ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

Sprachen zur Beschreibung und Vergleich zum Einsatzzweck von Sprachen zum Beschreiben von Kommunikationsabläufen

an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Kommunikationsinformatik
der Fakultät für Ingenieurwissenschaften

vorgelegt von
Majdi Taher und Alexander Huber

Saarbrücken, 02. August 2019

Inhaltsverzeichnis

1 Einleitung

Es gibt in der Welt der Informatik verschiedene Arten von Sprachen zur Beschreibung von Kommunikationsabläufen. So können zum einen beispielsweise Domänenspezifische Sprachen verwendet werden oder globale Modellierungssprachen. Diese genannten Arten von Sprachen dienen zur Abstraktion und zum besseren Verständnis zwischen Mensch und Computer. Sie sind sogenannte Formale Sprachen, welche künstlich erstellte Sprachen sind und sich für eine präzise Beschreibung von Eigenschaften und Verhalten eignen. Das hier vorgestellte Projekt soll ausgwählte Formale Sprachen in hinsicht auf ihre Funktionsweise und Kommunikationsabläufe kritisch beobachten, beschreiben und vergleichen. Am Ende soll diese Ausarbeitung eine Aussage über die Eignung von Einsatzzwecken der Sprachen bei Kommunikationsabläufen treffen.

1.1 Problemstellung

Da formale Sprachen künstliche Sprachen sind und einen Kontext über eine formale mathematische Syntax beschrieben werden, werden diese bevorzugt zur Abstraktion von Sachverhalten wie der von Kommunikationsabläufen in Systemen verwendet. Nun können unterschiedliche Sprachen einen Sachverhalt unterschiedlich gut oder schlecht wiedergeben. Wobei gut oder schlecht, abhängig des Einsatzortes ist und welche formalen Kriterien eine wiedergabe erleichtert oder erschwert.

1.2 Zielsetzung

In dieser Projektarbeit gilt es den Einsatz unterschiedlicher Sprachen im Einsatz eines Kommunikationssystems zu beschreiben und zu vergleichen. Der Vergleich ist mit eigens gewählten Bewertungskriterien vorzunehmen und über den Einsatzort der Sprachen in einem Kommunikationssystem zu beschreiben. Dazu sollen die Informationen über die ausgewählten Modellierungssprachen aus den Dokumenten der Spezifikationen **SDL!** Z.100 [**??**], **MSC!** Z.120 [**??**] und OMG **UML!** [**??**] entnommen werden. Die Kriterien zum Vergleich werden über die verfügbare Literatur zusammen getragen und dann die Sprachen mit ihnen ausgewertet.

1.3 Gliederung

Die Arbeit gliedert sich wie folgt in weitere Kapitel. Im Folgenden wird in Kapitel 2 auf die technischen Grundlagen der einzelnen Sprachen, deren Modellkonzeption, Notation und Geschichte eingegangen. Danach konzentriert sich das dritte Kapitel auf die Bewertungskriterien, nach welchen wir die einzelnen Sprachen hinsichtlich ihrer Vor- und Nachteile beschreiben. In Kapitel 4 werden dann die Sprachen hinsichtlich der gennanten Kriterien aus Kapitel 3 analysiert und die Vor- bzw. Nachteile beschrieben. Das letzte Kapitel 5 handelt von den Erfahrungen und Erkenntnissen, die während der Arbeit gesammelt wurden. Sie wird noch einmal überblickt und schließt mit einem Fazit ab.

2 Grundlagen

Im Gegensatz zu weiten Teilen der Informatik besteht in der Telekommunikation seit langem die Notwendigkeit, herstellerunabhangige und präzise Standards für Kommunikationsprotokolle und -dienste zu erstellen. Diese Anforderung hat dazu gefuhrt, dass man sich schon frühzeitig mit der Entwicklung der formalen Modellierungssprachen Message Sequence Chart (MSC) und Specification and Description Language (SDL) beschaftigt hat. Im Gegensatz etwa zur Unified Modeling Language (UML) liegt MSC und SDL eine formale Semantik zugrunde, die die Bedeutung der einzelnen Sprachkonstrukte klar und eindeutig mathematisch definiert.
MSC und SDL sind von der International Telecommunication Union standardisiert worden. Fur beide Sprachen existieren eine graphische Reprasentation (MSC /GR bzw. SDL/-GR) fur die Benutzung durch Menschen und eine textuelle Notation (MSC/PR bzw. SDL/PR) fur eine maschinelle Weiterbearbeitung. MSC und SDL werden haufig gemeinsam im Software-Entwicklungsprozess eingesetzt. MSC dient dabei zur Anforderungsdefinition, Testfallspezifikation und Dokumentation, wahrend SDL in der Spezifikationsphase und der Implementierungsphase eingesetzt wird.

2.1 SDL! (SDL!)

Die **SDL!** (SDL, engl. Spezifikations und Beschreibungssprache) ist eine von der **ITU-T!** (**ITU-T!**) (ITU-T, engl. Internationalen Telekommunikations Vereinigung für Standardisierung der Telekommunikation) standardisierte objektorientierte Modellierungssprache und wurde erstmalig 1976 definiert. Die aktuellste Version von **SDL!** ist SDL-2010, welche eine Überarbeitung der SDL-Version SDL-2000 aus dem Jahre 1999 ist. Wenn im folgenden von **SDL!** gesprochen wird, wird immer die Version **SDL!**-2010 gemeint. In den letzten Überarbeitungen wurde die Definition von **SDL!** um objekt-orientierte Aspekte erweitert und mit Sprachen wie UML und ASN.1 harmonisiert. **SDL!** wird zur Beschreibung von Telekommunikationssystemen, deren Abläufe, sowie für Protokoll Definitionen und in verteilten Systemen eingesetzt. Der Hauptfokus von **SDL!** ist, sich einen genauen Überblick über das Verhalten genannter Systeme zu machen, wobei Eigenschaften mit anderen Techniken beschrieben werden müssen. Dazu konzentriert sich die Anwendung der Spezifikation unter anderem auf Echtzeitanwendungen, welche in der Spezifikation genauer zusammengefasst und beschrieben werden.

2.1.1 Spracheigenschaften

Die Eigenschaften der Sprachdefinition von **SDL!** sind folgend beschrieben[REC111P.3]:

bstrakte Grammatik Die abstrakte Grammatik von **SDL!** wird von einer abstrakten Syntax und statischen Bedingungen beschrieben. Die Abstrakte Syntax kann entweder mit einer textbasierten Grammatik oder einem grafischen Metamodell erstellt werden.

onkrete Grammatik Die konkrete Syntax wird durch eine grafische Syntax, statischen Bedingungen und Regeln für die grafische Syntax beschrieben. Beschrieben wird sie durch die erweiterte Backus-Naur Form. Wenn jedoch in der abstrakten Grammatik ein grafisches

Metamodell verwendet wurde, ist es erlaubt dieses um kontrkete Eigenschaften zu erweitert und zu verwenden.

Semantik Die Semantik beschreibt ein Konstrukt,samt dessen Eigenschaften, Interpretation und dynamischen Bedingungen.

Model Ein Model gibt Notationen eine Abbildungsform, wenn diese keine direkte abstrakten Syntax besitzen.

2.1.1.1 Metamodell

Es werden Anstrengungen unternommen, jedoch existiert derzeit kein öffentlich zugängliches Metamodell von **SDL!**, welches alle Aspekte der Sprache in sich vereinigt. So hat die **ITU-T!** selbst ein Meta-Metamodell auf Grundlage von **UML!** (**UML!**) zu erstellen, jedoch deckt diese Definition, welche auch SDL-UML genannt wird, nur Teile der Sprachdefinition von **SDL!** ab.

2.1.2 Architektur

Die **SDL!**-Spezifikation erlaubt die struckturierte Trennung in Diagramme und Packete und somit ein aufteilen eines Systems in viele kleinere Teilsysteme, wodurch die Entwicklung in größeren Gruppen vereinfacht wird. Dies fürht ebenzu dazu, dass andere Sprachen in Packete definiert werden können und so zur Sprachdefinition hinzukommen können. Die Struktur ist hierarchich gegliedert und besteht aus Prozessen, welche logisch in Blöcke gegliedert werden können und durch endliche Automaten dargestellt werden. Diese Blöcke können in sich selbst gegliedert werden und bilden einzeln oder zusammen dann die höchste Ebene, das System. Diese Strucktur wird anhand folgender Abbildung x.x veranschaulicht.

2.1.2.1 Agenten

In der Sprachdefinition von **SDL!** ist von sogenannten Angenten die rede. Ein Agent beschreibt eine Menge an Angenten. Diese Agenten bestehen aus Attributen, Prozeduren oder endlichen Automaten und wiederum enthaltenen Agenten. **SDL!** besitzt zwei Arten von Agenten, die eben genannten Blöcke und Prozesse. Folgende Abbildung x.x veranschaulicht den Aufbau der Architektur.

System In dem **SDL!**-System werden die die Funktionsbausteine hierarchich struckturiert.

Block Prozesse werden logisch in Blöcke eingeteilt. Ein Block kann auch wiederrum in einen Block logisch eingeteilt werden.

Prozess Prozessagenten bestehen aus **EFSM!** (**EFSM!**)s

Prozedur Prozessagenten bestehen aus **EFSM!**s

2.1.3 Kommunikation

2.1.3.1 Asynchrone Kommunikation

2.1.3.2 Synchrone remote procedure calls

2.1.4 Verhalten

2.1.5 Daten

In SDL können auf zwei Arten daten beschrieben werden, einerseits mit dem **ADT!** (**ADT!**)(abstract data type) oder mit **ASN1!** (**ASN1!**). Dadurch können Daten unter Sprachen ausgetasucht werden und bereits bestehende Datenstruckturen wiederverwendet werden. **ADT!** hingegen definiert keine Datenstruckturen sondern ledichlich einen Satz an Werten, Operations und Bedingungen. Folgende Abbildung x.x veranschaulicht dies an einem Beispiel. **SDL!** beschreibt noch einen fortgeschritteren Ansatz von **ADT!**, wo Operationen unter anderem zum verstecken von Datenmanipulation verwendet werden, dies kann bei weiterem Interesse in der Spezifikation [Spezifikation eintragen] nachgelesen werden.

2.1.6 Vererbung

2.1.7 Diagramarten

2.2 Unified Modeling Language (UML)

2.2.1 Historie und Ziel

Die Unified Modeling Language (UML) ist eine durchgängige Modellierungssprache von der organisatorischen Beschreibung von Geschäftsprozessen bis zu direkt ausführbaren Modellen, d.h. bis zur Implementierung. Sie ist über ISO standardisiert (ISO /IC 19501). Ein erster Ansatz wurde 1990 auf der Grundlage verschiedener Notationssysteme entwickelt. Die Standardisierung, Pflege und Weiterentwicklung der Sprache wurde an die OMG übergeben, die die Sprache im Jahr 1997 zur Version UML 1.1 weiterentwickelte. Seit Ende der 1990er Jahre haben zahlreiche Personen und Institutionen intensiv an der UML Version 2.0 gearbeitet, die im Jahr 2006 vollständig fertig gestellt und Anfang 2009 von der leicht überarbeiteten Version 2.2 abgelöst wurde. Eine Standardisierung durch die International Standardization Organization (ISO) hat die Version 2.2 allerdings noch nicht erreicht. Diese bleibt bisher nur der Version 1.4.2 vorbehalten.

2.2.2 Metamodell

UML legt Spracheinheiten fest, die auf verschiedenen Ebenen agieren. Mit diesen drücken Sie die Struktur und das Verhalten eines Systems aus. Einige Elemente nutzt die Modellierungssprache, um sich selbst zu definieren. Die Meta-Modellierung umfasst alle Elemente von UML, auch solche, die UML selbst beschreiben. Dafür nutzt es vier hierarchisch angeordnete Ebenen (M0 bis M3).

Die Meta-Metaebene M3 spezifiziert die Metadaten der Modellierungssprache und deren Zusammenhänge mithilfe der Meta Object Facility (MOF). Sie definiert das Metamodell. Zudem befähigt Sie den Metadaten-Transfer. Das von der OMG definierte Format XMI ist ein praktisches Tool, um objektorientierte Daten auf Meta-Metaebene zwischen

Entwicklungstools zu teilen. Die Object Constraint Language (OCL), eine deklarative Programmiersprache, ergänzt UML und reguliert Randbedingungen der jeweiligen Modellierung. Als Textsprache wirkt sie jedoch nur unterstützend, statt selbst für Modellierung zur Verfügung zu stehen.

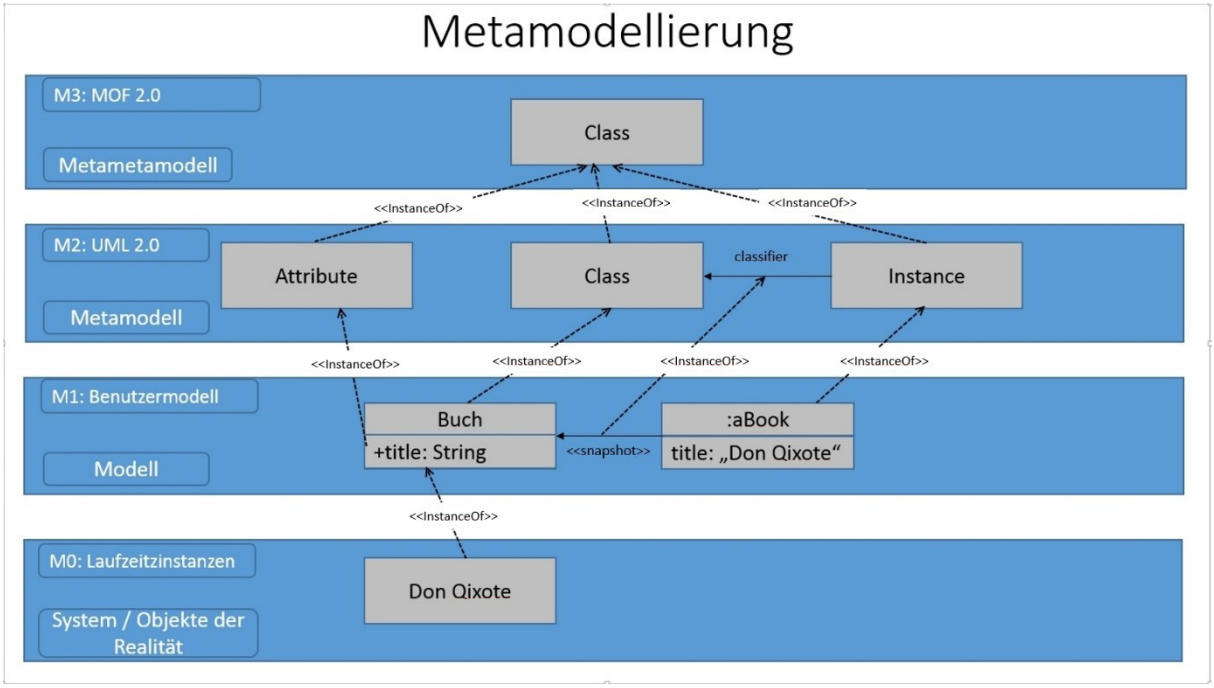

Abbildung 2.1: Die Metamodellierung zeigt die hierarchische Beziehung zwischen den Sprachebenen
Quelle : Ionos - Digital Guide

Die obere Grafik zeigt die Metamodellierung von UML 2.0. Ebene M0 ist die grundlegende Ebene. Sie stellt konkrete, reale Objekte und einzelne Datensätze dar – z. B. ein Objekt oder eine Komponente. Ebene M1 umfasst alle Modelle, die die Daten der Ebene M0 beschreiben und strukturieren. Das sind UML-Diagramme wie das Aktivitätsdiagramm oder das Paketdiagramm (weiter unten erklärt). Um den Aufbau dieser Modelle zu definieren, legen Metamodelle der Ebene M2 die Spezifikationen und Semantik der Modellelemente fest.

Wollen Sie ein verständliches UML-Diagramm erstellen, müssen Sie das Metamodell UML mit seinen Regeln kennen. Die höchste Ebene, M3, ist ein Metamodell des Metamodells. Die erwähnte Meta Object Facility arbeitet auf einer abstrakten Ebene, die Metamodelle definiert. Diese Ebene definiert sich selbst, da sonst weitere, übergeordnete Meta-Ebenen entstünden.

2.2.3 Kurzbeschreibung

Bei der Unified Modeling Language (UML) handelt es sich nicht um eine bestimmte Methode, sondern vielmehr um einen Sammelbegriff für grafische Methoden der objektorientierten Entwicklung und Dokumentation von Software (Object Oriented Design – OOD). Dies umfasst Methoden und Notationen für Planung, Design, Entwurf und Implementierung von Software, die seit den 90er Jahren durch die Object Management Group (die auch BPMN pflegt) zu einem offiziellen Standard, der UML, zusammengeführt wurden.
Im Gegensatz zu anderen Methoden, die primär auf die Modellierung von Prozessen abzielen, kann die UML direkt zur Software-Entwicklung genutzt werden.
Die objektorientierte Sichtweise, auf der UML basiert, zieht ausgehend von der realen Welt Objekte heraus, die mit Attributen beschrieben werden. Die Objekte werden zu Klassen verdichtet, wenn Eigenschaften und Verhalten der Objekte identisch oder ähnlich sind. Klassen können daher als Baupläne für die zu erzeugenden Objekte (die Instanzen einer Klasse) interpretiert werden. Die Objekte, Klassen, Attribute und Methoden bilden die Basis für sämtliche Diagrammtypen der UML. Die verschiedenen Diagrammtypen können in

statische Modelle (-> Strukturdiagramme)
- das Klassendiagramm,
- das Kompositionsstrukturdiagramm (auch: Montagediagramm),
- das Komponentendiagramm,
- das Verteilungsdiagramm,
- das Objektdiagramm,
- das Paketdiagramm und
- das Profildiagramm.

und
dynamische Modelle (-> Verhaltensdiagramme)
- das Aktivitätsdiagramm,
- das Anwendungsfalldiagramm (auch: Use-Case o. Nutzfalldiagramm),
- das Interaktionsübersichtsdiagramm,
- das Kommunikationsdiagramm,
- das Sequenzdiagramm,
- das Zeitverlaufsdiagramm und
- das Zustandsdiagramm

unterteilt werden.
Statische Modelle (wie z.B. das Klassendiagramm) zeigen die Beziehungen zwischen den Klassen und den beteiligten Akteuren auf. Demgegenüber zeigen dynamische Modelle (wie z.B. das Sequenzdiagramm) den Prozessablauf auf. Aufgrund der Vielzahl an Diagrammtypen ergeben sich vielfältige Anwendungsmöglichkeiten, da jeder Typ eine spezifische Sicht auf den zu modellierenden Prozess sowie das System ermöglicht.
Im Folgenden werden einige ausgewählte UML-Diagrammtypen erläutert, wobei jeweils der Nutzen im Rahmen des Geschäftsprozessmanagements herausgearbeitet wird.

Das Anwendungsfalldiagramm (use case diagram) dient im Rahmen der Softwareentwicklung dem Einstieg in die Anforderungsanalyse. Gleichzeitig kann es zur Darstellung der relevanten Geschäftsprozesse einschließlich der Beziehung zu den an den Prozessen beteiligten Personen genutzt werden. Ein Anwendungsfall entspricht hierbei entweder einem Geschäftsprozess oder einem Teilprozess. Durch das Anwendungsfalldiagramm kann dann dargestellt werden, welche Akteure an den betrachteten Prozessen beteiligt sind und welche Prozesse weitere Prozesse beinhalten. Die Akteure sind dabei im Sinne von Rollen eines Benutzers innerhalb des Systems zu verstehen, wobei diese nicht zwingend menschlich sein müssen. Dies ist beispielsweise der Fall, wenn ein anderes System eingebunden wird. Die Anwendungsfälle werden in diesem Diagrammtyp über Ellipsen abgebildet, während die Akteure als Strichmännchen dargestellt werden. Die Beziehungen zwischen dem Anwendungsfall und den beteiligten Akteur werden über ungerichtete Kanten visualisiert.

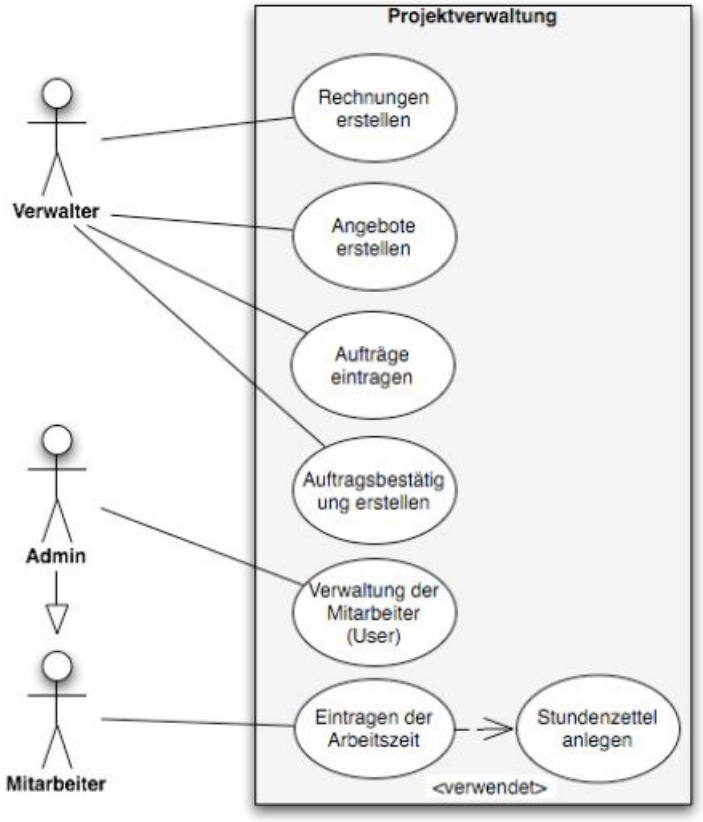

Abbildung 2.2: Beispiel eines Anwendungsfalldiagramms in UML
Quelle : DVZ datenverarbeitungszentrum mecklenburg-vorpommern gmbh

Anwendungsfalldiagramme weisen jedoch den Nachteil auf, dass keine Reihenfolge bei der Bearbeitung der Anwendungsfälle abgebildet werden kann. Es wird somit nicht deutlich, welcher Anwendungsfall vor einem anderen durchgeführt werden muss. Allerdings kann diese Reihenfolge beispielsweise durch andere Methoden der UML, wie das Aktivitätsdiagramm, kompensiert werden. Zudem muss jeder Anwendungsfall anhand einer Beschreibung dokumentiert werden. Hierbei bestehen jedoch keine Vorgaben, weshalb sich Inkonsistenzen und Redundanzen ergeben können. Darüber hinaus kann nicht sichergestellt werden, dass Dritte die Dokumentation auch verstehen. Daher sollte auf Vorlagen, die nicht offiziell zum Standard der UML gehören, zurückgegriffen werden.

Das Klassendiagramm (Class Diagram) eignet sich zur Darstellung von Klassenstrukturen innerhalb eines IKS. Klassendiagramme können nicht direkt für die Geschäftsprozessmodellierung verwendet werden. Stattdessen handelt es sich eine statische Darstellung von Klassen, Objekten und deren Beziehungen untereinander. Klassendiagramme beschreiben jedoch nur, dass eine Interaktion besteht; wie diese ausgestaltet ist kann jedoch nicht dargestellt werden.

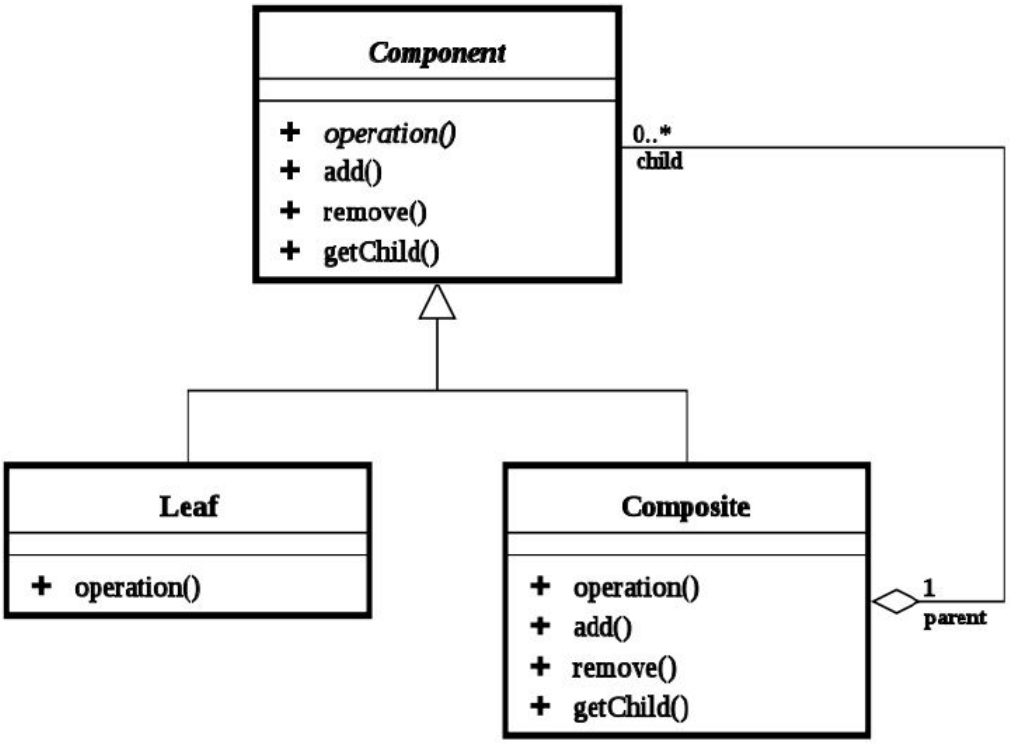

Abbildung 2.3: Beispiel eines Klassendiagramms in UML
Quelle : DVZ datenverarbeitungszentrum mecklenburg-vorpommern gmbh

Das Aktivitätsdiagramm oder auch Ablaufdiagramm (Activity Diagram) wird zur Darstellung von Abläufen verwendet. Im Mittelpunkt steht dabei die Visualisierung paralleler Abläufe. Aus diesem Grund eignet sich dieser Diagrammtyp in einem besonders hohen Maße zur Abbildung von Geschäftsprozessen, da diese zumeist Parallelitäten vorweisen. Aktivitätsdiagramme sind darüber hinaus zur Modellierung von Workflows und zur Verfeinerung von Anwendungsfällen geeignet. Des Weiteren sind Aktivitätsdiagramme in der Lage unterschiedliche Detaillierungsgrade wiederzugeben. So ist es unter anderem möglich ein anwendungsfallübergreifendes Diagramm zu erzeugen und anschließend die darin enthaltenen Anwendungen einzeln zu modellieren.

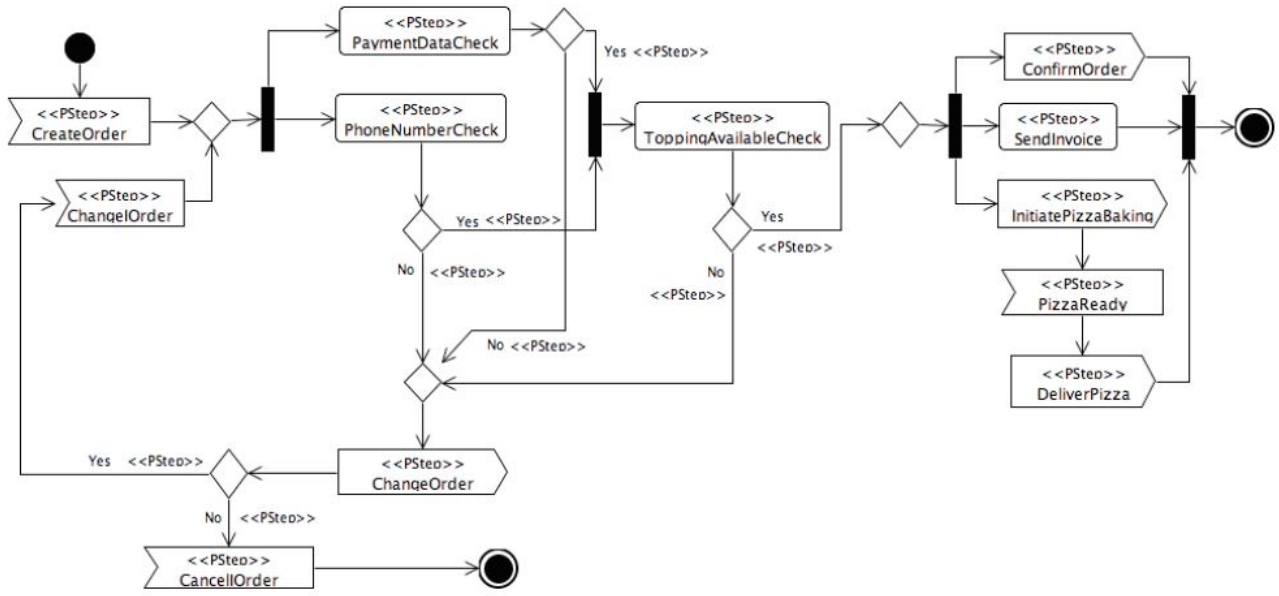

Abbildung 2.4: Beispiel eines Aktivitätsdiagramm in UML
Quelle : DVZ datenverarbeitungszentrum mecklenburg-vorpommern gmbh

2.2.4 UML-Diagramme: Eine Übersicht

Die folgende Übersicht zeigt übergeordnete Kategorien und Anwendungsmöglichkeiten der einzelnen Diagrammtypen in Kurzform. Wenn Sie ein modellorientiertes Software-System, einen Anwendungsfall in der Wirtschaft o. Ä. visuell darstellen wollen, sollten Sie laut Empfehlung der UML-Task-Force vorher einen der UML-Diagrammtypen wählen. Erst dann lohnt es sich, eines der vielen UML-Tools zu wählen, da diese häufig eine gewisse Methode vorschreiben. Dann erstellen Sie das UML-Diagramm.

Kategorie	Diagrammtyp	Verwendung
Struktur	Klassendiagramm	Klassen visualisieren
	Objektdiagramm	Systemzustand in einem bestimmten Moment
	Komponentendiagramm	Komponenten strukturieren und Abhängigkeiten aufzeigen
	Kompositionsstrukturdiagramm	Komponenten oder Klassen in ihre Bestandteile aufteilen und deren Beziehungen verdeutlichen
	Paketdiagramm	Fasst Klassen in Pakete zusammen, stellt Pakethierarchie und -struktur dar
	Verteilungsdiagramm	Verteilung von Komponenten auf Rechnerknoten
	Profildiagramm	Veranschaulicht Verwendungszusammenhänge durch Stereotype, Randbedingungen etc.
Verhalten	Anwendungsfalldiagramm	Stellt diverse Anwendungsfälle dar
	Aktivitätsdiagramm	Beschreibt das Verhalten verschiedener (paralleler) Abläufe in einem System
	Zustandsautomatendiagramm	Dokumentiert, wie ein Objekt von einem Zustand durch ein Ereignis in einen anderen Zustand versetzt wird
Verhalten: Interaktion	Sequenzdiagramm	Zeitlicher Ablauf von Interaktionen zwischen Objekten
	Kommunikationsdiagramm	Rollenverteilung von Objekten innerhalb einer Interaktion
	Zeitverlaufsdiagramm	Zeitliche Eingrenzung für Ereignisse, die zu einem Zustandswechsel führen
	Interaktionsübersichtsdiagramm	Sequenzen und Aktivitäten interagieren

Abbildung 2.5: UML-Diagramme

Quelle : Ionos - Digital Guide

2.2.5 Vor- und Nachteile im Überblick

UML ist heute eine der dominierenden Sprachen für die Modellierung von betrieblichen Anwendungs- bzw. Softwaresystemen. Der erste Kontakt zu UML besteht häufig darin, dass UML-Diagramme im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind. UML-Diagramme gelten als Standard bei objektorientierter Modellierung.

Vorteile	Nachteile
Standardisierte, weit verbreitete Methode	Grundsätzliches IT-Wissen erforderlich
Ergebnisse der Modellierung in UML können zur Umsetzung in Software-Code verwendet werden. Es ist ein automatisierte Erzeugung eines Coderahmen aus den Diagrammen möglich wodurch der Programmier-aufwand reduziert wird	Sehr großer Umfang und Komplexität der Sprache (beispielsweise umfasst die Sprachspezifikation mehr als 1200 Seiten) verursacht Schwierigkeiten bei der Benutzung und erfordert einen hohen Einarbeitungsaufwand
Einfacher Modellaustausch möglich und damit bessere Modellwiederverwendung	Die UML wird hinsichtlich semantischer Inkonsistenzen, Konstruktmehrdeutigkeiten, inadäquaten Notationen und kognitiven Unzugänglichkeiten kritisiert
Eine große Anzahl an UML-Werkzeugen bietet Export-Möglichkeiten in Java, C und C# an. Der Quellcode kann direkt in die technische Implementierung der IT-Anwendung einfließen	Die UML wurde ursprünglich für die Entwicklung von Softwaresystemen konzipiert, daher sind die relevanten Beschreibungs- und Darstellungsmittel weniger auf die Prozessmodellierung ausgerichtet

Abbildung 2.6: UML: Vor- und Nachteile
Quelle : DVZ datenverarbeitungszentrum mecklenburg-vorpommern gmbh

2.3 MSC

MSC ist eine vom ITU-T als Recommendation Z.120 standardisierte graphische Spezifikationssprache. Sie dient zur Beschreibung des Kommunikationsverhaltens zwischen Systemkomponenten und deren Umgebung. Die Kommunikation wird durch den Austausch von Nachrichten (Messages) spezifiziert.
Die MSC-Sprache hat sich aus den OSI Time Sequence Diagrams zu einer vollst¨andigen graphischen Sprache mit formaler Syntax und Semantik entwickelt. Mit den Sequence Diagrams wurde eine Variante von MSC in die UML integriert. Obwohl beide Sprachen auf den gleichen Prinzipien beruhen, gibt es auf Grund der verschiedenen Anwendungsbereiche unterschiedliche Sprachkonstrukte und Unterschiede in der Darstellung. Durch eine Kombination der St¨arken von MSC und den Sequence Diagrams bem¨uht man sich zur Zeit verst¨arkt um eine Harmonisierung der beiden Diagrammarten.
In diesem Abschnitt werden die wichtigsten Konstrukte der MSC-Sprache an Hand von MSCs1 für das in der Einleitung zu dieser Abschnittreihe beschriebene Beispiel erläutert. Das Beispiel beschreibt einen Produktionsprozess, in dem eine Produktionszelle aus vier verschiedenen Einzelteilen vom Typ A, B, C und D zwei Produkte vom Typ AC und BDC herstellt.

2.3.1 Basic-MSC

Basic-MSC umfasst alle Sprachkonstrukte, die notwendig sind, um den Nachrichtenfluss zu spezifizieren. Diese Sprachkonstrukte sind: Instanz, Message, Environment, Action, Timer-Start, Timeout, Timer-Stop, Create, Stop und Condition.

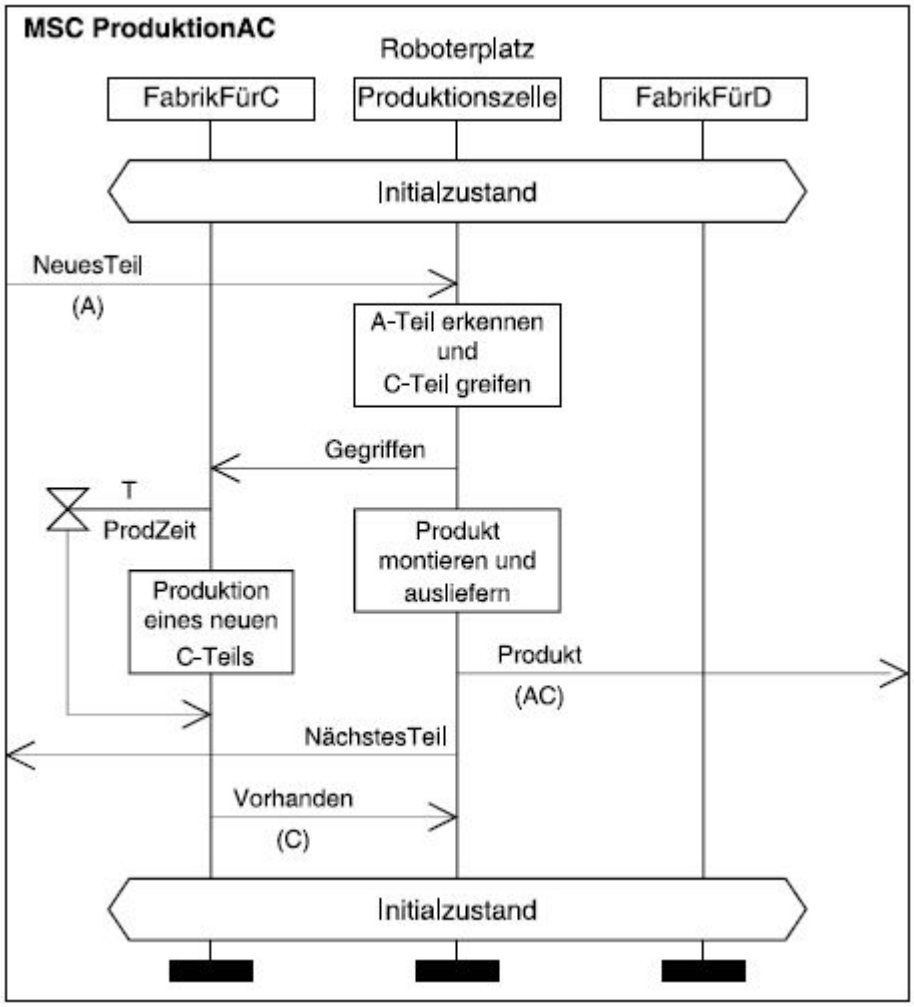

Abbildung 2.7: Ein Basic-MSC
Quelle : The Specification Languages MSC and SDL Part 1: Message Sequence Chart (MSC)

Ein Basic-MSC mit dem Namen ProduktionAC ist in Abbildung 1.6 gezeigt. Es beschreibt die Produktion eines aus einem A- und einem C-Teil zusammengesetzten Produkts. Das MSC abstrahiert von den Details und zeigt nur den Informationsaustausch zwischen den drei Hauptkomponenten des Produktionsprozesses: Der Produktionszelle und den zwei Fabriken für C- und D-Teile.
Der Produktionsprozess befindet sich zu Beginn in seinem Initialzustand: Das System ist initialisiert und Teile der Typen C und D sind gefertigt, eingelagert und der Produktionszelle zur Verfügung gestellt worden. Die Systemumgebung übergibt ein A-Teil an die Produktionszelle. Diese erkennt den Typ des Teils, greift das für die Produktion notwendige C-Teil und meldet das Entfernen des C-Teils vom Greifplatz an die Fabrik, die C-Teile herstellt. Ein Produkt vom Typ AC wird gefertigt und dann ausgegeben. Danach meldet die Produktionszelle die Bereitschaft für die Bearbeitung des nächsten A- oder B-Teils an die Systemumgebung. Parallel zu den Aktionen der Produktionszelle wird ein neues C-Teil produziert und zur Verfügung gestellt.

2.3.1.1 Instanz und Message

Die wichtigsten Sprachkonstrukte von Basic-MSC sind Instanz und Message. Instanzen sind Komponenten, die untereinander oder mit der Systemumgebung asynchron Messages austauschen können.
In der graphischen Form werden Instanzen als vertikale Linien oder alternativ als Säulen dargestellt. Innerhalb des Instanzkopfes wird der Instanzname spezifiziert, zus¨atzlich kann ein Typ angegeben werden (z.B. Instanz Produktionszelle vom Typ Roboterplatz in Bild 1). Das Ende einer Instanz kann durch ein Instanz- Ende-Symbol beschrieben werden. Ein Instanz-Ende bedeutet nicht, dass die Instanz gestoppt ist, sondern lediglich dass in dem MSC keine weiteren Ereignisse für die Instanz beschrieben werden.
Messages werden durch Pfeile dargestellt. Diese Pfeile k¨onnen horizontal oder geneigt sein, um den Zeitverlauf anzuzeigen. Eine Message definiert zwei Ereignisse: Der Pfeilanfang beschreibt das Senden und die Pfeilspitze bezeichnet die Verarbeitung einer Message. Die Beschriftung einer Message besteht aus ihrem Namen und optionalen Message-Parametern, die in Klammern angegeben werden m¨ussen (z. B. Message NeuesTeil mit dem Parameter A in Abbildung 1.6).
Entlang jeder Instanzachse wird eine Totalordnung der spezifizierten Message-Sende- und Message-Verarbeitungsereignisse angenommen. Ereignisse auf verschiedenen Instanzachsen werden durch Message-Kommunikation partiell geordnet, da eine Message zuerst gesendet werden muss, bevor sie verarbeitet werden kann.

2.3.1.2 Environment

Die Diagrammfläche eines MSC wird durch einen rechteckigen Rahmen begrenzt. Der Diagrammrahmen definiert die Systemumgebung und heißt Environment. Messages, die aus der Systemumgebung kommen oder an die Systemumgebung gesendet werden, beginnen und enden auf dem Environment (z.B. die Messages NeuesTeil und Produkt in Bild 1). Im Gegensatz zur Totalordnung entlang der Instanzachsen ist für Sende- und Verarbeitungsereignisse auf dem Environment keine Ordnung definiert.

2.3.1.3 Action

Zusä¨atzlich zur Message-Kommunikation können Aktionen von Instanzen in Form von Actions spezifiziert werden. Eine Action wird durch ein Rechtecksymbol dargestellt, das beliebigen Text enthalten kann (z. B. Action ‚Produkt montieren und ausliefern' in Abbildung 1.6).

2.3.1.4 Timer

Zur Beschreibung von Timern bietet MSC die Sprachkonstrukte Timer-Start, Timeout und Timer-Stop an. Timer-Start spezifiziert das Setzen, Timeout den Ablauf und Timer-Stop das Zur¨ucksetzen eines Timers. In MSC ist ein Timer immer einer Instanz zugeordnet. Die graphischen Timer-Symbole sind daher immer mit der dem Timer zugeordneten Instanz verbunden.

Ein Timer-Start wird durch ein mit der Instanzachse verbundenes Sanduhr-Symbol dargestellt. Ein zugehöriges Timeout wird durch einen Pfeil beschrieben, der am Sanduhr-Symbol beginnt und auf der Instanzachse endet. Ein Timer-Stop wird durch ein mit der

Instanzachse verbundenes Kreuz beschrieben. Ein Timersymbol wird mit dem Timernamen und optional mit Timer-Parametern versehen. Ein Timer-Start und das zugeh¨orige Timeout werden in Bild Abbildung 1.6 verwendet, um die Produktionszeit für ein Teil des Typs C zu modellieren. Der Timer hat den Namen T und den Parameter Prodzeit, der die Laufzeit des Timers spezifiziert.

2.3.1.5 Condition

Eine Condition beschreibt einen Zustand, der sich auf eine Menge der im MSC enthaltenen Instanzen bezieht. Graphisch werden Conditions durch Sechsecke dargestellt, die die Instanzen, auf die sich die Condition bezieht, ¨uberdecken. Conditions werden zur Beschreibung von wichtigen Systemzuständen benutzt. In Bild Abbildung 1.6 befinden sich zwei Conditions, die beide den globalen Systemzustand Initialzustand beschreiben.

2.3.1.6 Create und Stop

Die MSC-Sprache enthält die Konstrukte Create und Stop für die dynamische Erzeugung und Terminierung von Instanzen. Ein Create wird durch einen gestrichelten Pfeil mit optionalen Parametern beschrieben. Ein Create-Pfeil beginnt an der Erzeuger-Instanz und endet am Kopf der erzeugten Instanz. Eine Instanz kann sich selbst durch eine Stop-Aktion terminieren. Ein Stop wird graphisch durch ein Kreuz am Ende der Instanzachse spezifiziert.

2.3.2 Strukturelle Sprachkonstrukte

Strukturelle MSC-Sprachkonstrukte bezeichnen Sprachelemente, die über die Beschreibung des reinen Messageflusses hinausgehen. Mit ihnen lassen sich MSCs und MSC-Teile zu komplexeren Abläufen kombinieren (Inline- Expressions und High-Level-MSC), MSC-Diagramme in anderen MSC-Diagrammen wiederverwenden (References), MSC-Instanzen verfeinern (Decomposition) und allgemeine Ereignisstrukturen für Instanzen definieren (Coregion und General Ordering). Aus Platzgründen kann im Rahmen dieses Artikels nur auf Inline-Expressions, References und High-Level-MSC (HMSC) eingegangen werden.

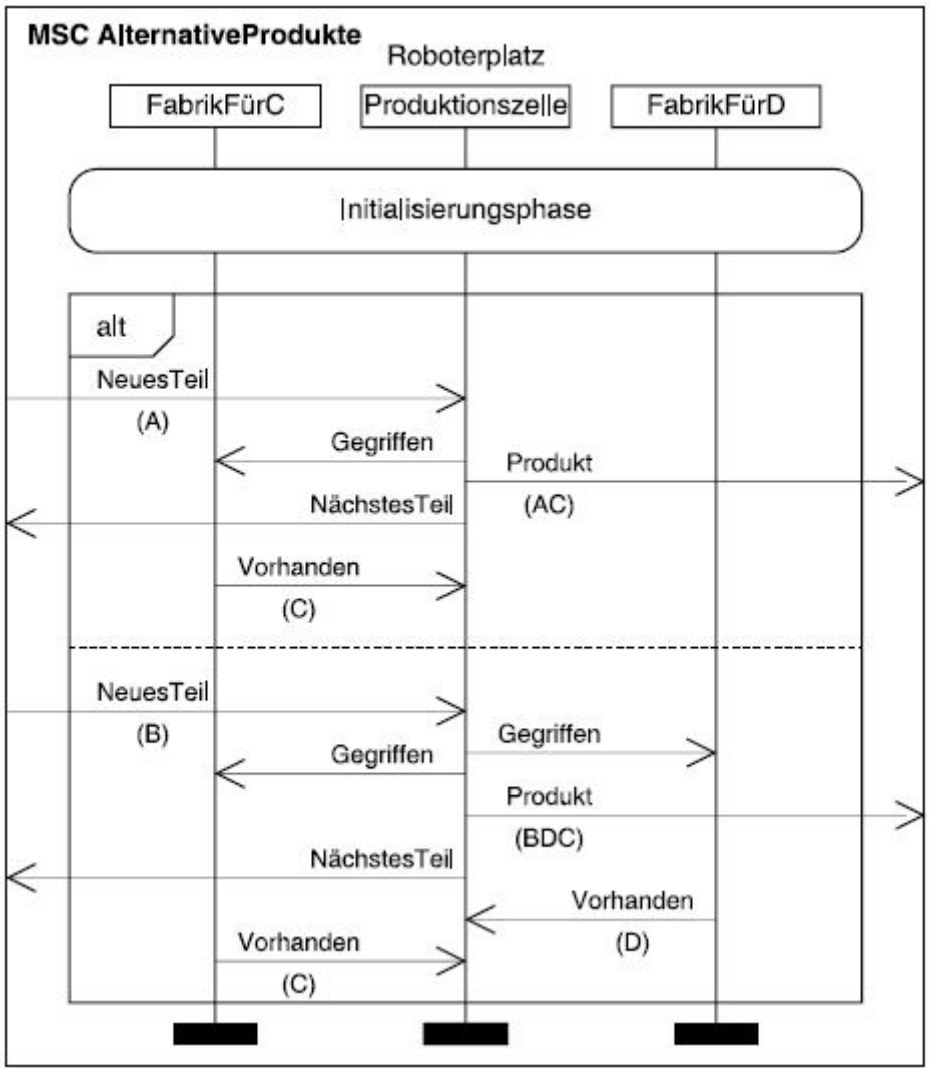

Abbildung 2.8: MSC mit strukturellen Sprachkonstrukten
Quelle : The Specification Languages MSC and SDL Part 1: Message Sequence Chart (MSC)

2.3.2.1 Inline-Expressions

Mit Inline-Expressions können Teilabl¨aufe, die innerhalb eines MSC-Diagramms spezifiziert worden sind, zu komplexeren Abläufen kombiniert werden. Für die Kombination bietet MSC die Operatoren alt, par, loop, opt und exc an. Sie erlauben es, die Wiederholung von Teilabläufen (loop-Operator), alternative Teilabl¨aufe (alt-Operator), die parallele Komposition von Teilabläufen (par-Operator), optionale Teilabläufe (opt-Operator) und Ausnahmen in Form von Teilabläufen (exc-Operator) zu spezifizieren. Graphisch werden Inline-Expressions als Rechtecke mit gestrichelten Linien als Separatoren f¨ur Teilabläufe dargestellt. Der Operator wird in der linken oberen Ecke spezifiziert. Eine Inline-Expression mit einem alt-Operator befindet sich in Abbildung 1.7 . Die alternativen Teilabläufe beschreiben die Fertigung von Produkten der Typen AC und BDC in Abhängigkeit des von der Systemumgebung übergebenen Basisteils (A oder B).

2.3.2.2 References

References ermöglichen es, MSCs in anderen MSCs wieder zu verwenden. Eine Reference referenziert ein anderes MSC über dessen Namen, d. h. eine Reference kann als Platzhalter für das referenzierte MSC angesehen werden. Graphisch werden References durch ein Rechteck mit abgerundeten Ecken dargestellt. Eine Reference befindet sich auch in Abbildung 1.7 . Sie referenziert ein MSC mit dem Namen Initialisierungsphase, das die Initialisierung des Produktionsprozesses beschreibt.

2.3.3 High-Level-MSC

High-Level-MSC (HMSC) erlaubt es, die Kombination von MSCs in Form eines gerichteten Graphen zu beschreiben. Die Knoten eines HMSC-Diagramms sind ein Anfangsknoten, Endknoten, Konnektoren, References und Conditions.
In HMSC konzentriert man sich auf die Darstellung der Kombination von MSC-Diagrammen und abstrahiert von den Instanzen und dem Messagefluss. HMSCDiagramme werden häufig auch Roadmaps genannt. Mit ihnen lässt sich die sequentielle, parallele und alternative Kombination von MSCs in einer sehr intuitiven Form beschreiben.

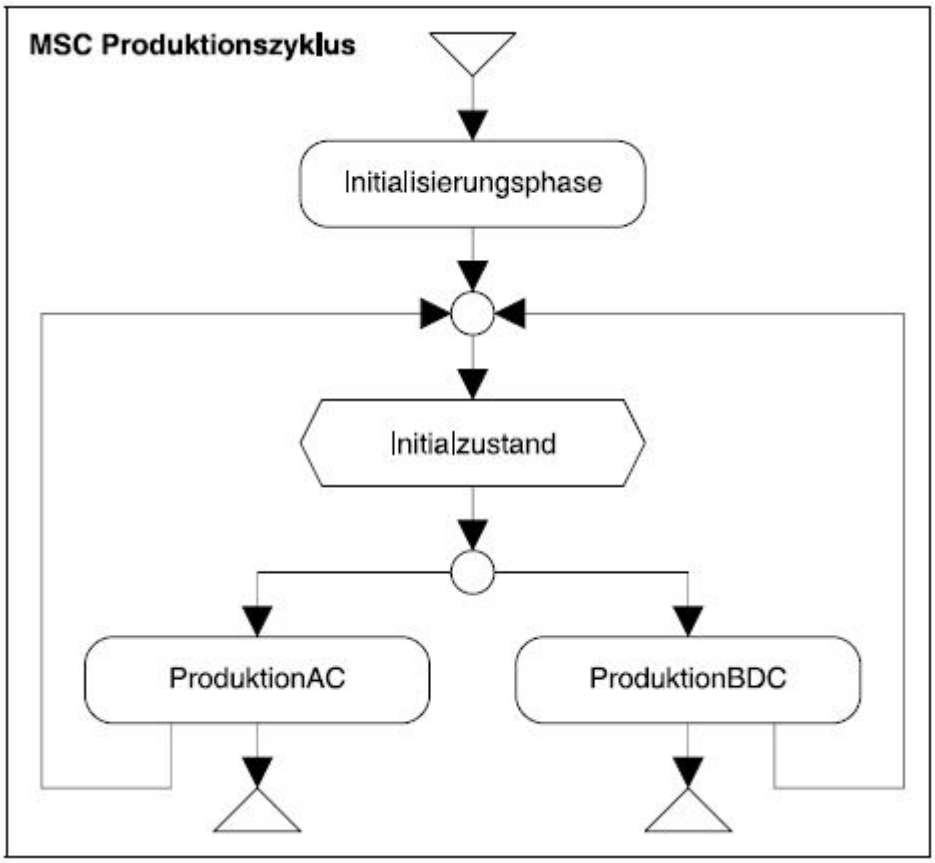

Abbildung 2.9: Ein High-Level-MSC
Quelle : The Specification Languages MSC and SDL Part 1: Message Sequence Chart (MSC)

Der HMSC in Abbildung 1.8 beschreibt die erwarteten AblÄufe des Produktionsprozess-Beispiels: Nach der Initialisierungsphase befindet sich der Produktionsprozess in seinem Initialzustand. Im Initialzustand können entweder Produkte vom Typ AC oder vom Typ BCD hergestellt werden. Nach der Herstellung eines Produkts befindet sich der Produktionsprozess wieder im Initialzustand und ein neues Produkt kann gefertigt werden, oder der Produktionsprozess endet.

2.3.4 Weitere Sprachkonstrukte

In diesem Artikel konnten aus Platzgründen nur die wichtigsten Elemente der MSC-Sprache vorgestellt werden. Neben den genannten Konstrukten enthält MSC noch einige weitergehende Konzepte, die MSC zu einer vollständigen Spezifikationssprache machen: Die zu einer Spezifikation gehörenden MSC-Diagramme können in einem MSC-Dokument gesammelt und strukturiert werden. MSC besitzt keine eigene Datensprache, aber eine allgemeine Datenschnittstelle, die es erlaubt, Datenbeschreibungen aus anderen Sprachen wie z. B. C, C++, SDL oder Java zu benutzen. Zur Spezifikation von Realzeitanforderungen können absolute Zeitpunkte und Zeitintervalle in MSC-Diagrammen spezifiziert werden. Weiterhin wurden UML-Konzepte zur objektorientierten Modellierung, wie z. B. Control-Flow und Procedure-Calls, übernommen.

2.4 Schluss

2.5 Referenz

3 Bewertungskriterien

3.1 Einführung

Im folgenden werden Kriterien vorgestellt, die helfen, eine Modellierungssprache zu bewerten. Dabei handelt es sich um Eigenschaften und deren Ausprägung in einer Modellierungssprache. Sie können auf die Modellierungssprache in ihrer Gesamtheit, aber auch auf einzelne Bestandteile oder Sprachmittel angewendet werden. In vielen Fällen bestehen zwischen den einzelnen Eigenschaften enge Beziehungen. Je nachdem für welchen Sprachbestandteil die Bewertung vorgenommen wird, können andere Maße für die Bewertung verwendet werden. Für viele der hier vorgestellten Merkmale gibt es jedoch keine exakten Bewertungsverfahren, inwieweit das Merkmal erfüllt ist oder nicht.
Grundsätzlich werden die Kriterien in drei Kategorien eingeteilt: die formalen, anwenderbezogenen und anwendungsbezogenen Kriterien. Die folgenden drei Abschnitte beschreiben diese Kriterien genauer.

3.2 Formale Kriterien

Die formalen Kriterien werden in die fünf Einzelkriterien Korrektheit, Vollständigkeit, Einheitlichkeit, Redundanzfreiheit und Strukturierbarkeit unterteilt.

3.2.1 Korrektheit

Eine Modellierungssprache genügt dem Kriterium der Korrektheit, wenn sie unzulässige Modelle eindeutig identifiziert und gleichzeitig erlaubt, die Menge aller zulässigen Modelle zu generieren. Zu unterscheiden ist dabei die syntaktische Korrektheit sowie die semantische Korrektheit der Modelle. Die semi-formalen Prozessmodellierungssprachen, welche für die Prozessdokumentation in Frage kommen, besitzen in der Regel keine eindeutig festgelegte Semantik, sodass hier vor allem die Möglichkeit der automatischen Überprüfung einer korrekten Syntax des Modells im Vordergrund steht. Das Kriterium der Korrektheit steht in enger Verbindung zum Grundsatz der Richtigkeit der Grundsätze ordnungsgemäßer Modellierung.

3.2.2 Vollständigkeit

Unter Vollständigkeit wird in diesem Zusammenhang die Vollständigkeit der Sprachbeschreibung verstanden. Alle Konstrukte sowie die Bedingungen ihrer Verwendung, die von der Modellierungssprache bereitgestellt werden, müssen eindeutig definiert und beschrieben sein (Süttenbach und Ebert 1997, S. 3).

3.2.3 Einheitlichkeit

Das Kriterium der Einheitlichkeit ist angelehnt an den Grundsatz der Klarheit der GoM. Einheitlichkeit bedeutet in diesem Kontext, dass alle Konstrukte der Sprache verständlich dargestellt und beschrieben werden. Ähnliche Konstrukte sollten somit auch in ähnlicher Weise spezifiziert werden.

3.2.4 Redundanzfreiheit

Die Redundanzfreiheit setzt voraus, dass die Sprache keine Mehrfachdefinition von Konstrukten vornimmt, die denselben Sachverhalt beschreiben. Ein und derselbe Sachverhalt sollte also nicht mit mehreren verschiedenen Elementen bzw. Symbolen der Modellierungssprache belegt sein. Auch dieses Kriterium fördert den Grundsatz der Klarheit. Indem gleiche realweltliche Sachverhalte in der Modellierungssprache durch gleiche Konstrukte ausgedrückt werden, wird zudem der Grundsatz der Vergleichbarkeit gefördert.

3.2.5 Strukturierbarkeit

Da Informationsmodelle eine hohe Komplexität aufweisen können, sollte die Sprache Konstrukte bereitstellen, welche die Strukturierung der modellierten Informationen unterstützt. Die Modellierungssprache muss also Zerlegungen in Prozesskomponenten oder Teilprozesse darstellen können. Die Schnittstellen zwischen den Komponenten müssen übersichtlich dargestellt werden. Verallgemeinerungen (Generalisierung) und Detaillierungen (Spezialisierung) müssen schlüssig nachvollziehbar sein. Die Strukturierung in Teilmodelle ermöglicht gleichzeitig die Wiederverwendbarkeit der Strukturen in anderen Modellen. So können etwa modellierte Teilprozesse in anderen Prozessen wieder aufgegriffen werden und müssen nicht mehrfach modelliert werden. Zusätzlich wird dadurch die Wartbarkeit der Modelle verbessert, da Änderungen in einem Modellteil, die Konsistenz der übrigen Modellteile nicht beeinflussen. Das Kriterium der Strukturierbarkeit unterstützt den Grundsatz des systematischen Aufbaus der Grundsätze ordnungsgemäßer Modellierung

3.3 Anwenderbezogene Kriterien

Modellierungssprachen bieten die Möglichkeit, Ideen und Gedanken mittels Modelle austauschen zu können. Modellierungssprachen sind Sprachen, die primär für menschliche Anwender als Mittel zur Kommunikation bestimmt sind. Die anwenderbezogenen Kriterien beleuchten dieses Verhältnis zwischen Modellierungssprache und Anwender. Die anwenderbezogenen Kriterien beschreiben das Verhältnis zwischen dem Anwender und der verwendeten Modellierungssprache. Der Anwender kann zum einen der Modellierer und zum anderen der Betrachter eines Modells sein. Die anwenderbezogenen Kriterien besitzen eine besondere Bedeutung, um Nutzern von Modellen das Verständnis zu erleichtern bzw. zu ermöglichen. Für den betrachteten Anwendungsfall der Prozessdokumentation, zur Nutzung als allgemeine Arbeitsgrundlage für die Mitarbeiter einer Organisation, spielen sie daher eine sehr große Rolle. Die Modellierungssprache ermöglicht das Erstellen von Modellen, um Ideen und Gedanken zwischen den Beteiligten auszutauschen. Sie sind somit in erster Linie für den menschlichen Anwender als Kommunikationsmittel zu verstehen.

3.3.1 Anschaulichkeit und Verständlichkeit

Hier geht es einerseits um eine dem Modellbegriff schon fast inhärente Forderung: Ein Modell sollte anschaulich sein. Andererseits ist hier an den Umstand zu denken, dass Modelle der Abbildung faktischer oder geplanter Realität dienen. Solche Abbildungen können grundsätzlich fehlbar sein. Ein Modell sollte seine Überprüfung an der Realität unterstützen.

Ein Modell ist dann anschaulich, wenn es möglichst direkt mit Wahrnehmungsmustern und Konzeptualisierungen des Betrachters korrespondiert. Wahrnehmungsmuster und Konzeptualisierungen streuen aber bekanntlich interpersonell und intrapersonell (der Mensch ist lernfähig). Anschaulichkeit in diesem Sinn hängt einerseits den jeweils gewählten Abstraktionen und Bezeichnern ab, andererseits von der jeweiligen Modellierungssprache (worauf noch einzugehen sein wird). Die Beurteilung der Beziehung zwischen Modell und Realität liegt in der Regel allein bei den Betrachtern. Die Anschaulichkeit eines Modells ist also grundsätzlich geeignet, seine Überprüfbarkeit zu fördern. Dabei ist es wesentlich, dass das Verhältnis des Modells zur Realität durch möglichst genaue Abbildungsvorschriften erläutert wird.

3.3.1.1 Konsequenzen für Modellierungssprachen

Da Wahrnehmungsmuster und Konzeptualisierungen weit und in subtiler Weise streuen, sind generelle Aussagen über die Wirkung einer Modellierungssprache auf die Anschaulichkeit eines Modells kaum möglich. Allgemein lässt sich vermuten, dass Syntax und Semantik einer Modellierungssprache tendenziell dann als anschaulich empfunden werden, wenn sie eine deutliche Nähe zu Sprachmitteln aufweisen, die der jeweilige Betrachter aus ihm vertrauten Sprachen kennt - also beispielsweise das generelle Muster SSubjekt, Prädikat, (Objekt)". Der Stand der Forschung in diesem Bereich muss allerdings als unbefriedigend angesehen werden. Es gibt nur wenige empirische Untersuchungen, die die Anschaulichkeit von konzeptuellen Modellen aus der Sicht verschiedener Betrachter zum Gegenstand haben (so etwa [GoSt90], [Hit95]). Dabei hat sich gezeigt, dass viele Betrachter erhebliche Schwierigkeiten hatten, die ihnen vorgelegten Modelle nachzuvollziehen. Ein solches Ergebnis kann kaum überraschen: Schließlich wurden Modellierungssprachen in der Regel nicht nach den Wünschen möglicher Anwender entworfen. Wir können an dieser Stelle lediglich zweierlei festhalten: Je mehr Diagrammarten geboten werden, desto größer die Wahrscheinlichkeit, dass eine Diagrammart im Kreis der Betrachter intuitiv verstanden wird. Je stärker eine Diagrammart die spezifischen Besonderheiten einer Domäne berücksichtigt, desto größer ist die Wahrscheinlichkeit, dass die Kenner dieser Domäne entsprechende Diagramme für anschaulich halten. Anschaulichkeit impliziert sinnvollerweise das Verstehen der verwendeten Modellierungssprache. Das empfiehlt die Forderung nach leichter Erlernbarkeit: Je geringer die Zahl der Modellierungskonzepte und je einfacher die zu berücksichtigenden Verknüpfungsregeln, desto leichter ist eine Sprache zu erlernen. Dabei muss allerdings berücksichtigt werden, dass diese Anforderung ggfs. im Konflikt mit dem Bemühen um Anschaulichkeit steht.

3.3.1.2 Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Auch wenn Modellierungssprachen unzweifelhaft einen erheblichen Einfluss auf die Anschaulichkeit von Modellen haben, ist uns die Erfassung und Beurteilung dieses Einflusses in befriedigender Weise nicht möglich. Wir können lediglich auf einige Indikatoren hinweisen. Die verschiedenen Diagrammarten verbessern tendenziell die Chance, individuell als anschaulich empfundene Sichten zu befriedigen. Diesem Vorteil steht allerdings ggfs. der Nachteil redundanter Sprachelemente gegenüber, wodurch die Erlernbarkeit der gesamten Sprache verschlechtert wird. Weiterhin sind domänenspezifische Diagrammarten geeignet, Anschaulichkeit für bestimmte Betrachtergruppen zu fördern. Als Beispiel sei eine Diagrammart zur Abbildung von Geschäftsprozessen im Rahmen der Entwicklung betrieblicher Informationssysteme genannt.

Darüber hinaus sind Symbole zur Kennzeichnung ergänzender Kommentare wünschenswert. Die Erlernbarkeit einer Sprache wird durch die Dokumentation erleichtert. Neben der verständlichen Beschreibung der Sprachkonzepte ist hier an die Erläuterung der Verwendung der Sprache zu denken - ggfs. durch Beispiele ergänzt.
Auch die Verständlichkeit der Sprache wirkt sich direkt auf die Anschaulichkeit der Sprache aus. Die Verwendung der einzelnen Sprachmittel sollte klar festgelegt sein, und es sollten Entscheidungshilfen bei Alternativen gegeben werden. Auch die Eindeutigkeit der Semantik der Sprachmittel ist eine wichtige Voraussetzung für eine verständliche Sprache.

3.3.2 Angemessenheit

Das Kriterium der Angemessenheit steht in direktem Zusammenhang mit der Mächtigkeit und dem Anwendungszweck einer Sprache. Die Angemessenheit ist immer relativ zum Anwendungszweck zu sehen. Sie bezieht sich sowohl auf die Sprache in ihrer Gesamtheit als auch auf die einzelnen Sprachkonzepte. Angemessenheit und Mächtigkeit bieten zusammen die Möglichkeit, zu untersuchen, ob eine Sprache für den Anwendungszweck ausreichende Konzepte bereitstellt, gleichzeitig aber keine überflüssigen Konzepte enthält.

3.3.2.1 Konsequenzen für Modellierungssprachen

Insbesondere bei sehr speziellen Sprachkonstrukten muß untersucht werden, ob diese Konstrukte nötig sind oder ob sie nur zu einer Erhöhung der Sprachkomplexität führen. Die Entscheidung, ob ein Sprachmittel angemessen ist oder nicht, ist jedoch häufig nicht objektiv vornehmbar, sondern hängt stark von den Präferenzen der jeweiligen Betrachter ab. Im allgemeinen gibt es immer Vor- und Nachteile für einzelne Sprachmittel, so dass ein sorgsames Abwägen notwendig ist.

3.3.2.2 Feststellbarkeit/Meßbarkeit der Beurteilungskriterien

Generelle formale Maße zur Bestimmung der Angemessenheit von Modellierungssprachen gibt es nicht. Die Untersuchung der Angemessenheit eines Sprachelementes kann daher nur subjektiv erfolgen. Dabei ist zu untersuchen, ob ein Sprachelement bedeutsam für den Anwendungszweck ist oder nicht.

3.3.3 Überprüfbarkeit

Modelle dienen der Abbildung faktischer oder geplanter Realität. Solche Abbildungen können grundsätzlich fehlbar sein. Ein Modell sollte seine Überprüfung an der Realität unterstützen. Der Einfluss der Modellierungssprache auf die Überprüfbarkeit eines Modells ist ambivalent. So geht es einerseits darum, eine möglichst genaue, intersubjektiv nachvollziehbare Überprüfung an der Wirklichkeit anzustreben. Andererseits ist zu berücksichtigen, dass solche objektivierten Verfahren nicht für alle Facetten eines Modells anwendbar sind und sich Modelle oft nicht auf faktische, sondern geplante Realität beziehen. In diesen Fällen erfolgt die Überprüfung allein durch die Einschätzung der Betrachter. Idealtypisch ist ein Modell dann als realistisch (im Hinblick auf Erwartungen) anzusehen, wenn im Kreise gutwilliger und sachkundiger Betrachter ein Konsens darüber erzielt wurde. Während die objektivierte Überprüfung eines Modells an der Realität tendenziell zur Forderung nach formalen Modellierungssprachen führt, ergeben sich aus

dem Bemühen um Konsensbildung andere Konsequenzen: Bekanntlich fördert Mehrdeutigkeit die Chance, einen Konsens zu erzielen. Ein Modellierungsansatz wie die SSoft System Methodology"([Che81]) lässt den Betrachtern eben wesentlich mehr individuelle Interpretationsspielräume als eine Sprache zur formalen Systemspezifikation. Wegen dieser widersprüchlichen Konsequenzen ist das Kriterium "Überprüfbarkeit"nicht zur Diskriminierung zwischen Modellierungssprachen geeignet.
Ein Modell ist eine vereinfachte und idealisierte Abbildung eines Problembereichs. Im Modell werden nur die für die Betrachtung wesentlichen Eigenschaften berücksichtigt und unwesentliche vernachlässigt. In Abb. ist die Beziehung zwischen Modell und zu modellierendem Sachverhalt dargestellt.

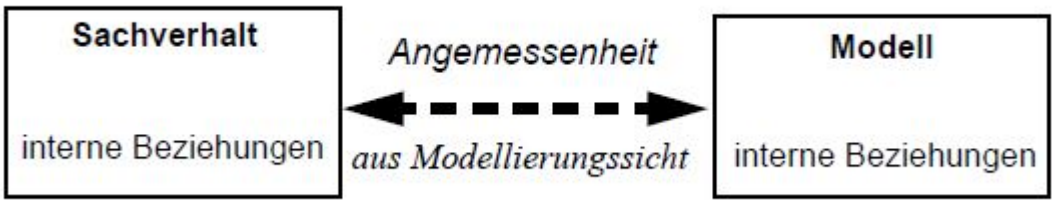

Abbildung 3.1: Beziehung zwischen Modell und zu modellierendem Sachverhalt
Quelle : EIN BEZUGSRAHMEN ZUR BEURTEILUNG OBJEKTORIENTIERTER MODELLIERUNGSSPRACHEN - VERANSCHAULICHT AM BEISPIEL VON OML UND UML , ULRICH FRANK und MICHAEL PRASSE

Modelle bedürfen immer der Überprüfung mit dem zu modellierenden Sachverhalt. Zwischen Sachverhalt und Modell existiert immer eine Beziehung, die ein Maß für die Angemessenheit des Modells ist. Die Bewertung der Angemessenheit eines Modells kann nur relativ zum Modellierungszweck erfolgen, da Modelle von unwesentlichen Details abstrahieren. Im allgemeinen kann diese Beziehung nicht direkt angeben werden, da der Sachverhalt meistens nur unklar spezifiziert ist1. Aus diesem Grund ist es auch nicht möglich, immer automatisch zu überprüfen, ob ein Modell den gewünschten Sachverhalt modelliert. Nur wenn der Sachverhalt selbst als exakte Beschreibung2 vorliegt und auch das Modell exakt beschrieben ist, wird ein formales Überprüfen erst möglich, und die Angemessenheit eines Modells kann formal bewertet werden. Meistens ist der Sachverhalt jedoch nicht klar spezifiziert.
Die einzige Möglichkeit ist dann, den Inhalt des Modells zu erfassen und zu überprüfen, ob dieser mit dem zu modellierenden Sachverhalt übereinstimmt. Je exakter und präziser ein Modell interpretiert werden kann, desto leichter kann es mit dem zu modellierenden Sachverhalt verglichen werden. Obwohl es nicht möglich ist, die Adäquanz zwischen einer eventuell unpräzisen Darstellung (Sachverhalt) und einer möglichst präzisen Darstellung (Modell) prinzipiell zu entscheiden, hilft eine präzise Darstellung und eindeutige Interpretation des Modells beim Vergleich mit dem zu modellierenden Sachverhalt. Eine Modellierungssprache, die es ermöglicht, Modelle exakt und eindeutig zu beschreiben und zu interpretieren, erleichtert daher wesentlich die Überprüfung des Modells.
Nur wenn Modelle eindeutig sind, ist eine korrekte Verständigung möglich. Dies verlangt eine eindeutige Interpretation der einzelnen Sprachmittel und Regeln zur syntaktischen Verknüpfung dieser Sprachmittel. Andernfalls bieten sich Interpretationsfreiräume, die zu Missverständnissen führen können. Die exakte Festlegung der abstrakten Syntax und Semantik der Sprache ist somit Voraussetzung für die eindeutige Interpretierbarkeit der

Modelle. Beim Betrachten eines Artefakts der Modellierungssprache hat man im allgemeinen nicht wie bei Programmen die Möglichkeit, durch Inspizieren oder Ausprobieren des Codes die Aufgabe der Software herauszubekommen. Vielmehr ist die Beschreibung der Metasprache die einzige Referenz.
Ein anderer Aspekt der Überprüfung betrifft die Nachvollziehbarkeit und Rückverfolgung von Modellierungsentscheidungen.

3.3.4 Mächtigkeit

Die Mächtigkeit einer Modellierungssprache bezieht sich unter anderem auf die Ausdrucksmächtigkeit der bereitgestellten Konzepte. Eine Modellierungssprache sollte sich auf die Modellierung konzentrieren. So können zu viele Konzepte zu einer unangemessen mächtigen Sprache führen. Zwischen Mächtigkeit und Angemessenheit muss daher ein ausgewogenes Verhältnis existieren.
Mächtigkeit kann aber auch relativ zur Mächtigkeit von Turingmaschinen betrachtet werden und ist dann ein Maß für die Berechnungsfähigkeit einer Modellierungssprache. Für Modellierungssprachen ist diese Betrachtung jedoch nicht ganz korrekt, da sie im allgemeinen keine Formalisierungen von Algorithmen sind.

3.3.4.1 Konsequenzen

Mächtigkeit und Angemessenheit bedingen einander. Die Mächtigkeit einer Sprache muss daher relativ zum Anwendungsbereich betrachtet werden. Geht man grundsätzlich davon aus, dass die hier betrachteten Modellierungssprachen der konzeptuellen Modellierung dienen, misst sich die Mächtigkeit einerseits an der Möglichkeit, als wesentlich erkannte Sachverhalte des abzubildenden Realitätsbereichs natürlich, also ohne aufwendige Rekonstruktionen, beschreiben zu können. Andererseits sollte die Modellierungssprache auch Möglichkeiten bieten, Informationen darzustellen, die für die Implementierung benötigt werden.

3.4 Schluss

Eine strukturierte Liste von Beurteilungskriterien lässt sich daraus schwer ableiten.

3.5 Referenz

[Süttenbach und Ebert 1997]: Süttenbach, Roger; Ebert, Jürgen (1997): A Booch Metamodel. Institut für Informatik, Fachbereich Informatik, Universität Koblenz-Landau. Koblenz (Fachberichte Informatik, 5). Online verfügbar unter http://www.uni-koblenz.de/ ist/-documents/Suettenbach1997ABM.pdf, zuletzt geprüft am 12.10.2015. [Obermeier et al. 2014]: Obermeier, Stefan; Fischer, Herbert; Fleischmann, Albert; Dirndorfer, Max (2014): Geschäftsprozesse realisieren. Ein praxisorientierter Leitfaden von der Strategie bis zur Implementierung. 2., aktual. Aufl. Wiesbaden: Springer Vieweg (SpringerLink). [Fischer et al. 2006]: Fischer, Herbert; Fleischmann, Albert; Obermeier, Stefan (2006): Geschäftsprozesse realisieren. Ein praxisorientierter Leitfaden von der Strategie bis zur Implementierung. 1. Aufl. Wiesbaden: Friedr.Vieweg und Sohn Verlag/GWV Fachverlage GmbH Wiesbaden (Aus dem Bereich IT erfolgreich nutzen).

[GoSt90] Goldstein, R.C.; Storey, V.C.: Some Findings on the Intuitiveness of Entity Relationship Constructs. In: Lochovsky, F.H. (Ed.): Entity Relationship Approach to Database

Design and Query. Amsterdam: Elsevier 1990
"[Hit95]"Hitchman, S.: Practitioner Perceptions on the Use of some Semantic Concepts in the Entity Relationship Model In: European Journal of Information Systems, Vol. 4, 1995, pp. 31-40
"[Che81]"Checkland, P.: Systems thinking, systems practice. Chichester et al.: Wiley 1981

4 Spezifische Kriterien von Sprachen

4.1 Anwendungsbezogene Kriterien

Anwendungsbezogene Kriterien beschreiben die Nutzung von Modellierungssprachen aus Sicht des Anwenders idR. abhängig von ihrem Domänenspezifischen Einsatz. So besitzen verschiedene Modellierungssprachen einen unterschiedlich großes Nutzungspotenzial im jeweiligen Bereich. Man spricht in diesem Zusammenhang auch von der Mächtigkeit der Sprache. Jedoch ist zu beachten, das Anwendungsbezogene Kriterien und Anwenderbezogene Kriterien konkurrieren können, da sich beispielsweise eine leicht erlernbare Sprache sich negativ auf die Mächtigkeit auswirken könnte und umgekehrt.

4.1.1 Nutzungspotenzial

Das Nutzungspotenzial gibt Aussagen darüber, wie gut die verwendete Sprache Eigenschaften in einem darzustellenden Sachverhalt beschreibt. Darunter fällt wie Präzise diese Aussagen sind und wie hoch der Detailgrad der darzustellenden Eigenschaft ist. Je größer dieser Sprachumfang ist, desto größer ist auch die Mächtigkeit der Sprache. Nun ist es schwierig eine Aussage darüber zu treffen, inwiefern eine Sprache für eine Anwendungsdomäne, wie der von Telekommunikationsprozessen, eine vollständige Beschreibung der Konzepte stellt.

4.1.2 Funktionalität

5 Bewertung der Eignung von Modellierungssprachen

5.1 Einführung)

5.2 UML)

UML ist heute eine der dominierenden Sprachen für die Modellierung von betrieblichen Anwendungs- bzw. Softwaresystemen. Der erste Kontakt zu UML besteht häufig darin, dass UML-Diagramme im Rahmen von Softwareprojekten zu erstellen, zu verstehen oder zu beurteilen sind. UML-Diagramme gelten als Standard bei objektorientierter Modellierung.

5.2.1 Eignung

Aus meiner persönlichen Sicht liegen die wesentlichen Vorteile bei dem Einsatz von UML in der breiten Unterstützung sämtlicher objektorientierter Grundsätze. Man kann an vielen Stellen ein und dasselbe Problem auf unterschiedliche Art und Weise lösen, ohne sich dabei durch die Vorgaben der Sprache eingeengt zu fühlen. Die Sprache lässt also den Softwareentwicklern den Freiraum, nach eigenen Vorstellungen zu modellieren. Besonders Vorteilhaft erscheint die Tatsache, dass UML auch die Erweiterungsmöglichkeiten durch eigene Sprachkonstrukte vorsieht (Metamodell), und damit wirklich jedem das Recht anbietet, den eigenen Notationsrahmen zu erschaffen. Diese Möglichkeit sollte aber nur in äußersten Notfällen (wenn es nicht anders geht) verwendet werden, da man sonst der Gefahr, sich vom Standard zu entfernen, entgegenläuft . Weiterer Vorteil liegt in der Standardisierung der UML. Die UML bietet eine fast perfekte Grundlage für die Kommunikation der verschiedenen Entwicklungsteams miteinander. Diagramme können nun nicht nur von den Fachleuten der Software –Firma , sondern auch von den außerhalb stehenden verstanden und verbessert werden. Lästiges Einarbeiten in die verschiedenen Notationen entfällt, falls sich alle grundsätzlich an die UML halten. Weiterer Vorteil liegt in der zunehmenden Verbreitung der Unified Modeling Language. Die exakten Zahlen sind zwar noch schwer abzuschätzen, es zeichnet sich jedoch ein wachsender Trend für den Einsatz der UML ab. Man kann die Vorteile von UML in vier wichtigste Punkte zusammenfassen wie folge:

- Die Vereinheitlichung der Terminologie und die Standardisierung der Notation führen zu einer massiven Erleichterung der Verständigung zwischen allen Beteiligten. - Die UML wächst mit Ihren Anforderungen an die Modellierung. Sie können mit der Erstellung einfacher Modelle beginnen, aber auch sehr komplexe Sachverhalte im Detail modellieren, da die UML eine mächtige Modellierungssprache ist. - Die UML baut auf bewährten und weit verbreiteten Ansätzen auf. Die UML wurde nicht im Elfenbeinturm erstellt, sondern hat sich zu grossen Teilen aus der Praxis und aus bestehenden Modellierungssprachen heraus entwickelt. Das gewährleistet die Einsatzfähigkeit und Praxisnähe der UML.

5.2.2 Uneignung

Die Nachteile lassen sich nun wiederum aus dem Umfang der Sprache ableiten. UML ist sehr vielfältig, so daß auch am Anfang sehr viel Aufwand für das Aneignen und Verstehen sämtlicher Sprachkonstrukte aufgebracht werden muß. Außerdem wird man in der Regel feststellen, dass viele Elemente sehr selten zum Einsatz kommen und damit eher als Ballast der Sprache angesehen werden. So habe ich in der Darstellung der UML Notation auf die Erläuterung der OCL Konstrukte (Object Constraint Language) verzichtet, obwohl OCL als formale Sprache zur Erweiterung der Semantik der UML Notation herangezogen werden kann. Damit lassen sich zwar Zusicherungen, Invarianten, Vor –und Nachbedingungen, Navigationspfade etc. kurz und aussagekräftig darstellen, aber man kann diese im begrenzten Maße auch durch UML Basiselemente beschreiben. Speziell für die Entwicklung der Software für eingebettete Systeme lässt sich anmerken, dass UML eine objektorientierte Modellierungssprache ist. Sämtliche Konzepte von UML können nur dann ausgenutzt werden, wenn wirklich objektorientiert programmiert wird. Es macht wenig Sinn, objektorientiert zu modellieren , wenn anschließend kein objektorientierter Code verwendet wird. Steht kein objektorientierter Compiler zur Verfügung, sollte man sich über den Einsatz anderer Werkzeuge und Spezifikationssprachen Gedanken machen. Man kann zwar im begrenzten Maße sämtliche objektorientierte Ansätze in den „strukturierten“ Code konvertieren (also z.B. von C++ nach C überführen), das Ergebnis lässt jedoch zu wünschen übrig, insbesondere leidet die Code- Qualität / Lesbarkeit darunter.

Man kann die Nachteile von UML in vier wichtigste Punkte zusammenfassen wie folge: - Die neue Notation muss zuerst erlernt werden. Es fallen somit Schulungskosten an und die Mitarbeiter müssen dafür Zeit aufwenden können. - Im Bereich der Software-Unterstützung ist noch einiges zu tun, bis wirklich benutzerfreundlich mit UML gearbeitet werden kann. - Durch die Vereinheitlichung der Modellierungssprachen geht die Vielfalt und Kreativität verloren. Kleine, vielleicht gute Modelle gehen verloren. - Der Wettbewerb wird durch die Monopolisierung zerstört.

5.2.3 Fazit

Wir versuchen in diesem Abschnitt die Vorteile und Nachteile von UML zusammenfassen und einen Überblick darüber geben.

Eignung	Uneignung
Standardisierte, weit verbreitete Methode	Grundsätzliches IT-Wissen erforderlich
Ergebnisse der Modellierung in UML können zur Umsetzung in Software-Code verwendet werden. Es ist eine automatisierte Erzeugung eines Coderahmen aus den Diagrammen möglich wodurch der Programmier-Aufwand reduziert wird	Sehr großer Umfang und Komplexität der Sprache (beispielsweise umfasst die Sprachspezifikation mehr als 1200 Seiten) verursacht Schwierigkeiten bei der Benutzung und erfordert einen hohen Einarbeitungsaufwand
Einfacher Modellaustausch möglich und damit bessere Modellwiederverwendung	Die UML wird hinsichtlich semantischer Inkonsistenzen, Konstruktmehrdeutigkeiten, inadäquaten Notationen und kognitiven Unzugänglichkeiten kritisiert
Einheitliche Toolunterstützung	Die UML wurde ursprünglich für die Entwicklung von Softwaresystemen konzipiert, daher sind die relevanten Beschreibungs- und Darstellungsmittel weniger auf die Prozessmodellierung ausgerichtet
Modellierungsmittel für die meisten Probleme im Software Engineering	Konzeptionelle Schwächen, im Requirements Engineering insbesondere bei der Formulierung von Anwendungsfällen
Breites Angebot an Werkzeugen, Büchern und Schulung	UML-Modelle sind Sammlungen von Einzelmodellen: Konsistenzprobleme, Problem des Zusammensuchens relevanter Information
einheitliche Notation für alle Aspekte	
Referenzen und Beziehungen möglich	
Erweiterungsmöglichkeit mittels Profile (Stereotypen/Tags) möglich	

Abbildung 5.1: Eignung und Uneignung von UML

5.3 Schluss

5.4 Referenz

einbinden

Abbildungsverzeichnis

Tabellenverzeichnis

Listings