Eric A. Nyamsi

IT-Lösungen auf Basis von SysML und UML

Anwendungsentwicklung mit Eclipse UML Designer und Eclipse Papyrus

2. Auflage

Springer Vieweg

IT-Lösungen auf Basis von SysML und UML

Eric A. Nyamsi

IT-Lösungen auf Basis von SysML und UML

Anwendungsentwicklung mit Eclipse UML Designer und Eclipse Papyrus

2., korrigierte und aktualisierte Auflage

Springer Vieweg

Eric A. Nyamsi
Karlsruhe, Deutschland

ISBN 978-3-658-40838-1 ISBN 978-3-658-40839-8 (eBook)
https://doi.org/10.1007/978-3-658-40839-8

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Planung/Lektorat: Reinhard Dapper
Springer Vieweg ist ein Imprint der eingetragenen Gesellschaft Springer Fachmedien Wiesbaden GmbH und ist ein Teil von Springer Nature.
Die Anschrift der Gesellschaft ist: Abraham-Lincoln-Str. 46, 65189 Wiesbaden, Germany

Inhaltsverzeichnis

Anwendungen der Modellierung in der Programmierung: Modeling4Programming

1

1.1 Überblick

Heute werden Modellierungen von Software mithilfe von Unified Modeling Language, abgekürzt UML, für die Qualitätssicherung und Wartbarkeit ermöglicht. UML stellt eine Sprache zur Modellierung sowohl von Strukturen als auch von Verhalten der Systeme dar. Mithilfe der Syntax von UML wurde eine neue Sprache zum Modellieren von physischen Systemen entwickelt: Systems Modeling Language, abgekürzt SysML.

Einerseits ist UML zur Modellierung von Software gedacht und andererseits ist SysML zur Modellierung der physischen Systeme geeignet. Es gibt keinen Konflikt zwischen beiden Sprachen. Die erste Sprache wird von der zweiten zum Modellieren bestimmter Systeme mithilfe der Diagramme wie z. B. Zusicherungs- oder Blockdiagramme ergänzt. Das erste Diagramm ermöglicht die Beschreibung physischer Gesetze mittels Formel. Das zweite Diagramm beschreibt Ergebnisse von Prozessen. Modellierungen der parallelen Prozesse physischer Systeme erfordern die Verwendung des Aktivitätsdiagramms von SysML.

Wozu braucht man beide Sprachen? Die Antwort ist einfach: Im Bereich Modellierung der objektorientierten Software ist UML als grafische Modellierungssprache ein Standard geworden. Die Verbindungen zwischen den grafischen Symbolen von UML ermöglichen die Modellierung von Software. UML wird zum Erstellen von Modellen für die Codegenerierung eingesetzt. Wobei diese Sprache zur Modellierung statischer Struktur von Software als Standard eingesetzt ist. Beispielsweise wird das Klassendiagramm von UML zum Modellieren von Klassen und Schnittstellen (Interface) der Systeme benutzt. Im Bereich modellbasierte Entwicklung erlaubt SysML, textuelle Anforderungen mithilfe der Annotationen zu beschreiben. Hierbei lassen sie sich mit anderen Elementen wie z. B. Systemkomponenten oder Interface verbinden. Dies ermöglicht die Nachverfolgbarkeit, genannt *Traceability,* zwischen den

E. A. Nyamsi, *IT-Lösungen auf Basis von SysML und UML,*
https://doi.org/10.1007/978-3-658-40839-8_1

Elementen. Das Anforderungsdiagramm auf Basis von *SysML* ist für das Dokumentieren der Anforderungen an eine Software oder ein System spezifisch. Dieses Diagramm rechtfertigt die Entstehung von *SysML* neben UML, wobei es zum Beschreiben der Anforderungen und deren Nachverfolgbarkeit geeignet ist. Weil die System- und Softwareentwicklung für eingebettete Systeme mit steigender Komplexität konfrontiert sind, stellen UML und SysML wichtige Modellierungssprachen zum Beschreiben von Zusammenhängen dar. Damit sind sie für System- und Softwareentwicklung hilfreich und zukunftsweisend.

Modeling4Programming abgekürzt *M4P* ist das neue Konzept und ermöglicht das Programmieren von Anwendungen mithilfe der Modellierung. Es ist auch ein Trend und basiert auf dem Erstellen der Codes mithilfe der Modellierung. Der Trend *„Vom Modellieren zum Programmieren“* fokussiert sowohl auf die Modellierungssprache UML als auch auf die Programmiersprachen wie z. B. Java oder C++. Hierbei werden Eigenschaften des Klassendiagramms von UML und die Codestruktur der Programmiersprache verwendet. Das heißt, aus einem Klassendiagramm wird ein Programm aus Codes erstellt. Ein Diagramm zeigt mehr als tausend Zeichen. Dies bedeutet, dass aus einem UML-Klassendiagramm viele Codes auf Basis von Java, C# oder C++ produziert werden. Das ist ein Trend!

Design Patterns werden auch mit UML zum Beschreiben der Funktionalität der Software modelliert. Dies erhöht die Akzeptanz der Software sowohl bei den Entwicklern als auch bei den Nutzern. Statt die Software direkt zu entwickeln, soll man sie zuerst mithilfe von UML modellieren. Anschließend beginnt man die Entwicklung der Software mithilfe der modellierten Komponenten. Angesichts dessen wird die innere Qualität geprüft. Eins der Ziele des Einsatzes der Modellierungssprache UML ist, die Qualitätssicherung mittels Modellierung zu gewährleisten.

SysML ermöglicht die Spezifikation der Komponente sowohl von Hard- als auch von Software, wobei sich SysML-Softwaremodelle durch UML-Modelle erweitern lassen. Bei der modellbasierten Entwicklung werden beide Sprachen zum Realisieren der Produkte kombiniert. Diese ermöglicht die Erstellung sowohl von Diagrammen als auch von ausführbaren Codes. Die modellgetriebene Entwicklung, genannt *Model Driven Development (MDD),* stellt eine moderne Entwicklungsmöglichkeit für objektorientierte Software dar. Hierbei wird von einem formalen Modell ausgehend die Software entwickelt. Open Source Frameworks wie z. B. *Eclipse Modeling Framework (EMF), Papyrus Framework* oder *Obeo UML Designer* unterstützen die MDD durch Codegeneratoren. Beispielsweise kann ein Klassendiagramm aus EMF mithilfe von Ecore Framework direkt in den Java-Quellcode übersetzt werden.

Sowohl Eclipse UML Designer als auch Eclipse Papyrus ermöglichen die professionelle objektorientierte Softwareentwicklung auf Basis von UML. Dieses Buch zeigt, wie man mithilfe von Eclipse UML Designer und Eclipse-Papyrus-Software oder Systemen auf Basis von UML und SysML modelliert.

Eine Software zu modellieren bedeutet, dass die Entwickler Teile der Software mithilfe standardisierter Tools beschreiben, damit sie effizient ist. Die Tools zur Beschreibung

eines Systems oder einer Software stellen moderne Modellierungssprachen dar: SysML und UML sind bekannte Modellierungssprachen. Der Titel *IT-Lösungen auf Basis von SysML und UML* deutet auf die Beschreibung von Software durch Modelle hin. SysML ist ein *Erbe* von der Modellierungssprache UML und erbt die Modellelemente und Diagrammtypen. Allerdings verbessert das SysML-Profil der UML die Semantik einiger Elemente. Das Kernelement von SysML ist der *Block,* welcher die Struktur zur objektorientierten Programmierung darstellt. Ein Block repräsentiert Komponente wie z. B. Softwareartefakte, Hardware, Funktionen, Prozesse oder auch Personen. Mithilfe der objektorientierten Modellierung werden Klassen oder Kompositionsstrukturdiagramme zu Blockdiagrammen bzw. zu internen Blockdiagrammen umgewandelt. Die aus der UML bekannten Ports wurden als Flowports für den Transport von Stoffen oder physikalischen Eigenschaften übernommen. Das Aktivitätsdiagramm wird um physikalische Objektflüsse und Aktivitätsparameter ergänzt. Andere Diagrammformen wie Use-Case-Diagramm, Sequenzdiagramm oder Zustandsdiagramm sind auch geerbt. SysML stellt sowohl einige aus UML bekannte Sprachelemente als auch Sprachkonstrukte in Bezug auf das Requirements Engineering. Damit lassen sich nicht nur funktionale Anforderungen mithilfe von Use Cases oder Interaktionsszenarien modellieren, sondern auch nichtfunktionale Anforderungen darstellen. Anforderungen (Requirements) sind in SysML ein wichtiges Sprachkonstrukt. Sie stellen Klassensymbole mit dem Stereotyp *Requirement* dar und können wie Klassensymbole eigene Strukturen (Elemente) haben, in denen insbesondere eine ID oder der Text der Anforderung selbst stehen kann.

SysML und UML ermöglichen die Anwendungen der Modellierung in der Programmierung und stellen damit den neuen Begriff *Modeling4Programming* dar. Zum einen ermöglichen diese Tools den Entwicklern die Beschreibung der Modellierungsdiagramme mithilfe der standardisierten Annotationen in einer Programmiersprache wie z. B. Java. Zum anderen ermöglichen die Modellierungsdiagramme mit SysML/UML die Spezifizierung der Systeme und der Software.

Das Buch fokussiert auf die grafische Spezifizierung der Systeme und Software mithilfe der Modellierungssprachen SysML und UML und deren Anwendung in der Programmierung. Die Modellierung soll das Programmieren der Komponente der Systeme ermöglichen. Wollen die Entwickler eines Transistors wie z. B. IGBT die Verluste mithilfe der Modellierung der Komponente bestimmen und damit reduzieren, werden die Teile des Systems mit SysML/UML modelliert, anschließend werden mithilfe einer Programmiersprache wie z. B. Java die Verluste mit der objektorientierten Programmierung erfasst. Die Modellierung hat einen Sinn, wenn ihre Anwendung in der objektorientierten Programmierung, das Verstehen der Funktionalität des Systems oder der Software erfolgt. Mit der Systems Modeling Language (SysML) stellte die Object Management Group (OMG) [1] eine formale Modellierungssprache für die Entwicklung eingebetteter Systeme mit Softwareanteilen zur Verfügung.

Das Buch gibt einen Überblick über Entwicklung von Tools auf Basis von Open Source, Eclipse-Papyrus und UML-Designer. Diese Modellierung-Frameworks ermöglichen die Beschreibung der Systeme oder Software mit SysML/UML bzw.

UML und damit deren Anwendungen in der Programmierung, wobei das Buch auf die Programmiersprachen Java und C++ fokussiert. Hierbei werden Diagrammbeschreibungen der Modellierung mithilfe der Java- und C++-Klassen in der Programmierung angewendet.

Ziel der Modellierung mit beiden Frameworks ist es, die modellbasierte Entwicklung der Systeme mithilfe der Modellierungsdiagramme von SysML/UML zu realisieren. Das Buch kombiniert die Vorteile von SysML und UML mithilfe der Frameworks Eclipse-Papyrus und UML-Designer.

Der Begriff *Modeling4Programming* enthält zwei Elemente: Modellierung mit SysML/UML und Programmierung mit objektorientierter Programmiersprache wie z. B. Java und C++. Das Kernthema des Buches ist die Kombination SysML/UML und Java/C++. Deshalb erklärt die Modellierung mittels Diagrammen die Beschreibung der Programmierung. Die Modellierung soll den Inhalt der Programmierung darstellen. Die Frameworks Papyrus und UML-Designer ermöglichen mithilfe der Diagramme die Realisierung der modellbasierten Entwicklung. Das Buch stellt mithilfe der Frameworks Papyrus und Obeo UML Designer Abhängigkeitsbeziehungen wie z. B. in Klassendiagrammen, Kontextdiagrammen und ggf. Datenflussdiagrammen dar.

1.2 Papyrus-Framework zur Modellierung mit SysML/UML

Papyrus ist ein erweiterbares Framework, das sowohl bestehende Modellierungssprachen à la UML 2.5 und SysML 1.4 unterstützt als auch die Erstellung eigener domänenspezifischer Modellierungssprachen erlaubt. Durch diese Anpassbarkeit deckt Papyrus potenziell verschiedenste Entwicklungsaspekte ab, darunter z. B. modellbasierte Simulationen, modellbasierte Tests und Sicherheitsanalysen. Damals war Papyrus eine Komponente, die zu bestehenden Eclipse-Installationen hinzugefügt werden konnte. Papyrus ist vieles: Im Bereich Open Source wird das Framework als UML-Modellierungswerkzeug angewendet, wobei man es herunterladen und sofort mit der Modellierung beginnen kann. Von Eclipse Papyrus 2.0.1-, 2.0.2-, 2.0.3-Neon-release bis 3.0.0- und 3.0.1-Oxygen-release wurden Installation und allgemeine User Experience verbessert, um den Einstieg zu erleichtern. Es ist zu bemerken, dass bei dem Schritt von Eclipse-Kepler zu Luna sich bei Papyrus einiges verbessert hat und es auch mit Eclipse-Mars viel positives Feedback von den Entwicklern und Kunden gibt. Die aktuelle Version Papyrus 6.2.0 auf Basis von UML 5.2.0 und SysML 1.6 wurde am 14.06.2022 veröffentlicht.

Die größte Stärke von Papyrus liegt in seiner Anpassbarkeit. Das Papyrus Framework ermöglicht Anwendern viele Spielräume bei der Modifizierung, um die Voraussetzungen einer bestimmten Domäne, Industrie oder Sprache zu erfüllen. Das macht Papyrus zu einem starken Framework, um UML-basierte, domänenspezifische Modellierungssprachen (DSML) zu entwickeln.

Papyrus stellt ein Eclipse-Projekt für die SysML- und UML-Modellierung dar. Von allen UML-Werkzeugen, die viele Entwickler bisher angewendet haben, hat das

Framework Papyrus die zur Spezifikation konforme Umsetzung von UML 2.5. Sowohl Kunden als auch Entwickler sind von dem Framework begeistert. Im Bereich sicherheitskritischer Systeme ermöglicht Papyrus mittels UML 2.5.0 und SysML 1.6 *(seit 15.12.2021)* den Entwicklern das Herstellen sicherer Produkte.

1.2.1 UML-Aktivitätsdiagramme mit Eclipse-Papyrus

Ziel des Ansatzes der Aktivitätsdiagramme ist es, Verhalten der Systeme bezüglich der Kontroll- und des Datenflussmodells zu modellieren. Hierbei beschreiben sie Prozesse aus der Sicht der Anwender, wobei die Aktivitätsdiagramme eine Kommunikationsgrundlage zwischen den Softwarearchitekt*innen und den -anwender*innen darstellen. Abb. 1.1 zeigt ein Aktivitätsdiagramm von Tasks zum Analysieren von Schwingkreiswechselrichter. Gemäß Abb. 1.1 verfügt das Aktivitätsdiagramm über Notationselemente wie z. B. Knoten, Aktionen oder Kontrollflüsse. Abb. 1.1 zeigt eine parallele Modellierung mithilfe von Fork-Phase, Work-Phase und Join-Phase. Mithilfe der Aktivitätsdiagramme führt die Parallelität die Aktivitäten aus. In Abb. 1.1 ist zu erkennen, dass Tasks sich parallel ausführen lassen, da diese keine Abhängigkeiten bezüglich der Reihenfolge aufweisen. Abb. 1.1 zeigt, dass sich Tasks-Ausführungen sowohl sequenziell verknüpfen als auch mit *Und-* beziehungsweise *Oder-Operationen* zusammenführen lassen.

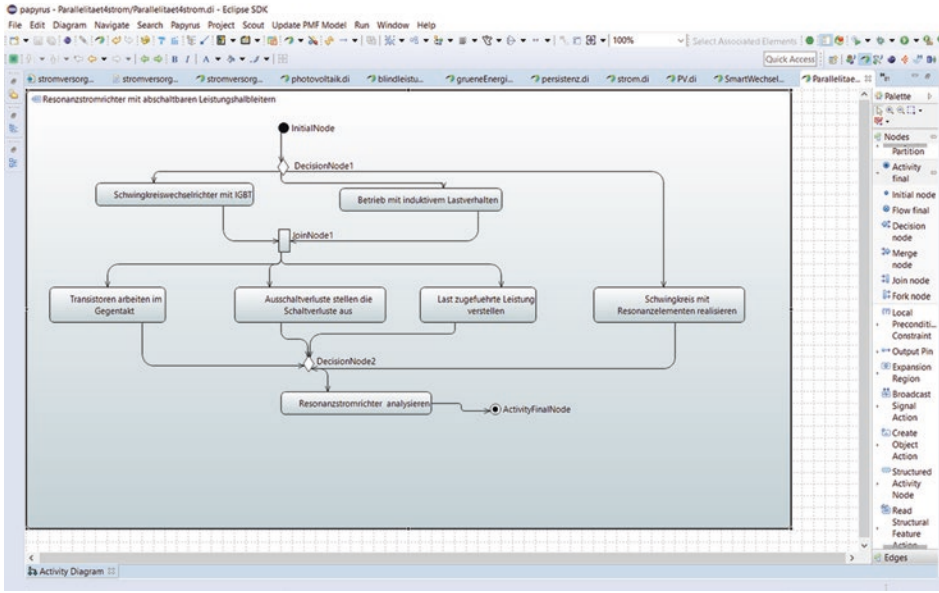

Abb. 1.1 Aktivitätsdiagramm von Tasks zum Analysieren von S*chwingkreiswechselrichtern*

1.2.2 Zustandsdiagramme mit Eclipse-Papyrus

Zustandsdiagramme beschreiben das Verhalten der Komponenten des Lebenszyklus mithilfe der Zustände und Zustandsübergänge. Hierbei werden *Klassifikatoren* wie z. B. Klassen oder Systeme beschrieben. Ziel des Einsatzes der Zustandsdiagramme ist es, Reaktionen von Systemen zu beschreiben [2]. Zum Modellieren des Verhaltens der Schnittstellen mit UML haben die Zustandsdiagramme einen Vorteil gegenüber Aktivitätsdiagrammen. Zustandsdiagramme werden unter anderem zum Beschreiben der Kommunikation zwischen zwei Komponenten in Bezug auf Schnittstellendesign angewendet.

Abb. 1.2 zeigt das Zustandsdiagramm zum Modellieren des Verhaltens von Schnittstellen mit UML in Bezug auf die Beschreibung der Kommunikation zwischen zwei Komponenten wie z. B. Navigation und Human Machine Interface, abgekürzt HMI (Display), in einem automobilen Infotainmentsystem. Gemäß Abb. 1.2 stehen Modellierungselemente, u. a. Region, Zustand, Transition oder Initialzustand, zur Verfügung. Transitionen verfügen über Auslöser, genannt *Trigger,* in Form des erwarteten, logischen Ereignisses und der Angabe über sendende und empfangende Komponenten [3]. Abb. 1.2 stellt eine Beschreibung des dynamischen Schnittstellenverhaltens mithilfe einer domänenspezifischen Sprache (DSL) wie z. B. Obeo Designer im automobilen Infotainment dar.

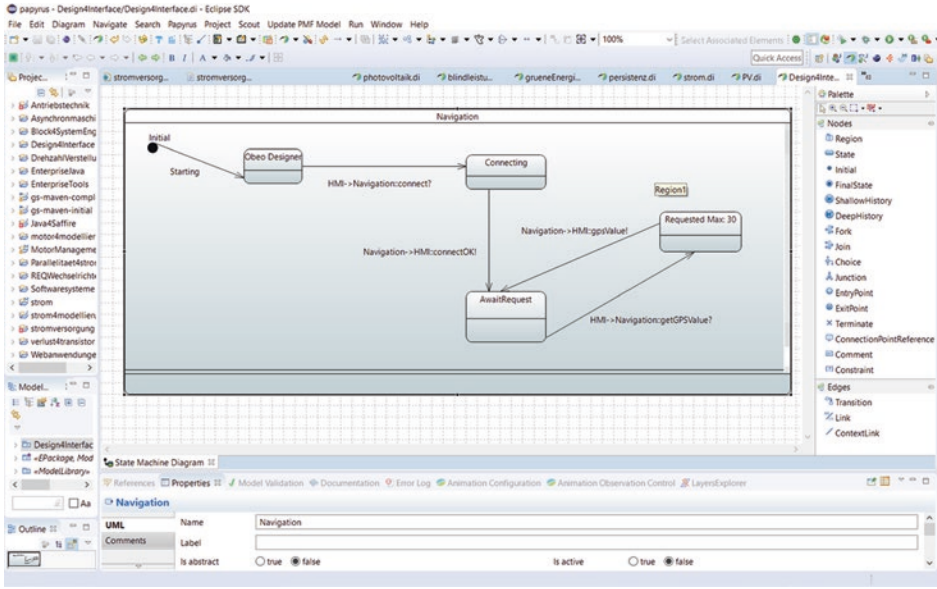

Abb. 1.2 *Zustandsdiagramm* zum Modellieren des Verhaltens von *Schnittstellen* mit *UML*

1.2.3 Das Erstellen von Sequenzdiagrammen mit Eclipse-Papyrus

Sequenzdiagramme stellen die Interaktionen zwischen verschiedenen Objekten in Bezug auf Ereignisse dar. Diese Diagramme beschreiben die dynamische Modellierung des Systems. Sequenzdiagramme werden zum Beschreiben der Funktionalitäten der Lebenslinien der Objekte und deren Interaktion mit anderen Objekten verwendet. Abb. 1.3 zeigt ein Sequenzdiagramm zum Beschreiben des dynamischen Schnittstellenverhaltens in Form einer Nachrichtensequenz. Abb. 1.3 zeigt eine synchrone Nachricht, dargestellt durch eine durchgezogene Linie mit einem ausgefüllten Spitz, als auch eine Antwortnachricht, dargestellt durch eine gestrichelte Linie mit einem gestrichelten Spitz. Die zweite Nachricht stellt eine Antwort auf einen Aufruf dar. Mithilfe des Sequenzdiagramms wird der Ablauf der Kommunikation über die Schnittstelle spezifiziert.

1.2.4 Das Erstellen von Anwendungsfalldiagrammen mit Eclipse-Papyrus

Anwendungsfalldiagramme beschreiben die Aktionen der Akteure zur Steuerung der Anwendungsfälle. Es geht darum, sowohl die Beziehungen als auch die Interaktionen zwischen den Akteuren und den Anwendungsfällen grafisch darzustellen [4]. Akteure

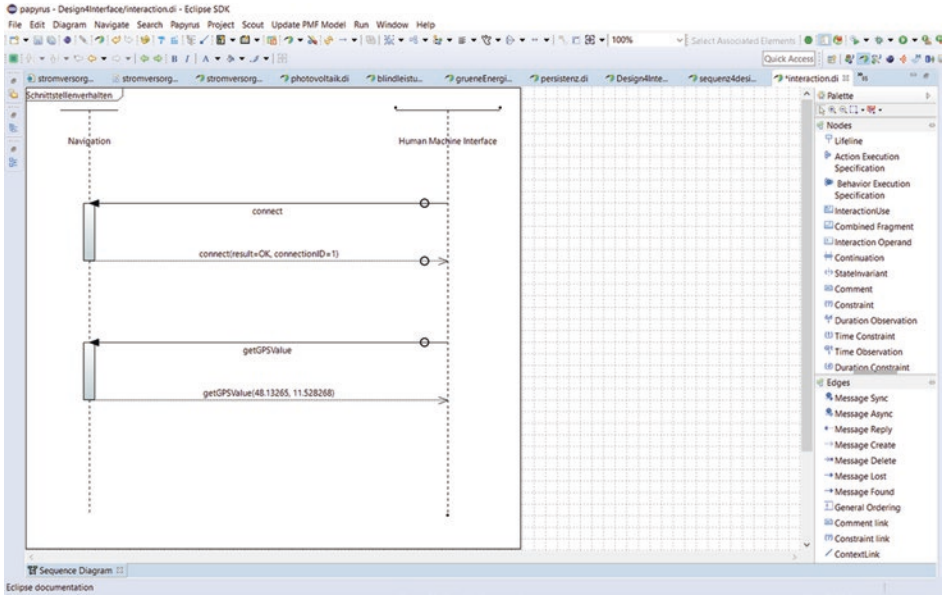

Abb. 1.3 *Sequenzdiagramm* zum Beschreiben der Funktionalitäten der Lebenslinien der Objekte und deren Interaktion mit anderen Objekten

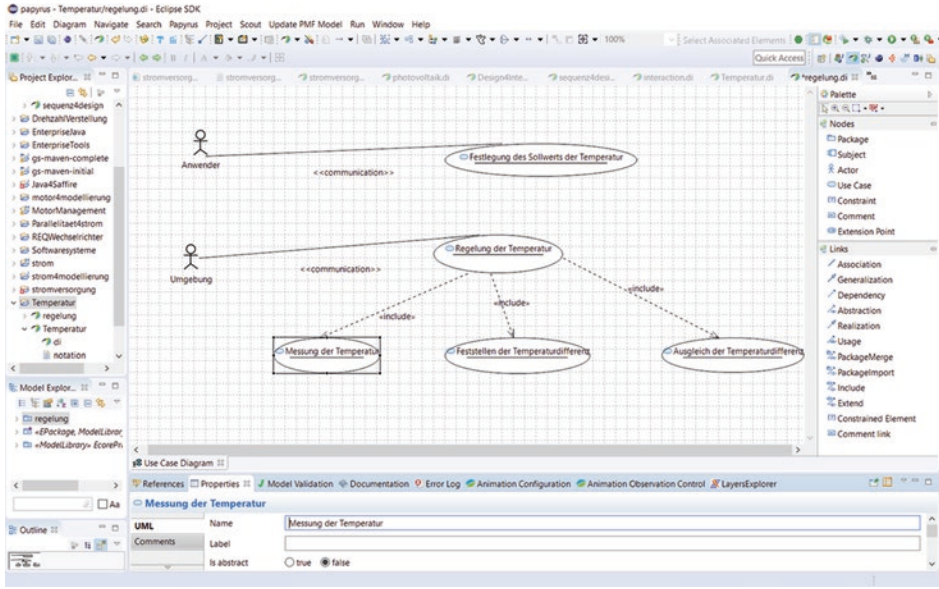

Abb. 1.4 Grafische Darstellung der Beziehungen sowie der Interaktionen zwischen den Akteuren und den Anwendungsfällen

und Anwendungsfälle sind in Verbindung, weil jemand etwas tun würde. Abb. 1.4 zeigt ein Anwendungsfalldiagramm zum Beschreiben der Temperaturregelung mit dem Open Source *Eclipse Papyrus.* Gemäß Abb. 1.4 verfügt das Anwendungsfalldiagramm über zum einen Akteure, wie z. B. Anwender oder Umgebung, und zum anderen Anwendungsfälle, wie z. B. *Regelung der Temperatur* oder *Festlegung des Sollwerts der Temperatur.* Es gibt eine Kommunikation zwischen Anwendern und Anwendungsfällen. Die Analyse der Beschreibung des Anwendungsfalldiagramms von Abb. 1.4 ermöglicht sowohl die Festlegung der Solltemperatur mithilfe des Akteurs *Anwender* als auch die Regelung der Temperatur mithilfe des Akteurs *Umwelt.* Hierbei ändert sich die Außentemperatur und das Programm versucht, die Temperatur durch Regelung auszugleichen. Die Beziehungen *communication* und *include* beschreiben eine Assoziation zwischen Akteuren und Anwendungsfällen bzw. einen Aufruf eines Anwendungsfalles.

1.3 Obeo-UML-Designer

UML Designer ist ein Open-Source-Tool der Firma Obeo zum Erstellen und Visualisieren der Diagramme auf Basis von UML 2.5. Das Framework wurde auf Basis der grafischen domänenspezifischen Sprache (DSL) Obeo Sirius entwickelt. Das Framework UML Designer ermöglicht den Entwicklern die Kombination von UML- und DSL-Modellierung. Der Fokus zum Verbessern der User Experience lag sowohl für Endnutzer auf Sirius-basierenden Tools wie z. B. UML Designer als auch für deren Entwickler.

Für Sirius 5 wurden viele Issues, unter anderem *Decorator*-Management und High-Resolution-Export, gefixt, und die Nutzerfreundlichkeit wurde verbessert. Ziel war es, für die Endnutzer einen besseren Workflow zu erschaffen. Das Resultat dieser Entwicklung ist ein neuer Editor, der es Nutzern erlaubt, alle Konzepte ihrer Modeling-Projekte an einer Stelle zu bearbeiten. Das schließt das Bearbeiten der semantischen Modelle, der nutzbaren *Viewpoints* und aller Darstellungen (Diagramme, Bäume, Tabellen und Matrizen) ein.

UML Designer verfügt über zehn Diagramme: Package Hierarchy, Class Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram, Use Case Diagram, Activity Diagram, State Machine, Sequence Diagram, Profile Diagram.

Mithilfe dieser Diagramme können die Entwickler die Transformation von UML zu DSL ermöglichen. Eclipse Sirius Version 5.0 wurde mithilfe von Eclipse Oxygen veröffentlicht und enthält viele neue Features.

1.3.1 Das Visualisieren der Diagramme auf Basis von UML 2.5

UML ermöglicht das Visualisieren der Struktur eines Diagramms bezüglich der grafischen Erklärung. Hierbei wird die Semantik der Diagramme mithilfe der Modellierungstools erklärt. Die Unified Modeling Language, abgekürzt UML, stellt eine visuelle Modellierungssprache dar, die sich über die Bereiche Architektur, Design und die Abläufe von Herstellungsprozessen erstreckt. Das Konzept von UML fokussiert auf eine grafische Standardnotation bezüglich des Aufzeichnungssystems mithilfe von Zeichen und Symbolen. UML wird als Modellierungssprache zur Beschreibung von Softwaresystemen eingesetzt, wobei die grafische Standardnotation mithilfe von Zeichen und Symbolen dargestellt ist. Die Modellierungstools von UML ermöglicht einen Einsatz in den großen Anwendungsbereichen. Dank dieser Modellierungstools gilt UML momentan als eine der wichtigen Modellierungssprachen für die Softwareentwicklung, wobei die Sprache für die Spezifikation, Konstruktion und Visualisierung von erklärenden Modellen verwendet wird. Sowohl für strukturelle als auch verhaltensbasierte Modellierung wird UML als visuelle Modellierungssprache für Architektur und Design eingesetzt. Tab. 1.1 zeigt die Modellierungstools zum Visualisieren der Diagramme mithilfe von Open Source Eclipse UML Designer.

Gemäß Tab. 1.1 sind zwei wichtige Fragen zum Modellieren der Anwendungen essenziell: Was soll die Anwendung sein? Was soll die Anwendung tun? Mithilfe dieser Fragen werden sowohl die strukturelle als auch die verhaltensbasierte Modellierung gefunden. UML Designer verfügt einerseits über Modellierungstools wie *Activity Tools* und andererseits über Diagrammtypen zum Modellieren der Anwendungen in Bezug auf Objektorientierung. Abb. 1.5 zeigt die Anwendung zum Realisieren der Effizienz des Asynchronmotors. Hierbei ermöglicht die parallele Modellierung mithilfe des Aktivitätsdiagramms vom Open Source Eclipse UML Designer die Erstellung sowohl der Interfaces der Kenngrößen wie z. B. Drehzahl oder Wirkungsgrad als auch der Klassen

Tab. 1.1 Tools zur Visualisierung der UML-Diagramme mit Obeo UML Designer

Anwendungs-modellierung	Was soll die Anwendung sein?	Was soll die Anwendung tun?	Beschreibung des Systems	Erläuterung der internen Struktur des Systems	Definieren der System-umgebung
Diagrammtyp	*Activity Diagram*	*Use Case Diagram*	*Component Diagram*	*Composite Structure Diagram*	*Deployment Diagram*
Model-lierungstools	*Activity Tools*	*Use Case*	Node/Edge	Node/Edge	Types/Relation-ships
Elemente	Node, Opaque Action, Control Flow …	Actor, Use Case, Subject	Component Class …/ Dependency, Connector …	Component, Class, Part/ Connector, Generalization	Node, Device …/Dependency, Deployment …

Modellierungstools von *Open Source Eclipse UML Designer*

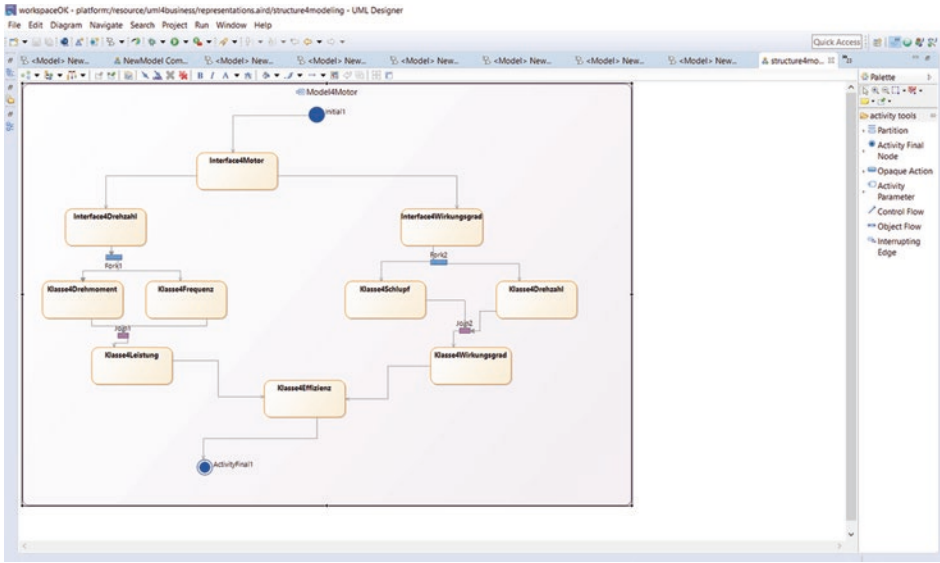

Abb. 1.5 Visualisierung der Analyse der Effizienz des Motors anhand des Aktivitätsdiagrammes

der Kenngrößen Drehmoment oder Schlupf. Gemäß Abb. 1.5 stellt die Anwendung die Berechnungen der Kenngrößen des Asynchronmotors mithilfe der Interfaces und Klassen dar. Diese Anwendung soll die Analyse der Effizienz des Motors mithilfe der Kenngrößen ermöglichen. Das Aktivitätsdiagramm, wie es auf der Abb. 1.5 zu sehen ist, zeigt das Verhalten des Asynchronmotors. Hierbei ist das Verhalten des Asynchron-motors bezüglich der Berechnungen der Kenngrößen dynamisch. Abb. 1.5 zeigt die Visualisierung der Analyse der Effizienz des Motors anhand des Aktivitätsdiagramms. Das Aktivitätsdiagramm zeigt den ablaufbasierten Kontrollfluss, genannt Prozesse zwischen Klassenobjekten zusammen mit geplanten Prozessen wie Tools-Workflows.

Mithilfe der Modellierungstools wie z. B. *Opaque Action* oder *Control Flows* werden Diagramme erstellt. Die Anwendung gemäß Abb. 1.5 ermöglicht die Darstellungen der Modellierung sowohl auf der vertikalen als auch auf der horizontalen Schicht. Die vertikale Schicht, wie sie in beiden Beispielen *Example 1–2* zu sehen ist, zeigt die Orientierung der Aktionen von oben nach unten. Es gibt zwei Säulen oder vertikale Schichten 1 und 2. In jeder Säule gibt es sowohl Interface als auch Klassen.

1. Vertikale Schicht:
   Interface4Drehzahl
   Klasse4Drehmoment
   Klasse4Frequenz + Klasse4Leistung
2. Vertikale Schicht
   Interface4Wirkungsgrad
   Klasse4Schlupf + Klasse4Drehzahl
   Klasse4Wirkungsgrad

Gemäß Example 1–2 zeigt die Anwendung eine parallele Struktur bezüglich der Position der vertikalen Schichten 1 und 2. Hierbei sind die vertikalen Schichten 1 und 2 zueinander parallel. Die Anwendung zeigt mithilfe der parallelen Modellierung zwei parallel vertikale Schichten.

Das Modellieren der Anwendung zur Effizienzanalyse des Asynchronmotors beginnt mit einem ausgefüllten Kreis, genannt *Initial1* zum Zeichen des Aktivitätsdiagramms.

Hierbei wird *Initial1* mit der ersten Aktion, genannt *Interface4Motor* verbunden, welche durch ein Rechteck mit abgerundeten Ecken dargestellt wird, wie es in Abb. 1.5 zu sehen ist.

Das Verbinden jeder Aktion mit anderen Aktionen ermöglicht das Modellieren des Verhaltens des Asynchronmotors mithilfe der Modellierungstools wie z. B. Linien, die schrittweise den Fluss des gesamten Prozesses darstellen.

Anwendungsfalldiagramme ermöglichen den Entwicklern zu ermitteln, was eine Anwendung tun soll. Hierbei werden Entwickler mithilfe des Anwendungsfalldiagramms eine Antwort auf die Frage *„Was soll die Anwendung zur Analyse der Motoreffizienz tun?“* finden. Zur Analyse des Verhaltens des Systems wird das Anwendungsfalldiagramm, genannt *Use Case Diagram,* verwendet, welches eine Beziehung zwischen Benutzern, genannt Akteur, und Anwendungen des Systems darstellt. Abb. 1.6 zeigt die Funktionalität eines Anwendungsfalldiagramms mithilfe von Eclipse UML Designer bezüglich der Analyse der Effizienz eines Asynchronmotors. Gemäß Abb. 1.6 verfügt das Diagramm über sechs Anwendungsfälle, an denen sowohl Computer als auch Entwickler interessiert sind. Abb. 1.6 zeigt, dass die Beziehungen zwischen den Anwendungsfällen untereinanderstehen, wie z. B. zwischen dem Anwendungsfall *Klasse4Hauptprogramm* und den Anwendungsfällen *Klasse4Effizienz implementieren* und *Klasse4Verluste implementieren.* Gemäß Abb. 1.6 steht der Akteur *Computer* in Beziehung zu dem Akteur *Entwickler* und zur Modellierung der Motoreffizienz, wobei der Computer von Software wie Eclipse UML Designer oder Eclipse Papyrus dargestellt ist. Der

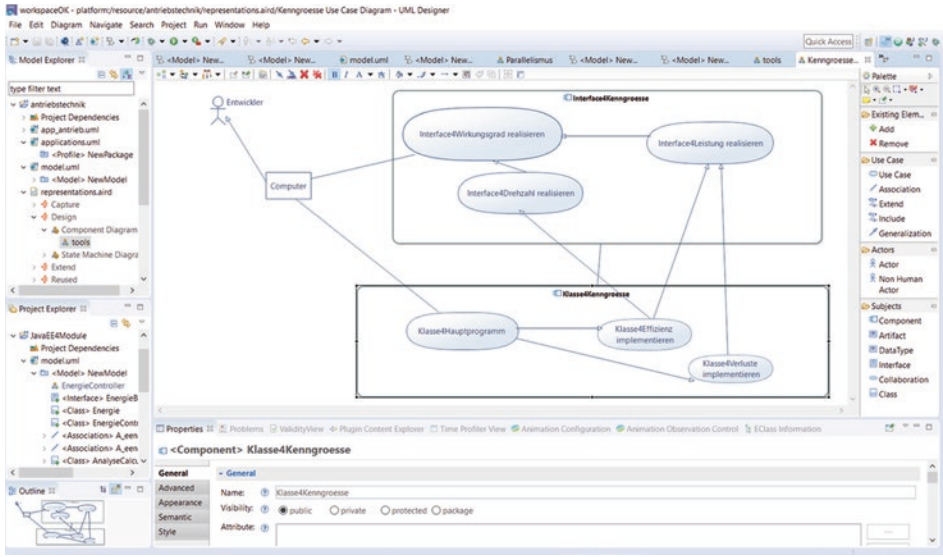

Abb. 1.6 Analyse der Effizienz eines Asynchronmotors mit dem Anwendungsfalldiagramm

Entwickler benutzt den Computer zum Erstellen des UML-Anwendungsfalldiagramms mit dem Ziel, die Motoreffizienz zu analysieren. Gemäß Abb. 1.6 zeigt das Diagramm zwei Arte der Beziehungen: Generalisierung und Assoziationen. Beispielsweise zeigt Abb. 1.6 Generalisierung zwischen den Anwendungsfällen *Interface4Leistung realisieren* und *Interface4Wirkungsgrad realisieren.* Außerdem gibt es eine Assoziation zwischen den Komponenten *Interface4Kenngroesse* und *Klasse4Kenngroesse.* Der Akteur *Computer* ist mit den vorherigen Komponenten mithilfe der Anwendungsfälle *Interface4Wirkungsgrad realisieren* und *Klasse4Hauptprogramm* assoziiert. Die Anwendungsfallspezifikationen stellen Informationen zu den Anwendungsfällen in Textform dar, wie es im Container *Important* zu sehen ist. Eine Beispieldarstellung von Anwendungsfallspezifikationen zeigt der Container *Important* [5, 6]. Ein Anwendungsfall eines Systems stellt eine Folge von Aktionen dar, welche ein System ausführt und die ein erkennbares Ergebnis von Wert für einen bestimmten Akteur liefert, während ein Anwendungsfalldiagramm mehrere Anwendungsfälle sowie die Beziehungen zwischen den Anwendungsfällen und den Personen, Gruppen oder Systemen enthalten, die miteinander interagieren, um den Anwendungsfall ausführen zu können [7].

Wichtig

Ziel ist es, möglichst einfach zu zeigen, was man mit dem zu bauenden Softwaresystem machen will, welche Fälle der Anwendung es also gibt.

Akteure werden als Strichmännchen dargestellt, welche sowohl Personen wie Kunden oder Administratoren als auch ein System darstellen können (bei Systemen wird manchmal auch ein Bandsymbol verwendet).

Anwendungsfälle werden in Ellipsen dargestellt. Sie müssen beschrieben werden (z. B. in einem Kommentar oder einer eigenen Datei).

Assoziationen zwischen Akteuren und Anwendungsfällen müssen durch Linien gekennzeichnet werden.

Systemgrenzen werden durch Rechtecke gekennzeichnet.

Include-Beziehungen werden mittels <<*include*>>, gestrichelter Linie und einem Pfeil zum inkludierten Anwendungsfall gekennzeichnet, wobei dieser für den aufrufenden Anwendungsfall notwendig ist.

Extend-Beziehungen werden mittels <<*extend*>>, gestrichelter Linie und einem Pfeil vom erweiternden Anwendungsfall gekennzeichnet, wobei dieser von dem aufrufenden Anwendungsfall aktiviert werden *kann,* aber nicht muss.

Tab. 1.2 zeigt Informationen über Darstellungsspezifikationen von Anwendungsfalldiagrammen bezüglich des Verhaltens des Systems. Hierbei können diese Anwendungsfallspezifikationen nach Bedarf mithilfe eines Texteditors dokumentiert werden. Tab. 1.2 beschreibt eine Darstellung einer Anwendungsspezifikation [7].

1.3.2 Überblicke über UML-Diagramme mit Eclipse-UML-Designer

Die OMG definiert den Zweck der UML wie folgt [8]:

Definition

I. Systemarchitekten und Softwareentwickler erhalten ein Tool für die Analyse, das Design und die Implementierung von softwarebasierten Systemen sowie für die Modellierung von Geschäfts- und ähnlichen Prozessen.
II. Die Branchensituation soll optimiert werden, indem dafür gesorgt wird, dass Tools zur visuellen Modellierung von Objekten miteinander kompatibel sind. Allerdings ist es für einen sinnvollen Austausch von Modelldaten zwischen Tools notwendig, dass eine einheitliche Notation und Semantik verwendet wird.

UML erfüllt die folgenden Anforderungen [8]:

Wichtig

- Festlegung einer formalen Definition für ein gemeinsames Metamodell, das auf *Meta Object Facility* (MOF) basiert und die abstrakte Syntax der UML spezifiziert. Die abstrakte Syntax definiert die UML-Modellierungskonzepte, ihre Attribute und ihre Beziehungen sowie die Regeln für die Kombination dieser Konzepte zur Entwicklung partieller oder kompletter UML-Modelle.
- Eine detaillierte Erklärung der Semantik für jedes einzelne UML-Modellierungskonzept. Die Semantiken definieren – in einer von der Technologie unabhängigen Art und Weise –, wie die UML-Konzepte von Computern realisiert werden.

Tab. 1.2 Beispieldarstellung einer Anwendungsfallspezifikation

Spezifikationstypen	Beschreibung
Anwendungsfallname	Gibt den Namen des Anwendungsfalls an. Normalerweise drückt der Name die Zielsetzung oder das erkennbare Ergebnis des Anwendungsfalls aus, beispielsweise für einen Bankautomaten *„Bargeld abheben“*
Kurzbeschreibung	Beschreibt die Rolle und den Zweck des Anwendungsfalls
Ereignisablauf	Stellt den Basisablauf (Fluss) und die alternativen Abläufe dar. Der Ereignisablauf (Ereignisfluss) beschreibt das Verhalten des Systems. Wie das System funktioniert, die Details der Darstellung oder die Details der Benutzerschnittstelle werden jedoch vom Ereignisablauf nicht beschrieben. Wenn Informationen ausgetauscht werden, muss der Anwendungsfall die ausgetauschten Informationen explizit angeben. Beispielsweise muss anstelle der folgenden Beschreibung *„Der Akteur gibt Kundendaten ein“* für eine Aktion die explizite Angabe *„Der Akteur gibt den Kundennamen und die Kundenadresse ein“* erfolgen
Basisablauf	Beschreibt das ideale, primäre Verhalten des Systems
Alternative Ereignisabläufe	Beschreibt Ausnahmen oder Abweichungen vom Basisablauf, z. B. die Art und Weise, wie sich das System verhält, wenn der Akteur eine falsche Benutzer-ID eingibt und die Benutzerauthentifizierung scheitert
Spezielle Anforderungen	Nicht funktionale Anforderungen, die spezifisch für einen Anwendungsfall sind, aber im Text des Ereignisablaufs für den Anwendungsfall nicht angegeben sind. Beispiele für spezielle Anforderungen sind gesetzliche Bestimmungen, Anwendungsstandards, Qualitätsattribute des Systems, einschließlich Benutzerfreundlichkeit, Zuverlässigkeit, Leistung und Servicefreundlichkeit, Betriebssysteme und Umgebungen, Anforderungen bezüglich der Kompatibilität und Designeinschränkungen
Vorbedingungen	Ein Status des Systems, der gegeben sein muss, bevor ein Anwendungsfall gestartet wird
Nachträgliche Bedingungen	Eine Liste möglicher Zustände des Systems nach Beendigung des Anwendungsfalls
Erweiterungspunkte	Ein Punkt im Ereignisablauf des Anwendungsfalls, an dem ein anderer Anwendungsfall referenziert wird

Darstellung einer Anwendungsspezifikation

- Bestimmung der von Menschen lesbaren Notationselemente für die Darstellung individueller UML-Modellierungskonzepte sowie die Festlegung von Regeln für die Kombination solcher Konzepte, um eine Vielzahl von Diagrammarten für verschiedene Aspekte des modellierten Systems erstellen zu können.
- Festlegung von Methoden, mit denen gewährleistet werden kann, dass UML-Tools mit dieser Spezifizierung übereinstimmen. Dies wird (in einer separaten Spezifizierung) durch eine XML-basierte Spezifizierung von zugehörigen Modell-Austauschformaten (XMI) unterstützt – konforme Tools müssen, diesen Prozess dann durchlaufen.

Eclipse UML Designer ist ein Open-Source-Projekt der Firma Obeo, welches auf Sirius 5 basiert. Eclipse Sirius stellt ein Werkzeug zum Erstellen der domänenspezifischen Modellierungswerkzeuge dar. UML Designer stellt ein Tool sowohl zum Erstellen als auch zum Visualisieren von UML-2.5-Modellen. Eclipse UML Designer verfügt über zehn generische UML-Diagramme: Package Hierarchy, Class Diagram, Component Diagram, Composite Structure Diagram, Deployment Diagram, Use Case Diagram, Activity Diagram, State Machine, Sequence Diagram und Profile Diagram. UML nutzt Modellierungstools zum Erstellen verschiedener Arten von Diagrammen: einerseits statische Diagramme zum Darstellen struktureller Aspekte eines Systems und andererseits verhaltensbasierte Diagramme zum Erfassen dynamischer Aspekte eines Systems. Der Container zur *Background Information* stellt zusätzliche Information über die Definition von UML.

Hintergrundinformation
Ziel der Entwicklung der Unified Modeling Language (UML) ist es, sowohl Semantik als auch Syntax visueller Modellierungssprachen in den Bereichen Architektur, Design und Implementierung komplexer Softwaresysteme bezüglich struktureller als auch verhaltensbasierter Modelle zu realisieren.

Die UML besteht aus verschiedenen Diagrammarten. UML-Diagramme stellen die Grenzen, die Struktur und das Verhalten von Systemen und deren Objekten dar.

UML verfügt über Tools, die UML-Diagramme anwenden, zum Generieren der Code in verschiedenen Programmiersprachen u. a. Java, C++ oder C#. UML bezieht sich sowohl auf objektorientierter Analyse als auch auf Design.

1.3.2.1 Strukturelle Diagramme

Strukturelle Diagramme beschreiben die Modellierung zeitinvarianter oder unveränderlicher Komponenten von Systemen, wobei diese Komponenten statisch sind [8].

1. **Klassendiagramm:** Das ist das am häufigsten verwendete UML-Diagramm und die wichtigste Grundlage für jede objektorientierte Lösung. Klassen in einem System, Attribute und Vorgänge sowie die Beziehung zwischen den einzelnen Klassen. Klassen werden gruppiert, um Klassendiagramme zu erstellen, wenn große Systeme als Diagramm dargestellt werden sollen.

2. **Komponentendiagramm:** Stellt die strukturelle Beziehung von Softwaresystemelementen dar. Wird am häufigsten für komplexe Systeme mit mehreren Komponenten eingesetzt. Komponenten kommunizieren über Schnittstellen.
3. **Kompositionsstrukturdiagramme:** Kompositionsstrukturdiagramme werden verwendet, um die interne Struktur einer Klasse darzustellen.
4. **Implementierungsdiagramme:** Illustriert die Systemhardware und die zugehörige Software. Nützlich, wenn eine Softwarelösung auf mehreren Maschinen mit individuellen Konfigurationen implementiert wird.

Paketdiagramme: Es gibt zwei spezielle Arten von Abhängigkeiten, die zwischen Paketen definiert werden: Paketimporte und Paketverschmelzungen. Pakete können die unterschiedlichen Ebenen eines Systems darstellen, um die Architektur zu visualisieren. Paketabhängigkeiten können so dargestellt werden, dass die Kommunikationsmechanismen zwischen verschiedenen Schichten erkennbar sind.

1.3.2.2 Verhaltensbasierte Diagramme

Verhaltensbasierte Diagramme beschreiben die Modellierung dynamischer Komponenten von Systemen [8].

1. **Aktivitätsdiagramm:** Grafisch dargestellte Geschäfts- oder Betriebsabläufe, um die Aktivität eines Teils oder einer Komponente in einem System zu visualisieren. Aktivitätsdiagramme werden alternativ zu den Zustandsdiagrammen verwendet.
2. **Sequenzdiagramm** zeigt, wie und in welcher Reihenfolge Objekte miteinander interagieren. Solche Diagramme repräsentieren Interaktionen für ein bestimmtes Szenario.
3. **Zustandsdiagramm:** Ähnlich wie Aktivitätsdiagramme beschreibt diese Art von Diagramm das Verhalten von Objekten, die in ihrem aktuellen Zustand unterschiedliche Verhaltensweisen an den Tag legen.
4. **Anwendungsfalldiagramm** stellt eine bestimmte Funktionalität eines Systems dar und wurde entwickelt, um zu illustrieren, wie Funktionen zueinander in Beziehung stehen und welche internen/externen Akteure es gibt.

Profildiagramm erweitert den UML-Standard zum Erstellen neuer Konzepte. Hierbei wirkt das Profildiagramm in dem Metamodellbereich zum Erstellen neuer Stereotype und Profile.

1.3.3 Beispiele von UML-Diagrammen

UML-Diagramme werden für bestimmte Zwecke zur Visualisierung sowohl des Unsichtbaren als auch des Sichtbaren von Software bzw. Systemen verwendet. Hierbei werden sie zum Beschreiben der Codes aus einem Programm eingesetzt. Klassendiagramme werden im Bereich Energietechnikinformatik zur Realisierung eines Prüfprotokolls

mithilfe einer Klasse und deren Eigenschaft genutzt. Ein Komponentendiagramm zeigt die Sicht oder die Struktur einer Software. Was man nicht sehen kann, wird mithilfe dieses Diagramms beschrieben. Hierbei wird die physikalische Struktur und die Qualität des Websystems dargestellt. Ein Zustandsdiagramm beschreibt die Zustände und Übergänge zum Analysieren des Verhaltens eines Systems. Ein Profildiagramm stellt die Modellierungen auf der Metaebene dar. Das Verteilungsdiagramm beschreibt die physikalischen Ressourcen von Applikation-Servern wie GlassFish, TomEE, WildFly oder Open Liberty.

1.3.3.1 Klassendiagramme für Stromversorgungstester

Klassendiagramme sind der wichtige Teil von UML. Sie erfüllen die wichtigsten Ziele der UML, weil sie Designelemente von der Codierung des Systems trennen. Ziel der Anwendung der UML in der Softwareentwicklung ist es, ein normiertes Modell zur Beschreibung eines objektorientierten Codierungsansatzes zu entwickeln. Weil Klassen die Struktur von Objekten darstellen, können Klassendiagramme als das Herzstück der UML verstanden werden. Die Diagrammkomponenten eines Klassendiagramms entsprechen den zu programmierenden Klassen, Hauptobjekten oder Interaktion zwischen Klasse und Objekt.

Die Klassenstruktur verfügt über ein Rechteck mit drei Schichten. Eine obere Schicht enthält den Namen der Klasse, die mittlere Schicht gibt ihre Attribute an und die unterste Schicht gibt Information über die Methoden oder Abläufe innerhalb der Klasse. In einem Klassendiagramm werden Klassen und Unterklassen gruppiert, um den strukturellen Zusammenhang zwischen den einzelnen Objekten zu verdeutlichen. Abb. 1.7 zeigt die Modellierung eines Prüfprotokolls für die Stromversorgung mithilfe des Klassendiagramms. Hierbei verfügen die Klassen Test A–E über Merkmale, wie z. B. Attribute und Operationen zum Durchführen des Tests für die Stromversorgung. Mithilfe der Klasse *ErgebnisA* werden die Ergebnisse anderer Klassen geprüft. Die Klasse *ErgebnisA* enthält zwei Attribute *fail* und *pass* sowie die Operation *ergebnisPruefen* zum Bestätigen des Ergebnisses jedes Tests, wobei die Ergebnisse der Unterklassen von der Oberklasse *ErgebnisA* abhängen.

1.3.3.2 Komponentendiagramme für die Websystemqualität

Komponentendiagramme zeigen sowohl die interne als auch die externe Sicht einer Software oder eines Systems. Hierbei werden verschiedene Komponenten dargestellt. Sie sind als Rechteck dargestellt und tragen in der linken, oberen Ecke das Komponentensymbol. Beziehungen können im Komponentendiagramm als gestrichelte Linien angegeben werden. Komponentendiagramme ermöglichen die Visualisierung der physikalischen Struktur des Systems. Außerdem sind die Komponentendiagramme sowohl bei der Analyse der Komponenten des Systems als auch bei deren Beziehungen hilfreich. Abb. 1.8 stellt die generische Architektur eines Websystems mithilfe des Komponentendiagramms dar. Gemäß Abb. 1.8 verfügt ein Websystem über vier Schichten: Frontend, Backend, Integration und Service. Die Komponente Benutzerschnittstelle, genannt *Frontend,*

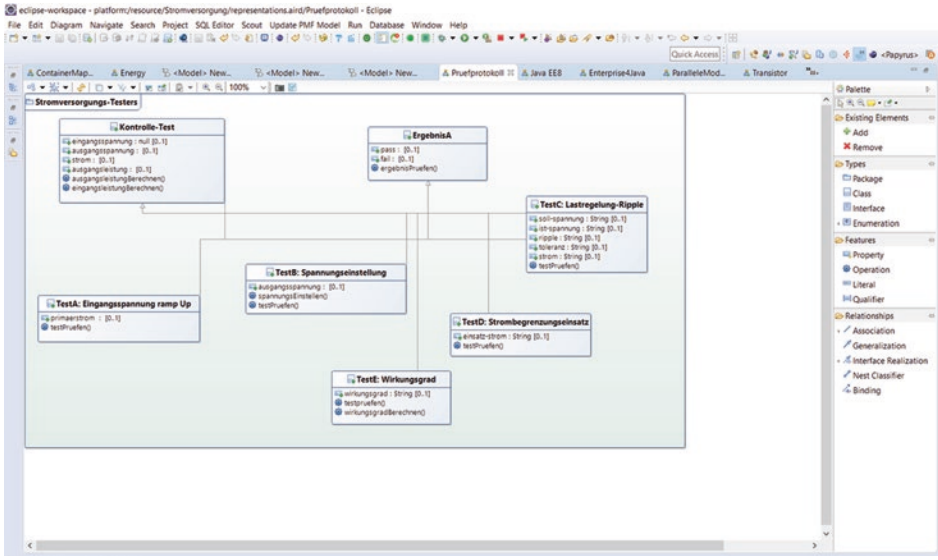

Abb. 1.7 Modellierung eines Prüfprotokolls für die *Stromversorgung* mithilfe des Klassendiagramms

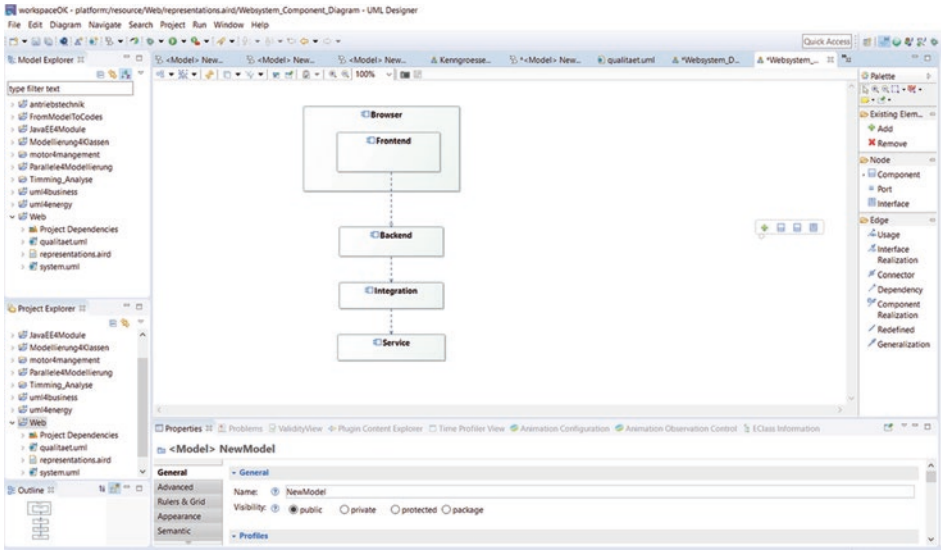

Abb. 1.8 Darstellung der generischen *Architektur* eines Websystems mithilfe des *Komponentendiagramms*

befindet sich im Browser. Die Komponente *Frontend* ist von der Komponente *Backend* geladen und bezieht ihre Daten von dieser. Eine Komponente *Integration* stellt die Vermittlungsschicht zur Anbindung der Daten- und Geschäftsdienste dar. Die Komponente *Service,* entkoppelt von der Komponente *Integration,* stellt REST- oder SOAP-Anwendungen für die Websysteme dar. Diese Entkopplung zwischen Komponenten, *Integration* und *Service* ermöglicht eine bessere Websystemqualität bezüglich der Wartbarkeit, Verfügbarkeit, Sicherheit oder Performance [9].

1.3.3.3 Zustandsdiagramme für die Websystemqualität

Zustandsdiagramme, genannt Maschinenzustandsdiagramme, stellen Abbildungen der Übergänge zwischen verschiedenen Objekten dar. Essenzielle Elemente der Zustandsdiagramme sind u. a. Zustände und Übergänge. Zustände werden mit Rechtecken mit abgerundeten Ecken dargestellt, die mit dem Namen des Zustands bezeichnet werden. Übergänge werden mit Pfeilen, die von einem Zustand zu einem anderen verlaufen, gekennzeichnet und geben die Zustandsveränderung an. Zustandsdiagramme verfügen über verschiedene Elemente wie z. B. Anfangszustand, Endzustand, Ereignis, Zustand, Übergangsverhalten, Austrittspunkt, Übergang und Auslöser.

Zustandsdiagramme finden viele Anwendungen wie z. B. Abbildung ereignisgesteuerter Objekte in einem reaktiven System oder Aufzeigen des Gesamtverhaltens einer Zustandsmaschine oder des Verhaltens mehrerer, miteinander in Beziehung stehender Zustandsmaschinen. Abb. 1.9 zeigt die Veranschaulichung eines Qualitätsszenarios in der Websystemqualität. Hierbei ist zu bemerken, dass die Messbarkeit eines essenziellen Faktors zum Erfüllen der Anforderungen der Websystemqualität angewendet wird. Abb. 1.9 zeigt das Zustandsdiagramm eines Qualitätsszenarios bezüglich des Verhaltens des Websystems. Zwischen Anfangs- und Endzustand des Zustandsdiagramms gibt es Übergänge wie z. B. *Stimulus* oder *Antwort* und Zustände wie z. B. *Quelle* oder *Messkriterium.* Gemäß Abb. 1.9 stellt die Quelle des Stimulus, den Ursprung des Reizes, u. a. User oder Administrator, dar. Der Übergang *Stimulus* beschreibt eine spezifische Kooperation des Zustands *Quelle* mit dem Websystem. Der Zustand *Artefakt* liegt in der Umgebung, wobei das Artefakt einen Funktionsblock darstellt. Dieser Übergang *Antwort* wird mithilfe des Zustands *Messkriterium* geprüft.

1.3.3.4 Profildiagramme für parallele Prozesse mit dem Asynchronmotor

Profildiagramme ermöglichen die Modellierungen der Metaebene. Außerdem spielen die Profildiagramme eine Nebenrolle zur Modellierung der Softwaresysteme.

Profildiagramme werden in der Metamodellebene verwendet, um **Klassen** mit zusätzlichen **Stereotypen** auszuzeichnen, welche die Bezeichnung <<stereotype>> oder bei Profilen erhalten. Profile stellen die Erweiterung des UML-Modells zum Erstellen von benutzerdefinierter Stereotypen, Eigenschaftswerte und Randbedingungen für Plattformen oder Domänen. Mithilfe der Profildiagramme lassen sich die UML-Diagramme an besondere Einsatzgebiete anpassen [10]. Abb. 1.10 zeigt das Profildiagramm