Fortschritt-Berichte VDI

VDI

Reihe 8

Mess-,
Steuerungs- und
Regelungstechnik

Nr. 1245

Dipl.-Inform. Henning Mersch,
Bielefeld

Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik

Lehrstuhl für
Prozessleittechnik
der RWTH Aachen

Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik

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.-Inform.**

Henning Mersch

aus Bielefeld

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

Tag der mündlichen Prüfung: 26. November 2015

Fortschritt-Berichte VDI

Reihe 8

Mess-, Steuerungs- und Regelungstechnik

Dipl.-Inform. Henning Mersch, Bielefeld

Nr. 1245

Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik

Lehrstuhl für Prozessleittechnik der RWTH Aachen

Mersch, Henning
Deterministische, dynamische Systemstrukturen in der Automatisierungstechnik
Fortschr.-Ber. VDI Reihe 8 Nr. 1245. Düsseldorf: VDI Verlag 2016.
142 Seiten, 77 Bilder, 3 Tabellen.
ISBN 978-3-18-524508-4, ISSN 0178-9546,
€ 52,00/VDI-Mitgliederpreis € 46,80.

Für die Dokumentation: Anlagen – Automatisierungstechnik – Systemstruktur – Modelle – Verteilte Systeme – Dynamik – Nachvollziehbarkeit – Flexibilität – OPC-UA

Für die Weiterentwicklung der Automatisierungstechnik ist die erweiterte Zusammenarbeit der automatisierungstechnischen Geräte wichtig. Viele aktuelle Themen, wie „Industrie 4.0" oder „Cyber Physical Systems" gehen davon aus, dass Informationen aus dem Engineering zur Produktionszeit bereit stehen. In der Automatisierungstechnik wird dafür immer mehr angestrebt, Modelle zur Beschreibung der unterschiedlichsten Sachverhalte zu nutzen. Dabei werden Themen-spezifische Modelle entwickelt, die unabhängig voneinander eigene Blickwinkel der Automatisierungstechnik auf eine Anlage beschreiben. Die vorliegende Arbeit beschreibt Mittel, um diese Modelle zur Anwendung zu bringen, verzichtet dabei aber auf ein zentralistisches Modell: Existierende, heterogene Modelle werden durch eine verteilte, dynamische Ausführungsumgebung für Modelle und Dienste in der Automatisierungstechnik nutzbar gemacht, welche kollaborative Ansätze zur gemeinsamen Datenhaltung auf unterschiedlichen Geräten ermöglichen.

Bibliographische Information der Deutschen Bibliothek
Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliographie; detaillierte bibliographische Daten sind im Internet unter http://dnb.ddb.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 http://dnb.ddb.de.

D 82 (Diss. RWTH Aachen University, 2015)

ISSN 0178-9546
ISBN 978-3-18-524508-4

Vorwort

Diese Dissertation entstand durch meine Tätigkeit als wissenschaftlicher Angestellter am Lehrstuhl für Prozessleittechnik der RWTH Aachen. An dieser Stelle möchte ich mich bei denen bedanken, die zum Gelingen der Arbeit beigetragen haben.

Mein besonderer Dank gilt Herrn Professor Dr.-Ing. Ulrich Epple. Die von ihm am Lehrstuhl geschaffene, offene und angenehme Arbeitsatmosphäre in einer konstruktiven und gut ausgestatteten Umgebung sind der Ausgangspunkt für diese Arbeit. Die fachlichen und teilweise auch kontroversen Diskussionen waren immer erfrischend und inspirierend für weitere Arbeiten.
Ebenso bedanke ich mich bei Herrn Professor Dr.-Ing. habil. Martin Wollschlaeger, Inhaber der Professur Prozesskommunikation der Technischen Universität Dresden, für die freundliche Übernahme der Rolle des Zweitgutachters.

Für intensive Diskussionen und die kooperative Arbeitsatmosphäre danke ich weiterhin meinen ehemaligen Kollegen, den Mitarbeiterinnen und Mitarbeitern des Lehrstuhls. Besonders erwähnen möchte ich hier Reiner Jorewitz, Martin Mertens, Gustavo Quirós und Markus Schlüter

Für die vielen organisatorischen Arbeiten gilt Martina Uecker und im Sekretariat Frau Bey der besondere Dank. Nicht unerwähnt bleiben sollen auch die vielen studentischen Hilfskräfte, die viele Tätigkeiten erst umsetzbar machen.

Weiterhin möchte ich gerne Leon Urbas, Andreas Gössling, Christian Kleegrewe und Wolfgang Mahnke danken, die immer wieder erfrischende Gedanken in privaten Gesprächen oder der gemeinsamen VDI-GMA Fachausschussarbeit geweckt haben.

Meiner Frau Tina danke ich von ganzem Herzen für die Geduld und die Zeit sowie den fachlichen Diskussionen und lektographischen Anmerkungen. Unserem Sohn Liam muss ich für so manche Stunde danken, die er schon in den frühen Jahren in Geduld üben musste.

Schließlich danke ich meinen Eltern Birgit und Werner Mersch, die mich zu diesem beruflichen und privaten Weg geführt haben und in vielen Situationen zum Gelingen dieser Arbeit beigetragen haben.

Verl, im Dezember 2015 *Henning Mersch*

»So you do what you do best. And you link to the rest.«
Jeff Jarvis, 22. Februar 2007 [Jar]

Inhaltsverzeichnis

Kurzfassung

Für die Weiterentwicklung der Automatisierungstechnik ist die erweiterte Zusammenarbeit der automatisierungstechnischen Geräte wichtig. Dieses gilt für alle Phasen einer Anlage: Von der Planung über die Produktion bis zur Wartung. Ebenso auch für die horizontale und vertikale Integration während der Produktion. Viele aktuelle Themen, wie „Industrie 4.0“ oder „Cyber Physical Systems“ gehen davon aus, dass Informationen aus dem Engineering zur Produktionszeit bereit stehen. Hierzu leistet diese Arbeit einen Beitrag.

Informationen werden heutzutage noch häufig entweder nicht elektronisch auswertbar gespeichert (beispielsweise als Grafiken) oder sind so abgelegt, dass nur einzelne Programme auf sie zugreifen können. Hierdurch sind die existierenden Informationen nicht so weit zugreifbar, wie sie es eigentlich sein könnten.

Modelle spielen hierbei eine entscheidende Rolle: Sie beschreiben Sachverhalte der Anlagen. Die meisten der heutigen Modelle werden bei ihrer Spezifikation in einem elektronisch abbild- und auswertbaren Format definiert, sodass ein Computer die Informationen sowohl bereitstellen, wie auch auswerten und damit nutzen kann. Werden diese Modelle zur Produktionszeit bereitgestellt und genutzt, werden hierdurch dynamische Änderungen ermöglicht, die heutzutage nicht üblich sind. In der Automatisierungstechnik wird deswegen immer mehr angestrebt, Modelle zur Beschreibung der unterschiedlichsten Sachverhalte zu nutzen. Modelle beschreiben unter anderem Systemstrukturen einer Anlage. Dabei werden Themen-spezifische Modelle entwickelt, die unabhängig voneinander jeweils eigene Blickwinkel der Automatisierungstechnik auf eine Anlage beschreiben.
Im Gegensatz dazu wurden Versuche, bei denen eine Domäne als Ganzes (wie beispielsweise Automatisierungstechnik) abgebildet werden soll, nicht von Erfolg gekrönt.
Die Modelle konnten sich beispielsweise nicht etablieren, weil eine Verbreitung nicht erreicht wurde. Dieses mag insbesondere daran gelegen haben, dass die jeweiligen Detaillierung von umfassenden Modellen (sogenannten „Welt-Modellen“) für spezifische Anwendungsfälle nicht ausreichend waren. Da diese Modelle in dem Fall nicht eingesetzt werden konnten, wurde wiederum auf Eigenentwicklungen gesetzt, was den Bestrebungen des Welt-Modells widersprach.
Die vorliegende Arbeit beschreibt Mittel, um die gleichen Ziele zu erreichen, aber auf ein zentralistisches Modell zu verzichten: Existierende, heterogene Modelle werden in allen Phasen und Ebenen einer verteilten Umgebung, wie in einem automatisierungstechnischen System, nutzbar gemacht.
Hierzu wird zum einen eine verteilte, dynamische *modellgetriebene Instanzumgebung* beschrieben. Sie kann als Erweiterung von existierenden Technologien begriffen werden, wird aber unabhängig von diesen dargestellt. Durch diese modellgetriebene Instanzumgebung wird erreicht,

Teile eines Modells auf unterschiedlichen Geräten bereitzustellen. Diese sind in einer einheitlichen Weise abfragbar und erkundbar. Hierdurch können die im Modell abgebildeten Systemstrukturen an einem „sinnvollen“ Ort abgelegt werden, der nach Kriterien, wie der häufigsten Nutzung, der höchsten Ausfallsicherheit oder der schnellsten Verfügbarkeit beim Zugriff erfolgen kann. Gleichzeitig bietet eine solche Ausführungsumgebung der Modelle die Möglichkeit dynamisch auf Änderungen zu reagieren: kollaborativ erfolgen Änderungen von unterschiedlichen Anwendungen. Dabei muss jedoch insbesondere die Transaktionssicherheit sowie Nachvollziehbarkeit (Determinismus) der Änderungen gegeben sein.
Zum anderen beschreibt die Arbeit ein Konzept zur *Interkonnektion* von Modellen. Hierbei werden Teile von Modellen, die unabhängig voneinander entworfen wurden, in Relation zueinander gesetzt. Interkonnektionen stellen dabei eine Form von *Relationen* dar, die jedoch keinerlei Abhängigkeit an Ausgangs- sowie Zielpunkt voraussetzen. Dieses erlaubt die Modellierung von zusätzlichen Sachverhalten, sodass die Interkonnektion von Modellen wiederum ein Modell ist.

Durch die Kombination dieser beiden Aspekte ergibt sich eine verteilte, deterministische und dynamische Ausführungsumgebung für Systemstrukturen. Voraussetzung ist ein gemeinsames Meta-Modell sowie Verständnis der Problematik. Als Folge können Modelle unabhängig voneinander entworfen werden. Eine solche Ausführungsumgebung muss dabei Schnittstellen bereitstellen, um die Informationen abzufragen und entsprechende Änderungen vorzunehmen. Erst hierdurch können aufbauende Anwendungen einen realen Nutzen aus den Konzepten ziehen.

Insgesamt ergibt sich so eine Lösung, um Modelle zu den unterschiedlichen Phasen und Ebenen einer Anlage - insbesondere auch zur Produktionszeit - zu nutzen. Die Integration der verteilten, unterschiedlichen Modelle beschreibt die informationstechnische Basis, um dynamisch auf Änderungen in den Systemstrukturen zu reagieren. Hierunter werden beispielsweise Änderungen der Anlagenstruktur durch Umbauten ebenso verstanden, wie unterschiedliche Produktionsaufträge.

Abstract

For the future development of automation technology the enhanced collaboration of automation devices is important. This is true for all phases of a plant, from planning to production to maintenance, as well as for the horizontal and vertical integration during the production. A lot of current topics like “Industry 4.0” or “Cyber Physical Systems” act on the assumption that engineering information is available during the production phase, which is not the case today. The dissertation addresses this topic.

Nowadays information is often stored not electronic analyzable (e.g. as graphic) or accessible only by single programs. So existing information is not accessible as far as it could be.

Models are acting as an important part: They describe characteristics of a plant. Most of today’s models are defined in an electronically representable and evaluable format by their specification. So a computer could host and provide these models as well as evaluate and use them. If they are provided and used during the production phase, dynamical changes are made possible, which is not usually the case nowadays.

Therefore, in automation technology models are used for the description of different topics. Topic-specific models are developed, that are independent from each other and describe different aspects of the domain of automation technology.
In contrast attempts to describe the whole domain of automation technology in one model were not successful. Those models could not be widely established since their adoption could not be achieved. This could be due to the fact that models describing a whole domain (“World-Models”) are not detailed enough to be used for specific cases. So specific models were required to be defined, which contradicts the purpose of whole domain model.

This work describes instruments without a centralized model: Existing, heterogeneous models can be used in all phases and levels of a distributed system like a plant in a homogenous way. Therefore a distributed, dynamic, model-driven execution environment is described. This could be seen as a further development of existing technologies, but is described independent of those. In this model-driven execution environment parts of a model could be provided by distributed devices. In a common way models are discover- and query-able. Therefore, information could be stored in a judicious place, that is defined by most frequent usage, highest reliability or fastest availability. At the same time such a model-driven execution environment provides for the possibility of dynamic changes: Changes are established in a collaborative way from different applications. For that purpose transaction security as well as comprehensibility (determinism) needs to be assured.

Additionally, this work describes the concept of *interconnections* of models: Parts of models, which are designed independent of each other, are put in relation. Interconnections are a special type of *relations* not having dependencies at start- or endpoint. This enables modeling of additional aspects, so interconnections of models are models again.

This combination of instruments represents a distributed, deterministic and dynamic model execution environment of system structures. A requirement for this is a common used metamodel as well as a complete understanding of the topic. Models can be specified independent of each other. A model execution environment will provide interfaces for querying information and for making changes to all models. Applications will be based on this.

The overall result is a solution, which makes the use of models feasible during all phases of a plant - especially during production time. The integration of the distributed models provides an information-technology foundation for dynamic changes on system architecture. This covers changes of plants due to rebuilding as well as production orders.

1 Einleitung

Pläne sind nichts, Planung ist alles.
Dwight D. Eisenhower

Vielfach wird über Modellierung in der Automatisierungstechnik berichtet. Dabei werden Einzelaspekte der Automatisierungstechnik bzw. der zu automatisierenden Anlagen durch Modelle beschrieben. Im Wesentlichen hat dieses drei Gründe, die sich nicht gegenseitig ausschließen:

Verständnisbildung Im Bereich der Forschung wird versucht die Formalisierung von Sachverhalten durch Modelle zu erreichen. Ziel ist dabei eher das Verständnis, sodass sich Folgerungen aus gewonnenen Zusammenhängen für neuartige Technologien erschließen.

Datenaustausch In Standardisierungsgremien geht es um die gemeinsame Darstellung von Sachverhalten, die zwischen Komponenten (elektronisch) ausgetauscht werden sollen.

Datenhaltung Modelle werden zur kontinuierlichen Abbildung der Realität formuliert: Aspekte der Realität werden im Modell festgehalten, um sie durch Computer-Programme abfragbar und veränderbar zu machen.

In der Praxis zeigen insbesondere die Modelle zum Datenaustausch meist einen hohen Detailierungsgrad. Dabei sind sie jedoch als heterogen zu bezeichnen, da sie keinerlei gemeinsame Elemente enthalten. Es gibt jedoch immer wiederkehrende Konstrukte, die sich teilweise nach und nach als eine Art Meta-Modell etabliert haben.

Meistens sind es spezielle Lösungen, damit beispielsweise zwei Anwendungen miteinander Daten austauschen können. Das Potenzial der Modelle wird so nur unzureichend genutzt.

Die vorliegende Arbeit soll die Potenziale der heterogenen Modelle weitreichend erschließbar machen. Dabei wird vorgeschlagen, eine Instanzumgebung zu etablieren, die von unterschiedlichen Anwendungen kollaborativ genutzt werden kann. Verschiedene Modelle werden in der Instanzumgebung repräsentiert und sind so zugreifbar. Drei Konzepte liegen der Arbeit zugrunde:

1. Die Modelle selber können aufgrund des Meta-Modells der Modellverwaltungsumgebung verteilt in der Anlagenstruktur (d.h. in der Engineering-Phase genau wie in der Produktions-Phase) abgelegt werden.

2. Die Modelle können, ohne selber eine Grundlage dafür bereitstellen zu müssen, untereinander in Relation gesetzt werden. Diese Interkonnektionen zwischen Modellen ist dabei selber wieder ein Modell, welches auf mindestens zwei Ausgangs-Modellen basiert.

3. Änderungen an den Modellen können automatisiert als Änderungen von Interkonnektionen formuliert werden, sodass diese Interkonnektionen einen wirklichen Mehrwert gegenüber den Einzelmodellen darstellen.

Abbildung 1.1 verdeutlicht diese Denkweise. Dabei wird besonders veranschaulicht, dass die Anwendungen von der möglichen Verteilung möglichst abstrahiert entwickelt werden sollten.

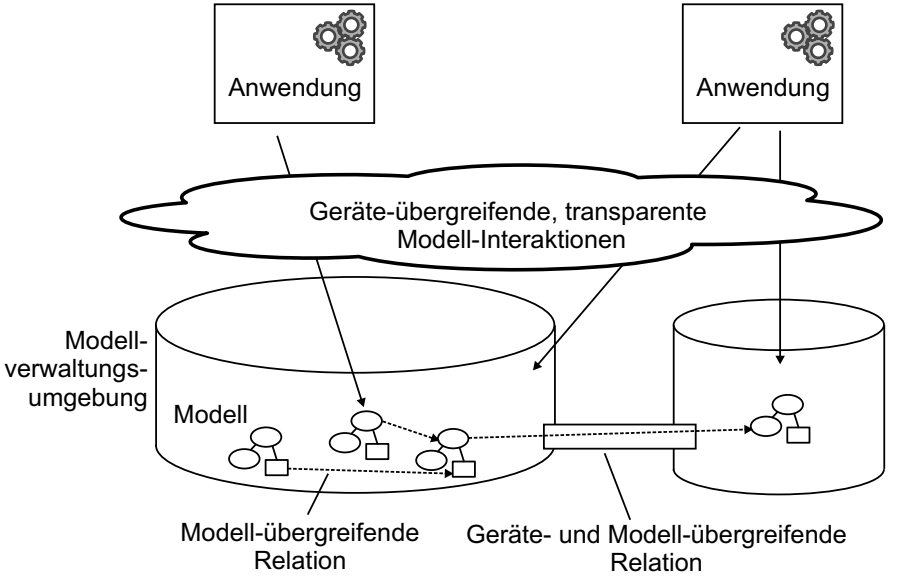


Abbildung 1.1: Konzeptuelles Schema der modellgetriebenen, verteilten Modellverwaltungsumgebung

Durch eine verteile Instanzumgebung, in der unterschiedliche Modelle veränderbar hinterlegt werden können, werden neue Anwendungsbereiche erschlossen.

Der Ansatz geht davon aus, dass die Anwendungen möglichst viele Daten als Modell in der Instanzumgebung ablegen und im Idealfall nur das Verhalten im Programm selber liegt. Diese Teilung von aktivem Verhalten und passiven Daten entspricht dabei der üblichen Denkweise eines Programmierers und Ingenieurs in der Projektierung.

Strukturänderungen in der Anlage können zur Laufzeit von nutzenden Anwendungen erkannt werden, wenn sie entsprechend durch Modelle abgebildet sind und diese angepasst werden. Diese Änderungsinformationen liegen in der Instanzumgebung nicht nur als Änderungsmitteilung vor, sondern sind aufgrund der abgebildeten Systemstruktur direkt analysierbar.

Die Folgen aus einer solchen Instanzumgebung - wenn sie eine entsprechende Verbreitung erreicht hat - adressieren zum einen modellbasiertes Engineering, wie es auch in [CHF14] analysiert wird. Aber auch Aspekte in der Produktionsphase wie Wandlungsfähigkeit von Anlagen werden ermöglicht, wie es durch einen „desktriptiven Ansatz“ der Automatisierung in [Nig14] beschrieben wird.

Zusammenfassend kann also formuliert werden, dass eine Vielzahl von Modellen für die Automatisierungstechnik entwickelt wurden und werden. Diese Modelle entstehen zwar aus unterschiedlicher Motivation heraus, könnten aber immer für die Abbildung von Spezifika der Automatisierungstechnik verwendet werden. Heute wird dieses Potenzial viel zu wenig genutzt. Eine Bereitstellung einer Instanzumgebung für diese Modelle, die unabhängig von konkreten Anwendungen einen kollaborativen Zugriff erlaubt, birgt erhebliche Potenziale.
Das gilt umso mehr, wenn die Anwendungen als aktive Teile der Modelle verstanden werden und wie die Modelle in einer Anlage verteilt verwaltet werden können.

Im Folgenden werden die Ziele der Arbeit konkret formuliert und entstehende Potenziale vorgestellt.

1.1 Ziele und Vision

Vision – *Die effizientere und geschicktere Nutzung von bereits existierenden Informationen ermöglicht neuartige Anwendungen in den unterschiedlichen Phasen und Ebenen einer Anlage.*

Aus der Vision lässt sich ableiten, dass Mechanismen zur Informationsbereitstellung entwickelt werden müssen. Diese sollten dabei sowohl für die unterschiedlichen Phasen einer Anlage (vgl. [19]) wie auch für die unterschiedlichen Ebenen nach der Automatisierungspyramide [Pol94] einsetzbar sein. Modelle werden, wie auch in aktueller Lehrliteratur [Urb12] beschrieben, heutzutage bereits verwendet, jedoch bei weitem nicht so effektiv, wie es sein könnte.

Es lassen sich somit drei Ziele formulieren, um die Bereitstellung der Informationen zu charakterisieren:

Ziel 1 – *Verbesserung der horizontalen Integration:*
Erschließung neuer Diagnose-, Analyse- und Reaktions-Möglichkeiten in der Produktions-Phase

Klassischerweise wird von *horizontaler Integration* gesprochen, wenn während der Produktion innerhalb einer Ebene der Automatisierungspyramide eine Integration verstärkt wird.
Durch das Auffinden von Geräten und einer Kommunikation ohne weitere Konfiguration, kann beispielsweise auf Sensor/Aktor-Ebene eine erweiterten Diagnose- und Analyse-Möglichkeit geschaffen werden.

Ziel 2 – *Verbesserung der vertikalen Integration:*
Nutzung gemeinsamer Dienste und Modelle für äquivalente Aufgaben

Es wird von *vertikaler Integration* gesprochen, wenn eine Interaktion zwischen mehreren Ebenen angestrebt wird.
Beispielsweise ist hier die standardisierte Bereitstellung von Messdaten vom Sensor direkt in die MES oder ERP Ebene zu nennen. Aber auch die gemeinsame Nutzung von Funktionen (wie z.B. die „Allgemeinen Systemdienste“ nach [Pol94]) von unterschiedlichen Ebenen kann eine Verbesserung erzielen. Ein einheitlicher „Meldedienst“ beispielsweise kann für die Archivierung von Messdaten ebenso geeignet sein, wie für die Protokollierung von Buchungsanfragen auf ERP Ebene [SME10].

Ziel 3 – *Verbesserung der zeitlichen Integration (entlang des Anlagenlebenszyklus):*
Engineering-Informationen auch während der Produktions-Phase.

Als *zeitliche Integration* kann verstanden werden, dass Informationen aus der Engineer-ing-Phase auch zur Produktionszeit (und in weiteren Phasen) nutzbar sind und damit auch für Änderungen an der bereits in Betrieb genommenen Anlage bereitstehen.
Beispielsweise ist hier die (teil-)automatisierte Inbetriebnahme von Anlagen-Modulen, die für unterschiedliche Produkte neu angeordnet werden müssen, zu nennen.

1.2 Konzepte der verteilten, modellgetriebenen Instanzumgebung

Zur Errichtung und zum Betrieb einer produktionstechnischen Anlage gehört heutzutage eine große Anzahl von Geräten (Sensoren, Aktoren, Engineering-, BuB-Station, ...) und Programmen (Automatisierungsausführung, Visualisierung, Verwaltung, Protokollierung, ...), die auf teilweise spezialisierten Geräten ausgeführt werden. Für sich genommen hat heutzutage jedes Gerät und jedes Programm seine spezielle, legitime Aufgabe. Diese Trennung führt zu separaten Datensätze, die voneinander unabhängig sind und durch andere Programme nicht zugreifbar sind. Eine spätere Verknüpfung der Daten ist aufwändig und sollte, soweit es geht, vermieden werden.

Im Folgenden sind Konzepte beschrieben, die im späteren Verlauf der Arbeit aufgegriffen werden:

Hoheit über Daten aufgeben Die hier vorgestellten Konzepte sehen vor, dass die Programme die Hoheit über ihre Daten aufgeben und möglichst weitreichend in Form von Modellen in einer zu beschreibenden Instanzumgebung ablegen, sodass sie von anderen genutzt werden können
Dafür ist aus Vorarbeiten bekannt, dass ein gemeinsames - möglichst schmales - Meta-Modell genutzt werden kann. Entsprechende Modelle konkretisieren das Meta-Modell.

Somit können Elemente der Modelle in einer Instanzumgebung instanziiert werden. Bietet die Instanzumgebung über das Meta-Modell definierte Manipulations- und Erkundungs-Möglichkeiten, können zu einem Zeitpunkt mehrere Modelle verwaltet und verändert werden. Da Informationen als Objekte von Modellen abgebildet sind, können diese durch spezielle Komponenten in Relation zwischen den Modellen gesetzt werden, wodurch letztendlich eine Struktur entsteht, die für andere Anwendungen erkundbar ist.

Verteilung der Daten Ein automatisierungstechnisches System wird als verteiltes System gesehen. Viele unterschiedliche Teilnehmer in einem Netzwerk operieren aktuell mit lokalen Daten. Um einen einheitlichen Modell-Raum zu erhalten, ist es wichtig die Instanzumgebung mit Verteilungsmöglichkeiten auszustatten. Somit kann jede Anwendung ihre Daten lokal bearbeiten und verwalten. Entsprechende Relationen über die Systemgrenzen hinweg sorgen für die Verknüpfungen.

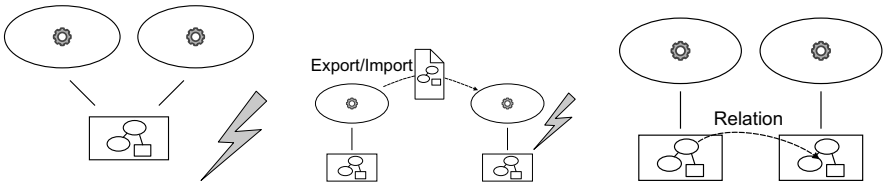
Im Folgenden werden Argumente dargestellt, die Potenziale und Möglichkeiten einer *verteilten, modellgetriebenen Instanzumgebung* charakterisieren und so die Vorteile aufzeigen.

„Standardisierung" als Chance Eine Möglichkeit der vereinheitlichten Darstellung der Daten ist die Standardisierung. Die IT-Branche verzichtet (teilweise) hierauf und arbeitet mit unscharfen Begriffen. Selbst Quasi-Normen werden nicht als solche bezeichnet: Die W3C verwaltet „Standards" mit weltweiter Bedeutung, wie beispielsweise das *http*-Protokoll. Sie nennt Ihre Dokumente „Recommendation", also Empfehlungen. Viele Vereinbarungen für Unix Systeme sind in „Request for comment" (RFC) festgehalten[1].
Auch für die Ingenieurswissenschaften stellt sich deswegen die Frage, ob die klassischen Standards in allen Themen die richtige Möglichkeit sind. Oder ob es nicht sinnvoller ist, dass jede Anwendung und jeder Hersteller eigene Modelle und Repräsentationen nutzt, diese jedoch in einem öffentlich zugänglichen System, damit eine Integration der Daten ermöglicht wird. Jeder Hersteller ist frei genau die Daten offen - im Sinne von zugreifbar - zu legen, die er für sinnvoll erachtet.

Alle diese Abbildungen würden in einer modellgetriebenen Instanzumgebung als Modelle verstanden. Die einzelnen Datensätze könnten nachträglich von allen Teilnehmern ausgewertet und auch verknüpft werden.

Unterbindung von Inkonsistenzen Heutige Engineering-Software in der Automatisierungstechnik speichert Daten in eigenen Formaten, sodass ein Zugriff von anderen Anwendungen nicht möglich ist. Da die Anwendungen jedoch Daten untereinander austauschen müssen, werden Schnittstellen geschaffen.

[1] Noch weiter gehen Initiatoren der OpenData-Projekte: Hier werden Daten grundsätzlich zwar strukturiert und Maschinenverarbeitbar angeboten, jedoch wird bewusst auf vorgeschriebene Standards verzichtet. Hauptsächlich sollen Behörden und öffentliche Einrichtungen dazu gebracht werden, möglichst vollkommen ihre Daten einer breiten Öffentlichkeit anzubieten. Also beispielsweise das Gewerbeamt eine Liste der lokalen Firmen, das Katasteramt die Straßennamen oder auch die Einwohnermeldeämter statistische Kennzahlen der gemeldeten Einwohner.
Der Zugriff und die Aggregation der Daten (und damit eine Form von „Informationsgenerierung") liegt folglich bei den Nutzenden, wie beispielsweise den Journalisten.
Durch den Verzicht auf feste Standards für die Daten wird die Teilnahme aus Sicht der Behörden erheblich vereinfacht.



(a) Ein Modell für alle Anwendungen: nicht beschreibbar

(b) Export/Import durch Datei: Inkonsistenz durch Kopie

(c) Modell-Interkonnektion mit gemeinsamem Meta-Modell

Abbildung 1.2: Datenaustausch zwischen Anwendungen

Die erste Möglichkeit ist eine gemeinsame Datenbasis zu definieren; also ein gemeinsames Datenmodell, welches von den beteiligten Anwendungen genutzt wird. Bei vielen Anwendungen wird das Modell sehr komplex, sodass die Akzeptanz des Modells wiederum sinkt. (Abbildung 1.2 (a)). Eine andere, weit verbreitete Möglichkeit ist ein Austauschformat zu definieren. Dabei exportiert eine Anwendung in einem definierten Modell (Format) die benötigten Daten; eine weitere Anwendung liest diese Daten ein. Nachher existiert also eine Kopie der Daten, sodass Inkonsistenzen durch die unterschiedliche Weiterverarbeitung entstehen. Diese müssen entweder aufwändig synchronisiert werden, oder es muss einen unidirektionalen Arbeitsfluss geben, sodass dieser Prozess nur einmalig stattfindet (Abbildung 1.2 (b)).

Sinnvoller und in dieser Arbeit angestrebt ist es, Verbindungen zwischen einzelnen Datensätzen der Anwendungen zu schaffen (Abbildung 1.2 (c)). Hierbei verbleiben die Daten in dem Modell der Anwendungen, jedoch sind Elemente davon zugänglich und können zwischen den Modellen in Relation gesetzt werden, wodurch auch ein Zugriff stattfinden kann. Ein gemeinsames Instanzsystem ersetzt durch den gemeinsamen Datenraum den bisherigen Datenaustausch.

Analogie: Strukturierte Festplatte Insgesamt kann so eine Analogie zu einer strukturierten Festplatte gezogen werden: Traditionell speichern alle Anwendungen in ihrem eigenen Format Dateien auf die lokale Festplatte. In großen Installationen ist eine verteilte Lösung in Form eines Netzwerk-Laufwerks oder NFS Systems üblich - allerdings hauptsächlich aus Gründen der Daten-Sicherung (Backup) und für die flexible Nutzung von Arbeitsplätzen.
Die abgelegten Daten sind in ihren proprietären Formaten als Datei gespeichert. Eine Verbindung zwischen einzelnen (Modell-)Teilen ist nicht möglich.

Die beschriebene Instanzumgebung bietet eine Möglichkeit, dieses Problem zu umgehen. Daten werden durch Anwendungen in die Instanzumgebung geschrieben, ähnlich wie in eine Datei. Durch die Zugriffsmechanismen ist es anderen Anwendungen aber möglich in diese Daten hineinzublicken und Beziehungen zu den Teildaten aufzubauen. Ebenso können Rückschlüsse aus den Daten oder den Änderungen der Daten gezogen werden.

Dienste – ohne Schnittstellenproblematik Vielfach wurde in den vergangen Jahren über eine Dienstorientierung auf Basis von „Dienst-orientierten Architekturen“ (kurz: SOA für „Service-oriented Architecture“; beispielsweise [SEE09]) gesprochen. Ziel ist es, Funktionalität in Dienste zu kapseln, sodass diese als „Blackbox“ ihre Aufgabe verrichten. Der interne Aufbau ist somit für den Nutzenden nicht wichtig.
Kern des Konzeptes ist es also, dass unabhängige Software-Komponenten über Schnittstellen sich gegenseitig aufrufen. Die Orchestrierung (also Verknüpfung) von Diensten führt dann zu höherwertigen Anwendungen, die eine Gesamtaufgabe erledigen.

Diese Dienst-basierte Architektur bedingt dabei wesentliche Standardisierung: Schnittstellen müssen auf mehreren Ebenen (von binärer Kodierung über Ausführungslogik bis hin zu den Aufruf- und Rückgabe-Parametern) spezifiziert werden. Zusätzlich muss in der Automatisierungstechnik bedacht werden, dass zu den Standardisierungsarbeiten auch noch unterschiedliche Kommunikations-Medien mit unterschiedlichen Eigenschaften berücksichtigt werden müssen.

Anwendungen in logischer Nähe der Instanzumgebung Bisher wurden nur in Modellen abgelegte „passive“ Informationen betrachtet. Wenn Informationen konzeptuell in der modellgetriebenen Instanzumgebung abgelegt sind, liegt es nahe, dass auch für die Anwendungen selber eine Ausführungsumgebung spezifiziert wird, die auf den gleichen Mechanismen beruht. Der Vorteil ist, dass ein einheitliches Konzept zur Kommunikation zwischen Anwendungen etabliert werden kann. Auch wird sich hier anbieten, die Anwendungen in einzelne Komponenten, die zwischen einander kommunizieren, abzubilden. Eine solche Ausführungsumgebung wird sich erstmal nicht wesentlich von den Dienstorientierten Architekturen unterscheiden, wie sie schon vielfach auch für die Automatisierungstechnik untersucht werden.

Aus diesem Grund beschränkt sich diese Arbeit auf die Möglichkeiten der modellgetriebenen Instanzumgebung und stellt die Möglichkeit der integrierten Ausführungsumgebung am Rande vor. Näher betrachtet wird allerdings die Möglichkeit aktive Komponenten (ob innerhalb einer Ausführungsumgebung oder außerhalb ausgeführt) in der Modell-Landschaft abzubilden (zu repräsentieren) und dadurch eine Erkundungsfunktion zu schaffen. Dieses ist eine der wesentlichen Herausforderungen (genannt „Discovery“), die die SOA zu meistern hat.

Dynamik und Determinismus zur Produktions-Phase Die oben beschrieben Informationen in der modellgetriebenen Instanzumgebung können nicht nur in der Planungs- und Engineering-Phase genutzt werden, sondern auch zur Laufzeit.

Erst durch eine gegenseitige Überwachung der Informations-Änderungen und Reaktion auf diese Änderungen ist eine konsistente Gesamterfassung der Informationen möglich. Es entsteht also eine *dynamische* Modelllandschaft, die sich zur Laufzeit an Änderungen anpassen kann.

Diese Änderungen können von beliebiger Struktur sein. Hierbei ist für die Akzeptanz insbesondere wichtig, dass die Reaktionen auf die Änderungen deterministisch sind, d.h. entscheidbar und auch nachvollziehbar: Wird unter den gleichen Umständen die gleiche Änderung gemacht, wird das System mit der gleichen Reaktion antworten.

Auf Basis dieser dynamischen Modell-Änderungen zusammen mit dem Determinismus wird es möglich, die „Self-X“ Konzepte, wie sie von IBM [Hor01] beschrieben wurden, in einer Weise zu definieren, dass sie auch in der Automatisierungstechnik Anwendung finden können.

Für das Erreichen der Vision ist es dabei wichtig, dass die Schnittstellen insbesondere zur modellgetriebenen Instanzumgebung frei verfügbar sind. Nur hierdurch können Spezialanwendungen beispielsweise von kleineren Engineering-Firmen zusätzlich integriert werden. Dieses hilft im Umkehrschluss auch den großen Herstellern, da für sie uninteressante Spezialanwendungen durch externe Dienstleister erbracht werden können.

1.3 Übersicht des Vorgehens

Nachdem die Einleitung eine grundlegende Motivation und Einordnung der Problematik vorgenommen hat, wird im Folgenden der Stand der Technik dargestellt (Kapitel 2). Dabei werden zum einen Begriffe und Konzepte beschrieben, die im Kontext der Arbeit wichtig sind. Hierzu zählt insbesondere das Abbilden von Informationen in Modellen. Da eine Interaktion von unterschiedlichen Software-Systemen unabdingbar ist, wird auch die Kommunikation betrachtet. Bevor neuartige Konzepte dargestellt werden, erfolgt aufgrund der Ausgangssituation eine Analyse der Anforderungen (Kapitel 3), die zur Umsetzung der Eingangs beschriebenen Ziele und Vision berücksichtigt werden müssen. Zur Umsetzung dieser Anforderungen sind Weiterentwicklungen und neuartige Konzepte notwendig. Diese beziehen sich zum einen auf die Verteilung der Modelle auf unterschiedliche Geräte. Zum anderen sollen existierende, Modell-artig abgebildete Informationen in Zukunft in Verbindung gesetzt werden (Kapitel 4). Um diese Konzepte vor dem Hintergrund der Verteilung anwendbar zu machen, werden Software-Komponenten beschrieben, die entsprechende Schnittstellen zur Abstraktion bereitstellen (Kapitel 5). Sie beziehen sich dabei zum einen auf die Verteilung und zum anderen wiederum auf die Verbindungen zwischen den Informationen, also die Suche über den Modell-Raum. Mit Hilfe dieser Basis-Komponenten können Anwendungen skizziert werden, die die Konzepte nutzen (Kapitel 6). Anwendungsbeispiele verdeutlichen abschließend die Potenziale der aufgezeigten Konzepte und Ziele.
Als Überblick sind die wichtigsten Begriffe in der Abbildung 1.3 als Stichpunkte dargestellt. Dabei wurde eine Einordnung in die jeweiligen Kapitel vorgenommen. Auch die Anordnung innerhalb der Abschnitte weißt auf Zusammenhänge zu den überliegenden oder unterliegenden Schichten hin.

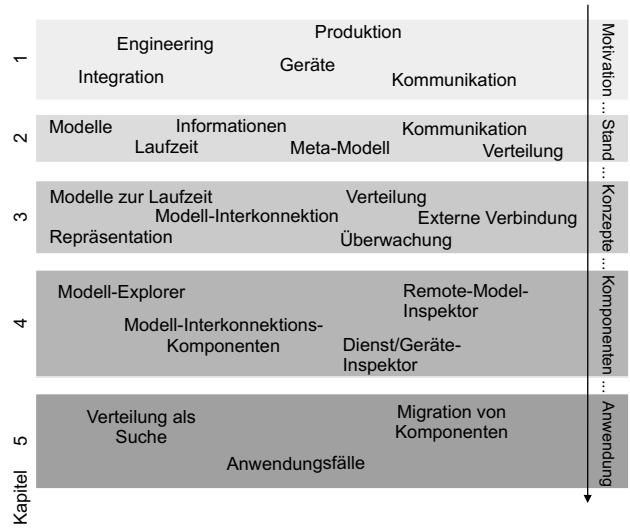


Abbildung 1.3: Übersicht der wichtigsten Begriffe im Kontext dieser Arbeit

Notationen in der Arbeit:

- Einzuführende Begriffe aus fremden Quellen werden durch „“ dargestellt.
- Eigene Begriffe werden bei erster Verwendung in *kursiv* dargestellt.
- typisierte Relationen werden mit _ Platzhaltern notiert.
- „Hintergrund“-Boxen: Diese erläutern den Ursprung einer Idee oder liefern eine Referenz mit einer kurzen Erklärung.
- Grafiken basieren auf UML, werden jedoch um eigene Elemente (z.B. Pfeile) erweitert.
- Erste Verwendung von Symbolen (wie Pfeilen) wird beschrieben.

2 Stand der Wissenschaft und Technik - mit Begriffsklärung

Interestingly, according to modern astronomers, space is finite. This is a very comforting thought – particularly for people who can never remember where they have left things.

Woody Allen

Die in dieser Arbeit vorgestellte Lösung beschreibt eine gemeinsame Datenbasis, die als strukturierte, verteilte Festplatte verstanden werden kann. Hierfür werden Modelle basierend auf Objekten und Klassen verwendet. Um einen Austausch sicherzustellen, ist als Grundlage wichtig, die unterschiedlichen Kommunikationssysteme, wie sie heute in der Automatisierungstechnik anzutreffen sind, darzustellen und auf ihre wesentlichen Unterschiede hin zu untersuchen. Weiterhin existieren bereits Systeme, um Modelle zur Laufzeit zu verwalten und Änderungen vorzunehmen. Sie stellen die technologische Basis der hier beschriebenen Lösung dar. Mit diesen existierenden Konzepten beschäftigen sich die folgenden Abschnitte.

Hintergrund:
Die hier adressierten Informationen sollen nicht nur zum Engineering genutzt werden. Während es bei dem Automation Service Bus [BMM12] oder auch dem Siemens TIAC-Portal hauptsächlich um die Integration bzw. Synchronität von Informationen während des Engineering-Prozesses geht, adressieren die hier beschriebenen Konzepte Informationen auch zu anderen Phasen, wie Inbetriebnahme / Produktion / Wartung / . . . .

2.1 Vom Wissen zu Maschinen-verarbeitbaren Modellen in der AT

Die VDI-Richtlinie 5610 [10] beschreibt den Zusammenhang der Begriffe „Wissen“, „Information“ und „Daten“. Hier wird formuliert, dass Daten „objektive Fakten“ darstellen, die jedoch ohne weiteres Wissen nicht deutbar sind. Sie sind als „Rohmaterial“ zu verstehen. Informationen werden verstanden als „strukturierte Daten“, die in einen Kontext gebracht werden können. Wissen schlussendlich ist „vernetzte Information“, sodass Vergleiche angestellt werden können und Entscheidungen getroffen werden. Abbildung 2.1 verdeutlicht den Zusammenhang.

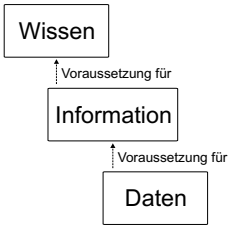


Abbildung 2.1: Verständnis der Begriffe Daten, Information und Wissen nach VDI-Richtlinie 5610

In Sinne dieser Richtlinie beschreibt ein Modell Wissen über die Realität bzw. einen Ausschnitt der Realität. Damit bildet es Informationen ab, die von Anwendungen im Sinne der VDI-Richtlinie 5610 genutzt werden können, um Entscheidungen zu treffen.

Definition: *In dieser Arbeit wird unter **Modell** eine Objekt-orientierte Beschreibung eines Ausschnitts der Realität verstanden.*
*Eine solche **Modell-Beschreibung** besteht dabei aus ihren Teilen wie auch den Beziehungen unter ihnen. Zum anderen wird unter einem Modell aber auch die Anwendung des Modelles auf einen konkreten Sachverhalt verstanden. Diese **Modell-Instanz** hat einen Bezug zu genau einem Sachverhalt der Realität.*

Im Objekt-orientierten Sinne handelt es sich bei der Modell-Beschreibung um Klassen und Relationen zwischen den Klassen. Die Instanzen dieser Klassen sind die Objekte und beziehen sich auf den konkreten Sachverhalt. In Abbildung 2.2 werden die geläufigsten Begriffe aus der Objektorientierung knapp dargestellt, werden jedoch vom Verständis her vorausgesetzt. Eine

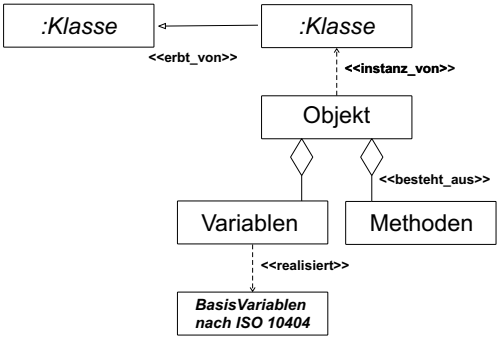


Abbildung 2.2: Aufbau von Objekten, sowie ihre Instanziierung

ausführlichere Darstellung ist vielfach in der Literatur beschrieben - beispielsweise in [Mey03].

Ein wichtiger Punkt ist, dass zeitliche Abläufe (wie ein Produktionsablauf) selber einen Ausschnitt der Realität darstellen und in Modellen abgebildet werden können.

Damit die unterschiedlichen Ausschnitte der Realität in Modellen abgebildet werden können, macht es Sinn, ein gemeinsames Meta-Modell zu nutzen.

Definition: *Ein **Meta-Modell** beschreibt, wie Modelle aufgebaut und strukturiert sind. Dazu zählen auch Sprachkonstrukte wie Klassen und Relationen.*

Eine genauere Beschreibung einer Architektur der Meta-Modell-Hierarchie mit unterschiedlichen Ebenen hat die „Object Management Group“ (OMG) in ihrer „Meta Object Facitlity“ (MOF) Spezifikation [24] beschrieben. In großen Teilen ist sie hier anwendbar.

Die genutzten Konzepte sind übliche Konzepte insbesondere der Objekt-orientierten Programmiersprachen. Sie werden hier als Basis verwendet und deswegen in aller Kürze beschrieben.

2.1.1 (Modell-)Relationen

Um Abhängigkeiten zwischen Klassen und damit auch ihre Instanzen darstellen zu können, existieren Relationen.

Definition: *Eine **Relation** ist eine gerichtete Verbindung zwischen mindestens zwei Objekten. Sie ist selber ein Objekt, wird instanziiert. Die Klasse der Relation stellt deren Typ dar - es handelt sich also um typisierte Relationen. Sie wird im Modell beschrieben.*

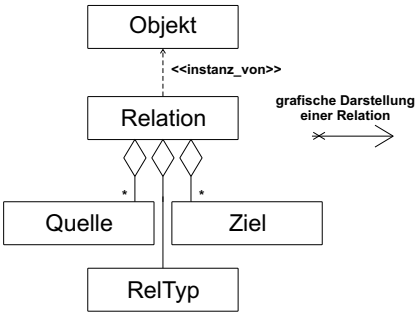


Abbildung 2.3: Aufbau von Relationen zwischen Objekten

Relationen werden typischerweise lediglich begriffen als „Nutzung“, d.h. wenn ein Objekt eine Funktion eines anderen Objektes aufruft, besteht zwischen ihnen eine Relation. Modelle bringen ihre eigenen Relationen mit. Daneben werden aber auch im Meta-Modell Relationen beschrieben: Weiter oben ist bereits die instanz_von Relation der Objekt-Orientierung erwähnt worden.

Allgemein in der Objekt-Orientierung bekannte Relationen sind beispielsweise die Aggregation oder Komposition. In dieser Arbeit werden die Relationen jedoch aktiv im Modell zur Beschreibung eingesetzt. D.h. die vom Meta-Modell bereitgestellte Klasse der Assoziationen „Relation“ hat immer Quellen und Ziele, gegebenenfalls jeweils mehrere davon (vgl. Abbildung 2.3). Von „Relation“ werden in den Modellen entsprechende Relationsklassen abgeleitet. Diese beschreiben jeweils

- Typen
- Quelle (und Multiplizität; in der Regel 1)
- Ziel (und Multiplizität)
- Eigenschaften.

der Relation. Die Eigenschaften können Aussagen über die Relation treffen. Beispielsweise könnten Relationen einen Zustand besitzen oder auch Qualitäten beschreiben.

Relationen sind **nicht** Bestandteil des Objektes, d.h. ein Objekt kann von einer Relation wissen, muss es jedoch nicht. Sollte eine Relation jedoch Bestandteil des eigenen Modells sein, so kann das Objekt die Relation nutzen.
Hierdurch wird das später vorgestellte Konzept der Integration von Modellen zur Laufzeit (vgl. Kapitel 4.5) möglich. Bereits bekannt ist ein ähnliches Verständnis aus dem Objektverwaltungssystem der ACPLT-Technologien ACPLT/OV [Mey03].

Modelle besitzen zwei Formen der Relationen. Für die beschriebene Aufgabenstellung ist es inhärent wichtig, beide in einer Form abzubilden, damit diese zur Laufzeit des Gesamtsystems erkundbar und somit nutzbar sind.

Die eine Form der Relation ist *strukturell* bedingt. Ein Beispiel für eine solche Abhängigkeit ist die Vererbung. D.h. eine Klasse eines Modells erbt Eigenschaften aus einem Obermodell, welches ggf. auch in anderen Modellen genutzt werden kann. Oder ein Modell definiert eine Relation zu einer Klasse eines anderen Modells. Solche Relationen sind üblich und auch vielfach verwendet.

Zum anderen gibt es aber auch Relationen, die bei der Modellierung (im Sinne von Entwicklung der Modelle), aber auch auf der Instanz-Ebene weitestgehend unberücksichtigt bleiben. Sie können als *anwendungsspezifische* Abhängigkeiten bezeichnet werden: Modelle sind technisch und semantisch unabhängig und werden durch unterschiedliche Gremien entwickelt, sowie auf Instanz-Ebene auch von unterschiedlichen Ingenieuren für konkrete Anlagen umgesetzt. Beispielsweise gibt es sehr wohl eine Verbindung zwischen den zu erfüllenden Prozessschritten eines Rezeptes zu den Anlagenteilen, die diese Prozessschritte ausführen können. Während diese Verknüpfung durch die IEC-61512 [3] Rezepte abgedeckt wird, wird eine Zuordnung von Elektroverkabelung zur Produktionszeit meist nicht vorgehalten. Die Verknüpfung der Engineering Daten der Elektroverkabelung mit Automatisierungsgeräten fehlt somit zur Laufzeit und wesentliche Aufgaben z.B. zur Diagnose bei Kommunikationsproblemen können nicht au-

tomatisiert erfolgen, obwohl die Daten in Modellform vorliegen. Somit sind die Modelle für sich betrachtet unabhängig, in der Anwendung sollten sie jedoch untereinander in Relation gebracht werden.

Eine wesentliche Wertschöpfung für eine schnellere Inbetriebnahme liegt in diesem Bereich: Der Integration von bisher unabhängigen Daten, sodass Änderungen in einem Datensatz zumindest semi-automatisch Änderungen in den anderen Datensätzen nach sich ziehen.

Gleichzeitig müssen Änderungen an einzelnen Elementen oder Relationen konsistent gehalten werden, d.h. es müssen Technologien bereitgestellt werden, um Änderungen zu erkennen; im Idealfall auch um hierauf (teil-)automatisiert reagieren zu können.

Während die strukturelle Abhängigkeit explizit im Modell hinterlegt ist und schon bei dem Entwurf von Modellen berücksichtigt wird, ist die implizite oder anwendungsspezifische Abhängigkeit der Modelle ein bisher weitgehend ungelöstes Problem.

Einige Arbeiten zielen in Richtung der Modell-Integration: So werden unterschiedliche, existierende Modelle wie die IEC 61850 [LMRU11], ISA-95 [BHMO13] und AutomationML [Sch13] in OPC-UA abgebildet, um eine höhere Integration der Daten zu erreichen und diese gleichzeitig in dem verteilten System der Anlagen zugreifbar zu machen. In Zusammenhang mit Konzepten wie „Plug&Play“ (Inbetriebnahme von automatisierungstechnischen Geräten ohne manuelle Konfiguration; vgl. beispielsweise [Hod13]) kann eine wesentliche Beschleunigung des Engineerings sowie der Instandhaltung erreicht werden.

2.1.2 Instanz-Struktur: Komponenten als Gruppierung

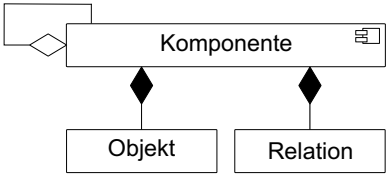


Abbildung 2.4: Komponenten fassen Objekte und Relationen zusammen

Ein wesentlicher Hintergrund der Objekt-Orientierung ist die Kapselung. Dabei werden mehrere Objekte (und Relationen) nur durch eine Schnittstelle angesprochen, sodass die interne Struktur verborgen wird. Ihre Architektur ist in Abbildung 2.4 dargestellt.

> **Definition:** *Eine „Komponente“ ist eine vorgefertigte, in sich strukturierte und unabhängig hantierbare Einheit, die zur Realisierung einer konkreten Rolle in einem System vorgesehen ist (nach [12]).*

Somit kann es sich z.B. um eine nicht näher festgelegte Menge von Objekten und Relationen handeln, die über eine Aufgabe und eine Schnittstelle verfügen. Ebenso kann es sich um

ein Klassen-Modell handeln, welches im Gesamtsystem zur Beschreibung eines bestimmten Sachverhaltes eingesetzt wird.

Zum einen werden hierdurch Komponenten austauschbar, wenn sie die gleiche Schnittstelle anbieten. Zum anderen muss ein Nutzer nur die Schnittstelle kennen und das Verständnis haben, welchen Zweck eine Komponente verfolgt.

2.1.3 Bestandteile einer Modell-Beschreibung

In Abbildung 2.5 wird dargestellt, welche Bestandteile eine Modell-Beschreibung im Allgemeinen ausmachen. Hierbei handelt es sich um „typische“ Bestandteile, wie sie vielfach in der Literatur vorkommen. Ein Modell beschreibt eine Menge von Klassen sowie die zugehörigen

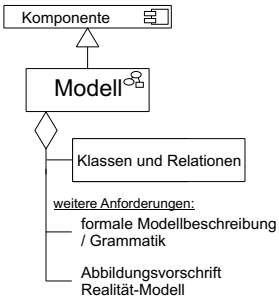


Abbildung 2.5: Ein Modell als strukturierende Komponente

Relationen. Diese werden instanziiert, um eine konkrete Ausprägung der Realität abzubilden. Weiterhin ist es nötig, dass eine formale Modell-Beschreibung vorliegt, die mögliche Strukturen der Klassen und Relationen beschreibt (Syntax). Ebenso muss die Abbildungsvorschrift von der Realität zum Modell enthalten sein (Semantik).

Damit die Existenz eines Modells in der Instanzumgebung erkundet werden kann, kann eine Modell-Beschreibung als Objekt im Modell-Raum repräsentiert werden, die selber instanziiert wird und gewissen Eigenschaften genügt (vgl. Abbildung 2.5).
Existierende Systeme repräsentieren das Modell selber nicht unbedingt als Instanz - jedoch ist eine meist äquivalente Struktur möglich[1].

2.2 Existierende Modelle der Automatisierungstechnik

Die im Folgenden kurz dargestellten Modelle haben eine weitreichende Sichtbarkeit in der Automatisierung, sind aber lediglich eine Auswahl. Sie werden nur äußert knapp beschrieben,

[1] OPC-UA: „Namespace“ und ACPLT-OV: „Library“

da ihre konkrete Verwendung für das weitere Verständnis der beschriebenen Konzepte nicht relevant ist. Die Liste soll als Anregung für die Verwendung des beschriebenen Systems verstanden werden.

Viele weitere Modelle sind in [SMS11] beschrieben. Auch Blatt 1 der VDI Richtlinie 3690 [1] beinhaltet hier eine sinnvolle Übersicht existierender Modelle. Ein Vergleich gerade in Bezug auf Unterschiede zwischen Fertigungstechnik und Prozesstechnik wird in [MBS+11] beschrieben.

AutomationML [13] Die IEC 62714 beschreibt AutomationML: Topologie, Geometrie, Kinematik und Logik werden in einem gemeinsamen XML Dokument repräsentiert. Hierfür werden andere Subformate genutzt, die durch andere Gruppen spezifiziert werden. AutomationML setzt diese dann in Relation. Hintergrund ist der Austausch dieser Informationen zwischen Engieneeringsystemen für eine Steigerung der Effizienz in der Planungs-Phase.

IEC 62424: CAEX [7] & PandIX [ERD11] Während CAEX eine allgemeine XML-Darstellung für hierarchische Systemstrukturen darstellt, bildet PandIX einen Vorschlag für die funktionale Strukturbeschreibung einer verfahrenstechnischen Anlage.

SysML [22] Basierend auf UML bietet SysML eine Beschreibungssprache für komplexe Systeme aller Art. Neben der System-Struktur selber sind auch Kriterien wie das Verhalten des Systems repräsentierbar.

ISA-95 [11] Die ISA-95 basiert auf der ISA-88. Die ISA-95 beschreibt eine Schnittstelle des MES Systems zum ERP-System. Hierdurch werden u.a. auch Ressourcen, wie Menschen [BHMO13], erfasst und abgebildet.

Geräte-Konfigurationen Beispielsweise bieten FDI oder FDTv2 Beschreibungen für Geräte-Konfigurationen an. Insgesamt gibt es eine Vielzahl, die aktuell unterstützt werden müssen. So spricht [Gös14] von dreizehn von den Herstellern zu unterstützenden Formaten. [Gös14] beschreibt dazu auch einen Ontologie-basierten Ansatz um dieses Problem zu adressieren.

XML Formats for IEC 61131-3 [28] Die IEC 61131 Teil 3 [4] beschreibt Sprachen für die Programmierung einer SPS. Durch diese Sprachen werden Konstrukte wie konkrete Funktionsbausteine definiert. Von einem 61131-3-kompatiblen Automatisierungssystem wird folglich verlangt, dass es eine Programmierschnittstelle für die definierten Sprachen bereitstellt. Basierend auf diesen Sprachen definiert die PLCopen eine formale Abbildung einer Steuerungsprogrammierung in XML. Diese kann (im Idealfall) als Austauschformat von Steuerungsprogrammen zwischen unterschiedlichen Automatisierungssystemen verwendet werden.

VDI Richtlinie 3682: Formalisierte Prozessbeschreibung [5] Die VDI Richtlinie 3682 stellt eine formale Abbildung zur Prozessbeschreibung bereit, die auf UML-Darstellungen basiert. Es werden sowohl kontinuierliche- wie auch chargen-orientierte Prozesse abbildbar.

Die Richtlinie adressiert gleichbleibende Prozesse und kann dadurch wesentlich kompakter gehalten werden, als beispielsweise WS-BPEL.

WS-BPEL [21] & andere WS-* Beschreibungen WebService-Beschreibungen (als WS* bezeichnet) beschreiben Schnittstellen, die zwischen Kommunikationspartnern genutzt werden um unterschiedliche Sachverhalte zu kommunizieren. Hierzu stehen eine Vielzahl von separaten, kombinierbaren Beschreibungen bereit.
Als Beispiel für solche Sachverhalte kann „WS-Business Process Execution Language“ (kurz WS-BPEL) angesehen werden. Diese stellt eine Beschreibungssprache für Geschäftsprozesse dar. Sie kann für die Repräsentation von Abläufen auf ERP Ebene verwendet werden. Eine Adaption auf Rezepte oder sogar „Sequential Function Charts“ – also Abläufe auf Steuerungsebene – ist vorstellbar.

Vielen Normen liegt ein Objekt-orientiertes Modell zugrunde, welches sie selber beschreiben. Diese Tatsache bringt zwei wesentliche Nachteile mit sich:

Umfang wächst Es führt zu Akzeptanz-Problemen, da die Normen nicht mehr schnell erfasst und überblickt werden können.

Werke in Details inkompatibel zueinander Dieses erschwert die Integration der beschrieben Modelle sowie die Interaktion der Modelle untereinander.

Durch gemeinsame Basis-Modelle werden sich diese Probleme für alle Seiten vereinfachen lassen.

Bestrebungen in diese Richtung auf formaler Seite gibt es durch den DKE-Arbeitskreis 931.0.4 in dem sogenannte „Kernmodelle“ [KE12] beschrieben werden, die genau diese Aufgaben übernehmen. Sie sollen dabei kompakt auf wenigen Seiten beschrieben werden, sodass sie fachlich fundiert und präzise sind, jedoch auch schnell überblickt werden können.
Zusätzlich gibt es Bestrebungen Modelle in OPC-UA (vgl. folgender Abschnitt) zu repräsentieren, welches die Probleme ebenso begegnet. Wenn viele Modelle eine Abbildung in OPC-UA besitzen, haben diese auf Meta-Modell-Ebene einen gemeinsamen Nenner und eine kompatible, technische Repräsentation.

Als Forschungsprojekt strebt beispielsweise „Secure plug and work“ [Sau13] eine Integration von AutomationML in OPC-UA an. [BHMO13] beschreibt das Konzept zu Integration von ISA-95 in OPC-UA und sieht Anwendungsgebiete im Bereich der MES; insbesondere wird dabei auf Qualitätsmaßnahmen und Wartungsarbeiten sowie Verwaltungs-Anwendungen verwiesen. Die XML-Darstellung der 61131-Funktionsbausteine durch die PLCopen wurde auch für OPC-UA entworfen [27]; zusätzlich definiert die PLCopen auch Kommunikationsbausteine, mit denen OPC-UA Kommunikation direkt aus der SPS ausgeführt werden kann. Hierdurch wird die immer stärker werdende Verzahnung von Steuerungen mit Informationen, die als Modell vorliegen, deutlich.

2.3 Kommunikation in der Automatisierungstechnik

Die derzeitigen Kommunikations-Technologien in automatisierungstechnischen Anlagen sind äußerst heterogen. Eine einheitliche, transparente Kommunikation wird zwar immer wieder adressiert, ist aber heute in der realen Anlage nicht existent.

In den letzten Jahren kommen immer mehr Technologien auf Ethernet-Basis auf den Markt, die zumindest auf der physikalischen Ebene kompatibel sind. Aufgrund der Natur der Märkte - Verkauf wird über Alleinstellungsmerkmale gewonnen - divergieren diese Produkte aber sowohl auf Hardware-Ebene (unterschiedliche Stecker) wie auch auf informationstechnischer Ebene, sodass sie im Endeffekt zwar gleiche Wurzeln haben, jedoch nicht kompatibel sind. Das Buch *Datenkommunikation in der Prozessindustrie* von Enste und Müller [EM07] bietet hier einen guten Überblick, der auf die in der Prozessindustrie üblichen Netze fokussiert.

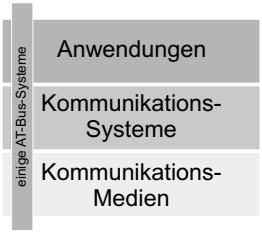


Abbildung 2.6: Grundsätzliche Teilung der Kommunikation

Abbildung 2.6 teilt diese Problematik in drei[2] Bereiche auf, die jeweils unterschiedliche Aufgaben haben. Auf der *Anwendungsebene* sollten die Abläufe und Daten im Vordergrund stehen. Die Forderung nach einer *transparenten Kommunikation* spielt hier ihre Vorzüge aus. So müssen Anwendungen sich nicht um die Zustellung der einzelnen Nachrichten kümmern.
In der Automatisierungstechnik werden immer unterschiedliche Bus-Systeme und Netzwerke für die unterschiedlichen Anforderungen wie Echtzeit-Verhalten oder Explosionsschutz existieren, weswegen es sinnvoll erscheint, aufbauend ein (möglichst nur ein einziges) Kommunikations-System zu haben, welches unabhängig von den unterliegenden *Kommunikations-Medien* eine einheitliche Vorstellung der Kommunikation abbildet.

[2] Abstraktion über das ISO/OSI Schichtenmodell [2], welches an dieser Stelle in dem Detailgrad nicht benötigt wird.

Hintergrund:
Aktuelle Kommunikations-Technologien versuchen in Richtung Anwendung zu wachsen, also Dienste für konkrete Anwendungsfälle bereitzustellen. Beispiel: Geräteidentifikation und Ausfallerkennung.
Dieses hat jedoch zwei entscheidende Nachteile: Zum einen schränkt es die Nutzbarkeit der Kommunikations-Technologie ein. Zum anderen sind die Anwendungen, die die Dienst-Schnittstellen nutzen, an diese Kommunikations-Technologie gebunden. Sie müssen zur Unterstützung von einer anderen Kommunikations-Technologie gleich eine neue, zusätzliche Anbindung bekommen.
Es erscheint also sinnvoll, unterschiedliche Ebenen zu etablieren und Technologien auf jeweils eine Ebene zu beschränken.

Im Folgenden werden generelle, existierenden Ansätze und Unterschiede der Markt-üblichen Lösungen in der industriellen Automation aufgezeigt und verglichen. Dabei wird verdeutlicht, dass existierende Konzepte bereits weitgehend vorhanden sind und genutzt werden können, um diese Ziele zu erreichen.

2.3.1 Kommunikations-Medien: Bussysteme und Alternativen

In der Automatisierung gibt es eine Vielzahl von Kommunikations-Technologien. Aus unterschiedlichen Gründen kommen die Möglichkeiten zumindest in verfahrenstechnischen Anlagen meistens gemischt vor. Zum einen sind meistens Umbauten und dadurch unterschiedliche Generationen in den Anlagen zu finden. Zum anderen haben die unterschiedlichen Kommunikations-Technologien auch unterschiedliche Anwendungsfälle, die sie abdecken. Einige sind z.B. für den Ex-Bereich geeignet und bringen eine eigene Energieversorgung mit oder zeichnen sich durch Kosten der Installation aus.

Diese unterschiedlichen Technologien werden im Folgenden als *Kommunikations-Me-dien* bezeichnet.

Definition: *Als Kommunikations-Medium werden sowohl die physischen Komponenten (Kabel, Ausrüstung in Geräten) wie auch die Software und Protokolle verstanden, welche die physischen Komponenten verwenden. Insgesamt bietet ein Kommunikations-Medium so die Möglichkeit, Daten von einem Gerät zum Anderen zu transportieren.*

Konventionelle (4..20mA) Verkabelung Traditionell ist die konventionelle 2-Draht-Verdrahtung zu sehen. Hier werden Signale direkt auf einem Kabel übertragen. Für digitale Signale gibt es sowohl eine Realisierung auf Basis der elektrischen Spannung wie auch auf Basis des Stroms. Im Gegensatz dazu hat sich für analoge Werte ein Strom zwischen 4..20mA durchgesetzt. Hier existiert aufbauend das HART Protokoll, welches eine zusätzliche, bidirektionale Kommunikation ermöglicht.

Auf elektrotechnischer Ebene handelt es sich bei der konventionellen Verdrahtung immer um eine 1:1 Verbindung von der I/O Karte zu einem Sensor/Aktor. Eine Kommunikation unter mehreren Geräten ist nicht möglich. Es gibt auch keine Möglichkeit eine Kommunikation zu anderen Teilnehmern als den I/O Karten aufzubauen. Damit handelt es sich bei der konventionellen Verdrahtung nicht um eine vollwertige Kommunikationsmöglichkeit, wie sie für die Instanzumgebung benötigt wird. Sie wird also für das beschriebene Gesamtsystem insofern ausgeschlossen, dass die hier betrachteten Geräte nicht per konventioneller Verdrahtung angeschlossen sein können. Es können aber übergeordnete Geräte eine Schnittstelle zu konventionell angeschlossenen Geräten darstellen (typischerweise als *Remote I/O* bezeichnet).

Bussysteme Weit verbreitet in heutigen Anlagen sind Bussysteme. Dabei handelt es sich um eine vollwertige, bidirektionale Kommunikation zwischen den angeschlossenen Geräten, wobei die Teilnehmer (nach IEV 351-20-10) nicht an der Weiterleitung der Daten mitwirken. Auf einem Bus kann somit zu einem Zeitpunkt elektrisch nur ein Gerät senden. Dieses wird meist von einem *Master* koordiniert. Bei einigen Systemen wird durch das Protokoll die technisch mögliche Kommunikation zwischen allen Teilnehmern auf die Kommunikation von Geräten zu dem Master begrenzt. In einem solchen System kann folglich auch nur der Master - ähnlich der oben beschriebenen I/O Karte - als Gerät im Sinne der modellgetriebenen Instanzumgebung gesehen werden.

Netzwerke Viele neuere Entwicklungen in der Automatisierung basieren auf dem Gedanken von (IT-)Netzwerken. Abgesehen von unterschiedlichen Verkabelungsformen existiert bei einem Netzwerk im Gegensatz zu einem Bussystem eine direkte Kommunikationsmöglichkeit von zwei (oder mehr) angeschlossenen Geräten, die diskrete Nachrichten austauschen. Dabei sind auch aktive Komponenten zur Weiterleitung von Daten zulässig (Gateways, Router, Switche)[3].
Im Gegensatz zu einem Bussystem kann ein Netzwerk zu einem Zeitpunkt somit meistens auch von unterschiedlichen Geräten gleichzeitig genutzt werden, sodass keine exklusive Nutzung des gesamten Netzwerks zu einem Zeitpunkt existiert.

Gemein ist den letzten zwei Technologien von Kommunikations-Medien, dass sie (auch wenn sie für den Echtzeit-Betrieb vorgesehen sind) immer eine Nicht-Echtzeit-Kommu-nikations-Möglichkeit mitbringen. Diese wird durch eine asynchrone Kommunikation erreicht, sodass der Sender sich nicht auf eine Antwort in einer definierten Zeitspanne verlassen kann: Diese Kommunikation wird zwischen der Echtzeit-Kommunikation übermittelt, wenn Ressourcen frei sind. Diese Nicht-Echtzeit-Kommunikation kann für die Instanzumgebung genutzt werden.

Gemeinsam haben Bus- und Netzwerksysteme auch, dass sie die verschiedenen Teilnehmer mit Adressen organisieren.

[3] Nach IEV 131-11-06 wird ein Netzwerk allgemein als „Menge von miteinander verbundenen Netzwerkelementen“ verstanden

Definition: *Eine **Adresse** ist eine Möglichkeit Teilnehmer innerhalb eines Gruppe zu identifizieren sowie zu adressieren. Eine Adresse ist also zum einen eindeutig einem Teilnehmer zuzuordnen, beschreibt aber auch, wie dieser auf Basis eines Kommunikationssystems zu erreichen ist.*

Die Kommunikation selber findet dann durch Nachrichten statt, die an diese Adressen gesendet werden.

Definition: *Eine **Nachricht** ist ein Datensatz, der von einem Teilnehmer an einen oder mehrere andere Teilnehmer gesendet wird. Eine Nachricht enthält zum einen Transport-Informationen (z.B. Ziel, Zeitpunkt, Sender) und zum anderen die zu transportierenden Nutzdaten.*

Kontinuierliche vs. diskrete Kommunikation Ein konzeptueller Unterschied zwischen der 4..20mA Verkabelung und den Bussystemen/Netzwerken besteht in der Kommunikation selber.

Bei einer direkten Verkabelung liegt mit einer analogen Änderung des Stroms (4..20mA) permanent zu jedem Zeitpunkt am Ziel ein Wert an. Dieser kann als „kontinuierliche“ oder „Signal-orientierte“ Kommunikation bezeichnet werden. Ein solches Verhalten muss bei Bussystemen und Netzwerken „aufwändig“ nachgebildet werden, da diese in Nachrichten kommunizieren. Es kann also als „diskrete“ Kommunikation bezeichnet werden. Hierfür werden zyklisch die entsprechenden Werte übertragen. Über den benötigten Zyklus muss sich dabei zur Zeit der Anlagenplanung und des Engineerings verständigt werden. Dieses ist also ein Nachteil für die modernen Kommunikationssysteme.

2.3.2 Formen der Kommunikation

Wenn ein Medium eine Kommunikation zwischen den angeschlossenen Teilnehmern zulässt, gibt es unterschiedliche Formen der Kommunikation.

Definition: *Eine **Kommunikations-Form** gibt an, welche Kardinalität zwischen Sender und Ziel besteht. Prinzipiell sind Unicast (1:1), Multicast (1:N) und Broadcast (1:*) üblich.*

Die unterschiedlichen Formen werden typischerweise wie folgt charakterisiert.

Unicast - 1:1 Unter Unicasting wird eine gerichtete Kommunikation zwischen genau zwei Kommunikationspartnern verstanden. Meistens erfolgt die Kommunikation dabei durch eine Verbindung, die erst aufgebaut wird und dann für mehrere Vorgänge verwendet werden kann bevor sie wieder abgebaut wird. Die Verbindung übernimmt dabei Aufgaben der Transaktionssicherung, d.h. das Erkennen und erneute Senden verloren-gegangener Nachrichten.

Davon abweichend ist beispielsweise eine UDP Kommunikation, die eine Unicasting-Kommunikation bereitstellt, ohne eine Verbindungs-orientierte Absicherung der Kommunikation, sodass ein Paketverlust nicht erkannt wird. Aufbauend gibt es „reliable UDP“ um dieses Manko zu beheben, sodass hier ohne eine Verbindung aufzubauen eine Transaktions-gesicherte Kommunikation zu Stande kommt.

Multicast - 1:N Beim Multicasting wird eine fest definierte Gruppe von Empfängern mit einer Nachricht versorgt. Das Prinzip existiert in der Realität selten, ideale Anwendungsfälle für TCP/IP-basiertes Multicasting sind Anwendungen wie beim Internetradio. Leider werden die nötigen Routing-Standards, die die Pakete zu den Empfängern bringen, nicht allgemein bereitgestellt.

Broadcast - 1:* Beim Broadcasting wird, wie der Name schon sagt, von einem Sender mehrere Empfänger angesprochen, die dem Sender nicht bekannt sein müssen. Broadcast Nachrichten bringen immer das Problem mit, dass sie eingedämmt werden müssen, damit sie nicht zu viel Kommunikationslast erzeugen. Gleichzeitig bringen Sie aber den Vorteil der Kontaktaufnahme ohne Vorprojektieren: Auf eine Broadcast-Nachricht können sich Teilnehmer melden und somit ist eine initiale Kommunikation möglich.
Bekanntestes Beispiel ist die Konfiguration per DHCP [16], wie sie im TCP/IP Netzwerk für die Vergabe der Kommunikations-Adressen eingesetzt wird.

Für die hier vorgestellten Konzepte ist die *Unicast Kommunikation* vorgesehen.

2.3.3 Kommunikations-Systeme für den Zugriff auf Modelle

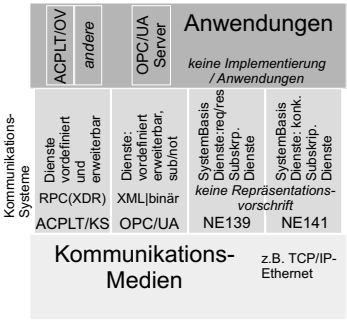


Abbildung 2.7: Zusammenfassung einiger Kommunikations-Realisierungen

In Abbildung 2.7 ist die vorherige Abbildung 2.6 für einige aktuelle Kommunikations-Systeme detaillierter dargestellt. Dabei wurde eine Auswahl getroffen, die sich auf Kommunikation für die Manipulation von ganz allgemeinen Objekt-Strukturen beschränkt. Die beiden konkreten

Systeme ACPLT und OPC-UA sind dabei gleichzeitig auch die Instanzumgebungen für die Modelle, wie sie später vorgestellt werden.

ACPLT/KS ACPLT/KS ist das Kommunikationssystem des ACPLT-Technologiepaketes. Die ursprüngliche Implementierung basiert auf Remote-Procedure-Calls (kurz: RPC [15]), um die eigentlichen Befehle auf TCP/IP abzubilden. Die Remote-Procedure-Calls sind dabei in ihrer ursprünglichen Form nur auf TCP/IP spezifiziert, jedoch existiert mittlerweile von ACPLT/KS eine 2. Version, die auch die Unterstützung für unterschiedliche Kommunikations-Medien mitbringt, dabei aber weiterhin auf die RPC-Repräsentation setzt.

OPC-UA OPC Unified-Architecture bietet durch entsprechende Dienste Zugriff auf Objektstrukturen. Dabei ist hervorzuheben, dass die Dienste selber durch einem „Kanal“ gesendet werden, der sowohl die Verschlüsselung / Signierung wie auch das Erkennen eines Verbindungsabbruches (und Wiederaufnahme der Kommunikation) übernimmt.

NE 139 und NE141 Die Namur hat sich in diesen beiden Empfehlungen mit dem Kommunikations-Bedarf von Systemen in der Prozesstechnik beschäftigt. Die NE141 spezifiziert dabei Details der NE 139 aus. Sie stellen bewusst nur Konzepte für Kommu-nikations-Systeme auf und abstrahieren dabei von den unterliegenden Kommunikations-Medien. Statt konkreten Repräsentationen werden Anforderungen an ein unterliegendes Mediums beschrieben.
Die NE141 beschreibt ganz konkrete „SystemBasisDienste“, mit der ein System erkundet (Modell-Erkundungsfunktionen) und verändert werden kann (Modell-Änderungs-Funktionen), gerade ohne sich dabei auf ein Medium zu beschränken. Die NE141 beinhaltet selber auch einen Vergleich mit der Kommunikation aus der ISA-S95 [11], sodass diese Norm hier nicht weiter betrachtet werden muss.

Alle Kommunikations-Systeme definieren eine Schnittstelle um einen einheitlichen Zugriff auf die interne (Objekt-)Struktur zu ermöglichen. Diese Struktur kann durch die Dienste der Schnittstelle bei allen Systemen sowohl erkundet, wie auch abgefragt und manipuliert werden. Wie schon in Abbildung 2.7 zu sehen ist, unterscheiden sich alle vier Systeme in einigen Details. Diese sind in der folgenden Tabelle zusammenfassend dargestellt.

	ACPLT/KS	OPC-UA	NE 139	NE 141
Beispiel für SystemBasisDienste:				
Hole Variablenwert	GetVar	Read		Lese Var.
Erzeuge Objekt	CreateObj	AddNodes		Erzeuge Obj.
Struktur-Erkundungs-Dienst	GetEP	Browse		Browse Attrib.
Eigenschaften:				
Erweiterbar (neu Schnittstellen)	✓	✓	✓	✓
Subskription/Benachrichtigung	×	✓	✓	✓

Alle vier Systeme beschreiben Nachrichten und deren Interaktion. Die NE 139 beschreibt dabei als elementare Form die einfachen Nachrichten, bei denen keinerlei Rückmeldung zu erwarten

sind. Eine solche Kommunikationsform kennen die anderen drei Systeme nicht. Hier hat jeder Aufruf einen Rückgabewert.

Alle vier Systeme definieren einen Satz von Schnittstellen-Diensten, der ausreichend ist um den Objektraum zu *erkunden, abzufragen und zu verändern*. Auch wenn die konkrete Benennung der Dienste unterschiedlich ist, gibt es hier einen weitreichenden Konsens über die Basis-Dienste, welche von den beiden Namur-Empfehlungen deswegen auch *SystemBasisDienste* genannt werden.

Darüber hinaus bieten alle Dienste bis auf ACPLT/KS ein Subskription/Benachrichtigungs-Mechanismus an. Dieser ist unterschiedlich stark ausgeprägt. Während die NE 139 den Mechanismus selber beschreibt, jedoch keine konkreten Befehle hierauf, beschreibt die NE141 und OPC-UA ganz konkrete Dienst-Schnittstellen z.B. um Benachrichtigungen über Abweichungen von Variablen zu erhalten.

Alle vier Systeme bieten die Möglichkeit erweiterte Dienste zu spezifizieren. In OPC-UA wird dieses beispielsweise für *Methoden* und *Programme* genutzt. In ACPLT/KS wird es derzeit jedoch nicht verwandt. Stattdessen werden allgemeine Nachrichten in einem atomaren Datentyp abgebildet und als serialisierte Variante übertragen[4]. So gibt es unterschiedliche Wege, um komplexe Daten-Typen als Aufrufstruktur zu übermitteln.

Die Anwendungen, die diese Kommunikation ausnutzen, existieren für ACPLT/KS und OPC-UA. Die Möglichkeiten der Objekt-Verwaltung als Instanzumgebung werden dabei im folgenden Kapitel betrachtet.

Insgesamt gesehen sind diese vier Systeme ähnlich, größtenteils sogar äquivalent. Es existiert also auf Kommunikations-System-Ebene ein allgemein gebräuchliches Verständnis von den Dienst-Aufrufen für die Abfrage, Erkundung und Manipulation einer Objekt-Struktur.

2.4 Instanzumgebung der Modelle

Modelle sind in erster Linie Beschreibungen von Daten. Um diese aktiv nutzen zu können, werden gerade in einem dynamischen Umfeld „Instanzen“ der Modelle gebraucht.

Definition: *Der **Modell-Raum** ist ein Speicherplatz, an dem Modelle und Klassen/Relationen der Modelle instanziiert werden. Es werden durch die vorliegenden Modell-Beschreibungen also Modell-Instanzen erzeugt. Die Klassen und Relationen werden dabei selber im Modell-Raum repräsentiert, sodass eine Erkundung dieser möglich ist.*
Die Objekte einer Instanz haben innerhalb des Modell-Raums eine Identifizierungsmöglichkeit (genannt Adresse), was ihre Eindeutigkeit widerspiegelt.

[4] Vergleiche Kapitel 5.3.

Zum einen wird also eine Instanzumgebung für Modelle auf Basis des zuvor beschrieben Meta-Modells benötigt. Zum anderen ist es aber ebenso wichtig, die Modell-Instanz-en im Modell-Raum automatisiert nutzbar zu machen. Diese Instanzen müssen zugreifbar und veränderbar sein. Sie brauchen also eine Kommunikationsschnittstelle. Diese beiden Aspekte, sowie deren Integration in die Dynamik werden im Folgenden beschrieben. Ebenso wird untersucht, in wie weit die Aspekte durch existierende Systeme abgedeckt werden.

Im Normalfall werden heutige Modelle und Strukturen als Datenstrukturen verstanden. Dabei können diese Datenstrukturen zwar auch einen Prozess beschreiben (z.B. IEC 61512 [3]), jedoch sieht man die Instanzen als „tote“ Objekte an, die von außen verändert werden.

Definition: *Als Instanzumgebung werden Systeme verstanden, die Klassen zu Objekten zu instanziieren und eine Schnittstelle zur Kommunikation anbieten. Hierzu zählt insbesondere die Möglichkeit einzelne Objekte und Relationen innerhalb des Objektbaums durch ihre Identifizierungsmöglichkeit zu adressieren und hierüber auch zu manipulieren.*
Eine ***(modellgetriebene) Instanzumgebung*** *bietet dieses für Modell-Beschreibungen und ist somit eine Modellverwaltungsumgebung[1].*
Damit sind die modellgetriebenen Instanzumgebungen die Programme, die einen Modell-Raum darstellen.

Es ist wichtig, dass nahezu alle Modelle der Automatisierungstechnik sich in das Meta-Modell abbilden lassen und damit in der Instanzumgebung repräsentiert werden können. In Kapitel 4.5.1 wird hierfür ein entsprechendes Vorgehensmodell aufgezeigt.

2.4.1 Existierende Instanzumgebungen für Modelle in die AT

Als Basis sind zwei wesentliche Technologien zu nennen, auf denen die Konzepte dieser Arbeit aufbauen. Da im späteren Verlauf Details der Systeme zum Thema Kommunikation und Objekt-Verwaltung genauer analysiert werden, wird hier nur eine kurze Einführung über die Historie gegeben.

Für beide Systeme gilt, dass eine Referenzimplementierung durch ihre jeweiligen Organisationen bereitgestellt wird. Diese Referenzimplementierung ist für beide Systeme kostengünstig und relativ „frei“ verfügbar, sodass eine weite Verbreitung unterstützt wird.
Dieses Konzept verspricht, dass ein Großteil der grundlegenden Implementierungen gut getestet ist. Durch die gleiche Referenzimplementierung sind Produkte unterschiedlicher Hersteller einfacher zu entwickeln, wodurch sie eher kompatibel zu einander sind, da sie auf einer gemeinsamer Softwarebasis aufbauen.

[1] In dieser Arbeit werden ausschließlich modellgetriebene Instanzumgebungen betrachtet.

2.4.1.1 ACPLT-Technologien

Die ACPLT-Technologien wurden am Lehrstuhl für Prozessleittechnik der RWTH Aachen entwickelt. Hierbei handelt es sich um eine Instanzumgebung, in der Modell-Instanzen verwaltet, erkundet und verändert werden können.

Die wesentlichen Grundlagen wurden in zwei Dissertationen [Alb03], [Mey03] beschrieben. Während sich [Alb03] auf die Schnittstellen bzw. Dienste, die als ACPLT-Kommunikations-System (ACPLT/KS) bezeichnet wird, fokussiert, werden in [Mey03] die Objektverwaltung (ACPLT/OV) beschrieben. Diese beiden Arbeiten ergänzen sich, jedoch ist beispielsweise das Kommunikationssystem auch ohne Objektverwaltung nutzbar.

Diese Basis-"Pakete" der ACPLT-Technologien, die vom Lehrstuhl als Open-Source bereitgestellt, weiterentwickelt und gepflegt werden, werden an verschiedenen Stellen durch (kommerzielle) Produkte ergänzt. So existieren Funktionsbaustein-Systeme und ACPLT-KS-Server, die eine Kopplung an alle gängigen Prozess-Leitsysteme anbieten.

2.4.1.2 OPC-UA

OPC-UA stellt ebenso wie ACPLT eine Instanzumgebung mit Verwaltungs-, Erkundungs- und Manipulations-Schnittstellen für Modelle dar. Modelle werden dabei in einem Objekt-orientierten Meta-Modell beschrieben.

Die OPC Foundation definiert unterschiedliche Kommunikationsschnittstellen für die Automatisierungstechnik. Die bekannteste ist OPC, welches in seiner klassischen Form auf Windows-Technologien basiert und hauptsächlich zum Datenzugriff in unterschiedlichen Automatisierungs-Geräten eingesetzt wird.

OPC-UA stellt dafür ein umfassendes Meta-Modell bereit, welches in unterschiedlichen Verfeinerungen für viele Einsatzzwecke genutzt werden kann. Die Verfeinerungen – Profile genannt – beziehen sich dann beispielsweise auf eine Gerätebeschreibungsrepräsentation. Dabei basiert das Meta-Modell auf genau den Objekt-orientierten Prinzipien, die oben beschrieben sind. Zusätzlich werden Dienste beschrieben, um die zur Laufzeit zu verwaltenden Modelle abzufragen und zu erkunden. Anzumerken ist hierbei die erstmals in der Automatisierungstechnik verwendeten Verschlüsselungs- und Authentifizierungs-Verfahren mittels X.509 [17] Zertifikaten. Eine Datenübertragung (z.B. über das Internet) ist damit in der gleichen Weise abgesichert, wie beispielsweise auch Online-Banking abgesichert ist.

Weitergehende Standards nutzen OPC-UA mittlerweile als Basis. Als Beispiele seien hier Gerätekonfigurationen per FDI oder FDT in Version 2 genannt.

Nicht unerwähnt bleiben soll aber auch mindestens ein ausbauwürdiger Punkt in der aktuellen OPC-UA Spezifikation: Es wird häufig bemängelt, dass keine Mechanismen bereitgestellt werden, um einen Teilbereich des aktuellen Instanz-Modells in einem OPC-UA-System zu laden

oder zu entladen, was z.B. bei einer Aktualisierung eines Teilsystems erforderlich sein kann. ACPLT stellt im Gegensatz dazu entsprechende Mechanismen bereit.

2.4.2 Aktive Komponenten im Modell: Dienste

Das hier beschriebene Konzept geht einen Schritt weiter als eine Instanzumgebung und lässt auch aktive Komponenten zu. Dieses sind Objekte, die selber ein Verhalten haben und ggf. auch Strukturveränderungen des Modell-Raums vornehmen können. Es erfolgt also eine Verschmelzung der Modelle zur Datenhaltung und zu Abläufe.

> **Definition:** *Wenn eine Instanzumgebung für Modelle auch in der Lage ist eine Ausführung für Objekte oder Komponenten anzubieten, kann von einer* ***Ausführungsumgebung*** *gesprochen werden.*

Als kleinste Einheit können dabei *aktive Komponenten* verstanden werden.

> **Definition:** ***Aktive Komponenten*** *können Aktionen auslösen. Die Ausführungsumgebung stellt dafür nötige Ressourcen und Schnittstellen bereit.*

Während die zuvor beschriebenen Ansätze der Kapselung durch Klassen und Komponenten auf Variablen beschränkt wurden, rücken die Schnittstellen und ihre Aufrufe in den Fokus.

Methoden, Funktionen, Prozeduren Eines der rudimentären Paradigmen der Programmierung sind die Prozeduren oder Methoden (auch: Funktionen)[5]. Sie sind in Programmiersprachen seit circa 1970 Standard und mit den Objekt-orientierten Erweiterungen, die ab circa 1990 hinzukamen, vereinbar.

Methoden sind analog zu mathematischen Funktionen einfache Konstrukte: Sie bestehen aus Aufrufparametern und Rückgabewerten. Im Sinne der Kapselung verbergen sie ihre interne Realisierung. Eine Methode

$$Liste\ sortierteListe = sortiere(Liste\ unsortierteListe)$$

beispielsweise kann eine Sortierung anbieten ohne etwas über den verwendeten Sortieralgorithmus auszusagen.

Funktionsbausteine Funktionsbausteine werden zur Programmierung von Speicherprogrammierbaren Steuerungen (SPS) eingesetzt und in der IEC 61131-3 [4] spezifiziert. Sie werden - teilweise in grafischer Notation - verbunden, indem Ausgänge mit Eingängen anderer Funktionsbausteine vernetzt werden.

[5] Die Informatik unterscheidet innerhalb dieser Begriffe. In Bezug auf das Thema Kapselung, was hier beschrieben wird, ist dies jedoch nicht relevant.

Funktionsbausteine kapseln dabei ebenso wie Methoden ihre Funktion, wobei die Kapselung bei ihnen weniger auf alternative, algorithmische Implementierung ausgerichtet ist. Stattdessen wird von der Ausführungsumgebung auf der SPS abstrahiert. So können unterschiedliche SPSen durch eine einheitliche (oder ähnlich aussehende) *Programmiersprache der Funktionsbausteine* vorgenommen werden.

Aus diesem Grund gibt es auch standardisierte Funktionsbausteine in der IEC61131-3, die von nahezu allen Engineering-Systemen angeboten werden.

Hintergrund:
Die IEC 61499 ermöglicht die Projektierung von Funktionsbausteinen in einer verteilten Umgebung. Im Gegensatz zu der Signal-orientierten Kommunikation in der IEC 61131-3 werden hier zusätzlich Nachrichten / Events gesendet. Die bisher hauptsächlich im akademischen Bereich betrachtete Norm wird in dieser Arbeit nicht weiter analysiert.

Anwendungen stellen eine Lösung für ein Problem bereit. Sie dienen also beispielsweise dazu, dass ein Mensch mit einer Maschine interagieren kann, oder auch, dass ein Wertebereich überwacht wird und ggf. ein Alarm ausgelöst wird.

Definition: *Eine **Anwendung** stellt eine Lösung für eine Aufgabe bereit. Es ist ein Programm, welches eine modellgetriebenen Instanzumgebung oder Dienste nutzt.*

Dabei wird nicht näher spezifiziert, ob sie Teil einer Ausführungsumgebung ist (beispielsweise durch sich nutzende Dienste), ein einzelner Dienst ist oder sogar eine externe Anwendung ist, die die Instanzumgebung (zur Modell-Repräsentation) nutzt.

2.4.2.1 Dienste - ein Versuch der Erfassung des Begriffes

Aufbauend auf den aktiven Komponenten kann ein Dienst beschrieben werden.

Es wird hier versucht eine allgemeinverständliche, weiträumige Definition zu liefern, da der Begriff selber im Sprachgebrauch nicht konkret festgelegt werden kann. Es gibt in der Literatur viele, teilweise auch widersprüchliche Definitionen des Begriffes „Dienst“ oder „Service“. Der Begriff leitet sich dabei aus „Dienst-orientierten Architektur“ von OASIS [26] ab, wo der Begriff jedoch nicht scharf definiert wird.

Definition: *Ein **Dienst** ist eine in sich abgeschlossene, logische Einheit. Er wird auf einem Gerät instanziiert oder „deployed“. Er ist in dieser Arbeit eine aktive Komponente, die eine Schnittstelle anbietet. Zum Aufruf der Dienst-Schnittstelle wird zusätzlich die Adresse des Dienstes, also die Adresse der aktiven Komponente benötigt.*

Es existiert immerhin ein nicht-scharfes, allgemeines Verständnis von einem Dienst:

Verteiltes System Eine Dienst-orientierte Architektur adressiert immer ein verteiltes System. Es kann sich auch bestehend aus mehreren Programmen auf einem Gerät befinden. Es wird immer über eine zu Grunde liegende Kommunikationsstruktur zwischen unterschiedlichen physikalischen Geräten gesprochen oder eine solche als Basis definiert. Ein Dienst ist dabei ein Teil einer Gesamtfunktionalität eines Gerätes.

Keine unterschiedliche Bezeichnung für Klassen und Instanzen Im Gegensatz zu der Objekt-Orientierung wird bei Diensten zwischen der Beschreibung des Dienstes und seiner Instanz im Wortlaut nicht unterschieden. Es ergibt sich aus dem Kontext, ob es sich um eine Instanz eines Dienstes, die dann auch eine Adresse hat, gesprochen wird, oder um eine „Dienst-Klasse“, die ggf. instanziiert werden kann.

Interner Zustand Dienste können einen internen Zustand haben und bieten durch die Methoden Zugriffs- und Manipulationsmöglichkeiten für diesen Zustand und die verwalteten Daten.

Ausführungsumgebung / Container Ein Dienst selber basiert auf einer Ausführungsumgebung, die von der konkreten Hardware des Gerätes abstrahiert. Dieses ist notwendig, um einen Dienst auf unterschiedlichen Geräten deployen zu können.

Gegenseitiger Aufruf Dienste können sich gegenseitig aufrufen. Hierdurch können, durch die Verschaltung von Diensten mit einfachen Funktionen, insgesamt höherwertige Funktionen dargestellt werden.

Klassen von Diensten In Abbildung 2.8 wird versucht eine Klassifikation der unterschiedlichen Verständnisse von Diensten darzustellen. Im Wesentlichen gibt es hierbei drei Klassen

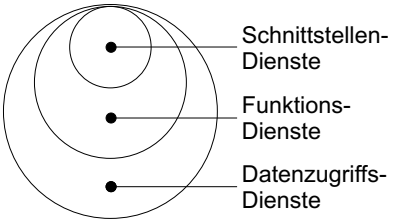


Abbildung 2.8: Versuch der Klassifizierung der Verständnisse zum Begriff „Dienst“.

von Diensten, wobei diese Klassen aufeinander aufbauen.

Die speziellste Klasse wird dabei *Schnittstellen-Dienste* genannt. Bei allen Verständnissen des Wortes „Dienst“ wird immer eine Aussage über Schnittstellen getroffen. Sie sind elementarer Bestandteil vor dem Hintergrund der Dienst-orientierten Architekturen.

Definition: *Unter einer **Dienst-Schnittstelle** wird die Aufruf-/Rückgabestruktur verstanden. Hierbei handelt es sich für einen Dienst um Aufrufparameter und ggf. das Antwortverhalten. Wenn über eine konkrete Instanz eines Dienstes gesprochen wird, beinhaltet die Schnittstelle zusätzlich die Adresse.*

Es existieren völlig unterschiedliche Vorstellungen, wie Schnittstellen zu spezifizieren sind. Beispielsweise bestehen WebServices meistens neben ihrer XML-basierten Kommunikation aus Schnittstellen-Spezifikationen in der WSDL-Sprache [31], die es ermöglicht WebServices-Aufrufe zu generieren und Antworten auszuwerten, also die syntaktischen Schnittstellenbeschreibung.

Als Vorläufer der WebServices können die *Remote-Procedure-Calls* (RFC 5322 [18]) gesehen werden. Auch hier existiert eine Schnittstellensprache, die jedoch zur Laufzeit nicht mehr genutzt wird. Der Gedanke hinter den RPCs war jedoch weniger die Dienst-Orientierung sondern eher einfache Methoden, wie sie oben beschrieben sind, über Netzwerk erreichbar zu machen. Im heutigen Sinne handelt es sich hierbei jedoch um einen Schnittstellen-Dienst. [6]

Eine übergeordnete Klasse der Schnittstellen-Dienste wird als *Funktions-Dienste* bezeichnet. Hierbei wird nicht nur über eine Schnittstelle gesprochen, sondern auch über die zu erbringende Funktion. Durch den internen Zustand eines Dienstes sind die Aufrufe nicht zwangsweise unabhängig voneinander.
Beispielsweise kann hier ein Inkrementierungs-Dienst gesehen werden: Bei jedem Aufruf gibt er einen erhöhten Wert zurück. Aber auch ein Archivierungs-Dienst kann hierunter fallen. Er stellt eine besondere Schnittstelle zu einem permanenten Datenspeicher dar und schreibt bzw. liest die Daten von dort.

Eine weitere Klasse der Dienste sind die *Datenzugriffs-Dienste*. Hierbei wird neben der Schnittstellenbeschreibung und der Funktion auch eine Aussage über die zu verwaltenden Daten gemacht. Datenzugriffs-Dienste teilen sich eine gemeinsame Datenbasis und bieten bestimmte Operationen auf dieser Datenbasis an.

So existieren beispielsweise Dienste in OPC-UA, um Daten abzufragen oder zu verändern. Alle manipulierten Daten der Dienste liegen dabei in dem einen OPC-UA Server, also der Instanzumgebung oder in dem Modell-Raum.

Orchestrierung & Choreographie: Dynamische Anwendungen Dienste können, wie oben beschrieben, sich untereinander aufrufen. Da viele Dienste existieren, existieren auch unterschiedliche Möglichkeiten des Ablaufs von Dienstaufrufen. Die Betrachtung dieser Möglichkeiten wird als *Orchestrierung* bezeichnet. Der konkrete Ablauf eines Vorgangs von Dienst-Aufrufen hingegen als *Choreographie*. In der IT-Welt existieren für beide Begriffe entsprechende Sprachen, die die Möglichkeiten weitestgehend beschreiben. Eine bekannte Orchestrierungssprache ist WS-BPEL [21] zu nennen. Für die Choreographie ist WS-CDL [33] ein Beispiel.

[6] Eine Komponente, welche einen Remote-Procedure-Call anbietet, ist im Sinne dieser Arbeit eine aktive Komponente. Der im späteren Verlauf vorgestellte Zustand einer solchen verhindert den Aufruf einer RPC, wenn der Dienst nicht bereit ist.

An dieser Stelle gibt es einige Arbeiten wie [Pel03], die versuchen die Begriffe im direkten Vergleich zu unterscheiden. Im allgemeinen Sprachgebrauch kann man aber nicht davon ausgehen, dass die Begriffe trennscharf verwendet werden, sodass von der allgemeinsten Annahme ausgegangen werden muss.

Definition: *Eine **Verschaltung** von Diensten beschreibt insgesamt eine (Dienst-orientierte) Anwendung.*

Um eine Verschaltung zu erreichen, müssen Dienste verwaltet werden; es muss also Komponenten geben, die Wissen über die Instanzen von Diensten verwalten. Diese müssen abfragbar sein.
Ein weiterer wichtiger Punkt ist die Beschreibung der Dienste: Eine einfache, untypisierte Liste der Adressen von Diensten ist nicht zielführend. Fragen nach dem Typ von Dienst sollten gestellt werden. Diese Aufgaben übernehmen Technologien, wie das Yellow-Paging. Hierbei werden (in Anlehnung an ein Branchen-Telefonbuch) die Dienste inkl. ihrer Typen verwaltet.
Im einfacheren Fall ist das Wissen natürlichsprachlich hinterlegt, sodass Menschen (manuell) die Verschaltung vornehmen. Sinnvoller erscheint, dass das Wissen Maschi-nen-auswertbar abgelegt wird. Nicht immer wird dabei eine reine Typ-Information ausreichend sein.

Herausforderungen Unabhängig von dem konkreten Verständnis von einem Dienst existieren Herausforderungen, die eine Dienst-Orientierung adressieren muss. In vielen (Forschungs-)Projekten sind immer wieder neue Konzepte für die Realisierung von Diensten für die Automatisierung entstanden. Hauptaufgabe ist dabei die Definition der Schnittstellen zwischen den Diensten. Dieses Problem ist auch heute noch ungelöst. So gibt es vielfach tragfähige Ausführungsumgebungen für Dienste, die konkrete Umsetzung der automatisierungstechnischen Funktionen bleibt aber aus.

Dazu ist insbesondere zu berücksichtigen, dass dieses Problem nicht ganzheitlich gelöst werden kann: Während Geräte auf den höheren Ebenen ausreichend Ressourcen bereitstellen, um beispielsweise auch Grammatiken zu verwalten und eine Überprüfung der Eingaben und Ausgaben vorzunehmen, stehen diese Ressourcen näher am Prozess nicht zur Verfügung.
Weiterhin ist die einfachere Integration von Geräten zwar wünschenswert, wird von der Industrie jedoch nur soweit unterstützt, wie es die eigenen Produkte nicht angreift. Die standardisierte Austauschbarkeit von Software und Hardware durch standardisierte Schnittstellen würde aber eine Austauschbarkeit ermöglichen, die Alleinstellungsmerkmale verhindert.
Mit diesen und anderen Argumenten ist eine Dienst-orientierte Architektur als Basis für die Automatisierung vom Feldgerät bis zur ERP Ebene nur schwer zu realisieren.

2.4.3 Existierende Ausführungsumgebungen für Dienste

Für Dienste aus der klassischen Informationstechnik gibt es unterschiedliche Ausführungsumgebungen; dort meist als „Container“ bezeichnet. Sie versuchen einem Entwickler von Diensten

umfangreiche Aufgaben abzunehmen und gleichzeitig eine Schnittstelle bereitzustellen, sodass sich der Entwickler auf seine eigentliche Aufgabe konzentrieren kann.

Zu den üblichen Aufgaben, die von einer solchen Ausführungsumgebung übernommen werden, zählen insbesondere:

Hardware-Abstraktion Durch ein unterlagertes Betriebssystem sowie die Ausführungsumgebung wird eine Abstraktion von der Hardware geschaffen.

Kommunikation Es werden Kommunikations-Mittel bereitgestellt, die auf einfachste Weise genutzt werden können. So werden Datenverbindungen verwaltet. Eintreffende Daten werden durch einfache Methodenaufrufe an den Code eines Entwicklers weitergereicht.

(De)Installations-Management Es wird eine einfache Möglichkeit bereitgestellt, einen Dienst zu installieren und anzubieten. Hierzu zählt auch, dass unterschiedliche Dienste verwaltet werden.

Erkundungsfunktion Die installierten Dienste können von außen gefunden, erkannt und angesprochen werden.

Am verbreitetsten sind die *Servlet-Container*, die eine Ausführungsumgebung für Java-Klassen bereitstellen. Als Beispiel für eine solche Ausführungsumgebung sei hier der „Apache Jakarta Tomcat Server“ [tom] genannt. Im Sinne der vorangegangenen Dienst-Definition handelst es sich also um einen Funktions-Dienst.

Im Gegensatz zu der oben beschriebenen Ausführungsumgebung für Modelle werden in einem Servlet-Container die Dienste jedoch gegeneinander abgeschottet. Das Ziel ist nicht gemeinsame Operationen auf Daten anzubieten, sondern vielmehr unabhängig voneinander die gleichen Ressourcen zu nutzen.

2.4.4 Aspekte von Anwendungen, Diensten und Apps

Zuvor sind die Begriffe der Anwendung und des Dienstes definiert worden. Die aktuell aufkommende Bezeichnung von Programmen als *App*s (z.B. [ZGPU12]) stammt aus dem „Mobile-Bereich“, also von Smartphones und Tablets.
Im Endeffekt kombiniert eine „App“ einige Aspekte von Anwendungen, aber auch von Diensten. Ein endgültiges Verständnis, was eine App ist, ist derzeit nicht zu finden. So fehlt auch eine konkrete Definition. Unterschiedliche Arbeitsbereiche und Anbieter stellen verschiedene Aspekte in den Vordergrund.

Apps setzen das Thema der Einfachheit gezielt um. Sie sind für den Endkunden-Markt entwickelt, wo Nicht-IT-Experten Apps erwerben und nutzen. Die Anwendungen in der Automatisierung werden von Experten ihrer jeweiligen Domäne entwickelt, jedoch von Domänen-fremden angewendet. Somit bieten die Prinzipien der Apps auch Möglichkeiten für die Automatisierungstechnik.

Im Rahmen dieser Arbeit sind einige wichtige Parallelen und Unterschiede zwischen Anwendungen, Diensten und Apps zu sehen.

Verzicht auf Konfiguration zur Inbetriebnahme Eine Anwendung auf einem klassischen Betriebssystem wird durch eine Installation auf dem Gerät betriebsbereit gestellt. Während der Installation bestimmt der Nutzer gewisse Ausprägungen der Anwendung - vom Installationsort auf der Festplatte bis zum Installationsumfang von optionalen Teilen der Anwendung.

Jede Ausführungsumgebung für Dienste bietet im Gegensatz dazu eine Konfigurations-freie Installations-Möglichkeit von Diensten. Je nach Realisierung werden unterschiedliche Mechanismen bereitgestellt, jedoch wird zur Installation eines Dienstes letztendlich immer nur eine Datei benötigt. Diese enthält neben dem eigentlichen auszuführenden Programm-Code auch entsprechende Meta-Informationen, die zur Installation nötig sind. Bei der Installation ist somit keinerlei zusätzliche Information nötig.

Apps stellen ebenso die Konfigurations-freie Installation bereit. Hierzu ist jedoch eine absolut einheitliche Ausführungsumgebung und Ressourcenbereitstellung auf allen Geräten nötig, welches nur durch eine starke Standardisierung oder Monopolstruktur (Ein Betriebssystem-Anbieter eines Smartphones definiert die Umgebung und Ressourcen zusammen mit dem Umgang mit den Ressourcen) realisiert werden kann.

Verzicht auf Kommunikation untereinander Anwendungen kommunizieren im ursprünglichen Sinne nicht untereinander. Es gibt jedoch Mechanismen um dieses zu ermöglichen. Sie werden im klassischen Umfeld jedoch wenig genutzt. Wenige Kommunikations-Standards zwischen Anwendungen haben es zu einer weitreichenden Verbreitung gebracht und viele diese Standards sind auf Netzwerk-Ebene definiert, d.h. sie sind gleichzeitig für die Kommunikation zwischen Systemen konzipiert. Beispielsweise sei hier COM / DCOM von Microsoft genannt.

Mit dem Begriff Dienst im Rahmen der Dienst-orientierten Architekturen wird verstanden, dass alleinstehende Komponenten auch untereinander direkt kommunizieren können. Die Orchestrierung, also die Verknüpfung von Diensten zu komplexeren Programmen, ist ein wesentlicher Bestandteil dieser Architektur.
Hierfür müssen Schnittstellen spezifiziert werden und diese Spezifikation wird auf Sender- wie auch Empfängerseite bei der Programmierung umgesetzt.
Viele Projekte zur Standardisierung beschäftigen sich ausschließlich mit der Definition von Schnittstellen. Zum Teil geschieht dies sogar auf Meta-Ebenen. Als Beispiel sei hier das WS-RF genannt, welches einen asynchronen Aufruf von WebServices definiert. Erst auf WS-RF aufbauend werden dann konkrete Nachrichten definiert.

Im Gegensatz zu Diensten interagieren Apps nicht (oder kaum) miteinander. Sie sind aus dieser Sicht eher traditionelle Anwendungen. Dieser bewusste Verzicht auf Schnittstellen stellt einen wesentlichen Erfolgsfaktor dar. Ein Entwickler programmiert gegen Schnittstellen des Betriebssystems, welche von einem Hersteller bereitgestellt werden. Für unterschiedliche Versionen eines Betriebssystems gibt es so auch entsprechende Dokumentation über die Unterschiede.

Verzicht auf spezifische Schnittstellen Wie bereits geschrieben kommunizieren Anwendungen nicht direkt untereinander. Durch die unterlagerte Schicht sind jedoch indirekte Kommunikationsformen vorgegeben, die genutzt werden. Die erfolgreichsten sind der gegenseitige Aufruf von Anwendungen oder der Austausch von Nutzerdaten. Es handelt sich also um generische Schnittstellen.

Im Gegensatz dazu verfolgen Dienste das Konzept der spezifischen Schnittstellen. Für jede Interaktion von Diensten kann und wird eine eigene Sprache definiert und verwendet.

Im Bereich der Apps werden nun neue, generische Formen durch Betriebssysteme eingeführt, die eine Kommunikation zwischen den Apps direkt ermöglichen, sich dabei jedoch auf einfachste Formen beschränken.
Hier seien die Intents[7] (vgl. Kapitel 4.3.7) genannt. Der Beispiel-Intent *url-open*: `http://www.plt.rwth-aachen.de` sorgt dafür, dass alle installierten Apps, die eine URL öffnen können (also beispielsweise alle Browser) die Möglichkeit haben, den Intent zu bearbeiten.
Auf diese Weise erfolgt eine komplette Entkopplung der Apps. Eine direkte Interaktion ist technisch zwar meist möglich, wird aber in den seltensten Fällen genutzt. Dieses ist gleichzeitig eine der Hauptunterschiede zwischen den sogenannten Apps von modernen Betriebssystemen und den Dienst-orientierten Architekturen.

Verzicht auf Verteilung Anwendungen sind von sich aus nicht-verteilt. Dienst-orientier-te Architekturen definieren sich über die Verschaltung (Orchestrierung) von einzelnen Dienst-Instanzen. Ein wesentlicher Aspekt der entsprechenden Ausführungsumgebung geht deswegen immer in Richtung „Discovery“ (Auffinden von Diensten) und Verwaltung der Dienste.

Bei einer Ausführungsumgebung für Apps entfällt dieser wesentliche Aspekt. Apps sind unabhängig voneinander ausführbar. Sie basieren einzig und allein auf der Ausführungsumgebung und ihren Fähigkeiten. Wenn eine Fähigkeit durch die Plattform nicht gegeben ist (beispielsweise veraltete Version), wird die App schon nicht zur Installation angeboten. Apps enthalten hierfür eine Meta-Beschreibung von ihren Anforderungen.
Apps selber kommunizieren somit auch ausschließlich mit festgelegten, expliziten Servern beispielsweise Cloud-Anbieter im Internet. Eine Verteilung auf unterschiedliche Geräte aus Skalierungsgründen erfolgt für Apps transparent und intern von den Cloud-Anbietern.

Fazit der Aspekte Alle Aspekte zeigen, dass Einfachheit der Schlüssel zum Erfolg ist. Dienste ohne ihre komplexen Aufruf- und Verteilungs-Mechanismen entsprechen im Wesentlichen den Apps, wie sie heute weit verbreitet sind. Auch Anwendungen ohne Ihren Installationsaufwand und ohne ihre Hardware-Abhängigkeit können als Apps angesehen werden.

Die im späteren beschriebenen, konkreten Dienste integrieren den Gedanken des Verzichtes auf komplexe Mechanismen ohne ihn zu verbieten. Denn für einzelne Probleme kann es immer sinnvoll sein, eine Ausnahme von der angestrebten Einfachheit vorzuziehen.

[7] Namensgebung aus dem Android Betriebssystem

2.4.5 Existierende Ausführungsumgebungen für Modelle und Dienste

Die schon zuvor referenzierten Systeme OPC-UA und ACPLT sind nicht nur Instanzumgebungen für Modelle sondern auch Ausführungssysteme mit einigen Besonderheiten.

ACPLT/OV Die Objektverwaltung von ACPLT bietet eine Instanzumgebung für Modelle. Modelle können geladen und wieder entladen werden. Die Selbstbeschreibung ist in vollem Umfang vorhanden. Von jedem Objekt kann also erkundet werden, von welcher Klasse es instanziiert wurde und ebenso wie die Klassen untereinander vererbt sind.

Methoden-Aufrufe selber sind in ACPLT/OV nur innerhalb des Systems erlaubt. Da aber Variablen-Werte gesetzt werden können, kann hierüber ein interner Methoden-Aufruf initiiert werden. Somit entfallen die Schnittstellenbeschreibungen bzw. diese werden vom System nicht unterstützt.

Die Ausführungsumgebung für ACPLT/OV ist prinzipiell erweiterbar, jedoch der normalen Betriebsart einer SPS nachempfunden. Nach einer Registrierung können Objekte sich zyklisch aufrufen lassen. Eine Scheduling-Komponente übernimmt dabei die Aufgabe der Verwaltung.

OPC - Unified Architecture Die Spezifikation von OPC-UA bietet durch „Programs“ (Part 10) eine Ausführungsumgebung für Modelle. Sie verlässt sich aktuell auf die Betriebssystem-Mittel. Es findet keine Abstraktion hiervon statt. Da OPC-UA jedoch aktuell hauptsächlich als Schnittstelle zu unterlagerten Systemen (wie einer SPS) verstanden wird, ist diese Funktion meistens in das unterlagerte System ausgelagert - und Eigenschaften der Ausführung sind in OPC-UA nur eingegrenzt abgebildet.

2.5 Verteilte Systeme

In vielen Bereichen finden sich verteilte Systeme. Insbesondere in der Informationstechnik haben sich dabei Bereiche gebildet, die grundsätzlich mit der Aufgabenstellung Ähnlichkeiten besitzen. Insbesondere zu nennen sind hier das „Grid-Computing“ und die aktuellen Themen des „Cloud-Computing“. Ihnen ist jedoch gemein, dass sie nicht die Verknüpfung von strukturierten Daten oder sogar Modellen adressieren. Während das Grid-Computing ganz klassisch im Bereich der Dienst-orientierten Architekturen Schnittstellen definiert um eine Interoperabilität zu schaffen, steht beim Cloud-Computing eher die Anwendung zum Anwender im Fokus. Diese soll immer verfügbar und hochgradig skalierbar sein.

In der Automatisierungstechnik ist aus verschiedenen Gründen der Einsatz solch hoch-dynamischer Systeme bisher nicht erfolgt. Die Dynamik des Internet mit permanent hinzukommenden und wegfallenden Teilnehmern, mit unterschiedlichen Kommunikationswegen der gleichen Teilnehmer und mit einem hohen Datenaufkommen zu unterschiedlichen Zielen sind Anforderungen, die in einer Produktionsanlage in der Ausprägung aktuell nicht zu sehen sind.

Bisher ist es in der Automatisierungstechnik nicht üblich, dass existierende Systeme ausgehende, eigenständige Kommunikation mit anderen, gleichartigen Systemen aufnehmen. Die Kommunikation erfolgt im Sinne der Automatisierungspyramide immer von oben nach unten, in seltenen Fällen auch innerhalb einer Ebene. Insbesondere OPC-UA bietet zwar Konzepte, um dieses aufzuweichen, jedoch wird es in realen Anlagen meist eher - wie auch traditionelles OPC - als zentrale Sammelstelle für Daten einer (Teil-)Anlage eingesetzt. Es stellt also beispielsweise mittels der realisierten Modelle bestimmte Prozesswerte bereit, die ein OPC-UA-Server aus dem Feld bzw. den Steuerungen aggregiert hat. Diese werden dann von überlagerten Systemen abgeholt. OPC-UA und der Modell-Raum stehen hierbei eher als Interoperabilitäts-Schicht zwischen Herstellern.

Die Verteilung von Modellen auf unterschiedliche Geräte ist in der Automatisierungstechnik hingegen bisher nicht adressiert. Sie wird auch in keinen existierenden Instanzumgebungen unterstützt.

Das aufkommende Thema „BigData“ - der Analyse von großen Datenmengen - könnte für Modelle und ihren Einsatz in der Realität in vielen Fachbereichen einen Fortschritt liefern. Da bei dem Thema insbesondere Daten im Fokus stehen, die nicht in Modellen abgelegt sind, ist es denkbar, dass die Modelle als Annotationen für die unstrukturierten Daten dienen.

Hintergrund:
Selbst, wenn die eigentliche Kommunikationsrichtung umgekehrt ist (z.B. SPS benötigt Qualitätsdaten über ein Produktionsgut aus dem MES System), wird davon nicht abgewichen: Die SPS stellt eine Anfrage als Variablen-Werte im OPC-UA-Server bereit und das MES-System „pollt“, ob eine Anfrage vorliegt und stellt die benötigten Informationen in einer anderen Variable bereit.
Um dieses Problem zu adressieren, bieten die Hersteller unterschiedliche, proprietäre Lösungen an. Eine standardisierte Lösung durch eine ausgehende Kommunikation von einer SPS ist nicht vorgesehen.
Die von der PLCopen und OPC Foundation spezifizierten OPC-UA-Kommunikationsbausteine ermöglichen dieses ineffiziente Verhalten umzudrehen: Per Methoden-Aufruf wird zu dem Zeitpunkt, wenn die SPS die Werte benötigt, eine Funktion auf MES-Ebene angestoßenen und direkt die benötigten Informationen abgefragt.

3 Analyse der Anforderungen

Perfektion ist nicht dann erreicht, wenn man nichts mehr hinzufügen, sondern wenn man nichts mehr weglassen kann.
Antoine de Saint-Exupéry

Die aufgezeigten Aspekte der derzeitigen Automatisierungstechnik und insbesondere des Engineerings bergen erhebliche Entwicklungspotenziale. Ausgehend von Erfahrungen an die zu unterstützenden Modelle wie auch die vorhandenen Instanzumgebungen (Kapitel 2.4.5) werden zusätzliche Anforderungen an die zu formulierende, verteilte und modellgestützte Ausführungsumgebung gestellt.

3.1 Ergänzende Anforderungen an Geräte und Umgebung

In Bezug auf die Hard- und Software gelten gleiche Voraussetzungen, wie für die vorgestellten Ausgangssysteme ACPLT und OPC-UA.

Es müssen Rechenkapazitäten bereitstehen, die dynamisch genutzt werden können, um die eintreffenden Dienst-Aufrufe sowohl an die Anwendungen wie auch die Instanzumgebung abzuarbeiten. Arbeitsspeicher muss bereitstehen, um die Modelle und ihre Informationen vorzuhalten. Für die Kommunikation im verteilten System müssen entsprechende Ressourcen bereitstehen. Alle diese Ressourcen werden für die Konzepte vorausgesetzt, da sie sich an das unterlagerte System richten.

3.2 Ergänzende Anforderungen an Meta-Modell und Instanzumgebung

Von unterschiedlicher Hardware- oder Software-Basis, die unterschiedliche Anforderungen an Realisierungen (wie z.B. Programmiersprachen oder APIs) stellt, wird an dieser Stelle abstrahiert. Die jeweilige Implementierung entscheidet über die konkreten Möglichkeiten.

Wie schon für die Ausgangssysteme gilt, dass ein Meta-Modell bereitgestellt werden muss. Entsprechende Instantziierungs-Vorgänge und Abfragen (Dienste) müssen existieren.

Komponenten sollen zur Laufzeit nachgeladen werden können. Damit ergibt sich ein Art Bibliotheks-Konzept, welches hinzufügen und entfernen von Bibliotheken ermöglicht. Entsprechende Dienste zur Manipulation der existierenden Komponenten sind dementsprechend zu definieren. Hierdurch können Modelle geladen / entladen werden.

Hintergrund:
OPC-UA bietet hierfür weder in der Spezifikation Konzepte noch in der Stack-Implementierung eine Lösung. Ein Server muss immer angehalten und wieder gestartet werden um eine Änderung an den Bibliotheken vorzunehmen. Bei ACPLT/OV ist das Bibliotheks-Nachladen eine Standard-Funktion, die sowohl programmtechnisch wie auch manuell genutzt wird.

Wichtig sind Relationen über Grenzen von Geräten hinweg. Im Idealfall bietet das System sowohl zur Erkundung, wie auch zu Änderung von Relationen transparente Mechanismen, sodass das Gerät, welches die Informationen selber speichert, unerheblich für den Zugriff ist. Diese Eigenschaft sollte im Meta-Modell abgebildet sein, um eine transparente Verteilung zu erreichen.

Die Komponenten - also Instanzen von Objekten mit allen ihren Eigenschaften - sollten migriert werden können. Dieses ist wichtig, wenn sich eine Änderung in der Anlage ergeben hat, sodass eine Reorganisation sinnvoll ist. Die Relationen der Objekte werden dabei beibehalten.
Wichtige Eigenschaften gerade hierbei sind die Nachvollziehbarkeit und Atomität, sodass zwischenzeitlich inkonsistente Zustände vermieden werden oder die Nutzung in diesem Zeitraum verhindert wird. Hierfür können die AKID-Eigenschaften (**A**tomität, **K**onsistenz, **I**solation und **D**auerhaftigkeit; vergleiche [HR01]) herangezogen werden.
Gleichzeitig sind die AKID-Eigenschaften wichtiger Bestandteil für den kollaborativen Zugriff, also den Zugriff von mehreren Anwendungen zu konkurrierenden Zeitpunkten.

Hintergrund:
Die Entwicklungen im Bereich der föderierten und verteilten Datenbanken haben auch Konzepte der AKID-Eigenschaften für verteilte Systeme beschrieben. Diese können hier zur Anwendung kommen. Eine gute Übersicht bietet [Con97].

Es erscheint sinnvoll, in einer Instanzumgebung sowohl die Abläufe (Programme, Anwendungen), wie auch die „passiven“ Modelle zu verwalten. Dabei nutzen nur die Abläufe die zeitliche Ausführungsmöglichkeiten. Durch eine technologisch einheitliche Basis für passive Modelle und Abläufe wird ein Zugriff einfacher und letztendlich auch effizienter sein. Trotzdem ist es selbstverständlich möglich, dass externe Komponenten, durch die Zugriffsmechanismen mit den Modell-Instanzen kommunizieren.
Anforderung ist also im Idealfall die beiden Arten von Objekten - nämlich passive Repräsentationen und aktive Komponenten - in einer gemeinsamen Ausführungsumgebung zu beheimaten.

Ein wichtiger Aspekt um eine sinnvolle Verteilung automatisiert zu erreichen, ist die Ressourcenverwaltung. Durch die Quantifizierung und den Vergleich von vorhandenen Ressourcen lassen sich Rückschlüsse ziehen, wie eine Verteilung auszusehen hat.
Die Verteilung kann jedoch auch nach statischen Regeln erfolgen, die weniger an der Geräte-Auslastung sondern an anderen Merkmalen festgemacht wird. Beispielsweise können bestimmte Modelle im Instanzsystem genau dort abgelegt werden, wo ihre entsprechende Anwendung hauptsächlich ausgeführt wird.
In [ME11] wird ein entsprechendes Ressourcenmodelle vorgestellt, welches auf diese Instanzumgebung angewendet werden kann.

3.3 Ergänzende Anforderungen an die Kommunikation

Die Kommunikation ist für ein verteiltes System die entscheidende Komponente. Voraussetzung für die Erfüllung aller Funktionen ist eine bidirektionale Kommunikation - im Normalfall also ein Netzwerk, welches eine Kommunikation zwischen den Systemen ermöglicht. Für entsprechende Basisdienste zum Hinzufügen und Entfernen der Modelle kann die NAMUR Empfehlung 139 [20] herangezogen werden. Sie wird in Kapitel 4.3 ausführlicher besprochen.

Ein wichtiger Punkt bei der Übertragung zwischen Geräten ist die semantisch definierte Übertragung von Datentypen. Eine Zahl sollte also beim Empfänger wieder als Zahl kodiert sein und dieses sollte durch das Kommunikationssystem sichergestellt werden. Dabei ist zu gewährleisten, dass nicht nur atomare Datentypen (Zahlen, Zeichenketten, ...) zu übertragen sind, sondern auch komplexe Datentypen (Strukturen oder fest definierte Reihen von Datentypen).

Hintergrund:
ACPLT stellt derzeit nur konzeptuell Datenstrukturen bereit, die der Nutzer definieren und atomar übertragen kann. OPC-UA kennt entsprechende Konstrukte. Gutes Beispiel sind hier die Methoden-Aufrufe, die dazu verwendet werden können einen Datensatz *en-Block* zu übertragen, zu verarbeiten und eine entsprechende Antwort zu liefern.

Hintergrund:
Ausgangspunkt für die Repräsentation ist in unterschiedlichen Ansätzen eine Beschreibung in Form einer XML Grammatik. Das zu übertragene XML Dokument stellt dann einen komplexen Datentypen dar. Die Übertragung selber erfolgt nicht zwangsläufig in XML. OPC-UA hat beispielsweise eine optimierte Binärdarstellung definiert, die auch im Normalfall verwendet wird.

Damit eine Kommunikation erfolgen kann, muss ein Kommunikationsmedium gegeben sein. Als Voraussetzung an das unterlagerte System wird davon ausgegangen, dass alle Teilnehmer untereinander direkt kommunizieren können, d.h. eine „Vollvermaschung“ der Teilnehmer auf logischer Ebene wird vorausgesetzt. Sobald ein einheitliches Netzwerk verfügbar ist, ist diese Bedingung erfüllt.

Damit die Komponenten und Anwendungen in einem verteilten System die Lokalität des Kommunikations-Zieles nicht berücksichtigen müssen, soll eine transparente Kommunikation gegeben sein. Explizit gilt hierfür, dass eine Kommunikation zu einer lokalen Komponente mit den gleichen Mitteln zu erfolgen hat, wie eine zu einer Entfernten. Zusätzlich können auch Kommunikationsmittel bereitgestellt werden, die nicht transparent sind - beispielsweise nur lokal zugreifbar sind.

Das Kommunikationsmedium sollte nach Möglichkeit eine Empfangsgarantie bieten. Dabei wird sichergestellt, dass eine gesendete Nachricht auch bei der Empfangskomponente angekommen ist, d.h. dass zum einen das Zielgerät erreichbar ist und zum anderen die Ziel-Komponente existiert. Ist dieses nicht der Fall, bekommt der Sender eine entsprechende Fehlermeldung. Die Empfangsgarantie sagt jedoch nichts über die Verarbeitung bzw. den Verarbeitungszeitpunkt aus.
Eine dem Kommunikations-Medium überlagerte Schicht kann hier durch eine zusätzliche Quittierung eine solche Empfangsgarantie für die Kommunikation anbieten, wenn sie benötigt wird. Bietet das Kommunikations-Medium von sich aus jedoch schon eine Empfangsgarantie, sollte diese unbedingt genutzt werden.

Auch wenn die Ressourcenverwaltung in dieser Arbeit nicht im Fokus steht, sei insbesondere auf die Herausforderungen der Laufzeiten in Bezug auf die Kommunikation, also das Antwortverhalten, hingewiesen. Diese müssen in einem Ressourcenmodell ggf. repräsentiert und berücksichtigt werden.

Die Instanzumgebung sollte Kommunikation zwischen Geräten grundsätzlich nutzen, um Informationen über den Zustand des Kommunikationspartners zu erfahren. Auf diese Weise kann ggf. eine explizite Überwachung indirekt erreicht werden.

3.3.1 Einheitliche, allgemeine Adressierung

Eine weitere, wichtige Anforderung im Bereich der Kommunikation ist die Adressierung. Unter Adressierung wird verstanden, die Partner einer Kommunikation festzulegen. Wichtig ist dabei in jedem Fall, dass die Partner eindeutig identifiziert werden können.

Eigentlich ist die Adressierung damit eine Identifizierung, d.h. in erster Linie deutet eine unterschiedliche Adresse auf unterschiedliche Kommunikationspartner hin. Erst in zweiter Linie ist eine Adresse ein Mittel, um Kommunikation an diesen Kommunikationspartner zu richten.

Durch die unterschiedlichen Kommunikations-Medien wird deutlich, dass es auch unterschiedliche Adressierungsarten gibt. Dabei ist wichtig, dass Anwendungen, die auf einer Ausführungsumgebung aufbauen, eine Adresse durchaus als atomares Datenkonstrukt handhaben könne, d.h. es betrachtet die interne Struktur einer Adresse nicht. Damit muss die Anwendung keine Kenntnisse um das Kommunikations-Medium haben, welches die Realisierung einer transparenten Kommunikation über unterschiedliche Medien fördert. Daraus folgt direkt, dass die Anwendungen entkoppelt sind von dem Medium, über das sie kommunizieren. Wichtig ist wei-

terhin, dass Adressen unterschieden werden können. Wenn eine entsprechende einheitliche Repräsentation definiert wurde, müssen auch innerhalb des Kommunikationssystems Komponenten lediglich Wissen über einzelne Teile haben.

Zu beachten ist, dass die Adressierung nichts mit den Formen der Kommunikation (unicast, multicast oder broadcast - vgl. Kapitel 2.3.2) zu tun hat. Adressierung dient ausschließlich dem Identifizieren des Kommunikationspartners sowie dem Kommunikations-Aufbau durch das unterliegende Medium.

Somit ist es aus Sicht der Anwendungen erstrebenswert, ein Medien-übergreifende, einheitliches Adressierungs-Schema zu haben, über dessen internen (je nach Medium unterschiedlichen) Aufbau eine Anwendung keine Informationen haben muss.

3.4 Bezug der Anforderungen

Die beschriebenen Anforderungen sind nicht gleichberechtigt nebeneinander zu sehen. Die Anforderungen werden in zwei Weisen in einen Bezug gesetzt. Dieses wird in der Tabelle 3.4 dargestellt.
Zum einen wird ein Bezugspunkt beschrieben. Hierunter wird verstanden, ob die Anforderung an das unterlagerte System gestellt wird oder an das System, wie es in den folgenden Kapiteln beschrieben wird. Zum anderen wird eine Gewichtung vorgenommen. Dazu werden drei Prioritätsklassen definiert:

Voraussetzung (kurz: *V*) Eine solche Anforderung ist unerlässlich, damit ein entsprechendes System realisiert werden kann.

Optional (kurz: *O*) Eine solche Anforderung ist nicht unerlässlich, würde aber zu erheblichen Einbußen in dem Funktionsumfang führen, wenn sie nicht vorhanden ist.

Zusatz (kurz: *Z*) Eine solche Anforderung ist grundsätzlich verzichtbar, jedoch ermöglicht sie weitergehende Verwendung oder einfachere Handhabung.

	Bezugspunkt	**Gewichtung**
Geräte und Umgebung:		
Rechenkapazitäten	unterlagert	V
Arbeitsspeicher	unterlagert	V
Kommunikation	unterlagert	V
Meta-Modell und Instanzumgebung:		
Meta-Modell	unterlagert	V
Nachladbarkeit	unterlagert	O
System-übergreifende Relation	hier	V
Migration von Objekten	hier	Z
Aktive Komponenten	unterlagert / hier	O
Verwaltung der Ressourcen	hier	O

	Bezugspunkt	Gewichtung
Kommunikation:		
Vollvermaschung	unterlagert	V
(Basis-)Datentypen	unterlagert	V
Komplexe Datentypen	unterlagert	O
Transparente Kommunikation	hier	V
Empfangsgarantie	unterlagert	O
Antwortverhalten	hier	Z
Überwachung	hier	Z
Adressierung der Systeme	unterlagert	V

3.5 Nachvollziehbarkeit und Verständlichkeit

Eine wichtige Anforderung, die durch viele Aspekte beeinträchtigt wird, ist die Nachvollziehbarkeit.

Hierunter müssen zwei wesentliche Aspekte verstanden werden. Zum einen ist eine theoretische Nachvollziehbarkeit zu nennen. Das System darf dafür keine Reaktionen zufällig treffen. Zusammen mit einer Protokollierung aller Änderungen und Ereignisse ist das Verhalten des Systems *nachvollziehbar* oder *deterministisch*.
Es bietet sich an für Entwicklung, Engineering und Betrieb unterschiedliche Ebenen der Protokollierung einzuführen.

Allerdings wird gerade in einem verteilten System eine solche theoretische Nachvollziehbarkeit schnell durch die Nutzer nicht anerkannt, da sie nicht mehr den Eindruck haben, das System verstehen zu können. Es sollte also darauf geachtet werden, dass zusätzlich die Reaktionen verständlich sind. Ein Nutzer, der eine Änderung beobachtet oder auch initiiert, sollte die Reaktionen im System also erahnen und verstehen können, ohne dass dafür Protokolle und ähnliches zu lesen sind.

Insgesamt kann man also sagen, dass die Verständlichkeit wichtig ist für die Akzeptanz des Gesamtsystems. Die Nachvollziehbarkeit jedoch ist für die Entwicklung wichtig und ggf. auch um Zulassungsaspekte zu erreichen.

4 Modell-Architektur für dynamische, verteilte Systemstrukturen

Nutzen Sie das Wissen und Entwickeln Sie Systeme mit Stolz und Leidenschaft.

Maik Pfingsten, `http://www.zukunftsarchitekten-podcast.de`

In diesem Kapitel werden Konzepte für (Meta-)Modelle, deren Verteilung und Verwaltung vorgestellt. Der Fokus dieses Kapitels liegt dabei auf den Konzepten der Systemstruktur und insbesondere deren Repräsentationen; nicht auf den Schnittstellen- und Komponenten-Beschreibungen. Das folgende Kapitel 5 beschreibt Basis-Komponenten, die zur Verwaltung einer verteilten, modellgetriebenen Instanzumgebung nötig sind, sodass sich eine System-Architektur ergibt.

Dabei verdeutlicht ein Modell der AT-Geräte-Struktur als Beispiel, wie die Konzepte anzuwenden sind. Dieses einfach gehaltene Modell wird eingangs beschrieben. Anzumerken ist dabei, dass dieses Modell auch von der Instanzumgebung selber verwendet wird, da es eine Kommunikations-Struktur der adressierbaren, automatisierungstechnischen Geräte repräsentiert.
Somit verdeutlicht das Beispiel zum einen die vorgestellten Konzepte, zum anderen wird es durch die in Kapitel 5 beschriebenen Basis-Komponenten genutzt und ist damit selber Bestandteil der Instanzumgebung.

4.1 Beispiel-Modell: AT-Geräte-Struktur

Als *automatisierungstechnische Geräte* einer Anlage werden die Geräte verstanden, die insgesamt die Steuerung der Anlage übernehmen, d.h. diese Geräte reichen von den Sensoren und Aktoren, die direkt in den Prozess eingreifen, über die Automatisierungsstationen (SPS) und entsprechende Engineering-Systeme bis zu den MES- und Asset-Management-Geräten.

Eine modellgetriebene Instanzumgebung benötigt dieses Modell der AT-Geräte-Struktur, um Aussagen über Geräte abbildbar zu machen. Dieses wird erreicht, indem Relationen von anderen Modellen zu Geräten dieses Modells erstellt werden. Gleichzeitig werden intern die entsprechenden Informationen des Modells benötigt; beispielsweise zur Modellierung von Kommunikationsverbindungen oder zur Darstellung der Komponenten-Verteilung.

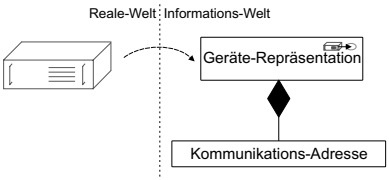


Abbildung 4.1: Vereinfachte Modellierung des AT-Geräte-Struktur Modells

Wie in der Abbildung 4.1 verdeutlicht, wird jedes Gerät, welches im Sinne des Konzeptes ansprechbar sein soll, als eine Instanz der Geräte-Repräsentation dargestellt. Als einziger Parameter wird die Kommunikations-Adresse modelliert. Sie reicht aus, um das Gerät selber eindeutig zu identifizieren.

Die hierdurch abgebildeten Geräte können durch weitere in der Instanzumgebung genutzte Modelle in Bezug gesetzt werden.

4.1.1 Erweiterung: Routing

Sollte keine Vollvermaschung der Kommunikation (vgl. Anforderungen in Kapitel 3.3) vorliegen, kann eine Ergänzung erfolgen.

In realen Anlagen existiert eine Vielzahl von unterschiedlichen Kommunikations-Medien. Eine Menge von auf dem Markt erhältlichen Kopplern oder Gateways zwischen den Kommunikations-Medien verdeutlicht diese Heterogenität.

Um diese Gegebenheit zu adressieren und gleichzeitig die Mächtigkeit der modellgetriebenen Instanzumgebung zu verdeutlichen, kann das AT-Gerätemodell durch eine *route_nach*-Relation erweitert werden. Wenn hierdurch die Gateway Komponenten erfasst und entsprechend in Relation gesetzt werden, kann ein zu realisierender Routing-Algorithmus als Teil einer Kommunikations-Komponente auch über Gateways hinweg eine Kommunikation herstellen.

4.2 Abbildung der Realität: Repräsentationen im Modell

Wie in Kapitel 2.1 beschrieben, bilden Modelle Teile der Realität ab. Hierdurch wird klar, dass zu jeder Komponente, aber auch zu jeder Aufgabe / jedem Vorgang aus der Realität eine Möglichkeit geschaffen werden kann, um diese in dem Modell zu repräsentieren. Diese Teilung in passive und aktive Repräsentationen wird angewendet, weil sie grundsätzlicher Natur ist: Während die abgebildete Struktur „nur“ als (passive) Objekte in Modellen existieren und für andere zugreifbar / änderbar sind, sind abzubildende Aufgaben als Anwendungen zu modellieren, die somit die Verhaltensstruktur darstellen. Hierbei handelt es sich also um zeitliche Abläufe. Die können lediglich abgebildet sein, also als weitere passive Objekte im Modell-Raum existieren.

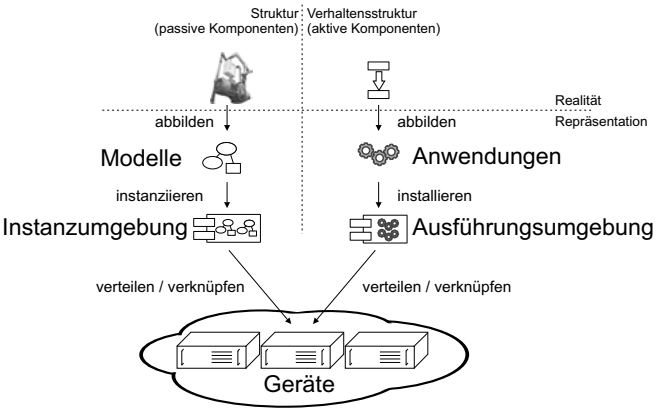


Abbildung 4.2: Realität von Komponenten und Ausführungen auf Geräten

Weiterführend ist es jedoch, wenn es sich um aktive Komponenten handelt, die entweder auf Änderungen hin reagieren oder auch *pro-aktiv* handeln können. Abbildung 4.2 stellt beide Arten gegenüber.
Diese Teilung von Struktur und Verhalten entspricht dem gewohnten Prinzip der Programmierung. Zum Vergleich: SPSen auf Basis von IEC 61131-3 verändern durch ihre programmierte Steuerung (aktives Element) Prozessabbilder von Ein- und Ausgängen (passives Element), die in jedem Zyklus gelesen und geschrieben werden. Programme in der IT (aktive Elemente) arbeiten häufig auf Datensätzen (passive Elemente). Auch im Software-Engineering erkennt man diese Teilung beispielsweise am Modell-View-Controller Entwurfsmuster (siehe [Bal08]).

In Kapitel 2.4 wurde bereits dargestellt, dass für eine Verwendung der Modelle eine Instanzumgebung sinnvoll ist. Hier werden die Objekte, aber auch die zugehörigen Modell-Informationen (Klassen) zugreifbar. Als Erweiterung soll eine Ausführungsumgebung existieren - optional als Bestandteil der Instanzumgebung - die auch zeitliche Abläufe, wie die Anwendungen, zulässt.

Definition: *In der Informations-Welt existieren **passive Komponenten**. Sie repräsentieren Informationen der realen Welt, nehmen aber selber keine Änderungen an dieser oder den Informationen in der Instanzumgebung vor. Im Normalfall werden passive Komponenten der Informations-Welt durch Modelle in ihrer Form und Abhängigkeit beschrieben. Entsprechende Instanzen werden in der **Instanzumgebung** verwaltet und können dort durch entsprechende Schnittstellen abgefragt und verändert werden.*

Definition: ***Aktive Komponenten** gehören zu der Ausführungs-Welt. Hier werden Aufgaben, die in der Anlage zu vollbringen sind, durch Anwendungen abgebildet. Diese Anwendungen können in einer zu beschriebenen **Ausführungsumgebung** installiert werden. Die Ausführungsumgebung bestimmt einen Ort der Ausführung indem eine Verteilung auf den Geräten verwaltet wird.*

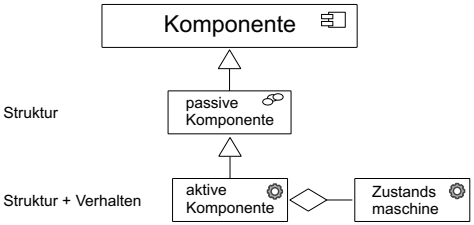


Abbildung 4.3: Modell der passiven und aktiven Komponenten

Diese beiden Umgebungen nutzen schlussendlich Ressourcen auf den bereitstehenden Geräten sowie die Kommunikations-Medien zwischen den Geräten.

Als Anschluss an existierende Begriffe wird in Abbildung 4.3 verdeutlicht, dass aktive und passive Komponenten beides Spezialisierung einer Komponente sind.

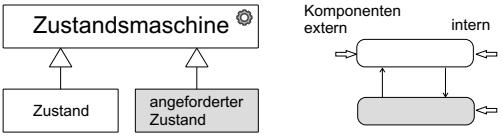


Abbildung 4.4: Grundlagen der Zustandsmaschine einer *aktiven Komponente*

Aktive Komponenten besitzen grundsätzlich eine Zustandsmaschine (Abbildung 4.4. Damit wird das Verhalten[1] selbst zu einem erkund- und änderbaren Strukturmodell. Ein solches Prinzip ist für passive Komponenten nicht wichtig, jedoch kann ein Zustand (keine Zustandsmaschine!) zur Repräsentation auch erforderlich sein.

Eine Besonderheit gilt für Programme, die außerhalb der Ausführungsumgebung laufen: Sie können eine Repräsentation einer aktiven Komponente besitzen um am Gesamtsystem teilzunehmen. Dabei sind sie selbst für die Synchronität der Zustandsmaschine (und weiterer Informationen) verantwortlich.

4.2.1 Zustandsmaschine für aktive Komponenten

Da aktive Komponenten im Sinne der vorangegangenen Definition einen Ablauf haben, beinhalten sie auch einen sich verändernden Zustand. Dieser wird in der Repräsentation abgebildet, sodass er im Laufzeitmodell erkundbar ist. Insbesondere ist ein solcher Zustand wichtig bei der Instanziierung. Eine Komponente hat so beispielsweise die Möglichkeit, während des Instanziierungsvorgangs schon erkundbar aber nicht ansprechbar zu sein. In dieser Phase können beispielsweise Relationen aufgebaut werden oder Kommunikationspartner gefunden werden.

[1] Soweit es in der Zustandsmaschine abgebildet ist, denn der eigentliche ausgeführte Programmcode ist hierdurch nicht repräsentiert.

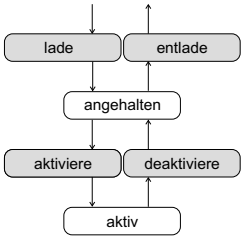


Abbildung 4.5: Beispiel einer einfachen Zustandsmaschine für die Interaktion mit der Ausführungsumgebung

Danach werden z.B. minimale Zustandsmaschinen für Dienste definiert, sodass die Ausführungsumgebung sich dieser Zustände, z.B. zur Migration von Diensten, bedienen kann.

Als Basis werden in einer Zustandsmaschine zwei Typen von Zuständen gesehen: *Zustände* und *angeforderte Zustände*. Durch diese Übergangszustände werden die Übergänge zwischen den Zuständen zeitlos. Ein Zugriff von außerhalb der Komponente kann die Zustandsmaschine nur in einen angeforderten Zustand bringen, worauf die Komponente folglich reagieren sollte und den zugehörigen „Zustand“ schlussendlich erreichen sollte. Die Komponente selber hingegen kann nur Übergänge zu einem Zustand vollziehen. Abbildung 4.4 verdeutlicht dieses.

Zustandsmaschine für aktive Komponenten und die Interaktion mit der Ausführungsumgebung Die Granularität mit der der Zustand abgebildet wird, hängt von der Komponente und dem Anwendungsfall ab. An dieser Stelle wird ein allgemeines Zustandsmodell beschrieben. Je nach Ausprägung des Gesamtsystems sind dabei einige Zustände nicht sinnvoll realisierbar, welches im Folgenden beschrieben wird. Hierdurch wird verdeutlicht, welche Voraussetzungen zum einen an die Entwicklung der aktiven Komponente, wie auch an die Ausführungsumgebung gestellt werden.

Die Instanziierung einer aktiven Komponente kann als *laden* bezeichnet werden. Hiernach befindet sich die Komponente erst mal in einem angehaltenen Zustand, womit sie „lediglich“ die Eigenschaften einer passiven Komponente hat. Sie kann von anderen Komponenten als aktive Komponente erst genutzt werden, wenn sie durch den Befehl *aktivieren* in den aktiven Zustand gebracht wird. Beim Laden können also einmalige Initialisierungsvorgänge vorgenommen werden - beispielsweise Ressourcen-Allokation. Beim *Aktivieren* werden hingegen wiederkehrende Initialisierungsvorgänge wie Verbindungsaufbauten vorgenommen. Durch den *deaktivieren* Befehl begibt sich die Komponente wieder in den gleichen Zustand wie nach dem *laden* und kann von hier aus auch entladen werden, sodass sie nicht weiter existiert. Abbildung 4.5 veranschaulicht die Zustände mit den entsprechenden Übergängen.

Dieses Modell ist auf die wesentlichen Aspekte, die zum Realisieren von Dynamik benötigt werden, begrenzt. Es stellt somit eine einfache Möglichkeit dar, die typischen Lebenszyklen mit den Anforderungen abzudecken. Es ist jedoch nicht als allumfassendes Zustandsmodell

gedacht. Wenn die Zustände selber als Klassen modelliert werden, können spezialisierte Zustandsmaschinen durch Vererbung geschaffen werden indem abgeleitete Klassen eine interne (feinere) Zustandsmaschine besitzen. So kann die Ausführungsumgebung lediglich die generischen, hier beschriebenen Zustände kennen; intern können jedoch feiner granulierte Abläufe modelliert werden.

Hintergrund:
Ein ähnliches Modell (speziell für Funktionsbausteine in verteilten Systemen) ist in der IEC61499 Part1 [6] definiert.
In OPC-UA sind solche Zustandsmodelle allgemein abbildbar. Dabei wird durch OPC-UA ein allgemeines Meta-Modell bereitgestellt, in dem Zustände und Übergänge definiert werden. Ein solches konkretes Zustandsmodell kann an beliebige Objekte, vornehmlich „OPC-UA Programs“ (Part 10 von [9]) angehangen werden, um den Verlauf abzubilden.

4.2.2 Die Komponenten-Repräsentation

In vorangegangenen Kapiteln wurde bereits die Abbildung der physikalischen, automatisierungstechnischen Geräte in einem Modell erläutert. An dieser Stelle wird eine allgemeine Repräsentation für Komponenten beschrieben, da eine Repräsentation nicht nur für Komponenten der realen Welt von Vorteil ist. Genauso werden auch Repräsentationen für aktive und auch passive Komponenten benötigt.

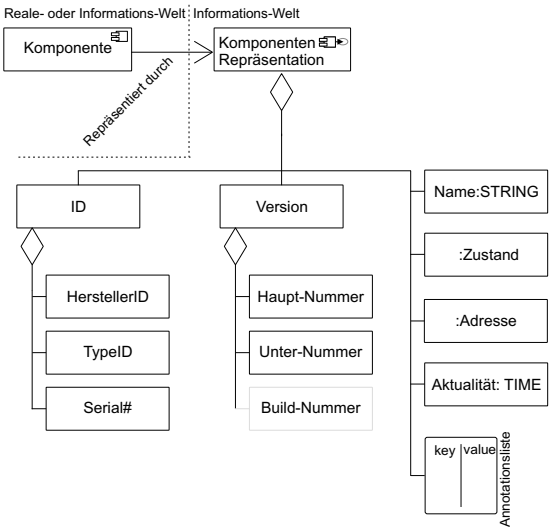


Abbildung 4.6: Detailaufbau einer Repräsentation

Elemente einer Komponenten-Repräsentation In Abbildung 4.6 ist die Struktur einer allgemeinen Komponenten-Repräsentation veranschaulicht. Es handelt sich also selber um eine passive Komponente, die alle statischen oder dynamischen Daten, die einer realen Komponente zugehörig sind, hält und so einheitlich zugreifbar macht.

Dabei bietet eine Repräsentation immer eine *ID*. Eine ID ist vom Verständnis her eine UUID im Sinne von [30], also eine systemweit (ggf. sogar weltweit) eindeutige Identifikations-Möglichkeit für ein konkretes Gerät. Sie wird hier als Struktur aus drei Teilen - jeweils als String abbildbar - aufgebaut. Eine *Hersteller-ID* sorgt dafür, dass Hersteller von Komponenten identifiziert werden können und gleichzeitig auch, dass sie unabhängig voneinander ihre IDs verwalten können. Ferner gibt es eine *TypeID*, die unterschiedliche Gerätetypen festlegt. Dabei spezifiziert der Hersteller eine ggf. interne Struktur des Typs und auch die Semantik, die in dieser Struktur liegt. Schließlich ergibt eine Serien-Nummer die Möglichkeit, Instanzen gleichen Typs eindeutig zu identifizieren.

Hintergrund:
Das Prinzip einer solchen strukturierten Identifikations-Möglichkeit ist weit verbreitet - Beispiele sind die USB-Spezifikation oder ISOBUS (ISO11783, [8]).

Versionen können ebenfalls strukturiert werden. Eine *Hauptnummer* gibt dabei die eigentliche Version an, eine Unterversion in erster Linie die Strukturierung in einer Produktreihe. Dazu wird hier verlangt, dass gleiche *Unterversionen* auch technisch kompatible, d.h. vor allem äquivalente Versionen bezeichnen. Um prinzipiell unterschiedliche, jedoch äquivalente Versionen auseinander zu halten kann eine *Build*-Nummer als dritter Teil hinzugezogen werden.

Hintergrund:
Ähnlich wird es in Quellcode-Verwaltungssystemen oder auch für Software-Produkte allgemein gehandhabt.

Um auch einen sprechenden Namen für eine Komponente angeben zu können, ist ein Feld *Name* vorgesehen. Als wichtig wird erachtet, dass dieses Datum nicht als Referenz dient, sondern nur der visuellen Darstellung für menschliche Nutzer. Als Referenz dient insgesamt hingegen die Adresse [2].

Hintergrund:
Entsprechend hat jede Node in OPC-UA einen *Displayname*n definiert, welcher für Anzeigen (auch internationalisiert) bereitsteht. Für die Referenzierung wird eine NodeId genutzt, welche eine Node eindeutig in einem Server identifiziert.

[2] Mitteils einer ID können Komponenten identifiziert, aber nicht adressiert (d.h. angesprochen) werden. Es müsste zusätzlich der „Speicherort“ der ID gesucht werden. Eine Adresse liefert beides. Der Typ der Adresse hängt vom Instanzsystem ab. Laut Anforderung müssen dort Instanzen zu identifizieren sein. Das Prinzip der EPRs im späteren Kapitel 4.3.1 stellt eine Möglichkeit der Strukturierung dar.

Der *Zustand* einer Repräsentation stellt den eigentlichen Zustand einer Komponente dar; ist hier jedoch nur ein Datenfeld.

In den später abgeleiteten Klassen der Komponenten-Repräsentation wird auf spezifische Überlagerungen dieses generischen Prinzips hingewiesen.

4.2.3 Unspezifizierte, flexible Annotationen für Repräsentationen

Einer der wichtigsten Teile einer Komponenten-Repräsentation ist seine Annotations-Liste (Abbildung 4.7), um zusätzliche Daten - allgemein und dynamisch - von den Komponenten bei deren Repräsentation ablegen zu können.

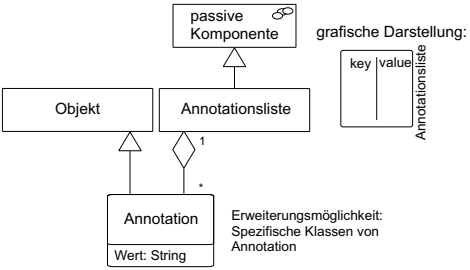


Abbildung 4.7: Liste von Annotationen zu einer Komponente - grafisch vereinfacht als Schlüssel-Wert-Tabelle

Dieses kann auf konzeptueller Ebene mit den Stereotypen von UML (vgl. UML-Profile in [25]) verglichen werden. Anwendungen annotieren Komponenten also an ihrer Komponenten-Repräsentation und ordnen ihnen so Eigenschaften zu. Diese sind veränderlich und dynamisch zur Laufzeit.

Diese Annotationen werden dabei allgemein als Schlüssel-Wert-Paar repräsentiert, welche gesammelt als Liste von der Komponenten-Repräsentation im Modell (vgl. Kapitel 4.2.3) verwaltet und abgelegt werden, sodass sie dort zugreifbar sind.

Hintergrund:
Die Idee für diese Liste ist abgeleitet aus den Headern nach der RFC 5322 „Internet Message Format“ [18], wie sie unter anderem auch für Mails verwendet werden.
Hier werden einige verpflichtende, einige freiwillige Schlüssel mit ihren Bedeutungen für den Mail-Versand definiert. Beispiel-Schlüssel sind *From*, *To* oder auch *Reply-To*, die entsprechende eMail-Adressen darstellen. Auf der anderen Seite sind alle Mail-Systeme berechtigt, neue Schlüssel-Wert-Paare in den Mail-Header einzufügen. Dieses passiert beispielsweise durch Spam-Erkennungs-Systeme. Der Empfänger einer Nachricht muss demzufolge wissen, mit welchem Spam-Erkennungs-System seine Mails untersucht wurden.

Durch eine Klasse *Annotation*, deren Instanzen den Namen des Schlüssels tragen und eine Variable *Wert* haben, wird eine minimale Basis-Eigenschaft definiert. Hiervon können sich die Anwendungen, die Annotationen hinzufügen, eigene Annotations-Klassen ableiten, die zusätzliche, Wert-Repräsentationen oder auch mehrere Werte unter einem Schlüssel ablegen. Daraus folgt also, dass jede Anwendung seine eigenen Annotations-Formate sowie Semantik für die Interpretation der ablegten Einträge in der Annotations-Liste mit sich bringt.

Beim Zugriff gilt ein kooperativer Ansatz, d.h. Komponenten sollten keine Werte überschreiben, die sie nicht selber angelegt haben. Ebenso sollten nicht weiter gepflegte Werte gelöscht werden. Man kann, wenn auch konzeptuell nicht notwendigerweise, den schreibenden Zugriff beschränken.

So können beispielsweise Geräteeigenschaften, die in einer solchen Liste abgelegt werden, nachträglich mit Informationen angereichert werden, die ein Gerät von Haus aus nicht über sich eintragen kann. So kann ein Betriebsstundenzähler installiert werden, der kontinuierlich als Annotation der Geräte-Repräsentation den Zeitraum des Betriebes unter einem vom Betriebsstundenzähler-Dienst definierten Schlüssel anhängt.

Auch können komplexe Erweiterungen wie Schnittstellenbeschreibungen von Diensten hier hinterlegt werden.

Vorteile der Annotationen Die Annotations-Liste stellt also eine zentrale Sammelstelle für Annotationen zu einer Komponente dar, die von unterschiedlichen anderen Komponenten zugeschrieben werden. Traditionell würde man dieses als interne Datenhaltung der anderen Komponenten verstehen, sodass jede Komponente seinen eigenen Daten-Raum schafft und in sich verwaltet.

Im Gegensatz zu einer internen Datenhaltung der Komponenten bietet dieser zentrale Ablage-Punkt von Annotationen mehrere Vorteile:

Zum einen können andere Komponenten hier ihre zu der Komponente gehörenden Annotationen ablegen, ohne selber eine Liste der Komponenten verwalten zu müssen.

Zum anderen werden die Annotationen in einem semi-formalen Format abgelegt und sind allgemein zugänglich, wobei ohne das Wissen über die Semantik der Schlüssel, die Werte nicht sinnvoll ausgewertet werden können. Auf diese Weise werden Annotationen von „fremden“ Komponenten einfach ignoriert. Etwas anders sieht es aus, wenn eine Anwendung den Inhalt einem Menschen repräsentiert. Dieser kann direkt dem Schlüssel eine Semantik und damit eine (vermutete) Bedeutung zuordnen.

Welche Daten als Annotation; welche als Datenfelder (abgeleitete Klassen) modellieren? Prinzipiell könnten auch die Elemente wie ID, Version, Aktualität, EPR und Name einer Komponenten-Repräsentation als Annotations-Element abgelegt werden.

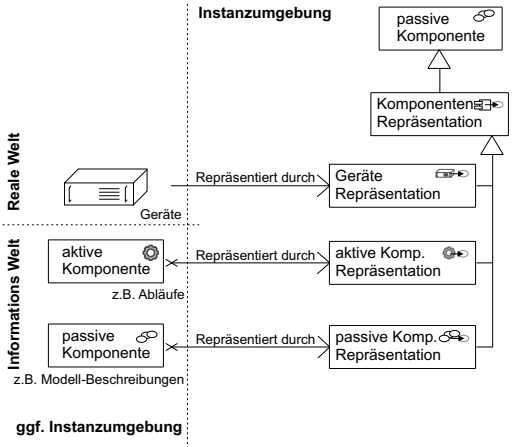


Abbildung 4.8: Konzept der Trennung von Komponenten und ihren allgemeinen Haushaltsdaten

Da es sich hierbei aber um grundsätzliche Elemente handelt, die jede Komponente als Repräsentation haben sollte, werden diese nicht in der Annotations-Liste hinterlegt. In dieser sind also ausschließlich Daten erfasst, die aus Sicht der Komponenten-Repräsentation optionale Informationen enthalten.

4.2.4 Unterschiedliche Komponenten-Repräsentationen

Das Prinzip der Repräsentationen lässt sich konsequent auf Komponenten im Allgemeinen anwenden, wie in Abbildung 4.8 dargestellt: Sowohl ein Gerät (etwas aus der realen Welt), wie auch aktive und passive Komponenten innerhalb der Modell-Landschaft besitzen eine Repräsentation. Teilweise fällt die Repräsentation mit der eigentlichen Komponente zusammen, da hierdurch weniger Verwaltungsaufwand entsteht.
Es ist jedoch auch möglich die Komponente und ihre Repräsentation zu trennen. Hierdurch wird es zum einen möglich, externe Komponente zu erkunden oder auch zu interagieren. Beispielsweise hat eine externe Anwendung eine Repräsentation und kann damit gefunden und genutzt werden. Zum anderen unterstützen separate Repräsentationen auch eine Verteilung des Modell-Landschaft auf unterschiedliche Geräte.

So wird es möglich, Relationen zwischen Modellen und den Komponenten darzustellen. Dabei spielt es keine Rolle, ob eine Komponenten-Repräsentation eine Repräsentation für eine aktive Software-Komponente (z.B. Anwendung oder Dienst), ein Gerät oder eine passive Software-Komponente, wie ein Modell, darstellt.

Damit eine Zuordnung der Komponenten-Repräsentation gegeben sein kann, wird die Relation *repräsentiert_durch* definiert. Sie zeigt von der eigentlichen Komponente auf ihre Repräsenta-

tion. Somit ist eine eindeutige Zuordnung zwischen der Komponente und ihrer Repräsentation gegeben. Selbstverständlich ist eine solche Relation nur realisierbar, wenn sowohl die Repräsentation als auch die eigentliche Komponente auf einer gemeinsamen Instanzumgebung aufbauen.

Eine der wichtigen Eigenschaften dieser Repräsentation ist, dass sie alle Eigenschaften und Informationen über eine Komponente modelliert, ohne die Komponente selber zu sein. Eine Repräsentation kann also z.B. auch entfernt verwaltet werden.

Ein weiterer Anwendungsfall ist eine Repräsentations-Kopie, die alle Eigenschaften der Repräsentation spiegelt - z.B. aus Performancegründen. Um die Aktualität vermerken zu können, verfügen die Repräsentations-Kopien über das Datenfeld *Aktualität*, in welchem der Zeitpunkt der letzten Aktualisierung hinterlegt wird.

4.2.5 Beispiel: AT-Geräte-Struktur als Komponenten-Repräsentation

Das AT-Geräte-Struktur-Modell basiert auf einer Repräsentation von Geräten im Modell-Raum, ist also selber ein Beispiel für diese Repräsentationen. Für die Verwendung des Modells zur

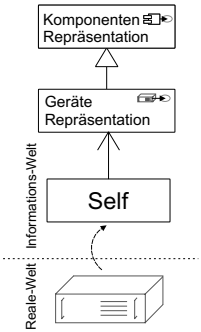


Abbildung 4.9: Das AT-Geräte-Struktur-Modell auf Basis der Trennung von Informationswelt und Realität

Laufzeit, wie sie genauer im späteren Kapitel beschrieben ist, wird jedem Gerät eine Instanz der Geräte-Repräsentation *Self* (Abbildung 4.9) zugeordnet. Dieses Objekt repräsentiert damit die Eigenschaften des Gerätes. Primär stellt es, wie zuvor beschrieben, die Kommunikations-Adresse dar. Aber auch ein Status, der durch die Oberklasse Komponenten-Repräsentation vorhanden ist, kann hier den allgemeinen Betriebszustand abbilden.

Werden aus anderen Modellen Aussagen über dieses Gerät getroffen, können diese durch entsprechende Relationen oder Annotationen dem Gerät zugeordnet werden, wodurch sie für Dritte erkundbar werden.

Als Annotationen könnte beispielsweise der aktuelle Ressourcenverbrauch der Geräte beschrieben werden. Sollten Anwendungen eine Auswahlmöglichkeit für das Deployment haben, können sie entsprechend wenig belastete Geräte nutzen.

4.3 Kommunikation im automatisierungstechnischen Kontext

Kommunikation ist das wichtigste Mittel in einem verteilten System. Es aktiv einzusetzen, ist eine der wichtigsten Grundlagen für die Automatisierung von Anlagen, basierend auf verteilten Modellen. Deswegen soll an dieser Stelle eine Einordnung der Kommunikationsschichten vorgenommen werden. Gleichzeitig werden Formen der Kommunikation beschrieben, die auf Anwendungsebene genutzt werden können. Die Unterschiede werden diskutiert und beschrieben.

Ziel ist es, die unterschiedlichen Eigenschaften der Kommunikation darzustellen. Bei einer Realisierung müssen die Ähnlichkeiten des Kommunikationssystems (Feldbus / Ethernet) mit den hier beschriebenen Anforderungen in Einklang gebracht werden.

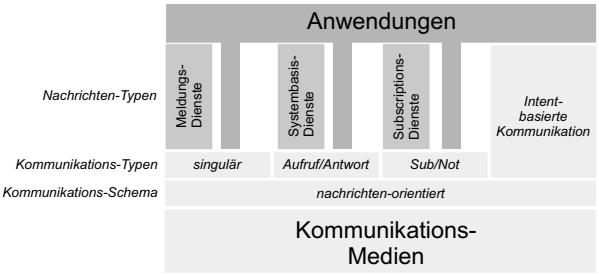


Abbildung 4.10: Zusammenfassendes Kommunikations-Verständnis

In Kapitel 2.3 wurde aufgezeigt, dass es in vielen Bereichen eine mehr oder weniger einheitliche Vorstellung von der Kommunikationsstruktur insgesamt gibt. Wie ebenfalls dort beschrieben, ist insbesondere die Mischung des reinen Transports von Kommunikations-Medien mit der Anwendungsebene kritisch zu sehen, da hierdurch eine Erweiterbarkeit sowie eine unabhängige Entwicklung von unterschiedlichen Produkten verhindert wird.

Um die unterschiedlichen Aufgaben und Anforderungen der Kommunikation in einer Automatisierung zu adressieren, sind unterschiedliche Bus- und Netzwerksysteme notwendig und üblich.

Das im Folgenden beschriebene Konzept berücksichtigt bisherige Arbeiten und versucht gleichzeitig die semantischen Ebenen in einer Weise aufzuteilen, dass zukünftige Entwicklungen sich klar in die drei Schichten einordnen lassen. Es ist als Überblick in Abbildung 4.10 dargestellt.

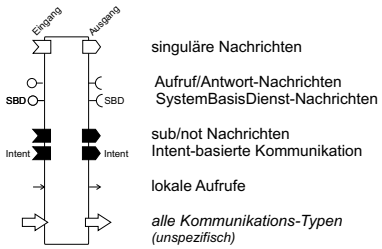


Abbildung 4.11: Visuelle Darstellung der Kommunikationstypen

Gleichzeitig wird für die Referenzierung und Adressierung eine einheitliche Adressierungsform auf Basis der „EndPointReferences“ vorgestellt [3].

Abbildung 4.11 stellt die im Weiteren verwendete grafische Notation dar. Zusätzlich zu den vier im Folgenden dargestellten Kommunikations-Typen wird als Sonderfall die lokale Kommunikation dargestellt. Eine allgemeine Notation soll *unspezifisch* darstellen, dass eine Komponente grundsätzlich kommunizieren kann.

4.3.1 Referenzierung über Systemgrenzen hinweg

Die transparente Kommunikation wurde in den Anforderungen in Kapitel 3 formuliert. Ebenso wurde beschrieben, dass eine einheitliche, allgemeine Adressierung benötigt wird.

Es ist sinnvoll, ganz allgemein eine Adresse zu definieren, die nicht durch das Kommunikations-Medium selber beschrieben wird und damit auch nicht von diesem System abhängig ist.

Durch Medien-spezifische Erweiterungen oder unterschiedliche Nutzung der Datenfelder einer solchen Adresse wird allen Komponenten im Gesamtsystem eine Handhabung der Adressen möglich und das ohne, dass sie sich um den Inhalt einer Adresse kümmern müssen.

Im Gesamtsystem gibt es drei unterschiedliche Arten von Adressen:

Kommunikations-Adresse Adresse (abhängig vom Kommunikations-Medium) um ein Gerät zu identifizieren. (Beispiel: Ethernet IP Adresse)

Dienst-Adresse Zugriffspunkt auf einen Dienst, oder ein Programm innerhalb eines Gerätes. (Beispiel: Ethernet Port)

[3] Viele der Aspekte im Folgenden sind ähnlich zu den in Kapitel 2.3 vorgestellten Systemen - es geht in diesem Kapitel also auch um ein einheitliches Verständnis, sodass für die modellgetriebene Instanzumgebung die Kommunikations-Anforderungen erörtert werden.

Komponenten-Adresse Referenz auf ein Element innerhalb des Dienstes; innerhalb des Instanzsystems eine Komponente/Objekt. (Beispiel: Pfad in einer URL, also „/untermenue/webseite.html“)

Die zuvor beschriebene Adresse soll genau eine Komponente auf einem Gerät identifizieren können. Um eine begriffliche Eindeutigkeit zu erreichen, wird für eine solche Systemweit-eindeutige Adresse der Begriff *EndPointReference* (EPR) in Anlehnung an die Spezifikation in „WS-Addressing“ genommen: Eine EPR (Abbildung 4.12) beschreibt den Zugriffspunkt auf

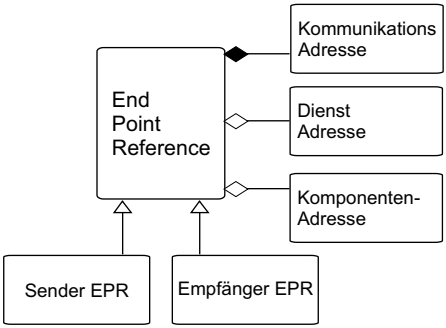


Abbildung 4.12: Verständnis des grundsätzlichen Aufbaus einer Adresse als *End Point Reference*

ein Ziel; eine Referenz. Das Ziel kann dabei unterschiedlicher Art sein, wie aus den obigen Begriffen hervorgeht.

Es kann wichtig sein, dass die Komponenten unterschiedliche Adressen auseinander halten können, indem sie die Datenfelder der Adresse vergleichen. Unterscheidet sich ein Datenfeld, sind es im Sinne der Adresse unterschiedliche Referenzen.

Sollte eine Komponente eine EPR zur Kommunikation nutzen, muss diese EPR zu dem Kommunikations-Medium passen. Andernfalls sind entsprechende Router-Dienste zu spezifizieren, die eine Brücke herstellen.

Mapping auf EPRs Eine solche Adressierungsmöglichkeit über die Systemgrenzen ist im Endeffekt neuartig. Jedoch bietet beispielsweise die EPR aus der W3C Empfehlung *Web Services Addressing* [34] eine sinnvolle Ausgangsbasis für eine Formulierung des allgemeinen Datensatzes „Adresse“.

WebServices sind - wie Eingangs beschrieben - ausschließlich auf XML definierte Kommunikationen und Beschreibungen. So sind auch EPRs laut [34] als XML definiert:

```
<wsa:EndpointReference>
    <wsa:Address>xs:anyURI</wsa:Address>
    <wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?
    <wsa:Metadata>xs:any*</wsa:Metadata>?
</wsa:EndpointReference>
```

Dieses heißt jedoch nicht zwangsläufig, dass auch eine XML-basierte Kodierung verwendet werden muss. Aus Performance-Gründen kann eine effizientere Repräsentation erfolgen, wie bereits in der Anforderungs-Phase ausgeführt.

Um die oben beschrieben Verknüpfung zu erreichen, erfolgt eine Abbildung der Teile einer EndPointReference auf die WS-Addressing Definition.

Kommunikations-Adresse `<wsa:Address>`

Dienst-Adresse `<wsa:ReferenceParameters>`

Komponenten-Adresse `<wsa:Metadata>xs:any*</wsa:Metadata>`

Die Kommunikations-Adresse muss dabei durch Ihren Typen `xs:anyURI` ein Protokoll definieren, welches eindeutig einem Kommunikations-Medium im Sinne von Kapitel 4.3.2 zuzuordnen ist.

Zutreffend ist auch die Erweiterbarkeit durch `@any` Elemente z.B. an Stelle des Adress-Elements. Hierdurch wird sichergestellt, dass die unterschiedlichen Belange der Kommunikations-Medien auch repräsentiert werden können. Als Beispiel ist hier IP zu nennen, welche in erster Näherung durch eine Protokollangabe in dem `<wsa:Address>`-Feld in der Form `http://127.0.0.1` zwar zulässig ist. Aber sie ist in dem Beispiel nur deswegen eindeutig, weil implizit von einer TCP/IP Kommunikation ausgegangen wird. Es kann also, falls das Protokoll `http:` per UDP verwendet werden soll, eine entsprechende Annotation an der Adresse vorgenommen werden.

Wichtig ist, dass dieses nur eine Möglichkeit ist, eine allgemeine Adressierung darzustellen. Es wird auch klar, dass in anderen Problemfeldern äquivalente Probleme auf ähnliche Weisen gelöst wurden.

4.3.2 Kommunikations-Medien

Wie im Kapitel 2.3 beschrieben, existiert auf Ebene der Kommunikations-Medien ein Bus- oder Netzwerk-System, welches sich ausschließlich um den Transport der Daten kümmern sollte.

Hierfür ist auf einem Gerät neben den eigentlich zu transportierenden Daten eine Kommunikations-Adresse für die Schnittstelle auf dem Gerät befindlicher Komponenten wichtig.

Eine solch minimale Schnittstelle des Kommunikations-Mediums nach oben kann durch ggf. nötige Parameter zur Echtzeit-Behandlung oder zum Fehlerverhalten **optional** ergänzt werden. Diese Ergänzungen können durch die aufbauenden Schichten folglich genutzt werden, wenn sie entsprechende Anforderungen haben.

Technische Details - wie beispielsweise ob die Kommunikation selber Verbindungs-orientiert im Sinne von TCP realisiert wird oder nicht, werden von dieser Ebene intern gehandhabt[4].

Eine Fehlerschnittstelle stellt einen Rückkanal bereit. Dabei kann das Medium eine fehlgeschlagene Kommunikation den aufbauenden Komponenten mitteilen.
Nach NE139 [20] sollten die aufbauenden Systeme jedoch auch ohne diesen Rückkanal auskommen, d.h. nur wenn es für die Anwendung besonders wichtig ist, ist eine Quittierung des Empfangs zu realisieren.

4.3.3 Nachrichten-basierte Kommunikation

Bei dieser Ebene handelt es sich eher um eine *imaginäre* Ebene. Im Endeffekt wird sie von allen Bus- und Netzwerksystemen, die in der Automatisierungstechnik eine Rolle spielen, umgesetzt.

> **Definition:** *Eine **Nachricht** ist ein Datensatz, welcher zu einem Zeitpunkt von einem Sender erzeugt, ggf. durch ein Kommunikations-Medium, versendet wird, um von einem Empfänger verarbeitet zu werden. Als Sender und Empfänger sind in der Modell-Welt Komponenten zu verstehen.*

Weitergehende Vereinbarungen - beispielsweise eine Zuordnung von Anfrage-Nachricht und Antwort-Nachricht eines Dienstes - erfolgt aufbauend auf Anwendungsebene. Dabei werden grundsätzlich die Nachrichten unabhängig voneinander versandt. Dieses begründet ein asynchrones Verhalten: Es werden sowohl auf Sender-, wie auch auf Empfänger-Seite, keine Ressourcen blockiert. Die Systeme laufen ungehindert weiter, bis die eintreffende Antwort verarbeitet wird.

Hintergrund:
Wie in der NE139 beschrieben, wird jegliche Kommunikation als Nachricht angesehen. Auch OPC-UA versendet Dienstaufrufe asynchron als Nachricht.

[4] In der NE139 wird eine Kommunikation immer basierend auf einer Verbindung gesehen. Allerdings handelt es sich dabei nicht zwangsweise um eine technische Realisierung einer Verbindung im Sinne von Auf-/Abbau einer TCP-Verbindung, sondern eher um eine „Kennen"-Verbindung. Die miteinander kommunizierenden Teilnehmer kennen also gegenseitig ihre Adressierung.

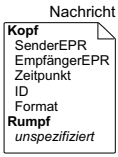


Abbildung 4.13: Datenfelder einer Nachricht

Prinzipieller Aufbau einer Nachricht Um eine Vorstellung davon zu bekommen, was eine Nachricht ist, wird hier eine Beschreibung geliefert. Nicht alle Daten müssen dabei explizit übermittelt werden - vielfach kann beispielsweise durch das Kommunikations-Medium der Absender einer Nachricht bereitgestellt werden, sodass der Transport diese Daten nicht explizit zusätzlich notwendig ist.

Eine Nachricht besteht im Wesentlichen aus zwei Teilen, wie auch in Abbildung 4.13 dargestellt: Einem Kopf und einem Rumpf.

Der Rumpf einer Nachricht enthält die eigentlichen Nutzdaten. Die Kommunikation überträgt diesen Teil der Nachricht *unverändert*. Der Empfänger einer Nachricht spezifiziert das Format; der Sender muss also dieses Format der Nachricht kennen und berücksichtigen.

Im Kopf der Nachricht befinden sich folgende Felder:

Sender: EPR Eine EndPointReference auf die Absender-Komponente und damit Ersteller einer Nachricht.

Empfänger: EPR Eine EndPointReference auf die Empfänger-Komponente einer Nachricht.

Zeitstempel: TIME Zeitstempel des Erstellens. Hierdurch kann der Empfänger Nachrichten anhand des Eingangs oder des Erstellens abarbeiten.

ID: int Eine Nummerierung der Nachrichten durch den Sender. Auf diese Weise kann der Empfänger auf eine potenzielle Antwort Bezug nehmen. Der Sender muss seine IDs der aktuell verschickten Nachrichten zum einen eindeutig halten und zum anderen auch verwalten, sodass er einen Bezug zwischen gesendeter Nachricht und einer empfangenen Antwortnachricht herstellen kann.

Format: int Das Format kodiert die Struktur des Rumpfes auf Anwendungsebene. Die Empfänger-Komponente kann anhand des Formates also erkennen, in welcher Struktur der Rumpf abgelegt ist. Eine Nutzung der Information auf Transportebene wird nicht vorgenommen.

Hintergrund:
Die Teilung von Daten, die zum Transport relevant sind (Kopf), vom für die kommunizierenden Partner relevanten Daten ist der RFC 5322 “Internet Message Format„ [18] nachempfunden. Hier hat sich diese strikte Separation bewährt.

Struktureller Ablauf Eine Nachricht wird von der versendenden Komponente erzeugt. Dieses kann als Instanziierung eines Objektes „Nachricht“ im Objekt-orientierten Instanzsystem erfolgen oder als „struct“ in einer 61131-3-Sprache. Folgend wird die Parametrisierung vorgenommen, bevor die Nachricht der Kommunikations-Komponente des sendenden Gerätes zum Versand übergeben wird. Nach einer Serialisierung der Nachricht erledigt diese durch Nutzung der Kommunikation mit der Ressourcenverwaltung den Transport zum Zielgerät. Hier ist die lokal vorhandene Kommunikations-Komponente während des Empfangs in der Lage, die Existenz der Empfänger-Kompo-nente zu überprüfen und einen Empfang der Nachricht ggf. abzulehnen. Der Kommunikationskanal wird also lediglich kurzfristig gebraucht und kann je nach Transportmedium aufrecht erhalten oder abgebaut werden.

Auf diese Weise bekommt die sendende Komponente mit, dass sowohl das Empfänger-Gerät, wie auch die Empfänger-Komponente existieren. Die geforderten Eigenschaften der transparenten Kommunikation sowie der Empfangsgarantie sind damit gegeben. Wichtig ist dabei zu bedenken, dass der Empfang einer Nachricht keine Garantie über die Verarbeitung oder Beantwortung nach sich zieht. Die Verantwortung und ggf. erforderliche Garantien hierfür sind durch eine Qualitätsbeschreibung durch den Anwenderdienst abbildbar, jedoch in dem beschriebenen Nachrichten-basierten Paradigma explizit nicht vorgesehen.

Durch die pro Gerät zentrale Rolle der Kommunikations-Komponente werden mehrere Vorteile erschlossen. Zum einen können die versendeten und empfangenen Nachrichten selber zur Überwachung der Geräte genutzt werden, nach dem Gedanken „Wenn ein Gerät eine Nachricht gesendet oder empfangen hat, wird es aktuell erreichbar und aktiv sein“. Zum anderen ist die Nachrichten-basierte Kommunikation gerade dazu da, eine asynchrone Kommunikation zu ermöglichen. Also ist eine bewusste Toleranz von Verzögerungen auf Anwendungsseite prinzipiell akzeptabel. Andernfalls würde auf eine synchrone Kommunikationsform zurückgegriffen werden. Durch die Kommunikations-Komponente kann eine zentrale Komponente den maximalen Kommunikationsbedarf des Gerätes gegenüber dem Kommunikations-Medium reglementieren, sodass die einzelnen Komponenten hiervon genauso entlastet werden, wie von der Verwaltung der Kommunikation selber.

Status als lokale Variable Das Kommunikations-Medium bietet meistens die Möglichkeit Fehler zu erkennen und zu beheben bzw. bietet einen Rückkanal, wenn das Empfangsgerät nicht erreichbar ist. Diese Information kann in eine Status-Variable des Nachrichten-Kopfes auf lokaler Ebene abgebildet werden.

Hierdurch kann insgesamt eine sendende Komponente mitbekommen, wenn entweder das Zielgerät nicht erreichbar ist oder die dort adressierte Komponente nicht existent ist.

Die Verarbeitung der empfangenen Nachricht sowie auch der Fehler-Status der zu sendenden Nachricht ist dabei auf Anwendungsebene zu realisieren, da unterschiedliche Reaktionen möglich und sinnvoll sind.

Aus Sicht der Instanzumgebung und des Kommunikationssystems sind insbesondere folgende Fehlerquellen denkbar

- Zugriffsverweigerung: Die empfangene Komponente akzeptiert von dem Absender diese / alle Nachrichten nicht.
- Ressourcen-Problem: Der Empfänger kann die Anfrage derzeit nicht verarbeiten, da keine Ressourcen bereitstehen.
- Typ-Verletzung („Mismatch“): Der Rumpf entspricht nicht dem vom Empfänger spezifizierten Format

Diese Gruppen von Problemen bieten einen Ansatz-Punkt für eine allgemein verständliche Formulierung von Fehler-Antworten.

Aufbauend auf dieser Nachrichten-basierten „Denkweise“ lassen sich unterschiedliche Typen für die Kommunikation beschreiben.
Dienste, die ihre Schnittstellen mittels dieser Typen anbieten, können aufbauend auf den Typen beschrieben werden. Dabei ist festzustellen, dass für einige Typen bereits gemeinsame Konzepte vorhanden und in Nutzung sind. Für andere Typen scheint die Verwendung eher marginal zu sein.

Weiterhin ist zu beachten, dass auch diese Typen ggf. durch weitere Typen erweitert werden können. Beispielhaft wird hier der *Typ 4: Intent-basierte Kommunikation* beschrieben, der eine weitergehende Entkopplung der kommunizierenden Software-Komponenten beschreibt, in der Automatisierungstechnik aber noch nicht existent ist.

Unterschiedliche Repräsentationen In dieser Arbeit wird bewusst auf die Abbildung der Nachrichten in z.B. XML oder eine andere Sprache verzichtet. Eine solche Repräsentation zum Transport kann zum einen beliebig gewählt werden. Zum anderen kann sie aber auch durch das Kommunikations-Medium vorgegeben sein.

Darstellung von Nachrichten-Typ-Definitionen Als Darstellung für Nachrichten-Typen und Formate wird folgende Notation vorgeschlagen:

TYP - Nachricht	*NAME*	PARAMETER
BESCHREIBUNG		

Der TYP bezeichnet die (im Folgenden definierten) Typen von Kommunikations-Nachrichten. Es werden also ggf. mehrere Nachrichten (Anfrage und Antwort) in einer Darstellung beschrieben.

Der NAME ist eine Bezeichnung des Nachrichten-Typs. Er dient lediglich dem besseren Verständnis.

Die PARAMETER bezeichnen die Schnittstelle selbst, d.h. die Daten, die dort übermittelt werden müssen. Die Darstellung ist vom Kommunikations-Typ abhängig.

Eine BESCHREIBUNG gibt eine Erklärung vom Inhalt der Nachricht.

4.3.4 Typ 1: singuläre Kommunikation

Die einfachste Form der Kommunikation sind Meldungen vom Sender zum Empfänger, ohne eine Antwort. Diese Form ist auch in der NE139 beschrieben, wird bisher jedoch im AT-Umfeld nicht genutzt. Wichtig ist, dass die Meldungen hier nicht mit Leitsystem-technischen Meldungen im Sinne von Melden und Alarmieren verwechselt werden. An dieser Stelle sind Meldungen eine einfache Einweg-Nachrichten-Übermittlung.

singulär - Nachricht	*NAME*	PARAMETER
Diese Mitteilungen werden ohne Antwort dem Empfänger zugestellt. Fehler auf Ebene des Kommunikations-Mediums werden, soweit feststellbar, mitgeteilt.		

Die PARAMETER bezeichnen die Daten, die im Körper der Nachricht übertragen werden.

4.3.5 Typ 2: Aufruf/Antwort Kommunikation

Diese Form der Kommunikation ist die typische Form für Dienst-orientierte Architekturen, sowie auch nahezu alle aktuellen Kommunikationsformen im AT-Umfeld.

Auf eine Aufruf-Nachricht folgt eine festgelegte Antwort-Nachricht, d.h. es gibt immer Paare von Aufruf-Antwort-Kommunikation. Die Antwort erfolgt zeitverzögert. Fehlerfälle werden durch entsprechende Antworten kommuniziert. Erfolgt keine Antwort, ist dieses ein Fehlverhalten des Empfängers.

Die vom Prinzip in NE 139 und konkret in NE 141 beschriebenen „SystemBasisDienste“ (SBD) sind Beispiele für diesen Typ. Sie definieren unter anderem den Zugriff, die Erkundung und die Manipulation von Objekt-orientierten Strukturen.

A/**A** - Nachricht	*NAME*	EINGABE $\rightarrow$ AUSGABE
Hier wird ein Tupel von Aufruf- und Antwort-Nachrichten beschrieben.		

Die EINGABE bezeichnet dabei die PARAMETER der Aufruf-Nachricht; die AUSGABE entsprechend die PARAMETER der Antwort-Nachricht.

4.3.6 Typ 3: Subskription/Benachrichtigungs-Kommunikation

Basierend auf der Nachrichten-basierten Kommunikation ist auch eine Subskription/Benachrichtigungs-Kommunikation zu sehen. Dabei meldet sich ein Kommunikationspartner bei einem Informationsanbieter an. Er abonniert also Neuigkeiten bei dem Informationsanbieter. Dieser antwortet mit den Benachrichtigungs-Nachrichten. Je nach verwendeter Sprache können bei der Subskription-Nachricht auch Qualifizierungen angegeben werden. Es ist beispielsweise denkbar, dass ein *delta*-Faktor angegeben wird, der eine Minimal-Abweichung eines Messwertes definiert, bei der eine Benachrichtigung erfolgen soll.

Ein typischer Anwendungsfall ist (gerade für OPC-UA) die Visualisierung: Hier werden per „Subscription“ Prozesswerte dem Bediener einer Anlage aktuell angezeigt, selbst wenn diese sich nur selten ändern.

Genau genommen handelt es sich bei dieser Kommunikation um einen zusätzlichen Typen, der als optional bezeichnet werden kann. Durch eine entsprechend hoch-frequen-tierte Aufruf/Antwort Kommunikation kann eine Subskription/Benachrichtigungs-Kom-munikation ersetzt werden. Allerdings überwiegen die Vorteile dieses Kommunikations-Typen, wenn bei seltenen Ereignissen der Kommunikations-Partner schnell in Kenntnis gesetzt werden muss.

Hintergrund:
Bekannteste Vertreter dieser Kommunikationsform für Endanwender sind die RSS-Feeds von Webseiten. Sie bieten in einem „standardisierten“ Format [29] Informationen über Änderungen auf einer Webseite.
Bei den RSS-Feeds wird auch gleichzeitig dargestellt, wie unterschiedlich dieser Kommunikationstyp realisiert werden kann: Während es für den Endanwender aussieht, als ob sein RSS-Programm von den Webseiten-Anbietern über die Änderungen informiert wird, holt das RSS-Klienten-Programm technisch gesehen zyklisch die RSS-Feeds ab und überprüft so die Aktualität.
Technisch handelt es sich also um ein klassisches „Polling“ (zyklisches Abholen), während es sich dem Endanwender gegenüber wie ein Subskription/Benachrichtigungs-Kommunikationstyp verhält.

Aufgrund der unterschiedlichen Realisierungsmöglichkeiten der internen Verwaltung, sowie der Verbreitung der Benachrichtigungs-Nachrichten, wird hier auf eine konkrete Beschreibung verzichtet. OPC-UA spezifiziert eine Reihen von konkreten Subskription/Benachrichtigungs-Schnittstellen in den jeweiligen Teilen. Die NE139 nennt auf diesem Typ aufbauende Dienste *Subskriptionsdienste*.

Ganz allgemein hat die OMG in ihre „Enterprise Collaboration Architecture“ [23] eine umfassende Modellierung eines Subskription/Benachrichtigungs-Systems beschrieben.

Nachrichten bei Subskription/Benachrichtigungs-Kommunikation An dieser Stelle sollen die prinzipiellen Informationsinhalte der Subskription- sowie Benachrichtigungs-Nachrichten beschrieben werden.

singulär - Nachricht	*SUBSCRIPTION*	EPR, SUB-BESCHREIBUNG
Eine Aufforderung an den Empfänger, Benachrichtigungen zu versenden, wenn Bedingungen zutreffen.		

Die Subskription-Nachricht besteht aus einem Tupel: Zum einen muss eine EPR für den Empfang von Benachrichtigungs-Nachrichten gegeben sein. Zum anderen muss eine Beschreibung der Informationen existieren. Als Beispiel ist bereits ein Prozesswert und ein zugehöriger delta-

Faktor erklärt worden. Diese Beschreibungssprache ist Anwendungs-spezifisch. Die anbietende Komponente muss also eine entsprechende Verwaltung erledigen.

Es ergibt sich automatisch die Möglichkeit, diese Subskription zu beenden:

singulär - Nachricht	*DE-SUBSCRIPTION*	EPR, SUB-BESCHREIBUNG
Eine Aufforderung an den Empfänger, keine weiteren Benachrichtigungen zu versenden		

Es bietet sich gerade bei den SUBSCRIPTION wie auch DE-SUBSCRIPTION Nachrichten an, dass sie als Typ 2 realisiert werden, sodass der Aufbau und Abbau der Benachrichtigungen quittiert wird. Hierdurch erhält der Empfänger der Benachrichtigungen auch einen Zeitpunkt, ab wann er mit Benachrichtigungen rechnen kann.

Falls die per SUB-BESCHREIBUNG definierten Kriterien zutreffen, wird EPR durch eine Nachricht informiert.

singulär - Nachricht	*NOTIFICATION*	NOTIFICATION-BESCHREIBUNG
Information der EPR, dass ein Ereignis eingetreten ist.		

Die NOTIFICATION-BESCHREIBUNG erfolgt wiederum auf Anwendungsebene und beschreibt das Ereignis, welches aufgetreten ist. Der Anwendungs-Entwickler muss hier entscheiden, welche Details des Ereignisses in der NOTIFICATION enthalten sind, also welche Informationen eine Anwendung sinnvollerweise bekommt.

Vereinfachte Darstellung Auf das Wesentliche konzentriert, kann eine Subskription/Benachrichtigungs-Kommunikation wie folgt dargestellt werden:

S/N - Nachricht	*NAME*	SUB-BESCHREIBUNG $\rightsquigarrow$ NOT-BESCHREIBUNG
Eine Aufforderung an den Empfänger Benachrichtigungen zu versenden, wenn Bedingungen zutreffen.		

4.3.7 Typ 4: Indirekte Kommunikation per *Intents*

An dieser Stelle wird ein weiterer, neuer Kommunikations-Typ vorgeschlagen: Die Intent-basierte Kommunikation.

Um Komponenten weitgehender zu entkoppeln, sind Verwaltungskomponenten notwendig. Diese „vermitteln“ Kommunikation zwischen Komponenten, die sich direkt auf Kommunikationsebene nicht kennen und auch nicht miteinander kommunizieren.

Es handelt sich bei der Intent-basierten Kommunikation im Wesentlichen um eine spezielle Form der Subskription/Benachrichtigungs-Kommunikation.
Dafür wird für die Benachrichtigung eine spezielle $NOTIFICATION$-Nachricht definiert, die eine besonders einfache Form hat: Lediglich ein Schlüssel-Wert-Paar, wobei beide Teile aus einer einfachen Zeichenkette bestehen.

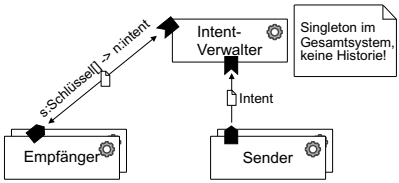


Abbildung 4.14: Konzept der Intent-basierten Kommunikation als Erweiterung der Subskription/Benachrichtigungs-Kommunikation

Intent - Nachricht	*INTENT-SCHLUESSEL*	WERT
Eine Information einer Anwendung über das Auftreten eines Ereignisses.		

Diese Nachrichten werden *Intents* genannt und von Informationsanbietern an eine zentrale Komponente gesendet. Komponenten lassen sich durch diese zentrale Stelle informieren, wenn sie mit für sie interessanten Schlüsseln eintreffen.
Im Sinne des Subskription/Benachrichtigungs-Prinzips handelt es sich also um das Subskribieren auf bestimmte Schlüssel an einer systemweit einheitlichen Intent-Kommunikations-Komponente.

Der Schlüssel definiert das Ereignis selber. Wenn eine Anwendung diesen Schlüssel kennt, weiss sie auch um den WERT bzw. mit den enthaltenen Informationen umzugehen.

Durch die einfache Subskription/Benachrichtigungs-Kommunikation können Anwendungen Intents empfangen:

S/N - Nachricht	*INTENTSUBSKRIPTION*	EPR, INTENT-SCHLÜSSEL $\rightsquigarrow$ INTENT
Eine Aufforderung an den Intent-Verwalter, Benachrichtigungen über eintreffende Intents zu versenden		

Der INTENT-SCHLÜSSEL ist dabei genau der SCHLÜSSEL, den die gesendeten Intent-Nachrichten auch haben. INTENT beinhaltet als Benachrichtigung sowohl den Schlüssel, wie auch den Wert. Da eine EPR ggf. für unterschiedliche SCHLÜSSEL angemeldet ist, wird SCHLÜSSEL hier analog zum Typen einer Nachricht verwendet: Er beschreibt den Inhalt des INTENT.

Ablauf Eine Anwendung kann eine Absicht (engl. Intent) äußern, indem sie eine Nachricht an eine systemweit zentrale Komponenten, den *Intent-Verwalter*, versendet. Dieser leitet die Nachricht entsprechend an Empfänger weiter. Dafür melden sich Empfänger mit für sie interessanten Schlüsseln an dem Intent-Verwalter vorab an, sodass aus ihrer Sicht eine Subskription/Benachrichtigungs-Kommunikation entsteht. Abbildung 4.14 verdeutlicht den Vorgang, wobei auch klar wird, dass die Empfänger von Intents als Subskription/Benachrichtigungs-Kommunikation realisiert werden, das Senden jedoch auf singulären Nachrichten basiert.

Es ist dabei sinnvoll, dass die Intent-Nachrichten sofort zugestellt werden und es keine Historie und dadurch keine nachträglichen Reaktionen gibt. Hierdurch wird klar, dass es sich um eine aktuelle Absicht einer Anwendung für die zu dem Zeitpunkt „interessierten“ Empfänger handelt.

Hintergrund:
Der Begriff *Intent* stammt insbesondere aus dem Android-Betriebssystem. Die Subskription einer Anwendung wird hier durch ihre META-INF Datei „erledigt“, d.h. schon die Basis-Beschreibung einer Android-App hat die Möglichkeit zu spezifizieren, bei welchen Intent-Nachrichten eine App benachrichtigt werden möchte. Es erfolgt keine standardisierte Festlegung der Schlüssel und auch keinerlei Beschreibung der Nachrichteninhalte.

Das Prinzip scheint insbesondere für die Benachrichtigung bei Modell-Änderungen sinnvoll, wenn - wie eingangs beschrieben - viele Komponenten auf Änderungen in der Modell-Instanz reagieren sollten. Kapitel 4.5.3 beschreibt deswegen die Nutzung dieser Schnittstelle als ein Anwendungsfall für Modelländerungen.

Wer definiert die Intent-Schlüssel? Es erscheint sinnvoll, hier keine Hierarchie oder festgelegte Schlüssel-Liste zu pflegen. Das Konzept hat sich bei modernen Betriebssystemen genau *ohne* eine vorab Festlegung bewiesen: So können Komponenten-Entwickler sich bilateral auf einen Schlüssel einigen. Dieser kann durch den Intent-Verwalter eine Kommunikation zwischen den Komponenten darstellen. Für den Intent-Verwalter ist jeder Schlüssel lediglich ein String, der mit zuvor empfangenen Subskriptionen verglichen wird.

Durch dieses Prinzip wird eine „Einfachst-Kommunikation“ realisiert. Die sendenden sowie empfangenden Komponenten werden komplett von der Adressaten-Verwaltung freigestellt. Dieses geht auf Kosten der Nachvollziehbarkeit, da eine sendende Instanz nicht sicher sein kann, dass es überhaupt einen Empfänger gibt.

Das System bietet sich also für eine Art gezielte Broadcast-Nachrichten-System an.

4.3.8 Lokale Kommunikation

Auch ein Aufruf einer Methode, die durch eine API bereitgestellt wird, kann als Kommunikation zwischen Komponenten betrachtet werden. Im Sinne des allgemeinen Verständnisses ist dann beispielsweise das Objekt oder die Objekt-Referenz die Definition des Empfängers.

Zwei Aspekte sind grundsätzlich einfacher bei der lokalen Kommunikation, als bei den zuvor beschriebenen.

1. Kein Erreichbarkeits-Problem - Kommunikation muss so konzeptioniert sein, dass ein Ausfall des jeweiligen Gegenübers berücksichtigt wird. Bei einer lokalen Kommunikation ist dies nicht der Fall.

2. Enge Bindung: Durch eine lokale Schnittstelle sind die beiden Kommunikationspartner technologisch ähnlich. Beispielsweise sind Kodierungsvorschriften durch eine gemeinsame Plattform gleich, oder es kann sogar auf einen gemeinsamen Speicherbereich zugegriffen werden, sodass nur Zeiger auf Speicher übergeben werden und die Daten selber nicht transportiert werden müssen.

Es erscheint deswegen sinnvoll, lokale Kommunikation getrennt zu betrachten:

Hier wird aber auch ersichtlich, dass eine lokale Kommunikation in aktuellen Systemen durchaus auch als entfernte Kommunikation realisiert werden kann: Wenn eine Kommunikation zwischen autarken Prozessen auf einem Gerät stattfindet, können ähnliche Probleme auftreten, wie bei einer entfernten Kommunikation der anderen Typen.

Das heißt auch, dass als lokale Kommunikation in diesem Sinne nur die Kommunikation der Komponenten innerhalb einer Umgebung verstanden wird. Sobald beispielsweise unabhängige Programme gestartet werden, handelt es sich um eine Interprozesskommunikation. Für eine solche können die Typen 1-4 verwendet werden, da wiederum die (A)Synchronität beachtet werden muss.

4.4 Dienst-Modell: aktive, dynamische Komponenten

Bereits in der Motivation wurde verdeutlicht, dass der Fokus dieser Arbeit nicht auf der Modellierung der aktiven Komponenten liegt. Jedoch ist die klassische Dienst-Orientierung mit ihrer Schnittstellen-orientierten Kommunikation eine Herausforderung, die angegangen werden sollte. Schließlich müssen auch die für die Instanzumgebung vorgesehenen aktiven Komponenten kommunizieren.

Eine Anwendung wird als eine Gruppe von Diensten verstanden. Ein Dienst hat im Wesentlichen aktive Komponenten. Damit sind Dienste also zum einen eine Strukturierungs-Möglichkeit, zum anderen aber auch eine technologische Realisierung von verteilten Anwendungen.

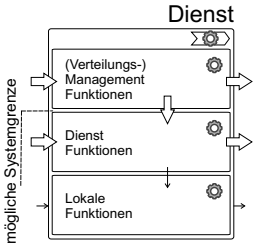


Abbildung 4.15: Aufteilung eines Dienstes in Funktionen

Zum einen kann ein Dienst lokal wirken, d.h. lokale Funktionen eines unterliegenden (Betriebs-)Systems aufrufen oder Werte von lokal angeschlossenen Sensoren/Aktoren ermitteln. Diese Schicht ist die unterste Schicht der *lokalen Kommunikation*.

Ein Dienst nutzt dabei die in Kapitel 2 beschriebenen Variablen. Dabei kann zwischen zwei Variablenarten unterschieden werden. Zum einen existieren *statische Variablen*, die beispielsweise eine Parametrierung beinhalten. Sie sind vorgegeben und können bei Verlust **nicht** neu berechnet werden. Zum anderen existieren *dynamische Variablen*, die durch andere Dienste oder den Dienst selber berechnet / bereitgestellt werden. Beispielsweise werden Prozesswerte, die sich permanent ändern, als solche angesehen.

Damit bilden statische Variablen zusammen mit den internen Variablen die *Konfiguration eines Dienstes*, während die dynamischen Variablen das Verhalten beeinflussen. Dieses kann als Eigenschaften der Dienste modelliert werden. Zur Sicherung (und damit auch zur Verlagerung auf ein anderes Gerät) eines Dienstes ist es lediglich nötig die statischen Variablen sowie den internen Zustand (Stichwort: Serialisierung) zu sichern.

Aufbauend sind die eigentlichen Dienst-Funktionen zu sehen. Hierunter werden die Vorgänge verstanden, die den Dienst selber ausmachen. Dazu gehören zum einen nötige Berechnungen, zum anderen auch die Anwendungs-orientierte Kommunikation - ggf. basierend auf der lokalen Kommunikation.

Getrennt von den eigentlichen Funktionen der Anwendung sollten die Management-Funktionen aufgefasst werden. Hierbei handelt es sich um Funktionen zur Verwaltung der Dienste. Beispielsweise gehört dazu das Deployment, also das Starten und Stoppen eines Dienstes. Aber auch ein Algorithmus zur Bestimmung der Verteilung (und damit der Orchestrierung der Dienste) würde in diese Kategorie fallen.
Aus diesem Grund kann die Management-Funktion eines Dienstes auch durchaus auf einem anderen System ausgeführt werden, wenn eine entsprechende Bindung von Management-Funktion zu den zu verwaltenden Dienst-Funktionen bzw. Komponenten realisiert wird.

Die unterschiedlichen Funktions-Arten sind in Abbildung 4.15 dargestellt.

Hintergrund:
Die Trennung von Management-Funktionen von den eigentlichen Funktionen ist angelehnt an das Konzept von IBM im Bereich des Autonomic Computing [Hor01].

Abbildung 4.16 stellt die Kommunikations-Möglichkeiten eines Dienstes schematisch dar.

Abbildung der Dienste in die Repräsentations-Welt Dienste sind aktive Komponenten und erhalten nach Kapitel 4.2.2 also eine eigenen Eintrag als Dienst-Repräsentation.
Hierdurch werden sie zum einen auffindbar im Gesamtsystem, zum andern können die aktuellen Beziehungen zu anderen Diensten durch die Management-Funktion auch im Modell durch

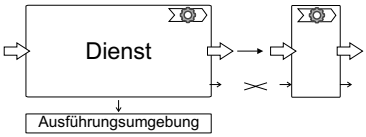


Abbildung 4.16: Schematische Darstellung der Dienst-Kommunikation

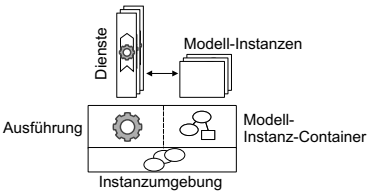


Abbildung 4.17: Aufbau einer Instanzumgebung für Modelle **und** Dienste

entsprechende Zugriffs-Relationen repräsentiert werden. Hierdurch wird es z.B. auch möglich, Umschaltungen im Modell abzulegen (z.B. als Vorbereitung auf Ausfälle, vgl. Kapitel 6.4) und überwachen zu lassen (vgl. Kapitel 4.6.2.1).

4.5 Die modellgetriebene Instanzumgebung

Nachdem sowohl die Komponenten-Repräsentationen, wie auch das Konzept der aktiven Komponenten beschrieben sind, wird folgend die *verteilte, modellgetriebene Instanzumgebung* vorgestellt. Die Verteilung der Modelle auf unterschiedliche Geräte und die Verbindung von unabhängigen Modellen werden im späteren Verlauf ergänzt.

Es handelt sich bei einer Instanzumgebung um weiterentwickelte, existierende Konzepte, wie sie in Kapitel 2.4 beschrieben sind. Ein Instanz-Container wird als Programm bereitgestellt, welches in der Lage ist, alle vom Meta-Modell abgeleiteten Modelle darzustellen. Dafür werden Klassen und Relationen zur Laufzeit aus Bibliotheken instanziiert. Diese Instanziierung und Verknüpfung erfolgt dabei entweder intern durch Abläufe oder durch externe Befehle zur Manipulation der Objektstruktur.

Einige wichtige Aspekte sollten bei der Entwicklung der modellgetriebenen Instanzumgebung beachtet werden:

Selbst-Beschreibung Klassen in den Komponenten-Bibliotheken sollten erkundbar sein, also selber durch Objekt-Repräsentationen dargestellt werden.

Ausführung Im Idealfall wird - wie in Abbildung 4.17 verdeutlicht - eine Instanzumgebung sowohl für die Modell-Instanzen, wie auch für die aktiven Komponenten (Dienste, Anwendungen)

bereitgestellt. Durch die zuvor beschriebenen Repräsentationen dieser Komponenten ist es aber möglich aktive Komponenten *neben* der eigentlichen Instanzumgebung auszuführen.

Dynamische Ergänzung der Fähigkeiten Für eine Realisierung ist es wichtig, dass die entsprechenden Komponenten-Bibliotheken nachgeladen werden können. Dieses ist insbesondere dann wichtig, wenn eine dynamische Systemstruktur aktiv genutzt wird - z.B. auch zur Laufzeit neue Funktionen in einem System etabliert werden sollen.

Kommunikation Sie sollte ebenso wie Objekte im Modell-Raum repräsentiert werden, d.h. offene Kommunikations-Kanäle sollten im Modell-Raum dargestellt werden. Somit wird die Kommunikation erkundbar.

4.5.1 Vom Modell zum Instanz-Modell

Wie im Kapitel 2.2 beschrieben, werden heutzutage vielfach Modelle als Abbildung von bestimmten Eigenschaften der Anlagen entworfen und teilweise auch standardisiert. Wie bereits geschildert, gibt es dabei Modelle, die sehr formal beschrieben sind, und andere, die weniger formal beschrieben sind.

Damit ein Modell hier einsetzbar ist, muss es in eine Struktur aus Klassenbeschreibungen mit Eigenschaften sowie Relationen zwischen den Objekten der Klassen überführbar sein (Klassen-Modell). Instanzen einer solchen Modell-Beschreibung werden im Kontext der Instanzumgebung als *Instanz-Modell* bezeichnet.

Der Grad der Formalität ist dabei untergeordnet: Ist eine Struktur beschrieben bzw. das Modell in eine solche überführbar, ist sie auch im Sinne der Instanzumgebung nutzbar. Selbstverständlich steigt bei einer formaleren, genaueren Abbildung der Informationen (vgl. Kapitel 2.1) auch der Nutzen.

Systematisches Vorgehen Es stellt sich die Frage, wie ein Modell für die Umgebung nutzbar gemacht wird. Wie zuvor beschrieben, werden viele Modelle heutzutage als XML spezifiziert, d.h. es gibt eine XML Grammatik, die die Sprache syntaktisch beschreibt und eine Spezifikation, die das Verständnis der Sprache beisteuert.

Dazu gibt es eine Vorgehensbeschreibung, wie eine Abbildung der Modelle in XML vorgenommen werden kann. Welche Probleme dabei berücksichtigt werden müssen ist in der VDI Richtlinie 3690 [1] beschrieben: Das Ergebnis der Vorgehensbeschreibung ist im Erfolgsfall eine XML Schema Beschreibung, die eine XML Struktur beschreibt.

Die Nutzung einer solchen XML Struktur im Instanzsystem ist einfach: Beispielsweise sind DOM-Parser in der Lage, eine Objekt-Struktur aus einem XML Dokument bereitzustellen. Vollautomatisch (z.B. durch einen gängigen XML-Parser) werden alle XML-Elemente als Objekte abgebildet. Diese sind untereinander analog zum XML-Dokument verschachtelt. Es existiert

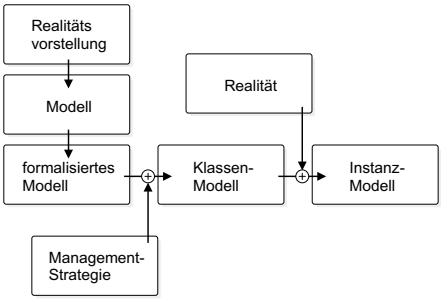


Abbildung 4.18: Entwicklungs-Phasen eines Modells für die Instanzumgebung

also eine Relation *contains*; weitere Relationen sind ebenfalls möglich. Dieses Verfahren zeigt, dass zumindest alle Modelle nach der VDI-Richtlinie 3690 *prinzipiell* als Laufzeit-Modelle nutzbar sind.

In Anlehnung an die VDI Richtlinie 3690 ist in Abbildung 4.18 dargestellt, wie ein Modell für die hier verwendete Instanzumgebung entstehen kann. Nachdem die Übergänge von der Realität zum Modell und auch zum formalisierten Modell in der Richtlinie abgehandelt sind, erfolgt ein semi-automatisierter Schritt zum Klassen-Modell. Hierbei werden insbesondere Strategien für die Verteilung zum Modell hinzugefügt - dieses beinhaltet das Konzept des Modell-Masters, welches im Folgenden beschrieben wird. Das Klassen-Modell wird in der Instanzumgebung etabliert. Damit ist dieses Modell nachher zugreifbar und erkundbar, wie man es auch aus ACPLT (normalerweise als „Library“ bezeichnet) oder OPC-UA (Bezeichnung: „Namespace“) gewohnt ist. Werden die enthaltenen Klassen für die Nutzung in Bezug auf die Realität instanziiert, kann von einem *Instanz-Modell* gesprochen werden.

4.5.1.1 Modell-Master

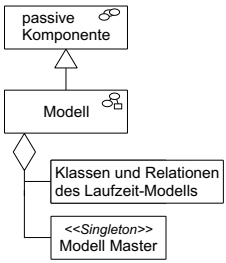


Abbildung 4.19: Elemente eines Modells für die flexible Instanzumgebung

Eine wichtige Rolle für ein Instanz-Modell spielt der Modell-Master, dargestellt in Abbildung 4.19. Diese Komponente, die in einer Instanzumgebung einmalig pro Modell instanziiert wird

(als Entwicklungspattern „Singleton“ nach [Bal08]), stellt den Einstiegspunkt für die Erkundung der Instanzen des Modells dar. Die Instanz kann also mit der vorgestellten Instanz des Modells aus Kapitel 2.1.3 gleichgestellt werden.

Definition: *Jedes Laufzeit-Modell besitzt neben den Klassen und Relationen einen **Modell-Master**, der die zentrale Verwaltung der Modell-Beschreibung (Klassen) sowie auch der verteilten Instanzen übernimmt. Er existiert genau einmal pro Modell in einem (verteilten) System und stellt auch den Einstiegspunkt für die Modellerkundung von Anwendungen dar.*

4.5.2 Sprache der Modell-Änderungen

Die Manipulation von Modell-Instanzen zur Laufzeit selber ist durch wenige Grund-Befehle ausreichend beschreibbar, das heißt, es genügt eine Sprache von wenigen Kommandos, um die Modelle zu erstellen und zu verändern.

An dieser Stelle werden die Modell-Änderungsfunktionen in aller Kürze beschrieben, da sie entsprechend der NE 139 und 141 definiert sind. Sie entsprechen also dem Aufruf/Antwort (Typ 2) der vorherigen Typisierung der Kommunikation.

Anlegen von Objekt Legt ein Objekt von gegebenen Typen an.

Entfernen von Objekt Entfernt ein Objekt und alle in Verbindung stehenden Relationen.

Anlegen von Relation Legt eine Relation von gegebenen Typen zwischen zwei Objekten an.

Entfernen von Relation Entfernt die Relation.

Setzen Wert Setzt einen Variablen-Wert.

Hole Wert Holt einen Variablen-Wert.

Kombination Atomare Ausführung einer Reihe der vorstehenden Befehle. Auch die Modell-Erkundungsfunktionen können hier verwendet werden.

Aufbauende Operationen, die beispielsweise häufige Kombinationen aus Befehlen vereinfacht ausführen, sind denkbar und könnten für eine Realisierung auch als Dienst nachgeladen werden.

Hintergrund:
Datenbanken kennen eine solche Möglichkeit mit „stored procedures“. Dieses sind Programme, die in der Datenbank selber ablaufen, wenn sie aufgerufen werden.
OPC-UA kennt dieses Konzept ebenso, indem *Methoden* spezifiziert werden können, die Schnittstellen bereitstellen.

Wichtig ist dabei, dass es zunächst keinerlei Annahmen über existierende Objekte oder Relationen gibt.
So impliziert ACPLT durch die verwendeten Pfade eine Relation „/“ („ov_contains“), die als Unter-Objekt interpretiert werden kann. Eine solche allgegenwärtige Relation kann ggf. bei der Abbildung von existierenden Modellen zu Problemen führen, da nicht alle Modelle eine Relation mit dieser Semantik verwenden oder diese auch in unterschiedlicher Weise verwenden. Aus diesem Grund verzichtet die hier beschrieben Instanzumgebung auf die Vorgabe einer solchen Basis-Relation.

Nicht zu jedem Zeitpunkt können ausschließlich valide Modelle, d.h. Modelle, deren Gültigkeit überprüft werden kann, existieren. Änderungen zur Laufzeit durch einzelne Aktionen werden immer wieder zu Inkonsistenzen führen[5].
Hiermit müssen grundsätzlich die Anwendungen umgehen können. Sie sollten den Teil des Modells soweit ignorieren, wie nötig. Implizite Annahme ist, dass nicht valide Teilmodelle gerade in Bearbeitung durch einen anderen Dienst sind.

Trotz der grundsätzlichen Möglichkeit eines inkonsistenten Zustandes können beliebige Befehle in einer ununterbrochenen (d.h. atomaren) Kombination ausgeführt werden. Dies ist insbesondere wichtig, um die zuvor definierten AKID-Eigenschaften einhalten zu können. Um diese AKID-Eigenschaften einhalten zu können, sind entsprechende Konzepte aus der Datenbank-Welt wichtig: „Semaphoren“, „Locking“ und „Rollback“-Mechanismen helfen, eine Folge von Befehlen entweder ganz oder nicht auszuführen.

Eine Anwendung kann durch die Atomität der AKID-Eigenschaften den für sie wichtigen Bereich des Modells validieren, bevor Änderungen vorgenommen werden. Hierfür wird ein Tupel aus Erkundungs-Funktion (zur Validierung) und Modell-Änderung als Kombinations-Basisoperation vorgenommen. Schlägt die Erkundungs-Funktion fehl, wird die Modell-Änderung nicht ausgeführt.

4.5.3 Änderungs-Benachrichtigungen

Um Änderungen in einer modellgetriebenen Instanzumgebung nutzen zu können, müssen Komponenten von den Änderungen erfahren und somit reagieren können.

Das einfachste Vorgehen wäre, wenn Anwendungen, die Objekte erkunden, diese mit dem vorherigen (bekannten) Zustand vergleichen und entsprechende Änderungen vornehmen würden. Dieses würde jedoch dazu führen, dass die Anwendungen den vorherigen Zustand aller interessanten Modell-Teile speichern müssten. Zusätzlich müsste dieses Vorgehen zyklisch ausgeführt werden, wodurch Performance-Engpässe entstehen könnten.

[5] Dieses ist vergleichbar mit der Programmierung - während einer Programmierung entstehen immer wieder ungültige Anweisungen - erst zu diskreten Zeitpunkten werden Übersetzungen angestoßen.

Aus diesem Grund wird eine Nachrichten-basierte Schnittstelle auf Basis der Subskription/Benachrichtigungs-Typs (Typ 3) vorgeschlagen. Eine Anwendung kann sich somit für Änderungen in den Modell-Instanzen am Instanzsystem subskribieren und bekommt eine Nachricht, wenn sich etwas für die Anwendung entscheidendes geändert hat. Durch die Analyse der Befehle und Parameter können die Empfänger-Dienste entscheiden, ob eine Reaktion erforderlich ist.

Damit stellt sich die Frage nach einer Sprache in der die „interessanten“ Teile der Objekte spezifiziert werden können, sodass möglichst wenige Benachrichtigungs-Nachrichten versandt werden müssen. An dieser Stelle sind vielfache Möglichkeiten denkbar - beispielsweise auch wieder spezifische Erweiterungen der Instanzumgebung als zusätzliche, nachladbare Funktion.

Als „Einfachst-Realisierung“ können sämtliche Befehle, die zur Manipulation der Modell-Instanzen an die Instanzumgebung abgesandt werden, an alle angemeldeten Dienste weitergeleitet werden. Wenn nur wenige Dienste Interesse an Änderungen haben, kann es auch durch so eine „Einfachst-Realisierung“ zu einem wesentlichen Geschwindigkeitsvorteil kommen.

Bei dieser einfach gehaltenen Sprache steht eine Referenz auf ein existierendes Objekt im Mittelpunkt und wird um eine Änderungsbedingung ergänzt, die Auslöser für eine Benachrichtigung ist.

Eine Subskription-Nachricht für Änderungen ist damit ein Tupel:

S/N - Nachricht	*Modell-Änderungen*	REFERENZ, VAR $\|\|$ REL $\rightarrow$ ÄNDERUNG
Subskribiert den Sender beim Empfänger für durch die Eingabe-Parameter bestimmte Änderungen in der Modell-Struktur.		

Mit Bezug auf das in $REFERENZ$ definierte Objekt werden entweder Änderungen der Variablen oder der Relationen mitgeteilt. Implizit ist immer eine Benachrichtigung enthalten, welches über das Löschen des $REFERNZ$-Objektes informiert.

Da, wie vorausgesetzt, auch die Klassen im Modell-Raum abgebildet und mit den Instanzen verbunden werden, kann auch das Erzeugen neuer Objekte überwacht werden: Eine Subskription auf das Klassenobjekt als Referenz sowie der Eigenschaft REL bewirkt, dass über das Anlegen eines Objektes benachrichtigt wird.

Eine Benachrichtigungs-Nachricht ist dabei ähnlich wie eine Subskription-Nachricht:

singulär - Nachricht	*Modell-Änderung*	$Quelle, (Value \vee Ziel)$
Information über die Modell-Änderung (vgl. Text).		

Neben dem Objekt, von dem die Änderung ausgeht (Referenz $Quelle$) und der Art der Veränderung wird auch ein Ziel der Änderung übermittelt. Bei Variablen ist dieses der neue Wert ($Value$), bei Verbindungen das neue Ziel-Objekt, und bei dem Erzeugen einer Instanz wird die Referenz auf das neue Objekt (Referenz $Ziel$) überliefert.

An dieser Stelle wird auch klar, dass diese Sprache einfach gehalten ist. So sind sinnvolle Erweiterungen aus dem leittechnischen Umfeld denkbar: Für die Änderung eines Variablenwertes könnte ein delta-Faktor angegeben werden, welcher geringe Abweichungen für Prozesswerte von einer Benachrichtigung ausschließt.

Komplexere Beschreibungssprachen für Änderungen sind denkbar. Sie könnten nach dem Prinzip von XPATH [32] als Muster, also Teile der Modellstruktur – Objekte, Variablenwerten und (typisierten) Relationen – beschrieben werden. Sollte in der Instanzumgebung eine Änderung auf diese Beschreibung zutreffen, wird eine Änderungs-Benachrichtigung erzeugt.

Insgesamt ist dabei zu beachten, dass das Ausführen dieser Muster-Suche nicht mehr Ressourcen verbraucht, als insgesamt eingespart werden.

Wichtig ist an dieser Stelle, dass die Schnittstelle selber entfernt zugreifbar ist, sodass die Überwachungen auch von entfernten Diensten ausgeführt werden kann. Insgesamt wird durch diese entfernten Zugriffe eine Änderungs-Reaktion des Gesamtsystems erreicht, die auch insgesamt auf lokale Änderungen reagieren kann.

4.5.3.1 Alternative Realisierung: Intents

Eine andere Möglichkeit, gerade in Bezug auf die Verbreitung der Änderungs-Benachrichtigungen, sind die Intents, wie sie in Kapitel 4.3.7 als zusätzlichen Kommunikationstyp vorgestellt wurden.

Dabei wird ein Intent-Typ *Modell-Änderung* definiert. Für diesen können sich alle interessierten Komponenten anmelden. Sie bekommen daraufhin **alle** Änderungen des Modells mit. Die Intent-Nachrichten werden dabei durch eine Instanziierung der Basis-Klasse des Meta-Modells initiiert: Wenn ein Objekt / eine Relation des interessanten Modells erzeugt wird, wird eine Intent-Nachricht versandt.

Zu berücksichtigen für eine Abwägung ist dabei die Anzahl der Änderungs-Benachrichtigungen, sowie die Verteilung der Modelle. Durch die zentralisierte Bearbeitung der Intent-Nachrichten kann es, verglichen mit der oben beschriebenen Lösung, zu einem größeren Kommunikationsbedarf führen.
Auf der anderen Seite steht die geringere Grundlast: Eine interessierte Komponente muss sich lediglich für Modell-Änderungen anmelden. Die später erfolgenden Subskriptionen auf Objekte, die überwacht werden müssen, entfallen.

Diese Realisierungs-Beschreibung verdeutlicht, wie unterschiedlich die Intent-basierte Kommunikation von der klassischen Kommunikation ist. Sie entkoppelt die interessierten Komponenten von den zu beobachtenden Ereignissen.

4.5.4 Ausführungsumgebung: Dienste als partielle, Aufgaben-orientierte Teil-Anwendungen

Im Gegensatz zu den passiven Komponenten, die zwar dynamisch verändert werden können, jedoch selber keinerlei Aktionen auslösen (vgl. Kapitel 4.2), bietet ein Dienst die Möglichkeit, Abläufe auszuführen.

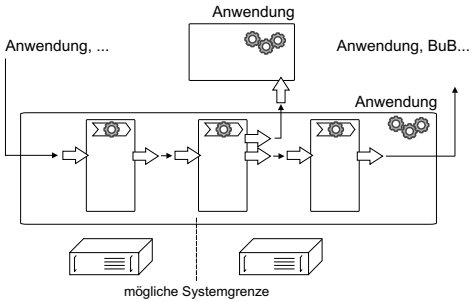


Abbildung 4.20: Orchestrierte Dienste als Anwendung

Ein Dienst ist eine aktive Komponente auf einem Gerät. Er erfüllt eine (Teil-)Aufgabe. Unter Anwendungen werden Kombinationen aus Diensten verstanden, die eine Gesamtaufgabe erfüllen, wie sie vom Endanwender verlangt wird. Unter Orchestrierung wird das Verbinden von Diensten, also das gegenseitige bekanntmachen, verstanden. Damit können solche Orchestrierungen von Diensten als *verteilte Anwendung* verstanden werden, wie es in Abbildung 4.20 dargestellt ist.

Integration im Laufzeitmodell

Durch seine Dienst-Repräsentation ist ein Dienst selber auffindbar und damit auch nutzbar. Auf diese Weise kann auf aufwändige Registrierungs-Komponenten, wie sie normalerweise in SOA-Systemen ein elementarer Bestandteil sind, verzichtet werden. Eine Registrierung eines Dienstes erfolgt durch das Anlegen seiner Dienst-Repräsentation im Laufzeitmodell.

Vielfach werden auch Beschreibungen der Fähigkeiten als eigene Schnittstelle eines Dienstes spezifiziert und diese in (ggf. zentralen) Komponenten verwaltet. Diese können zwar existieren, sind aber als Annotationen in der Dienst-Repräsentation (allgemeiner aktive Komponenten-Repräsentationen) abzulegen, sodass sie dort direkt durch die Dienst-Repräsentation zugehörig zum Dienst erkundbar und auch aktualisierbar sind.

Andere Sichtweise: Dienste als komplexe Modell-Änderungs-Schnittstellen

Wenn ein Dienst als Modell-verarbeitende Teil-Anwendung verstanden wird, der - wie der grundlegende Gedanke - alle seine Daten im Laufzeitmodell ablegt, ist auch eine andere Sichtweise auf die Funktion eines Dienstes zulässig:

Ein Dienst, der alle seine Daten im Laufzeitmodell ablegt und keinen weiteren Zustand oder weitere Variablen hat, stellt eine spezialisierte Schnittstelle zum Laufzeitmodell dar. Er ist äquivalent zu Basis-Operationen, die direkt im Laufzeitmodell ausgeführt werden zzgl. seines eigenen Verhaltens.

Dieses gilt nicht für alle Dienste - beispielsweise würde ein Dienst zur Erfassung von Messwerten seine Daten nicht aus dem Modell beziehen, sondern von einem am Gerät angeschlossenen Sensor.

4.6 Modell- und Gerätegrenzen

Klassisch liegen Modelle unabhängig (d.h. ohne Relationen zwischeneinander) auf Instanzumgebungen, die jeweils auf einem Gerät existieren.

Eingangs wurde beschrieben, dass die beiden Begrenzungen aufgehoben werden sollen:

Modelle in Relation Modelle untereinander in Relation setzen zu können hat den Vorteil, dass einzelne Modelle unabhängig voneinander entwickelt werden können. Aufbauend können dann weitere Modelle Relationen zwischen den Modellen beschreiben.

Über Gerätegrenzen hinweg Geräte separieren bisher Informationen, die in unterschiedlichen Modellräumen vorliegen. Diese Trennung erscheint hinderlich, denn verteilt abgelegte Daten z.B. unterschiedlicher Anwendungen, stehen im Sinne von Modellen durchaus in Beziehung.

Im Folgenden wird für beide Problemstellungen jeweils ein Lösungskonzept vorgestellt.

4.6.1 Modelle in Relation: Modell-Interkonnektion

Eine der wesentlichen Vorteile, die durch die Modelle zur Laufzeit entstehen, ist die Verfügbarkeit von Informationen. Häufig liegen diese Informationen derzeit in unterschiedlichen Programmen bzw. deren Dateien und eigenen Formaten unverknüpft und unzugänglich vor. Dieses betrifft insbesondere Engineering-Systeme, deren Daten traditionell zur Produktions-Phase nicht verfügbar sein mussten.

Wenn eine modellgetriebene Instanzumgebung diese Informationen in strukturierter Weise - in mehreren, unabhängigen Modellen - zugänglich bereithält, stehen die Modell-Instanzen

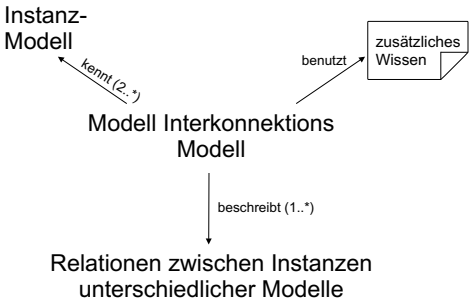


Abbildung 4.21: Relationen zwischen unabhängigen Modellen - Konzept der Modell-Interkonnektion

nebeneinander und sind für alle Anwendungen erkundbar. Hierdurch können Engineering-Informationen in der Produktions-Phase genutzt werden.

Einen weiteren Mehrwert wird geschaffen, wenn auch Beziehungen zwischen den strukturierten Detail-Informationen der Modelle abgebildet werden. Es erscheint also sinnvoll die Modell-Instanz-Teile untereinander zu verknüpfen.

Definition: *Eine Verknüpfung zwischen Komponenten verschiedener Modelle wird* ***Modell-Interkonnektion*** *genannt.*

Um Teile von Modellen untereinander in Beziehung setzen zu können, muss zusätzliches Wissen vorliegen, welches die Verbindung repräsentiert. Dieses kann insgesamt wieder als eigenes Modell verstanden werden.

Somit werden die folgenden Informationen für ein Modell von Modell-Interkonnektionen benötigt:

- Verständnis beider Modellen, die in Verbindung gesetzt werden sollen.
- Verständnis von der Verbindung, die zwischen den Modell-Instanzen aufgebaut werden soll (eigener Modell-Inhalt).
- Beschreibung der Interkonnektionen (Interkonnektions-Klassen).
- Meistens wird eine dritte Informationsquelle nötig sein, d.h. zusätzliches Wissen, welches aussagt, welche Relation zwischen welchen Modell-Instanz-Teilen anzulegen ist (repräsentiertes Wissen).

Dieses Modell ist von den Modellen, dessen Komponenten verbunden werden, abhängig. Vereinfacht wird die Idee der Modell-Interkonnektionen in Abbildung 4.21 verdeutlicht.

Damit besteht eine Modell-Interkonnektion zu einem Teil aus einem eigenen Modell, welches auf Basis von (zwei oder mehr) Modellen beschreibt, welche Relationen existieren. Eine ent-

sprechende Realisierung der Modell-Interkonnektion kann demzufolge eigene Relationen mitbringen, die zwischen Elementen der anderen Modelle in der Instanzumgebung angelegt werden. Das Modell der Modell-Interkonnektionen zwischen den Modellen kann aber selbstverständlich auch komplexere Beschreibungen als einfache Relationen zwischen Modell-Instanz-Teilen enthalten.

Daraus lässt sich folgende Definition für ein Modell von Modell-Interkonnektionen ableiten:

Definition: ***Modell-Interkonnektionen*** *werden in* ***Modell-Interkonnektions-Modellen*** *(MIM) beschrieben. Ein MIM bezieht sich mindestens auf zwei Modelle (abhängige Modelle). Ein MIM hängt von diesen zwei Modellen ab, nutzt sie zur Selbstbeschreibung und bringt Wissen in das Gesamtsystem ein (repräsentiertes Wissen) aufgrund dessen Interkonnektionen instanziiert werden können.*

Die Reaktion auf Modell-Änderungen (Anlegen/Löschen der Modell-Interkonnektionen) ist Aufgabe der Modell-Interkonnektions-Komponenten, die später beschrieben werden.

4.6.1.1 Beispiel: AT-Geräte-Struktur und Anlagenstruktur

Als praktisches Beispiel soll wiederum das AT-Geräte-Struktur-Modell dienen.

Während das AT-Geräte-Struktur-Modell lediglich Aussagen über die Existenz sowie Adressen der AT-Geräte einer Anlage macht, kann ein Anlagenstrukturmodell in PandIX (vgl. 2.2) die Anlage selber abbilden.

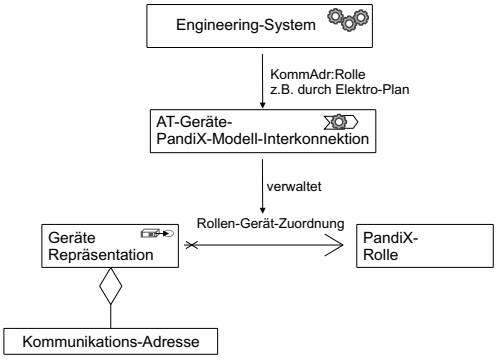


Abbildung 4.22: Skizze der Modell-Interkonnektion AT-Geräte-Struktur mit PandIX

Sinnvoll erscheint eine Verbindung zwischen den Rollen der Geräte aus der Anlagenstruktur und den Kommunikations-Adressen bzw. den Modell-Instanz-Objekten des AT-Gerätestruktur-Modells aufzubauen. Das repräsentierte Wissen, welches Gerät zu welcher Rolle gehört, muss

die Modell-Interkonnektion aus dem Engineering-System beziehen. Abbildung 4.22 verdeutlicht die Interkonnektion.

4.6.2 Über Gerätegrenzen hinweg: Verteilte (Modell-)Laufzeiten

Die Integration von Modellen auf einem Gerät ermöglicht den Zugriff auf die Informationen durch unterschiedliche Anwendungen. In den heutigen Anlagen existieren eine große Anzahl von Geräten, die ein Interesse haben können, auf diese Informationen zuzugreifen bzw. selber Informationen in der Instanzumgebung abzulegen. Das Ziel dieses Kapitels ist es eine gemeinsame, **verteilte** Modell-Instanzumgebung zu konzipieren, in der sich die zuvor beschriebenen Modelle abbilden und über Gerätegrenzen hinweg in Verbindung setzen lassen.

Hierdurch können zur Laufzeit Neuerungen dynamisch eingebracht werden, indem das Modell zuerst instanziiert wird und nachträglich in der Anlage auf den verteilten Geräten in Relation gesetzt wird. Es folgt, dass es sinnvoll ist, Verteilungsaspekte für Modell-Instanzsystem zu konzipieren. Die Anlage selber wird also als ein verteiltes System mit Modell-Instanzsystem verstanden.

Verteilte Anwendungen können so eine verteilte Modell-Laufzeit selbstverständlich nutzen - insbesondere um Erkundungen über die Repräsentationen der Dienste durchzuführen, wie in Kapitel 6.1 beschrieben wird. Durch eine entsprechende verteilt Ausführungsumgebung können die Anwendungen selber die Verteilung nutzen, wodurch der SOA-Kerngedanke aufgenommen wird.

Eine alternative, naheliegende Lösung wäre, dass - ähnlich wie im traditionellen OPC-Ansatz - alle Daten auf einem zentralen Server abgebildet werden und alle Geräte als Klienten Zugriff bekommen. An dieser Stelle sollen jedoch Konzepte zur realen Verteilung der Modell-Instanzen über Gerätegrenzen hinweg konzipiert werden. Hierdurch ist es beispielsweise möglich, die klassische Datenhaltung beizubehalten. Jedes Gerät behält also die Informationen lokal auf seinem Speichermedium, die es hauptsächlich bearbeitet. Diese Informationen werden lediglich in Relation zu anderen gebracht und müssen dabei ggf. über Gerätegrenzen hinweg gezogen werden.

4.6.2.1 Externe Verbindungen

Für eine Realisierung bedarf es also eines Konzeptes um die Elemente eines modellgetriebenen Instanzsystems untereinander auf unterschiedlichen Geräten in Relation zu bringen: Es sind Verbindungen von einzelnen Modell-Instanz-Objekten und -Relationen auf unterschiedlichen Geräten bereitzustellen.

Dafür wird das Konzept der *Externen Verbindungen* entworfen. Dabei ist wichtig zu verstehen, dass diese Kommunikation aus Sicht der Modelle sowie auch der Modell-Erkundung (vgl. Ka-

pitel 5.4) transparent sein muss, d.h. es existiert eine Kommunikationsmöglichkeit, die diese externe Verbindungen als normale Verbindungen erscheinen lässt.

Hintergrund:
Mit der „ExpandedNodeId“ bietet OPC-UA ähnliche Möglichkeiten an: Hier wird das Ziel in der Namespace-URI abgelegt und bezieht sich auf eine „NodeId“ eines anderen Servers. Im Gegensatz zu dem hier vorgestellten Konzept ist dieses jedoch ein einseitiger Verweis. Das Ziel hat keinerlei Möglichkeit festzustellen, dass auf dieses verwiesen wird. Hierdurch fehlt die Möglichkeit, bei einer Modell-Erkundung vom Ziel auf die Quelle zu schließen.

Das Konzept sieht vor, dass Kopien eines Objektes auf unterschiedlichen Geräten existieren. Diese beiden Objekte werden durch die Klasse *Externe Verbindung* (Ext.-Verbindung) als äquivalent dargestellt. Nebenbedingung ist dafür, dass beide Geräte über die entsprechenden Klassenbeschreibungen verfügen.

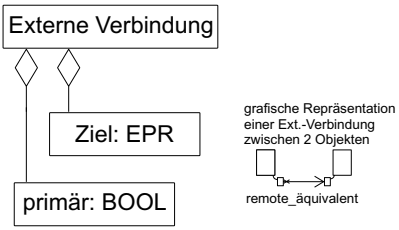


Abbildung 4.23: Ext.-Verbindung für die Repräsentation eines entfernten Zugriffspunktes

Das zentrale Objekt ist eine Instanz der Klasse Ext.-Verbindung (Abbildung 4.23). Dieses speichert als EPR die Objekt-Referenz (Kapitel 4.3.1) auf einem entfernten System. Dabei ist die EPR eine Referenz auf ein Objekt im entfernten Daten-Modell. Ein Paar von Ext.-Verbindungs-Objekten kann so selber als Relation *remote_äquivalent* verstanden werden.

Das Meta-Modell beschreibt Objekte und Relationen zwischen Objekten. Damit diese per Ext.-Verbindung über Systemgrenzen hinweg als äquivalent dargestellt werden, muss für beide die Verteilung durch das Ext.-Verbindung Konzept beschrieben werden:

Ext.-Verbindung: Objekte Damit beide Objekte äquivalent behandelt werden können, ist es nötig, dass immer auf beiden Seiten ein Ext.-Verbindungs-Objekt angelegt wird, das jeweils auf das andere Objekt referenziert (vgl. Abbildung 4.24).

Wichtig ist, dass hierdurch nur die Äquivalenz dargestellt wird. Es wird keine Synchronisation der Daten vorgenommen. Ein Ext.-Verbindungs-Objekt hat deswegen einen Boolean *primär*, welches angibt, ob das lokale Objekt als primär anzusehen ist.

Im Falle, dass zwei Objekte als äquivalent markiert werden, handelt es sich bei der Ext.-Verbindung also (vorläufig) um eine $1:1$ Beziehung.

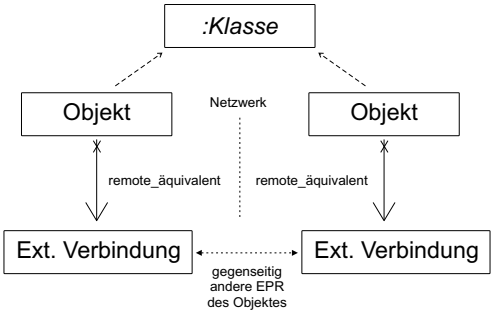


Abbildung 4.24: Abbildung des Konzeptes Ext.-Verbindung auf ein Objekt über Geräte-Grenzen hinweg

Ext.-Verbindung: Relationen Dieses Prinzip ist übertragbar auf Relationen. Hier ist vorstellbar, dass eine Quelle auf einem Gerät liegt, während ein Ziel sich auf einem anderen Gerät befindet.

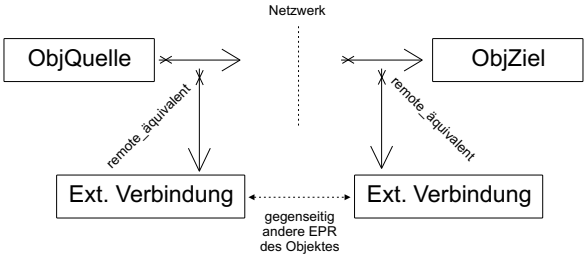


Abbildung 4.25: Abbildung des Konzeptes auf eine Relation über Geräte-Grenzen hinweg

Abbildung 4.25 zeigt, dass die Verbindung, die auf ein entferntes Objekt zeigt, selber ein Objekt ist und damit als Quelle für die *remote_äquivalent*-Relation dienen kann. Eine Relation, die auf ein entferntes Objekt zeigt, hat lokal „offene Enden“: Entweder hat sie keine Quelle oder kein Ziel. Die fehlende Information wird mittels der Ext.-Verbindungs-Objekte abgebildet: Per *remote_äquivalent*-Relation werden die „offenen Enden“ miteinander verknüpft. Es wird also das Relations-Objekt als entfernte Kopie per Ext.-Verbindung dargestellt, was insgesamt zu der Modellierung führt, dass die Verbindung auf einem anderen Gerät „weitergeführt“ wird.

Damit eine $1 : N$ Relation abgebildet werden kann, muss eine Quelle mit mehreren Ext.-Verbindung Objekten zu ihren N Zielen abgebildet werden. Hieraus ergibt sich, dass auch *remote_äquivalent* eine $1 : N$ Relation sein muss.

4.6.2.2 Änderungs-Benachrichtigungen

In Kapitel 4.5.3 wurde eine Schnittstelle beschrieben, die, basierend auf dem Subskription/Benachrichtigungs-Typ, Änderungen in der Objekt-Struktur in der modellgetriebenen Instanzumgebung aktiv entsprechenden Komponenten mitteilt.

Diese Schnittstelle kann im verteilten Umfeld genutzt werden: Die Ext.-Verbindungs-Objekte müssen sich bei dem als primär markierten Objekt für Änderungen subskribieren. Hierdurch werden Änderungen auf dem entfernten Gerät bekannt.

Durch das Konzept der Annotationen (vgl. Kapitel 4.2.3) ist auch die Synchronisation von Objekt-Kopien mit ihrem Original realisierbar: Es müssen lediglich die Schlüssel-Wert-Paare übertragen werden.

4.6.2.3 Zugriff auf die verteilte Modell-Instanzen

In einem nicht-verteilten Szenario ist das Auffinden einer Modell-Instanz vergleichsweise einfach. Je nach Implementierung können entweder vorgegebene Identifikationsmechanismen [6] als Einstiegspunkt vereinbart werden, oder es kann über die vorhandenen Klassen-Definitionen der Instanzen ermittelt werden.

Im verteilten Fall ist das Vorgehen komplizierter. Es muss die Frage beantwortet werden, wie eine Anwendung eine oder alle Modell-Instanz(en) von einem Modell finden kann.

Es gibt im Wesentlichen vier Lösungs-Ansätze:

Vorprojektierte Kommunikations-Adresse & Einstiegspunkt auf Geräten Erweiterung der Lösung für das lokale Finden der Modell-Instanzen. Ein Nachteil ist, dass eine Adresse auf allen Teilnehmern vorprojektiert werden muss.

Suche auf Kommunikationsebene Spezielle Nachrichten für die Suche von Komponenten können etabliert werden, um Modell-Instanzen zu finden. Diese Lösung würde die Anwendungen direkt von der unterliegenden Kommunikation abhängig machen und die in Kapitel 2.3 geforderte Unabhängigkeit der Kommunikation von den Daten in Frage stellen.

Such-Dienst Ein Dienst mit speziellen Nachrichten für die Suche von Komponenten kann etabliert werden um Modell-Instanzen zu finden. Eine solche Lösung bietet die volle Flexibilität, jedoch muss der entsprechende Dienst zur Ausführung gebracht werden und belegt somit selber Ressourcen. Diese Lösung wurde in [Dri05] ausführlich dargestellt.

Suche über gemeinsames Meta-Modell Das Meta-Modell beinhaltet Verwaltungs-Strukturen um geladene Modelle und daraus instanziierte Objekte aufzufinden. Wenn diese Informationen system-übergreifend genutzt werden können, ergibt sich eine Form des Yellow-

[6] In ACPLT wäre dieses ein OV-Pfad; in OPC-UA eine NodeId

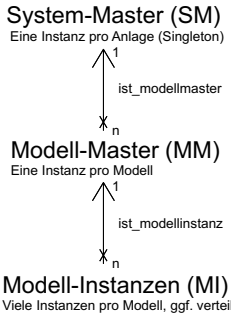


Abbildung 4.26: Relationen um Modelle verteilt zu handhaben

Paging-Konzeptes: Das Meta-Modell kann genutzt werden um alle Modelle auf allen Geräten aufzufinden.

An dieser Stelle wird sich auf die vierte mögliche Lösung konzentriert. Dabei wird jedoch in dem verteilten Modell-Raum eine zentrale, vorprojektierte Yellow-Paging-Adres-se (EPR) definiert. Hier werden die Einstiegspunkt für die jeweiligen Modelle; also die Modell-Master durch ihre EPR erkundbar.

Ein wesentlicher Vorteil dieser Lösung ist, dass keine explizite Schnittstellenbeschreibung bzw. Nachrichten-Spezifikation notwendig ist. Die im Folgenden beschriebene Lösung bildet sowohl den initialen Einstiegspunkt für ein Modell, wie auch für die Verteilung der Modell-Instanzen eine interne Erkundungsmöglichkeit. Auf diese Weise ist das System unabhängig von dem verwendeten Kommunikations-Medium realisierbar.

Die Suche nach in Modellen abgelegten Informationen lässt sich für Anwendungen in drei Schritte aufteilen:

1. Suche nach einem zentralen Einstiegspunkt für das jeweilige Modell (*Modell-Master*) des gesuchten Modells an der Yellow-Paging-Adresse.

2. Suche mit Hilfe der Modell-Master nach der oder den Instanz(en) des Modells.

3. Suche nach den gewünschten Informationen innerhalb der gefundenen Modell-Instanz ist dann äquivalent zu der Suche auf einem lokalen System.

Die ersten beiden Stufen werden genauer betrachtet, die dritte Stufe ist Modell-spezifisch und bedarf hier also keiner weiteren Beschreibung.

Suche nach Modell-Master: Organisation der verteilten Modelllandschaft Es wird eine vorprojektierte, gemeinsame Adresse als gemeinsamer Einstiegspunkt angenommen (Abbildung 4.26).

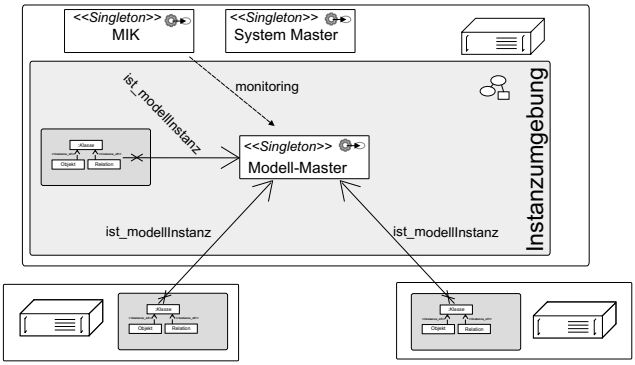


Abbildung 4.27: Vorschlag für die Realisierung des Zusammenspiels aus MIK und Modell mit Modell-Master.

Unter dieser Yellow-Paging-Adresse ist eine Komponente *System-Master* (SM) ansprechbar. Im Sinne des Yellow-Paging verwaltet dieser eine „Liste“ der Modell-Master. Im Sinne der hier vorstellten modellgetriebene Instanzumgebung wird diese Liste durch Relationen vom Typ *ist_modellmaster* vom System-Master zu den einzelnen Modell-Master der Modelle repräsentiert.

Anfragende Anwendungen müssen also lediglich den Relationen ist_modellmaster zu den Modell-Mastern folgen.

Für neu installierte Modelle oder genauer ihre Modell-Master gilt hingegen, dass sie eine solche Relation ist_modellmaster anlegen müssen, womit sie sich in die Liste der Modelle eintragen. Sie kann - und wird im Normalfall - eine Ext.-Verbindung sein, da der Modell-Master auf einem anderen Gerät liegt als der System-Master.

Die zentrale Rolle dieser System-Master Komponente ist aus Verfügbarkeits- sowie Ressourcen-Sicht nicht weiter kritisch, da eine Erkundung der Modell-Instanzen nur selten erfolgt - beispielsweise bei der Inbetriebnahme eines neuen Gerätes oder bei der Umstrukturierung von Anwendungen.

Suche mit Hilfe der Modell-Master nach der oder den Instanz(en) des Modells Da anfragende Anwendungen von dem zu erkundenden Modell eine Modell-Vorstellung benötigen, ist es auch hinnehmbar, dass spezifische Realisierungen des Modell-Masters existieren - die Anwendungen müssen also spezielle Kenntnisse über den Modell-Master besitzen. Die Modell-Master können so auf unterschiedliche Weise realisiert werden.

An dieser Stelle wird lediglich *eine* Realisierung-Möglichkeit beschrieben.

Viele Modelle, wie auch das Beispiel-Modell der AT-Geräte-Struktur, werden durch ihre Anwendung nur einen Wurzelknoten in einem System haben.
Für den Modell-Master heißt dieses, dass er gleichzeitig das Wurzel-Element beinhalten oder sogar darstellen kann. Damit ist in diesem Fall eine Suche nach der richtigen Modell-Instanz hinfällig. Eine suchende Anwendung kann also durch ihr Modell-Wissen, ausgehend vom durch den Modell-Master gefundenen Wurzel-Knoten die Informationen suchen. In diesem Fall gibt es also lediglich eine Modell-Instanz (MI), wie auch in Abbildung 4.27 dargestellt.

Es gibt jedoch auch Modelle, welche mehre Modell-Instanzen aufweisen. Vorstellbar sind hier beispielsweise Konfigurations-Datensätze für Geräte. Diese können in einer Struktur als Modell abgelegt werden. So wird für jedes Gerät unabhängig von anderen Geräten die Konfiguration darstellbar sein.
Eine suchende Anwendung findet also durch den System-Master den Modell-Master, der eine Liste von Relationen vom Typ *ist_modellinstanz* unterhält. Hierüber kann die suchende Anwendung die gewünschte Instanz finden.

Die Modell-Instanzen können dabei entweder lokal auf dem Modell-Master organisiert werden, oder es werden entsprechende Relationen zu anderen Geräten hergestellt, auf denen andere Teile der Modell-Instanzen abgelegt sind. Diese Verbindungen werden durch das zuvor beschrieben Ext.-Verbindung Konzept realisiert.

4.6.2.4 Dienst-Orchestrierung auf Basis der verteilten Modelle

Von diesem Konzept können auch aktive Komponenten / Dienste profitieren.

Äquivalent zu der Suche nach im Modell abgelegten Informationen durch den System-Master, den Modell-Master und die Modell-Instanzen, kann auch eine Suche nach Diensten erfolgen. Diese Suche erfolgt dabei durch die Suche nach dem Modell-Master für das Dienste-Modell. Dieser enthält dabei - im einfachsten Fall - eine Relation zu allen in der Anlage verfügbaren Dienst-Repräsentationen. Hierdurch sind also die Dienste prinzipiell auffindbar.

Die Suche gestaltet sich jedoch aufwändig, da im Endeffekt alle Repräsentationen (selbst wenn diese auf einem Gerät zentralisiert abgelegt sein sollten) durchsucht werden müssen. Einen Ausgangspunkt zur Reduktion können auch die Klassenbeschreibungen der Dienste und die instanz_von-Relation sein.

Vielversprechender ist es, wenn Modelle für die Aufgabenbeschreibung existieren würden. Die Dienst-Repräsentationen könnten entsprechend mit den Aufgabenbeschreibungen, die sie erfüllen können, in Relation gesetzt werden. Eine solche Aufgabenbeschreibung für Dienste ist jedoch aufwändig und nur sinnvoll, wenn eine entsprechende Standardisierung betrieben wird - entsprechende Versuche der Dienst-Beschreibung sind im Bereich der Dienst-orientierten Architekturen bisher leider nicht von Erfolg gekrönt.

4.6.2.5 Beispiel: Verteilte Modellierung der AT-Geräte-Struktur

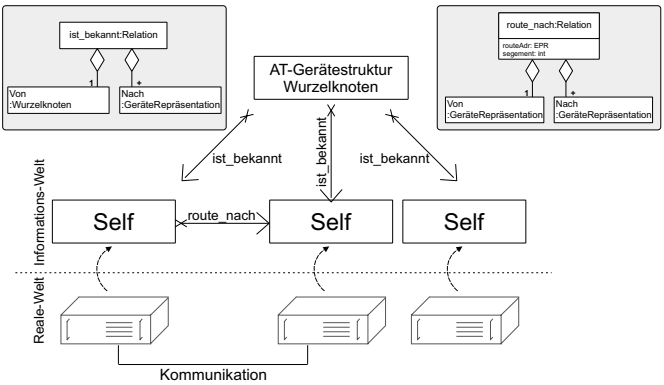


Abbildung 4.28: Das AT-Geräte-Struktur-Modell auf Basis der Trennung von Informationswelt und Realität

Zu jedem Gerät gehört eine Instanz „Self“. Unter dieser Instanz vom Typ *Geräte-Reprä-sentation* verwaltet ein Gerät seine eigene Addressierungsmöglichkeit in Form der EPR, sodass sie im Modell-Raum erkundbar ist. Dieses Objekt ist zum einen für die Handhabung der Annotationen auf einem Gerät vorhanden, also z.B. damit eine aktive Komponente bestimmen kann, unter welcher Adresse sie erreichbar ist: Die EPR einer Komponente ergibt sich aus der Kommunikations-Adresse der Self-Komponente und der lokalen Dienstadresse, wie in Kapitel 4.3.1 definiert.

Zum anderen kann es aber auch durch den Basis-Zugriff der Instanzumgebung von außen erkundet werden. Dabei wird eine zentrale Instanz in einer Anlage für die Modell-Instanzen vorgeschlagen. Die grundlegende AT-Geräte-Struktur wird also auf einem zentralen Gerät abgelegt. Teil-Anlagen oder auch komplexe andere Teilnehmer können sich aber von diesem Gerät durch die Mechanismen der verteilten Modelle auf anderen Geräten befinden und dort erkundbar bleiben. Die Adresse des zentralen Einstiegspunktes um die AT-Geräte-Struktur zu erkunden wird per System-Master zugreifbar gemacht werden.

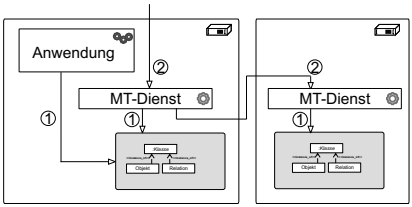
Abbildung 4.28 stellt für die Struktur-Abbildung zwei Relationen dar: Die *ist_bekannt* Relation, die die Liste der Geräte darstellt und die *route_nach* Relation, die die Verbindungen zu anderen Geräten, also insgesamt die Kommunikationsstruktur, darstellt.

4.6.2.6 Transparenter Zugriff auf verteilte Modell-Instanzen

Aus dem bisherigen Beschriebenen leitet sich ab, dass eine Anwendung, die verteilte Modelle erkunden will, den Ext.-Verbindung Relationen folgen muss, d.h. sie muss selber in der Lage sein, lokale von externen Verbindungen zu unterscheiden. Bei Ext.-Verbindungen muss die Re-

ferenz des gegenüberliegenden Ext.-Verbindungs-Objektes entsprechend über die hinterlegte Adresse bezogen werden und dort die weitere Erkundung vorgenommen werden.

Diese Problematik kann jedoch aufgelöst werden, indem die Kommunikation mit dem eigentlichen modellgetriebenen Instanzsystems nicht direkt, sondern durch eine besondere Schnittstelle erfolgt. Beide Lösungen sind in Abbildung 4.29 dargestellt.



① SystemBasisDienst ② SystemBasisDienst (SBD') mit EPR als Referenz

Abbildung 4.29: Zugriff einer Anwendung auf die Modell-Laufzeit: Entweder direkt per SystemBasis-Dienst oder mittels des MT-Dienstes

Dieser Modell-Transparenz-Dienst (MT-Dienst) bietet alle SystemBasisDienste an, nutzt als Referenz sowohl für Aufrufe, wie auch für Antworten, eine EndPointReference, die neben der Kommunikations-Adresse auch mit der internen Referenz des Zielsystems versehen ist. Sie wird als *SystemBasisDienst'* (SBD') bezeichnet. Die Schnittstelle leitet dabei SystemBasis-Dienst-Aufrufe, die lokal erfolgen, entsprechend an das Zielsystem, welches in der EPR vermerkt ist, weiter.

Dadurch, dass die Antworten - egal ob sie lokal ausgeführt wurden oder auf einem anderen System - immer EPRs enthalten, kann die aufrufende Anwendung diese EPRs als Referenz nutzen. Das heißt, dass die EPR mit ihren verschiedenen Elementen **nicht** auf ihre Bestandteile untersucht wird, sondern insgesamt als Referenz genutzt wird.

Die Komponente Modell-Explorer in Kapitel 5.4 realisiert später dieses Konzept für ein Gerät.

Hieraus folgt, dass Anwendungen, wenn sie auf Informationen in potenziell verteilte Modell-Instanzen zugreifen wollen, immer gegen den MT-Dienst entwickelt werden, nicht jedoch gegen die SystemBasisDienste selber.

Selbst Verbindungen über Systemgrenzen hinweg können von dieser Schnittstelle durch entsprechendes Anlegen der Ext.-Verbindungs-Relationen instanziiert werden, ohne, dass die aufrufende Anwendung die Verteilung berücksichtigen muss. Die vorgeschlagene Modell-Interkonnektion basiert also sinnvollerweise auf dieser Schnittstelle.

Im Gegensatz dazu sind Anwendungen zu nennen, die die Verteilung aktiv steuern. Sie benötigen den direkten Zugriff auf die realen SystemBasisDienste, um die Modell-Instanzen direkt anlegen zu können.

Einhalten der AKID-Eigenschaften über Systemgrenzen hinweg In Kapitel 4.3.5 wurden die SystemBasisDienste beschrieben.

Werden diese verteilt ausgeführt, so müssen die AKID-Eigenschaften auch hier berücksichtigt werden. In der Literatur gibt es weitreichende Maßnahmen um Änderungen auf verteilten Systemen unter Berücksichtigung der AKID-Eigenschaften auszuführen. Hier sind insbesondere Konzepte aus dem Bereich der föderierten Datenbanksysteme zu sehen, wie in Kapitel 2.5 erwähnt.

4.7 Effizienz der Konzepte

An dieser Stelle kann keine konkrete Abschätzung für die Effizienz der Konzepte vorgenommen werden, da bisher weder die unterliegenden Geräte und Kommunikations-Medien festgelegt sind, noch die konkreten Realisierungen der Modelle und ihrer Interaktion sowie deren Verteilung fest stehen.

Im Wesentlichen sind zwei der präsentierten Konzepte für eine Effizienz-Einschätzung relevant:

Zum einen bieten die Modell-Interkonnektionen eine Möglichkeit, Verbindung zwischen bisher getrennten Informationen durch ein eigenes Modell darzustellen. Die Verbindungen innerhalb der Daten-Modelle der Anwendungen sind dabei von untergeordneter Bedeutung, da sie nur einen geringen Speicherbedarf und (in der lokalen Variante) keinen Kommunikationsbedarf benötigen. Kritischer müssen hier die Verwaltung der Modell-Interkonnektionen betrachtet werden. Durch die geforderte Dynamik muss einem Hinzufügen eines Objektes in einem Modell ggf. eine neue Modell-Interkonnektion erstellt oder gelöscht werden. Damit müssen Änderungen im modellgetriebenen Instanzsystems aktiv durch entsprechende Komponenten (in Kapitel 5.5 vorgestellt) überwacht werden. Dieses kann in Bezug auf die Laufzeiteffizienz kritisch sein.

Zum anderen ist das Verteilen von Modell-Instanzen über Systemgrenzen hinweg je nach Anwendungsfall zu analysieren, also die verteilte Modell-Instanzumgebung insbesondere das Ext.-Verbindungs-Konzept. Die für die Manipulationen der Modell-Instanzen vorzunehmende Kommunikation selber ist äquivalent zu den existierenden Umgebungen. Zusätzlich ist jedoch der Kommunikations-Bedarf des Suchens und der Synchronisation durch die Ext.-Verbindungs-Objekte zu betrachten.

4.7.1 Modell-Interkonnektionen und ihre Etablierung

Modell-Interkonnektionen sind Relationen und damit lediglich einfache Objekte. Sie benötigen im Verhältnis zu den Modell-Instanzen also kaum Speicher. Da es sich um Abbildungen im Modell handelt, benötigen sie selber, solange es keine Ext.-Verbindungen sind, keinerlei Rechenzeit.

Anders sieht es da bei den Komponenten aus, die die Modell-Interkonnektionen verwalten. Sie benötigen zum einen Rechenleistung, um aus Änderungen ggf. neue Relationen zu etablieren. Zum anderen werden sie Gebrauch von der Subskription/Benachrichtigungs-Schnittstelle der modellgetriebenen Instanzumgebung machen. Dieses kann ein Problem werden, denn im Idealfall sollten alle involvierten Modelle auf einem Gerät lokal instanziiert werden, sodass lediglich lokale Kommunikation nötig ist.

Zusammenfassend gilt:

- Modell-Interkonnektionen selber benötigen einen geringen Speicher.
- Die Verwaltung der Modell-Interkonnektionen benötigen Speicher-, Rechen- und Kommunikations-Ressourcen für Änderungs-Benachrichtigungen durch die Subskription/Benachrichtigungs-Schnittstelle der modellgetriebenen Instanzumgebung.
- Instanziierung der Verwaltungs-Komponenten ist möglichst auf den Geräten vorzunehmen, auf denen entsprechende Modell-Instanzen zu berücksichtigen sind.

4.7.2 Verteilungsaspekte

Wie eingangs bereits beschrieben, sind unterschiedliche Varianten für die Verteilung der Modell-Instanzumgebungen denkbar. Hieraus ergibt sich, dass über die nötige Kommunikation zwischen den Geräten keine endgültige Aussage getroffen werden kann.

So kann eine zentrale Laufzeit, wie es klassischem OPC Umfeld ist, dazu führen, dass keinerlei Kommunikation zur Verwaltung der verteilten Modelle auf Netzwerkebene vorgenommen werden muss. Wenn ein solcher zentralistischer Ansatz jedoch auch für die Modelle des Engineerings genutzt werden soll, ist dieses schwer vorstellbar, da eine Vielzahl von Geräten existiert, die eigene Modelle bzw. Daten beheimaten.

Es macht Sinn, dass Modelle und Modell-Instanzen möglichst lokal auf genau dem Gerät installiert werden, auf dem sie am meisten genutzt werden, d.h. auf genau den unterschiedlichen Rechnern der Engineering-Systeme.

Es können also Kriterien beschrieben werden, die bei einer Verteilung zu berücksichtigen sind.

Ein lokaler Zugriff von einer Anwendung auf eine Modell-Instanz ist in jedem Fall vorzuziehen. Insbesondere in der Engineering-Phase ist eine Hauptaufgabe der Anwendungen die Änderung der Modell-Instanzen, sodass hier eine Verteilung der Modelle auf die jeweiligen Geräte der Anwendungen sinnvoll ist. Hierdurch werden die Modell-Änderungen im Effizienzsinne „günstiger“, während die Modell-Interkonnektionen häufiger durch Ext.-Verbindungen realisiert werden müssen.

Grundsätzlich ist aber die Minimierung der Ext.-Verbindungen ein wesentlicher Aspekt - da diese Verbindungen zum einen ggf. aktiv überwacht werden müssen (vgl. Kapitel 5.6), zum

anderen auch entsprechende Ressourcen belegen. Eng verknüpfte Modelle können so besser auf einem Gerät instanziiert werden.

Es ergibt sich, dass der Grad der Verteilung eine wesentliche Rolle spielt. Da zur Engineering-Phase andere Arbeitsabläufe/Programme auf die Modelle zugreifen, als zur Produktions-Phase, kann eine Reorganisation des Gesamtmodells zwischen Engi-neering- und Produktions-Phase sinnvoll sein.

Zusammenfassend gilt also:

- Minimierung der externen Kommunikation für Geräte
- Minimierung der Ext.-Verbindungen
- Migration von verteiltem Engineering zu einer zentralen Struktur zur Laufzeit

4.8 Integrationsmöglichkeiten in die bestehende AT-Geräte-Landschaft

In den vorstehenden Kapiteln wurde allgemein über ein Gerät in der Automatisierungstechnik gesprochen. Hieraus ergibt sich die Frage, auf welchen Geräten die beschrieben Instanzumgebung sinnvoll zu realisieren ist.

Die Voraussetzungen in Bezug auf die Hardware (Speicher, Rechenleistung und Kommunikation) müssen selbstverständlich erfüllt sein. Im Prinzip lässt sich also eine solche Instanzumgebung für die Systemstrukturen auf Sensor- oder Aktor-Ebene realisieren. Auch Szenarien wie „Intelligente Produkte“ aus dem Kontext „Industrie 4.0“ sind denkbar. So existieren beispielsweise RFID-Leser (also Sensoren), die ihre Daten per OPC-UA als Server bereitstellen. Die in Zukunft vorhandenen Ressourcen und Anwendungsfälle werden hier jedoch Grenzen auflösen, die heute nicht absehbar sind.

Durch die Entkopplung von physikalischem Gerät und Repräsentation in der Informations-Welt ist es nicht nötig, dass jedes Gerät „seine“ Instanzumgebung mit sich bringt. Nach entsprechender Initialisierung beispielsweise in der Inbetriebnahme-Phase ist es möglich, dass sämtliche Handhabungen der Modelle entsprechend „entfernt“ auf einem anderen Gerät erfolgt.

5 Komponenten einer verteilten, modellgetriebenen Ausführungsumgebung

Das Ganze ist mehr als die Summe seiner Teile.
Aristoteles

Nachdem zuvor die Modelle und Konzepte für die verteilte, modellgetriebene Instanzumgebung vorgestellt wurden, liegt der Fokus dieses Kapitels auf den Basis-Komponenten, die die Systemstruktur nutzbar und anwendbar machen - also der System-Architektur.

Diese Komponenten übernehmen dabei zum Großteil Verwaltungs- und Abstraktionsaufgaben, d.h. es handelt sich um lokale - möglicherweise auf allen Geräten instanziierte Komponenten, die den letztendlichen Anwendungen eine Schnittstelle bereitstellen. Wichtig ist hierbei die Modularität der Umgebung in drei Richtungen:

Modell Die im vorangegangenen Kapitel beschriebenen Konzepte zur Modell-Interkonnektion, sowie der Verteilung von Modellen. Sie sind auch unabhängig von den hier vorgeschlagenen Komponenten anwendbar.

Komponenten Die im Folgenden beschrieben Komponenten sind unabhängig voneinander realisierbar, d.h. auch wenn das Gesamtkonzept nicht umgesetzt werden kann oder soll, sind die Ideen anwendbar.

Realisierung Um am Gesamtsystem teilzunehmen, müssen nicht alle Komponenten auf allen Geräten instanziiert sein. Es kann also insbesondere bei Ressourcen-Problemen immer zu Geräten mit geringerem Funktionsumfang kommen - bis hin zu reinen Klienten-Geräten, die das Gesamtsystem lediglich erkunden und selber nicht teilnehmen. Ihre Repräsentation würde also auf anderen Geräten verwaltet werden.

Insgesamt wird so eine verteilte Modell-Instanzumgebung beschrieben. Diese wird *IMLAUF* genannt - *Interkonnektion von Modellen zur Laufzeit*. Auf die Modelle der Instanzumgebung können zum einen externe Programme durch die beschriebenen SystemBasisDienste aus Kapitel 4.3.5 zugreifen. Es können aber auch Dienste als Komponenten in einer IMLAUF-Instanzumgebung als Ausführungsumgebung realisiert werden, die wiederum höherwertige Aufgaben anbieten.

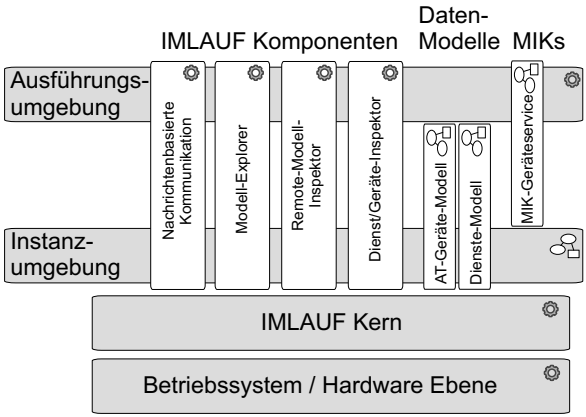


Abbildung 5.1: Eine grobe Übersicht der Basiskomponenten und Architektur des Ausführungssystems

Übersicht über die IMLAUF Komponenten Abbildung 5.1 gibt einen Überblick der Komponenten. Vor der eigentlichen Komponentenbeschreibung wird hier eine Übersicht gegeben. Der Fokus liegt dabei auf dem Zusammenspiel und den Rollen der einzelnen Komponenten:

Als Ausgangspunkt, wie in den Anforderungen in Kapitel 3 definiert, dient eine Softwarekomponente, die Zugriff auf die Ressourcen des Gerätes ermöglicht. Die jeweiligen Anbieter der Geräte sind an dieser Stelle gefordert, eine Ausführungsmöglichkeit bereitzustellen. Diese Ebene wird durch ein Betriebssystem wie Microsoft Windows, Linux oder eine Firmware dargestellt.

Die grundlegendste, wichtigste Komponente ist der IMLAUF Kern. Diese Komponente stellt die Schnittstelle zwischen dem Hersteller des Gerätes und einer einheitlichen Ausführungsumgebung dar, auf denen die hier beschriebenen Modelle und Anwendungen ausgeführt und verwaltet werden.
Dabei werden die verwalteten, wesentlichen Ressourcen in die Kategorien Speicher, Rechenleistung und Kommunikationsleistung eingeteilt. Diese Ressourcen werden den aufbauenden Diensten bereitgestellt, sodass diese die Ressourcen nutzen können. Die Klassifikation und Verwaltung dieser Ressourcen wird als Option bei der detaillierteren Beschreibung im Kapitel 5.1 dargestellt[1].

Obwohl alle im Folgenden beschriebenen Dienste, sowie auch alle Dienste einer Anwendung im Endeffekt diese Komponente nutzen, gibt es zwei Haupt-Komponenten, die die Basisaufgaben abbilden:

Das Instanzsystem bietet die Umgebung, um die Modell-Instanzen zu verwalten. Im Wesentlichen besteht es aus dem in Kapitel 4.5 beschriebenen Container, der die Objekte und Re-

[1] Diese Verwaltung der lokalen Ressourcen hat dabei mit einem verteilten Ressourcenmanagement, welches in den Anforderungen als optional gekennzeichnet wurde, nur insofern etwas gemein, dass es die Informationen über lokale Ressourcen bereitstellen muss.

lationen der Modelle verwaltet und eine entsprechende Erkundung dieser - lokal - ermöglicht. Hier wird durch das im Kapitel 4.6.1 beschriebene Konzept der Modell-Interkonnektion ein gemeinsamer Instanzraum für alle aktiven Dienste geschaffen: Die Ausführungsumgebung. Der einheitliche Objekt-Raum, der die gemeinsame Verwaltung der passiven und aktiven Komponenten in der Ausführungsumgebung darstellt, wurde in Kapitel 4.5.4 beschrieben. Hierbei ist zu beachten, dass die Daten im Modell-Raum abgelegt werden sollen. Gerade ausgehend von bisherigen Anwendungen, ist dies ein großer Wandel: Bisher halten Programme ihre Daten intern und verhindern den Zugriff.

Die Ausführungsumgebung bietet die Schnittstelle für Instanziierung, Anmeldung und Erkundung der aktiven Komponenten bzw. den zuvor definierten Diensten. Im Gegensatz zur Instanzumgebung ist eine Erkundung der Dienste selber nicht vorgesehen, da das Prinzip der Dienste eine Kapselung vorsieht; also einen Dienst als Blackbox ansieht, der sich über seine Schnittstellen definiert und nicht über seine interne Realisierung. Über die vorgestellte Repräsentation der Dienste ist eine Erkundung möglich. Eine weitergehende Erkundung kann auf eine semantische Beschreibung ihrer Funktionalität basieren, die als Annotation an den Repräsentationen hinterlegt wird.

Aufbauend werden drei Typen von Komponenten realisiert. Alle greifen dabei auf die Haupt-Komponenten zurück.

Der erste Typ sind die Klassen-/Instanz-Modelle, die die im Instanzsystem abgelegten Repräsentanzen beschreiben. Hier sind in der modellgetriebenen Instanzumgebung insbesondere das Dienst- sowie Gerätemodell zu nennen. Das Gerätemodell stellt ein (einfaches) Modell für die Verwaltung von bekannten Geräten dar, welches in Kapitel 4 entwickelt wurde.

Der zweite Typ von Komponenten stellen Dienste dar. In einer modellgetriebenen Instanzumgebung existieren einige wichtige Dienste:

- Direkt mit der Ausführungsumgebung ist das *Lokale Service Management* verbunden. Es bietet lokale Funktionen, die von den Diensten zu Verwaltungszwecken genutzt werden können. Nach außen hin bietet es Schnittstellen an, um Dienste zu laden und zu entladen, sowie eine Übertragungsmöglichkeit für die benötigten Daten der Diensttypen. Es nutzt dabei zur Verwaltung entsprechende Relationen vom AT-Geräte-Struktur-Modell und Dienste-Modell in der Instanzumgebung.

- Die Nachrichten-basierte Kommunikation stellt eine zentrale Komponente des Versandes und des Empfangs für asynchrone Nachrichten zwischen den Geräten dar. Diese wird von allen Diensten genutzt.

- Die beiden Komponenten *Dienst-/Geräte Inspektor* und *Remote-Modell-Inspektor* verwalten andere Geräte und überwachen ihre Verfügbarkeit. Aufbauende Dienste können hier ihr Interesse an entfernten Diensten oder Geräten anmelden und werden bei einem gewollten oder ungewollten Strukturwandel von der Änderung informiert. Es handelt sich also um Komponenten zur Vereinfachung der Verteilung.

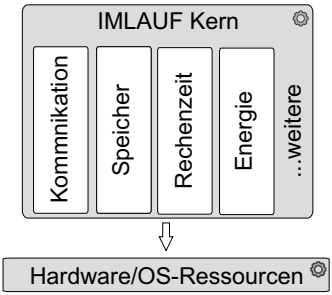


Abbildung 5.2: Ressourcen werden von der Hardware / dem Betriebssystem genutzt. Die Verwaltung und Bereitstellung wird durch den IMLAUF Kern abgebildet.

- Für die Erkundung der Modell-Instanzen bietet der *Modell-Explorer* eine Schnittstelle an. Hierbei wird eine Abstraktion von lokaler und entfernter Modell-Erkundung geschaffen, sodass Dienste, die den Modell-Explorer nutzen, keine eigene Verwaltung der Verteilung der Modelle bzw. Modell-Relationen benötigen.

Der dritte Typ von Komponenten ist aufbauend auf den Haupt-Komponenten sowie den Modellen in der Instanzumgebung zu verstehen: Die Modell-Interkonnektions-Komponen-ten (MIK) - sie realisieren und überwachen die Modell-Interkonnektionen aus Kapitel 4.6.1.
Als Teil der modellgetriebenen Instanzumgebung wird hier ein *Ext.-Verbindungs-MIK* vorgestellt, welches Verbindungen über Gerätegrenzen hinweg anlegt und verwaltet. Es stellt die Basis für die transparente Erkundung des *Modell-Explorers* bereit.

Im Weiteren werden die Komponenten im Detail beschrieben und spezifiziert.

5.1 Ressourcen-Abstraktion: IMLAUF-Kern

Der IMLAUF-Kern (Abbildung 5.2) stellt die eigentliche Instanzumgebung bereit, also die Kopplung und Abstraktion der Hardware bzw. des unterliegenden Betriebssystems. Nach oben bietet sie den aufbauenden Komponenten die Schnittstelle an. So werden die Hardware-Ressourcen an dieser Stelle den aufbauenden Komponenten bereitgestellt. Da sie stark implementierungsabhängig ist, kann an dieser Stelle wenig über die Realisierung gesagt werden.

Ressourcen-Verwaltung Es kann sinnvoll sein, dass die Ressourcen selber *explizit* verwaltet werden. Dieses beinhaltet das Allokieren und Freigeben der Ressourcen, sowie das Beobachten von Engpässen. Alle diese Informationen können in einem Ressourcen-Modell abgebildet werden, womit sie in der Instanzumgebung den Modellen zur Verfügung stehen.

Hintergrund:
Ansätze, diese Verwaltung der Ressourcen zu realisieren, ohne die Anwendungsentwicklung hiermit zu verkomplizieren, bietet die Domäne der virtuellen Maschinen, wie sie bei allen Cloud-Computing Lösungen eingesetzt werden.

Es sind mindestens die folgenden Ressourcen-Kategorien auf einem Gerät zu berücksichtigen:

Rechenleistung Die bereitstehende, allgemeine Rechenleistung.

Speicher Der bereitstehende Speicher. Aufgeteilt in flüchtigen und permanenten Speicher.

Kommunikation Die Kommunikationsmöglichkeiten und deren Bandbreite.

Energie Der Energiekonsum, ggf. in Relation zur den anderen Ressourcen.

weitere Weitere Ressourcen, beispielsweise gerätespezifische Ressourcen (remanenter Speicher etc.)

Die Beschreibung dieser Ressourcen muss quantifizierbar sein, das heißt es muss eine Metrik vorliegen. Gleichzeitig müssen die Relationen zwischen den Ressourcen beschreibbar sein - so hängt beispielsweise der Energiebedarf einer CPU von seiner angeforderten Rechenleistung ab. Auf diese Weise können Dienste entwickelt werden, die (teil-)automatisch die Verteilung von Diensten ermöglichen. Auch kann ein Ressourcen-Mangel im Modell erkannt und entsprechende Alarme oder Umorganisationen ausgelöst werden.

In der Automatisierungstechnik werden einfachere Modelle für die Ressourcenabbildung nur sehr begrenzt und abstrakt eingesetzt. Als ein Beispiel kann die SPS Programmierung genommen werden. Hier wird bei vielen Engineering-Firmen die beim Engineering entstehende Code-Größe oder der allozierte Speicher zur Laufzeit als Referenz genutzt, um zu erkennen inwieweit eine SPS ausgelastet ist. Rechen-Ressourcen werden dabei ebenso abstrahiert, wie Kommunikationsbedarf und Energie.
Die so vorgenommene Vereinfachung ist für die Beschaffung und den Vergleich von Geräten sicherlich sinnvoll - für die Planung von Ressourcen zur Laufzeit jedoch ggf. nicht hinreichend.

Eine detaillierte Definition solcher Metriken scheint heute aber nicht möglich. Insbesondere auf Seiten der Programme, die Ihren Bedarf beschreiben müssen, ist dieses (basierend auf der theoretischen Informatik) nur in Abhängigkeit vom internen Zustand, sowie der Eingabe möglich. Hier gilt es also einen geschickten Zwischenweg für eine Ressourcenbeschreibung für die Automatisierungstechnik zu finden.

Hintergrund:
Siemens stellt zur Laufzeit Probleme mit Rechen-Ressourcen einer SPS im OB82 bereit, wobei man im OB80 eine Reaktion definieren kann - beispielsweise einen Diagnosealarm. Es können also Alarme und Reaktionen zur Laufzeit definiert werden, sobald ein Schwellwert erreicht wurde und die Erfüllung der Zykluszeit in Gefahr ist.

Ein sinnvoller und bewusster Umgang mit den Ressourcen erfordert zum einen eine weitgehende Kenntnis der Realisierung und damit der bereitstehenden Ressourcen. Zum anderen ist aber auch eine Ressourcen-Kalkulation der konsumierenden Komponenten notwendig. Beides wurde im IT-Umfeld, gerade im Bereich des Grid-Computing – beispielsweise durch JSDL [14] – vollzogen. Bei Bedarf kann sie also adaptiert werden, wie es bereits in [ME11] konzeptuell beschrieben ist.

5.2 Die modellgetriebene Instanzumgebung

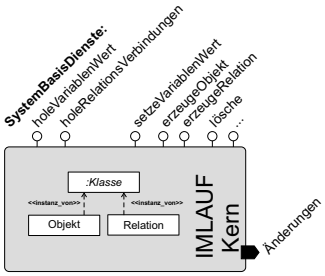


Abbildung 5.3: Schnittstellen der Instanzumgebung

Die in Kapitel 4.5 beschriebene Instanzumgebung der Modelle verwaltet die Modell-Instanzen, also Objekte und Relationen der Modelle.

Sie nutzt dabei Ressourcen und bietet die SystemBasisDienste (vgl. Kapitel 4.3) an, um die Modell-Instanzen zu manipulieren, wie in Abbildung 5.3 dargestellt.

Die interne Architektur der Komponente wird an dieser Stelle aus zwei Gründen nicht konkret beschrieben: Zum einen existieren bereits Industrie-taugliche Lösungen (Kapitel 2.4.1), die die Aufgabe übernehmen können und so als Realisierungen bzw. Realisierungsspezifikationen angesehen werden können. Da der interne Aufbau nur durch die SBD einsehbar ist, können zum anderen aber auch unterschiedliche Prinzipien im Detail ausgearbeitet werden. Zusätzlich ist es wichtig, dass die Realisierung möglichst Speicher- und Rechenleistungs-sparsam geschieht, d.h. ggf. auch angepasst an die entsprechenden unterliegenden Gegebenheiten.

Wie in der Übersicht beschrieben, werden auch aktive Komponenten – also Abläufe – in IMLAUF zur Ausführung gebracht.

Hintergrund:
Die Instanzumgebung ist die Objektverwaltung der Modell-Instanzen und kann damit mit dem Objektverwaltungssystem ACPLT/OV und der Server-Komponente von OPC-UA verglichen werden. Über die Ausführungsmöglichkeiten dieser Lösungen ist zuvor bereits in Kapitel 4.5.4 berichtet worden.

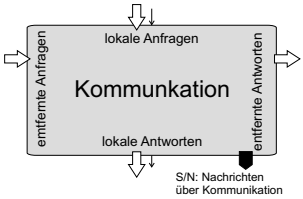


Abbildung 5.4: Schnittstellen der Kommunikations-Komponente

Wichtig für die Reaktion auf Änderungen und damit die Dynamik des Gesamtsystems ist die Änderungs-Benachrichtigung, wie sie in Kapitel 4.5.3 spezifiziert ist. So muss es für Komponenten möglich sein, bei beliebigen Änderungen an dem Modell informiert zu werden und hierauf reagieren zu können.

5.3 Nachrichten-basierte Kommunikation in IMLAUF: MsgSys

Im Kapitel 4.3 wurde ein einheitliches Verständnis der Kommunikation beschrieben.

Die Kommunikations-Komponente *MsgSys* realisiert die Nachrichten-basierte Kommunikation, wie sie in Kapitel 4.3 dargestellt ist. Die unterschiedlichen Kommunikations-Typen (Abbildung 5.4) sowie die Nachrichten-Formate werden auf Basis dieser Komponente durch die Dienste beschrieben und realisiert - die Kommunikations-Komponente ist also hiervon unabhängig.

Einzig die **Intent-basierte Kommunikation** benötigt neben dieser Kommunikations-Komponente eine weitere, zentrale Komponente zum Empfang und Weiterleiten der Intent-Nachrichten. Der prinzipielle Aufbau wurde dabei in Kapitel 4.3.7 bereits dargestellt. Die Umsetzung hiervon bedarf keiner gesonderten Beschreibung.

Diese IMLAUF-Komponente bietet zusätzlich zu der Nachrichten-basierten Kommunikation lokale Schnittstellen an, die es den lokalen Anwendungen möglichst einfach machen, Nachrichten abzuschicken bzw. zu empfangen. Nach außen hin bietet die Komponente die Nachrichten-orientierte Schnittstelle zum Empfang aller Nachrichten für das lokale Gerät an.

Durch die zentrale Rolle auf einem Gerät können auch lokal zuzustellende Nachrichten durch die gleichen Schnittstellen versandt werden. Da die Komponente die Lokalität der Kommunikation durch Analyse des Nachrichten-Kopfes feststellt, kann hierbei auf eine Nutzung des Kommunikations-Mediums verzichtet werden. Dieses Vorgehen bietet die Eigenschaft einer *transparenten Kommunikation*. Der Sender muss also nicht erst berücksichtigen, ob eine Nachricht lokal oder entfernt zuzustellen ist.

MsgSys-Realisierung In Abbildung 5.5 wird verdeutlicht, dass aktive Komponenten durch Nachrichten kommunizieren. Nachrichten sind dabei in erster Linie Objekte mit bestimmten

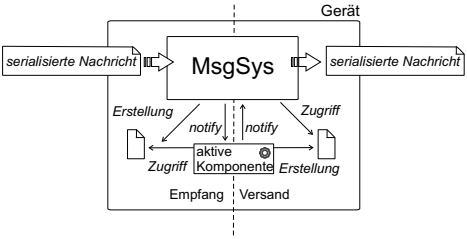


Abbildung 5.5: Übertragung einer Nachricht von einer Sender-Komponente zum Empfänger

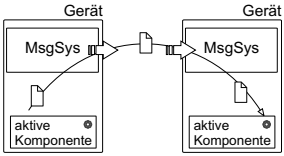


Abbildung 5.6: Aufbau einer Nachricht, Nachrichten-Verarbeitung in einem Gerät

Datenfeldern, wie sie in Kapitel 4.3.3 beschrieben sind. Der Umgang mit diesen Objekten - letztendlich eine Übertragung der Objekte von einem Gerät auf ein anderes - kann durch eine zentrale Komponente erfolgen. Hierdurch wird die Handhabung innerhalb der aktiven Komponenten eingespart. Die Objekte werden von der versendenden Komponente instanziiert und parametriert. Folgend wird die MsgSys Komponente informiert, dass die Nachricht versandt werden soll. Diese greift auf das Objekt zu und serialisiert es, falls es über Gerätegrenzen hinweg übertragen wird. Auf Empfängerseite wird das Objekt wieder hergestellt und eine Benachrichtigung geht an die empfangende Komponente.

Die Abbildung 5.6 verdeutlicht das Vorgehen aus Sicht eines Gerätes beim Senden und Empfangen einer Nachricht durch das MsgSys. Hauptaufgabe ist eine Kommunikations-Möglichkeit durch das entsprechende Kommunikations-Medium zu ermöglichen, wie es die Nachricht als Kommunikations-Adresse der EPR spezifiziert. Dabei wird keine direkte Verbindung zur Empfänger-Komponente aufgebaut, sondern eine serialisierte Version des Nachrichten-Objektes wird an die MsgSys-Komponente des Zielsystems gesandt. Diese kann aus der serialisierten Nachricht wieder ein Nachrichten-Objekt erstellen und die im Kopf spezifizierte Empfänger-Komponente über den Erhalt der Nachricht informieren.

Hintergrund:
Für das in der Realität verbreitete Ethernet mit TCP/IP bedeutete eine solche Realisierung zusätzlich, dass in ggf. vorhandenen Firewalls nur der eine, explizite Port geöffnet werden muss, was eine wesentliche Vereinfachung für die Inbetriebnahme darstellt.

Entsprechend schnell kann auch eine lokale Nachrichten-basierte Kommunikation erfolgen: Hier wird auf die Serialisierung verzichtet und nur die Referenz auf das Nachrichten-Objekt an die Ziel-Komponente weitergereicht.

Modellierung der Kommunikation innerhalb der Instanzumgebung Im Sinne der Repräsentation ist es insbesondere für zukünftige Anwendungsfälle mit flexiblen Kommunikations-Strukturen sinnvoll, die Kommunikation selber auch in der Instanzumgebung abzulegen. D.h. sowohl Kommunikations-Kanäle der Kommunikations-Medien werden dort als Objekt abruf- und erkundbar, wie auch die Aufrufe (Nachrichten) selber.

Durch eine so weitreichende Abbildung der Kommunikation im Modell-Raum werden für andere Anwendungen auch Reaktionen auf neue und fehlgeschlagene Verbindungen sichtbar und dadurch handhabbar. Spezielle Komponenten können hierauf reagieren und Maßnahmen einleiten.

Beobachtungs-Schnittstelle Die IMLAUF-Komponente bietet eine Schnittstelle auf Basis von der Subskription-/Benachrichtigungs-Kommunikation (Typ 3) an. Beliebige Komponenten können sich hier auf Ereignisse von ein-/ausgehenden Nachrichten subskribieren[2].

Das heißt, dass der Status der Nachrichten publiziert wird. Hierzu zählen insbesondere auch Fehler-Zustände. Hierdurch kann eine Komponente, wie beispielsweise das im späteren Verlauf beschrieben DGI eine Überwachung von Geräte realisieren. Ebenso ist aber denkbar, dass Reaktionen auf das Antwortverhalten (Zeitverzug) allgemein sichtbar gemacht werden.

Hintergrund:
Diese Abbildung der Kommunikation selber ist nicht verbreitet. ACPLT/KS bildet – in der Version KS2nd – für TCP/IP Verbindungen eintreffende Verbindungen als Objekte ab, sodass auch eintreffende Dienst-Aufrufe die aktuelle Kommunikationsumgebung erkunden können.
Bei OPC-UA hingegen wird die Kommunikation selber strikt von dem Objektraum getrennt. Zwar gibt es in der aktuellsten Version der Spezifikation „SessionDiagnosticsSummary“, jedoch ist diese dem Administrator vorbehalten und steht somit nicht für die Modellierung bzw. Programme, die aufgrund von den Informationen agieren, zur Verfügung.

5.4 Transparente Erkundung von Daten – Modell-Explorer

Als grundlegende Eigenschaften wurden sowohl das eigentliche Meta-Modell der Informationen auf einem Gerät, Modelle, die dieses Meta-Modell nutzen, wie auch Verbindungen der Modell-Instanzen zwischen Geräten dargestellt.

[2] Durch diese Schnittstelle hat jeder Dienst Zugriff auf alle Nachrichten. Eine Vertraulichkeit ist also nicht gegeben. Da die kollaborative Manipulation der Daten im Fokus steht, ist dieses im Normalfall zu vertreten. Beschränkungen und Zugriffsrechte, wie sie in OPC-UA vorhanden sind, lassen sich auch hier anwenden.

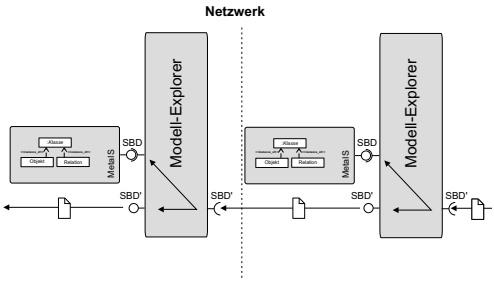


Abbildung 5.7: Nachrichten-basierte, transparente Kommunikation für die verteilte Modell-Erkundung

Um einen transparenten Zugriff – im Sinne der Kommunikation – zu den Informationen zu erhalten, bietet der Modell-Explorer eine Schnittstelle an, die auf der Schnittstelle der SBD beruht. Der Zugriff muss, da eine nicht-lokale Kommunikation benötigt wird, asynchron erfolgen, weswegen die Nachrichten-basierte Lösung hier genutzt wird, wie sie in Kapitel 4.6.2.6 vorgestellt wurde.

Hierdurch können Anfragen an das Modell unabhängig von der Lokalisierung des Objektes gestellt werden. D.h. erhält ein Modell-Explorer eine Nachricht, analysiert dieser als erstes, ob die lokale Instanzumgebung angesprochen wird. Sollte dieses der Fall sein, kann jeder Nachrichten-Typ auf einen SBD der Instanzumgebung abgebildet werden. Sollte eine entfernte Instanzumgebung angesprochen werden, wird der Inhalt der Nachricht an das Zielsystem weitergeleitet, dort bearbeitet und eine Antwort-Nachricht an den ersten Model-Explorer geschickt, sodass dieser die Antwort an den Anfragenden erzeugen kann.

Für dieses Weiterleiten müssen die ursprünglichen Nachrichten erst mal gespeichert werden. Bei der Weiterleitungs-Nachricht werden Typ und Inhalt beibehalten, jedoch die Empfangsadresse auf den aktuelle Modell-Explorer gesetzt, sodass die Antwort an diesen zurückkommt. Die IDs in den Köpfen der Nachrichten ermöglichen dabei die Zuordnung der Weiterleitungs-Antwort-Nachrichten zu den ursprünglichen Anfragen.

Handhabung von Ext.-Verbindungen

Gleichzeitig *löst* der Modell-Explorer die Ext.-Verbindungen auf. Das heißt, sollte eine Anfrage auf eine Ext.-Verbindung stoßen, wird diese verfolgt, der SBD'-Aufruf entsprechend abgeändert. Beispielsweise wird eine Liste Relations-Verbindungen, die auf einem entfernten System fortgeführt wird, als gemeinsames Ergebnis zurückgeliefert. Dabei werden unterschiedliche EPRs für die Elemente des Ergebnisses verwandt.
Auch das Anlegen von Relationen, welches unterschiedliche Systeme betrifft, wird durch den Modell-Explorer transparent geregelt. Hierfür muss der Modell-Explorer die entsprechenden Objekte der Ext.-Verbindung im Instanzsystem anlegen und parametrieren. Außerdem muss die Nachricht auch dem Modell-Explorer auf dem Zielsystem weitergeleitet werden, damit hier der zweite Teil der Ext.-Verbindung angelegt wird.

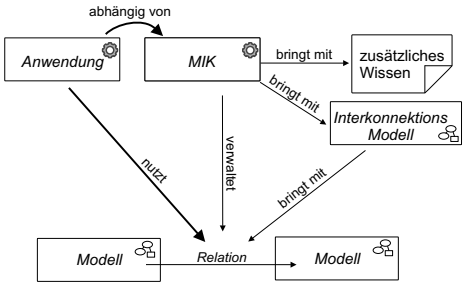


Abbildung 5.8: Konzept der Modell-Interkonnektion zur Laufzeit

Dienste oder Anwendungen, die also auf ein verteiltes Datenmodell zugreifen wollen, aber das Ext.-Verbindung-Konzept nicht selber beachten wollen, haben hier die Möglichkeit einen transparenten Zugriff zu erhalten. Lediglich die gelieferte Kommunikations-Adresse muss mit berücksichtigt - aber nicht weiter verarbeitet - werden.

Keine Änderungs-Benachrichtigungen zwischen Geräten

Änderungs-Benachrichtigungen, die nicht per Modell-Explorer angeboten werden, sind nur realistisch zu spezifizieren, wenn entweder übermäßig viele Kommunikations-Ressourcen bereitstehen oder eine effektive Möglichkeit der Eindämmung von Änderungs-Benachrichtigungen gefunden wurde. An dieser Stelle sollen die Änderungen im Modell deswegen nur lokal für Komponenten bereitstehen. Änderungen auf Kommunikationsebene - also hinzukommende oder weggefallene Kommunikationspartner selber - werden dafür durch andere Komponenten verwaltet, die im Folgenden beschrieben werden.

Insgesamt erfüllt der Modell-Explorer also die Aufgabe, die Verteilung der Daten auf unterschiedliche Instanzumgebungen zu abstrahieren - die eingesetzte EPR ist ein eindeutiger Bezeichner für ein Objekt, muss aber von der Anwendung nicht im Detail betrachtet werden. Zu seinen Aufgaben gehören also zum einen die Auflösung der EPR und das Ausführen der Befehle auf lokalen oder entfernten Systemen. Zum anderen auch die Abstraktion über die Verteilung selber, indem das Mittel der Ext.-Verbindung berücksichtigt wird.

5.5 Prinzip der Modell-Interkonnektions-Komponenten (MIK)

Im Kapitel 4.6.1 wurde das Konzept der Modell-Interkonnektionen vorgestellt. Es sorgt für Verbindungen von bisher unverbundenen Informationen zwischen Modell-Instanzen.

Um solche Verbindungen zur Laufzeit aufzubauen und aktuell zu halten, wird hier das Prinzip der Modell-Interkonnektions-Komponente (MIK) beschrieben.

Es handelt sich hierbei um eine bestimmte Gruppe von Diensten, die im eigentlichen Sinne selber keine Schnittstellen anbieten. Diese Modell-Interkonnektions-Komponenten beobachten die Instanzumgebung der Modelle. Sollten für sie relevante Veränderungen in den Modell-Instanzen auftreten, reagieren sie entsprechend ihres eigenen Modells.

Anwendungen sollen diese zusätzlichen Informationen, die in den Modell-Interkonnektions-Relationen vorhanden sind, nutzen. Damit hängen diese Anwendungen zum einen von den Modellen ab, die sie klassisch verarbeiten, zum anderen aber auch von den MIKs, da sie sich auf das Vorhandensein der Modell-Interkonnektions-Relationen verlassen. Abbildung 5.8 verdeutlicht das Prinzip der MIK zur Laufzeit.

Verteilte Modelle und das MIK Um die Modell-Instanzen in der verteilten Instanzumgebung zu finden, wurde bereits das Konzept der Modell-Master (Kapitel 4.5.1.1), sowie das damit verbundene Yellow-Paging vorgestellt. Dieses wird von den MIK genutzt, um Instanzen von für sie interessanten Modelle aufzufinden. Das Konzept der Ext.-Verbindungen wurde bereits bei den Basismodellen beschrieben, ebenso das generische Prinzip der MIKs.

Um Änderungen von allen Geräten erhalten zu können, müssen die MIK sich aktiv bei allen Geräten für Modell-Änderungen bei der dortigen Instanzumgebung anmelden, da nach dem vorherigen Kapitel Modell-Änderungen nicht generell verbreitet werden.

Für einen einfachen Fall, dass eine MIK zwei Modelle beobachtet und bei Bedarf Relationen anlegt, ist dieses handhabbar: Wenn die Modelle jeweils auf einem Gerät instanziiert sind und das MIK auch auf einem der Geräte instanziiert wurde, müssen nur die Modell-Änderungen des anderen Gerätes (mit Bezug auf das zweite Modell) übertragen werden.

Ein MIK sollte die Modell-Änderungen sowie Beobachtungen immer durch den Modell-Explorer vornehmen. Auf diese Weise werden Ext.-Verbindungen automatisch angelegt, falls diese nötig sind.

5.5.1 Problem: Schleifenbildung

Nicht nur durch die Modell-Interkonnektionen, sondern eigentlich immer, wenn unterschiedliche Komponenten Veränderungen an gemeinsamen Daten vornehmen, gilt es eine Schleifenbildung der Änderungen zu vermeiden.

Mit Schleifenbildung ist hier gemeint, dass eine Komponente A eine Aktion x ausführt. Diese Veränderung der Daten bedingt, dass Komponente B eine Aktion x^{-1} ausführt. Daraus folgt, dass Komponente A wiederum die Aktion x ausführt. Es entsteht also eine unendliche Schleife. Solche Schleifen können durchaus auch über mehrere Modelle und Aktionen entstehen.

Theoretische Lösungsskizze: Vorab-Analyse Theoretisch könnten solche potenziell auftauchenden Schleifen vorab untersucht werden. Es wird also aufgrund der Änderungen an den

Modellen und ihren Relationen untersucht, ob durch die Veränderungen ein Zyklus entstehen kann.

Hierfür sind allerdings die Modelle, Modell-Interkonnektions-Modelle und auch Modell-Transformationen (d.h. Änderungen der Modelle durch die Anwendungen) erforderlich. Da die Modelle durch weitere MIK voneinander abhängen können, sind auch diese notwendig zu untersuchen. Damit ergibt sich in der Praxis das Problem, dass im Endeffekt wieder ein „Welt-Modell“ benötigt wird, also ein einheitliches Verständnis von allen Informationen, die in den Modellen abgelegt sind.

Theoretische Lösungsskizze: Schleifenerkennung zur Laufzeit Eine weitere Lösung ist eine Schleifenerkennung zur Laufzeit. Dafür werden Änderungen am Modell protokolliert. Sollten gleiche Änderungen wiederholt auftreten, werden diese erkannt. Hierfür können existierende Lösungen aus dem Themengebiet der Termersetzungssysteme angewendet werden [Ohl02].

Praktisch gesehen... In der Praxis sollten solche Schleifen jedoch kaum eine Rolle spielen, denn die Modelle sind in sich geschlossen. Die Modell-Interkonnektionen führen Verbindungen zwischen Modellen ein, die eine zusätzliche Information einbringen, d.h. sie beeinflussen selber nicht die unterliegenden Modelle. Auch ist es unwahrscheinlich, dass unterschiedliche Modell-Interkonnektionen gegenseitig Schleifen beschreiben, denn sie sollten eigene Aufgaben und damit einen jeweiligen Mehrwert erbringen.

Trotzdem besteht die Gefahr der Schleifenbildung - zumindest in der Theorie.

5.5.2 Beispiel: MIK AT-Geräte-Dienste

Diese Modell-Interkonnektions-Komponente verbindet die Dienstrepräsentationen mit den entsprechenden AT-Geräte-Komponenten.

Durch die entstehenden Verbindungen können aufbauende Anwendungen erkunden, welche Dienste auf einem Gerät laufen. Hierdurch können beispielsweise Dienste gefunden werden, die möglichst nahe sind. Ebenso können Dienste gefunden werden, die auf Geräten mit vielen freien Ressourcen laufen.

5.6 Überwachung der Umgebung - Remote-Model-Inspektor (RMI)

Mit den oben beschriebenen Komponenten können die Elemente der Modelle miteinander verbunden werden. Dabei kann eine Erkundung über die Systemgrenzen hinweg transparent erfolgen, wenn die Kommunikation über den Model-Explorer erfolgt. Lokale Änderungen sind direkt erkennbar, indem die Subskription/Benachrichtigungs-Schnittstelle *Änderungen* der Instanzumgebung genutzt wird.

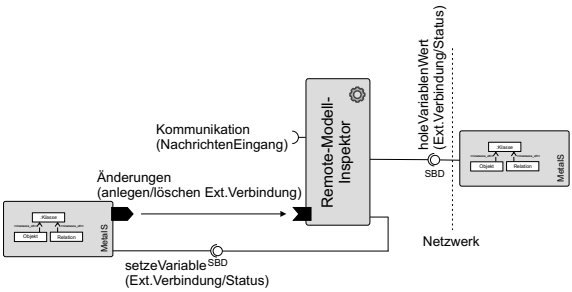


Abbildung 5.9: Überwachung der Ext.-Verbindungen durch den RMI

Als eine Aufgabe wurde in den Anforderungen zusätzlich die Dynamik beschrieben. Hierzu gehören neben dem Hinzukommen von neuen Geräten auch der geplante oder ungeplante Ausfall von Geräten. Dieser wird ebenso nutzbar gemacht, wie andere strukturelle Änderungen im Modell.

Auf Ebene der Instanzumgebung sind Ext.-Verbindungen ein Indikator dafür, dass ein Interesse von einer lokalen Komponente an dem Fortbestand des Zielsystems liegt. Die beschriebene Komponente *Remote-Model-Inspektor* existiert als Basis-Komponente auf jedem Gerät.

Sie geht von allen Instanzen der Ext.-Verbindungen auf ihrem Gerät aus. Sie bildet aus der Liste der Ext.-Verbindungs-Objekte eine Geräteliste der zu überwachenden Geräte, indem sie lokale Repräsentationen für die Geräte anlegt. Alle diese Geräte werden überwacht.

Durch die Zusammenfassung in einer Komponente werden zwei Ziele erreicht:

- Die nötige Kommunikation wird minimiert: Zum einen wird anderweitige Kommunikation berücksichtigt, zum anderen wird ein Gerät auch nur einmal überwacht.
- Die interessierten Komponenten müssen nicht selber eine Überwachung realisieren.

Dieses Konzept hat also nur etwas mit der Erkennung einer Änderung zu tun. Es bildet in keiner Weise einen Schutz vor Datenverlust.

Die RMI Komponente wird durch das MsgSys von eintreffenden Nachrichten unterrichtet, wie in Abbildung 5.9 dargestellt. Berücksichtigt man, dass Geräte nur insgesamt ausfallen (nicht jedoch Software-Komponenten auf ihnen, wie in Kapitel 5.3 beschrieben) kann der RMI aus eintreffenden Nachrichten schließen, dass die Ext.-Verbindung-Instanzen, die auf den Absender zeigen, weiterhin gültig sind. Der RMI kann also in diesem Fall die Aktualitäts-Zeit aller Ext.-Verbindungs-Instanzen auf die aktuelle Uhrzeit setzen.
Dieses stellt eine passive Überwachung dar. Sie erzeugt keinerlei Kommunikationslast. Dieses hat jedoch den Nachteil, dass keine Aussagen über die Erreichbarkeit der Ext.-Verbindungs-Zielsysteme gemacht werden können, wenn nicht „zufällig“ eine Kommunikation stattfindet.

Deshalb können Komponenten die Ext.-Verbindungs-Instanzen mit einer Annotation *Überwachungs-Intervall* ausstatten. In diesem Fall überprüft der RMI die Existenz des Ziels im entsprechenden Zeitrahmen. Dafür wird durch einen Aufruf des SBD' zum Holen einer Status-Variablen der Ext.-Verbindungs-Instanz auf der entfernten Instanzumgebung durchgeführt. Hierbei handelt es sich also um eine qualifizierte, aktive Überwachung.

Die folgende Komponente Dienst/Geräte-Inspektor nutzt dieses Konzept um eine allgemeine Überwachungsfunktion von Diensten und Geräten anzubieten.

5.7 Verwaltung der Dienste und Geräte - Dienst/Geräte-Inspektor (DGI)

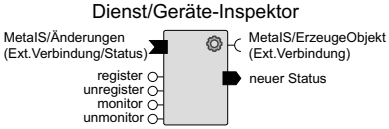


Abbildung 5.10: Schnittstelle des Dienst/Geräte-Inspektor

Der DGI stellt die zentrale Schnittstelle zwischen den Diensten, sowie der modellgetriebenen Instanzumgebung dar. Er übernimmt dabei mehrere Aufgaben. Seine Schnittstellen sind in Abbildung 5.10 zusammengefasst.

Zum einen übernimmt er die Rolle einer lokalen, zentralen Anmeldestelle für Dienste, d.h. Dienste registrieren sich selber bei dieser lokalen DGI Komponente. Diese übernimmt einige Arbeitsschritte, um den Dienst ins Gesamtsystem einzubinden. Hierzu zählen insbesondere das Anlegen der Dienst-Repräsentation im Dienst-Modell der Instanzumgebung.

Der zuvor dokumentierte RMI überwacht den Zustand aller Ext.-Verbindungs-Objekte in der lokalen Instanzumgebung in Bezug auf deren zugehörigen Ext.-Verbindungs-Objekte auf den entfernten Systemen. Damit auch eine Änderung der Dienste und Geräte erkannt werden kann, sorgt der DGI für eine entsprechende Überwachung dieser Komponenten.
Deren Aufgabe ist es also, auf Anfrage von Diensten Änderung an der Dienstverteilung an die anfragenden Dienste weiterzuleiten. Auf diese Weise können die Dienste auf Strukturänderungen reagieren ohne selber eine Überwachung zu realisieren. Dienste müssen lediglich für sie interessante „Partner“ bei dem DGI anmelden. Änderungen beziehen sich dabei - wie auch bei der Ext.-Verbindungs-Überwachung des RMI - auf den Zustand, wie er in Kapitel 4.2.1 definiert ist. Damit wird auch klar, dass alle aktiven Komponenten bzw. deren Repräsentationen, überwacht werden können.
Implizit nimmt der DGI dabei an, dass ein einmal instanziierter Dienst immer auf einem Gerät läuft. Wenn also ein Interesse an einem Dienst besteht, zieht so ein Ausfall des zugehörigen Gerätes auch eine entsprechende Benachrichtigung nach sich.

Zum Erfüllen der Rolle der lokalen, zentralen Anmeldestelle wird vom DGI die lokale Schnittstelle *register()* angeboten. Nutzt ein startender Dienst diese, wird seine Repräsentation „Self“ angelegt. Gleichzeitig wird er mit dem Modell-Master des Dienst-Modells verbunden, sodass er auffindbar (Kapitel 4.4) ist.

Sollte ein Dienst gelöscht (*deregister()*) werden, wird die Repräsentation entsprechend gelöscht. Durch die Änderungsmitteilung werden auch andere Komponenten über diese Änderung informiert - damit also auch die Kommunikations-Komponente, sodass für diesen Dienst keine Mitteilungen mehr entgegen genommen werden.

Ein Dienst, der eine Abhängigkeit zu einem anderen Dienst oder Gerät hat, kann diese durch den DGI überwachen lassen. Dafür nutzt er die *monitor()*-Schnittstelle. Bei Aufruf wird der DGI eine entsprechende Zuordnung von dem abhängigen Dienst zu dem zu überwachenden Dienst im Daten-Modell per *interesse_an*-Relation ablegen. Falls es sich um eine RemoteLink-Verbindung zu einem externen Dienst handelt, wird er dabei auch ein entsprechendes Überwachungs-Intervall definieren. Der RMI sorgt so wieder für eine entsprechende Überwachung des Status. Durch die Schnittstelle *unmonitor()* wird die entsprechende Verbindung wieder aufgehoben.

Sollte sich der Status eines Ext.-Verbindungs-Objektes ändern, informiert der DGI entweder über eine Aufruf- oder Nachrichten-orientiert Schnittstelle *neuerStatus* den abhängigen Dienst.

5.8 Migration von traditioneller Datenhaltung zur Repräsentation der Information in einer verteilten, modellgetriebenen Instanzumgebung

Alle vorgestellten Komponenten sowie Konzepte basieren auf der Annahme, dass Anwendungen und deren Hersteller bereit sind, ihr properitätren Datenablagen aufzugeben, damit die Anwendungs-Daten in der Instanzumgebung abgelegt werden können.

Um eine solche Änderung der Datenhaltung zu ermöglichen, ist ein Migrationspfad unverzichtbar. Insbesondere stellt sich bei Änderungen dieser Größenordnung immer die Frage, ab wann sich die Änderungen für die Beteiligten auszahlen.

Durch die Verbindungen von den einzelnen Informationen ergibt sich die Möglichkeit, dass selbst der Datenaustausch zwischen zwei Anwendungen, der bisher klassisch über Export-Import-Funktionen realisiert wurde, profitieren kann. Wenn beide Anwendungen ihre bisherigen Strukturen in die gemeinsame Modell-Instanzumgebung ablegen und entsprechende Relationen zwischen den Einzel-Informationen schaffen, ist der Benefit sofort erkennbar: Eine doppelte Datenhaltung und damit Inkonsistenzen werden vermieden.

Die nötigen Änderungen an den internen Programmstrukturen sollten – je nach Realisierung – nicht allzu komplex sein. Im Prinzip muss eine Abbildung des bisherigen Modells auf das Meta-Modell passieren, welches von der Instanzumgebung bereitgestellt wird.

Wenn der Zugriff auf das Meta-Modell offen gestaltet wird, können weiter Anwendungen direkt ihre Daten ablegen und Relationen zu existierenden Informationen setzen, um damit neue Informationen auch selber auszunutzen. Ein Benefit ist also unmittelbar erkennbar.

Für die Umsetzung einer gemeinsamen, modellgetriebenen Instanzumgebung kann ein Konsortium gegründet werden, wie es das für die Kommunikationssysteme in der Automatisierungstechnik schon länger gibt. Es ermöglicht allen Interessierten sich einzubringen und erhöht damit die Akzeptanz des Gesamtansatzes. Ansätze für solche Kooperationen von Experten sind bereits vorhanden. Da in der Arbeit auch beschrieben, seien hier beispielhaft PLCopen, AutomationML und die OPC Foundation genannt.

5.9 Prototypen der Komponenten

Die beschriebenen Komponenten wurden größtenteils prototypisch realisiert. Die Implementierung fokussierte dabei auf die Aufgabenstellungen, die durch öffentliche sowie industrielle Forschungsprojekte am Lehrtuhl für Prozessleittechnik vorhanden waren.

Technologisch wurde dabei ACPLT (Kapitel 2.4.1.1) gewählt. Die Verfügbarkeit sowie Offenheit waren ein ausschlaggebender Grund für diese Entscheidung. Zusätzlich konnten so grundlegende Kommunikations-Implementierungen frühzeitig auch in Projekten eingesetzt werden, die mit den hier beschriebenen Komponenten ansonsten wenig zu tun hatten.

Die folgenden Komponenten wurden prototypisch realisiert.

KS2nd Die ursprüngliche Kommunikation ACPLT/KS bietet eine Schnittstelle um eine Instanzumgebung zu beherbergen. Die Kommunikation selber ist also nicht als Objekte in der Instanzumgebung modelliert, da es eine unterliegende Schicht war. KS2nd bietet die Möglichkeit, Objekte in der Instanzumgebung anzulegen, zu parametrieren und hierdurch Kommunikation (sowohl eingehende, wie auch ausgehende) zu nutzen.

MsgSys (Kapitel 5.3) Die Nachrichten-basierte Kommunikation mit ihren Verwaltungskomponenten auf Basis von KS2nd ermöglicht es Anwendungen das Konzept zu nutzen. Es hat sich gerade in einem verteilten Umfeld zur Reaktion auf Geräte-Struktur-Änderungen bewährt.

Modell-Explorer (Kapitel 5.4) Modell-Erkundungen können mittels des Modell-Explorers über Instanzumgebungen hinweg ohne Berücksichtigung der Verteilung erfolgen. Dafür wurden auch das Konzept der entfernten Verbindungen (Kapitel 4.6.2.1) ansatzweise realisiert.

RMI (Kapitel 5.6) Der Remote-Model-Inspektor kann bisher die Überwachung von konkreten Datensätzen (Konfigurationsdaten) auf lokalen und entfernten Geräten vornehmen.

DGI (Kapitel 5.7) Zur Geräteüberwachung wurde der Dienst/Geräte-Inspektor teilweise realisiert. Hierbei wird ein Ausfall eines Gerätes erkannt, sodass Reaktionen zur Neustrukturierung erfolgen.

Die Prototypen wurden im Rahmen von Forschungsaufträgen an realen Demoanlagen in Betrieb genommen, womit die Einsatzfähigkeit sowohl der Konzepte, wie auch der Realisierung überprüft wurde. Die Ergebnisse flossen in die Veröffentlichungen [ME12], [ME11], [MKE10] und [MKE09] ein.

6 Anwendungen: Dynamik auf Basis der verteilten, modellgetriebenen Ausführungsumgebung

A distributed system is a collection of independent computers that appears to its users as a single coherent system.
Andrew S. Tanenbaum

Um die Anwendbarkeit, sowie die Mächtigkeit der verteilten, modellgetriebenen Instanzumgebung zu demonstrieren, werden in diesem Kapitel unterschiedliche Anwendungen aufgezeigt. Ziel ist es, nicht nur abstrakt die Vorteile im Raum stehen zu lassen, sondern ganz konkret greifbar zu machen.

Als erstes wird beschrieben, wie eine Anwendung zur allgemeinen Suche nach Strukturen realisiert werden kann. Dieses ist eine der wichtigsten Funktionen um eine Dynamik im Gesamtsystem realisieren zu können. Hierdurch wird die Möglichkeit geschaffen, automatisch oder semi-automatisch eine Verknüpfung von Informationen und Diensten zu erreichen und somit auf Änderungen reagieren zu können.
Folgend wird eine Anwendung zur Verlagerung von Software-Komponenten beschrieben. Dieses verdeutlicht die Dynamik des Gesamtsystems, nicht nur in Bezug auf die Struktur der Anlage (Reaktion auf Umbauten), sondern auch die Möglichkeiten, unterschiedliche Verteilungen nachträglich zu ändern - beispielsweise das in Kapitel 4.7.2 beschrieben Umverteilen zwischen Engineering- und Produktions-Phase.
Diese beiden Anwendungen sind aufbauend auf der Instanzumgebung zu sehen und nicht Bestandteil dieser.

Darauf folgende Anwendungsfälle existieren als alltägliche Aufgabe in der Anlagenplanung, sowie Inbetriebsetzung. Hier soll möglichst beispielhaft ein breites Spektrum abgedeckt werden, welches Lösungen für heutzutage alltägliche Probleme beschreibt.

6.1 Verteilung in der Automatisierungstechnik als Suche

Durch die Verteilung der Komponenten entsteht das Problem des Auffindens. Allgemein gesehen ist dies kein neues Problem. Es wird durch die Verteilung auf Geräte nur deutlicher. Auch in klassischen Anlagenarchitekturen ist es immer wieder schwierig festzustellen, an welchem

Port einer I/O Karte welches Gerät verbunden ist und - viel wichtiger - wie die Zuordnung im Prozess (also zu der PLT-Stelle) zu sehen ist.

Da das modellgetriebene Instanzsystem aktiv mit der Verteilung umgeht, können solche Fragen von Anwendungen durch Wissen von den entsprechenden Modellen beantwortet werden.

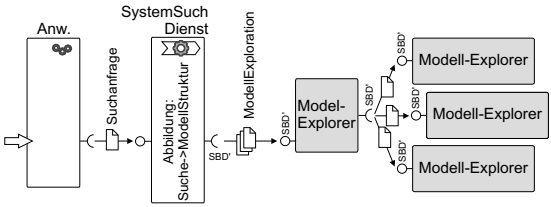


Abbildung 6.1: Der Suchdienst übernimmt die Auswertung einer Suchanfrage auf Basis einer Strukturbeschreibungs-Sprache, die auf dem Meta-Modell basiert.

Eine Anwendung sucht einen Dienst. Hierfür wendet sich der suchende Dienst mit einer Suchanfrage an einen *SystemSuchDienst*. Im Sinne der Kommunikation handelt es sich dabei um eine asynchrone Kommunikation; formuliert in der Schnittstellensprache des SystemSuchDienstes. Für die Formulierung der Suchanfrage benötigt die Anwendung das Wissen des Modells über das sie sucht.

Durch die Möglichkeit eines gemeinsamen Meta-Modells lassen sich alle Anfragen in einer Sprache, die auf dem Meta-Modell aufbaut, formulieren, d.h. die Ausführung einer Suche ist nicht mehr von den konkreten Modellen abhängig und kann von einem *SystemSuchDienst* für alle Modelle ausgeführt werden.

Hintergrund:
Die Trennung ist äquivalent zu Datenbank-Sprachen zu sehen: Datenbank-Sprachen wie SQL sind unabhängig vom Datenbankschema formuliert. Ihre Aufrufe enthalten entsprechende Konstrukte, um die zu lesenden/schreibenden Daten zu formulieren.

In Abbildung 6.1 ist der Ablauf einer Such-Anfrage dargestellt. Durch den lokalen SystemSuchDienst kann ein Dienst eine Suchanfrage bearbeiten lassen, ohne dass sich ein anfragender Dienst überhaupt um die Auswertung der Anfrage oder die Verteilung des Modells selber kümmern muss. Dafür übernimmt der SystemSuchDienst die Auswertung der Anfrage und stellt entsprechende Modell-Explorations-Anfragen an den lokalen Modell-Explorer, der ggf. diese Anfragen verteilt bearbeitet. Während die Formulierung der Suchanfragen eine eigene Sprache darstellt, ist die Kommunikation zu dem Modell-Explorer in den SystemBasisDiensten' (SBD') formuliert.

Formulierung der Suchanfragen

Somit wird die Suchanfrage also in einer Form der Strukturbeschreibung mit folgenden Elementen bestehen

Klassen und Relationen sind die Hauptbeschreibungsmerkmale der Suchanfrage, sie beschreiben die Grundstruktur des Zielsystems.

Variablen-Werte können als Bedingungen vorgegeben werden (z.B. "Variable $PV <= 4.2$").

Referenzen auf Instanzen werden insbesondere als absolute Ausgangspunkte verwendet.

Dabei ist es wichtig zu verstehen, dass diese Strukturbeschreibung zwar von den Modellbeschreibungen eingegrenzt wird, diese jedoch in keiner Weise eine valide (Teil-)Struktur des oder der Modelle selber sind. Sie stellen vielmehr ein Muster dar, was durch den SystemSuch-Dienst gefunden werden soll.

Hintergrund:
In der IT-Welt existieren vergleichbare Sprachen zur Suche in XML Daten. Durch XPATH [32] werden die Elemente, Kindelemente und Werte/Namen beschrieben. Ein entsprechender Prozessor liefert eine Liste aller XML Elemente mit den formulierten Eigenschaften innerhalb eines XML Dokumentes.

Beispielanfragen

Die folgenden Beispiele sind natürlich-sprachlich formuliert und sollen Ideen für Suchanfragen geben, aber auch die Mächtigkeit der Such-Anfrage-Sprache aufzeigen.

Welche Ventile sind offen? Suche über Anlagenstruktur-Modell, Aktualwerte per Relationen in einem Steuerungs-Modell.

Wie hoch ist Temperatur in B1? Suche über Anlagenstruktur-Modell nach Temperatursensoren von B1, dann Dienst-Modell für den Sensor um dort per AT-Geräte-Struktur die EPR abzufragen und durch den Dienst den Aktualwert zu ermitteln.

Welche Geräte haben Ihre MTBF erreicht? Suche über AT-Geräte-Struktur oder der Anlagenstruktur, dann der Annotationen der Dienst-Repräsentationen für die Geräte.

6.2 Migration von Komponenten

Die Migration von Komponenten wurde in unterschiedlichen Zusammenhängen angesprochen. An dieser Stelle wird sie als Anwendung für die modellgetriebene, verteilte Instanzumgebung beschrieben.

Definition: *Eine **Migration** ist, eine Komponente durch eine andere zu ersetzen. Dabei ist klar definiert, welche internen Informationen übernommen werden und welche durch die Migration verloren gehen.*

Es geht dabei um die Verschiebung von Komponenten. Dabei werden an dieser Stelle insbesondere zwei Szenarien betrachtet:

Örtliche Migration Die Verlagerung einer Komponente von einem Gerät auf ein anderes. Beispielsweise zur Optimierung der Modell-Verteilung.

Versions-Migration Versionsupgrade einer Komponente unter Beibehaltung ihrer Eigenschaften / Konfiguration. Beispielsweise als Software-Update.

In beiden Fällen soll eine Instanziierung einer Komponente erfolgen, die die Aufgaben und damit die Konfiguration in Form von Variablen und Relationen einer Vorgänger-Komponente übernimmt. Vorteil einer solchen Anwendung ist insbesondere, dass Unterbrechungen des gesamten Systems minimiert werden. Bei bisherigen Systemen ist eine Migration von existierenden Instanzen nicht realisiert.

Einige Voraussetzungen für eine Migration in der modellgetriebenen Instanzumgebung müssen erfüllt sein:

- Die Klassenbeschreibungen der Instanzen müssen auf den beteiligten Geräten vorhanden sein.
- Eine Definition der zu übernehmenden Variablen und Annotationen ist bereitzustellen.
- Die Zustandsmaschine der Komponenten bietet einen Zustand zur Konfiguration an. Dieser Zustand entkoppelt die Instanz vom restlichen System insoweit, dass die Komponente konfiguriert werden kann, ohne dass Zugriffe / Nutzungen erfolgen. In der vorgeschlagenen Zustandsmaschine aus Kapitel 4.2.1 kann das der Zustand *laden* sein.

Der Ablauf einer Migration ist nun:

1. Instanziierung der neuen Instanz. Sie verbleibt in dem Konfigurations-Zustand.
2. Versetzen der alten Instanz in den Konfigurationszustand.
3. Kopieren der Konfiguration (statische und interne Variablen; vgl. Kapitel 4.4) und Annotationen.
4. Löschen der Relationen zur alten Instanz mit gleichzeitigem Anlegen der Relationen zur neuen Instanz.
5. Aktivieren der neuen Instanz (vom Konfigurationszustand in den normalen Zustand).
6. Löschen der alten Instanz.

6.3 Anwendungsfall 1: Abbildung einer Remote I/O

Um eine bestmögliche Integration der Laufzeit-Modelle zu schaffen, ist es sinnvoll auch die Anbindung an den Prozess und damit der Aktualwerte im Modell-Raum abzubilden.

Dafür wird hier ein Dienst beschrieben, der Aktualwerte in die modellgetriebene Instanzumgebung abbildet, welche beispielsweise von einem Sensor erfasst werden.

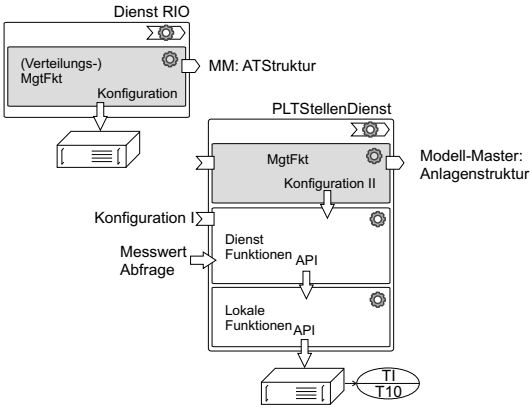


Abbildung 6.2: Messwerte für die Instanzumgebung

Ausgegangen wird von einem Gerät, welches die Funktionen einer Remote I/O übernimmt, also einzelne (Mess-)Werte auf einem normalerweise digitalen Bus- oder Netz-werk-System bereitstellt.

Ein solches Gerät ist bei diesem Beispiel das „unterste“ Gerät, welches in der Lage ist, die modellgetriebene Instanzumgebung auszuführen. Entsprechend werden die einzelnen Signale, die an der Remote I/O angeschlossen sind, als physikalisch verfügbare, lokale Ressourcen angesehen.

Als erstes wird ein Dienst vorgesehen, der die Remote I/O selber abbildet. Er meldet sich - wie im AT-Geräte-Struktur-Modell vorgesehen - am Modell-Master an und wird in die flache Struktur des Modells integriert. Fortan ist die Remote I/O im Dienstsystem als Ressource bekannt und kann verwendet werden.

Aufbauend wird pro Kanal an der Remote I/O ein Dienst vom Typ *PLTStellenDienst* verwendet. Dieser Dienst, der z.B. bei einer Hardware-Konfiguration instanziiert werden kann, stellt die Brücke zwischen Messsignal und Instanzumgebung dar. Dieser Dienst sollte entsprechend bekannt gemacht werden. Am naheliegensten ist es, dass er in der Anlagenstruktur, z.B. also am PandIX-Modell mit einer entsprechenden Relation zu einem *PLCPoint* verbunden wird. Dieses wäre die Aufgabe eines MIK, welches als Zusatzinformationen die Relation PLT-Stelle

im Prozess zu Gerät kennt - wie sie in Kapitel 4.6.1.1 beschrieben ist. Anwendungen können so die EPR dieses Dienstes entdecken und auf die PLT-Stelle und damit den entsprechenden Wert zugreifen.

Konfiguration

Neben diesen Schnittstellen ist es naheliegend, auch eine Konfigurations-Schnitt-stelle für die Geräte anzubieten. Idealerweise werden existierende Modelle zur Gerätekonfiguration wie FDI / FDTv2 in die modellgetriebene Instanzsystem integriert und die Konfiguration der Sensoren kann durch Verbindung mit den entsprechenden Informationen der Modellinstanzen vorgenommen werden (Konfiguration I in der Abbildung 6.2).

Wenn eine solche Schnittstelle angeboten wird, kann diese zur Konfiguration z.B. eines speziellen Dienstes genutzt werden. Es kann aber auch als Management-Funktion angesehen werden. Dabei können die Konfigurationsdaten mit der Anlagenstruktur verknüpft abgelegt und von hier bezogen werden (Konfiguration II in der Abbildung 6.2).

Ziel

Wenn die Aktualwerte der Sensorik und die Stellwerte der Aktoren im Modell verknüpft abgelegt werden, können diese über weitere Modelle in Relation gesetzt werden. Dadurch ergeben sich vielfältige Diagnose und Überwachungsmöglichkeiten, insbesondere können auch Abfragen formuliert werden, die nicht vorprojektiert sind.

Als Beispiel kann hier die „OPC UA Information Model for IEC 61131-3“ [27] dienen, die die Steuerungsbausteine einer 61131-3 Programmierung mit Aktualwerten der Ein-/Ausgänge in dem OPC-UA Objektraum abbildet.

6.4 Anwendungsfall 2: Vorbereitung auf Ausfälle

In der zuvor beschriebenen Architektur wurde durch das Ext.-Verbindungs-Konzept eine Möglichkeit geschaffen, die Komponenten über Änderungen der Struktur - ob geplant oder nicht - zu informieren. Es macht Sinn, dass bei einer Veränderung nicht jede Komponente informiert wird, sondern nur die Komponenten, die auch reagieren müssen. Diese subskribieren sich nach dem Ext.-Verbindungs-Konzept mit dem entsprechenden Gegenüber und bekommen die Änderung per Benachrichtigung mit, wie in Kapitel 4.6.2.1 beschrieben.

Um in angemessener Zeit und mit gewünschter Zuverlässigkeit Reaktionen ausführen zu können, bedarf es ggf. einer Vorbereitung in Form eines Vorgehens-Konzeptes. Hier wird ein grundsätzliches Konzept für die Management-Funktionen des Dienstes beschrieben, um diesen mit einem *Redundanz-Ansatz* auszustatten. Dargestellt ist das Szenario in Abbildung 6.3. Ausgangspunkt ist ein Dienst, der ein regelmäßiges Interesse an einem anderen Dienst hat – potenziell auf einem anderen Gerät. Deren Dienst-Repräsentationen sind durch eine Relation

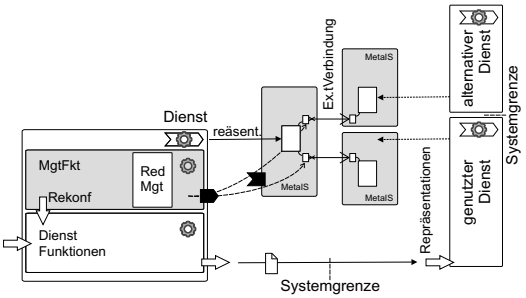


Abbildung 6.3: Vorbereitung auf eine Umschaltung aus Sicht der Managementfunktionen

- also ggf. auch eine Ext.-Verbindung - miteinander verknüpft. Hierdurch ist eine Überwachung durch die DGI-Komponente gegeben.

Eine zusätzliche, und von den restlichen Aufgaben der Management-Funktionen des Dienstes unabhängige, Redundanz-Komponente sorgt dafür, dass der Status der Ext.-Verbindung überwacht wird. Wird hier bemerkt, dass der Zieldienst nicht weiter erreichbar ist, kann auf einen alternativen Dienst umgeschaltet werden.

Die Auswahl des alternativen Dienstes ist dabei durch den Dienst selber gegeben und vorab vorbereitet: Die Suche nach Kommunikationspartnern im Bereich der Orchestrierungsaufgabe der Management-Komponente kann sogar im Voraus eine Suche nach einem alternativen Kommunikations-Partner starten und dessen EPR vorhalten.

Es besteht auch die Möglichkeit eine vorzeitige Reservierung der Ressourcen im zweiten Zieldienst vorzunehmen, sodass die Umschaltung schneller ist. Dafür muss die Zielanwendung jedoch „Kenntnis“ von dem Redundanz-Konzept haben in dem Sinne, dass eine Reservierung von Ressourcen, ohne diese zu nutzen, erfolgen kann.

6.5 Anwendungsfall 3: Adressierung durch PLT-Stelle

Aufbauend auf der Kommunikation kann ein weitergehender, höherwertiger Kommunikationsdienst realisiert werden, der verdeutlicht, welches Potenzial in der Interkonnektion von Modellen steckt (Abbildung 6.4).

In Kapitel 4.6.1.1 wurde bereits die prinzipielle Verknüpfung von dem im Kapitel 4.1 vorgestellten AT-Geräte-Struktur-Modell mit einem Anlagen-Struktur-Modell, wie z.B. das CAEX-basierte PandIX (vgl. Kapitel 2.2), dargestellt.

Wenn diese Interkonnektion als gegeben angesehen wird, kann ein Dienst *PLTStellenKommunikation* definiert werden, der hierauf aufbaut. Er nimmt Nachrichten mit einer Adressierung an

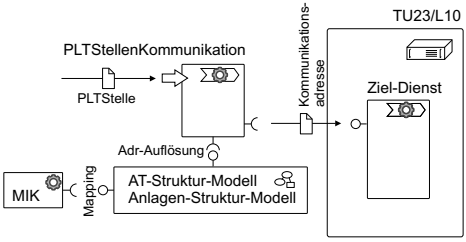


Abbildung 6.4: Schematischer Ablauf der Auflösung von PLT-Stelle zu Kommunikations-Adresse

die PLT-Stelle (also z.B. *TU10.T12*) entgegen und löst diese durch die verbundenen Modelle in Kommunikations-Adressen zu dem PLTStellenDienst auf.

Der Vorteil eines solchen Dienstes liegt auf der Hand: Dienste oder ganze Anwendungen können, basierend auf der Anlagenstruktur, Ihre Kommunikation projektieren - die Zustellung der Nachrichten wird von dem beschriebenen Dienst übernommen.

Hiermit ist es zum Beispiel möglich die Kommunikation auch schon zu planen und zu projektieren, wenn die AT-Struktur noch nicht installiert bzw. geplant ist. Die PLT-Stellen-Namen stehen in der Anlagenplanung schon fest und können so bereits verwandt werden.

6.6 Anwendungsfall 4: IEC61131-3-Programmierung im Modell

In der Konzeption wurde ein Eingriff der verteilten Instanzumgebung in die Steuerung selber ausgeschlossen, da traditionell eine Steuerungsprogrammierung keine wirkliche Verteilung adressiert. Wenn mehrere SPSen gemeinsam eine Anlage steuern, haben die einzelnen Geräte einzelne Teilaufgaben und kommunizieren explizit.

Im Verständnis der verteilten Instanzumgebung stellt die Steuerung im Sinne der IEC 61131-3, ausgeführt auf einer SPS weiterhin - in erster Linie - eine monolithische Komponente dar. Im Folgenden werden Integrationsansätze gezeigt, wie eine Interaktion möglich ist, ohne die bewährten Konzepte und Programmierparadigmen der SPS-Programmierung zu verlassen.

Grundsätzliches Ein Steuerungsprogramm wird beim Engineering erstellt und wird traditionell auf eine SPS übertragen - meistens, nachdem ein Kompiliervorgang eine Programmiersprache der IEC 61131-3 in ein Hardware-nahes Kompilat der SPS übersetzt hat (Abbildung 6.5). Diese Programmierung wird dann auf der SPS fortwährend ausgeführt. Für eine Änderung der Steuerung muss diese meist angehalten werden. Ausgenommen sind sogenannte „Live-Updates“ / „Online-Changes“, welche aber Hersteller-abhängig nur bestimmte Veränderungen des Steuerungsprogramms zulassen.

Zugriff auf das Modell der Steuerung Im Sinne der modellgetriebenen Instanzumgebung kann eine Steuerung selber z.B. als Instanz des in Kapitel 2.2 beschriebenen PLCopen Formates abgebildet werden.

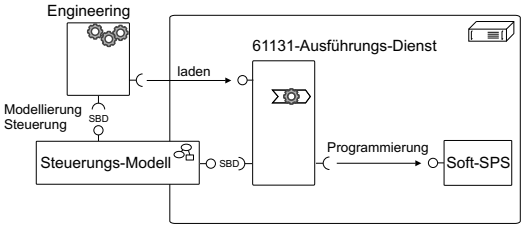


Abbildung 6.5: Schematischer Ablauf der modellgestützten Steuerungsprogrammierung

Legt ein Engineering System diese Steuerung in der modellgetriebenen Instanzumgebung ab, kann es durch eine Relation *ausgeführt_auf* eine Assoziation zu einem 61131-Ausführungs-Dienst aufbauen. Dieser existiert für jede beteiligte SPS genau einmal, sodass eine eindeutige Zuordnung gegeben ist. Der 61131-Ausführungs-Dienst wird durch das Engineering informiert, sobald eine neue Version der Steuerung auf die Soft-SPS geladen werden soll.

Eine dynamische Erkennung einer Änderung mit sofortiger Inbetriebnahme der neuen Steuerung verbietet sich, da beim Programmieren zwischenzeitlich Inkonsistenzen auftauchen. Das heißt auch, dass eine Beobachtung der Modellstruktur an dieser Stelle nicht sinnvoll ist - das Engineering muss zu einem diskreten Zeitpunkt einen Befehl geben, die Änderungen zu übernehmen.

Kommunikation zwischen SPS & modellgetriebener Instanzumgebung Betrachtet man das Vorausgegangene, bleibt also die SPS. Sie verarbeitet ein Programm in einer IEC 61131-3 Sprache und führt dieses ab einem diskreten Zeitpunkt ggf. nach einer Kompilierung aus.

Trotzdem kann es für zukünftige Anwendungen sinnvoll sein, Zugriff auf die Informationen, die in der modellgetriebenen Instanzumgebung abgelegt sind, zu ermöglichen.

Einen Vorschlag für die Kommunikation zwischen einer solchen Umgebung mit einer SPS ist in [ME12] beschrieben. Von diesem Punkt ausgehend, können Konstrukte innerhalb der SPS geschaffen werden, um auf die Modell-Instanzen zuzugreifen zu können. Die vorliegende, modellgetriebene Instanzumgebung abstrahiert von der Verteilung, sodass die SPS Programmierung im Endeffekt nur entsprechende Nachrichten an den Modell-Explorer stellt, um auf alle Modelle und Informationen Zugriff zu haben.

6.6.1 Probleme der konsequenten Umsetzung

Naheliegend wäre in einer konsequenten Umsetzung auf das Kompilat der Programmierung in Form einer 61131-3-Sprache zu verzichten und eine komplette Modell-Repräsentation der Pro-

grammierung anzustreben. Die Bausteine und Konstrukte eines Steuerungsprogramms würden so als einzelne Elemente eines „61131-Modells“ im Modell-Raum abgelegt und damit auch ein Abfrage ermöglichen.

Hierdurch könnten Dienste aus dem Modell heraus die Kontrolle über den Prozess übernehmen und direkten Zugriff auf Sensoren und Aktoren erhalten. Im Extremfall könnte man Funktionsbausteine als einzelne Dienste, die miteinander kommunizieren, modellieren und so eine Steuerung darstellen.

Gegeben wäre so die komplette und nahtlose Integration der Steuerung in die Modelllandschaft, wobei der Zugriff auf sämtliche Informationen der modellgetriebenen Instanzumgebung als Vorteil im Vordergrund stehen.

Es sind zwei wesentliche Probleme für eine komplette Integration zu beachten:
Zum einen wird eine Änderung durch das Engineering direkt und unmittelbar Auswirkungen auf die Steuerung haben. Temporäre inkompatible Zustände, wie sie bei einer Programmierung durchaus üblich sind, müssten erkannt und abgefangen werden. Um den Prozess in jedem Fall sinnvoll weiterführen zu können, muss also in jedem Fall die letzte, bekannte sinnvoll Steuerungsprogrammierung vorgehalten werden und zu einem vom Engineering vorgegeben Prozess umgeschaltet werden. Dieser Zeitpunkt ist auch im traditionellen Umfeld als das Laden / Download bekannt. In diesem Punkt ist das Kompilieren also in der einen oder anderen Form notwendig und eine unmittelbare Ausführung der Funktionsbausteine nicht möglich. Zum anderen ist auch der Performance-Gewinn durch eine Kompilierung auf das Zielsystem nicht zu vernachlässigen. Dabei werden neben den Ausführungsoptimierungen auch Ressourcen-Abschätzungen vorgenommen.

Zusammengenommen kann also festgehalten werden, dass ein diskreter Zeitpunkt existieren muss, um eine Umschaltung von einer Version der Steuerung zu einer neueren Version vorzunehmen. Zu diesem Zeitpunkt kann ohne Probleme auch eine Kompilierung auf das Zielsystem erfolgen. Die oben vorgestellte Abbildung der Modellierung der Steuerung zur Laufzeit mit anschließendem Übertragen auf die SPS in traditionellen Programmiersprachen erscheint damit ideal.

7 Zusammenfassung

The best way to predict the future is to invent it.
Alan Kay

Zusammenfassend formuliert, beschreibt die Arbeit Konzepte und Methoden, die vielfältigen Modelle in der Automatisierungstechnik effizienter zu nutzen.

In der Automatisierungstechnik existieren eine Vielzahl von unabhängig voneinander entwickelten Modellen, die größtenteils auf Basis sehr ähnlicher Meta-Modelle formuliert werden.
Ausgegangen wird von einer modellgetriebenen Instanzumgebung. Die bisher existierenden Konzepte sehen dabei vor, dass Modelle in einer Instanzumgebung verwaltet werden. Diese bietet Kommunikationsschnittstellen zur Abfrage, Erkundung und Manipulation der Modelle an und ist eine abgeschlossene Umgebung.

Da die Modelle unabhängig entwickelt wurden, stehen sie selbst, wenn sie in einer gemeinsamen Instanzumgebung verwaltet werden, nicht in Relationen zueinander. Hierdurch werden teilweise wesentliche Informationen nicht abgebildet bzw. Modelle mit ambiguenten Informationen werden genutzt.
Die Arbeit stellt das Konzept der *Modell-Interkonnektionen* vor. Hierbei werden Modelle formuliert, die aufbauend auf anderen Modellen Relationen, zwischen diesen beschreiben. Ambiguitäten werden hierdurch vermieden und die Modelle können weiterhin unabhängig voneinander entwickelt werden, was aufgrund des Expertenwissens der Modelle unbedingt notwendig ist.

Wenn Relationen zwischen Modellen aufgebaut werden sollen, ist es sinnvoll auch Relationen zwischen Instanzumgebungen und damit Geräten zuzulassen. Konzepte und Mechanismen um dieses transparent aus Sicht der Modelle zu realisieren sind bisher nicht existent, da Modelle in ihrer jeweiligen Instanzumgebung betrachtet werden. Die *Externen Verbindungen* beschreiben das Konzept der Verteilung über Gerätegrenzen hinweg. Vorgestellte Komponenten bieten entsprechende Schnittstellen, sodass beim Abfragen die Verteilung nicht beachtet werden muss.

Die vorgeschlagenen Konzepte und Komponenten wurden größtenteils prototypisch im Rahmen von Forschungsprojekten realisiert und an Demonstrationsanlagen in Betrieb genommen. Hiermit wurde die Einsatzfähigkeit zumindest im Rahmen der begrenzten Möglichkeiten nachvollziehbar dargelegt.

Zusätzlich wurden Kommunikationstypen formuliert, welche z.B. für die nötige Synchronität eingesetzt werden können. Hierzu zählt auch der Vorschlag von *Intent-basierten Kommunikation*, welche für modellgetriebene Instanzumgebungen eingesetzt werden kann. Dabei verteilt

eine Basiskomponente Modell-Änderungen auf Basis von „Schlüsseln“, sodass eine indirekte Kommunikation von Modell und Dienst/Anwendung stattfinden kann, ohne dass zurvor eine Erkundung des Modells stattfinden muss.

7.1 Ausblick

Eine Software, welche die vorgeschlagenen Konzepte realisiert, sollte nicht überstürzt realisiert werden. Die verwendeten Technologien, um die Kommunikation abzubilden, stellen eine gemeinsame Plattform dar, die später von allen Anwendern unterstützt werden muss. Entsprechende Vorarbeiten und Untersuchungen zur Geräteunterstützung sind hier wichtig.

Auf Seiten der Modelle sind Abbildungen auf das Meta-Modell der Instanzumgebung zu vollziehen. Diese sollten durch die entsprechenden Experten der Modelle geschehen. Im Nachgang werden dann durch eine Kombination der Experten die entsprechenden Interaktionsmodelle beschrieben. Die Abbildung selber wird dabei z.B. für OPC-UA bereits erstellt – so existieren Arbeitsgruppen, um AutomationML auf OPC-UA abzubilden, wie es auch in [Sch13] beschrieben ist.

Für die Verteilung über Instanzumgebungen hinweg, ist einer der wesentlichen Arbeitspunkte die Transaktionssicherheit (AKID-Eigenschaften) der Modell-Veränderungen gerade im verteilten Umfeld sicherzustellen.

Ein weiterer, wichtiger Punkt ist die Adressierung von Ambiguitäten zwischen Modellen. Bilden Modelle gleiche Informationen ab, ist die einfachste Lösung, entsprechende Interkonnektionen zu definieren - beispielsweise äquivalent_zu. Weitergehende Arbeiten können hier jedoch auch dafür sorgen, dass unterschiedliche Sichtweisen auf die gleichen Daten existieren, also die Daten nicht doppelt in den Modellen abgelegt werden.

Begriffsverzeichnis

Adresse Möglichkeit einen Kommunikationspartner sowohl zu identifizieren, wie auch anzusprechen. 21, 49

Aktive Komponente Komponente mit Verhalten (Ablauf). 27, 45

Anwendung Erfüllt einen Zweck dem Anwender gegenüber. 28

Ausführungsumgebung Aufbauend auf Instanzumgebung auch eine zeitliche Komponente (Scheduler) um aktive Komponenten auszuführen. 27

Dienst (Teil-)Anwendung, die entfernt über Schnittstellen angesprochen werden kann, siehe Kapitel 2.4.2.1. 28

Dienst-/Geräte-Inspektor (DGI) Dienst, welcher im Auftrag von Anwendungen/Diensten die Überwachung von entfernten Geräten oder Diensten auf Geräten übernimmt. 94, 106

Dienst-Schnittstelle Aufruf-Beschreibung eines Dienstes. 30

End-Point-Ressource (ERP) Eine strukturierte Adresse. 51, 55, 56, 81

Externe Verbindung Verbindung zwischen zwei Instanzumgebungen. 80, 81, 85, 87, 89, 101, 104, 115

Geräte-Repräsentation Modell-Repräsentation eines Gerätes. 79

Instanzumgebung Anwendung zum Modell-Raum mit Schnittstellen zur Manipulation der Modelle / Instanzen. 25

Intent indirekte Kommunikation über eine Systemkomponente. 65, 75, 98

Kommunikations-Medium Ebene aus physikalischem Übertragungsmedium ggf inkl. Software zur Nutzung. 18, 19

Komponente vorgefertigte, in sich strukturierte und unabhängig hantierbare Einheit[...] (nach [12]). 14

Lokale Service Management Dienst, welcher die lokale Verwaltung der Dienste auf einem Gerät übernimmt. 94

Meta-Modell Konstrukte um Modelle zu beschreiben. 12

Migration Ablösung einer Komponente durch eine andere - entweder zeitlich (Versionsupdate) oder örtlich (Verlagerung). 113

Modell (Teil-) Abbildung der Realität. 11

Modell-Erkundungsfunktion Dienst-Schnittstelle um die existierenden Modelle / Instanzen zu erkunden. 23, 72

Modell-Explorer Dienst, welcher die SystemBasisDienste durch eine transparente Kommunikation ermöglicht. 95

Modell-Instanz (MI) Modell in einem Modell-Raum oder Instanzumgebung. 86, 95

Modell-Interkonnektion Verbindung zwischen zwei unabhängigen Modellen. 78

Modell-Interkonnektions-Modell (MIM) Modell, das Modell-Interkonnektionen beschreibt. 79

Modell-Interkonnketions-Komponente (MIK) Komponente, die die Verwaltung der Modell-Interkonnektionen eines Modells übernimmt. 95, 102

Modell-Master Verwalter eines Modells und seiner Verteilung. 72, 84, 114

Modell-Raum Speicherort für Modell. 24

Modell-Änderungsfunktion Dienst-Schnittstelle um die existierenden Modelle / Instanzen zu manipulieren. 72

Nachricht Datensatz, der zwischen Anwendungen/Diensten übertragen wird. 21, 58

Objekt-Referenz Referenz auf ein Objekt. 81

Passive Komponenten Komponenten ohne eigene Handlung zur reinen Datenhaltung. 45

Remote-Modell-Inspektor (RMI) Dienst, welcher im Auftrag von Modellen oder Diensten die Überwachung eines entfernten Modells übernimmt. 94, 104

Remote_äquivalent Relation, die zwei Objekte als äquivalent kennzeichnet. 82

SystemBasisDienst (SBD) Schnittstelle zur Modell-Erkundung oder -Manipulation nach NE 139/141. 24, 62, 88, 97, 101

Literaturverzeichnis

[Alb03] ALBRECHT, Harald: *On Meta-Modeling for Communication in Operational Process Control Engineering*, RWTH Aachen University, Diss., 2003. – ISBN 3-18-397508-4

[Bal08] BALZERT, Helmut: *Lehrbuch der Softwaretechnik: Softwaremanagement*. Spektrum Akademischer Verlag, 2008. – ISBN: 978-3827411617

[BHMO13] BRANDL, Dennis ; HUNKAR, Paul ; MAHNKE, Wolfgang ; ONO, Toshio: OPC UA and ISA 95. In: *atp - Automatisierungstechnische Praxis* 01/02 (2013)

[BMM12] BIFEL, Stefan ; MORDINYI, Richard ; MOSER, Thomas: Integriertes Engineering mit Automation Service Bus. In: *atp - Automatisierungstechnische Praxis* 12 (2012)

[CHF14] CHRISTIANSEN, L. ; HOERNICKE, M. ; FAY, A: Modellgestütztes Engineering - Basis für die Automatisierung der Automatisierung. In: *atp* 03 (2014), S. 18–26

[Con97] CONRAD, Stefan: *Föderierte Datenbanksysteme. Konzepte der Datenintegration*. Springer-Verlag, 1997. – ISBN 3-540-63176-3

[Dri05] DRINJAKOVIC, Dino: *Zugriff auf Informationen und Dienste in einem verteilten Automatisierungssystem mit selbstkonfigurierenden, semantischen Ordnungsstrukturen*. VDI-Verlag, 2005 (Fortschritt-Beriche VDI: Reihe 8, Mess-, Steuerungs- und Regelungstechnik). – ISBN 9783185057083

[EM07] ENSTE, Udo. ; MÜLLER, Jochen: *Datenkommunikation in der Prozessindustrie: Darstellung und anwendungsorientierte Analyse*. Oldenbourg Industrieverlag, 2007. – ISBN 9783835631168

[ERD11] EPPLE, Ulrich ; REMMEL, Markus ; DRUMM, Oliver: Modellbasiertes Format für RI-Informationen - Verbesserter Datenaustausch für das PLT-Engineering. In: *atp - Automatisierungstechnische Praxis* 1/2 (2011)

[Gös14] GÖSSLING, Andreas: *Device Information Modeling in Automation - A computer-scientific approach*. 2014. – `http://slubdd.de/katalog?TN_libero_mab216034823`

[Hod13] HODEK, Stefan: *Methode zur vollautomatischen Integration von Feldgeräten in industrielle Steuerungssysteme*, Techn. Univ. Kaiserslautern, Diss., 2013. – ISBN 978-3-943995-35-0

[Hor01] HORN, Paul: Autonomic Computing: IBM's Perspective on the State of Information Technology. (2001). www.research.ibm.com. – (letzter Besuch: 1. Februar 2013)

[HR01] HÄRDNER, Theo ; RAHM, Erhard: *Datenbanksysteme - Konzepte und Techniken der Implementierung (2nd Edition)*. Springer, 2001. – ISBN: 978-3-540-42133-7

[Jar] JARVIS, Jeff: *New rule: Cover what you do best. Link to the rest*. Website. http://buzzmachine.com/2007/02/22/new-rule-cover-what-you-do-best-link-to-the-rest/. – (letzter Besuch: 26. November 2015)

[KE12] KAMPERT, David ; EPPLE, Ulrich: Kernmodelle für die Systembeschreibung - Ein Konzept zur Vereinfachung. In: *atp - Automatisierungstechnische Praxis* 7/8 (2012)

[LMRU11] LEHNHOFF, Sebastian ; MAHNKE, Wolfgang ; ROHJANS, Sebastian ; USLAR, Mathias: IEC 61850 based OPC UA Communication - The Future of Smart Grid Automation. In: *Proceedings of 17th Power Systems Computation Conference* (2011)

[MBS+11] MERSCH, Henning ; BEHNEN, Daniel ; SCHMITZ, Dominik ; EPPLE, Ulrich ; BRECHER, Christian ; JARKE, Matthias: Gemeinsamkeiten und Unterschiede der Prozess- und Fertigungstechnik - Commonalities and Differences of Process and Production Technology. In: *at - Automatisierungstechnik* 1 (2011), Januar, Nr. 1, 7-17. http://www.oldenbourg-link.com/doi/abs/10.1524/auto.2011.0891

[ME11] MERSCH, Henning ; EPPLE, Ulrich: Requirements on Distribution Management for Service-Oriented Automation Systems. In: *Proceedings of Emerging Technologies and Factory Automation (ETFA) 2011* (2011). – ISBN: 978-1-4577-0016-3

[ME12] MERSCH, Henning ; EPPLE, Ulrich: Concepts of service-orientation for process control engineering. In: *Proceedings of IEEE Multi-Conference on Systems, Signals & Devices: Systems, Analysis and Automatic Control 2012* (2012). – ISBN: 978-1-4673-1589-0

[Mey03] MEYER, Dirk: *Objektverwaltungskonzept für die operative Prozessleittechnik*, RWTH Aachen University, Diss., 2003. – ISBN: 3-18-397508-4

[MKE09] MERSCH, Henning ; KLEEGREWE, Christian ; EPPLE, Ulrich: Neue Konzepte zur Selbstkonfiguration leittechnischer Komponenten. In: *VDI-Berichte 2067 - AUTOMATION 2009* 1 (2009), S. 105ff.. – ISBN: 978-3-18-092067-2

[MKE10] MERSCH, H. ; KLEEGREWE, C. ; EPPLE, U.: Service-orientation on behalf of self-configuration for the automation environment. In: *Proceedings of The 36th Annual Conference of the IEEE Industrial Electronics Society (Phoenix, AZ)* 0 (2010), S. 1373–1378. – ISBN 978-1-4244-5226-2

[Nig14] NIGGEMANN, Oliver: Industrie 4.0 ohne modellbasierte Softwareentwicklung - Und warum es ohne Modelle nicht gehen wird... In: *atp - Automatisierungstechnische Praxis* 05 (2014), S. 22–29

[Ohl02] OHLEBUSCH, Enno: *Advanced Topics in Term Rewriting*. Springer, 2002. – ISBN: 978-0387952505

[Pel03] PELTZ, Chris: Web Services Orchestration and Choreography. In: *IEEE Computer* (2003)

[Pol94] POLKE, Martin: *Prozessleittechnik*. Oldenbourg, 1994. – ISBN: 978-3486225495

[Sau13] SAUER, Olaf: *Forschungsprojekt „Secure plug and work“*. Website. `http://www.bmbf.de/de/14626.php`. Version: 2013. – (letzter Besuch: 26. November 2015)

[Sch13] SCHLEIPEN, Miriam: *Adaptivität und semantische Interoperabilität von Manufacturing Execution Systemen (MES)*, KIT Scientific Publishing, Karlsruhe, Diss., 2013. – ISBN: 978-3-86644-955-8

[SEE09] SCHLÜTTER, Markus ; EPPLE, Ulrich ; EDELMANN, Thomas: Dienstesysteme für die Leittechnik - Ein Einblick. In: *VDI-Berichte 2067 - Automation 2009: Fit for Efficiency* (2009)

[SME10] SCHLÜTTER, Markus ; MERSCH, Henning ; EPPLE, Ulrich: Ordnungsschemata für Dienste in der Leittechnik. In: *Proceedings of Entwurf komplexer Automatisierungssysteme - EKA 2010* 0 (2010), S. 317–325. – ISBN: 978-3-940961-41-9

[SMS11] SCHLEIPEN, Miriam ; MÜNNEMANN, Ansgar ; SAUER, Olaf: Interoperabilität von Manufacturing Execution Systems (MES) - Durchgängige Kommunikation in unterschiedlichen Dimensionen der Informationstechnik in produzierenden Unternehmen. In: *at - Automatisierungstechnik* 07 (2011), S. 413–424

[tom] *Apache Tomcat*. `http://tomcat.apache.org/`. – (letzter Besuch: 26. November 2015)

[Urb12] URBAS, Leon: *Process Control Systems Engineering*. Oldenbourg Industrieverlag, 2012. – ISBN 978-3-8356-3198-4

[ZGPU12] ZIEGLER, Jens ; GRAUBE, Markus ; PFEFFER, Johannes ; URBAS, Leon: Beyond app-chaining: Mobile app orchestration for efficient model driven software generation. In: *Proceedings of Emerging Technologies and Factory Automation (ETFA) 2012* 0 (2012)

Normen und Richtlinien

[1] VDI/VDE 3690: XML in der Automation.

[2] ISO/IEC 7498: Information technology - Open Systems Interconnection - Basic Reference Model: The basic model., 1994.

[3] IEC / DIN EN 61512: Batch control, 1997.

[4] IEC 61131-3: Programmable Controllers - Part 3: Programming languages, 2nd edition, 2003.

[5] VDI/VDE 3682: Formalisierte Prozessbeschreibungen, 2005.

[6] IEC 61499: Funktionsbaustrine für industrielle Leitsysteme, 2006.

[7] IEC PAS 62424: Representation of process control engineering - Requests in P&I diagrams and data exchange between P&ID tools and PCE-CAE, 2007.

[8] IEC/ISO 11783: Tractors and machinery for agriculture and forestry - Serial control and communications data network. Norm, 2007.

[9] IEC 62541: OPC Unified Architecture Part 1-10, 2008.

[10] VDI/VDE 5610: Wissensmanagement im Ingenieurwesen, 2009.

[11] ISA-95 - Enterprise Control Systems, 2010.

[12] DIN SPEC 40912: Kernmodelle - Beschreibung und Beispiele, 2014.

[13] IEC 62714: Datenaustauschformat für Planungsdaten industrieller Automatisierungssysteme - Automation markup language. Norm, 2015.

[14] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Andreas Savva. Job Submission Description Language (JSDL) Specification, 2005.

[15] ITU Telecommunication Standardization Sector (ITU-T). RFC 1050: RPC: Remote Procedure Call Protocol specification, 1988. `http://www.ietf.org/rfc/rfc1050` (letzter Besuch: 26. November 2015).

[16] ITU Telecommunication Standardization Sector (ITU-T). RFC 2131: Dynamic Host Configuration Protocol, 1997. `http://www.ietf.org/rfc/rfc2131.txt` (letzter Besuch: 26. November 2015).

[17] ITU Telecommunication Standardization Sector (ITU-T). RFC 2459: Internet X.509 Public Key Infrastructure, 1999. http://www.ietf.org/rfc/rfc2459 (letzter Besuch: 26. November 2015).

[18] ITU Telecommunication Standardization Sector (ITU-T). RFC 5322: Internet Message Format, 2008. http://www.ietf.org/rfc/rfc5322 (letzter Besuch: 26. November 2015).

[19] Namur. NA 35 - Abwicklung von PLT-Projekten (Handling PCT Projects), 2003.

[20] Namur. NE 139 - Informationsschnittstellen in der Prozessautomatisierung; Betriebliche Eigenschaften, 2012.

[21] OASIS Web Services Business Process Execution Language (WSBPEL) TC. Web Services Business Process Execution Language Version 2.0 (WS-BPEL). Website. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (letzter Besuch: 26. November 2015).

[22] Object Management Group. Systems Modeling Language (SysML). http://www.sysml.org/docs/specs/OMGSysML-v1.3-12-06-02.pdf (letzter Besuch: 26. November 2015).

[23] Object Management Group. Enterprise Collaboration Architecture (ECA) Specification, 02 2004. http://www.omg.org/cgi-bin/doc?formal/04-02-01.pdf (letzter Besuch: 26. November 2015).

[24] Object Management Group. MetaObject Facility (MOF), 2005. http://www.omg.org/mof/ (letzter Besuch: 26. November 2015).

[25] Object Management Group. Unified Modeling Language (UML). Website, 2012. http://www.uml.org/ (letzter Besuch: 26. November 2015).

[26] Organization for the Advancement of Structured Information Standards. Reference Model for Service Oriented Architecture 1.0., OASIS Standard , 2006. http://docs.oasis-open.org/soa-rm/v1.0/ (letzter Besuch: 26. November 2015).

[27] PLCopen. OPC UA Information Model for IEC 61131-3. http://www.plcopen.org/pages/tc4_communication/ (letzter Besuch: 26. November 2015).

[28] PLCopen. XML Formats for IEC 61131-3, 2009. http://www.plcopen.org/pages/tc6_xml/downloads/tc6_xml_v201_technical_doc.pdf (letzter Besuch: 26. November 2015).

[29] RSS Advisory Board. RSS 2.0 Specification. Website, 2009. http://www.rssboard.org/rss-2-0 (letzter Besuch: 26. November 2015).

[30] The Open Group. Distributed Computing Environment. http://www.opengroup.org/dce/ (letzter Besuch: 26. November 2015).

[31] W3C. Web Services Description Language (WSDL) 1.1. W3C Recommendation. http://www.w3.org/TR/wsdl.html (letzter Besuch: 26. November 2015).

[32] W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation, 11 1999. http://www.w3.org/TR/xpath/ (letzter Besuch: 26. November 2015).

[33] W3C. Web Services Choreography Description Language Version 1.0. W3C Recommendation, 2004. http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/ (letzter Besuch: 26. November 2015).

[34] W3C. Web Services Addressing 1.0 - Core. W3C Recommondation, 2006. http://www.w3.org/TR/ws-addr-core/ (letzter Besuch: 26. November 2015).

Lebenslauf - Henning Mersch

Persönliche Daten

Geburtsdatum:	12. April 1978
Geburtsort:	Bielefeld

Ausbildungsdaten

Schulausbildung:	bis 06/1997 Max-Planck-Gymnasium, Bielefeld Abschluss: Allgemeine Hochschulreife
Zivildienst:	07/1997 – 07/1998
Hochschulausbildung:	10/1998 – 10/2004 Studium naturwissenschaftliche Informatik Universität Bielefeld Abschluss: Diplom-Informatik

Tätigkeiten

wiss. Angestellter:	11/2004 – 09/2005 Praktische Informatik, Technische Fakultät Universität Bielefeld
wiss. Angestellter:	09/2005 – 06/2007 Verteilte Systeme und Grid Computing, Zentralinstitut für angewandte Mathematik Forschungszentrum Jülich
wiss. Angestellter:	07/2007 – 08/2012 Lehrstuhl für Prozessleittechnik RWTH Aachen University
Produktmanager:	seit 10/2012 Beckhoff Automation GmbH & Co. KG, Verl

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 978-3-18-524508-4