angehängte Zusammenhangsmodelle übernommen. In jedem Fall werden alle Merkmale kopiert. Die Operation hat keine Auswirkung auf unterlagerte (mit *metaModel:hasPart* angebundene) Einheiten. Sie gibt ihren Status, den Identifikator des neuen zeitbezogenen Einheitenabbilds, ihre Revisionsnummer und eine Liste der unterlagerten Einheiten zurück. Letztere kann genutzt werden, um ihnen nötigenfalls auch eine neue Revision zu geben.

buryEntity

Diese Operation markiert das Vergehen einer Einheit in ihrer Abbildung. Die Entität wird damit als vergangen aufgefasst, ihre Lebenszyklusinformationen bleiben jedoch erhalten. Eine vollständige Löschung gibt es nicht. Lediglich die Verfolgung der Einheit endet. Die Operation erhält den Identifikator oder den Erst-Identifikator als Parameter. Ihre Rückgabewerte sind ihr Status und eine Liste unterlagerter Einheiten. Auf diese wirkt sich die Operation nicht aus. Das bedeutet für alle unterlagerten Einheiten muss sie separat aufgerufen werden. Dies ist nötig, da das Vergehen einer umgebenden Einheit nicht das Vergehen der unterlagerten Einheiten bedeuten muss.

getEntities

Diese Operation dient der Abfrage der verwalteten Einheiten. Sie nimmt einen Filter für Identifikatoren der Einheitstypen, einen Filter für Einheitsbezüge, ein boolesches Flag, das angibt, ob auch bereits vergangene Einheiten angegeben werden sollen, und ein weiteres boolesches Flag, das angibt, ob auch Informationen früherer Einheitenabbildungen und Lebenszyklusprozesse angegeben werden sollen, als Parameter an. Die Operation liefert ihren Status und eine Liste von Beschreibungen der gefundenen Einheiten zurück. Die Beschreibungen bestehen aus allen für Einheitenabbildungen festgelegten Attributen. Es wird immer die aktuellste Abbildung geliefert.

getAttached

Diese Operation dient der Abfrage des mit einer Einheit verbundenen Lebenszyklusprozesses, der Zusammenhangsmodelle und Zusammenhangsbeschreibungen. Parameter ist der

Identifikator der Einheitenabbildung oder ihr Erst-Identifikator und ihre laufende Nummer. Die Operation gibt neben ihrem Status eine Liste der verknüpften Beschreibungen und Modelle zurück.

getCurrentSnapshot

Diese Operation dient der Abfrage der derzeitigen (aktuellsten) zeitbezogenen Abbildung einer Einheit. Dazu erhält sie den Identifikator einer beliebigen Abbildung oder den Erst-Identifikator als Parameter. Die Operation liefert neben ihrem Status den Identifikator, den Erst-Identifikator und die laufende Nummer der derzeitigen Abbildung zurück.

getEntityHistory

Diese Operation fragt die Vergangenheit einer Einheit ab. Sie erhält den Identifikator oder den Erst-Identifikator mit laufender Nummer der Einheit als Parameter. Das Ergebnis ist neben dem Status eine Liste der Identifikatoren, laufenden Nummern und Zeiträumen aller vergangenen Abbildungen. Wird als Parameter eine spezielle Abbildung (mit direktem Identifikator oder laufender Nummer) abgefragt, so werden alle vorigen Abbildungen zurück geliefert. Wird nur mit dem Erst-Identifikator ohne laufende Nummer abgefragt, so werden alle Abbildungen bis zur neuesten zurückgegeben.

getSnapshotAtTime

Diese Operation dient der Abfrage der Abbildung einer Einheit zu einem bestimmten Zeitpunkt. Parameter sind der Identifikator oder der Erst-Identifikator und der besagte Zeitpunkt. Eine laufende Nummer der Abbildung kann übergeben werden, wird aber ignoriert. Rückgabewerte sind neben dem Status der Identifikator, der Erst-Identifikator und die laufende Nummer der angefragten Abbildung. Falls es zum abgefragten Zeitpunkt keine Abbildung der Einheit gab, wird nur der Status zurückgegeben.

setFollowUpSnapshot

Diese Operation dient der verteilten Verwaltung einer Entität. Wechselt die Verwaltung einer Einheit in ein anderes System, so wird dem vorigen Verwaltungssystem mit dieser Operation mitgeteilt, wo die Abbildung des weiteren Lebenszyklus der Einheit zu finden ist. Parameter sind der Identifikator oder der Erst-Identifikator der Einheitenabbildung, der Identifikator der neuen Abbildung im entfernten Verwaltungssystem und der Zeitpunkt der Übergabe. Rückgabewert der Operation ist ihr Status.

attachLifeCycleProcessToEntity

Diese Operation bildet zusammen mit den beiden nachfolgenden den Dienst `entityHandlingServiceModels`. Sie verknüpft eine Einheitenabbildung mit einer Prozessbeschreibung eines Lebenszyklusprozesses. Parameter sind die Identifikatoren der Einheit und der Prozessbeschreibung sowie der Verknüpfungszeitpunkt. Letzterer darf leer sein, wobei der Aufrufszeitpunkt der Operation gesetzt wird. Zu jedem Zeitpunkt kann eine Einheit nur mit genau einem Lebenszyklusprozess verknüpft sein. Folglich schlägt die Operation fehl, wenn bereits eine Verknüpfung besteht. Diese muss vorher

mit `detachFromEntity` gelöst werden. Die Operation passt die Attribute des zeitbezogenen Einheitenabbilds entsprechend der Verknüpfung an. Wenn die Lebenszyklusprozessbeschreibung aktive Zusammenhangsmodelle enthält, beginnen diese zu arbeiten. Rückgabewert der Operation ist ihr Status.

attachRelationToEntity

Diese Operation verknüpft eine Einheitenabbildung mit einem Zusammenhangsmodell oder einer Zusammenhangsbeschreibung. Parameter sind die Identifikatoren der Abbildung und des Zusammenhangsmodells bzw. der Zusammenhangsbeschreibung. Aktive Modelle beginnen sofort zu arbeiten. Es können beliebig viele Zusammenhangsmodelle bzw. Zusammenhangsbeschreibungen an eine Einheitenabbildung gekoppelt werden, da sie, sofern sie aktive Modelle tragen, auf ihre Ziele durch Hinzufügen von Aussagen wirken und durchaus mehrere Aussagen zu demselben Merkmal erlaubt sind. Rückgabewert der Operation ist ihr Status.

detachFromEntity

Diese Operation entkoppelt ein Zusammenhangsmodell, eine Zusammenhangsbeschreibung oder einen Lebenszyklusprozess von einer Einheitenabbidung. Parameter sind die Identifikatoren der Abbildung und des Modells bzw. der Beschreibung. Aktive Modellelemente stellen sofort die Arbeit ein. Zusammenhangsmodelle werden lediglich entkoppelt. Für Lebenszyklusprozessbeschreibungen wird der Zeitpunkt des Abkoppelns in den Attributen der Einheitenabbildung vermerkt. Rückgabewert der Operation ist ihr Status.

getValueAtTime und getValueHist

Diese beiden Operationen bilden den Dienst `entityHandlingServiceAdditional`. Ihre Funktionalität lässt sich auch durch kombinierte Nutzung anderer Operationen des `entityHandlingServiceData` und des Dienstes zum Merkmaldatenaustausch umsetzen. Sie erkennen eine spezifische Aussage zu einem Merkmal einer Einheit und verfolgen diese zurück zu einem bestimmten Zeitpunkt oder über einen bestimmten Zeitraum. Dazu wird das zu dem Merkmal gehörige Einheitenabbild zum spezifizierten Zeitpunkt oder Zeitraum verfolgt und daraufhin das Merkmal und die passende Aussage abgefragt, wobei das Merkmal über Typ und Klarnamen und die Aussage über Typ und aussagende Instanz erkannt wird. Die gefundenen Aussagen werden zusammen mit einem Status zurückgegeben.

5.5.4 Dienst für den Umgang mit Ereignisrepräsentationen

Dieser Dienst dient dem Umgang mit Ereignissen. Sie werden außerhalb der Entitätenverwaltung hantiert, können jedoch Entitäten referenzieren, um ihren Zusammenhang abzubilden. Umgekehrt können Entitäten über spezielle Merkmale (vergleiche Anhang A) auf Ereignisse verweisen. Ereignisse werden nicht in Zusammenhänge einbezogen, da sie systemspezifisch beschrieben werden.

createEvent

Diese Operation legt ein neues Ereignis an. Parameter sind der menschenlesbare Name, ein Zeitstempel, eine Liste betroffener Einheiten, der Betreff, die Beschreibung und systemspezifische Daten. Letztere und der Zeitstempel sind optional. Wird kein Zeitstempel angegeben, wird der Aufrufzeitpunkt der Operation verwendet. Rückgabewert ist neben dem Status der Identifikator des angelegten Ereignisses.

readEvent

Diese Operation dient dem Auslesen einer Ereignisbeschreibung. Sie erhält den Identifikator des Ereignisses als Parameter und liefert neben ihrem Status alle das Ereignis beschreibenden Daten zurück (alles was mit createEvent gespeichert wurde).

addEntityToEvent

Diese Operation fügt einer Ereignisbeschreibung eine betroffene Einheit hinzu. Dazu erhält sie die Identifikatoren der Ereignisbeschreibung und des zeitbezogenen Einheitenabbilds als Parameter. Rückgabewert ist der Status der Operation.

getEvents

Diese Operation dient der Abfrage von Ereignisbeschreibungen. Dazu erhält sie Filter für den Identifikator, betroffene Einheiten und die Beschreibung sowie ein Zeitfenster als Parameter. Das Zeitfenster kann auch ausgeschlossen werden. Die Antwort enthält neben dem Status eine Liste von Ereignisbeschreibungen mit allen dazu gehörigen Daten.

5.6 Komponenten des Gesamtsystems und ihre Aufgaben

Aufbauend auf den beschriebenen Datenmodellen und Schnittstellen können Komponenten entwickelt werden, die in einem Gesamtsystem zusammenspielen. Dabei werden die Informationen unternehmensübergreifend ausgetauscht und daraus ein Mehrwert erzeugt. Kern des Informationsaustauschs ist die Nutzung eindeutiger Identifikatoren als Referenz zu Repräsentationen unter Nutzung der in Abschnitt 5.5 beschriebenen Dienste. Im Folgenden werden die Komponententypen beschrieben und anschließend das aus ihnen entstehende Gesamtsystem vorgestellt.

5.6.1 Definitionsdatenbanken

Definitionsdatenbanken dienen der Schaffung einer gemeinsamen Semantikbasis. Sie gliedern sich in Datenbanken für

- Merkmaldefinitionen,
- Aussagedefinitionen,
- Zusammenhangsmodelle,
- Zusammenhangsbeschreibungen und

- Lebenszyklusprozessbeschreibungen.

Dabei kann ein System auch mehrere der genannten Aufgaben erfüllen.

Merkmaldefinitionsdatenbanken legen fest, wie Daten für bestimmte Merkmale zu interpretieren sind. Um unternehmensübergreifend nutzbar zu sein, müssen sie frei zugänglich sein. Außerdem muss jedes allgemeine Merkmal global eindeutig identifizierbar sein. Dies wird durch die Nutzung der in Abschnitt 5.2 vorgestellten Identifikatoren erreicht. Dabei ist die Namensraum-URL der Merkmaldatenbank zugeordnet. Neben der eindeutigen Identifikation sollte sie als Zugriffsadresse auf die Merkmaldatenbank selbst dienen können. Die Merkmaldatenbanken stellen eine einheitliche Dienstschnittstelle bereit, die der Abfrage von Definitionen sowie dem Hinzufügen neuer Definitionen dient. Die Schnittstelle setzt die in Abschnitt 5.5.1 dargestellten Dienste um, soweit sie die in der Datenbank verwalteten Definitionen betreffen.

Um Einheiten beschreiben zu können, müssen ihre definierten Merkmale mit Inhalt gefüllt werden. Dies wird durch Aussagen der Form

$$Aussage \text{ verknüpft } Merkmal \text{ mit } Wert \tag{5.1}$$

umgesetzt. Aussagen werden ebenso wie Merkmale in Datenbanken definiert. Dabei werden die gleichen Attribute verwendet wie für Merkmale. Der zugehörige Dienst verhält sich analog. Durch die semantisch eindeutige Definition der Aussage in Verbindung mit den Identifikatoren können sie später maschinell ausgewertet werden.

Neben Merkmalen und Aussagen werden auch Zusammenhänge und Prozesse definiert. Auch diese haben eindeutige Identifikatoren und menschenlesbare Beschreibungen. Zusätzlich können sie mit formalen Modellen beschrieben werden. Auf letztere wird in den Definitionen nur verwiesen. Die Modelle können zwar in derselben Datenbank verwaltet werden, aber da ihre Semantik systemspezifisch beschrieben wird, werden sie nicht direkt in die Repräsentationen der Beschreibungen aufgenommen. Die Abkehr von einheitlichen Beschreibungen von Zusammenhangsmodellen hin zu systemspezifischen ist darin begründet, dass diese Modelle in Verwaltungsservern ausführbar sein sollen, um direkt Daten verarbeiten zu können. Dabei können verschiedenste Systeme und Plattformen als Basis dienen. Eine in dieser Hinsicht agnostische Beschreibungsmöglichkeit wäre bspw. durch die Definition einer virtuellen Umgebung möglich. Dabei kann jedoch nicht die Anwendbarkeit auf alle möglichen Probleme mit vertretbarem Aufwand garantiert werden. Außerdem würde die Definition einer solchen Umgebung den Rahmen dieser Arbeit bei weitem sprengen. Daher werden systemspezifische Modellbeschreibungen erlaubt, sofern sie die in der Beschreibung des Zusammenhangsmodells angegebenen Rollenbeschreibungen für Quellen und Ziele auflösen können. Im Rahmen dieser Arbeit werden in Kapitel 6 zwei Möglichkeiten der Definition ausführbarer Modelle gegeben.

5.6.2 Entitätenverwaltungsserver

Die Entitätenverwaltungsserver sind die Kernelemente des Systems. Sie haben drei Kernaufgaben:

1. Sie tragen durch Definition von Prototypen für Einheitenabbilder zum Aufbau einer unternehmensübergreifenden Semantik bei;

2. Sie halten die Informationen zu allen von ihnen verwalteten Entitäten;

3. Sie führen formale Zusammenhangsmodelle aus, setzen dabei verschiedene Informationen in Verbindung und wirken dadurch proaktiv bei der Generierung neuer Informationen zu den Entitäten mit.

Ihre erste Aufgabe erfüllen sie durch die Festlegung, durch welche Merkmale eine Entität beschrieben wird. Dabei sind diese Merkmalfestlegungen nicht zwingend vollständig. Sie dienen nur der Herstellung eines gemeinsamen Basisverständnisses. Je nach betrachteten Aspekten können Prototypen und Einheitenabbilder später erweitert werden. Dadurch ist sichergestellt, dass auch unternehmensübergreifend ein gemeinsames Verständnis bzw. Modell einer jeden Entität vorhanden ist und gleichzeitig unternehmensintern (oder auch nur gewerkeintern) weitere spezifische Informationen verarbeitet werden können. Die zweite Aufgabe wird durch Speicherung aller Aussagen, die zu den Merkmalen der Entitäten getroffen werden, umgesetzt. Diese Aussagen können, ebenso wie die zu einer Einheit verwalteten Merkmale, jederzeit abgefragt werden. Dadurch kann auf einfache Art und Weise eine Beschreibung des Zustands einer Entität abgerufen werden. Auch strukturelle Zusammenhänge zwischen Entitäten (besonders Kapselung) können durch derartige Merkmale abgebildet werden.

Die Abbildung von Zusammenhängen zwischen verschiedenen Eigenschaften einer oder mehrerer Entitäten wird durch Zusammenhangsbeschreibungen umgesetzt. Diese dienen der menschenlesbaren Beschreibung komplexer und möglicherweise multidirektionaler Zusammenhänge. Um die maschinelle Auswertung zu vereinfachen und dabei die Wiederverwendbarkeit zu verbessern, werden den Zusammenhangsbeschreibungen Zusammenhangsmodelle unterlagert, die jeweils einen gerichteten (unidirektionalen) Zusammenhang beschreiben. Diese werden mit maschinenauswertbaren Modellen unterfüttert, deren Datenquellen und Ziele als Rollen beschrieben werden. Somit können die formalen Modelle im Entitätenverwaltungsserver ausgeführt werden und generieren dabei neue Informationen über die verwalteten Entitäten. Durch die Nutzung der Rollen und einfachen, unidirektionalen Modelle, können diese in einer Vielzahl von Zusammenhangsbeschreibungen wiederverwendet werden und sind wesentlich einfacher zu entwickeln.

Der Begriff der Rolle beschreibt in diesem Zusammenhang eine Art Pfad zu den zu verwendenden Daten. Dieser ist bei der Entwicklung des Zusammenhangsmodells jedoch noch nicht exakt bekannt. Daher wird dieser Pfad nicht über die Identifikatoren von Einheitenabbildern, Merkmals- und Aussagenrepräsentationen realisiert, sondern über deren Typen, wobei abgesehen von der letzten Aussage nur solche des Typs *metaModel:definedIdentificator* verwendet werden dürfen. Auf diese Art werden Wege zu Informationen beschrieben, die in unterlagerten oder übergeordneten Entitäten zu finden sind, und semantisch eine bestimmte Bedeutung (Merkmalsdefinition und Aussagendefinition) haben. Kann ein Quellpfad zu mehreren Objekten aufgelöst werden, so werden alle Werte abgerufen und müssen in Listenform verarbeitet werden. Bei mehreren Zielen werden alle gesetzt. Die Modelle müssen entsprechend angelegt werden, um damit umzugehen.

Entitätenverwaltungsserver bieten die in Abschnitt 5.5.3 beschriebene Schnittstelle an. Um den Nutzerzugriff zu erleichtern, spannen sie dabei, soweit, wie es durch die Zugriffs- und Kommunikationsberechtigungen im Netz möglich ist, ein Peer-Netzwerk auf, in dem Anfragen zu spezifischen Entitäten oder deren Eigenschaften automatisch an den richtigen Adressaten weitergeleitet werden. Dadurch führt eine Anfrage an einen beliebigen Verwaltungsserver zur Rückgabe der angefragten Daten. Die Verwaltungsserver stellen den zentralen Punkt der Nutzerinteraktion und der Datenhaltung dar.

5.6.3 Historiendatenbanken

Historiendatenbanken halten Daten, die sich häufig ändern. Sie dienen als Informationsquelle (und Senke) für die Vorgangsmodelle. Der entitätsbezogene Zugriffsweg auf die Daten in einer Historiendatenbank wird über die speziellen Aussagen *metaModel:dataURL* und *metaModel:dataInterfaceType* abgebildet. Historiendatenbanken sind für das Gesamtsystem nicht unbedingt nötig, da alle Daten prinzipiell auch in Entitätenverwaltungsservern gehalten werden können. Trotzdem gibt es Gründe, sie einzubinden. Erstens können hoch spezialisierte Systeme mit entsprechenden Protokollen und Übertragungsraten für große Datenmengen verwendet werden. Zweitens ermöglicht die referenzbasierte Schnittstelle über externe URLs die Anbindung der vorhandenen Datenerfassungssysteme als Historiendatenbank.

5.6.4 Ereignisdatenbanken

Ereignisdatenbanken speichern Informationen zu Ereignissen ab, die mit Entitäten in Verbindung stehen. Die Ereignisse werden von zeitbezogenen Einheitenabbildern referenziert und umgekehrt. Da Ereignisse mit systemspezifischen Daten gespeichert werden und ihre Semantik in den Definitionsdatenbanken nicht notwendigerweise abgebildet ist, werden sie nicht für die Ausführung formaler Modelle für Zusammenhänge und Prozesse herangezogen. Sie dienen der Auswertung und Verfolgung der Vorgänge im Betrieb durch den Menschen oder durch ein auf die systemspezifische Semantik zugeschnittenes weiteres System.

5.7 Struktur und Interaktion im Gesamtsystem

Die vorgestellten Komponenten bilden unternehmensübergreifende Gesamtsysteme. Diese Systeme sind durch die Partnerschaften und Handelsbeziehungen zwischen den Unternehmen definiert, nicht vorab festlegbar und ständigem Wandel unterworfen. Für die umfassende Verwaltung der Entitäten bilden sich lose Netze aus den verschiedenen Servern, die zu verschiedenen Unternehmen gehören. Im Folgenden wird zunächst dargestellt, wie die einzelnen Systeme verortet sein können und anschließend wird anhand dreier beispielhafter Zugriffsszenarien die Interaktion gezeigt.

Die meisten Entitäten wechseln während ihrer Lebensphase zwischen verschiedenen Verantwortungsbereichen. Der häufigste Wechsel ist dabei der vom Erzeuger zum Nutzer. Um eine Entität nun über diese gesamte Zeit verfolgen zu können, müssen die Systeme der verschiedenen Verantwortungsbereiche interagieren können. Dazu müssen sie in der Lage sein, erstens miteinander zu kommunizieren und zweitens die Übereinstimmung der Semantik der verschiedenen Systeme sicherzustellen.

Die gemeinsame Semantik wir durch Definitionsdatenbanken hergestellt bzw. abgeglichen. Entscheidet sich ein Unternehmen dafür, eine solche Datenbank selbst zu betreiben, so muss es allen Partnern die Abfrage der für sie relevanten Informationen ermöglichen. Um eine Explosion von Synonymen zu verhindern ist es sinnvoll, möglichst viele Merkmale unabhängig von den Unternehmen zu definieren und zu verwalten. Dies kann durch die Nutzung genormter Merkmallisten und deren zu Verfügung Stellung durch öffentliche Datenbanken, wie bspw. die Elektropedia [2], erreicht werden. Definitionsdatenbanken sind

[2]Online-Version des IEV: http://www.electropedia.org/

demnach sowohl unabhängig von Herstellern und Anwendern als auch in deren Hand sinnvoll. Wichtig ist lediglich, dass der Zugang zu den gehaltenen Definitionen ermöglicht wird.

Die eigentliche Verfolgung der Entitäten findet auf Verwaltungsservern der mit der Entität in Zusammenhang stehenden Organisationen statt. Dabei kann eine Entität von mehreren Parteien bspw. Hersteller, Anwender und Wartungsdienstleister unabhängig verfolgt werden. Haben die Parteien Kenntnis von weiteren Verwaltern einer Entität, so können sie die fremden Abbilder der Einheit durch Merkmale des Typs *metaModel:otherRepresentation* referenzieren. Dies ermöglicht, sofern der Zugriff von den beteiligten erlaubt wird, den unternehmensübergreifenden Zugriff auf die erfassten Daten. Um die Nutzung der verfügbaren Informationen zu erleichtern, ist es dabei sinnvoll, wenn alle Verwalter einer Einheit diese zumindest hinsichtlich der für alle interessanten Eigenschaften gleich modellieren. Eine Möglichkeit ist, dass der Hersteller einer Einheit die von ihm für sie verwendeten Merkmale in Form eines Einheitsprototypen anderen Parteien zur Verfügung stellt. Letztere können die Abbilder in ihren Verwaltungssystemen darauf aufbauen (Vererbung) und um weitere für sie relevante Merkmale ergänzen. Solche Typdatenbanken als Teil von Entitätenverwaltungsservern sind demnach sowohl für die konkret beteiligten Parteien, als auch als öffentliche Systeme sinnvoll.

Interessieren sich verschiedene Parteien für dieselben Eigenschaften einer Einheit und verwalten diese Einheit unabhängig voneinander, so kann mit Zusammenhangsmodellen ein Abgleich der aufgenommenen Daten stattfinden. Beispielhaft sei ein Hersteller genannt, der die geleisteten Arbeitsstunden einer ausgelieferten Maschine überwachen möchte. Er hat für die Eigenschaft geleistete Arbeitsstunden ein Merkmal im Entitätsprototyp angelegt. Dieser wird sowohl in der Entitätenverwaltung des Anwenders als auch in der des Herstellers überwacht. Nun ist es dem Hersteller möglich, sofern der Anwender ihm Zugriff erlaubt, mit einem formalen Modell einer Zusammenhangsbeschreibung die Ausprägung des Merkmals der entfernten Repräsentation (beim Anwender) auszulesen und darüber die Repräsentation in seiner Verwaltung zu aktualisieren.

Ein weiterer Anwendungsfall für Zusammenhangsmodelle ist die Daten(vor)verarbeitung für Partnerorganisationen. Beispielhaft sei eine regelbare Pumpe genannt, deren Lastverhalten und Einsatzbedingungen vom Hersteller zum Zwecke der präventiven Wartung analysiert werden sollen. Die einfachste Möglichkeit ist der direkte Zugriff des Herstellers auf die Prozessdaten des Anwenders. Dies wird jedoch kaum ein Anwender dulden, da er dem Hersteller interne Daten offenbaren müsste. Hier kann es sinnvoll sein, dass der Kunde ein vom Hersteller geliefertes oder selbst erstelltes Zusammenhangsmodell, das die Daten aggregiert und einen Wartungsindikator an den Hersteller liefert, innerhalb seines Systems anwendet. Damit sind einerseits die Daten des Anwenders geschützt und der Hersteller erhält andererseits die für ihn wichtigen Informationen. Selbst wenn das Vorgangsmodell dabei vom Hersteller geliefert wurde, sieht er selbst nicht die konkreten Datenquellen. Lediglich die rollenbasierten Pfade sind ihm bekannt. Wenn diese dabei nur Entitätsprototypen verwenden, die beiden Parteien ohnehin bekannt sind (Die letzten Merkmal- und Aussagetypen müssen dem Hersteller ohnehin bekannt sein, da er sein Modell sonst nicht erstellen könnte.), erhält der Hersteller als Lieferant des Vorgangsmodells keine ihm vorher unbekannten Informationen.

Da die formalen Modelle, die einen Zusammenhang beschreiben, systemspezifisch umgesetzt werden, ist für das beschriebene Szenario wichtig, dass das System des Anwenders das Modell des Herstellers unterstützt. Der Hersteller kann dem Anwender jedoch auch ein System zur Verfügung stellen, bspw. in Form einer Virtuellen Maschine, das sein Modell

ausführt. Dabei ist zwischen den Beteiligten abzuklären und auf technologischer Ebene sicherzustellen, dass beide Parteien nur die jeweils freigegebenen Daten der jeweils anderen Partei erhalten. Inwieweit die formalen Modelle sich zur Laufzeit erkunden oder auslesen lassen ist nicht festgelegt. Je nach Wunsch der Beteiligten können die Interna offen oder versteckt sein. Dies ermöglicht auch die Verringerung des zwischen den Parteien übertragenen Datenvolumens, da Daten direkt an der Quelle aggregiert und vorverarbeitet werden können, bevor sie übertragen werden. Auch die verteilte Auswertung der Daten ist möglich.

Die Zugriffsrechte auf Merkmale und Aussagen sind im Datenmetamodell selbst nicht abgebildet. Es liegt jedoch nahe, sie systemintern instanzbezogen zu verwalten, da die Berechtigungen sich instanzbezogen unterscheiden werden. Diese Berechtigungsinformationen können vom System bei der Ausführung der Zugriffsdienste ausgewertet werden. Die Authentizität der an einem Zugriff beteiligten Parteien ist beim Zugriff sicherzustellen. Dies ist Aufgabe des zu Grunde liegenden Kommunikationssystems. Für das Standard-Binding der Zugriffsdienste bspw. kann dies durch Nutzung von HTTP Secure (Übertragung von HTTP unter Nutzung des Transport Layer Security Protokolls) (HTTPS) mit beidseitigen Zertifikaten sichergestellt werden.

Die drei nachfolgenden Zugriffsszenarien dienen als Interaktionsbeispiele bei der Verwaltung einer Entität. Aus Gründen der Einfachheit wurden nur Hersteller und Anwender der Einheit abgebildet. Selbstverständlich kann es weitere Parteien geben (bspw. Wartungspartner), die dieselbe Einheit ebenfalls verfolgen. In den Abbildungen werden die von unabhängigen Organisationen betriebenen Definitionsdatenbanken und Entitätenverwaltungsserver in einer Wolke dargestellt. Die Entitätenverwaltungsserver sind dabei nur aufgrund ihrer Typdatenbanken interessant, da diese für die Definition von Einheitszustandsrepräsentationen für andere Organisationen herangezogen werden können. Der gestrichelt umrandete Bereich ist der Verantwortungsbereich des Herstellers der Entität. Der durchgezogen umrandete Bereich ist der Verantwortungsbereich des Anwenders. Der gestrichelt umrandete Verwaltungsserver im Verantwortungsbereich des Anwenders wird vom Hersteller betrieben, um für ihn interessante Daten vorzuverarbeiten. Des Weiteren sind Definitionsdatenbanken, Historiendatenbanken und Ereignisdatenbanken sowie Zugriffsklienten von Hersteller und Anwender dargestellt. Die Entität selbst liegt im Verantwortungsbereich des Anwenders.

Das erste Szenario (Abbildung 5.7) ist der Informationszugriff von einem Klienten des Anwenders, bspw. um den aktuellen Verschleißzustand einer Entität in Erfahrung zu bringen. Dazu fragt er Daten aus dem Entitätenverwaltungsserver des Anwenders ab. Um die Semantik der Daten in Erfahrung zu bringen, stehen ihm Definitionsdatenbanken des Anwenders, des Herstellers und unabhängiger Organisationen zur Verfügung. Weiterhin greift die Klient des Anwenders anhand von in Merkmalen referenzierten Quellen auf eine Historiendatenbank zu und überprüft die mit der Entität in Zusammenhang stehenden Ereignisse in der Ereignisdatenbank. Zusätzlich kann er durch Abfrage der früheren Abbilder der Einheit ihre Vergangenheit vor ihrem Eintritt in seinen Verantwortungsbereich in Erfahrung bringen. Dazu fragt er den Verwaltungsserver des Herstellers ab.

Das zweite Szenario (Abbildung 5.8) beleuchtet den Zugriff des Klienten des Herstellers. Dieser greift auf die Informationen in seinem Entitätenverwaltungsserver zu. Historiendatenbanken können auch beim Hersteller vorhanden sein und abgefragt werden, sind aber im Beispiel nicht dargestellt. Ereignisse, wie bspw. vergangene Wartungsaufträge des Kunden werden aus Ereignisdatenbanken abgefragt. Zur Ermittlung der Semantik der Daten

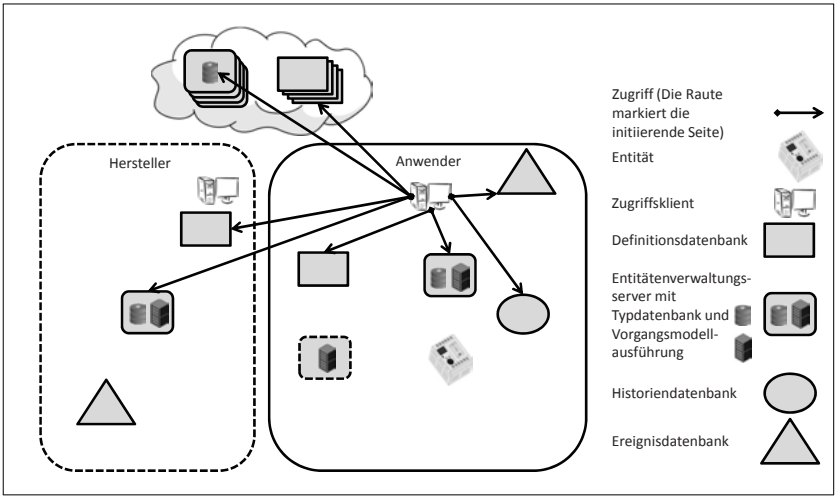


Abbildung 5.7: Interaktionen im Gesamtsystem ausgehend von einem Zugriffsklienten des Nutzers einer Entität.

zieht der Hersteller wieder Semantikdatenbanken und Typdatenbanken heran. Er nutzt jedoch nicht die des Anwenders, da diese nur zusätzliche von letzterem betrachtete Eigenschaften beschreibt. Um aktuelle Informationen zur Entität zu erhalten, leitet der Entitätenverwaltungsserver des Herstellers einige Anfragen an den des Anwenders weiter. Zudem gibt es Zugriffe über die zum Anwender ausgelagerte Ausführung der Vorgangsmodelle. Diese können, wie auch der Entitätenverwaltungsserver des Anwenders, auf dessen Historiendatenbanken zugreifen (soweit, wie der Anwender aufgrund von Übereinkünften mit dem Hersteller den Zugriff gestattet).

Das dritte Szenario (Abbildung 5.9) zeigt Interaktionen auf Basis der Entitätenverwaltungsserver von Anwender und Hersteller. Der Entitätenverwaltungsserver des Anwenders legt auf Basis von Typdaten aus einer öffentlichen Datenbank (wie bspw. eCl@ss), aus der Datenbank des Herstellers und eigenen Typdaten eine Abbildung der Einheit an. Dabei überprüft er die Semantik der definierten Merkmale mit den ihm zur Verfügung stehenden Semantikdatenbanken und verhindert nach Möglichkeit die Nutzung synonymer Merkmale oder verknüpft diese durch Zuweisungsrelationen. Anschließend wird die Entität vom Verwaltungsserver verfolgt. Dabei erhalten die dort ablaufenden Zusammenhangsmodelle Informationen von der Entität selbst und aus Historiendatenbanken. Letztere können von der Entität selbst mit Daten versorgt werden. Außerdem verknüpft der Verwaltungsserver Ereignisse, die mit der Entität in Zusammenhang stehen, mit ihrer Abbildung. Parallel dazu wird die Einheit im Verwaltungsserver des Herstellers verfolgt. Dieser erhält Informationen von Zusammenhangsmodellen, die der Hersteller in Absprache mit dem Anwender in dessen Verantwortungsbereich betreibt. Diese Zusammenhangsmodelle erhalten ihre Daten vom Entitätenverwaltungsserver und der Historiendatenbank des Anwenders innerhalb der festgelegten Zugriffsrechte. Der Entitätenverwaltungsserver des

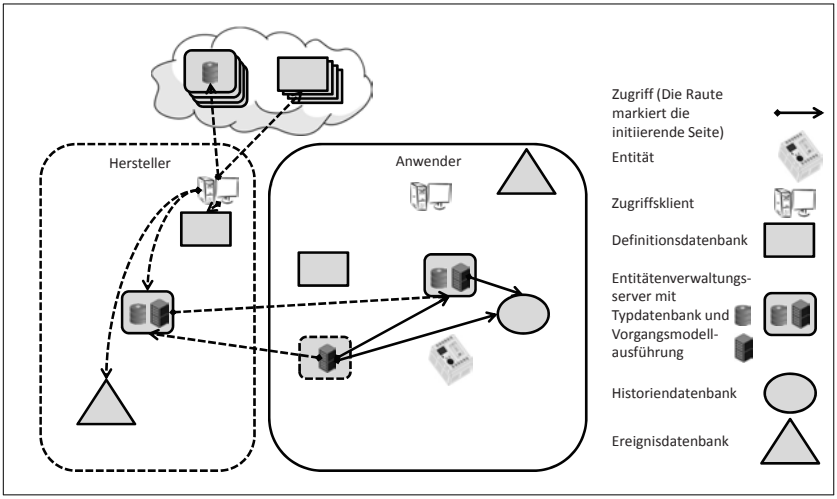


Abbildung 5.8: Interaktionen im Gesamtsystem ausgehend von einem Zugriffsklienten des Herstellers einer Entität während diese beim Nutzer ist und verwaltet wird.

Herstellers generiert Ereignismeldungen und verknüpft sie mit seiner Abbildung der Einheit. Die Abbildungen der Einheit bei Hersteller und Anwender können sich gegenseitig mit dem Merkmal *metaModel:otherRepresentation* referenzieren.

Die drei Szenarien stellen grundlegende Interaktionsmöglichkeiten dar. Darüber hinaus können beliebig komplexe Netzwerke unter Beteiligung vieler Parteien existieren. Durch die Anwendung der URL-basierten Identifikatoren sind die Querbeziehungen einfach verfolgbar. Auch sind die Systeme, die ein mit einem Identifikator identifiziertes Objekt verwalten, durch die Namensraum-URL adressiert (vergleiche Abschnitt 5.2) und können so direkt angesprochen werden. Die Zugriffe auf Historiendatenbanken und andere externe Datenquellen benötigen geeignete Adapter. Die Auswahl des geeigneten Adapters kann anhand des Wertes des Merkmals *metaModel:dataInterfaceType* erfolgen. Ihre Implementierung obliegt den Anbietern der Verwaltungssysteme.

5.8 Bezug zur Verwaltungsschale

Die bisher dargelegten Verwaltungsvorgänge gehen immer von der Verwaltung mehrerer Entitäten in einem Verwaltungsserver aus. Die Idee der Verwaltungsschale im Verständnis von Industrie 4.0 besagt, dass einer Einheit eine spezifische Komponente zu ihrer Verwaltung zugewiesen wird. Übertragen auf das hier vorgestellte Konzept ist dies also ein Verwaltungsserver, der genau eine Einheit verwaltet. Da Verwaltungsschalen eindeutig adressierbar sind, können sie direkt als Anbieter des in Abschnitt 5.5.3 vorgestellten Dienstes fungieren. In diesem Falle dient der Identifikator *metaModel:self* als Identifikation des ersten Abbilds der von der Verwaltungsschale verwalteten Einheit. Damit wird der vorgestellte Dienst zur Schnittstelle der Verwaltungsschale für die Abfrage der Lebenszyklus-

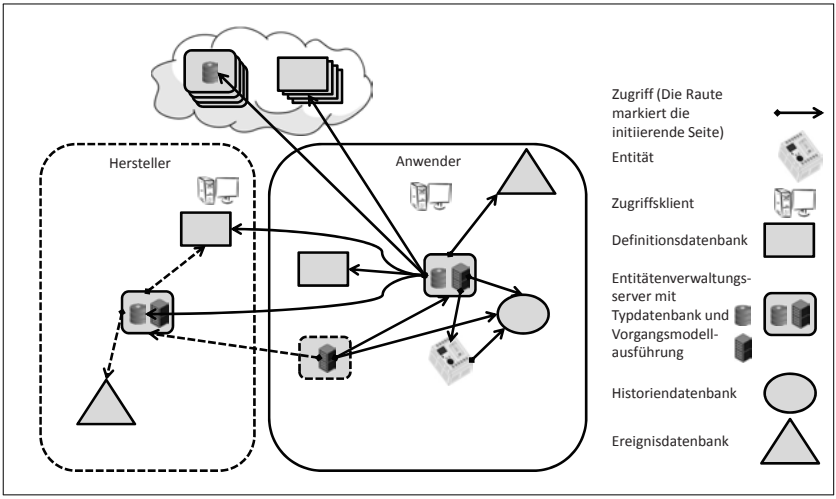


Abbildung 5.9: Interaktionen im Gesamtsystem ausgehend von Interaktionen zwischen Entitätenverwaltungsservern. Die betreffende Entität befindet sich beim Anwender und wird von ihm und vom Hersteller verwaltet.

informationen der verwalteten Einheit. Hat eine Einheit verschiedene Verwaltungsschalen in verschiedenen Verantwortungsbereichen, so verhalten sich diese analog zu den vorigen Ausführungen. Ob die Verwaltung des Lebenszklusses der Einheiten von zentralen Servern für mehrere Einheiten ausgeführt wird, oder von Komponenten für einzelne Einheiten, ist eine Designentscheidung des Nutzers.

6 Prototyp und Evaluation

Im Folgenden werden zunächst vier Anwendungsfälle für die Verwaltung von Lebenszyklusdaten beschrieben. Anschließend wird der auf der vorgestellten Schnittstelle basierende Prototyp eines Entitätenverwaltungssystems beschrieben. Dieser wird anhand der Anwendungsfälle evaluiert. Dabei wird zum einen Herausgestellt, wie breit die Anwendungsmöglichkeiten eines solchen Verwaltungssystems sind und zum anderen, wie weit diese mit der entwickelten Schnittstelle bewältigt werden können.

6.1 Beschreibung der Anwendungsfälle

Die prototypische Implementierung eines Entitätenverwaltungssystems muss sich an vier Anwendungsfällen messen. Jeder der Anwendungsfälle ist dabei darauf ausgelegt, bestimmte Anforderungen abzubilden. Dabei wird die Breite der Anforderungen an das Verwaltungssystem, wie auch die Breite seiner Nutzungsmöglichkeiten deutlich. Die Anwendungsfälle betreffen:

1. Datenvielfalt und Flexibilität in der retrospektiven Ist-Beschreibung von Einheiten;
2. Unternehmensübergreifende Zusammenführung einheitsspezifischer Informationen;
3. Proaktive Nutzung der anfallenden Informationen im Umfeld einer Einheit und
4. Verfolgung virtueller Einheiten und Prädiktion der Eigenschaften realer Einheiten.

Diese Anwendungsfälle werden im Folgenden beschrieben. Ihre Umsetzung mit dem Prototypen wird in Abschnitt 6.3 vorgestellt und evaluiert.

Datenvielfalt und Flexibilität

Dieser Anwendungsfall demonstriert die retrospektive Verfolgung des Zustands verfahrenstechnischer Anlagenmodule während ihres Einsatzes in verschiedenen Anlagenkonfigurationen beim Anwender. Da die Module über eigene Kleinleitsysteme verfügen, tragen sie jeweils ihre eigene Verwaltungseinheit in sich. Diese wurde vor der Übergabe an den Anwender vom Hersteller initial mit Informationen zum Auslieferungszustand gefüllt. Den Anwender interessiert zusätzlich, welche Medien zu welchem Zeitpunkt in einem Modul verarbeitet wurden und wie die Anlagenkonfiguration, in der das Modul eingesetzt wird, strukturiert ist. Folglich wird das Modell des Moduls im Verwaltungssystem erweitert (Flexibilität), wobei neben einfachen, werttragenden Eigenschaften auch strukturelle Eigenschaften abgebildet werden (Datenvielfalt). Abbildung 6.1 zeigt den Anwendungsfall. Wie in der Abbildung angedeutet ist die retrospektive Verfolgung der Eigenschaften Teil der Verwaltungsschale eines jeden Moduls. Ebenso haben die Anlagen, in denen die Module eingesetzt werden, Verwaltungsschalen. Diese referenzieren sich gegenseitig und drücken damit die Struktur des Anlagenverbundes aus.

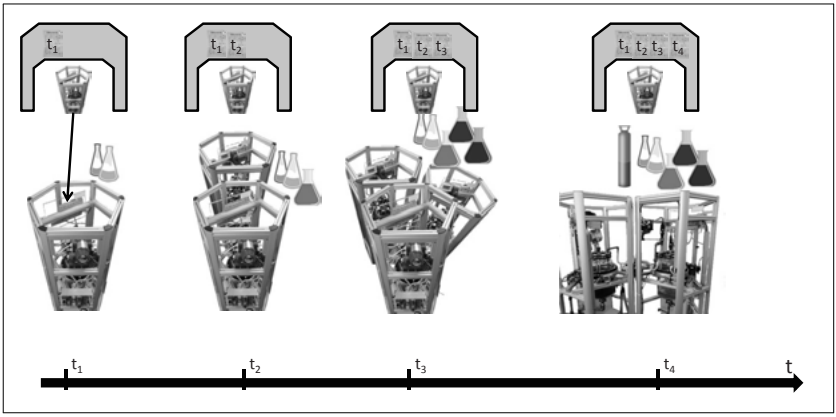


Abbildung 6.1: Zu jedem Anlagenmodul werden vom Hersteller angegebene und vom Nutzer definierte Eigenschaften verwaltet. Letztere betreffen unter anderem die Anlagenstruktur, in der das Modul eingesetzt wird, und die mit dem Modul verarbeiteten Stoffe. So werden werttragende und strukturelle Informationen nach Nutzerdefinition verwaltet.

Unternehmensübergriff

In diesem Anwendungsfall wird von der Wartung einer Werkzeugmaschine durch einen Wartungsdienstleister ausgegangen. Dieser hat bereits vorher Arbeiten an der spezifischen Maschine und an anderen Maschinen gleichen Typs durchgeführt und dabei ermittelte Informationen in seinem Entitätenverwaltungssystem abgelegt. Ferner wird die Maschine vom Nutzer verwaltet, wobei u. a. Betriebsbedingungen erfasst werden. Schlussendlich hält der Hersteller der Maschine Informationen zu ihrem Auslieferungszustand und zur weiteren Entwicklung ihres Typen. Das Wartungspersonal vor Ort greift nun auf alle drei Entitätenverwaltungssysteme zu, um die aktuellsten Informationen zur Werkzeugmaschine selbst und zu ihrem Typen (dies betrifft bspw. auch Wartungsanweisungen) in die Arbeit einfließen zu lassen, wie in Abbildung 6.2 dargestellt.

Proaktive Verwaltung

Dieser Anwendungsfall zeigt die Möglichkeiten aktiver Modelle in der Entitätenverwaltung. Dazu wird aus den Aktualdaten (Drehzahl, Einschaltzustand, Laufzeit etc.) einer Pumpe ihr Verschleißzustand abgeleitet. Dies geschieht durch ein aktives Modell in der Lebenszyklusabbildung der Pumpe. Diese Abbildung ist Teil der Verwaltungsschale der Pumpe. Sie wird folglich in einer der Instanz zugeordneten Softwarekomponente umgesetzt. Letztere ist jedoch in diesem Anwendungsfall - im Gegensatz zu den vorher vorgestellten Modulen - nicht in der Pumpe selbst, sondern im übergeordneten Leitsystem verortet, wie Abbildung 6.3 zeigt.

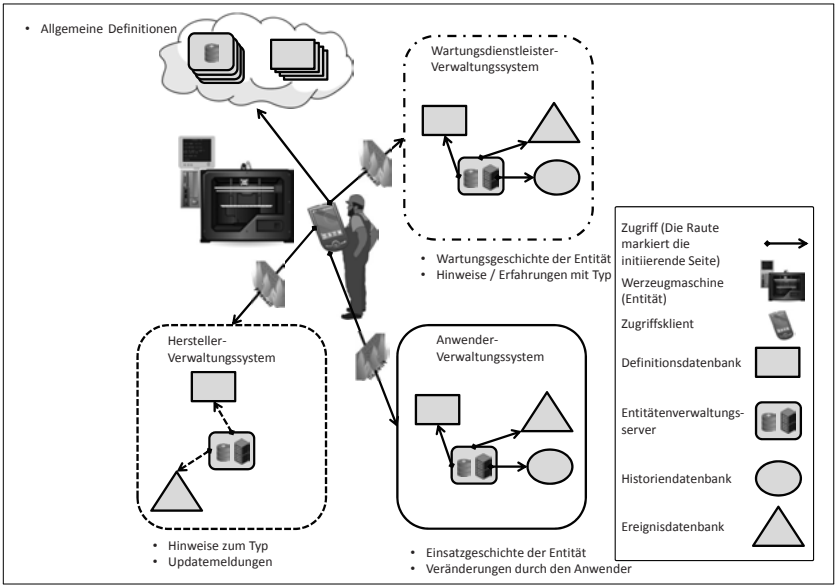


Abbildung 6.2: Der Wartungstechniker vor Ort fragt Informationen zu einer Maschine und zu ihrem Typ aus verschiedenen Quellen ab. Dabei interagiert auch er mit den Verwaltungssystemen des Herstellers und des Anwenders der Maschine. Diese geben Informationen nach intern definierten Regeln preis.

Virtuelle Einheiten und Prädiktion

Der vierte Anwendungsfall betrachtet Einheiten, die realistisch schwer abzugrenzen und stetigem Wandel unterworfen sind. In einem kontinuierlich arbeitenden Reaktor wird ein Polymer erzeugt. Wegen der kontinuierlichen Durchströmung und Änderung von Konzentrationen und äußeren Bedingungen muss für die Verwaltung eine Diskretisierung des Produktstromes stattfinden. Zugeleitete Edukte und das ausgeleitete, vollständig abreagierte Produkt werden dabei in einfache Volumeneinheiten aufgeteilt, da sich hier keine Reaktionen ergeben und Veränderungen vernachlässigt werden können. Während der Reaktion jedoch vergehen die Edukte und das Produkt entsteht. Es findet also eine Umwandlung der Einheiten statt. Um dies abzubilden, wird die Mischung der im Reaktor befindlichen Reaktanden als zusätzliche verwaltete Einheit betrachtet. Ihre Eigenschaften werden aus denen der zugeleiteten Edukte und der äußeren Bedingungen abgeleitet und mit aktiven Modellen berechnet. Daraus ergeben sich durch Diskretisierung über die Zeit wiederum die Eigenschaften der diskretisierten Volumina des ausgeleiteten Produkts. Diesen werden Verwaltungseinheiten zugewiesen, die von den aktiven Modellen mit Daten gefüllt werden, wie in Abbildung 6.4 dargestellt.

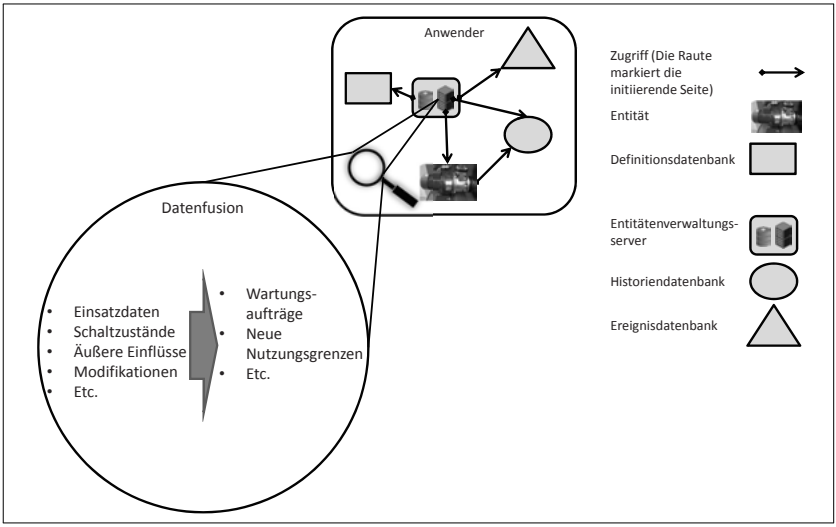


Abbildung 6.3: Die Pumpe ist im Verwaltungssystem mit einem aktiven Modell verbunden, das ihre Eigenschaften und Umgebungsbedingungen verknüpft und daraus neue Informationen generiert. Damit werden bspw. Wartungsaufträge generiert.

6.2 Aufbau des Prototypen

Als Prototyp der Entitätenverwaltung wurde jeweils eine Implementierung eines Verwaltungsservers und eines Klienten umgesetzt. Definitionsdatenbanken wurden nicht umgesetzt, da diese erstens keine aktiven Komponenten haben müssen und zweitens der Abgleich der Semantik verschiedener Definitionen nicht Fokus dieser Arbeit ist. Der Entitätenverwaltungsserver wurde im Laufzeitsystem ACPLT/rte umgesetzt, da dieses eine solide Grundlage für die Handhabung von Objektmodellen bildet und aktive Elemente unterstützt werden. Außerdem erlaubt es das Hinzufügen von Modellen und ihre Erkundung zur Laufzeit. Dabei wird auf die Technologien ACPLT/ks und OPC-UA gesetzt. Der Klient wurde in C# implementiert, da hier eine automatische Transformation der XML-Beschreibung der Schnittstelle in direkt nutzbare Klassen möglich ist. Beide Komponenten werden im Folgenden näher vorgestellt.

6.2.1 Entitätenverwaltungsserver

Die Implementierung des Verwaltungsservers ist als Bibliothek für die Laufzeitumgebung ACPLT/rte umgesetzt. Sie besteht aus zwei logischen Ebenen. Erstens gibt es die als Klassen umgesetzten Modellelemente (vergleiche Abbildung 5.5) und zweitens die Dienstschnittstelle. Es können mehrere Instanzen der Entitätenverwaltung gleichzeitig auf einem System laufen. Die Laufzeitumgebung stellt dabei die grundlegenden Funktionen zur Kommunikation und Verwaltung instanziierter Objekte zur Verfügung. Die Implementierung erfolgte in ANSI-C.

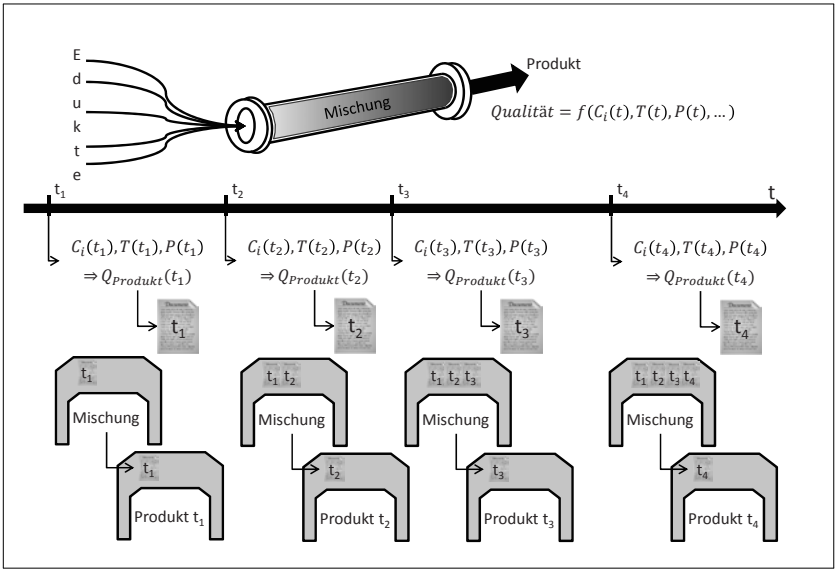


Abbildung 6.4: Die Mischung im Reaktor wird als Entität verwaltet. Aus ihren gemessenen Eigenschaften zu bestimmten Zeitpunkten ergeben sich die Eigenschaften des ausgeleiteten Produktstroms. Dieser wird diskretisiert und die sich daraus ergebenden Einheiten wieder als Entität verwaltet und mit den prädizierten Eigenschaften verknüpft.

Die in Abschnitt 5.4 vorgestellten Modellelemente wurden nahezu exakt umgesetzt. Lediglich einige Namen unterscheiden sich. Dies ist jedoch dem Nutzer verborgen, da er ausschließlich über die Dienstschnittstelle interagiert. Durch die exakte Umsetzung der Klassen und Datentypen gestaltet sich die Anbindung der Dienstschnittstelle einfach. Die in der Bibliothek definierten Klassen können zur Laufzeit instanziiert und modifiziert werden. Damit werden die Modelle der Einheiten dynamisch erzeugt. Diese können mit ACPLT/ks oder OPC-UA ohne Vorwissen erkundet werden. Letzteres ist zwar für die Nutzung der Entitätenverwaltung nicht notwendig, da hier auf die Dienstschnittstelle zurückgegriffen wird, ermöglicht aber die einfache Ankopplung zusätzlicher, nutzerspezifischer Funktionalität. Abbildung 6.5 zeit einen Ausschnitt der Instanzstruktur im Server.

Ein Teil der implementierten Klassen hat aktive Elemente. Dies betrifft insbesondere die Umsetzung von formalen Zusammenhangsmodellen. Diese werden, wenn sie mit einer Einheitenabbildung verknüpft sind, ausgeführt und können selbst über die Dienstschnittstelle mit anderen Verwaltungsservern interagieren. Formale Zusammenhangsmodelle werden in zwei Formen unterstützt:

1. Als Funktionsbausteinnetze unter Nutzung der vom Verwaltungsserver geladenen Bausteinbibliotheken und

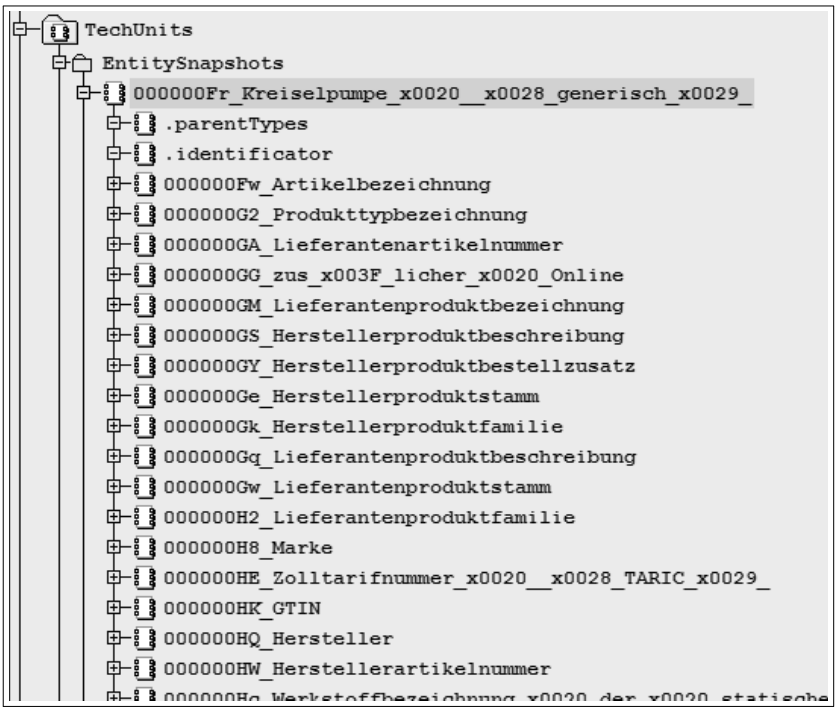


Abbildung 6.5: Einblick in die Instanzstruktur innerhalb des Verwaltungsservers unter Nutzung des ACPLT/KS-Klienten iFBSpro. Die dargestellten Daten wurden im dritten Anwendungsfall eingesetzt.

2. Als lua[1]-Skripte.

In beiden Fällen wird eine API zur Nutzung der Dienste der Entitätenverwaltung angeboten. Funktionsbausteinnetze wurden als Beschreibungsmöglichkeit gewählt, da sie eine intuitive Basis zur Entwicklung kontinuierlich arbeitender Modelle darstellen und strukturell zur Laufzeit erkundbar sind (bei Nutzung der verwendeten Laufzeitumgebung). Ein solches Modell ist in Abbildung 6.6 abgebildet, wobei die Darstellung durch eine Erkundung des Modells zur Laufzeit in einem Browser erzeugt wurde. Zusätzlich sollten Modelle aus einer Skriptsprache erzeugbar sein, um sie dynamisch in einen Server laden zu können, ohne auf die unterliegende Hardware oder das Vorhandensein spezifischer (Baustein-)Bibliotheken angewiesen zu sein. Die Wahl fiel dabei auf lua, weil diese Sprache sowohl als Skript, als auch als Bytecode geladen werden kann. Dadurch können sowohl zur Laufzeit lesbare Modelle, als auch dem Nutzer verborgene Funktionalität mit derselben Umgebung umgesetzt werden. Ferner ist die lua-Umgebung für gute Performanz bekannt und sehr einfach zu integrieren.

[1]Siehe lua.org

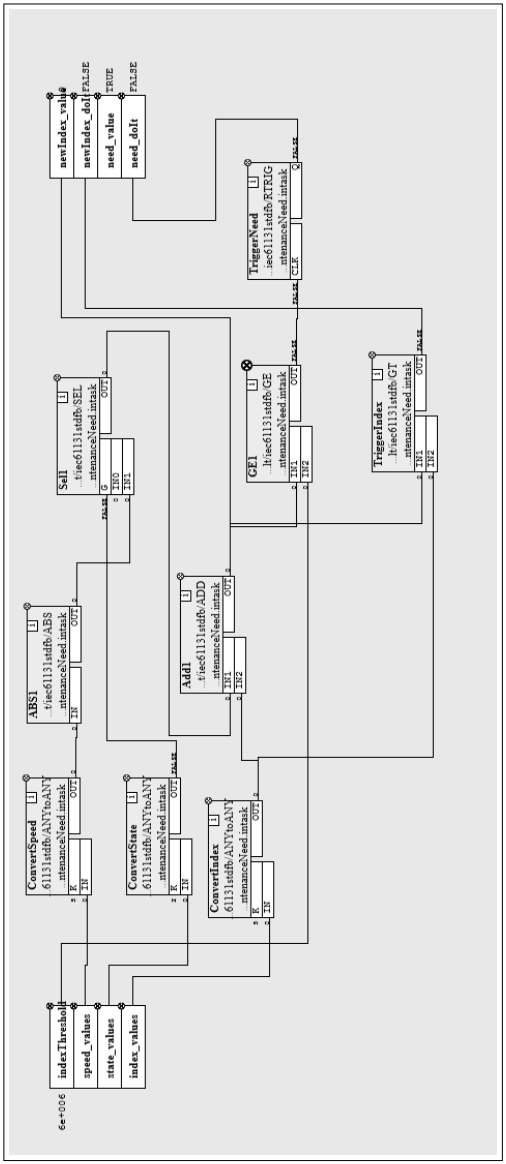


Abbildung 6.6: Ein formales Modell als Funktionsbausteinstruktur. Dieses Modell wurde für den dritten Anwendungsfall eingesetzt.

Das zweite logische Element, die Dienstschnittstelle, wurde als Erweiterung des Kommunikationssystems der Laufzeitumgebung umgesetzt. Sie ist als Klasse umgesetzt, die beim Aufruf eines Verwaltungsdienstes durch einen Klienten dynamisch instanziiert und

nach Abschluss der Interaktion mit diesem Klienten wieder gelöscht wird. Die Instanz der Schnittstellenklasse stellt also das Kopfelement des Kommunikationskanals zwischen Klient und Server dar. Die Struktur der XML-Beschreibung der Dienste erlaubt eine modulare Umsetzung der Serialisierungs- und Deserialisierungsfunktionen. Dadurch blieb der Implementierungsaufwand auch ohne Nutzung externer Bibliotheken in einem annehmbaren Rahmen. Nach Auswertung der Aufrufnachrichten werden im Laufzeitsystem die entsprechenden Objekte erzeugt, gelöscht oder manipuliert und eine Antwortnachricht gesendet. Die Durchführung der Dienste ist nicht interruptiv. Das Laufzeitsystem entscheidet selbstständig, wann ein Aufruf ausgeführt wird. Die Aufrufreihenfolge bleibt dabei gewahrt.

6.2.2 Klient

Der Klient wurde in C# unter Nutzung von Winforms implementiert. Er stellt die Nutzerschnittstelle der Entitätenverwaltung dar. Dabei wurden zwei Module umgesetzt. Das erste Modul ist eine C#-Implementierung der Dienstschnittstelle. Sie erlaubt die Nutzung der Verwaltungsdienste für Entitäten und Merkmale. Diese werden vollständig abgebildet. Die Dienste für Semantikdefinitionen und Ereignisse wurden hier ausgelassen, da sie für die Evaluation nicht notwendig sind. Durch eine automatische Transformation der XML-Beschreibung der Schnittstelle zu C#-Klassen konnte die Implementierungszeit deutlich verkürzt werden. Das zweite Modul nutzt die Schnittstellenklassen und stellt die eigentliche Nutzerschnittstelle dar. Diese ist als Fenster mit mehreren Untereinheiten aufgebaut, die die verschiedenen Operationen abbilden. Direkt dem Nutzer zugänglich sind Operationen zur Erzeugung von Prototypen und Einheitenabbildern, sowie zum Hinzufügen von Merkmalsrepräsentationen. Außerdem können die verwalteten Einheiten gefiltert abgefragt und die Historie einer Einheit erkundet werden. Informationen zu diesen Entitäten und ihren Eigenschaften können dann interaktiv abgefragt werden. Ferner gibt es eine Schnittstelle, die das Auslesen tabellarischer Dateien erlaubt, aus denen größere Einheitenabbilder oder Prototypen erzeugt werden. Diese Schnittstelle ermöglicht auch die An- und Abkopplung weiterer Modelle und Beschreibungen. Der Klient ist auf die interaktive Nutzung ausgelegt. Ein Klient zur maschinellen Nutzung bspw. innerhalb eines Automatisierungssystems wäre ebenso möglich. Für die Evaluation ist dies jedoch nicht nötig, da hier die Dateischnittstelle genutzt werden kann. Abbildung 6.7 zeigt das Hauptfenster des Klienten und Abbildung 6.8 die Merkmale der verwalteten Einheit.

6.3 Umsetzung und Evaluation der Anwendungsfälle

Im Folgenden wird die Umsetzung der vier genannten Anwendungsfälle gezeigt. Dabei wird die entwickelte Systemstruktur und Schnittstelle evaluiert, indem die Umsetzung und damit verbunden die Interaktion der Systemkomponenten und die Datenabfragen Schritt für Schritt dargestellt werden. Daraus wird abgeleitet, was die Systemstruktur in diesem Zusammenhang leisten kann.

6.3.1 Anwendungsfall 1: Datenvielfalt und Flexibilität

Dieser Anwendungsfall geht von Anlagenmodulen aus, die ihre eigene Steuerung und weitere Intelligenz in sich tragen. Dies schließt einen Entitätenverwaltungsserver ein, der eine

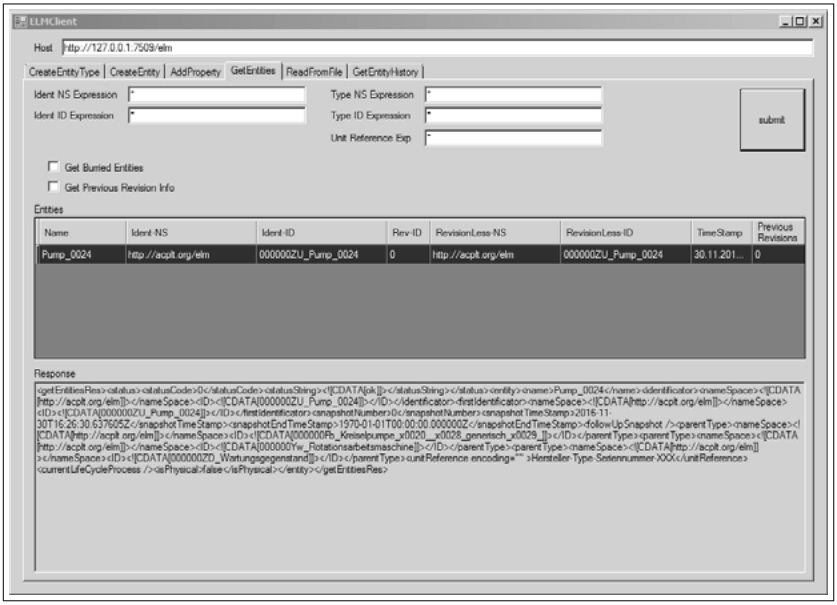


Abbildung 6.7: Hauptfenster des Klienten mit einer betrachteten Entität.

Verwaltungsschale des jeweiligen Moduls umsetzt. Was ihre interne Struktur und Daten angeht, so ist diese in besagter Verwaltungsschale gesichert. Dies wurde vom Hersteller umgesetzt und basiert dadurch auf dem Zustand zum Zeitpunkt der Fertigstellung des Moduls und den Daten, die der Hersteller zur Verwaltung vorgesehen hat. Abbildung 6.9 zeigt einen Ausschnitt dieser Merkmale.

Der Nutzer der Module ist an weiteren Informationen interessiert. Daher fügt er dem Modell des Moduls durch Nutzung der Operation `addProperty` dynamisch weitere zu verarbeitende Merkmale hinzu. Abbildung 6.10 zeigt dies im Anzeigefenster des Klienten. Im in diesem Anwendungsfall betrachteten Zeitraum ändert sich die interne Konfiguration des Moduls nicht. Daher bleiben die vom Hersteller angegebenen Merkmale unverändert. Das Modul wird jedoch in verschiedenen Anlagen eingesetzt, an verschiedenen Verknüpfungspunkten mit anderen Modulen verknüpft und mit verschiedenen Chemikalien betrieben. Dazwischen erfolgen Reinigungen. Bei all diesen Vorgängen ändern sich die vom Anwender genutzten Merkmale. Dies wird von den zuständigen Mitarbeitern per Dienstaufruf durch Hinzufügen von Aussagen zu den Merkmalen vermerkt[2]. Beim Wechsel zwischen verschiedenen Anlagenkonfigurationen werden neue Einheitszustandsabbildungen angelegt, wie Abbildung 6.11 zeigt.

Diese Abbildungen beschreiben den Lebenszyklus des Moduls. Die Abfrage, wie in Abbildung 6.11 dargestellt, erfolgt mit der Operation `getEntityHistory`. Dabei werden auch die Gültigkeitszeiträume der Abbildungen abgefragt. Dies ist in Abbildung 6.11

[2]Eine automatisierte Verarbeitung ist auch möglich, aber nicht Teil dieses Anwendungsfalls.

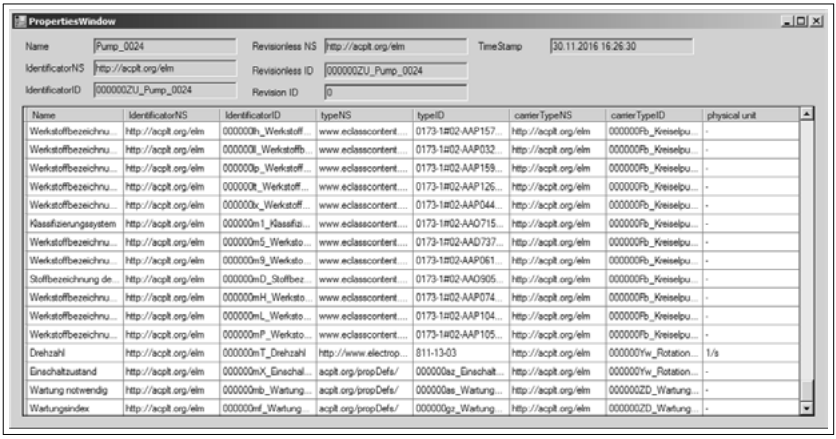


Abbildung 6.8: Merkmale einer Entität dargestellt im Klienten. Es handelt sich um dieselbe Entität wie in Abbildung 6.5.

in der unten dargestellten kompletten Antwortnachricht sichtbar. Mit der Operation `getSnapshotAtTime` kann auch die zu einem bestimmten Zeitpunkt gültige Abbildung der Entität abgefragt werden. Anschließend werden mit `getProperties` die zu diesem Zeitpunkt verwalteten Merkmale erkundet und schließlich mit `getStatements` die Aussagen zu den Merkmalen von Interesse ausgewertet. Diese Schritte können auch durch den Aufruf von `getValueAtTime` umgesetzt werden. Diese Operation fasst die vorgenannten Vorgänge zusammen. Sie gehört zur erweiterten Dienstschnittstelle. Abbildung 6.12 zeigt das Resultat einer Aussagenabfrage.

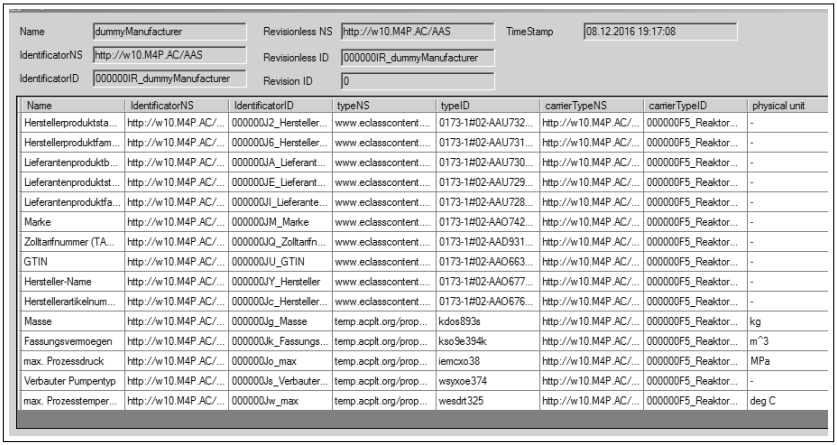
Es zeigt sich, dass das System in der Lage ist, Einheitenmodelle dynamisch zu erweitern und zu erkunden. Des Weiteren werden die Merkmale mit Werten gefüllt, wobei ihre Änderungshistorie erhalten bleibt, da jeder vermerkte Wert durch eine eigene Aussage dargestellt wird. Schließlich erfolgt dienstbasiert die Abfrage der Werte und ihrer Veränderungen und der Verfolgungszeiträume der Entität selbst. Da mit den Merkmalen auch strukturelle Eigenschaften, wie die Verbindung zweier Module, abgebildet werden können, ist die Erfassung und retrospektive Verfolgung von Lebenszyklusinformationen vollständig umsetzbar.

6.3.2 Anwendungsfall 2: Unternehmensübergriff

Dieser Anwendungsfall geht von einem Wartungstechniker aus, der eine Werkzeugmaschine beim Anwender warten muss und dazu Informationen zur konkreten Maschine von verschiedenen Quellen anfordert. Zwei Anforderungen werden dabei herausgestellt:

1. Das Auffinden der Informationen zu einer Einheit in verschiedenen, unternehmensübergreifend verteilten Systemen und

2. Die Zugriffsmöglichkeit auf die verschiedenen Systeme von einem (mobilen) Endgerät in einem beliebigen Netz.



Name	dummyManufacturer	Revisionless NS	http://w10.M4P.AC/AAS	TimeStamp	08.12.2016 19:17:08		
IdentificatorNS	http://w10.M4P.AC/AAS	Revisionless ID	000000IR_dummyManufacturer				
IdentificatorID	000000IR_dummyManufacturer	Revision ID	0				

Name	IdentificatorNS	IdentificatorID	typeNS	typeID	carrierTypeNS	carrierTypeID	physical unit
Herstellerproduktsta...	http://w10.M4P.AC/...	000000J2_Hersteller...	www.eclasscontent....	0173-1#02-AAU732...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Herstellerproduktfam...	http://w10.M4P.AC/...	000000J6_Hersteller...	www.eclasscontent....	0173-1#02-AAU731...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Lieferantenproduktb...	http://w10.M4P.AC/...	000000JA_Lieferant...	www.eclasscontent....	0173-1#02-AAU730...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Lieferantenproduktst...	http://w10.M4P.AC/...	000000JE_Lieferant...	www.eclasscontent....	0173-1#02-AAU729...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Lieferantenproduktfa...	http://w10.M4P.AC/...	000000JI_Lieferante...	www.eclasscontent....	0173-1#02-AAU728...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Marke	http://w10.M4P.AC/...	000000JM_Marke	www.eclasscontent....	0173-1#02-AAO742...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Zolltarifnummer (TA...	http://w10.M4P.AC/...	000000JQ_Zolltarifn...	www.eclasscontent....	0173-1#02-AAD931...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
GTIN	http://w10.M4P.AC/...	000000JU_GTIN	www.eclasscontent....	0173-1#02-AAO663...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Hersteller-Name	http://w10.M4P.AC/...	000000JY_Hersteller	www.eclasscontent....	0173-1#02-AAO677...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Herstellerartikelnum...	http://w10.M4P.AC/...	000000Jc_Hersteller...	www.eclasscontent....	0173-1#02-AAO676...	http://w10.M4P.AC/...	000000F5_Reaktor...	-
Masse	http://w10.M4P.AC/...	000000Jg_Masse	temp.acplt.org/prop...	kdos893s	http://w10.M4P.AC/...	000000F5_Reaktor...	kg
Fassungsvermoegen	http://w10.M4P.AC/...	000000Jk_Fassungs...	temp.acplt.org/prop...	kso9e394k	http://w10.M4P.AC/...	000000F5_Reaktor...	m^3
max. Prozessdruck	http://w10.M4P.AC/...	000000Jo_max	temp.acplt.org/prop...	iemcxo38	http://w10.M4P.AC/...	000000F5_Reaktor...	MPa
Verbauter Pumpentyp	http://w10.M4P.AC/...	000000Js_Verbauter...	temp.acplt.org/prop...	wsyxoe374	http://w10.M4P.AC/...	000000F5_Reaktor...	-
max. Prozesstemper...	http://w10.M4P.AC/...	000000Jw_max	temp.acplt.org/prop...	wesdrt325	http://w10.M4P.AC/...	000000F5_Reaktor...	deg C

Abbildung 6.9: Ausschnitt der vom Hersteller verwendeten Merkmale zur Beschreibung des Moduls.

Die Auffindbarkeit der Daten wird durch das Datenmodell sichergestellt, die Zugriffsmöglichkeit durch die Nutzung eines geeigneten Kommunikationssystems in Verbindung mit entsprechenden Zugriffsregelungen.

Zum Auffinden der zu einer Einheit gespeicherten Informationen wird in erster Linie die Operation `getEntities` verwendet, wobei der Filter für die Referenz zur tatsächlichen Einheit verwendet wird. So kann in verschiedenen Systemen einfach geprüft werden, ob sie Informationen zu einer konkreten Einheit verwalten. Des Weiteren wird das Merkmal *metaModel:otherRepresentation* in bereits gefundenen Repräsentationen abgefragt, um bereits bekannte Repräsentationen in anderen Systemen zu finden. Findet also ein Wartungstechniker im System des Anwenders diesen Verweis auf ein System eines Dritten, so kann er auch dorthin eine Abfrage starten. Damit kann ein Netz von Verwaltungssystemen gefunden und abgefragt werden, um ein vollständiges Bild der Einheit zu erhalten. Schlussendlich können die Namensräume der abgefragten Einheitsrepräsentationen und besonders ihrer Typen verglichen werden. Damit kann bspw. auf weitere Informationen des Herstellers zum Typ der Werkzeugmaschine zugegriffen werden.

Unternehmensübergreifende Zugriffsmöglichkeiten benötigen durch die Beteiligten freigegebene und gesicherte Zugriffswege. Die Nutzung von HTTP bzw. HTTPS als primäres Übertragungsprotokoll vereinfacht dies, da die meisten Unternehmen ihre Firewalls ohnehin für Zugriffe mit diesem Protokoll von außen und nach außen einrichten, weil es in vielen Unternehmensanwendungen Verwendung findet. Ferner ist die Nutzung von Authentifizierung und Verschlüsselung im Protokoll bereits vorgesehen. Mit einem entsprechenden Klienten-Zertifikat kann sich der Wartungstechniker somit gegenüber allen anderen beteiligten Parteien ausweisen. Diese können dann entscheiden, ob sie den Zugriff zulassen. Die Authentifizierung mit Klienten-Zertifikaten ist vorteilhaft, da sie keine vom Nutzer einzugebenden Passwörter benötigt und dabei sicherer ist als ein solches. Zu beachten ist jedoch, dass sich mit einem solchen System ohne Passwort nicht der Nutzer, sondern das

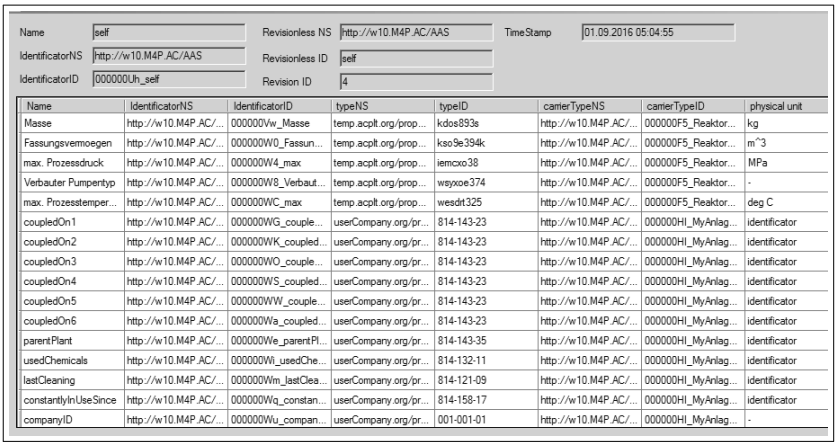


Abbildung 6.10: Ausschnitt der vom Hersteller verwendeten Merkmale und vom Anwender hinzugefügte Merkmale zur Beschreibung des Moduls.

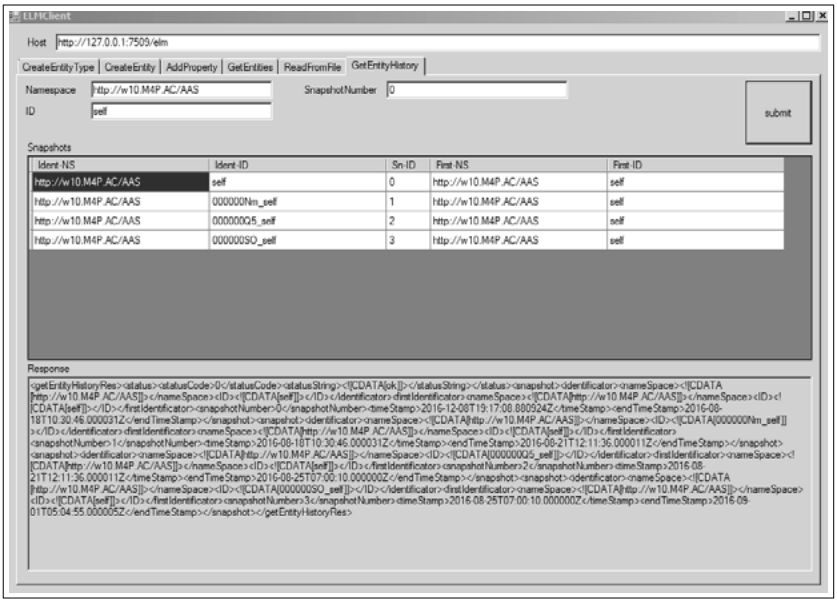


Abbildung 6.11: Ausschnitt der vom Hersteller verwendeten Merkmale und vom Anwender hinzugefügte Merkmale zur Beschreibung des Moduls.

Name	IdentificatorNS	IdentificatorID	TypeNS	TypeID	StatingInstanceNS	StatingInstanceID	StateTime	TimeStamp	Equality	Value
otherModule	http://w10.M4P....	0000000Xa_other...	type NS	type ID	http://m4p.ac/eln	worker123	21.08.2016 12:11:36	21.08.2016 12:1...	equals	<identificator><n...

Abbildung 6.12: Aussage zum Merkmal *coupledOn2* des Anlagenmoduls. Diese bezeichnet ein anderes Modul, das an Anschlusspunkt 2 verbunden ist. Damit kann die Anlagenstruktur abgebildet werden.

Gerät ausweist. Bei Verlust müssen also entsprechende Maßnahmen ergriffen werden. Dies ist Stand der Technik und entsprechende Systeme zur Verwaltung und Validierung der Zertifikate sind verfügbar.

Es wird hier davon ausgegangen, dass der Wartungstechniker über das Internet Verbindung zu den verschiedenen Parteien aufnimmt. Ob er dabei das Netz des Anwenders mit nutzt, oder bspw. eine UMTS-Verbindung nutzt, hängt von den Vereinbarungen zwischen Wartungsunternehmen und Anwender ab. Auch VPN kann für die Verbindung zu verschiedenen Parteien genutzt werden. Dies kann notwendig werden, wenn eine Partei nicht bereit ist, die Schnittstelle direkt an das Internet anzuschließen. Die Nutzung von VPN muss den beteiligten Parteien vorab bekannt sein, um sie einrichten zu können. Ferner ist sie nicht per se sicherer als eine direkte Verbindung über HTTPS.

Die Ausführungen zeigen, dass das System in der Lage ist, unternehmensübergreifend Informationen zu finden und durch die Dienstschnittstelle bereitzustellen. Ferner wird das genutzte Protokoll HTTP bereits häufig für die unternehmensübergreifende Kommunikation eingesetzt und die Konfiguration ist entsprechend gut umsetzbar. Die Sicherstellung von Integrität, Vertraulichkeit und Zugriffsberechtigung der angefragten Daten ist Stand der Technik und somit ebenfalls ohne Schwierigkeiten im Unternehmen einsetzbar.

6.3.3 Anwendungsfall 3: Proaktive Verwaltung

Proaktive Verwaltung nutzt aktive Modelle, um erfasste Informationen weitergehend zu verarbeiten und daraus entweder weitere Eigenschaften derselben oder anderer Einheiten abzuleiten. Dabei können auch neue Repräsentationen erzeugt werden. Unter Betrachtung des Datenmodells ist die Frage, wie aktive Modellelemente die notwendigen Informationen auffinden, am wichtigsten. Die einzelnen Implementierungen können Funktionalität anbieten, um über einfache Wertzuweisungen - wie sie für die Ziele aktiver Modelle definiert wurden - hinausgehende Aktionen auszuführen.

Zum Auffinden von Quelldaten und Zuweisungszielen wurden im Datenmodell Pfade definiert, die der Struktur

$$Einheitentyp \rightarrow Merkmaltyp \rightarrow Aussagentyp \rightarrow Einheitentyp \rightarrow \cdots \rightarrow Aussagentyp$$

folgen, wobei ein solcher Pfad mit einem Aussagentyp enden muss und alle vorhergehenden Aussagen vom Typ *metaModel:definedIdentificator* sein müssen. Die erste Bedingung stellt sicher, dass am Ende des Pfades ein Wert steht und die zweite, dass vorher durch die Nutzung der Identifikatoren Pfadzusammenhänge hergestellt werden können. Der Startpunkt eines jeden Pfades ist die mit einem Modell konkret verbundene Einheitszustandsabbildung. Durch die Nutzung der Typen sind diese Pfade nicht zwingend eindeutig. Hat eine Einheit mehrere Merkmale desselben Typs oder ein Merkmal mehrere Aussagen desselben Typs, so wird mehr als ein Ziel gefunden. Damit müssen die Modelle entsprechend umgehen können. Eine Möglichkeit mehrfache Funde zu verhindern sind zusätzliche Filter.

Diese können nur begrenzt generischer Natur sein, da sie sich auf Identifikatoren beziehen müssten, die naturgemäß keinen semantischen Bezug zum identifizierten Objekt haben. Konkret bedeutet das, dass sich die Identifikatoren zweier aufeinander folgender zeitbezogener Einheitszustandsabbildungen gravierend unterscheiden können. Eine Filterung auf dieser Ebene ist daher nur in spezifischen Fällen sinnvoll, wenn die Ziele eines Pfades exakt bekannt sind. Dies widerspricht dem generischen Ansatz der Zusammenhangsmodellierung.

In der Umsetzung des Anwendungsfalls im Prototypen wird ein funktionsbausteinbasiertes Modell verwendet, um aus der Drehzahl einer Pumpe und ihrem Einschaltzustand einen Wartungsindex zu generieren. Das Modell wurde bereits in Abbildung 6.6 dargestellt. Abbildung 6.13 zeigt die Außenansicht des Modells. Der Wartungsindex wird in einem Ziel zwischengespeichert (*newIndex_values*), welches im nächsten Durchlauf des Modells wieder als Quelle einfließt (*index_values*). Wird ein parametrierter Schwellwert (*indexThreshold*) überschritten, wird die Notwendigkeit der Wartung angezeigt (*need*). Quellen und Ziele wurden vorher als Namen der jeweiligen Pfade festgelegt (*speed, state, index, newIndex* und *need*).

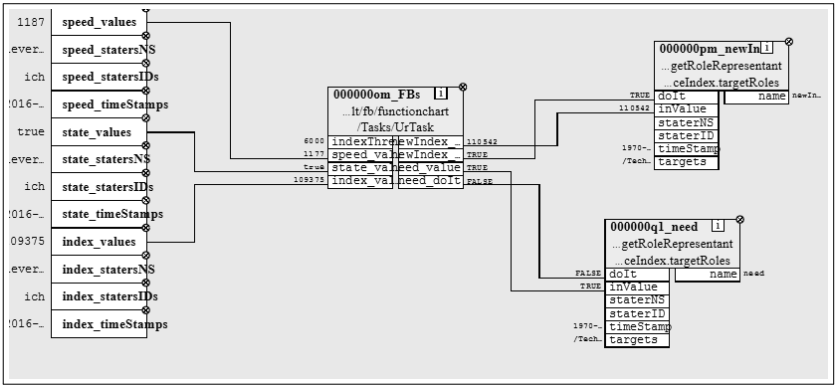


Abbildung 6.13: Außenansicht des Funktionsbausteinmodells zur Beurteilung der Wartungsnotwendigkeit einer Pumpe.

Die hier verwendeten Pfade verlaufen von der Einheitszustandsabbildung zu einem Merkmal, das die gewünschte Eigenschaft abbildet und dann zu einer den Ist-Wert darstellenden Aussage. Der folgende Pfad wurde im konkreten Modell verwendet, um die Wartungsnotwendigkeit anzuzeigen:

Typ	**Einheitstyp**	**Merkmaltyp**	**Aussagentyp**
Namensraum	http://acplt.org/elm	http://acplt.org/propDefs/	http://acplt.org/elm
ID	000000a5_Wartungsgegenstand	000000as_Wartung_notwendig	000000nU_isValue

Mit längeren Pfaden ist es möglich, auch Strukturen zwischen verschiedenen Einheiten abzubilden. Ferner wurden im Prototypen neben Funktionsbausteinmodellen auch luaskriptbasierte Modelle implementiert. Im ersten Fall wird vorab ein Prototypenmodell an-

gelegt, das bei Verknüpfung an eine Einheitenabbildung kopiert und wie in Abbildung 6.13 dargestellt verknüpft wird. Lua-Skripte werden direkt als formale Modelle übergeben. Bei Verknüpfung mit einer Einheitenabbildung wird ein Interpreterbaustein erzeugt und das Skript (oder lua-Binärformat) hineingeladen und genauso verknüpft, wie das Funktionsbausteinmodell. Über die Wert-zuweisende Funktionalität hinaus können Bausteine (oder lua-Funktionsaufrufe) die Dienste der Entitätenverwaltung abrufen und damit bspw. auch neue Abbildungen generieren. Mit dieser Funktionalität kann das System proaktiv Funktionen im Verwaltungssystem übernehmen und flexibel auf die verwalteten Einheiten anwenden.

6.3.4 Virtuelle Einheiten und Prädiktion

Dieser Anwendungsfall erweitert den vorangegangenen hin zur Nutzung der aktiven Modellelemente zur Prädiktion der Qualität eines Produkts. Dabei wird auch die bereits genannte Fähigkeit des Demonstrators genutzt, mehr als nur Wertzuweisungen durch aktive Modelle zu steuern. Dies dient hier der Veranschaulichung der Möglichkeiten. Für die Prädiktion selbst ist es nicht zwingend notwendig.

Ausgangssituation ist ein kontinuierlich arbeitender Reaktor, der mit Edukten versorgt wird, die durch Diskretisierung über geflossenes Volumen als Einheiten verwaltet werden. In einem bestimmten Zeitraum treten also je nach Fließgeschwindigkeit eine veränderliche Anzahl von Edukteinheiten in den Reaktor ein. Für diese Einheiten sind ihre Zusammensetzung, ihre Temperatur, der Druck in der Anlage und weitere Qualitätsgrößen bekannt. Nun kommt es zur Reaktion. Dabei werden besagte Einheiten umgewandelt in Produkteinheiten, deren Qualität von den Edukteinheiten und den Bedingungen im Reaktor abhängt. Mit einem aktiven Modell kann dies abgebildet und damit die Qualität des Produkts ermittelt werden. Die Abbildung des Vergehens der Edukteinheiten und Entstehens der Produkteinheiten kann entweder unabhängig vom Prozess erfolgen oder durch die aktiven Modelle umgesetzt werden. Im ersten Fall bleiben die Edukteinheiten sozusagen liegen, bis sie manuell als vergangen markiert werden und die Produkteinheiten entstehen periodisch, wobei die berechneten Qualitätsdaten von den aktiven Modellen als Wertzuweisungen gesetzt werden. Hier wird folglich keine spezielle Eigenschaft des Verwaltungssystems gefordert. Die Auffindung der neuen, noch nicht mit Werten belegten, Abbildungen der Produkteinheiten erfolgt dabei wieder über die bereits vorgestellten Pfade. Das aktive Modell ist dem Reaktor zugeordnet. In ihn eintretende Edukteinheiten werden über ein Merkmal *anlagenNamensraum:eingehendeEdukteinheit* und die Aussage *metaModel:definedIdentificator*[3] mit ihm verknüpft. Ebenso werden die Produkteinheiten durch das Merkmal *anlagenNamensraum:entstandeneProdukteinheit* mit ihm verknüpft. In diesem Fall wird es immer mehrere mögliche Quellen und Ziele geben, da der Reaktor über längere Zeiträume dieselbe Abbildung haben wird. Die Auffindung der korrekten Edukt- und Produkteinheiten erfolgt dabei über die Zeitstempel der Aussagen, die in den Modellen genutzt werden können, wie Abbildung 6.13 zeigt. Für den zweiten Fall muss es aus den aktiven Modellen heraus möglich sein, Einheiten vergehen und entstehen zu lassen. Dies erfordert eine entsprechende Schnittstelle im Verwaltungssystem, in dem die Modelle ablaufen. Dies wurde im Demonstrator umgesetzt. In der dargestellten Konstellation hat

[3]Ein anderer Aussagentyp, der einen Identifikator beinhaltet wäre an dieser letzten Stelle des Pfades auch möglich.

diese Art der Umsetzung jedoch noch keinen Vorteil. Wenn aber gefordert ist, dass die Reaktion der Edukte auch im Reaktor verfolgt werden muss, weil sie bspw. empfindlich auf über den Reaktionsweg hinweg schwankende Bedingungen reagiert, können virtuelle Einheiten helfen, die das halb-reagierte Produkt im Reaktor abbilden. Mit dieser Methode können auch über den Reaktor hinweg an verschiedenen Stellen gemessene Umgebungsbedingungen in die Modelle einfließen. Der Reaktor kann also als segmentierter Bilanzraum abgebildet werden.

Alle solchen Modelle müssen nicht zwingend auf real existierende Einheiten angewendet werden. Wird die Abbildung einer physischen Einheit losgelöst von dieser Einheit genutzt, so ist diese Abbildung selbst letztlich die Einheit von Interesse. Sie ist eine gedankliche Einheit, die die Merkmale einer physischen Einheit trägt. Dies wird hier als virtuelle Einheit bezeichnet. Sie sind besonders als Planungs- und Prädiktionsmodelle von Interesse. Um die Nutzung virtueller Einheiten zu vereinfachen, trägt jede Einheitenabbildung ein Attribut, das zeigt, ob es sich um die Abbildung einer physischen Einheit handelt oder nicht.

Dieser Anwendungsfall zeigte die Möglichkeit der Nutzung aktiver Modelle zur Qualitätsprädiktion, wobei zusätzlich die Mächtigkeit der im Demonstrator umgesetzten Schnittstelle zur Handhabung von Einheitsabbildungen in aktiven Modellen herausgestellt wurde. Ferner wurde die Nutzung virtueller Einheiten als Planungs- und Prädiktionsmittel dargestellt.

7 Diskussion und Ausblick

Die Erfassung und Verarbeitung aller in einem Unternehmen anfallenden Informationen ist ein wichtiges Entwicklungsziel sowohl in der Automatisierungstechnik, als auch darüber hinaus. Sie ist ein Schlüsselelement für die Umsetzung von Industrie 4.0. In diesem Umfeld wurden mehrere Wertschöpfungsketten identifiziert. Diese verknüpfen verschiedene Unternehmen durch den Austausch von (Produkt-) Einheiten. Letztere sind folglich die Kernelemente des Unternehmensübergriffs. Daher ist es sinnvoll, die Verwaltung der erfassten Daten mit den Einheiten zu verknüpfen, deren Eigenschaften sie abbilden. Die zu diskreten Zeitpunkten erfassten Daten ergeben auf einem Zeitstrahl dargestellt ein Abbild des Lebenszyklusses der zugehörigen Einheit. Dabei hat jede existierende Einheit allein aufgrund ihrer Existenz einen Lebenszyklus. Seine Verfolgung in einem Informationssystem macht die Einheit zur Entität. Die Einordnung der erfassten Daten anhand eines Zeitstrahls hat eine einfache Konsolidierbarkeit zur Folge, da erfasste Messwerte sich - von Fehlmessungen abgesehen - im Rahmen der Messgenauigkeit nicht widersprechen können. Sofern also Vertrauensbereiche und Randbedingungen mit abgebildet werden, können alle erfassten Daten ohne Probleme in Verbindung gesetzt und zusammengefasst werden.

Bislang gibt es kein System, dass die Erfassung und Verarbeitung für beliebige strukturierte und unstrukturierte Informationen erlaubt und ihren Austausch auch über Unternehmens- und Verantwortungsbereichsgrenzen unterstützt. Die vorhandenen Systeme und Ansätze gehen entweder von vorab festgelegten Einheitsstrukturen aus oder besagte Strukturen werden in einem aufwändigen Prozess standardisiert. Eine für die Nutzer des Systems einfache Möglichkeit, zusätzliche Informationen abzubilden, besteht meist nicht. Auch der Austausch der erfassten Informationen wird auf technologischer Ebene meist nicht festgelegt oder die Spezifikation ist nicht zugänglich, wodurch Interoperabilität zwischen verschiedenen Systemen nicht gewährleistet ist. Aus diesem Grund wurde im Rahmen dieser Arbeit ein Datenmodell entwickelt, dass die notwendige Flexibilität bereit stellt. Dieses Datenmodell wurde mit einer dienstbasierten Schnittstelle verknüpft, um den Datenaustausch auf technologischer Ebene zu spezifizieren und Interoperabilität zu erzeugen. Beides wurde in einem Demonstrator umgesetzt und anhand vierer Anwendungsfälle evaluiert. Dabei wurde differenziert, welche Fähigkeiten Datenmodell und Schnittstelle zuzuschreiben sind, und welche durch die spezifische Umsetzung des Demonstrators hinzukommen.

Die angesprochene Flexibilität erreicht das Datenmodell durch seine Struktur. Jedes Abbild einer Einheit setzt sich aus einer Aggregation von Merkmalsreferenzen und daran angebundenen Aussagen zusammen. Dabei sind die abgebildeten Eigenschaften der Einheiten zwar in Prototypen zusammengefasst, können aber jederzeit erweitert werden. Auch können für die Abbildung einer Einheit mehrere Prototypen gleichzeitig genutzt werden. Damit kann jeder Nutzer sein spezifisches Modell einer Einheit beliebig erweitern. Durch das gemeinsame Metamodell kann er gleichzeitig ihm vorher unbekannte Abbildungen erkunden und verstehen. Dabei ist die Definition der Merkmale von besonderer Wichtigkeit, da erst sie die Semantik der Merkmale zugänglich macht. Hier ist die Möglichkeit, jederzeit selbst Definitionen zu erzeugen und zu publizieren, die notwendige Voraussetzung für Flexi-

bilität. Dabei müssen alle Elemente global eindeutig identifizierbar sein, unabhängig davon, wer sie definiert hat. Dies wird durch die Nutzung von weltweit eindeutig zugewiesenen URLs als Namensraum-bildende Elemente erreicht. Innerhalb eines solchen Namensraumes ist der Inhaber der URL für die Eindeutigkeit verantwortlich. Diese Herangehensweise bringt den weiteren Vorteil, dass besagte URLs gleichzeitig als Abfrageadresse für Definitionen dienen können. Des Weiteren sind Registraturen und Verzeichnisdienste bereits vorhanden und haben sich bewährt. Ein Nachteil dieser Flexibilität der Definitionen ist die Möglichkeit der semantischen Doppelungen. Dem kann nur auf organisatorischer Ebene begegnet werden. Ferner ist davon auszugehen, dass sich nach einiger Zeit gewisse öffentlich zugängliche Datenbanken branchenspezifisch etablieren werden. Da Doppelungen trotzdem unvermeidbar sind, sind zur einfacheren Auflösung Querverweise zwischen synonymen Beschreibungen vorgesehen. Die operative Nutzung der erweiterbaren Einheitenabbilder wurde in Anwendungsfall eins erfolgreich geprüft. Sie wird vollständig vom Datenmodell hergestellt. Organisatorische Möglichkeiten der Verhinderung von Definitionsdoppelungen wurden dabei nicht evaluiert.

Die Spezifikation einer Dienstschnittstelle zum Austausch sowohl von Definitions- als auch von Einheiteninformationen unter Nutzung von Web-Technologien stellt Interoperabilität her und ermöglicht die Erkundung erzeugter Abbildungen ohne über das Metamodell hinausgehendes Vorwissen. Wie in Anwendungsfall zwei gezeigt wurde, kann jederzeit, von überall aus auf die benötigten Informationen zugegriffen werden. Dabei kann die Integrität und Vertraulichkeit der Daten durch bewährte Technologien sichergestellt werden. Ferner erlauben die WSDL-Dokumente und XML-Schemata der Spezifikation die automatisierte Generierung von Quelltext für Dienstnutzer und Dienstanbieter.

Die Abbildung von Zusammenhängen zwischen Eigenschaften derselben Einheit oder zwischen verschiedenen Einheiten ist ebenso im Modell verankert. Dabei wird jedoch keine formale Beschreibung der Zusammenhänge festgelegt. Dies ist der Vielfalt der möglichen Zusammenhänge geschuldet. Für verschiedene Anwendungen sind entsprechend verschiedene Formalismen zweckmäßig. Folglich wurde eine formale Beschreibung zwar vorgesehen, jedoch in ihrer Umsetzung offen gelassen, um die für den konkreten Fall bestmögliche Darstellung nutzen zu können. Die Mächtigkeit formaler Zusammenhangsmodelle wurde in den Anwendungsfällen drei und vier demonstriert. Sie lassen sich sowohl zur Abschätzung sonst nicht zugänglicher Informationen, als auch als Prädiktionsmittel einsetzen. Diese Möglichkeiten sind jedoch entscheidend von der konkreten Implementierung des Systems abhängig. Es sind ausschließlich die Mittel zur Auffindung beliebiger Informationen im Verwaltungssystem durch die Spezifikation gegeben. Die Art und Weise, diese Informationen in einem formalen Modell zu nutzen, wird erst durch die Implementierung festgelegt.

Bezogen auf die Nutzung der Entitätenverwaltung im Zusammenhang mit intelligenten Einheiten in einem Industrie 4.0-Umfeld seien noch einmal die verwendeten Identifikatoren angeführt. Diese erlauben auch die Adressierung der Einheiten selbst in einem Internet der Dinge (*self*-Identifikator). Dies setzt voraus, dass eine solche Einheit über eine eigene Adresse verfügt und selbstständig die spezifizierten Kommunikationsdienste umsetzt. Dadurch kann die Einheit selbst zum Zugriffspunkt ihrer Lebenszyklusabbildung werden. Auch kann sie auf weitere Verwaltungssysteme verweisen, die Informationen zu ihrem Lebenszyklus beinhalten. So können Zugriffswege zu relevanten Informationen mit der Einheit mitgeführt werden, die Informationen selbst unterliegen jedoch der Hoheit des verwaltenden Systems.

Schlussendlich sei in Erinnerung gerufen, dass die vorgestellte Art der Einheitenmodel-

lierung nicht auf technische Einheiten beschränkt ist. Es können beliebige Dinge physischer oder gedanklicher Natur sein, die mit beliebigen Eigenschaften und Zusammenhängen beschrieben werden. Dabei ist zu jeder einzelnen Aussage der Beschreibung über ihre Bedeutung hinaus klar, für welchen Zeitpunkt sie gültig ist und wer für sie verantwortlich ist. Damit können auch Ideen in ihrer Entwicklung verfolgt oder Entscheidungsprozesse nachvollzogen werden.

Anhang A

Spezielle Elemente des Metamodells

Dieser Anhang beschreibt die im Metamodell festgelegten Merkmal- und Aussagenbeschreibungen tabellarisch in Form von Name-Wert Paaren der Beschreibungsattribute. Als NamespaceURL (Namensraum-URL) wird implizit immer „metaModel“ angenommen. Die Beschreibungen sind in englischer Sprache ausgeführt.

A.1 Merkmalbeschreibungen

1. metaModel:hasPart

Name	Value (Description)
name	hasPart
identifier.ID	hasPart
definition	States a part (logical or physical subelement) of this entity.
remark	
documents	
synonyms	

2. metaModel:partOf

Name	Value (Description)
name	partOf
identifier.ID	partOf
definition	States an entity this entity is a part (logical or physical subelement) of.
remark	
documents	
synonyms	

3. metaModel:refersTo

Name	Value (Description)
name	refersTo
identifier.ID	refersTo
definition	States the physical entity this logical entity refers to (or the other way around).
remark	
documents	
synonyms	

4. metaModel:hasFormalModel

Name	Beschreibung
name	hasFormalModel
identifier.ID	hasFormalModel
definition	Refers to a formal model for a RelationModel.
remark	
documents	
synonyms	

5. metaModel:hasRelationModel

Name	Beschreibung
name	hasRelationModel
identifier.ID	hasRelationModel
definition	Refers to a RelationModel for a RelationDescription.
remark	
documents	
synonyms	

6. metaModel:concernsEvent

Name	Beschreibung
name	concernsEvent
identifier.ID	concernsEvent
definition	States that an entity concerns (caused, is influenced by or similar) an event.
remark	
documents	
synonyms	

7. metaModel:otherRepresentation

Name	Value (Description)
name	otherRepresentation
identifier.ID	otherRepresentation
definition	References representations of this unit in different management systems. This reference is revisionless.
remark	An entity may be managed in more than one system at the same time.
documents	
synonyms	

A.2 Aussagebeschreibungen

1. metaModel:definedIdentificator

Name	Beschreibung
name	definedIdentificator
identifier.ID	definedIdentificator
definition	States that an Identificator is defined by the user or system. Always carries an Identificator as value.
remark	Mostly used to state references to models or other instances in the lifecycle management system.
documents	
synonyms	

1. metaModel:dataURL

Name	Beschreibung
name	dataURL
identifier.ID	dataURL
definition	States the URL over which the data corresponding to a property can be read out.
remark	
documents	
synonyms	

1. metaModel:dataInterfaceType

Name	Beschreibung
name	dataInterfaceType
identifier.ID	dataInterfaceType
definition	States the interface type to be used when data is retrieved from a database referenced by a *metamodel:dataURL* statement.
remark	
documents	
synonyms	

A.3 Identifikatoren

1. metaModel:self

   Bei Aufruf eines Dienstes einer Verwaltungsschale besagt dieser Identifikator, dass auf die von der Verwaltungsschale verwaltete Einheit Bezug genommen wird.

Anhang B

WSDL-Beschreibung der Dienstschnittstelle

Die Beschreibung der Dienstsätze erfolgt im Folgenden in WSDL 2.0. Dabei wird nur der abstrakte Teil, also Typen und Interfaces, und ein Standard-Binding beschrieben. Dieses Binding ist HTTP-basiert und muss von allen Dienstanbietern unterstützt werden. Aus Gründen des Lesbarkeit wurden weitere Zeilenumbrüche in die unten eingefügten WSDL-Dateien eingebracht. Diese sind mit dem Zeichen ↪ markiert. Die Dateien selbst können unter der im *targetNamespace* angegebenen URL heruntergeladen werden.

B.1 Gemeinsame Definitionen

Das folgende Listing enthält die gemeinsamen Typdefinitionen.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xs:schema targetNamespace="http://www.levertz.xyz/elm/commons" xmlns:lmg="
       ↪ http://www.levertz.xyz/elm/commons" xmlns:xs="http://www.w3.org/2001/
       ↪ XMLSchema">
3    <xs:complexType name="identificator">
4      <xs:annotation>
5        <xs:documentation>identificator type</xs:documentation>
6      </xs:annotation>
7      <xs:sequence>
8        <xs:element name="nameSpace" type="xs:anyURI">
9          <xs:annotation>
10           <xs:documentation>Globally unique URL for name space. The party
                 ↪ responsible for this namespace has to own and maintain the
                 ↪ URL.</xs:documentation>
11         </xs:annotation>
12       </xs:element>
13       <xs:element name="ID" type="xs:anyURI">
14         <xs:annotation>
15           <xs:documentation>Unique Idntifier within the namespace. The party
                 ↪  responsible for the namespace has to ensure uniqueness of
                 ↪ the IDs.</xs:documentation>
16         </xs:annotation>
17       </xs:element>
18     </xs:sequence>
19   </xs:complexType>
20   <xs:complexType name="reference">
21     <xs:annotation>
22       <xs:documentation>Reference to a model or similar</xs:documentation>
```

```
23      </xs:annotation>
24      <xs:complexContent>
25        <xs:extension base="lmg:identificator">
26          <xs:attribute name="typeNamespace" type="xs:anyURI" use="required"/>
27          <xs:attribute name="typeID" type="xs:anyURI" use="required"/>
28        </xs:extension>
29      </xs:complexContent>
30    </xs:complexType>
31    <xs:complexType name="status">
32      <xs:sequence>
33        <xs:element name="statusCode" type="xs:int">
34          <xs:annotation>
35            <xs:documentation>Statuscode of operration. 0 is OK. </
                ↪ xs:documentation>
36          </xs:annotation>
37        </xs:element>
38        <xs:element name="statusString" type="xs:string" nillable="true"
            ↪ minOccurs="0">
39          <xs:annotation>
40            <xs:documentation>Human readable status description. Optional, can
                ↪  be left empty.</xs:documentation>
41          </xs:annotation>
42        </xs:element>
43      </xs:sequence>
44    </xs:complexType>
45    <xs:complexType name="statusOnlyResponse">
46      <xs:annotation>
47        <xs:documentation>Basic response type</xs:documentation>
48      </xs:annotation>
49      <xs:sequence>
50        <xs:element name="status" type="lmg:status"/>
51      </xs:sequence>
52    </xs:complexType>
53    <xs:complexType name="genericResponse">
54      <xs:annotation>
55        <xs:documentation>Basic response type</xs:documentation>
56      </xs:annotation>
57      <xs:complexContent>
58        <xs:extension base="lmg:statusOnlyResponse">
59          <xs:sequence>
60            <xs:element name="identificator" type="lmg:identificator" nillable
                ↪ ="true">
61              <xs:annotation>
62                <xs:documentation>Object identificator. Which object is meant
                    ↪ is specified in the service description.</
                    ↪ xs:documentation>
63              </xs:annotation>
64            </xs:element>
65          </xs:sequence>
66        </xs:extension>
67      </xs:complexContent>
68    </xs:complexType>
69    <xs:complexType name="idFilter">
70      <xs:annotation>
71        <xs:documentation>Filter for Identificators (regExp based)</
            ↪ xs:documentation>
```

```
72      </xs:annotation>
73      <xs:sequence>
74        <xs:element name="nameSpaceExp" type="xs:string">
75          <xs:annotation>
76            <xs:documentation>RegExp to filter against the nameSpace part.</
                  ↪ xs:documentation>
77          </xs:annotation>
78        </xs:element>
79        <xs:element name="idExp" type="xs:string">
80          <xs:annotation>
81            <xs:documentation>RegExp to filter against local id part.</
                  ↪ xs:documentation>
82          </xs:annotation>
83        </xs:element>
84      </xs:sequence>
85    </xs:complexType>
86    <xs:complexType name="rolePath">
87      <xs:annotation>
88        <xs:documentation>Path to find something to fill a role.</
              ↪ xs:documentation>
89      </xs:annotation>
90      <xs:sequence>
91        <xs:annotation>
92          <xs:documentation>The sequenze is followed through. The last element
                ↪  has to be a statementType. All propertyTypes and
                ↪ statementTypes before have to belong to the metaModel
                ↪ namespace. The first match on a type is used.</
                ↪ xs:documentation>
93        </xs:annotation>
94        <xs:choice maxOccurs="unbounded">
95          <xs:element name="entityType" type="lmg:identificator"/>
96          <xs:element name="propertyType" type="lmg:identificator"/>
97          <xs:element name="statementType" type="lmg:identificator"/>
98        </xs:choice>
99      </xs:sequence>
100   </xs:complexType>
101   <xs:complexType name="roleDescription">
102     <xs:annotation>
103       <xs:documentation>Structure describing a role.</xs:documentation>
104     </xs:annotation>
105     <xs:sequence>
106       <xs:element name="name" type="xs:NMTOKEN"/>
107       <xs:element name="path" type="lmg:rolePath"/>
108       <xs:element name="treatValueAsReference" type="xs:boolean"/>
109     </xs:sequence>
110   </xs:complexType>
111 </xs:schema>
```

B.2 Dienste für Semantikdefinitionen

Das folgende Listing enthält die Nachrichten-Typen für die Dienste zum Semantikaustausch.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```
<xsd:schema targetNamespace="http://www.levertz.xyz/elm/semantics" xmlns:xsd
    ↪ ="http://www.w3.org/2001/XMLSchema" xmlns:lms="http://www.levertz.xyz/
    ↪ elm/semantics" xmlns:lmg="http://www.levertz.xyz/elm/commons">
  <xsd:import namespace="http://www.levertz.xyz/elm/commons" schemaLocation=
      ↪ "C:\Users\lars.PLT\Diss\Specs\commons.xsd"/>
  <xsd:complexType name="setAttributeParams">
    <xsd:annotation>
      <xsd:documentation>Parameters for the setAttribute operation</
          ↪ xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="parentObject" type="lmg:identificator">
        <xsd:annotation>
          <xsd:documentation>Object to which the attribute is set / added</
              ↪ xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="name" type="xsd:NMTOKEN">
        <xsd:annotation>
          <xsd:documentation>Attribute name</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="value" type="xsd:anyType">
        <xsd:annotation>
          <xsd:documentation>Attribute value</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="createNonExisting" type="xsd:boolean" default="
          ↪ false" minOccurs="0"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="detachModelParams">
    <xsd:annotation>
      <xsd:documentation>Parameters for the detachModel operation</
          ↪ xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
      <xsd:element name="parentNS" type="xsd:anyURI">
        <xsd:annotation>
          <xsd:documentation>Globally unique URL for name space of the
              ↪ parent object. The party responsible for this namespace has
              ↪ to own and maintain the URL.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="parentID" type="xsd:anyURI">
        <xsd:annotation>
          <xsd:documentation>Unique Idntifier of the parent object within
              ↪ the namespace. The party responsible for the namespace has
              ↪ to ensure uniqueness of the IDs.</xsd:documentation>
        </xsd:annotation>
      </xsd:element>
      <xsd:element name="childNS" type="xsd:anyURI">
        <xsd:annotation>
          <xsd:documentation>Globally unique URL for namespace of the cild
              ↪ object. The party responsible for this namespace has to own
              ↪ and maintain the URL.</xsd:documentation>
```

```
45          </xsd:annotation>
46        </xsd:element>
47        <xsd:element name="childID" type="xsd:anyURI">
48          <xsd:annotation>
49            <xsd:documentation>Unique Idntifier for the cild object within the
                  ↪  namespace. The party responsible for the namespace has to
                  ↪ ensure uniqueness of the IDs.</xsd:documentation>
50          </xsd:annotation>
51        </xsd:element>
52      </xsd:sequence>
53    </xsd:complexType>
54    <xsd:complexType name="createDefinitionParams">
55      <xsd:annotation>
56        <xsd:documentation>Parameters for the createPropertyDefinition and
              ↪ createStatementDefinition operations</xsd:documentation>
57      </xsd:annotation>
58      <xsd:sequence>
59        <xsd:element name="name" type="xsd:NMTOKEN">
60          <xsd:annotation>
61            <xsd:documentation>Human readable name. Not unique.</
                  ↪ xsd:documentation>
62          </xsd:annotation>
63        </xsd:element>
64        <xsd:element name="semanticDefinition" type="xsd:string"/>
65        <xsd:element name="remark" type="xsd:string" minOccurs="0"/>
66        <xsd:element name="documentRef" type="xsd:anyURI" minOccurs="0">
67          <xsd:annotation>
68            <xsd:documentation>Reference to a spcification document.</
                  ↪ xsd:documentation>
69          </xsd:annotation>
70        </xsd:element>
71        <xsd:element name="knownSynonym" type="lmg:identificator" minOccurs="0
              ↪ " maxOccurs="unbounded"/>
72      </xsd:sequence>
73    </xsd:complexType>
74    <xsd:complexType name="getDefinitionParams">
75      <xsd:annotation>
76        <xsd:documentation>Parameters for the getPropertyDefinition and
              ↪ getStatementDefinition operations</xsd:documentation>
77      </xsd:annotation>
78      <xsd:sequence>
79        <xsd:element name="identificatorNS" type="xsd:anyURI">
80          <xsd:annotation>
81            <xsd:documentation>Globally unique URL for name space. The party
                  ↪ responsible for this namespace has to own and maintain the
                  ↪ URL.</xsd:documentation>
82          </xsd:annotation>
83        </xsd:element>
84        <xsd:element name="identificatorID" type="xsd:anyURI">
85          <xsd:annotation>
86            <xsd:documentation>Unique Idntifier within the namespace. The
                  ↪ party responsible for the namespace has to ensure uniqueness
                  ↪  of the IDs.</xsd:documentation>
87          </xsd:annotation>
88        </xsd:element>
89        <xsd:element name="getSystemSpecificAttributes" type="xsd:boolean"
```

```
        ↪ default="false" minOccurs="0"/>
90    </xsd:sequence>
91  </xsd:complexType>
92  <xsd:complexType name="getDefinitionResponse">
93    <xsd:annotation>
94      <xsd:documentation>Response message of getPropertyDefinition and
            ↪ getStatemnentDefinition operations</xsd:documentation>
95    </xsd:annotation>
96    <xsd:complexContent>
97      <xsd:extension base="lmg:genericResponse">
98        <xsd:sequence>
99          <xsd:element name="attribute" minOccurs="0" maxOccurs="unbounded">
100           <xsd:complexType>
101             <xsd:complexContent>
102               <xsd:extension base="xsd:anyType">
103                 <xsd:attribute name="name" type="xsd:NMTOKEN" use="
                        ↪ required"/>
104               </xsd:extension>
105             </xsd:complexContent>
106           </xsd:complexType>
107         </xsd:element>
108       </xsd:sequence>
109     </xsd:extension>
110   </xsd:complexContent>
111 </xsd:complexType>
112 <xsd:complexType name="createRelationModelParams">
113   <xsd:annotation>
114     <xsd:documentation>Parameters for the createRelationmodel operation</
          ↪ xsd:documentation>
115   </xsd:annotation>
116   <xsd:sequence>
117     <xsd:element name="description" type="lms:description"/>
118     <xsd:element name="sourceRole" type="lmg:roleDescription" minOccurs="0
          ↪ " maxOccurs="unbounded"/>
119     <xsd:element name="targetRole" type="lmg:roleDescription" minOccurs="0
          ↪ " maxOccurs="unbounded"/>
120   </xsd:sequence>
121 </xsd:complexType>
122 <xsd:complexType name="addFormalModelToRelationModelParams">
123   <xsd:annotation>
124     <xsd:documentation>Parameters for the addFormalModelToRelationModel
          ↪ operation</xsd:documentation>
125   </xsd:annotation>
126   <xsd:sequence>
127     <xsd:element name="identificator" type="lmg:identificator">
128       <xsd:annotation>
129         <xsd:documentation>Identificator of the relationModel to add
              ↪ formalModel to.</xsd:documentation>
130       </xsd:annotation>
131     </xsd:element>
132     <xsd:element name="model" type="xsd:anyType">
133       <xsd:annotation>
134         <xsd:documentation>Formal model in system specific representation<
              ↪ /xsd:documentation>
135       </xsd:annotation>
136     </xsd:element>
```

```
137     </xsd:sequence>
138   </xsd:complexType>
139   <xsd:complexType name="addFormalModelToRelationModelResponse">
140     <xsd:annotation>
141       <xsd:documentation>Response message for the
              ↪ addFormalModelToRelationmodel operation.</xsd:documentation>
142     </xsd:annotation>
143     <xsd:complexContent>
144       <xsd:extension base="lmg:statusOnlyResponse">
145         <xsd:sequence>
146           <xsd:element name="formalModel" type="lmg:identificator"/>
147           <xsd:element name="relationModel" type="lmg:identificator"/>
148         </xsd:sequence>
149       </xsd:extension>
150     </xsd:complexContent>
151   </xsd:complexType>
152   <xsd:complexType name="readParams">
153     <xsd:annotation>
154       <xsd:documentation>Parameters for the readRelationModel and
              ↪ readRelationDescription operations</xsd:documentation>
155     </xsd:annotation>
156     <xsd:sequence>
157       <xsd:element name="identificatorNS" type="xsd:anyURI">
158         <xsd:annotation>
159           <xsd:documentation>Globally unique URL for name space. The party
                  ↪ responsible for this namespace has to own and maintain the
                  ↪ URL.</xsd:documentation>
160         </xsd:annotation>
161       </xsd:element>
162       <xsd:element name="identificatorID" type="xsd:anyURI">
163         <xsd:annotation>
164           <xsd:documentation>Unique Idntifier within the namespace. The
                  ↪ party responsible for the namespace has to ensure uniqueness
                  ↪  of the IDs.</xsd:documentation>
165         </xsd:annotation>
166       </xsd:element>
167       <xsd:element name="getSystemAttributes" type="xsd:boolean" default="
              ↪ false" minOccurs="0">
168         <xsd:annotation>
169           <xsd:documentation>Specifies wether system specific attrributes
                  ↪ should be read out.</xsd:documentation>
170         </xsd:annotation>
171       </xsd:element>
172       <xsd:element name="getCompleteSubmodels" type="xsd:boolean" default="
              ↪ false" minOccurs="0"/>
173     </xsd:sequence>
174   </xsd:complexType>
175   <xsd:complexType name="readRelationModelResponse">
176     <xsd:annotation>
177       <xsd:documentation>Response message for the readRelationModel
              ↪ operation</xsd:documentation>
178     </xsd:annotation>
179     <xsd:complexContent>
180       <xsd:extension base="lmg:statusOnlyResponse">
181         <xsd:sequence>
182           <xsd:element name="description" type="lms:description"/>
```

```
183           <xsd:element name="sourceRole" type="lmg:roleDescription"
                  ↪ minOccurs="0" maxOccurs="unbounded"/>
184           <xsd:element name="targetRole" type="lmg:roleDescription"
                  ↪ minOccurs="0" maxOccurs="unbounded"/>
185           <xsd:choice minOccurs="0">
186             <xsd:element name="model" type="xsd:anyType" maxOccurs="
                    ↪ unbounded"/>
187             <xsd:element name="modelReference" type="lmg:reference"
                    ↪ maxOccurs="unbounded"/>
188           </xsd:choice>
189           <xsd:element name="attribute" minOccurs="0" maxOccurs="unbounded">
190             <xsd:complexType>
191               <xsd:complexContent>
192                 <xsd:extension base="xsd:anyType">
193                   <xsd:attribute name="name" type="xsd:NMTOKEN" use="
                          ↪ required"/>
194                 </xsd:extension>
195               </xsd:complexContent>
196             </xsd:complexType>
197           </xsd:element>
198         </xsd:sequence>
199       </xsd:extension>
200     </xsd:complexContent>
201   </xsd:complexType>
202   <xsd:complexType name="description">
203     <xsd:annotation>
204       <xsd:documentation>Description type e.g. for relationDescriptions.
              ↪ Serves as parameter type for the CreateRelationDescription
              ↪ operation and inside the readRelationDescription response
              ↪ message.</xsd:documentation>
205     </xsd:annotation>
206     <xsd:sequence>
207       <xsd:element name="name" type="xsd:NMTOKEN">
208         <xsd:annotation>
209           <xsd:documentation>Human readable name. Not unique.</
                  ↪ xsd:documentation>
210         </xsd:annotation>
211       </xsd:element>
212       <xsd:element name="description" type="xsd:string"/>
213       <xsd:element name="remark" type="xsd:string" minOccurs="0"/>
214       <xsd:element name="documentRef" type="xsd:anyURI" minOccurs="0">
215         <xsd:annotation>
216           <xsd:documentation>Reference to a spcification document.</
                  ↪ xsd:documentation>
217         </xsd:annotation>
218       </xsd:element>
219     </xsd:sequence>
220   </xsd:complexType>
221   <xsd:complexType name="readRelationDescriptionResponse">
222     <xsd:annotation>
223       <xsd:documentation>Resonse message for the readRelationDescription
              ↪ operation</xsd:documentation>
224     </xsd:annotation>
225     <xsd:complexContent>
226       <xsd:extension base="lmg:statusOnlyResponse">
227         <xsd:sequence>
```

```
228           <xsd:element name="description" type="lms:description"/>
229           <xsd:choice minOccurs="0">
230             <xsd:element name="relationModel" type="
                  ↪ lms:readRelationModelResponse" maxOccurs="unbounded"/>
231             <xsd:element name="relationModelReference" type="lmg:reference"
                  ↪ maxOccurs="unbounded"/>
232           </xsd:choice>
233           <xsd:element name="attribute" minOccurs="0" maxOccurs="unbounded">
234             <xsd:complexType>
235               <xsd:complexContent>
236                 <xsd:extension base="xsd:anyType">
237                   <xsd:attribute name="name" type="xsd:NMTOKEN" use="
                        ↪ required"/>
238                 </xsd:extension>
239               </xsd:complexContent>
240             </xsd:complexType>
241           </xsd:element>
242         </xsd:sequence>
243       </xsd:extension>
244     </xsd:complexContent>
245   </xsd:complexType>
246   <xsd:complexType name="attachRelationModelToDescriptionParams">
247     <xsd:annotation>
248       <xsd:documentation>Parameters for the attachRelationmodelToDescription
            ↪ operation</xsd:documentation>
249     </xsd:annotation>
250     <xsd:sequence>
251       <xsd:element name="description" type="lmg:identificator"/>
252       <xsd:element name="relationModel" type="lmg:identificator"/>
253     </xsd:sequence>
254   </xsd:complexType>
255   <!-- Message elements -->
256   <xsd:element name="setAttributeReq" type="lms:setAttributeParams"/>
257   <xsd:element name="setAttributeRes" type="lmg:genericResponse"/>
258   <xsd:element name="detachModelReq" type="lms:detachModelParams"/>
259   <xsd:element name="detachModelRes" type="lmg:genericResponse"/>
260   <xsd:element name="createPropertyDefinitionReq" type="
        ↪ lms:createDefinitionParams"/>
261   <xsd:element name="createPropertyDefinitionRes" type="lmg:genericResponse"
        ↪ />
262   <xsd:element name="getPropertyDefinitionReq" type="lms:getDefinitionParams
        ↪ "/>
263   <xsd:element name="getPropertyDefinitionRes" type="
        ↪ lms:getDefinitionResponse"/>
264   <xsd:element name="createStatementDefinitionReq" type="
        ↪ lms:createDefinitionParams"/>
265   <xsd:element name="createStatementDefinitionRes" type="lmg:genericResponse
        ↪ "/>
266   <xsd:element name="getStatementDefinitionReq" type="
        ↪ lms:getDefinitionParams"/>
267   <xsd:element name="getStatementDefinitionRes" type="
        ↪ lms:getDefinitionResponse"/>
268   <xsd:element name="createRelationModelReq" type="
        ↪ lms:createRelationModelParams"/>
269   <xsd:element name="createRelationModelRes" type="lmg:genericResponse"/>
270   <xsd:element name="addFormalModelToRelationModelReq" type="
```

```
        ↪ lms:addFormalModelToRelationModelParams"/>
271   <xsd:element name="addFormalModelToRelationModelRes" type="
        ↪ lms:addFormalModelToRelationModelResponse"/>
272   <xsd:element name="readRelationModelReq" type="lms:readParams"/>
273   <xsd:element name="readRelationModelRes" type="
        ↪ lms:readRelationModelResponse"/>
274   <xsd:element name="createRelationDescriptionReq" type="lms:description"/>
275   <xsd:element name="createRelationDescriptionRes" type="lmg:genericResponse
        ↪ "/>
276   <xsd:element name="attachRelationModelToDescriptionReq" type="
        ↪ lms:attachRelationModelToDescriptionParams"/>
277   <xsd:element name="attachRelationModelToDescriptionRes" type="
        ↪ lmg:genericResponse"/>
278   <xsd:element name="readRelationDescriptionReq" type="lms:readParams"/>
279   <xsd:element name="readRelationDescriptionRes" type="
        ↪ lms:readRelationDescriptionResponse"/>
280   <xsd:element name="createProcessDescriptionReq" type="lms:description"/>
281   <xsd:element name="createProcessDescriptionRes" type="lmg:genericResponse"
        ↪ />
282   <xsd:element name="attachRelationModelToProcessDescriptionReq" type="
        ↪ lms:attachRelationModelToDescriptionParams"/>
283   <xsd:element name="attachRelationModelToProcessDescriptionRes" type="
        ↪ lmg:genericResponse"/>
284   <xsd:element name="readProcessDescriptionReq" type="lms:readParams"/>
285   <xsd:element name="readProcessDescriptionRes" type="
        ↪ lms:readRelationDescriptionResponse"/>
286 </xsd:schema>
```

Das folgende Listing enthält die auf den vorab dargestellten Definitionen aufbauende Dienstbeschreibung.

```
1  <?xml version="1.0"?>
2  <wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl" xmlns:whttp="http:
       ↪ //www.w3.org/ns/wsdl/http" xmlns:lms="http://www.levertz.xyz/elm/
       ↪ semantics" targetNamespace="http://www.levertz.xyz/elm/semantics">
3    <wsdl:documentation>
4      This file defines the Services for handling and exchange
5      of semantic definitions. The services are grouped to
6      support either propertyDefinitions, statementDefinitions,
7      relationModels or relationDescriptions.
8    </wsdl:documentation>
9    <!--
10     type definitions for messages and their elements
11   -->
12   <wsdl:types>
13     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace
         ↪ ="http://www.levertz.xyz/elm/semantics">
14       <xsd:include schemaLocation="C:\Users\lars.PLT\Diss\Specs\
           ↪ semanticsMessages.xsd"/>
15     </xsd:schema>
16   </wsdl:types>
17   <!--
18   The generic Interface is the base interface for all
19   further interfaces. It provides a generic set-operation.
20   -->
21   <wsdl:interface name="genericOperations">
```

```
22      <wsdl:operation name="setAttribute" pattern="http://www.w3.org/ns/wsdl/
            ↪ in-out">
23        <wsdl:documentation>
24        This operation set an arbitrary attribute of an Object.
25      The object's identificator, the attribute's name and value
26      are given as parameters. The value can be of any type. As
27      a new Revision of the containing object may be created
28      (depending on the service provider's configuration), the
29      identificator of the (new) object is returne together with
30      the operation status.
31      </wsdl:documentation>
32        <wsdl:input messageLabel="In" element="lms:setAttributeReq"/>
33        <wsdl:output messageLabel="Out" element="lms:setAttributeRes"/>
34      </wsdl:operation>
35      <wsdl:documentation>
36      Operations all service sets have in common.
37    </wsdl:documentation>
38    </wsdl:interface>
39    <!--
40    The relationBase Interface is the base interface for interfaces
41    handling relationModels and relationDescriptions.
42    It provides a generic detach-operation.
43    -->
44    <wsdl:interface name="relationBase" extends="lms:genericOperations">
45      <wsdl:operation name="detachModel" pattern="http://www.w3.org/ns/wsdl/in
            ↪ -out" style="http://www.w3.org/ns/wsdl/style/iri">
46        <wsdl:documentation/>
47        <!-- insert documentation here -->
48        <wsdl:input messageLabel="In" element="lms:detachModelReq"/>
49        <wsdl:output messageLabel="Out" element="lms:detachModelRes"/>
50      </wsdl:operation>
51      <wsdl:documentation/>
52      <!-- insert documentation here -->
53    </wsdl:interface>
54    <!--
55    The propertyDefinitionService interface provides operations
56    to handle property definitions. It extends the genericOperations
57    interface.
58    -->
59    <wsdl:interface name="propertyDefinitionService" extends="
          ↪ lms:genericOperations">
60      <wsdl:operation name="createPropertyDefinition" pattern="http://www.w3.
            ↪ org/ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
61        <wsdl:documentation/>
62        <!-- insert documentation here -->
63        <wsdl:input messageLabel="In" element="lms:createPropertyDefinitionReq
              ↪ "/>
64        <wsdl:output messageLabel="Out" element="
              ↪ lms:createPropertyDefinitionRes"/>
65      </wsdl:operation>
66      <wsdl:operation name="getPropertyDefinition" pattern="http://www.w3.org/
            ↪ ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
67        <wsdl:documentation/>
68        <!-- insert documentation here -->
69        <wsdl:input messageLabel="In" element="lms:getPropertyDefinitionReq"/>
```

```
70      <wsdl:output messageLabel="Out" element="lms:getPropertyDefinitionRes
            ↪ "/>
71    </wsdl:operation>
72    <wsdl:documentation/>
73    <!-- insert documentation here -->
74  </wsdl:interface>
75  <!--
76  The statementDefinitionService interface provides operations
77  to handle statement definitions. It extends the genericOperations
78  interface.
79  -->
80  <wsdl:interface name="statementDefinitionService" extends="
        ↪ lms:genericOperations">
81    <wsdl:operation name="createStatementDefinition" pattern="http://www.w3.
          ↪ org/ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
82      <wsdl:documentation/>
83      <!-- insert documentation here -->
84      <wsdl:input messageLabel="In" element="
            ↪ lms:createStatementDefinitionReq"/>
85      <wsdl:output messageLabel="Out" element="
            ↪ lms:createStatementDefinitionRes"/>
86    </wsdl:operation>
87    <wsdl:operation name="getStatementDefinition" pattern="http://www.w3.org
          ↪ /ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
88      <wsdl:documentation/>
89      <!-- insert documentation here -->
90      <wsdl:input messageLabel="In" element="lms:getStatementDefinitionReq
            ↪ "/>
91      <wsdl:output messageLabel="Out" element="lms:getStatementDefinitionRes
            ↪ "/>
92    </wsdl:operation>
93    <wsdl:documentation/>
94    <!-- insert documentation here -->
95  </wsdl:interface>
96  <!--
97  The relationModelService interface provides operations
98  to handle relationModels. It extends the relationBase
99  interface.
100 -->
101 <wsdl:interface name="relationModelService" extends="lms:relationBase">
102   <wsdl:operation name="createRelationModel" pattern="http://www.w3.org/ns
          ↪ /wsdl/in-out">
103     <wsdl:documentation/>
104     <!-- insert documentation here -->
105     <wsdl:input messageLabel="In" element="lms:createRelationModelReq"/>
106     <wsdl:output messageLabel="Out" element="lms:createRelationModelRes"/>
107   </wsdl:operation>
108   <wsdl:operation name="addFormalModelToRelationModel" pattern="http://www
          ↪ .w3.org/ns/wsdl/in-out">
109     <wsdl:documentation/>
110     <!-- insert documentation here -->
111     <wsdl:input messageLabel="In" element="
            ↪ lms:addFormalModelToRelationModelReq"/>
112     <wsdl:output messageLabel="Out" element="
            ↪ lms:addFormalModelToRelationModelRes"/>
113   </wsdl:operation>
```

```
114     <wsdl:operation name="readRelationModel" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
115       <wsdl:documentation/>
116       <!-- insert documentation here -->
117       <wsdl:input messageLabel="In" element="lms:readRelationModelReq"/>
118       <wsdl:output messageLabel="Out" element="lms:readRelationModelRes"/>
119     </wsdl:operation>
120     <wsdl:documentation/>
121     <!-- insert documentation here -->
122   </wsdl:interface>
123   <!--
124   The relationDescriptionService interface provides operations
125   to handle relationDescriptions. It extends the relationBase
126   interface.
127   -->
128   <wsdl:interface name="relationDescriptionService" extends="
          ↪ lms:relationBase">
129     <wsdl:operation name="createRelationDescription" pattern="http://www.w3.
            ↪ org/ns/wsdl/in-out">
130       <wsdl:documentation/>
131       <!-- insert documentation here -->
132       <wsdl:input messageLabel="In" element="
              ↪ lms:createRelationDescriptionReq"/>
133       <wsdl:output messageLabel="Out" element="
              ↪ lms:createRelationDescriptionRes"/>
134     </wsdl:operation>
135     <wsdl:operation name="attachRelationModelToDescription" pattern="http://
            ↪ www.w3.org/ns/wsdl/in-out">
136       <wsdl:documentation/>
137       <!-- insert documentation here -->
138       <wsdl:input messageLabel="In" element="
              ↪ lms:attachRelationModelToDescriptionReq"/>
139       <wsdl:output messageLabel="Out" element="
              ↪ lms:attachRelationModelToDescriptionRes"/>
140     </wsdl:operation>
141     <wsdl:operation name="readRelationDescription" pattern="http://www.w3.
            ↪ org/ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
142       <wsdl:documentation/>
143       <!-- insert documentation here -->
144       <wsdl:input messageLabel="In" element="lms:readRelationDescriptionReq
              ↪ "/>
145       <wsdl:output messageLabel="Out" element="
              ↪ lms:readRelationDescriptionRes"/>
146     </wsdl:operation>
147     <wsdl:documentation/>
148     <!-- insert documentation here -->
149   </wsdl:interface>
150   <!--
151   The processDescriptionService interface provides operations
152   to handle relationDescriptions. It extends the relationBase
153   interface.
154   -->
155   <wsdl:interface name="processDescriptionService" extends="lms:relationBase
          ↪ ">
156     <wsdl:operation name="createProcessDescription" pattern="http://www.w3.
            ↪ org/ns/wsdl/in-out">
```

```
157       <wsdl:documentation/>
158       <!-- insert documentation here -->
159       <wsdl:input messageLabel="In" element="lms:createProcessDescriptionReq
              ↪ "/>
160       <wsdl:output messageLabel="Out" element="
              ↪ lms:createProcessDescriptionRes"/>
161     </wsdl:operation>
162     <wsdl:operation name="attachRelationModelToProcessDescription" pattern="
            ↪ http://www.w3.org/ns/wsdl/in-out">
163       <wsdl:documentation/>
164       <!-- insert documentation here -->
165       <wsdl:input messageLabel="In" element="
              ↪ lms:attachRelationModelToProcessDescriptionReq"/>
166       <wsdl:output messageLabel="Out" element="
              ↪ lms:attachRelationModelToProcessDescriptionRes"/>
167     </wsdl:operation>
168     <wsdl:operation name="readProcessDescription" pattern="http://www.w3.org
            ↪ /ns/wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
169       <wsdl:documentation/>
170       <!-- insert documentation here -->
171       <wsdl:input messageLabel="In" element="lms:readProcessDescriptionReq
              ↪ "/>
172       <wsdl:output messageLabel="Out" element="lms:readProcessDescriptionRes
              ↪ "/>
173     </wsdl:operation>
174     <wsdl:documentation/>
175     <!-- insert documentation here -->
176   </wsdl:interface>
177
178   <wsdl:binding name="defaultPropertyDefinitionServiceBinding" interface="
          ↪ lms:propertyDefinitionServices" type="http://www.w3.org/ns/wsdl/http
          ↪ ">
179     <wsdl:operation ref="lms:createPropertyDefinition" whttp:method="PUT"/>
180     <wsdl:operation ref="lms:getPropertyDefinition" whttp:method="GET"
            ↪ whttp:location="getPropertyDefinition"/>
181     <wsdl:operation ref="lms:setAttribute" whttp:method="POST"/>
182   </wsdl:binding>
183   <wsdl:binding name="defaultStatementDefinitionServiceBinding" interface="
          ↪ lms:statementDefinitionServices" type="http://www.w3.org/ns/wsdl/
          ↪ http">
184     <wsdl:operation ref="lms:createStatementDefinition" whttp:method="PUT"/>
185     <wsdl:operation ref="lms:getStatementDefinition" whttp:method="GET"
            ↪ whttp:location="getStatementDefinition"/>
186     <wsdl:operation ref="lms:setAttribute" whttp:method="POST"/>
187   </wsdl:binding>
188   <wsdl:binding name="defaultRelationModelServiceBinding" interface="
          ↪ lms:relationModelServices" type="http://www.w3.org/ns/wsdl/http">
189     <wsdl:operation ref="lms:createRelationModel" whttp:method="PUT"/>
190     <wsdl:operation ref="lms:addFormalModelToRelationModel" whttp:method="
            ↪ POST"/>
191     <wsdl:operation ref="lms:readRelationModel" whttp:method="GET"
            ↪ whttp:location="readRelationModel"/>
192     <wsdl:operation ref="lms:detachModel" whttp:method="DELETE"
            ↪ whttp:location="detachModel"/>
193     <wsdl:operation ref="lms:setAttribute" whttp:method="POST"/>
194   </wsdl:binding>
```

```
195   <wsdl:binding name="defaultRelationDescriptionServiceBinding" interface="
          ↪ lms:relationDescriptionServices" type="http://www.w3.org/ns/wsdl/
          ↪ http">
196     <wsdl:operation ref="lms:createRelationDescription" whttp:method="PUT"/>
197     <wsdl:operation ref="lms:attachRelationModelToDescription" whttp:method
            ↪ ="POST"/>
198     <wsdl:operation ref="lms:readRelationDescription" whttp:method="GET"
            ↪ whttp:location="readRelationDescription"/>
199     <wsdl:operation ref="lms:detachModel" whttp:method="DELETE"
            ↪ whttp:location="detachModel"/>
200     <wsdl:operation ref="lms:setAttribute" whttp:method="POST"/>
201   </wsdl:binding>
202   <wsdl:binding name="defaultProcessDescriptionServiceBinding" interface="
          ↪ lms:relationDescriptionServices" type="http://www.w3.org/ns/wsdl/
          ↪ http">
203     <wsdl:operation ref="lms:createProcessDescription" whttp:method="PUT"/>
204     <wsdl:operation ref="lms:attachRelationModelToProcessDescription"
            ↪ whttp:method="POST"/>
205     <wsdl:operation ref="lms:readProcessDescription" whttp:method="GET"
            ↪ whttp:location="readProcessDescription"/>
206     <wsdl:operation ref="lms:detachModel" whttp:method="DELETE"
            ↪ whttp:location="detachModel"/>
207     <wsdl:operation ref="lms:setAttribute" whttp:method="POST"/>
208   </wsdl:binding>
209 </wsdl:description>
```

B.3 Dienste für Merkmaldatenaustausch

Das folgende Listing enthält die Nachrichten-Typen für die Dienste zum Merkmaldatenaustausch.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="http://www.levertz.xyz/elm/properties"
       ↪ xmlns:lmg="http://www.levertz.xyz/elm/commons" xmlns:lmp="http://www.
       ↪ levertz.xyz/elm/properties" xmlns:xsd="http://www.w3.org/2001/
       ↪ XMLSchema">
3    <xsd:import namespace="http://www.levertz.xyz/elm/commons" schemaLocation=
         ↪ "C:\Users\lars.PLT\Diss\Specs\commons.xsd"/>
4    <xsd:complexType name="property">
5      <xsd:annotation>
6        <xsd:documentation>Structure to describe the property of a unit.</
             ↪ xsd:documentation>
7      </xsd:annotation>
8      <xsd:sequence>
9        <xsd:element name="name" type="xsd:NMTOKEN"/>
10       <xsd:element name="identificator" type="lmg:identificator"/>
11       <xsd:element name="typeID" type="lmg:identificator"/>
12       <xsd:element name="carrierTypeID" type="lmg:identificator"/>
13       <xsd:element name="physicalUnit" type="xsd:string"/>
14     </xsd:sequence>
15   </xsd:complexType>
16   <xsd:complexType name="statement">
17     <xsd:sequence>
18       <xsd:element name="name" type="xsd:NMTOKEN"/>
```

```
19      <xsd:element name="identificator" type="lmg:identificator"/>
20      <xsd:element name="statementType" type="lmg:identificator"/>
21      <xsd:element name="statingInstance" type="lmg:identificator"/>
22      <xsd:element name="stateTime" type="xsd:dateTime"/>
23      <xsd:element name="timeStamp" type="xsd:dateTime"/>
24      <xsd:element name="equality" type="lmp:equality"/>
25      <xsd:element name="value" type="xsd:anyType"/>
26    </xsd:sequence>
27  </xsd:complexType>
28  <xsd:simpleType name="equality">
29    <xsd:restriction base="xsd:NMTOKEN">
30      <xsd:enumeration value="equals"/>
31      <xsd:enumeration value="lessThan"/>
32      <xsd:enumeration value="lessOrEqual"/>
33      <xsd:enumeration value="notEqual"/>
34      <xsd:enumeration value="greaterThan"/>
35      <xsd:enumeration value="greaterorEqual"/>
36      <xsd:enumeration value="notDefined"/>
37    </xsd:restriction>
38  </xsd:simpleType>
39  <xsd:complexType name="getPropertiesParams">
40    <xsd:annotation>
41      <xsd:documentation>Parameters for the getProperties operation.</
          ↪ xsd:documentation>
42    </xsd:annotation>
43    <xsd:sequence>
44      <xsd:element name="carrierNS" type="xsd:anyURI">
45        <xsd:annotation>
46          <xsd:documentation>Globally unique URL for name space of the
              ↪ porpertyCarrier. The party responsible for this namespace
              ↪ has to own and maintain the URL.</xsd:documentation>
47        </xsd:annotation>
48      </xsd:element>
49      <xsd:element name="carrierID" type="xsd:anyURI">
50        <xsd:annotation>
51          <xsd:documentation>Unique Idntifier of the propertyCarrier within
              ↪ the namespace. The party responsible for the namespace has
              ↪ to ensure uniqueness of the IDs.</xsd:documentation>
52        </xsd:annotation>
53      </xsd:element>
54      <xsd:element name="typeNSExp" type="xsd:string">
55        <xsd:annotation>
56          <xsd:documentation>RegExp to filter against the nameSpace part.</
              ↪ xsd:documentation>
57        </xsd:annotation>
58      </xsd:element>
59      <xsd:element name="typeIDExp" type="xsd:string">
60        <xsd:annotation>
61          <xsd:documentation>RegExp to filter against local id part.</
              ↪ xsd:documentation>
62        </xsd:annotation>
63      </xsd:element>
64      <xsd:element name="carrierTypeNSExp" type="xsd:string">
65        <xsd:annotation>
66          <xsd:documentation>RegExp to filter against the nameSpace part.</
              ↪ xsd:documentation>
```

```
67          </xsd:annotation>
68        </xsd:element>
69        <xsd:element name="carrierTypeIDExp" type="xsd:string">
70          <xsd:annotation>
71            <xsd:documentation>RegExp to filter against local id part.</
                  ↪ xsd:documentation>
72          </xsd:annotation>
73        </xsd:element>
74      </xsd:sequence>
75    </xsd:complexType>
76    <xsd:complexType name="getPropertiesResponse">
77      <xsd:annotation>
78        <xsd:documentation>Response message for the getProperties operation.</
              ↪ xsd:documentation>
79      </xsd:annotation>
80      <xsd:complexContent>
81        <xsd:extension base="lmg:statusOnlyResponse">
82          <xsd:sequence>
83            <xsd:element name="property" type="lmp:property" minOccurs="0"
                  ↪ maxOccurs="unbounded"/>
84          </xsd:sequence>
85        </xsd:extension>
86      </xsd:complexContent>
87    </xsd:complexType>
88    <xsd:complexType name="getStatementsParams">
89      <xsd:annotation>
90        <xsd:documentation>Parameters for the getStatements operation.</
              ↪ xsd:documentation>
91      </xsd:annotation>
92      <xsd:sequence>
93        <xsd:element name="propertyNS" type="xsd:anyURI">
94          <xsd:annotation>
95            <xsd:documentation>Globally unique URL for the name space of the
                  ↪ carryinf propertyto query. The party responsible for this
                  ↪ namespace has to own and maintain the URL.</
                  ↪ xsd:documentation>
96          </xsd:annotation>
97        </xsd:element>
98        <xsd:element name="propertyID" type="xsd:anyURI">
99          <xsd:annotation>
100           <xsd:documentation>Unique Idntifier of the property within the
                  ↪ namespace. The party responsible for the namespace has to
                  ↪ ensure uniqueness of the IDs.</xsd:documentation>
101         </xsd:annotation>
102       </xsd:element>
103       <xsd:element name="typeNSExp" type="xsd:string">
104         <xsd:annotation>
105           <xsd:documentation>RegExp to filter against the nameSpace part.</
                  ↪ xsd:documentation>
106         </xsd:annotation>
107       </xsd:element>
108       <xsd:element name="typeIDExp" type="xsd:string">
109         <xsd:annotation>
110           <xsd:documentation>RegExp to filter against local id part.</
                  ↪ xsd:documentation>
111         </xsd:annotation>
```

```
112       </xsd:element>
113       <xsd:element name="statingInstanceNSExp" type="xsd:string">
114         <xsd:annotation>
115           <xsd:documentation>RegExp to filter against the nameSpace part.</
                ↪ xsd:documentation>
116         </xsd:annotation>
117       </xsd:element>
118       <xsd:element name="statingInstanceIDExp" type="xsd:string">
119         <xsd:annotation>
120           <xsd:documentation>RegExp to filter against local id part.</
                ↪ xsd:documentation>
121         </xsd:annotation>
122       </xsd:element>
123     </xsd:sequence>
124   </xsd:complexType>
125   <xsd:complexType name="getStatementsResponse">
126     <xsd:annotation>
127       <xsd:documentation>Response message for the getStatements operation.</
            ↪ xsd:documentation>
128     </xsd:annotation>
129     <xsd:complexContent>
130       <xsd:extension base="lmg:statusOnlyResponse">
131         <xsd:sequence>
132           <xsd:element name="statement" type="lmp:statement" minOccurs="0"
                ↪ maxOccurs="unbounded"/>
133         </xsd:sequence>
134       </xsd:extension>
135     </xsd:complexContent>
136   </xsd:complexType>
137   <xsd:complexType name="addPropertyParams">
138     <xsd:annotation>
139       <xsd:documentation>Parameters for the addProperty operation.</
            ↪ xsd:documentation>
140     </xsd:annotation>
141     <xsd:sequence>
142       <xsd:element name="carrierID" type="lmg:identificator"/>
143       <xsd:element name="name" type="xsd:NMTOKEN"/>
144       <xsd:element name="typeID" type="lmg:identificator"/>
145       <xsd:element name="carrierTypeID" type="lmg:identificator"/>
146       <xsd:element name="physicalUnit" type="xsd:string" minOccurs="0"/>
147     </xsd:sequence>
148   </xsd:complexType>
149   <xsd:complexType name="addStatementParams">
150     <xsd:annotation>
151       <xsd:documentation>Parameters for the addStatement operation.</
            ↪ xsd:documentation>
152     </xsd:annotation>
153     <xsd:sequence>
154       <xsd:element name="carrierID" type="lmg:identificator"/>
155       <xsd:element name="name" type="xsd:NMTOKEN"/>
156       <xsd:element name="typeID" type="lmg:identificator"/>
157       <xsd:element name="statingInstanceID" type="lmg:identificator"/>
158       <xsd:element name="stateTime" type="xsd:dateTime" minOccurs="0"/>
159       <xsd:element name="timeStamp" type="xsd:dateTime" minOccurs="0"/>
160       <xsd:element name="equality" type="lmp:equality"/>
161       <xsd:element name="value" type="xsd:anyType"/>
```

```
162     </xsd:sequence>
163   </xsd:complexType>
164   <!-- Message elements -->
165   <xsd:element name="getPropertiesReq" type="lmp:getPropertiesParams"/>
166   <xsd:element name="getPropertiesRes" type="lmp:getPropertiesResponse"/>
167   <xsd:element name="addPropertyReq" type="lmp:addPropertyParams"/>
168   <xsd:element name="addPropertyRes" type="lmg:genericResponse"/>
169   <xsd:element name="getStatementsReq" type="lmp:getStatementsParams"/>
170   <xsd:element name="getStatementsRes" type="lmp:getStatementsResponse"/>
171   <xsd:element name="addStatementReq" type="lmp:addStatementParams"/>
172   <xsd:element name="addStatementRes" type="lmg:genericResponse"/>
173 </xsd:schema>
```

Das folgende Listing enthält die auf den vorab dargestellten Definitionen aufbauende Dienstbeschreibung.

```
1  <?xml version="1.0"?>
2  <wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl" xmlns:whttp="http:
       ↪ //www.w3.org/ns/wsdl/http" xmlns:lmp="http://www.levertz.xyz/elm/
       ↪ properties" targetNamespace="http://www.levertz.xyz/elm/properties">
3    <wsdl:documentation>
4      This file defines the Services for handling and exchange
5      of property information. The services are grouped to
6      support either property or statement handling.
7    </wsdl:documentation>
8    <!--
9      type definitions for messages and their elements
10   -->
11   <wsdl:types>
12     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace
           ↪ ="http://www.levertz.xyz/elm/properties">
13       <xsd:include schemaLocation="C:\Users\lars.PLT\Diss\Specs\
             ↪ propertiesMessages.xsd"/>
14     </xsd:schema>
15   </wsdl:types>
16   <wsdl:interface name="propertyHandlingService">
17     <wsdl:operation name="getProperties" pattern="http://www.w3.org/ns/wsdl/
           ↪ in-out" style="http://www.w3.org/ns/wsdl/style/iri">
18       <wsdl:documentation/>
19       <!-- insert documentation here -->
20       <wsdl:input messageLabel="In" element="lmp:getPropertiesReq"/>
21       <wsdl:output messageLabel="Out" element="lmp:getPropertiesRes"/>
22     </wsdl:operation>
23     <wsdl:operation name="addProperty" pattern="http://www.w3.org/ns/wsdl/in
           ↪ -out">
24       <wsdl:documentation/>
25       <!-- insert documentation here -->
26       <wsdl:input messageLabel="In" element="lmp:addPropertyReq"/>
27       <wsdl:output messageLabel="Out" element="lmp:addPropertyRes"/>
28     </wsdl:operation>
29     <wsdl:documentation/>
30     <!-- insert documentation here -->
31   </wsdl:interface>
32   <wsdl:interface name="statementHandlingService">
33     <wsdl:operation name="getStatements" pattern="http://www.w3.org/ns/wsdl/
           ↪ in-out" style="http://www.w3.org/ns/wsdl/style/iri">
34       <wsdl:documentation/>
```

```
35        <!-- insert documentation here -->
36        <wsdl:input messageLabel="In" element="lmp:getStatementsReq"/>
37        <wsdl:output messageLabel="Out" element="lmp:getStatementsRes"/>
38      </wsdl:operation>
39      <wsdl:operation name="addStatement" pattern="http://www.w3.org/ns/wsdl/
            ↪ in-out">
40        <wsdl:documentation/>
41        <!-- insert documentation here -->
42        <wsdl:input messageLabel="In" element="lmp:addStatementReq"/>
43        <wsdl:output messageLabel="Out" element="lmp:addStatementRes"/>
44      </wsdl:operation>
45      <wsdl:documentation/>
46      <!-- insert documentation here -->
47    </wsdl:interface>
48
49    <wsdl:binding name="defaultPropertyHandlingServiceBinding"
50       interface="lmp:propertyHandlingService"
51       type="http://www.w3.org/ns/wsdl/http">
52      <wsdl:operation ref="lmp:getProperties"
53         whttp:method="GET"
54         whttp:location="getProperties"/>
55      <wsdl:operation ref="lmp:addProperty"
56         whttp:method="POST"/>
57    </wsdl:binding>
58    <wsdl:binding name="defaultStatementHandlingServiceBinding"
59       interface="lmp:statementHandlingService"
60       type="http://www.w3.org/ns/wsdl/http">
61      <wsdl:operation ref="lmp:getStatements"
62         whttp:method="GET"
63         whttp:location="getStatements"/>
64      <wsdl:operation ref="lmp:addStatement"
65         whttp:method="POST"/>
66    </wsdl:binding>
67
68  </wsdl:description>
```

B.4 Dienste für Entitätsdatenaustausch

Das folgende Listing enthält die Nachrichten-Typen für die Dienste zum Entitätsdatenaustausch.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <xsd:schema targetNamespace="http://www.levertz.xyz/elm/entities" xmlns:xsd=
       ↪ "http://www.w3.org/2001/XMLSchema" xmlns:lmp="http://www.levertz.xyz/
       ↪ elm/properties" xmlns:lme="http://www.levertz.xyz/elm/entities"
       ↪ xmlns:lmg="http://www.levertz.xyz/elm/commons">
3    <xsd:import namespace="http://www.levertz.xyz/elm/commons" schemaLocation=
         ↪ "C:\Users\lars.PLT\Diss\Specs\commons.xsd"/>
4    <xsd:import namespace="http://www.levertz.xyz/elm/properties"
         ↪ schemaLocation="C:\Users\lars.PLT\Diss\Specs\propertiesMessages.xsd"
         ↪ />
5    <xsd:complexType name="unitReference">
6      <xsd:annotation>
```

```
7       <xsd:documentation>Reference to the real unit this representation
            ↪ belongs to.</xsd:documentation>
8     </xsd:annotation>
9     <xsd:simpleContent>
10      <xsd:extension base="xsd:string">
11        <xsd:attribute name="encoding" type="xsd:NMTOKEN" use="required">
12          <xsd:annotation>
13            <xsd:documentation>Hint how the system specific reference is to
                  ↪ be used.</xsd:documentation>
14          </xsd:annotation>
15        </xsd:attribute>
16      </xsd:extension>
17    </xsd:simpleContent>
18  </xsd:complexType>
19  <xsd:complexType name="entity">
20    <xsd:annotation>
21      <xsd:documentation>Entity description.</xsd:documentation>
22    </xsd:annotation>
23    <xsd:sequence>
24      <xsd:element name="name" type="xsd:NMTOKEN"/>
25      <xsd:element name="identificator" type="lmg:identificator"/>
26      <xsd:element name="firstIdentificator" type="lmg:identificator"/>
27      <xsd:element name="snapshotNumber" type="xsd:unsignedInt"/>
28      <xsd:element name="snapshotTimeStamp" type="xsd:dateTime"/>
29      <xsd:element name="snapshotEndTimeStamp" type="xsd:dateTime" nillable=
            ↪ "true"/>
30      <xsd:element name="followUpSnapshot" type="lmg:identificator"/>
31      <xsd:element name="previousSnapshot" minOccurs="0" maxOccurs="
            ↪ unbounded">
32        <xsd:complexType>
33          <xsd:complexContent>
34            <xsd:extension base="lmg:identificator">
35              <xsd:attribute name="revisionID" type="xsd:string" use="
                    ↪ required"/>
36              <xsd:attribute name="timeStamp" type="xsd:dateTime" use="
                    ↪ required"/>
37              <xsd:attribute name="endTimeStamp" type="xsd:dateTime" use="
                    ↪ required"/>
38            </xsd:extension>
39          </xsd:complexContent>
40        </xsd:complexType>
41      </xsd:element>
42      <xsd:element name="parentType" type="lmg:identificator" minOccurs="0"
            ↪ maxOccurs="unbounded"/>
43      <xsd:element name="unitReference" type="lme:unitReference"/>
44      <xsd:element name="currentLifeCycleProcess" type="lmg:identificator"/>
45      <xsd:element name="previousLifeCycleProcess" minOccurs="0" maxOccurs="
            ↪ unbounded">
46        <xsd:complexType>
47          <xsd:complexContent>
48            <xsd:extension base="lmg:identificator">
49              <xsd:attribute name="timeStamp" type="xsd:dateTime" use="
                    ↪ required"/>
50              <xsd:attribute name="endTimeStamp" type="xsd:dateTime" use="
                    ↪ required"/>
51            </xsd:extension>
```

```
52            </xsd:complexContent>
53          </xsd:complexType>
54        </xsd:element>
55        <xsd:element name="isPhysical" type="xsd:boolean"/>
56      </xsd:sequence>
57    </xsd:complexType>
58    <xsd:complexType name="createEntityTypeParams">
59      <xsd:annotation>
60        <xsd:documentation>Parameters for the createEntityType operation.</
              ↪ xsd:documentation>
61      </xsd:annotation>
62      <xsd:sequence>
63        <xsd:element name="name" type="xsd:NMTOKEN"/>
64        <xsd:element name="description" type="xsd:string"/>
65        <xsd:element name="remark" type="xsd:string" minOccurs="0"/>
66        <xsd:element name="documentRef" type="xsd:anyURI" minOccurs="0">
67          <xsd:annotation>
68            <xsd:documentation>Reference to a spcification document.</
                  ↪ xsd:documentation>
69          </xsd:annotation>
70        </xsd:element>
71        <xsd:element name="parentType" type="lmg:identificator" minOccurs="0"
              ↪ maxOccurs="unbounded"/>
72      </xsd:sequence>
73    </xsd:complexType>
74    <xsd:complexType name="createEntityParams">
75      <xsd:annotation>
76        <xsd:documentation>Parameters for the createEntity operation.</
              ↪ xsd:documentation>
77      </xsd:annotation>
78      <xsd:sequence>
79        <xsd:element name="name" type="xsd:NMTOKEN"/>
80        <xsd:element name="parentType" type="lmg:identificator" maxOccurs="
              ↪ unbounded"/>
81        <xsd:element name="unitReference" type="lme:unitReference"/>
82        <xsd:element name="isPhysical" type="xsd:boolean"/>
83        <xsd:element name="timeStamp" type="xsd:dateTime" minOccurs="0"/>
84      </xsd:sequence>
85    </xsd:complexType>
86    <xsd:complexType name="createNewEntitySnapshotParams">
87      <xsd:annotation>
88        <xsd:documentation>Parameters for the createNewEntitySnapshot
              ↪ operation.</xsd:documentation>
89      </xsd:annotation>
90      <xsd:sequence>
91        <xsd:choice>
92          <xsd:element name="identificator" type="lmg:identificator"/>
93          <xsd:element name="firstIdentificator" type="lmg:identificator"/>
94        </xsd:choice>
95        <xsd:element name="keepStatements" type="xsd:boolean"/>
96        <xsd:element name="keepRelations" type="xsd:boolean"/>
97        <xsd:element name="timeStamp" type="xsd:dateTime"/>
98      </xsd:sequence>
99    </xsd:complexType>
100   <xsd:complexType name="createNewEntitySnapshotResponse">
101     <xsd:annotation>
```

```
102       <xsd:documentation>Response message for the createNewEntitySnapshot
              ↪ operation.</xsd:documentation>
103     </xsd:annotation>
104     <xsd:complexContent>
105       <xsd:extension base="lmg:genericResponse">
106         <xsd:sequence>
107           <xsd:element name="snapshotNumber" type="xsd:unsignedInt"/>
108           <xsd:element name="childEntity" type="lmg:identificator" minOccurs
                  ↪ ="0" maxOccurs="unbounded"/>
109         </xsd:sequence>
110       </xsd:extension>
111     </xsd:complexContent>
112   </xsd:complexType>
113   <xsd:complexType name="buryEntityParams">
114     <xsd:annotation>
115       <xsd:documentation>Parameters for the buryEntity operation. The
              ↪ identificator may point to the newest (current) revision or the
              ↪ first one (revisionless identificator).</xsd:documentation>
116     </xsd:annotation>
117     <xsd:sequence>
118       <xsd:element name="identificatorNS" type="xsd:anyURI">
119         <xsd:annotation>
120           <xsd:documentation>Globally unique URL for name space of the
                  ↪ entity representation. </xsd:documentation>
121         </xsd:annotation>
122       </xsd:element>
123       <xsd:element name="identificatorID" type="xsd:anyURI">
124         <xsd:annotation>
125           <xsd:documentation>Unique Idntifier of the entity representation
                  ↪ within the namespace.</xsd:documentation>
126         </xsd:annotation>
127       </xsd:element>
128     </xsd:sequence>
129   </xsd:complexType>
130   <xsd:complexType name="buryEntityResponse">
131     <xsd:annotation>
132       <xsd:documentation>Response message for the buryEntity operation.</
              ↪ xsd:documentation>
133     </xsd:annotation>
134     <xsd:complexContent>
135       <xsd:extension base="lmg:statusOnlyResponse">
136         <xsd:sequence>
137           <xsd:element name="childEntity" type="lmg:identificator" minOccurs
                  ↪ ="0" maxOccurs="unbounded"/>
138         </xsd:sequence>
139       </xsd:extension>
140     </xsd:complexContent>
141   </xsd:complexType>
142   <xsd:complexType name="attachLifeCycleProcessToEntityParams">
143     <xsd:annotation>
144       <xsd:documentation>Parameters for the attachLifeCycleProcessToEntity
              ↪ operation.</xsd:documentation>
145     </xsd:annotation>
146     <xsd:sequence>
147       <xsd:choice>
148         <xsd:element name="identificator" type="lmg:identificator"/>
```

```
149         <xsd:element name="firstIdentificator" type="lmg:identificator"/>
150       </xsd:choice>
151       <xsd:element name="lifeCycleProcessIdentificator" type="
              ↪ lmg:identificator"/>
152       <xsd:element name="timeStamp" type="xsd:dateTime" minOccurs="0"/>
153     </xsd:sequence>
154   </xsd:complexType>
155   <xsd:complexType name="attachRelationToEntityParams">
156     <xsd:annotation>
157       <xsd:documentation>Parameters for the attachRelationToEntity operation
              ↪ .</xsd:documentation>
158     </xsd:annotation>
159     <xsd:sequence>
160       <xsd:choice>
161         <xsd:element name="identificator" type="lmg:identificator"/>
162         <xsd:element name="firstIdentificator" type="lmg:identificator"/>
163       </xsd:choice>
164       <xsd:element name="relationIdentificator" type="lmg:identificator"/>
165       <xsd:element name="timeStamp" type="xsd:dateTime" minOccurs="0"/>
166     </xsd:sequence>
167   </xsd:complexType>
168   <xsd:complexType name="detachFromEntityParams">
169     <xsd:annotation>
170       <xsd:documentation>Parameters for the detachFromEntity operation.</
              ↪ xsd:documentation>
171     </xsd:annotation>
172     <xsd:sequence>
173       <xsd:choice>
174         <xsd:element name="relationModel" type="lmg:identificator"/>
175         <xsd:element name="relationDescription" type="lmg:identificator"/>
176         <xsd:element name="lifeCycleProcess" type="lmg:identificator"/>
177       </xsd:choice>
178       <xsd:choice>
179         <xsd:element name="identificator" type="lmg:identificator"/>
180         <xsd:element name="firstIdentificator" type="lmg:identificator"/>
181       </xsd:choice>
182     </xsd:sequence>
183   </xsd:complexType>
184   <xsd:complexType name="getEntitiesParams">
185     <xsd:annotation>
186       <xsd:documentation>Parameters for the getEntities operation.</
              ↪ xsd:documentation>
187     </xsd:annotation>
188     <xsd:sequence>
189       <xsd:element name="identNSExp" type="xsd:string">
190         <xsd:annotation>
191           <xsd:documentation>RegExp to filter against the identificator of
                  ↪ the entity.</xsd:documentation>
192         </xsd:annotation>
193       </xsd:element>
194       <xsd:element name="identIDExp" type="xsd:string">
195         <xsd:annotation>
196           <xsd:documentation>RegExp to filter against local id part of the
                  ↪ entitie's identificator.</xsd:documentation>
197         </xsd:annotation>
198       </xsd:element>
```

```
199       <xsd:element name="typeNSExp" type="xsd:string">
200         <xsd:annotation>
201           <xsd:documentation>RegExp to filter against the nameSpace part of
                  ↪ the entityType.</xsd:documentation>
202         </xsd:annotation>
203       </xsd:element>
204       <xsd:element name="typeIDExp" type="xsd:string">
205         <xsd:annotation>
206           <xsd:documentation>RegExp to filter against local id part of the
                  ↪ entityType.</xsd:documentation>
207         </xsd:annotation>
208       </xsd:element>
209       <xsd:element name="unitRefFilter" type="xsd:string">
210         <xsd:annotation>
211           <xsd:documentation>RegExp to filter against unit reference.</
                  ↪ xsd:documentation>
212         </xsd:annotation>
213       </xsd:element>
214       <xsd:element name="getBurried" type="xsd:boolean">
215         <xsd:annotation>
216           <xsd:documentation>Also get burried entities.</xsd:documentation>
217         </xsd:annotation>
218       </xsd:element>
219       <xsd:element name="getPreviousSnapshotInfo" type="xsd:boolean">
220         <xsd:annotation>
221           <xsd:documentation>Fill in information about previous revisions
                  ↪ and lifeCycleProcesses.</xsd:documentation>
222         </xsd:annotation>
223       </xsd:element>
224     </xsd:sequence>
225   </xsd:complexType>
226   <xsd:complexType name="getEntitiesResponse">
227     <xsd:annotation>
228       <xsd:documentation>Response message for the getEntities operation.</
              ↪ xsd:documentation>
229     </xsd:annotation>
230     <xsd:complexContent>
231       <xsd:extension base="lmg:statusOnlyResponse">
232         <xsd:sequence>
233           <xsd:element name="entity" type="lme:entity" minOccurs="0"
                  ↪ maxOccurs="unbounded"/>
234         </xsd:sequence>
235       </xsd:extension>
236     </xsd:complexContent>
237   </xsd:complexType>
238   <xsd:complexType name="getByIdentificator">
239     <xsd:annotation>
240       <xsd:documentation>Parameters for simple get operations. A Specific
              ↪ revision is either referenced directly by its identificator or
              ↪ by the identificator of the first revision (revisionless
              ↪ identificator) and the revision ID</xsd:documentation>
241     </xsd:annotation>
242     <xsd:sequence>
243       <xsd:element name="identificatorNS" type="xsd:anyURI"/>
244       <xsd:element name="identificatorID" type="xsd:anyURI"/>
245       <xsd:element name="snapshotNumber" type="xsd:unsignedInt" minOccurs
```

```
        ↪ ="0"/>
246     </xsd:sequence>
247   </xsd:complexType>
248   <xsd:complexType name="getAttachedResponse">
249     <xsd:annotation>
250       <xsd:documentation>Response message for the getAttached operation.</
            ↪ xsd:documentation>
251     </xsd:annotation>
252     <xsd:complexContent>
253       <xsd:extension base="lmg:statusOnlyResponse">
254         <xsd:sequence>
255           <xsd:element name="lifeCycleProcess" type="lmg:identificator"
                ↪ minOccurs="0"/>
256           <xsd:choice minOccurs="0" maxOccurs="unbounded">
257             <xsd:element name="relationModel" type="lmg:identificator"/>
258             <xsd:element name="relationDescription" type="lmg:identificator
                  ↪ "/>
259           </xsd:choice>
260         </xsd:sequence>
261       </xsd:extension>
262     </xsd:complexContent>
263   </xsd:complexType>
264   <xsd:complexType name="snapshotIdentification">
265     <xsd:annotation>
266       <xsd:documentation>Structure with identificators for a specific
            ↪ snapshot</xsd:documentation>
267     </xsd:annotation>
268     <xsd:sequence>
269       <xsd:element name="identificator" type="lmg:identificator"/>
270       <xsd:element name="firstIdentificator" type="lmg:identificator"/>
271       <xsd:element name="snapshotNumber" type="xsd:unsignedInt"/>
272     </xsd:sequence>
273   </xsd:complexType>
274   <xsd:complexType name="getCurrentSnapshotResponse">
275     <xsd:annotation>
276       <xsd:documentation>Response message for the getCurrentSnapshot
            ↪ operation.</xsd:documentation>
277     </xsd:annotation>
278     <xsd:complexContent>
279       <xsd:extension base="lmg:statusOnlyResponse">
280         <xsd:sequence>
281           <xsd:element name="currentSnapshot" type="
                ↪ lme:snapshotIdentification"/>
282         </xsd:sequence>
283       </xsd:extension>
284     </xsd:complexContent>
285   </xsd:complexType>
286   <xsd:complexType name="getEntityHistoryResponse">
287     <xsd:annotation>
288       <xsd:documentation>Response message for the getEntityHistory operation
            ↪ .</xsd:documentation>
289     </xsd:annotation>
290     <xsd:complexContent>
291       <xsd:extension base="lmg:statusOnlyResponse">
292         <xsd:sequence>
293           <xsd:element name="snapshot" type="lme:snapshotIdentification"
```

```
                ↪ maxOccurs="unbounded"/>
294           </xsd:sequence>
295         </xsd:extension>
296       </xsd:complexContent>
297     </xsd:complexType>
298     <xsd:complexType name="getSnapshotAtTimeParams">
299       <xsd:annotation>
300         <xsd:documentation>Parameters for the getSnapshotAtTime operation.</
              ↪ xsd:documentation>
301       </xsd:annotation>
302       <xsd:complexContent>
303         <xsd:extension base="lme:getByIdentificator">
304           <xsd:sequence>
305             <xsd:element name="time" type="xsd:dateTime"/>
306           </xsd:sequence>
307         </xsd:extension>
308       </xsd:complexContent>
309     </xsd:complexType>
310     <xsd:complexType name="getSnapshotAtTimeResponse">
311       <xsd:annotation>
312         <xsd:documentation>Response message for the getSnapshotAtTime
              ↪ operation.</xsd:documentation>
313       </xsd:annotation>
314       <xsd:complexContent>
315         <xsd:extension base="lmg:statusOnlyResponse">
316           <xsd:sequence>
317             <xsd:element name="snapshot" type="lme:snapshotIdentification"
                  ↪ minOccurs="0"/>
318           </xsd:sequence>
319         </xsd:extension>
320       </xsd:complexContent>
321     </xsd:complexType>
322     <xsd:complexType name="setFollowUpSnapshotParams">
323       <xsd:sequence>
324         <xsd:element name="entity" type="lme:getByIdentificator"/>
325         <xsd:element name="newSnapshot" type="lme:snapshotIdentification"/>
326         <xsd:element name="startTime" type="xsd:dateTime">
327           <xsd:annotation>
328             <xsd:documentation>Point in time at which the new revision starts
                  ↪ .</xsd:documentation>
329           </xsd:annotation>
330         </xsd:element>
331       </xsd:sequence>
332     </xsd:complexType>
333     <xsd:complexType name="getValueAtTimeParams">
334       <xsd:annotation>
335         <xsd:documentation>Gets a value at a specific point in time. Either a
              ↪ property is taken and filters for statement types and stating
              ↪ instances are set, or a statement is taken. In the latter case
              ↪ the statement types and the stating instances must fit exactly.
              ↪ The parent property of the statement is filtered by type and
              ↪ clear name. When there are multiple fitting values all are
              ↪ returned.</xsd:documentation>
336       </xsd:annotation>
337       <xsd:sequence>
338         <xsd:choice>
```

```
339         <xsd:sequence>
340           <xsd:element name="statementNS" type="xsd:anyURI"/>
341           <xsd:element name="statementID" type="xsd:anyURI"/>
342           <xsd:element name="filterByStatementType" type="xsd:boolean"
                  ↪ default="true" minOccurs="0"/>
343           <xsd:element name="filterByStatingInstance" type="xsd:boolean"
                  ↪ default="true" minOccurs="0"/>
344         </xsd:sequence>
345         <xsd:sequence>
346           <xsd:element name="propertyNS" type="xsd:anyURI"/>
347           <xsd:element name="propertyID" type="xsd:anyURI"/>
348           <xsd:element name="statementTypeNSExp" type="xsd:string"/>
349           <xsd:element name="statementTypeIDExp" type="xsd:string"/>
350           <xsd:element name="statingInstanceNSExp" type="xsd:string"/>
351           <xsd:element name="statingInstanceIDExp" type="xsd:string"/>
352         </xsd:sequence>
353       </xsd:choice>
354       <xsd:element name="time" type="xsd:dateTime"/>
355     </xsd:sequence>
356   </xsd:complexType>
357   <xsd:complexType name="getValueAtTimeResponse">
358     <xsd:complexContent>
359       <xsd:extension base="lmg:statusOnlyResponse">
360         <xsd:sequence>
361           <xsd:element name="statement" type="lmp:statement" minOccurs="0"
                  ↪ maxOccurs="unbounded"/>
362         </xsd:sequence>
363       </xsd:extension>
364     </xsd:complexContent>
365   </xsd:complexType>
366   <xsd:complexType name="getValueHistParams">
367     <xsd:annotation>
368       <xsd:documentation>As getValueAtTime but concerns a time span between
              ↪ timeStart and time.</xsd:documentation>
369     </xsd:annotation>
370     <xsd:complexContent>
371       <xsd:extension base="lme:getValueAtTimeParams">
372         <xsd:sequence>
373           <xsd:element name="timeStart" type="xsd:dateTime" minOccurs="0"/>
374         </xsd:sequence>
375       </xsd:extension>
376     </xsd:complexContent>
377   </xsd:complexType>
378   <!-- Message elements -->
379   <xsd:element name="createEntityTypeReq" type="lme:createEntityTypeParams
          ↪ "/>
380   <xsd:element name="createEntityTypeRes" type="lmg:genericResponse"/>
381   <xsd:element name="createEntityReq" type="lme:createEntityParams"/>
382   <xsd:element name="createEntityRes" type="lmg:genericResponse"/>
383   <xsd:element name="createNewEntitySnapshotReq" type="
          ↪ lme:createNewEntitySnapshotParams"/>
384   <xsd:element name="createNewEntitySnapshotRes" type="
          ↪ lme:createNewEntitySnapshotResponse"/>
385   <xsd:element name="buryEntityReq" type="lme:buryEntityParams"/>
386   <xsd:element name="buryEntityRes" type="lme:buryEntityResponse"/>
387   <xsd:element name="attachLifeCycleProcessToEntityReq" type="
```

```
      ↪ lme:attachLifeCycleProcessToEntityParams"/>
388   <xsd:element name="attachLifeCycleProcessToEntityRes" type="
      ↪ lmg:statusOnlyResponse"/>
389   <xsd:element name="attachRelationToEntityReq" type="
      ↪ lme:attachRelationToEntityParams"/>
390   <xsd:element name="attachRelationToEntityRes" type="lmg:statusOnlyResponse
      ↪ "/>
391   <xsd:element name="detachFromEntityReq" type="lme:detachFromEntityParams
      ↪ "/>
392   <xsd:element name="detachFromEntityRes" type="lmg:statusOnlyResponse"/>
393   <xsd:element name="getEntitiesReq" type="lme:getEntitiesParams"/>
394   <xsd:element name="getEntitiesRes" type="lme:getEntitiesResponse"/>
395   <xsd:element name="getAttachedReq" type="lme:getByIdentificator"/>
396   <xsd:element name="getAttachedRes" type="lme:getAttachedResponse"/>
397   <xsd:element name="getCurrentSnapshotReq" type="lme:getByIdentificator"/>
398   <xsd:element name="getCurrentSnapshotRes" type="
      ↪ lme:getCurrentSnapshotResponse"/>
399   <xsd:element name="getEntityHistoryReq" type="lme:getByIdentificator"/>
400   <xsd:element name="getEntityHistoryRes" type="lme:getEntityHistoryResponse
      ↪ "/>
401   <xsd:element name="getSnapshotAtTimeReq" type="lme:getSnapshotAtTimeParams
      ↪ "/>
402   <xsd:element name="getSnapshotAtTimeRes" type="
      ↪ lme:getSnapshotAtTimeResponse"/>
403   <xsd:element name="setFollowUpSnapshotReq" type="
      ↪ lme:setFollowUpSnapshotParams"/>
404   <xsd:element name="setFollowUpSnapshotRes" type="lmg:statusOnlyResponse"/>
405   <xsd:element name="getValueAtTimeReq" type="lme:getValueAtTimeParams"/>
406   <xsd:element name="getValueAtTimeRes" type="lme:getValueAtTimeResponse"/>
407   <xsd:element name="getValueHistReq" type="lme:getValueHistParams"/>
408   <xsd:element name="getValueHistRes" type="lme:getValueAtTimeResponse"/>
409 </xsd:schema>
```

Das folgende Listing enthält die auf den vorab dargestellten Definitionen aufbauende Dienstbeschreibung.

```
1  <?xml version="1.0"?>
2  <wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl" xmlns:whttp="http:
       ↪ //www.w3.org/ns/wsdl/http" xmlns:lme="http://www.levertz.xyz/elm/
       ↪ entities" targetNamespace="http://www.levertz.xyz/elm/entities">
3    <wsdl:documentation>
4      This file defines the Service for handling and exchange
5      of entity information.
6    </wsdl:documentation>
7    <!--
8      type definitions for messages and their elements
9    -->
10   <wsdl:types>
11     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace
         ↪ ="http://www.levertz.xyz/elm/entities">
12       <xsd:include schemaLocation="C:\Users\lars.PLT\Diss\Specs\
           ↪ entitiesMessages.xsd"/>
13     </xsd:schema>
14   </wsdl:types>
15   <wsdl:interface name="entityHandlingServiceData">
16     <wsdl:operation name="createEntityType" pattern="http://www.w3.org/ns/
         ↪ wsdl/in-out">
```

```
17        <wsdl:documentation/>
18        <!-- insert documentation here -->
19        <wsdl:input messageLabel="In" element="lme:createEntityTypeReq"/>
20        <wsdl:output messageLabel="Out" element="lme:createEntityTypeRes"/>
21      </wsdl:operation>
22      <wsdl:operation name="createEntity" pattern="http://www.w3.org/ns/wsdl/
            ↪ in-out">
23        <wsdl:documentation/>
24        <!-- insert documentation here -->
25        <wsdl:input messageLabel="In" element="lme:createEntityReq"/>
26        <wsdl:output messageLabel="Out" element="lme:createEntityRes"/>
27      </wsdl:operation>
28      <wsdl:operation name="createNewEntitySnapshot" pattern="http://www.w3.
            ↪ org/ns/wsdl/in-out">
29        <wsdl:documentation/>
30        <!-- insert documentation here -->
31        <wsdl:input messageLabel="In" element="lme:createNewEntitySnapshotReq"
              ↪ />
32        <wsdl:output messageLabel="Out" element="lme:createNewEntitySnapshoRes
              ↪ "/>
33      </wsdl:operation>
34      <wsdl:operation name="buryEntity" pattern="http://www.w3.org/ns/wsdl/in-
            ↪ out" style="http://www.w3.org/ns/wsdl/style/iri">
35        <wsdl:documentation/>
36        <!-- insert documentation here -->
37        <wsdl:input messageLabel="In" element="lme:buryEntityReq"/>
38        <wsdl:output messageLabel="Out" element="lme:buryEntityRes"/>
39      </wsdl:operation>
40      <wsdl:operation name="getEntities" pattern="http://www.w3.org/ns/wsdl/in
            ↪ -out" style="http://www.w3.org/ns/wsdl/style/iri">
41        <wsdl:documentation/>
42        <!-- insert documentation here -->
43        <wsdl:input messageLabel="In" element="lme:getEntitiesReq"/>
44        <wsdl:output messageLabel="Out" element="lme:getEntitiesRes"/>
45      </wsdl:operation>
46      <wsdl:operation name="getAttached" pattern="http://www.w3.org/ns/wsdl/in
            ↪ -out" style="http://www.w3.org/ns/wsdl/style/iri">
47        <wsdl:documentation/>
48        <!-- insert documentation here -->
49        <wsdl:input messageLabel="In" element="lme:getAttachedReq"/>
50        <wsdl:output messageLabel="Out" element="lme:getAttachedRes"/>
51      </wsdl:operation>
52      <wsdl:operation name="getCurrentSnapshot" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
53        <wsdl:documentation/>
54        <!-- insert documentation here -->
55        <wsdl:input messageLabel="In" element="lme:getCurrentSnapshotReq"/>
56        <wsdl:output messageLabel="Out" element="lme:getCurrentSnapshotRes"/>
57      </wsdl:operation>
58      <wsdl:operation name="getEntityHistory" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
59        <wsdl:documentation/>
60        <!-- insert documentation here -->
61        <wsdl:input messageLabel="In" element="lme:getEntityHistoryReq"/>
62        <wsdl:output messageLabel="Out" element="lme:getEntityHistoryRes"/>
63      </wsdl:operation>
```

```
64      <wsdl:operation name="getSnapshotAtTime" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out" style="http://www.w3.org/ns/wsdl/style/iri">
65        <wsdl:documentation/>
66        <!-- insert documentation here -->
67        <wsdl:input messageLabel="In" element="lme:getSnapshotAtTimeReq"/>
68        <wsdl:output messageLabel="Out" element="lme:getSnapshotAtTimeRes"/>
69      </wsdl:operation>
70      <wsdl:operation name="setFollowUpSnapshot" pattern="http://www.w3.org/ns
            ↪ /wsdl/in-out">
71        <wsdl:documentation/>
72        <!-- insert documentation here -->
73        <wsdl:input messageLabel="In" element="lme:setFollowupSnapshotReq"/>
74        <wsdl:output messageLabel="Out" element="lme:setFollowupSnapshotRes"/>
75      </wsdl:operation>
76      <wsdl:documentation/>
77      <!-- insert documentation here -->
78    </wsdl:interface>
79    <wsdl:interface name="entityHandlingServiceModels">
80      <wsdl:operation name="attachLifeCycleProcessToEntity" pattern="http://
            ↪ www.w3.org/ns/wsdl/in-out">
81        <wsdl:documentation/>
82        <!-- insert documentation here -->
83        <wsdl:input messageLabel="In" element="
              ↪ lme:attachLifeCycleProcessToEntityReq"/>
84        <wsdl:output messageLabel="Out" element="
              ↪ lme:attachLifeCycleProcessToEntityRes"/>
85      </wsdl:operation>
86      <wsdl:operation name="attachRelationToEntity" pattern="http://www.w3.org
            ↪ /ns/wsdl/in-out">
87        <wsdl:documentation/>
88        <!-- insert documentation here -->
89        <wsdl:input messageLabel="In" element="lme:attachRelationToEntityReq"/
            ↪ >
90        <wsdl:output messageLabel="Out" element="lme:attachRelationToEntityRes
            ↪ "/>
91      </wsdl:operation>
92      <wsdl:operation name="detachFromEntity" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out">
93        <wsdl:documentation/>
94        <!-- insert documentation here -->
95        <wsdl:input messageLabel="In" element="lme:detachFromEntityReq"/>
96        <wsdl:output messageLabel="Out" element="lme:detachFromEntityRes"/>
97      </wsdl:operation>
98    </wsdl:interface>
99    <wsdl:interface name="entityHandlingServiceAdditional">
100     <wsdl:operation name="getValueAtTime" pattern="http://www.w3.org/ns/wsdl
            ↪ /in-out">
101       <wsdl:documentation/>
102       <!-- insert documentation here -->
103       <wsdl:input messageLabel="In" element="lme:getValueAtTimeReq"/>
104       <wsdl:output messageLabel="Out" element="lme:getValueAtTimeRes"/>
105     </wsdl:operation>
106     <wsdl:operation name="getValueHist" pattern="http://www.w3.org/ns/wsdl/
            ↪ in-out">
107       <wsdl:documentation/>
108       <!-- insert documentation here -->
```

```
109         <wsdl:input messageLabel="In" element="lme:getValueHistReq"/>
110         <wsdl:output messageLabel="Out" element="lme:getValueHistRes"/>
111       </wsdl:operation>
112     </wsdl:interface>
113
114     <wsdl:binding name="defaultEntityHandlingServiceDataBinding" interface="
            ↪ lme:entityHandlingServiceData" type="http://www.w3.org/ns/wsdl/http"
            ↪ >
115       <wsdl:operation ref="lme:createEntityType" whttp:method="PUT"/>
116       <wsdl:operation ref="lme:createEntity" whttp:method="PUT"/>
117       <wsdl:operation ref="lme:createNewEntitySnapshot" whttp:method="POST"/>
118       <wsdl:operation ref="lme:buryEntity" whttp:method="DELETE"
              ↪ whttp:location="buryEntity"/>
119       <wsdl:operation ref="lme:getEntities" whttp:method="GET" whttp:location=
              ↪ "getEntities"/>
120       <wsdl:operation ref="lme:getAttached" whttp:method="GET" whttp:location=
              ↪ "getAttached"/>
121       <wsdl:operation ref="lme:getCurrentSnapshot" whttp:method="GET"
              ↪ whttp:location="getCurrentRevision"/>
122       <wsdl:operation ref="lme:getEntityHistory" whttp:method="GET"
              ↪ whttp:location="getRevisionHistory"/>
123       <wsdl:operation ref="lme:getSnapshotAtTime" whttp:method="GET"
              ↪ whttp:location="getRevisionAtTime"/>
124       <wsdl:operation ref="lme:setFollowUpSnapshot" whttp:method="POST"/>
125     </wsdl:binding>
126     <wsdl:binding name="defaultEntityHandlingServiceDataBinding" interface="
            ↪ lme:entityHandlingServiceData" type="http://www.w3.org/ns/wsdl/http"
            ↪ >
127       <wsdl:operation ref="lme:attachLifeCycleProcessToEntity" whttp:method="
              ↪ POST"/>
128       <wsdl:operation ref="lme:attachRelationToEntity" whttp:method="POST"/>
129       <wsdl:operation ref="lme:detachFromEntity" whttp:method="POST"/>
130     </wsdl:binding>
131     <wsdl:binding name="defaultEntityHandlingServiceAdditionalBinding"
            ↪ interface="lme:entityHandlingServiceAdditional" type="http://www.w3.
            ↪ org/ns/wsdl/http">
132       <wsdl:operation ref="lme:getValueAtTime" whttp:method="GET"
              ↪ whttp:location="getRevisionAtTime"/>
133       <wsdl:operation ref="lme:getValueHist" whttp:method="GET" whttp:location
              ↪ ="getRevisionAtTime"/>
134     </wsdl:binding>
135 </wsdl:description>
```

B.5 Dienste für Ereignisdatenaustausch

Das folgende Listing enthält die Nachrichten-Typen für die Dienste zum Ereignisdatenaustausch.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xsd:schema targetNamespace="http://www.levertz.xyz/elm/events" xmlns:lmg="
      ↪ http://www.levertz.xyz/elm/commons" xmlns:lmv="http://www.levertz.xyz/
      ↪ elm/events" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3   <xsd:import namespace="http://www.levertz.xyz/elm/commons" schemaLocation=
        ↪ "C:\Users\lars.PLT\Diss\Specs\commons.xsd"/>
```

```
4   <xsd:complexType name="eventDescription">
5     <xsd:sequence>
6       <xsd:element name="name" type="xsd:NMTOKEN"/>
7       <xsd:element name="timeStamp" type="xsd:dateTime"/>
8       <xsd:element name="concernedEntities" minOccurs="0" maxOccurs="
            ↪ unbounded">
9         <xsd:complexType>
10          <xsd:complexContent>
11            <xsd:extension base="lmg:identificator">
12              <xsd:attribute name="concern" type="lmv:concern" use="optional
                    ↪ " default="genericConcern"/>
13            </xsd:extension>
14          </xsd:complexContent>
15        </xsd:complexType>
16      </xsd:element>
17      <xsd:element name="subject" type="xsd:string"/>
18      <xsd:element name="description" type="xsd:string"/>
19      <xsd:element name="systemData" type="xsd:anyType" minOccurs="0"/>
20    </xsd:sequence>
21  </xsd:complexType>
22  <xsd:simpleType name="concern">
23    <xsd:restriction base="xsd:NMTOKEN">
24      <xsd:enumeration value="causedBy"/>
25      <xsd:enumeration value="influences"/>
26      <xsd:enumeration value="genericConcern"/>
27    </xsd:restriction>
28  </xsd:simpleType>
29  <xsd:complexType name="createEventParams">
30    <xsd:annotation>
31      <xsd:documentation>Parameters for the creataEvent operation.</
            ↪ xsd:documentation>
32    </xsd:annotation>
33    <xsd:sequence>
34      <xsd:element name="event" type="lmv:eventDescription"/>
35    </xsd:sequence>
36  </xsd:complexType>
37  <xsd:complexType name="readEventParams">
38    <xsd:annotation>
39      <xsd:documentation>Parameters for the readEvent operation.</
            ↪ xsd:documentation>
40    </xsd:annotation>
41    <xsd:sequence>
42      <xsd:element name="identificator" type="lmg:identificator"/>
43    </xsd:sequence>
44  </xsd:complexType>
45  <xsd:complexType name="readEventResponse">
46    <xsd:annotation>
47      <xsd:documentation>Response messagen for the readEvent operation.</
            ↪ xsd:documentation>
48    </xsd:annotation>
49    <xsd:complexContent>
50      <xsd:extension base="lmg:statusOnlyResponse">
51        <xsd:sequence>
52          <xsd:element name="event" type="lmv:eventDescription"/>
53        </xsd:sequence>
54      </xsd:extension>
```

```
55      </xsd:complexContent>
56    </xsd:complexType>
57    <xsd:complexType name="addEntityToEventParams">
58      <xsd:annotation>
59        <xsd:documentation>Parameters for the addEntityToEvent operation.</
              ↪ xsd:documentation>
60      </xsd:annotation>
61      <xsd:sequence>
62        <xsd:element name="event" type="lmg:identificator"/>
63        <xsd:element name="entity">
64          <xsd:complexType>
65            <xsd:complexContent>
66              <xsd:extension base="lmg:identificator">
67                <xsd:attribute name="concern" type="lmv:concern" use="optional
                      ↪ " default="genericConcern"/>
68              </xsd:extension>
69            </xsd:complexContent>
70          </xsd:complexType>
71        </xsd:element>
72      </xsd:sequence>
73    </xsd:complexType>
74    <xsd:complexType name="getEventsParams">
75      <xsd:annotation>
76        <xsd:documentation>Parameters for the getEvents operation.</
              ↪ xsd:documentation>
77      </xsd:annotation>
78      <xsd:sequence>
79        <xsd:element name="identificatorNSExp" type="xsd:string">
80          <xsd:annotation>
81            <xsd:documentation>RegExp to filter against the nameSpace part.</
                  ↪ xsd:documentation>
82          </xsd:annotation>
83        </xsd:element>
84        <xsd:element name="identificatorIDExp" type="xsd:string">
85          <xsd:annotation>
86            <xsd:documentation>RegExp to filter against local id part.</
                  ↪ xsd:documentation>
87          </xsd:annotation>
88        </xsd:element>
89        <xsd:element name="timeWindowStart" type="xsd:dateTime"/>
90        <xsd:element name="timeWindowEnd" type="xsd:dateTime"/>
91        <xsd:element name="excludeTimeWindow" type="xsd:boolean"/>
92        <xsd:element name="entityNSExp" type="xsd:string">
93          <xsd:annotation>
94            <xsd:documentation>RegExp to filter against the nameSpace part.</
                  ↪ xsd:documentation>
95          </xsd:annotation>
96        </xsd:element>
97        <xsd:element name="entityIDExp" type="xsd:string">
98          <xsd:annotation>
99            <xsd:documentation>RegExp to filter against local id part.</
                  ↪ xsd:documentation>
100         </xsd:annotation>
101       </xsd:element>
102       <xsd:element name="descriptionFilter" type="xsd:string">
103         <xsd:annotation>
```

```
104             <xsd:documentation>regExp to filter against desription field.</
                    ↪ xsd:documentation>
105           </xsd:annotation>
106         </xsd:element>
107       </xsd:sequence>
108     </xsd:complexType>
109     <xsd:complexType name="getEventsResponse">
110       <xsd:annotation>
111         <xsd:documentation>Response message for the getEvents operation.</
                ↪ xsd:documentation>
112       </xsd:annotation>
113       <xsd:complexContent>
114         <xsd:extension base="lmg:statusOnlyResponse">
115           <xsd:sequence>
116             <xsd:element name="event" type="lmv:eventDescription" minOccurs="0
                    ↪ " maxOccurs="unbounded"/>
117           </xsd:sequence>
118         </xsd:extension>
119       </xsd:complexContent>
120     </xsd:complexType>
121     <!-- Message elements -->
122     <xsd:element name="createEventReq" type="lmv:createEventParams"/>
123     <xsd:element name="createEventRes" type="lmg:genericResponse"/>
124     <xsd:element name="readEventReq" type="lmv:readEventParams"/>
125     <xsd:element name="readEventRes" type="lmv:readEventResponse"/>
126     <xsd:element name="addEntityToEventReq" type="lmv:addEntityToEventParams"/
            ↪ >
127     <xsd:element name="addEntityToEventRes" type="lmg:statusOnlyResponse"/>
128     <xsd:element name="getEventsReq" type="lmv:getEventsParams"/>
129     <xsd:element name="getEventsRes" type="lmv:getEventsResponse"/>
130 </xsd:schema>
```

Das folgende Listing enthält die auf den vorab dargestellten Definitionen aufbauende Dienstbeschreibung.

```
1  <?xml version="1.0"?>
2  <wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl" xmlns:whttp="http:
       ↪ //www.w3.org/ns/wsdl/http" xmlns:lmv="http://www.levertz.xyz/elm/
       ↪ events" targetNamespace="http://www.levertz.xyz/elm/events">
3    <wsdl:documentation>
4      This file defines the Service for handling and exchange
5      of event information.
6    </wsdl:documentation>
7    <!--
8      type definitions for messages and their elements
9    -->
10   <wsdl:types>
11     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" targetNamespace
           ↪ ="http://www.levertz.xyz/elm/events">
12       <xsd:include schemaLocation="C:\Users\lars.PLT\Diss\Specs\
             ↪ eventsMessages.xsd"/>
13     </xsd:schema>
14   </wsdl:types>
15   <wsdl:interface name="eventHandlingService">
16     <wsdl:operation name="createEvent" pattern="http://www.w3.org/ns/wsdl/in
           ↪ -out">
17       <wsdl:documentation/>
```

```
18        <!-- insert documentation here -->
19        <wsdl:input messageLabel="In" element="lmv:createEventReq"/>
20        <wsdl:output messageLabel="Out" element="lmv:createEventRes"/>
21      </wsdl:operation>
22      <wsdl:operation name="readEvent" pattern="http://www.w3.org/ns/wsdl/in-
            ↪ out" style="http://www.w3.org/ns/wsdl/style/iri">
23        <wsdl:documentation/>
24        <!-- insert documentation here -->
25        <wsdl:input messageLabel="In" element="lmv:readEventReq"/>
26        <wsdl:output messageLabel="Out" element="lmv:readEventRes"/>
27      </wsdl:operation>
28      <wsdl:operation name="addEntityToEvent" pattern="http://www.w3.org/ns/
            ↪ wsdl/in-out">
29        <wsdl:documentation/>
30        <!-- insert documentation here -->
31        <wsdl:input messageLabel="In" element="lmv:addEntityToEventReq"/>
32        <wsdl:output messageLabel="Out" element="lmv:addEntityToEventRes"/>
33      </wsdl:operation>
34      <wsdl:operation name="getEvents" pattern="http://www.w3.org/ns/wsdl/in-
            ↪ out" style="http://www.w3.org/ns/wsdl/style/iri">
35        <wsdl:documentation/>
36        <!-- insert documentation here -->
37        <wsdl:input messageLabel="In" element="lmv:getEventsReq"/>
38        <wsdl:output messageLabel="Out" element="lmv:getEventsRes"/>
39      </wsdl:operation>
40      <wsdl:documentation/>
41      <!-- insert documentation here -->
42    </wsdl:interface>
43
44    <wsdl:binding name="defaultEventHandlingServiceBinding"
45       interface="lmv:eventHandlingService"
46       type="http://www.w3.org/ns/wsdl/http">
47      <wsdl:operation ref="lmv:createEvent"
48        whttp:method="PUT"/>
49      <wsdl:operation ref="lmv:readEvent"
50        whttp:method="GET"
51        whttp:location="readEvent"/>
52      <wsdl:operation ref="lmv:addEntityToEvent"
53        whttp:method="POST"/>
54      <wsdl:operation ref="lmv:getEvents"
55        whttp:method="GET"
56        whttp:location="getEvents"/>
57    </wsdl:binding>
58
59
60  </wsdl:description>
```

Literaturverzeichnis

[1] Akkiraju, R., Farrell, J., Miller, J., Nagarajan, M., Schmidt, M.-T., Sheth, A., und Verma, K., "Web Service Semantics - WSDL-S," W3C, W3C Member Submission, November 7th 2005, http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/. [Online]. URL: http://www.w3.org/Submission/2005/SUBM-WSDL-S-20051107/

[2] Albrecht, H., "On Meta-Modeling for Communication in Operational Process Control Engineering," Fortschritt-Bericht, RWTH Aachen, 2002.

[3] Anderl, R., Strang, D., Picard, A., und Christ, A., "Integriertes Bauteildatenmodell für Industrie 4.0," *ZWF Zeitschrift für wirtschaftlichen Fabrikbetrieb*, Vol. 109, Nr. 1 - 2, S. 64 – 69, 2014.

[4] Anke, J., Wolf, B., Neugebauer, M., Eisenreich, K., Do, H. H., und Hackenbroich, G., "A middleware for real-world aware plm applications," in *Communication in Distributed Systems (KiVS), 2007 ITG-GI Conference*, Feb 2007, S. 1–12.

[5] Belshe, M., Peon, R., und Thomson, M., "Request for comments: 7540 - hypertext transfer protocol version 2 (http/2)," Internet Engineering Task Force (IETF), Tech. Rep., 2015, iSSN: 2070-1721. [Online]. URL: https://tools.ietf.org/html/rfc7540

[6] Biswas, A., Fenves, S. J., Shapiro, V., und Sriram, R., "Representation of heterogeneous material properties in the Core Product Model," *Engineering with Computers*, Vol. 24, Nr. 1, S. 43–58, 2008.

[7] Booth, D. und Liu, C. K., "Web Services Description Language (WSDL) Version 2.0 Part 0: Primer," W3C, W3C Recommendation, Jun 2007, http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626. [Online]. URL: http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626

[8] Braaksma, A., Klingenberg, W., und van Exel, P., "A review of the use of asset informations standards for collaboration in the process industry," *Computers in industry*, Vol. 62, S. 337 – 350, 2011.

[9] Brown, R. E. und Humphrey, B. G., "Asset mangagement for transmission and distribution," *IEEE power & energy magazine*, 2005.

[10] Burkett, W. C., "Product data markup language: a new paradigm for product data exchange and integration," *Computer-Aided Design*, Vol. 33, Nr. 7, S. 489 – 500, 2001. [Online]. URL: http://www.sciencedirect.com/science/article/pii/S0010448501000483

[11] Cassina, J., Taisch, M., Potter, D., und Parlikad, A. K., "Development of PROMISE Architecture and PDKM Semantic Object Model," in *Proceedings of the International Conference on Concurrent Enterprising*, Vol. 14, 2008.

[12] Cassina, J., Tomasella, M., Matta, A., Taisch, M., und Felicetti, G., *Advances in Production Management Systems*, ser. IFIP International Federation for Information Processing. Boston: Springer, 2007, Vol. 246, Kapitel Closed-loop PLM of Household Appliances: An Industrial Approach, S. 153 – 160.

[13] Chinnici, R., Haas, H., Lewis, A. A., Moreau, J.-J., Orchard, D., und Weerawarana, S., "Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts," W3C, W3C Recommendation, Jun 2007, http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626. [Online]. URL: http://www.w3.org/TR/2007/REC-wsdl20-adjuncts-20070626

[14] Christensen, E., Curbera, F., Meredith, G., und Weerawarana, S., "Web Services Description Language (WSDL) 1.1," W3C, W3C Note, March 15th 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315. [Online]. URL: http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[15] Collins-Sussman, B., Fitzpatrick, B. W., und Pilato, C. M., "Versionskontrolle mit subversion - für subversion 1.7," SVN, Tech. Rep. [Online]. URL: http://svnbook.red-bean.com/de/1.7/index.html

[16] Crnkovic, I. und Larsson, M., "Challenges of component-based development," *The Journal of Systems and Software*, Vol. 61, S. 201 – 212, 2002.

[17] Diedrich, C., Meyer, M., Evertz, L., und Schäfer, W., "Dienste in der Automatisierungstechnik - Automatisierungsgeräte werden I40-Komponenten," *Automatisierungstechnische Praxis : atp edition*, Vol. 12, 2014.

[18] *DIN 16557-3: Elektronischer Datenaustausch für Verwaltung, Wirtschaft und Transport (EDIFACT); Allgemeine Einführung für Einheitliche Nachrichtentypen (UNSMs)*, DIN Deutsches Institut für Normung e. V. Std., 4 1994.

[19] *DIN EN 61360-1: Genormte Datenelementtypen mit Klassifikationsschema für elektrische Bauteile - Teil 1: Definitionen - Regeln und Methoden*, DIN Deutsches Institut für Normung e. V. Std., Dez 2004.

[20] *DIN 4002: Merkmale und Geltungsbereiche zum Produktdatenaustausch*, DIN Deutsches Institut für Normung e. V. Std., Jul 2007.

[21] *DIN EN 61987-1: Industrielle Leittechnik - Datenstrukturen und -elemente in Katalogen der Prozessleittechnik - Teil 1: Messeinrichtungen mit analogen und digitalen Ausgängen*, DIN Deutsches Institut für Normung e. V. Std., Okt 2007.

[22] *DIN 4000-1: Sachmerkmal-Listen - Teil 1: Begriffe und Grundsätze*, DIN Deutsches Institut für Normung e. V. Std., Sep 2012.

[23] *DIN EN 61987-11: Industrielle Leittechnik - Datenstrukturen und -elemente in Katalogen der Prozessleittechnik - Teil 11: Merkmalleisten (ML) für Messgeräte für den elektronischen Datenaustausch - Allgemeine Strukturen*, DIN Deutsches Institut für Normung e. V. Std., Jul 2014.

[24] *DIN EN 61987-12: Industrielle Leittechnik - Datenstrukturen und -elemente in Katalogen der Prozessleittechnik - Teil 12: Merkmalleisten (ML) für Durchflussmessgeräte für den elektronischen Datenaustausch*, DIN Deutsches Institut für Normung e. V. Std., Mai 2014.

[25] *DIN SPEC 40912: Kernmodelle - Beschreibung und Beispiele*, DIN Deutsches Institut für Normung e. V. Std., Nov. 2014.

[26] *DIN EN 62714: Datenaustauschformat für Planungsdaten industrieller Automatisierungssysteme - Automation markup language*, DIN Deutsches Institut für Normung e. V. Std., 6 2015.

[27] eCl@ss e. V. (2015, 10) Übersicht über den eCl@ss Standard. [Online]. URL: http://www.eclass.de/eclasscontent/standard/overview.html.de. Abgerufen am: 16.10.2015

[28] Eigner, M. und Stelzer, R., *Product Lifecycle Management*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[29] Eisler, M., "Request for comments: 4506 - xdr: External data representation standard," Internet Engineering Task Force (IETF), Tech. Rep., 2006. [Online]. URL: https://tools.ietf.org/html/rfc4506

[30] El-Akruti, K., Dwight, R., und Zhang, T., "The strategic role of engineering asset management," *International Journal of Production Economics*, Vol. 146, Nr. 1, S. 227 – 239, 2013. [Online]. URL: http://www.sciencedirect.com/science/article/pii/S0925527313003149

[31] El-Akruti, K. O., "The strategic role of engineering asset management in capital intensive oragnisations," Dissertation, School of Mechanical, Materials and Mechatronic Engineering, University of Wollongong, 2012. [Online]. URL: http://ro.uow.edu.au/theses/3539/

[32] Epple, U., "Merkmale als Grundlage der Interoperabilität technischer Systeme," *at - Automatisierungstechnik*, Vol. 59, Nr. 7, S. 440–450, Jul. 2011.

[33] Evertz, L. und Epple, U., "Vergleich von SOA-Ansätzen unter den Gesichtspunkten des OASIS-Referenzmodells," in *AUTOMATION 2014: 15. Branchentreff der Mess- und Automatisierungstechnik*. Düsseldorf: VDI-Verlag GmbH, jul 2014.

[34] Evertz, L. und Epple, U., "Semi-Automatic Development of Service Adaptors from Property-Based Service Descriptions," in *ETFA 2015: IEEE 20th International Conference on Emerging Technologies and Factory Automation*. Piscataway, NJ: IEEE, 2015.

[35] Fay, A., Diedrich, C., Thron, M., Scholz, A., Puntel, P., Ladiges, J., und Holm, T., "Wie bekommt Industrie 4.0 Bedeutung?" *atp edition*, Vol. 57, Nr. 07-08, S. 30–43, 2015.

[36] Fensel, D., Fischer, F., Kopecký, J., Krummenacher, R., Lambert, D., und Vitvar, T., "WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web," W3C, W3C Member Submission, August 23rd 2010, http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/. [Online]. URL: http://www.w3.org/Submission/2010/SUBM-WSMO-Lite-20100823/

[37] Fenves, S. J., Choi, Y., Gurumoorthy, B., Mocko, G., und Sriram, R. D., *Master Product Model for the Support of Tighter Design-Analysis Integration*, National Institute of Standards and Technology Std. NISTIR 7004, 2003.

[38] Fenves, S., Foufou, S., Bock, C., Bouillon, N., und Sriram, R., *CPM2: a revised core product model for representing design information*, National Institute of Standards and Technology Std. NISTIR 7185, 2004.

[39] Fenves, S., Foufou, S., Bock, C., und Sriram, R., *A core product model for representing design information*, National Institute of Standards and Technology Std. NISTIR 6736, 2001.

[40] Foley, J. T., "An infrastructure for electromechanical appliances on the internet," Bachelor- und Masterarbeit, Massachusetts Institute of Technology, 1999.

[41] Främling, K. und Nyman, J., "Information architecture for intelligent products in the internet of things," *Beyond Business Logistics proceedings of NOFOMA*, S. 224 – 229, 2008.

[42] Freed, N. und Borenstein, N., "Request for comments: 2045 - multipurpose internet mail extensions (mime) part one: Format of internet message bodies," Internet Engineering Task Force (IETF), Tech. Rep., 1996, iSSN: 2070-1721. [Online]. URL: https://tools.ietf.org/html/rfc7540

[43] Gerlitz, T., Hansen, N., und Dernehl, S., Christianand Kowalewski, "artshop: A continuous integration and quality assessmentframework for model-based software artifacts," in *12. Dagstuhl-Workshop Modelbasierte Entwicklungeingebetteter Systeme (MBEES)*. fortiss Technischer Bericht, 2016, S. 13–22.

[44] *International Electrotechnical Vocabulary - Part 351: Control technology*, German Commission for Electrical, Electronic and Information Technologies of DIN and VDE Std., June 2008, identical with IEC 60050-351:2006-10. IEV 351.

[45] *EPC Tag Data Standard*, GS1 Std., 01 2014. [Online]. URL: http://www.gs1.org/sites/default/files/docs/tds/TDS_1_8_Standard_20140203.pdf

[46] Gudgin, M., Hadley, M., Mendelsohn, N., Lafon, Y., Moreau, J.-J., Karmarkar, A., und Nielsen, H. F., "SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," W3C, W3C Recommendation, April 2007, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/. [Online]. URL: http://www.w3.org/TR/2007/REC-soap12-part1-20070427/

[47] Hadlich, T., "Verwendung von Merkmalen im Engineering von Systemen," Dissertation, Otto von Guericke Universität Magdeburg, 2015. [Online]. URL: http://www.researchgate.net/profile/Thomas_Hadlich/publication/278301144_Verwendung_von_Merkmalen_im_Engineering_von_Systemen/links/557ea0b208aeea18b777d580.pdf

[48] Haffejee, M. und Brent, A. C., "Evaluation of an integrated asset life-cycle management (alcm) model and assessment of practices in the water utility sector," Water SA, Technical note Vol. 34 Nr. 2, 2008, iSSN 0378-4738.

[49] Heeg, M., "Ein Beitrag zur Modellierung von Merkmalen im Umfeld der Prozessleittechnik," Dissertation, RWTH Aachen, Dezember 2004.

[50] Hepp, M., Leukel, J., und Schmitz, V., "A quantitative analysis of product categorization standards: content, coverage, and maintenance of eCl@ss, UNSPSC, eOTD, and the RosettaNet Technical Dictionary," *Knowledge and Information Systems*, Vol. 13, S. 77 – 114, 2007.

[51] Höme, S., Grützner, J., Hadlich, T., Diedrich, C., Schnäpp, D., Arndt, S., und Schnieder, E., "Semantic Industry: Herausforderungen auf dem Weg zur rechnergestützten Informationsverarbeitung der Industrie 4.0," *at - Automatisierungstechnik*, Vol. 63, Nr. 2, S. 74–86, 2015.

[52] Common data dictionary. IEC International Electrotechnical Commission. [Online]. URL: http://std.iec.ch/iec61360. Abgerufen am: 22.10.2015

[53] *IEC/TR 62541-1OPC: Unified architecture - Part 1: Overview and concepts*, IEC International Electrotechnical Commission Std., February 2010.

[54] *IEC 62541-4: OPC Unified architecture - Part 4: Services*, IEC International Electrotechnical Commission Std., October 2011.

[55] *IEC 62264: Enterprise-control system integration*, IEC International Electrotechnical Commission Std., 5 2013.

[56] *IEC 62890: Life-cycle management for systems and products used in industrial-process measurement, control and automation*, IEC International Electrotechnical Commission Std., 10 2014.

[57] *IEC/TR 62541-1: OPC Unified architecture - Part 3: Address Space Model*, IEC International Electrotechnical Commission Std., 03 2015.

[58] *ISO 10303-1: Industrielle Automatisierungssysteme und Integration - Produktdatendarstellung und -austausch - Teil 1: Überblick und grundlegende Prinzipien*, ISO Internationale Organisation für Normung Std., 12 1994.

[59] *ISO 13584: Industrielle Automatisierungssysteme und Integration - Teilebibliothek*, ISO Internationale Organisation für Normung Std., Apr 2001.

[60] *ISO 15926-2: Industrielle Automatisierungssysteme und Integration - Integration von Lebens-Zyklusdaten für Prozeßanlagen einschließlich der Öl- und Gasproduktion - Teil 2: Datenmodell*, ISO Internationale Organisation für Normung Std., 12 2003.

[61] *ISO 15926-1: Industrielle Automatisierungssysteme und Integration - Integration von Lebens-Zyklusdaten für Prozeßanlagen einschließlich der Öl- und Gasproduktion - Teil 1: Überblick und Grundlagen*, ISO Internationale Organisation für Normung Std., 07 2004.

[62] *ISO/TS 15926-7: Industrielle Automatisierungssysteme und Integration - Integration von Lebens-Zyklusdaten für Prozeßanlagen einschließlich der Öl- und Gasproduktion - Teil 7: Implementierungsmethoden für die Integration verteilter Systeme: Vorlage für die Methodology*, ISO Internationale Organisation für Normung Std., 10 2011.

[63] *ISO/IEC 11179-3: Informationstechnik - Metadatenregistrierung (MDR) - Teil 3: Registrierungs-Metamodell und grundlegende Attribute*, ISO Internationale Organisation für Normung Std., Feb 2013.

[64] *ISO 55000: Asset management - Übersicht, Grundsätze und Begriffe*, ISO Internationale Organisation für Normung Std., 03 2014.

[65] *ISO/IEC/IEEE 12207: Systeme und Software-Engineering. Software Lebenszyklusprozesse*, ISO Internationale Organisation für Normung, IEC Internationale Elektrotechnische Kommission Std., 02 2008.

[66] *ISO/IEC TR 24748-1: System- und Software-Engineering - Lifecycle-Management - Teil 1: Leitfaden für Lifecycle-Management*, ISO Internationale Organisation für Normung, IEC Internationale Elektrotechnische Kommission Std., 10 2010.

[67] *ISO/IEC/IEEE 15288: System- und Software-Engineering - System-Lebenszyklus-Prozesse*, ISO Internationale Organisation für Normung, IEC Internationale Elektrotechnische Kommission, IEEE The Institute of Electrical and Electronics Engineers Std., 05 2015.

[68] *ISO/IEC/IEEE 15289: System und Software-Engineering - Inhalt von Lebenszyklus-Informationsprodukten (Dokumentation)*, ISO Internationale Organisation für Normung, IEC Internationale Elektrotechnische Kommission, IEEE The Institute of Electrical and Electronics Engineers Std., 05 2015.

[69] Kampert, D. und Epple, U., "Modeling Asset Information for Interoperable Software Systems," in *INDIN 2012: IEEE 10th International Conference on Industrial Informatics*, July 2012.

[70] Kampert, D. und Epple, U., "A Service Interface for Exchange of Property Information," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*. Piscataway, NJ: IEEE, 2013, S. 6920–6925.

[71] Kiritsis, D., Bufardi, A., und Xirouchakis, P., "Research issues on product lifecycle management and information tracking using smart embedded systems," *Advanced Engineering Informatics*, Vol. 17, Nr. 3–4, S. 189 – 202, 2003, intelligent Maintenance Systems. [Online]. URL: http://www.sciencedirect.com/science/article/pii/S1474034604000187

[72] Kiritsis, D., Rolstadås, A., und Moseng, B., "Promise final activity report," 507100 PROMISE - A Project of the 6th Framework Programme Information Society Technologies (IST), Tech. Rep., 2008, http://cordis.europa.eu/ist/st/projects.htm#promise - abgerufen am 06.04.2016.

[73] Leukel, J., Schmitz, V., und Dorloff, F.-D., "A modeling approach for product classification systems," in *Proceedings of the 13th International Workshop on Database and Expert Systems Applications*, 2002.

[74] MacKenzie, C. M., Laskey, K., McCabe, F., Brown, P. F., und Metz, R., "Reference Model for Service Oriented Architecture 1.0," OASIS, OASIS Standard, October 12th 2006, http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html. [Online]. URL: http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.html

[75] McCabe, F., Booth, D., Ferris, C., Orchard, D., Champion, M., Newcomer, E., und Haas, H., "Web Services Architecture," W3C, W3C Note, February 2004, http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/. [Online]. URL: http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/

[76] Mealling, R., "Request for comments: 5134 - a uniform resource name namespace for the EPCglobal Electronic Product Code (EPC) and related standards," Internet Engineering Task Force (IETF), Tech. Rep., 2008. [Online]. URL: https://tools.ietf.org/html/rfc5134

[77] Merschen, D., Duhr, Y., Ringler, T., Hedenetz, B., und Kowalewski, S., "Model-based analysis of design artefacts applying an annotation concept." in *Software Engineering*. Citeseer, 2012, S. 169–180.

[78] Mertens, M., "Verwaltung und Verarbeitung merkmalbasierter Informationen: vom Metamodell zur technischen Realisierung," Dissertation, RWTH-Aachen university, 2012.

[79] Mitchell, J. und Carlson, J., "Equipment asset management–what are the real requirements," *Reliability magazine*, Vol. 4, S. 14, 2001.

[80] Müller, J., "Automatisierung des Engineerings kommunikationsfähiger Anlagenkomponenten im Anlagennahen Asset Management in der Prozessindustrie," Dissertation, RWTH Aachen, Dezember 2007.

[81] N. N., "Git user manual," Tech. Rep. [Online]. URL: https://www.kernel.org/pub/software/scm/git/docs/user-manual.html

[82] N. N., "Promise architecture series - volume 2: Architecture reference: Promise core pac interface," The PROMISE Consortium, Tech. Rep., 02 2008, abgerufen am 22.04.2016, Benötigt Registrierung. [Online]. URL: http://cl2m.com/system/files/private/PROMISE%20AS%20Volume%202%20Architecture%20Reference%20Core%20PAC.pdf

[83] N. N., "Promise architecture series - volume 3: Architecture reference: Promise messaging interface (pmi)," The PROMISE Consortium, Tech.

Rep., 05 2008, abgerufen am 22.04.2016, Benötigt Registrierung. [Online]. URL: http://cl2m.com/system/files/private/PROMISE%20AS%20Volume%203%20Architecture%20Reference%20PMI.pdf

[84] N. N., "Promise architecture series - volume 4: Architecture reference: Promise pdkm system object model," The PROMISE Consortium, Tech. Rep., 06 2008, abgerufen am 22.04.2016, Benötigt Registrierung. [Online]. URL: http://cl2m.com/system/files/private/PROMISE%20AS%20Volume%201%20Architecture%20Overview%20V1.02.pdf

[85] N. N., "Life-Cycle-Management für Produkte und Systeme der Automation," ZVEI - Zentralverband Elektrotechnik- und Elektronikindustrie e. V., Fachverband Automation, Tech. Rep., 10 2010.

[86] N. N., "Industrie 4.0 - Gegenstände, Entitäten, Komponenten," VDI / VDE-Gesellschaft Mess- und Automatisierungstechnik, Statusreport, April 2014. [Online]. URL: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/VDI_Industrie_4.0_Komponenten_2014.pdf

[87] N. N., "VDI-Statusreport Industrie 4.0 - Wertschöpfungsketten," VDI / VDE-Gesellschaft Mess- und Automatisierungstechnik, Statusreport, April 2014. [Online]. URL: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/VDI_Industrie_4.0_Wertschoepfungsketten_2014.pdf

[88] N. N., "Referenzarchitekturmodell Industrie 4.0 (RAMI4.0)," VDI / VDE-Gesellschaft Mess- und Automatisierungstechnik, Statusreport, April 2015. [Online]. URL: https://www.vdi.de/fileadmin/vdi_de/redakteur_dateien/gma_dateien/Statusreport_Referenzmodelle_2015_v10_WEB.pdf

[89] Natis, Y., *Service-Oriented Architecture Scenario.* Gartner, 2003, iD Number: AV-19-6751.

[90] Niggemann, O. und Frey, C., "Data-driven anomaly detection in cyber-physical production systems," *at - Automatisierungstechnik*, Vol. 63, Nr. 10, S. 821 – 832, 2015.

[91] Oh, Y., hung Han, S., und Suh, H., "Mapping product structures between CAD and PDM systems using UML," *Computer-Aided Design*, Vol. 33, Nr. 7, S. 521 – 529, 2001. [Online]. URL: http://www.sciencedirect.com/science/article/pii/S0010448501000513

[92] Patil, L., Dutta, D., und Sriram, R., "Ontology-Based Exchange of Product Data Semantics," *IEEE Transactions on Automation Science and Engineering*, Vol. 2, Nr. 3, S. 213–225, 2005.

[93] Schlütter, M., Epple, U., und Edelmann, T., "Dienstesysteme für die Leittechnik – Ein Einblick," in *VDI-Berichte 2067, Automation 2009: Fit for Efficiency.* Düsseldorf: VDI Verlag, Jun. 2009, S. Kurzfassung: S. 21–24, Langfassung: auf beiliegender CD.

[94] Schrieber, R., Mühlhause, M., Wollschlaeger, M., Birkhofer, R., Niemann, J., Kalhoff, J., und Wickinger, J., "Generisches Lebenszyklusmodell für Produkte und Systeme der Automation," in *VDI-Berichte 2092, Automation 2010: Leading through Automation*. Düsseldorf: VDI Verlag, Jun. 2010.

[95] Schrieber, R., Wollschlaeger, M., Mühlhause, M., und Niemann, J., "Kompatibilität: Der zentrale schlüssel für nachhaltigkeit," *atp edition - Automatisierungstechnische Praxis*, Vol. 53, Nr. 11, S. 50, 2011.

[96] Schuman, C. A. und Brent, A. C., "Asset life cycle management: towards improving physical asset performance in the process industry," *International Journal of Operations & Production Management*, Vol. 25, Nr. 6, S. 566–579, 2005. [Online]. URL: http://dx.doi.org/10.1108/01443570510599728

[97] Sudarsan, R., Fenves, S., Sriram, R., und Wang, F., "A product information modeling framework for product lifecycle management," *Computer-Aided Design*, Vol. 37, Nr. 13, S. 1399 – 1411, 2005. [Online]. URL: http://www.sciencedirect.com/science/article/pii/S0010448505000400

[98] Thurlow, R., "Request for comments: 5531 - rpc: Remote procedure call protocol specification version 2," Internet Engineering Task Force (IETF), Tech. Rep., 2009. [Online]. URL: https://tools.ietf.org/html/rfc5531

[99] Usländer, T. und Epple, U., "Reference model of industrie 4.0 service architectures," *at - Automatisierungstechnik*, Vol. 63, Nr. 10, S. 858–866, 2015. [Online]. URL: http://www.degruyter.com/dg/viewarticle.fullcontentlink:pdfeventlink/$002fj$002fauto.2015.63.issue-10$002fauto-2015-0017$002fauto-2015-0017.pdf?format=INT&t:ac=j$002fauto.2015.63.issue-10$002fauto-2015-0017$002fauto-2015-0017.xml

[100] *VDI/VDE 3682: Formalisierte Prozessbeschreibungen*, Verein Deutscher Ingenieure Std., 5 2015.

[101] Wang, F., Fenves, S. J., Sudarsan, R., und Sriram, R. D., "Towards Modeling the Evolution of Product Families," in *ASME 2003 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. Volume 1: 23rd Computers and Information in Engineering Conference, Parts A and B*, Chicago, Illinois, USA, September 2003, S. 421–530.

[102] Xu, C., Gupta, S. K., Yao, Z., Gruninger, M., und Sriram, R., "Towards Computer-Aided Conceptual Design of Mechatronic Devices With Multiple Interaction-States," in *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, September 24–28, 2005, S. 455–467.

[103] Zha, X. F., Fenves, S. J., und Sriram, R. D., "A Feature-Based Approach to Embedded System Hardware and Software Co-Design," in *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, September 24–28, 2005, S. 609–620.

Online-Buchshop für Ingenieure

VDI nachrichten **BUCHSHOP**

Online-Shops

Fachliteratur und mehr - jetzt bequem online recherchieren & bestellen unter: www.vdi-nachrichten.com/Der-Shop-im-Ueberblick

Täglich aktualisiert: Neuerscheinungen VDI-Schriftenreihen

Im Buchshop von vdi-nachrichten.com finden Ingenieure und Techniker ein speziell auf sie zugeschnittenes, umfassendes Literaturangebot.

Mit der komfortablen Schnellsuche werden Sie in den VDI-Schriftenreihen und im Verzeichnis lieferbarer Bücher unter 1.000.000 Titeln garantiert fündig.

Im Buchshop stehen für Sie bereit:

VDI-Berichte und die Reihe **Kunststofftechnik:**

Berichte nationaler und internationaler technischer Fachtagungen der VDI-Fachgliederungen

Fortschritt-Berichte VDI:

Dissertationen, Habilitationen und Forschungsberichte aus sämtlichen ingenieurwissenschaftlichen Fachrichtungen

Newsletter „Neuerscheinungen":

Kostenfreie Infos zu aktuellen Titeln der VDI-Schriftenreihen bequem per E-Mail

Autoren-Service:

Umfassende Betreuung bei der Veröffentlichung Ihrer Arbeit in der Reihe Fortschritt-Berichte VDI

Buch- und Medien-Service:

Beschaffung aller am Markt verfügbaren Zeitschriften, Zeitungen, Fortsetzungsreihen, Handbücher, Technische Regelwerke, elektronische Medien und vieles mehr – einzeln oder im Abo und mit weltweitem Lieferservice

Die Reihen der Fortschritt-Berichte VDI:

1 Konstruktionstechnik/Maschinenelemente
2 Fertigungstechnik
3 Verfahrenstechnik
4 Bauingenieurwesen
5 Grund- und Werkstoffe/Kunststoffe
6 Energietechnik
7 Strömungstechnik
8 Mess-, Steuerungs- und Regelungstechnik
9 Elektronik/Mikro- und Nanotechnik
10 Informatik/Kommunikation
11 Schwingungstechnik
12 Verkehrstechnik/Fahrzeugtechnik
13 Fördertechnik/Logistik
14 Landtechnik/Lebensmitteltechnik
15 Umwelttechnik
16 Technik und Wirtschaft
17 Biotechnik/Medizintechnik
18 Mechanik/Bruchmechanik
19 Wärmetechnik/Kältetechnik
20 Rechnerunterstützte Verfahren (CAD, CAM, CAE CAQ, CIM ...)
21 Elektrotechnik
22 Mensch-Maschine-Systeme
23 Technische Gebäudeausrüstung

ISBN 978-3-18-5**25808**-4