Gottfried Wilhelm
Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Praktische Informatik
Fachgebiet Software Engineering

Konvertierung von SPEM nach Prolog

Bachelorarbeit

im Studiengang Informatik

von

Orhan Sarioglu

Prüfer: Prof. Dr. Kurt Schneider
Zweitprüfer: Prof. Dr. Heribert Vollmer
Betreuer: Dipl.-Math. Thomas Flohr

Hannover, 26. Mai 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbstständig und ohne fremde Hilfe verfasst habe. Es wurden keine anderen als die in der Arbeit angegebenen Quellen und Hilfsmittel verwendet.

_____

Hannover, 26. Mai 2007 Orhan Sarioglu

Danksagung

Ich bedanke mich bei Herrn Dipl.-Math. Thomas Flohr für die hervorragende Betreuung meiner Bachelorarbeit.

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Das Software Engineering Metamodel (kurz SPEM) [1] ist eine relativ neue Modellierungssprache zur Modellierung von Softwareprozessen. SPEM eignet sich vor allem zur grafischen Darstellung von Prozessen. Im Rahmen einer Abschlussarbeit im Fachgebiet Software Engineering ist bereits ein Editor entstanden, mit dem Prozesse in SPEM abgebildet werden können [5]. Ziel dieser Arbeit ist es die drei Diagrammtypen (Klassen-, Use-Case und Aktivitätsdiagramme), die in dem SPEM- Editor modelliert werden soweit in Prolog zu überführen, dass ein Tailoring-Experte damit einen gegebenen Prozess in Prolog verändern und an die Bedürfnisse und Einschränkungen eines konkreten Projektes anpassen kann. Damit soll erreicht werden, dass Umfang und Tiefe des Prozesses dem Projekt angemessen sind. Um dies zu ermöglichen muss ein Programm entwickelt werden der in XML abgespeicherte SPEM-Prozessmodelle in Prolog transformiert. Prolog eignet sich gut für die Lesbarkeit der Prozesse von Menschen und Maschinen. In diesem Zusammenhang sollen Prozesse mit Hilfe von Prolog optimiert werden. Dabei stellt Prolog ein mögliches Werkzeug für Prozessverbesserung dar.

1.2. Aufgabenstellung

Die ursprüngliche Aufgabenstellung hat sich während der Arbeit zu der Folgenden verändert:

Ziel der Arbeit ist es ein Programm zu erstellen, das SPEM in Prolog überführt und damit das Modellieren und Tailorn von Softwareprozessen in Prolog erlaubt. Für die Konvertierung ist die Verwendung von XSLT Stylesheets sinnvoll.

Neben dem zu erstellenden Programm, ist eine Ausarbeitung im Umfang von 40 bis 50 Seiten zu erstellen, die auf SPEM und Das Prolog-Prozessmodell und die notwendigen Konvertierungen eingeht.

1.3. Anforderungen an die Arbeit

Die ursprüngliche Aufgabenstellung hat sich nach Absprache mit meinem Betreuer im Verlaufe der Arbeit verändert, so dass meinem Betreuer in erster Linie eine vollständige Übersetzung in eine Richtung wichtig war.

Dabei sollten Folgende Anforderungen erfüllt sein:

- Die Übersetzung soll an einem echten Projekt erklärt werden.
- Die Prädikate und Regeln sollen übersichtlich und verständlich sein.
- SPEM soll soweit es für die Übersetzung wichtig ist erklärt werden.
- Nicht die Werkzeuge sondern das Konzept der Arbeit soll beschrieben werden.

- Die Transformation soll mittels des ANT-Skripts in Eclipse realisiert werden
- Am Ende soll auf die Nützlichkeit der Arbeit eingegangen werden

Die Rücktransformation sollte nicht mehr durchgeführt werden.

1.4. Gliederung

- In Kapitel 1 wird ein Einblick auf die Arbeit gegeben und die Grundanforderungen an die Arbeit gestellt.

- Kapitel 2 beschreit SPEM wie es für diese Arbeit von Bedeutung ist. In diesem Kapitel soll die Absicht und die Struktur von SPEM beschrieben werden. Dabei wird gezielt auf die zu übersetzenden Stereotypen und deren Beziehungen untereinander eingegangen. Die Regeln zur Wohlgeformtheit definieren hier die Grenzen und geben einen Überblick über mögliche Abhängigkeiten. Anschließend werden Beispiele der drei SPEM-Diagrammtypen an Hand des Softwareprojektbeispieles veranschaulicht.

- In Kapitel 3 wird die Umwandlung von SPEM in Prolog an grafischen Beispielen erklärt und anschließend für die Übersichtlichkeit in Tabellen dargestellt. Das Softwareprojekt soll dabei helfen sich die Umwandlung an einem echten und für uns bekannten Projekt leichter vorzustellen. Außerdem hat der Leser hier noch mal die Gelegenheit SPEM durch die verwendeten Beispiele besser zu verstehen. Des Weiteren wird die Übersetzung der Annotation erläutert und an einem kleinen Modell die technische Umwandlung beschrieben.

- Das vierte Kapitel beschreibt eine mögliche Verwendung der Arbeit. Dabei wird die Rolle der Arbeit in einem Prozessverbesserungskreislauf erläutert und mit einem Beispiel demonstriert.

- In Kapitel 5 werden die wichtigsten Punkte der Arbeit zusammengefasst und Ideen vorgeschlagen wie die Arbeit weiter ausgebaut und verbessert werden kann. Die Probleme die bei der Umwandlung aufgetreten sind werden hier erläutert.

- Im sechsten Kapitel wird erklärt wie das Programm zu bedienen ist und welche weiteren Mittel dafür benötigt werden.

2. SPEM

2.1. SPEM Einleitung

Ein Standard zur Beschreibung von Software Prozessen ist das von der Object-Management Group (OMG) entwickelte SPEM (Software Prozess Engineering Metamodel) [1]. Hierbei handelt es sich in erster Linie um ein Metamodell, mit dessen Hilfe konkrete Software Entwicklungsprozesse dargestellt werden können. Dabei wird das von SPEM beinhaltete Metamodell zur Definition von Prozessen und deren Komponenten eingesetzt. Der SPEM Ansatz ist ein objekt-orientierter Ansatz zur Modellierung von Software Prozessen und verwendet die UML als Basisnotation. Auch hier wird ähnlich zur UML ein 4-Schichten-Konzept zur Modellierung eingesetzt, wie es die Abbildung 1 veranschaulicht.

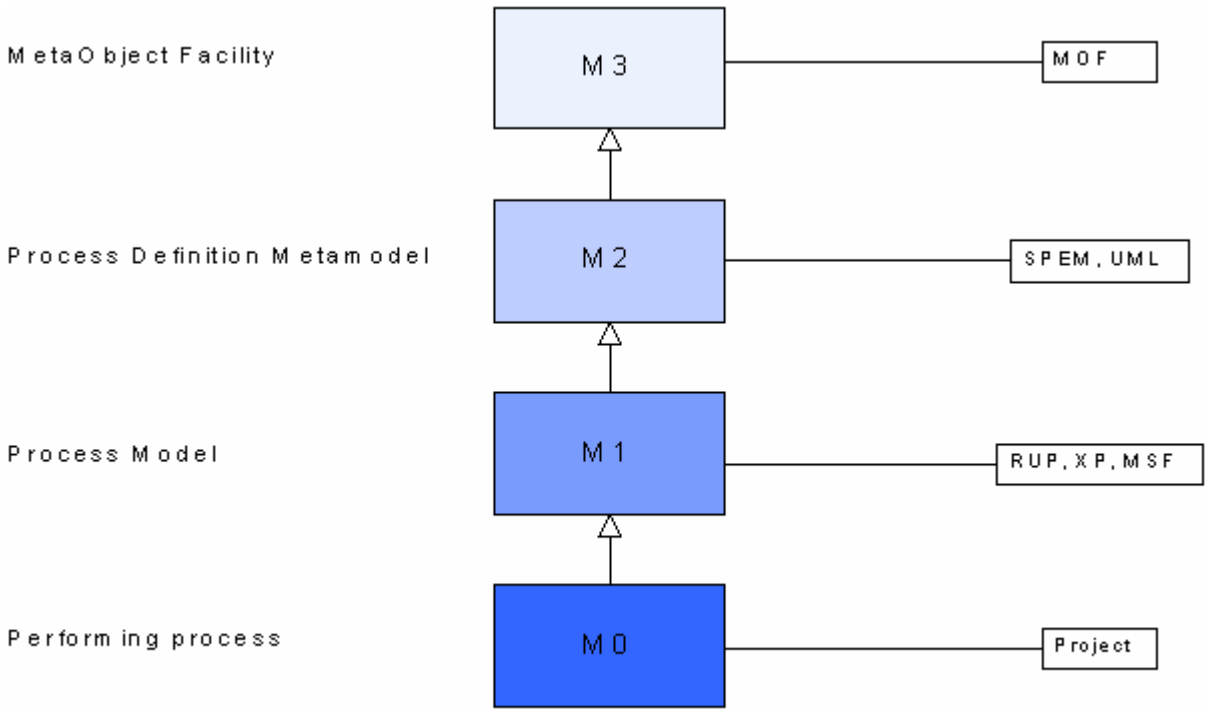


Abbildung 1: MOF Metamodell aus dem SPEM-Tutorial [4]

In diesem Modell wird ein tatsächlicher Prozess auf der Ebene M0 ausgeführt. Während seine Definition auf Ebene M1 stattfindet. Entsprechend der Metaebenenarchitektur bei MOF wird dann das Prozessmodell wiederum auf Ebene M2 modelliert usw. Ich konzentriere mich hier auf das Metamodell, das sich in der Ebene M2 befindet und als eine Dokumentvorlage für Level M1 dient.

MOF (Meta Object Facility) ist ein OMG-Standard, der eine gemeinsame und abstrakte Sprache für die Spezifikation von Metamodellen anbietet.

Die SPEM Spezifikation Version 1.1 [1], die im Januar 2005 von der OMG veröffentlicht wurde, ist als ein UML Profil strukturiert und liefert ein komplett auf MOF basierendes Metamodel. Im Prinzip verfeinern Profile nur die Standard Semantik der UML, indem sie weitere Beschränkungen und Interpretationen hinzufügen. SPEM ist der Software Prozess Modellierung gewidmet, so dass viele Besonderheiten der UML, die erforderliche Basis liefern, um Prozesse zu modellieren und viele andere UML Fertigkeiten liefern nützliche zusätzliche Entwurfskapazitäten.

2.2 Motivation

Um Prozesse disziplinierter und strukturierter zu gestalten wurden Varianten der Unified Prozesse in der Industrie-Anwendung benutzt. Prozesse waren selbst wie Produkte und veränderten und entwickelten sich ständig. Damit sie an den Organisations-Wechsel-Bedürfnissen und der Umgebung gerecht wurden, mussten Prozesse geführt und gestaltet werden. Eben der Bedarf an Prozess Modellierungs-Techniken und Technologien war aufgetaucht. Mit der Zeit existierte eine Vielzahl von verschiedenen Prozess-Modellierungs-Sprachen, die alle verschiedene Bezeichnungsweisen oder andere Bedeutung für dasselbe Wort oder Ausdruck verwendeten. In Erkenntnis dieses Bedarfs hat die OMG die SPEM Spezifikation definiert und herausgegeben. Da SPEM eine Untermenge von UML grafischer Notation verwendet und eine breite Gemeinde von Software Entwicklern mit UML vertraut sind, fällt es den Entwicklern, die sich mit anderen Prozessmodellierungssprachen auskennen nicht schwer, mit SPEM Prozesse zu modellieren. Das Definieren eines UML Profils erlaubt es dieser Gemeinde ihre Fachkenntnisse und Tools in der Software-Prozess-Modellierungs-Domäne wieder zu verwenden.

Der Unterschied zwischen UML und SPEM ist, dass UML ein Industriestandard für eine Modellierungssprache (Notation) von Systemen ist. SPEM dagegen ist ein Industriestandard für eine Modellierungssprache von Prozessen. SPEM beschreibt nicht, wie die Planung oder die Durchführung eines Prozesses auszusehen hat. Dafür gibt es genügend andere Projektvorgehensmodelle (z.B.: V-Modell XT (VXT)). Jedoch ist es die primere Aufgabe beider Standards (UML / SPEM), den Austausch von Prozessen und Systemen möglichst einfach zu halten, sodass jeder, der Wissen über UML / SPEM besitzt, diese auch lesen kann.

Grob gesprochen lässt sich SPEM in der Version 1.1 im Verhältnis zur UML folgendermaßen charakterisieren:

- SPEM ist eine definierte Teilmenge der Klassen des UML 1.4 Meta-Modells.
- SPEM definiert spezielle zusätzliche Bedingungen.
- SPEM definiert spezielle Stereotype, welche durch grafische Symbole repräsentiert werden.

Als UML-Profil erbt SPEM bereits die grundlegenden Konzepte wie die Vererbungsbeziehung, Abhängigkeiten und Assoziationen zwischen Modellelementen. Erspart sich jedoch auch

Konzepte wie beispielsweise Interfaces und n-fache Beziehungen. SPEM bietet vergleichbare Diagrammtypen wie die UML selbst an, wobei nur sechs wirklich sinnvoll für die Domäne der Softwareentwicklungs-Prozesse sind

2.3. Kern-Konzept von SPEM

Der Kerngedanke des SPEM ist die Idee, dass ein Softwareentwicklungsprozess ein Zusammenspiel zwischen abstrakten und aktiven Elementen ist (siehe Abbildung 2). Prozessrollen führen Tätigkeiten aus, die Activities (Tätigkeiten) genannt werden. Tätigkeiten werden als Maßeinheit der Arbeit beschrieben, die notwendig sind, um WorkProducts (Arbeitsprodukte) zu produzieren. Jede Prozessrolle kann für die Produktion von mehreren Arbeitsprodukten zuständig sein. Diese drei Klassen sind unbedingt notwendig für die Modellierung von Softwareprozessen. Sie bilden das Grundgerüst der SPEM Hauptklassen.

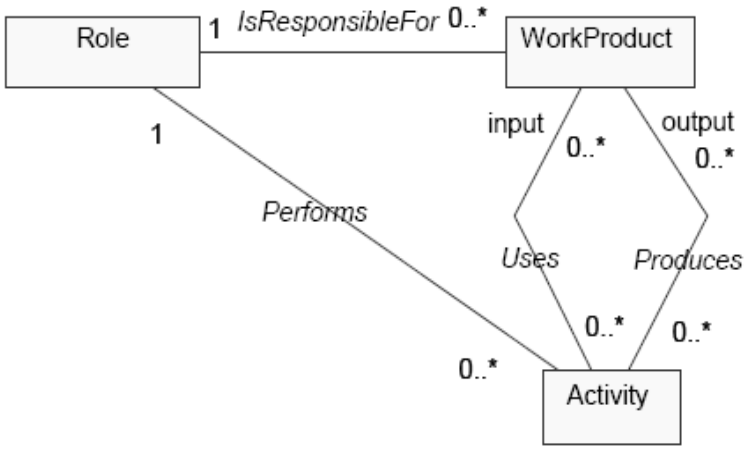


Abbildung 2: Kernkonzept von SPEM aus der SPEM Spezifikation [1]

2.4. SPEM Stereotype

Die nächste Abbildung gibt einen generellen Überblick darüber, welche Beziehungen zwischen den einzelnen SPEM Klassen möglich bzw. erlaubt sind. Das Letztere wird später im Kapitel Regeln zur Wohlgeformtheit noch mal aufgegriffen und genauer beschrieben. SPEM erlaubt nicht alle üblichen Beziehungen (aus der UML) zwischen den Klassen. Da die Regeln zur Wohlgeformtheit bestimmte Verbindungen zwischen den SPEM Elementen fest vorgeben, sind diese für die Umwandlung von Bedeutung.

Die nächste Abbildung stellt den Zusammenhang der zu übersetzenden Klassen dar. Diese und deren Verbindungen untereinander sollen im Folgenden kurz erläutert werden. Graphische Beispiele sollen den Leser dabei unterstützen.

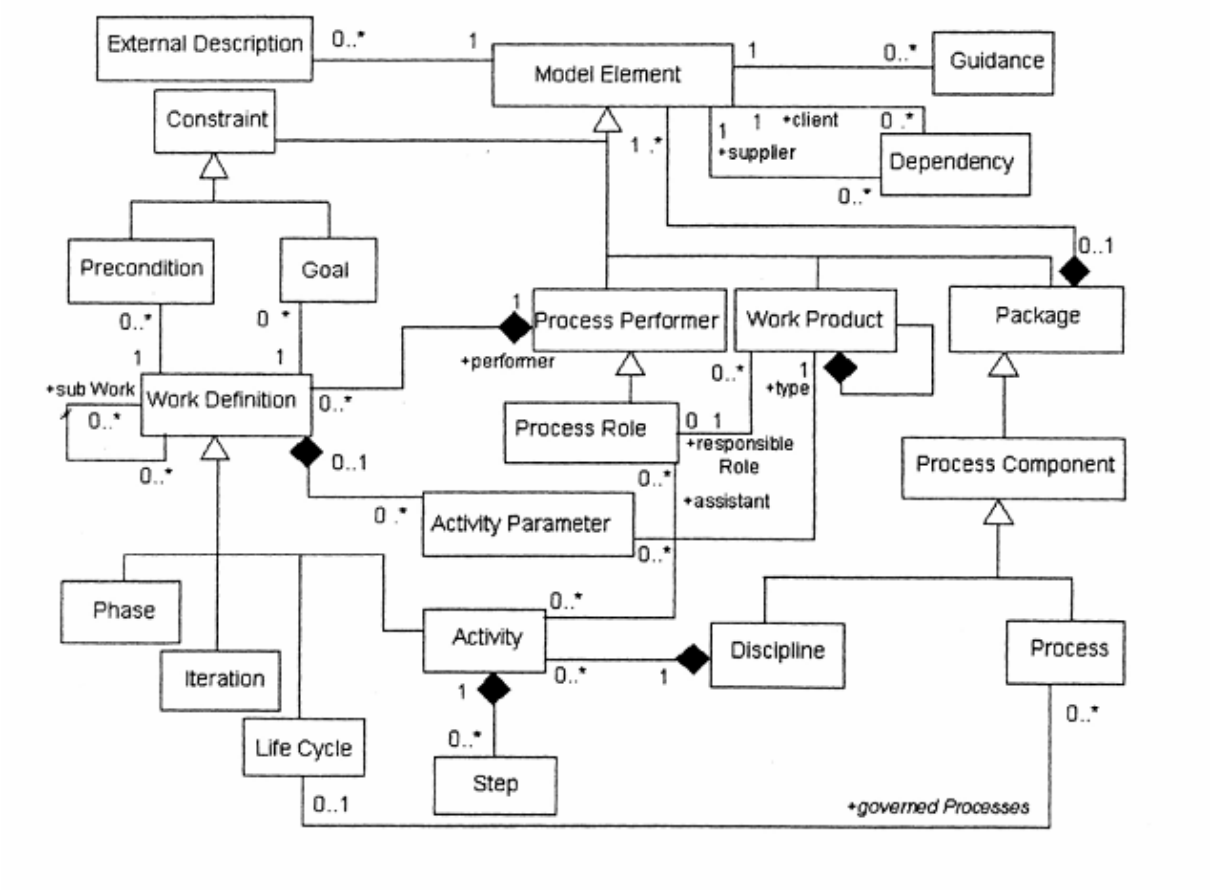


Abbildung 3: SPEM Hauptklassen

Das SPEM Metamodel enthält die folgenden Stereotype, die im Folgenden erklärt werden.

- Basis Elemente: ExternalDescription und Guidance
- Abhängigkeiten: Categorizes, Impacts, Precedes, Governs
- Prozess Struktur: WorkDefinition, WorkProduct, Activity, Step, ProcessRole
- Prozesskomponenten: ProzessComponent, Prozess, Discipline
- Prozesslebenszyklus: Lifecycle, Phase, Iteration, Precondition, Goal

Die Abbildung 3 stellt die Hierarchie zwischen den SPEM-Klassen, in einem vereinfachten Modell dar. Oberklasse aller SPEM Klassen ist das Model-Element, das durch die Klassen Guidance und ExternalDescription genauer beschrieben werden kann. Jedes Element hat eine externe Beschreibung, die die Benutzer sichtbare Schnittstelle liefert. Guidance unterstützen die

Model-Elemente mit denen sie verbunden sind. Sie sind keine WorkProducts (Arbeitsprodukte), weder im Prozess bearbeitet noch entworfen. Die Dependency-Klasse enthält alle möglichen Abhängigkeiten zwischen zwei Model-Elementen. Die Arten der Guidance und Dependency werden im Folgenden kurz beschrieben und im Kapitel Konzept der Umwandlung sind zu den Dependencies explizite Beispiele zu finden. Wichtig bei diesem Modell ist, dass alle Klassen über andere Klassen miteinander in Beziehung stehen. Alle die hier vorkommenden Stereotype werden in SPEM dazu verwendet Prozesse darzustellen**.**

Guidance liefern Anleitungen für alle Modelelemente mit denen sie verbunden sind. Man hat in SPEM die Möglichkeit die Anleitungen genauer zu spezialisieren. Die im Folgenden beschriebenen Guidance sind mögliche Anleitungen für Arbeitsbeschreibungen oder Prozesserzeugnisse.

2.4.1. Anleitungsarten

1. Technique

Eine „Technique“ ist ein ausführlicher genauer Algorithmus, der benutzt wird, um ein Arbeitsprodukt zu entwerfen. „Technique“ hilft, um die Fertigkeiten zu definieren, die erforderlich sind, um gezielte Typen der Activities (Aktivitäten) auszuführen. Eine mögliche „Technique“ wäre Brainstorming in einem Workshop um Anforderungen zu erheben.

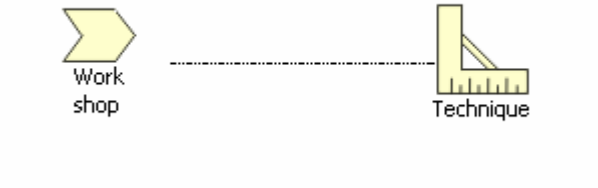


Abbildung 4: Technique

2. UML Profile

Ein „UML-Profil“ ist ein, auf eine bestimmte Domäne angepasste semantische Erweiterung der UML. Gutes Beispiel dafür ist die Modellierung des Softwareprojekts in SPEM (SPEM als UML Profil).

Abbildung 5: UML Profil

3. Checklist

Eine „Checkliste“ ist ein Dokument, das eine Liste der Elemente repräsentiert, die erfüllt werden müssen. In einem Quality-Gate müssen bestimmte Dokumente vorliegen und diese werden mit einer Checkliste kontrolliert.

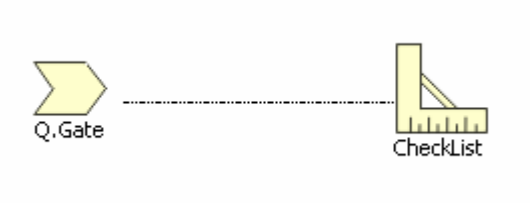


Abbildung 6: Checklist

4. Guideline

Eine „Guideline“ ist eine Reihe von Regeln und Empfehlungen, die festlegen wie ein bestimmtes Arbeitsprodukt aussehen muss oder organisiert sein muss.

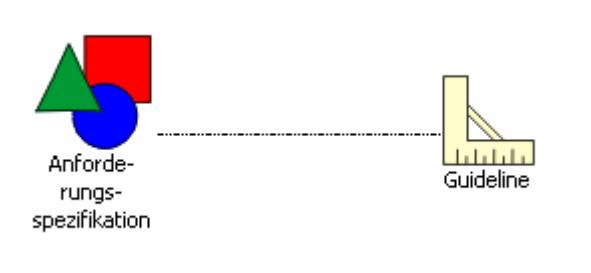


Abbildung 7: Guideline

5. Template

Ein „Template“ ist ein festgelegtes Dokument, das ein standardisiertes Format für bestimmte Teile der WorkProducts (Arbeitsprodukte) liefert: z.B. „Microsoft Word Template für Business UseCase Modellierung.

Abbildung 8: Template

2.4.2. Abhängigkeiten

1. Categorizes

Eine Categorizes Abhängigkeit agiert von einem Paket individuellen Prozess Element zu einem anderen und liefert Hilfsmittel, um Prozesselemente mit verschiedenen Kategorien zu vereinigen. Diese Eigenschaft ist allgemeinen nützlich und fungiert insbesondere in Verbindung mit Discipline (Disziplinen), um eine erste Kategorisierung aller Elemente zur Verfügung zu stellen.

2. Impacts

Eine Impacts Abhängigkeit agiert von einem Arbeitsprodukt zu einem anderen Arbeitsprodukt, um anzudeuten dass die Modifikation eines Arbeitsprodukts ein anderes entkräften könnte.

3. Precedes

Eine Precedes Abhängigkeit agiert von einer Aktivität zu einer anderen oder von einer WorkDefinition (Arbeitsbeschreibung) zu einer anderen, um anzudeuten, ob es eine Ende-Start oder Ende-Ende Abhängigkeit zwischen der beschriebenen Arbeit ist.

4. Governs

Das Ziel-Element wird vom Quell-Element geregelt.

Alle weiteren SPEM-Abhängigkeiten sind für die Arbeit nicht von Bedeutung.

2.4.3. Prozessstruktur

:

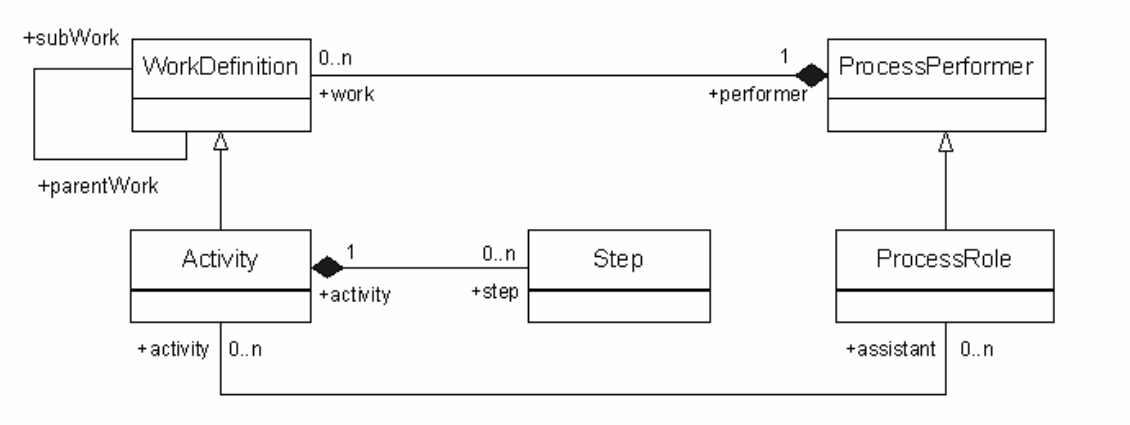


Abbildung 9: Processstruktur aus der SPEM Spezifikation [1]

Workdefinition

Eine Arbeitsbeschreibung ist eine Art Arbeitsablauf, der die Tätigkeiten, die in einem Prozess durchgeführt werden beschreibt. Ihre Hauptunterklasse ist die Aktivitäts- Klasse, aber auch Phase, Iteration und Lifecycle, sind ebenfalls Unterklassen der Arbeitsbeschreibung. Eine Arbeitsbeschreibung kann aus anderen Arbeitsbeschreibungen durch die Verwendung von Verbindungen genannt „subwork“ bestehen (siehe Abbildung 9). Die Aufspaltung kann mit einem Aktivitätsdiagramm modelliert werden. In so einem Fall wird die „subwork“ Assoziation von der Aktivitätsdiagrammstruktur abgeleitet. Eine Arbeitsbeschreibung hängt mit dem Arbeitsprodukt zusammen, das es durch die AktivityPrameter (Aktivitäts-Parameter) Klasse verwendet, die festlegt ob sie als Input oder Output verwendet wurden. Eine Arbeitsbeschreibung hat einen verantwortlichen ProcessPerformer (Prozessbearbeiter), der die Primärrolle darstellt, die diese Arbeitsbeschreibung im Prozess durchführt. Im Falle der Aktivitäten, die von einem Einzelnen oder von einer kleinen Gruppe durchgeführt werden, ist dies eine Prozessrolle und im Falle hochgradigen Arbeitsbeschreibungen ist dies häufig eine einzige Instanz der ProcessPerformer, die dem kompletten Prozess entspricht.

Activity (Aktivität) und Step (Schritt)

Aktivität ist die Hauptunterklasse der Arbeitsbeschreibung. Sie beschreibt ein Teil der Arbeit, die von einer Prozessrolle durchgeführt wird. Eine Aktivität kann aus mehreren Schritten bestehen, die Steps genannt werden.

Role (Prozessrolle)

Eine Prozessrolle ist verantwortlich für die Produktion von WorkProducts. Sie kann aus mehreren Personen bestehen und jede Person darf an mehreren Rollen beteiligt sein. Eine Prozessrolle kann eine Aktivität ausführen oder an ihr beteiligt sein.

WorkProdukt

Ein Arbeitsprodukt ist etwas, in einem Prozess Produziertes, Konsumiertes oder Modifiziertes. Es kann eine Art Information, ein Dokument, ein Model, Source Code usw. sein.

2.4.4. Prozesskomponenten

ProzessCoponent (Prozesskomponente)

Eine Prozesskomponente ist eine Menge von Prozessbeschreibungen, die intern konsistent und abgeschlossen sind und möglicherweise aus anderen Prozesskomponenten zusammengesetzt werden können, um einen ganzen Prozess darzustellen. Eine Prozesskomponente importiert einen unwillkürlichen Satz von Prozess Definitionen, die in SPEM durch Modelelemente modelliert sind.

Process

Ein Prozess ist ein spezieller Prozessbaustein, der den gesamten Prozess darstellt. Die Intention liegt daher weniger in der Verknüpfung mit anderen Prozessbausteinen, sondern eher in der ganzheitlichen Darstellung. Die Klasse „Prozess“ kann eine Familie von Prozessen darstellen, die ein Prozessbestandteil ist, aus dem heraus mehrfache überlappende Prozesse definiert werden können.

Discipline (Disziplinen)

Eine Disziplin ist eine Spezialisierung der Pakete, die die Aktivitäten innerhalb eines Prozesses dementsprechend zu einem Thema untergliedert. Die Unterteilung auf diese Weise besagt, dass die assoziierten Guidance- und Output-Arbeitsprodukte in gleicher Weise unter dem Thema kategorisiert sind. Das Einbinden einer Aktivität in eine Disziplin wird repräsentiert durch die Categorizes-Abhängigkeiten, mit der zusätzlichen Beschränkung, dass jede Aktivität exakt durch eine Disziplin kategorisiert wird.

2.4.5. Prozesslebenszyklus

Phase

Eine Phase ist eine Spezialisierung der Arbeitsbeschreibung, so dass seine Vorbedingung die Phaseneingangskriterien definieren und sein Ziel definiert die Phasenausgangskriterien. Phasen sind mit der zusätzlichen Einschränkung der Sequenzialität definiert; d.h. ihre Inkraftsetzung wird mit einer Serie von Meilensteinen ausgeführt und oft nimmt man minimale Überlappung von ihren Tätigkeiten rechtzeitig an.

Precondition (Vorbedingung) and Goal (Ziel)

Mit jeder Arbeitsbeschreibung können Vorbedingung und Ziele verbunden werden.
Vorbedingung enthält die Bedingung, die für den Beginn einer Arbeitsbeschreibung notwendig ist und das Ziel definiert den Status eines Prozessartefakts nach Ablauf dieser Aktivität (siehe Abbildung 10).

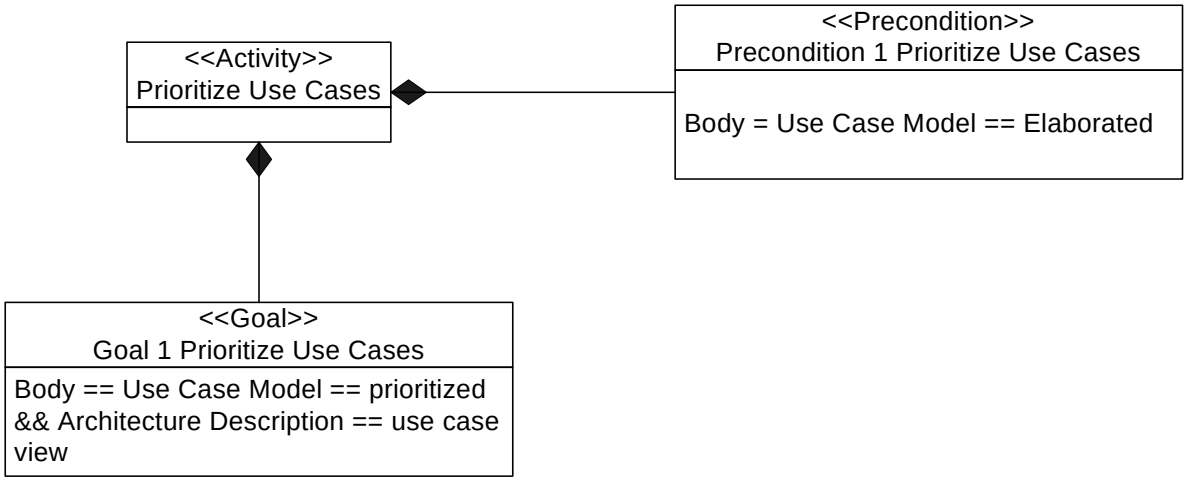


Abbildung 10: Beispiel für eine Precondition und Goal aus dem SPEM-Tutorial [4]

Lebenszyklen

Ein Lebenszyklus ist als eine Reihenfolge von Phasen definiert, die ein spezifisches Ziel erreichen. Es definiert das Verhalten von einem vollständigen Prozess, in einem gegebenen Projekt erlassen zu werden oder programmiert zu werden.

Iteration

Eine Iteration ist eine Folge von Aktivitäten, die durch einen kleinen Meilenstein abgeschlossen wird.

2.5. Diagrammtypen

UML Diagramme können benutzt werden, um verschiedene Perspektiven von Software-Prozess-Modellen zu veranschaulichen. SPEM benutzt folgende Diagrammtype:

- Klassendiagramm
- Aktivitätsdiagramm
- Use Case Diagramm
- Paketdiagramm
- Sequenzdiagramm
- Statechart Diagramm

Dadurch dass der SPEM-Editor nicht alle möglichen Diagrammtypen ermöglicht (mehr dazu im Kapitel Test und Probleme) werden in dieser Arbeit nur die Use-Case-, Aktivitäts- und Klassendiagramme behandelt.

2.5.1. Klassendiagramm

Klassendiagramme erlauben die Präsentation von folgenden Aspekten der Software Prozesse:

- Erbschaft: wird häufig dann eingesetzt, wenn eine Klasse eine speziellere Ausprägung einer anderen Klasse ist.
- Abhängigkeiten: stellt eine Abhängigkeit zwischen zwei Klassen dar
- Einfache Assoziation: stellt eine Beziehung zwischen zwei Klassen dar
- Kommentar, um auf die Guidance aufmerksam zu machen (zum Beispiel URL Verbindung)
- Verhältnisse zwischen ProzessPerformer oder ProcessRole und Arbeitsprodukt
- Struktur, Auflösung, und Abhängigkeiten von Arbeitsprodukten

Folgende Aspekte sollten jedoch nicht repräsentiert werden:

- Interface
- Template
- Ungefüllte Raute
- Qualifizierte Assoziationen
- N-fache Assoziationen

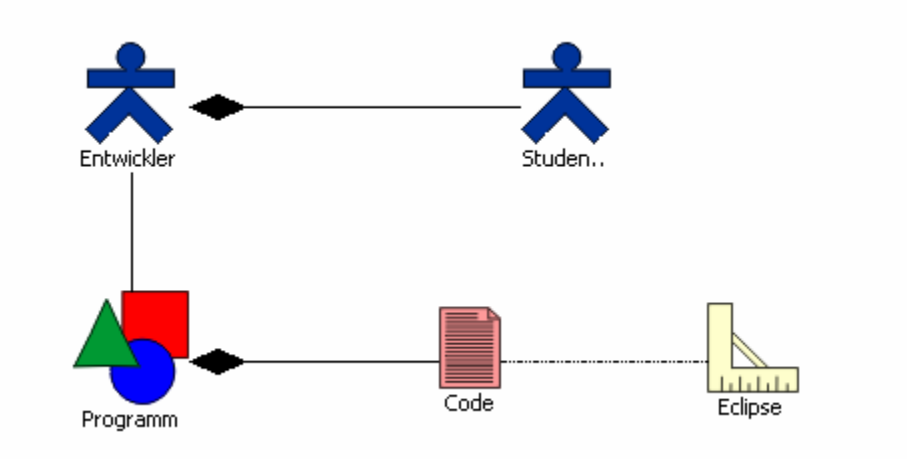


Abbildung 11: Beispiel für ein SPEM Klassendiagramm

Abbildung 11 veranschaulicht ein Klassendiagramm in SPEM. Dieses Modell stellt ein Beispiel aus dem Softwareprojekt dar. Es beschreibt einige, der oben erwähnten Beziehungen zwischen den Elementen. Wichtig hierbei ist dass in SPEM keine „enthalten in“ Abgängigkeiten (ungefüllte Raute) möglich sind. Deshalb wird überwiegend die Beziehung „besteht aus“ (gefüllte Raute) verwendet. Dabei besteht die Klasse zu dem der Diamant zeigt aus der Klasse von dem die Verbindung ausgeht.

2.5.2. Use-Case-Diagramm

Use-Case-Diagramme beschreiben die Beziehungen zwischen Prozessrollen und den Haupt-Arbeitsbeschreibungen. Sie zeigen das nach außen sichtbare Verhalten eines Elements.

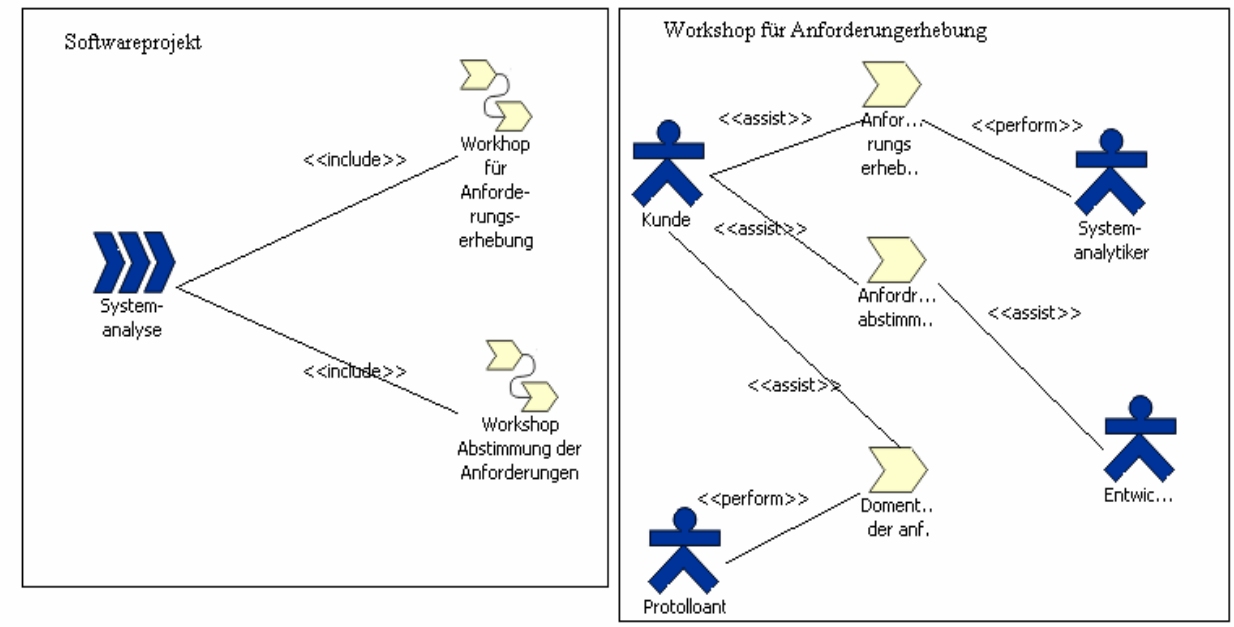


Abbildung 12: Beispiele für SPEM Use-Case-Diagramm

2.5.3. Aktivitätsdiagramm

Aktivitätsdiagramme zeigen den sequenziellen Ablauf von Aktivitäten mit ihren Input- und Output-Arbeitsprodukten. Swimlanes können die Verantwortlichkeitsbereiche der ProcessRoles (Prozessrollen) abgrenzen (siehe Abbildung 13).

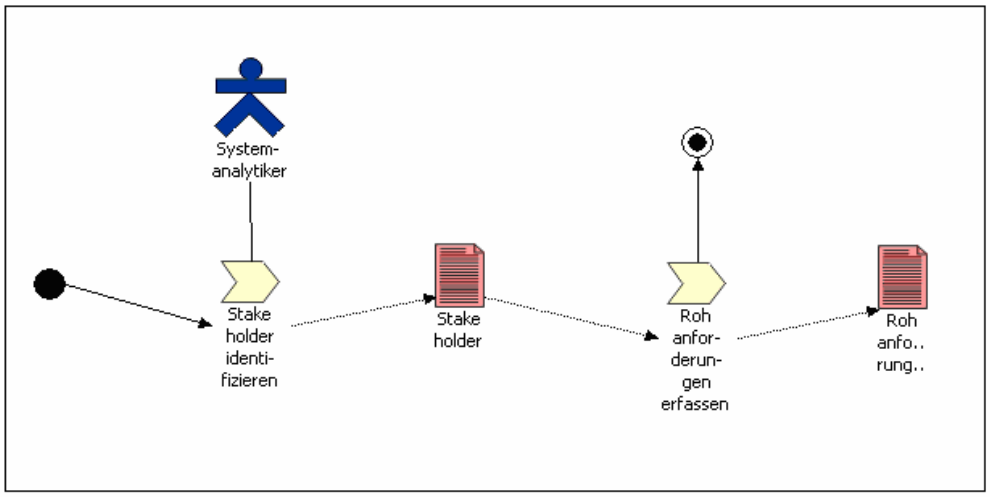


Abbildung 13: Beispiel für ein SPEM Aktivitätsdiagramm

2.6. Regeln zur Wohlgeformtheit

Die Regeln zur Wohlgeformtheit aus der SPEM-Spezifikation [1] definieren Grenzen zwischen den Beziehungen der einzelnen Elemente. Sie legen fest welche Verbindungen mit welchen Elementen erlaubt sind. Die folgenden Regeln zur Wohlgeformtheit aus der SPEM-Spezifikation sind für diese Arbeit relevant.

Namensraum

[C1]	Alle Assoziationen müssen einen eindeutigen Namen und dazugehörigen Classifier in dem Namensraum haben.

Beschränkungen

[C2]	Eine Beschränkung kann nicht auf sich selbst bezogen sein.

Assoziations-Ende

[C3]	Eine Instanz kann nicht durch eine Komposition zu mehr als einer Kompositions- Instanz gehören.

Endzustand

[C4]	Ein Endzustand kann nicht ausgehende Übergänge haben.

Anfangsknoten

[C5]	Ein Anfangsknoten darf höchstens einen Ausgehenden und keine eingehenden Verbindungen haben.

Categorizes

[C6]	Die Quelle muss eine Art Pakete sein.

Impacts

[C7]	Die Quelle und das Ziel müssen eine Art Arbeitsprodukt sein.

Import

[C8]	Die Quelle und das Ziel müssen eine Art Paket sein.

Precedes

[C9]	Die Quelle und das Ziel müssen eine Art Arbeitsbeschreibung sein.

Aktivität

[C10]	Jede Aktivität ist genau von einer Disziplin importiert.
[C11]	Jede Aktivität gehört zu einer Prozessrolle.

Prozessrolle

[C12]	Jede Arbeit muss eine Art Aktivität sein.

Step

[C13]	Ein Step hat keine assoziierten Aktionen.

Arbeitsbeschreibung

[C14]	Wo ein Aktivitätsgraph ist, ist ein subWork abgeleitet.
[C15]	Eine Arbeitsbeschreibung darf nicht mehr als eine Vor- und Nachbedingung haben [1].

Prozesskomponenten

[C16]	Keine Abhängigkeiten außerhalb von Prozesskomponenten.
[C17]	Keine Assoziationen außerhalb von Prozesskomponenten.

Lifecycle

[C 18]	Lebenszyklus darf nur Phasen beinhalten [1].

Fork

[C20]	Eine Verzweigung muss genau eine eingehende und mindestens zwei ausgehende Verbindungen besitzen
[C21]	Ein Verbund muss mindestens zwei eingehende und genau einen ausgehende Verbindung besitzen

3. Konzept der Umwandlung

In diesem Kapitel wird beschrieben wie SPEM in Prolog umgewandelt wird. Damit es nicht den Rahmen dieser Arbeit sprengt, werde ich nur kurz auf XSLT und Prolog eingehen. Danach werde ich die Umwandlung von SPEM in Prolog an Hand des Softwareprojekt-Beispiels beschreiben. Das Softwareprojekt soll dem Leser erleichtern die Transformation besser zu verstehen. Gezielte Beispiele im kleinen Kontext sollen die technische Umwandlung verdeutlichen, sind jedoch nicht vollständig. Die vollständige Übersetzung der Stereotypen befindet sich im letzten Teil dieses Kapitels und soll dazu dienen, dem Leser einen besseren Überblick zu geben. Im Anhang befindet sich dann die komplette Übersetzung des Softwareprojekts in Prolog.

3.1 Komponenten der Umwandlung

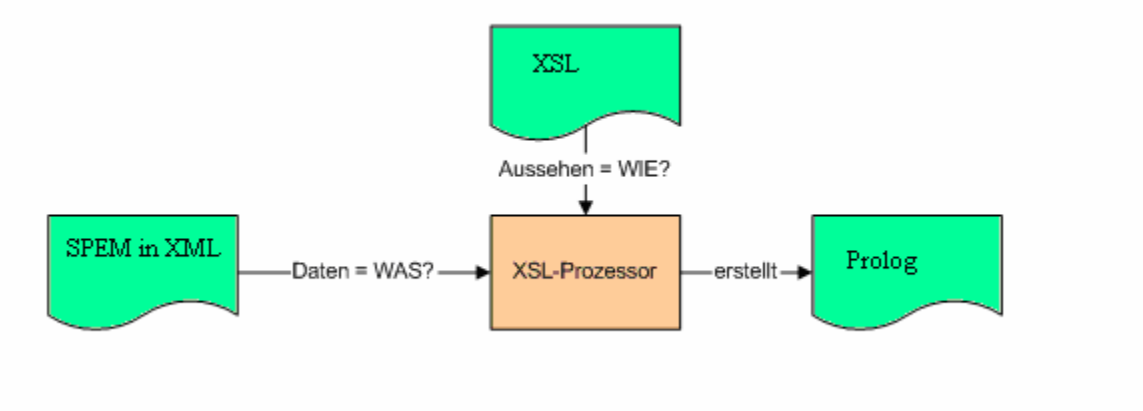


Abbildung 14: Konzept der Umwandlung

Die Abbildung 14 beschreibt den Fortgang der Umwandlung. SPEM wird in XML abgespeichert und zusammen mit dem XSL Stylesheet dem XSL Prozessor übergeben. Ich habe in dieser Arbeit das ANT Skript in Eclipse laufen lassen, der die Umwandlung von SPEM in diesen durchführt.

3.1.1. Extensible Stylesheet Language Transformations (XSLT)

XSLT ist ein Werkzeug, um XML Daten mit Hilfe eines Stylesheets in ein beliebiges Format zu konvertieren. Stylesheets sind XSLT-Programme, die die Umwandlungsregeln definieren und sind dabei ebenfalls nach den Regeln des XML-Standards aufgebaut. Ein XSLT Prozessor liest das Stylesheet ein und transformiert ein oder mehrere XML Dokumente nach den Stylesheet-Regeln in das gewünschte Ausgabeformat (siehe Abbildung 14). Dabei spricht man von der Transformation von einem Quellbaum in einen Ergebnisbaum. Innerhalb von XSLT werden Templates verwendet. Ein Template ist eine Vorlage für die Transformation bestimmter Knoten.

3.1.2. Prolog

Prolog ist eine logische Programmiersprache. Prolog-Programme bestehen aus einer Datenbasis, die Fakten und Regeln umfassen. Fakten können durch Prädikate beschrieben werden. Der Benutzer formuliert Anfragen an die Datenbasis. Der Prolog-Interpreter benutzt die Fakten und Regeln, um systematisch eine Antwort zu finden. Prädikate werden in Prolog und in dieser Arbeit durch „Prädikatsnamen/n“ beschrieben. Dabei steht das „n“ für die Anzahl der Stellen in diesem Prädikat. Entscheidend für diese Arbeit ist, dass Wörter, die mit Grossbuchstaben beginnen, in Prolog Variablen darstellen. Deshalb müssen die Namen der Elemente im SPEM-Editor mit Kleinbuchstaben beginnen.

3.2. Darstellung des Softwareprojektes in SPEM

Um das Softwareprojekt in SPEM zu modellieren habe ich mich in erster Linie auf die Bachelorarbeit von Marco Nötel [5] und meinen eigenen Erfahrungen aus dem Softwareprojekt bezogen. An manchen Stellen habe ich den Entwicklungsprozess mit fehlenden Elementen ergänzt

Das Softwareprojekt ist ein dreimonatiges Softwareentwicklungsprojekt in dem Studenten in 5-6 Personen-Gruppen die, Ihnen zu geteilten Aufgaben erfüllen. Dies soll den Studenten die Gelegenheit geben an einem echten Projekt zu arbeiten, um Erfahrungen zu sammeln. Meistens müssen sie ein Programm oder ein Spiel entwickeln, wobei das strukturierte Arbeiten mit echten Kundengesprächen und den verschiedenen Phasen im Vordergrund stehen soll [5]. In der Folgenden Abbildung 14 werden alle wichtigen Bestandteile des Softwareprojektes dargestellt und anschließend kurz beschrieben. Ein Softwareprojekt besteht aus Studenten und den Mitarbeitern aus dem Fachbereich „Software Engineering“, die die in der Abbildung 14 aufgeführten Rollen einnehmen. Die Dokumente werden in dem Prozess erstellt oder dienen als Erfahrungshilfen aus früheren Projekten.

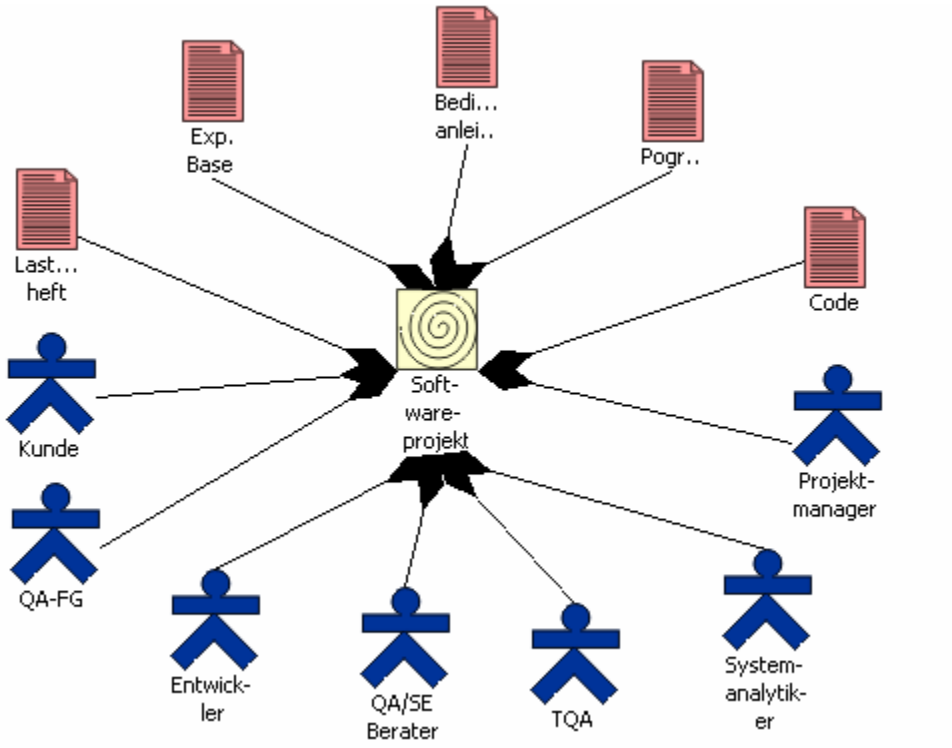


Abbildung 15: Rollen und Bestandteile des Softwareprojektes

Kunde: Der Kunde ist der Auftraggeber und Abnehmer der Software. In diesem Fall einer aus dem Institut.

Entwickler: Mit Entwicklern sind alle Studenten gemeint, die an der Softwareentwicklung arbeiten. Unter anderem Anforderungen erheben und Code schreiben, wobei ich hier noch für die Anforderungserhebung die Rolle „System Analytiker“ zusätzlich aufgeführt habe.

System Analytiker: Führt Interviews mit dem Kunden durch, soll alle Anforderungen aus dem Kunden „herauskitzeln“ und diese dann dokumentieren. Falls Missverständnisse existieren, sie finden und aufklären.

QA/SE Berater: Der QA/SE-Berater ist kein Kursteilnehmer sondern jemand aus dem Betreuerstab. Er berät die Studenten bei schwer lösbaren Problemen und fungiert bei allgemeinen Schwierigkeiten als Ansprechperson.

TQA: Der „technische Qualitätsverantwortliche des Teams“ ist für die Einhaltung von Qualitätssicherheitsmaßnahmen während der gesamten Entwicklung verantwortlich. Er informiert außerdem die Entwickler über ‚requirements’ für die anstehenden Quality Gates und wahrt die Einhaltung der formalen Richtigkeit der Dokumente.

Projektmanager: Der Projektmanager tritt nur bei der Abnahme der Software durch den Kunden auf. Er vertritt die Entwickler hier als Hauptverantwortlicher für das entstandene Produkt, selber ist er Mitglied des Betreuerstabes.

QA-FG: Der ‚Qualitätsverantwortliche der Firma FunGate' tritt nur in den Quality Gates auf und repräsentiert dort den ‚gatekeeper'. Er bestimmt, ob das Entwicklerteam mit der Arbeit Erfolg hatte und folglich das Quality Gate passieren darf oder nicht.

Lastenheft: Das Lastenheft gibt in natürlicher Sprache die Wünsche und Ansprüche des Kunden an die Software wieder. Aus diesen wird von den Entwicklern die Anforderungsspezifikation erarbeitet.

Exp.-Base: Die Experience-Base ist eine Datenbank, in welche vor allem die Entwickler, aber auch alle anderen Rollen Notizen für sich machen können. Der Nutzen einer Experience-Base resultiert vielleicht nicht im ersten Projekt, aber nach und nach sammeln sich dort eine Menge Erfahrungen. Die Erfahrungen können reichhaltiger Natur sein. Es können sowohl technische Dinge notiert sein, aber auch vorteilhaftes Verhalten gegenüber den Kunden oder aber ein sinnvoller Prozessablauf. Insgesamt gesehen ist die Experience-Base ein Instrument aus dem Bereich des erfahrungsbasierenden Lernens und Verbesserns.

Bedienungsanleitung: Sie klärt den Kunden über die Funktionsweise und Handhabung des Programms auf.

Code: Hiermit ist sämtlicher Quellcode und alle Bibliotheken, welche für das Kompilieren der Software notwendig ist gemeint.

Programm: Das Programm (=Software) ist das Produkt des Softwareentwicklungsprozesses. Es wird an den Kunden ausgeliefert.

Lid – Reflexionsdokumentation: Die Reflexionsdokumentation LID (engl. Deckel) schließt das Projekt ab, in dem es alle Erfahrungen sammelt. Es wird vom Experience-Engineer erstellt.

3.2.1 Softwareprojekt Aktivitätsdiagramm

Das Softwareprojekt besteht aus den 5 Phasen, die in Abbildung 15 dargestellt werden. Wobei die Erste und die Letzte Phase nicht direkt in den Entwicklungsprozess eingehen sondern vielmehr für Studenten gedacht sind, um sich vorzubereiten und am Ende Erfahrungen auszutauschen. Das Aktivitätsdiagramm aus Abbildung 15 werde ich mittels eines XSLT Stylesheets in Prolog überführen. Jedes Element in der Abbildung wird zu einem einstelligen Prädikat mit dem Typ und dem Namen des Elements. Die Verbindungen zwischen den Elementen stellen in SPEM bestimmte Beziehungen zwischen den Entitäten dar. Diese werden (außer diese, an denen ein Startnode oder Endnode beteiligt sind) in zweistellige Prädikate übersetzt.

- Die Verbindung (durchgezogene Kante mit einer Richtung) wird in dieser Arbeit zwischen Arbeitsbeschreibungen verwendet und gibt deren Abfolge an.

- Die Verbindung (gestrichelte Kante mit einer Richtung) wird zwischen Arbeitsprodukten und Prozessratefakten verwendet und beschreibt deren Ein- und Ausgaben.

- Die Verbindung (durchgezogene Kante ohne Richtung) zwischen Prozessrollen und Prozesselementen ordnet den Elementen die verantwortlichen Rollen zu.

- Die Verbindung (gestrichelte Kante ohne Richtung) wird nur für Anleitungen verwendet

- Alle weiteren Verbindungen haben eine beschriftete Kante, die die Beziehung zwischen den Elementen beschreibt.

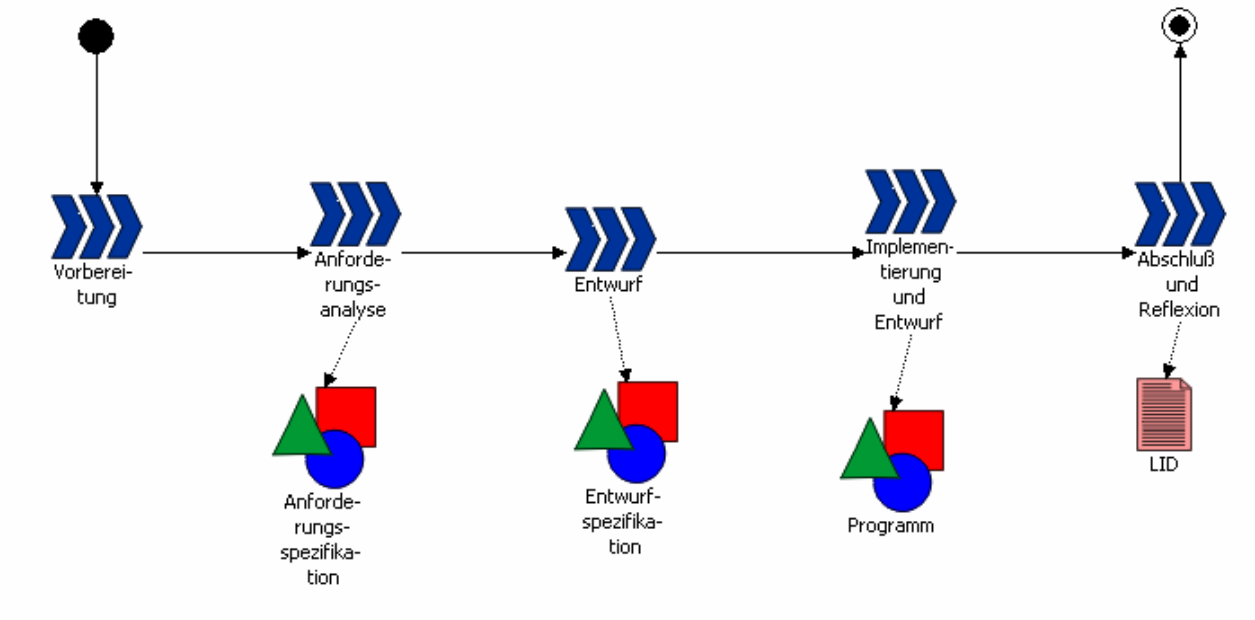


Abbildung 16: SPEM-Aktivitätsdiagramm

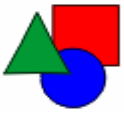
Jedes Element in SPEM wird in ein einstelliges Prädikat transformiert.

Element-Typ(Elementname).


Entwurf

wird zu phase (entwurf).

Abbildung 17: Notation für eine Phase


Programm

wird zu workproduct (programm).

Abbildung 18: Notation für ein WorkProdukt

Jede Verbindung (Transition) wird in zweistellige Prädikate transformiert, außer wenn eines der Elemente zwischen denen die Verbindung besteht ein Startknoten oder Endknoten darstellt (Abbildung 19 und 20).

Wenn der Anfangsknoten der gestrichelten Verbindung eine WorkDefintion oder eine Unterklasse der WorkDefinition (Lifecycle, Phase, Activity, Iteration oder Step) beschreibt und der Endknoten ein Workprodukt oder etwas anderes im Prozess erstelltes Element ist (siehe Abbildung 19), entsteht daraus in Prolog ein zweistelliges Prädikat phase_output/2.

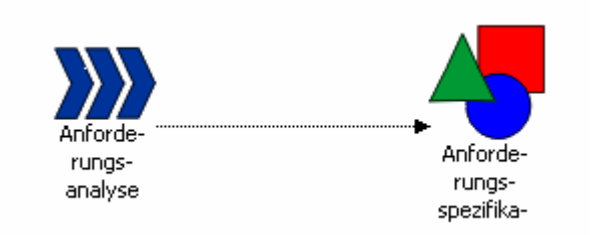


Abbildung 19: Beispiel für eine Phasen-Ausgabe

Die Abbildung 19 transformiert in Prolog sieht wie folgt aus:

phase_output (anforderungsanalyse, anforderungsspezifikation).

Dieses Prädikat besagt, dass das Arbeitsprodukt (WorkDefinition) „Anforderungsspezifikation“ in der Phase „Anforderungsanalyse“ erzeugt wird.

Wenn die Verbindung (durchgezogene Transition) als Anfangsknoten einen „Startnode“ besitzt und als Endknoten eine WorkDefinition oder Unterklassen dieser Workdefinition enthält (siehe Abbildung 20), gilt das einstellige Prädikat: startphase.

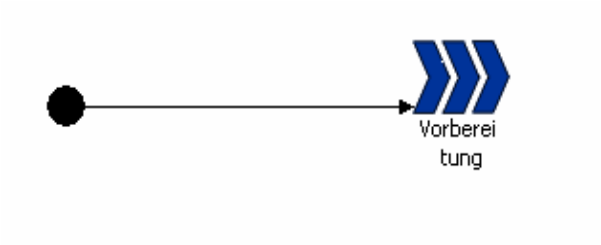


Abbildung 20: Beispiel für eine Startphase

Die Abbildung 20 übersetzt in Prolog sieht wie folgt aus:

start_phase (vorbereitung).

Wenn die Verbindung (durchgezogene Transition) als Anfangsknoten eine WorkDefinition oder Unterklassen dieser WorkDefinition besitzt und als Endknoten ein Endnode enthält
(siehe Abbildung 21), dann gilt das einstellige Prädikat: endphase.

Abbildung 21: Beispiel für eine Endphase

Die Abbildung 21 übersetzt in Prolog:

end_phase (abschluss_und_reflexion).

Wie bereits in Kapitel „Regeln zur Wohlgeformtheit“ erläutert definiert SPEM auch Einschränkungen bei den Beziehungen zwischen den SPEM Klassen. Die Folgenden Abhängigkeiten aus der SPEM Spezifikation [1] werden in dieser Arbeit übersetzt.

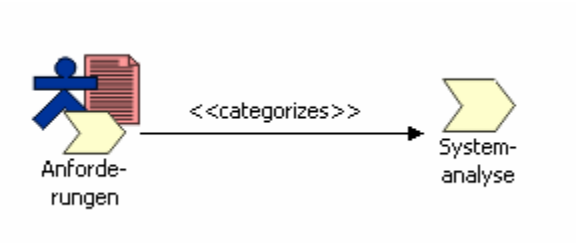


Abbildung 22: Categorizes-Abhängigkeit

Die Categorizes-Abhängigkeit ist nur zwischen einem Paket und einem anderen Modellelement erlaubt.

Abbildung 22 übersetzt in Prolog wird zu einem zweistelligen Prädikat:

categorizes (anforderungen, systemanalyse).

Dieses Prädikat besagt, dass das Modellelement „activity“ dem Prozesspaket zugeordnet und damit kategorisiert wird

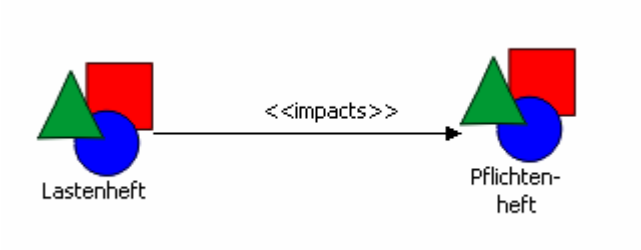


Abbildung 23: Impacts-Abhängigkeit

Diese Art von Abhängigkeit ist nur zwischen WorkProducts erlaubt und besagt, dass das erste WorkProduct das Zweite beeinflusst.

Übersetzt in Prolog entsteht daraus das Prädikat:

impacts (lastenheft, pflichtenheft).

Die nächste Abhängigkeit zwischen SPEM Elementen ist die Precedes-Abhängigkeit. Diese besteht nur zwischen Arbeitsbeschreibungen und legt die Reihenfolge von Arbeitsaufgaben fest.

Abbildung 24: Precedes-Abhängigkeit

Die Precedes-Abhängigkeit übersetzt:

precedes (interpretation,dokumentation).

Diese Übersetzung beschreibt, dass die Interpretation vor der Dokumentation kommt.

Die Governs-Abhängigkeit agiert zwischen zwei Elementen und besagt, dass das Zielelement von dem Quellelement geregelt wird (Abbildung 25).

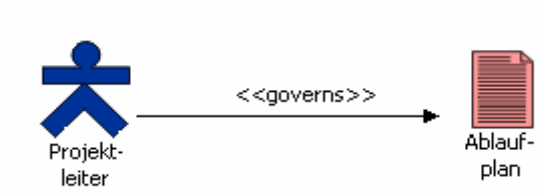


Abbildung 25: Governs-Abhängigkeit

Die Abbildung übersetzt in Prolog:

governs (projektleiter,ablaufplan).

In Aktivitätsdiagrammen hat man die Möglichkeit Abläufe durch ein fork zu synchronisieren oder zu parallelisieren. Hierbei muss man natürlich die Regeln zur Wohlgeformtheit für ein fork (siehe Kapitel 2) beachten. In der Abbildung 26 wird das gesamte Softwareprojekt mit den jeweiligen Phasen den Quality Gates und allen Eingaben und Ausgaben der Phasen beschrieben. Eine Transformation dieser Abbildung ist im Anhang zu finden. Diese Abbildung soll auch mit dazu dienen die Funktionale Korrektheit des Programms zu prüfen. Im Folgenden möchte ich jedoch noch auf die Beispiele der SPEM Aktivitätsdiagramme eingehen, die bisher noch nicht beschrieben wurden

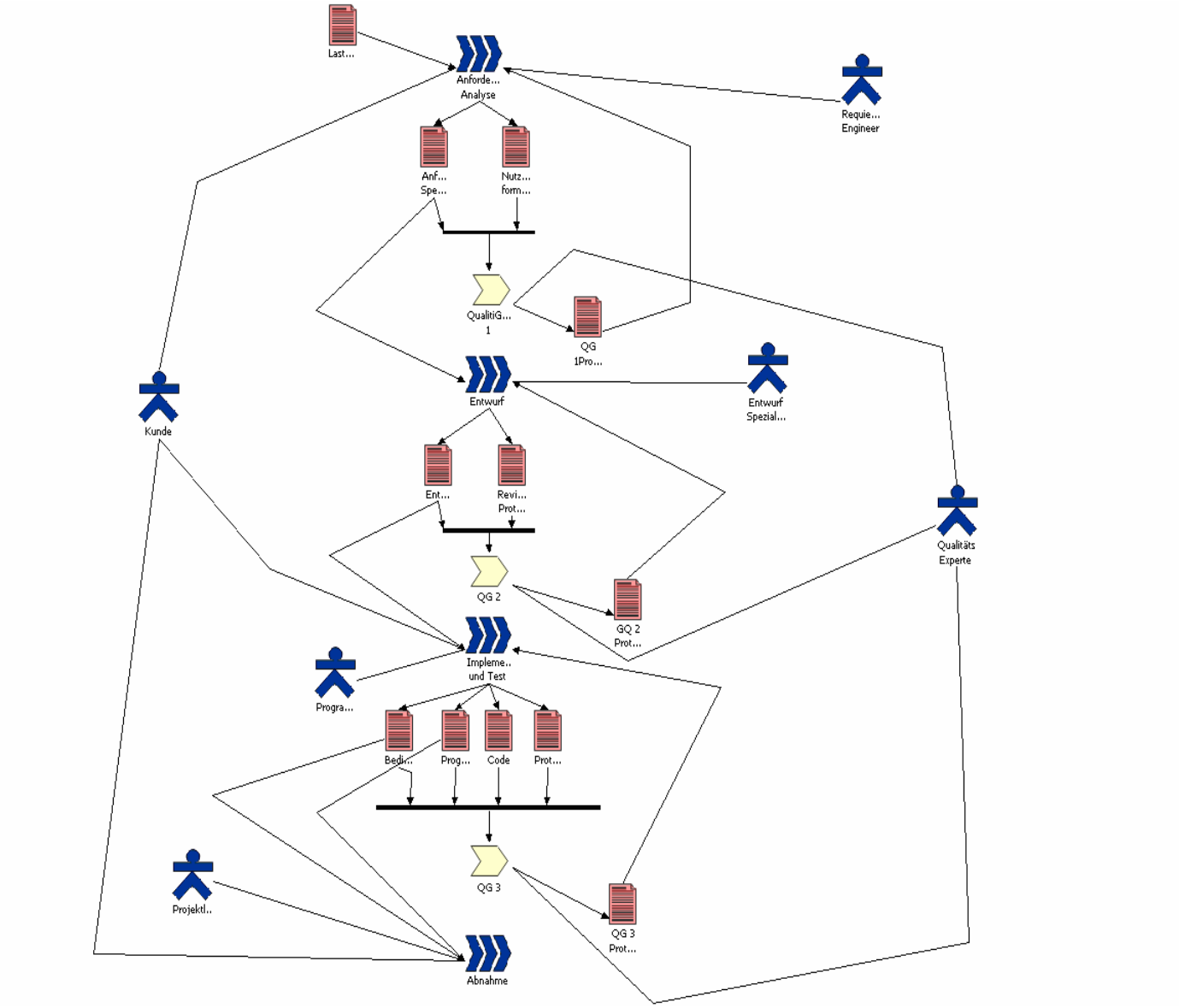
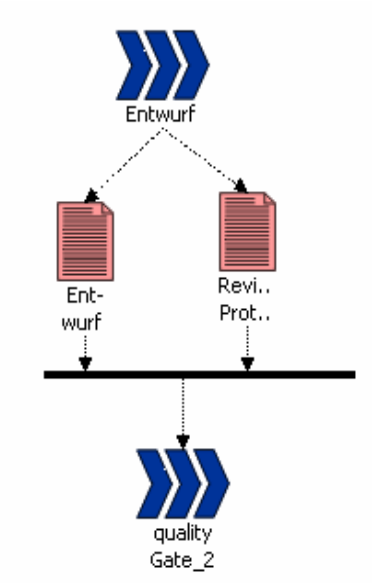


Abbildung 26: Aktivitätsdiagramm Softwareprojekt

Wenn ein Dokument oder ein anderes Prozessartefakt Ausgabe einer Arbeitsbeschreibung ist und direkt mit einem fork verbunden wird, der wiederum mit einer Folgephase verbunden ist (siehe Abbildung 27), so soll es eine Prolog Regel geben die, das zweistellige Prädikat phase_input herleitet.

phase_input (Phase1, Input):- verbunden (Input, Fork_ID), verbunden (Fork_ID, Phase1).



Die Phase Entwurf aus der Abbildung 27 hat zwei Dokumente als Ausgaben und diese Dokumente verschmelzen in die Folgephase Guality Gate2, in der geprüft wird, ob alle Kriterien für eine Weiterentwicklung erfüllt sind.

Abbildung 27: Beispiele für ein fork

Transformiert in Prolog ergibt die Abbildung 27 die neben den einstelligen Prädikaten, die die Elemente ermitteln die folgenden zweistelligen Prädikate:

phase_output (entwurf, entwurf).
phase_output(entwurf, review-protokoll).
verbunden (entwurf, Fork_ID).
verbunden (review-protiokoll,Fork_ID).
verbunden (Fork_ID, guality_gate2).

Wenn auf diese Prädikate die Regel „phase_input“ angewendet wird gelten die Prädikate:

phase_input (quality_gate2, entwurf).
phase_input (quality_gate2, review_protokoll).

Wenn zwischen zwei aufeinander folgenden Phasen ein Dokument oder ein Arbeitsprodukt liegt (siehe Abbildung 28) so soll es eine Regel in Prolog geben, die das Prädikat follow_phase/2 erzeugt.

follow_phase(Phase1,Phase2):-phase_output (Phase1,Element), phase_input (Element,Phase2).

Abbildung 28: Beispiel für eine followphase

Die Abbildung 28 wird in Prolog (nach dem die obige Regel angewendet wurde) zu:

follow_phase (quality_gate2, entwurf).

3.2.2. SPEM Use-Case-Diagramm

Use-Case-Diagramme in SPEM beschreiben die Arbeitsabläufe der Prozesse und ordnen ihnen verantwortliche Rollen zu. In der Abbildung 29 wird die Phase der Anforderungsanalyse aus Abbildung 24 in einer tieferen Ebene betrachtet. Die Rollen werden den einzelnen Aktivitäten zugeordnet, die sie ausführen, wobei mehrere Rollen an einer Aktivität teilnehmen können aber unterschiedliche Aufgaben ausführen.

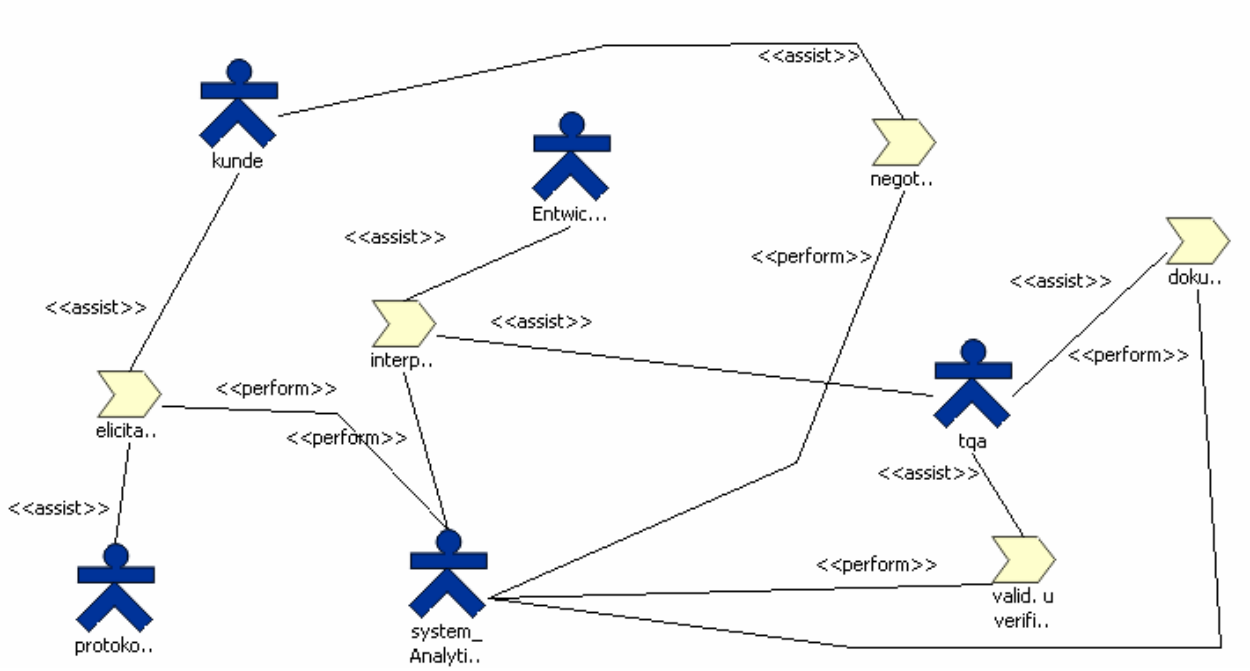


Abbildung 29: Use-Case-Diagramm Softwareprojekt

In SPEM hat man die Möglichkeit den Arbeitsbeschreibungen zwei verschiedene Verbindungstypen zu zuordnen. Eine Prozessrolle kann eine Aktivität ausführen (perform) oder an ihr teilnehmen (assist) (siehe Abbildung 30).

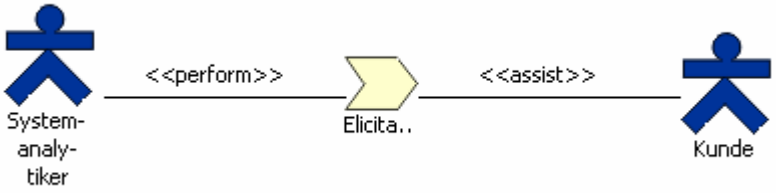


Abbildung 30: Perform- und Assistbeziehungen

Die Abbildung 30 übersetzt in Prolog erzeugt die folgenden zweistelligen Prädikate:

perform (system_analytiker,elicitation)

assist (kunde, elicitation)

Diese Prädikate besagen, dass der Systemanalytiker die Aktivität ausführt und der Kunde in ihr mitwirkt.

In Aktivitätsdiagrammen hat man auch die Möglichkeit mehrere Use-Cases miteinander in Beziehung zu bringen. Die Abbildung 31 beschreibt die Arbeitsbeschreibungen aus den Phasen Anforderungsanalyse und Entwurf. Die Include-Abhängigkeit ist ein ausgeklammerter Teil-Use-Case und Extends-Abhängigkeiten sind optionale Erweiterung des Use Cases.

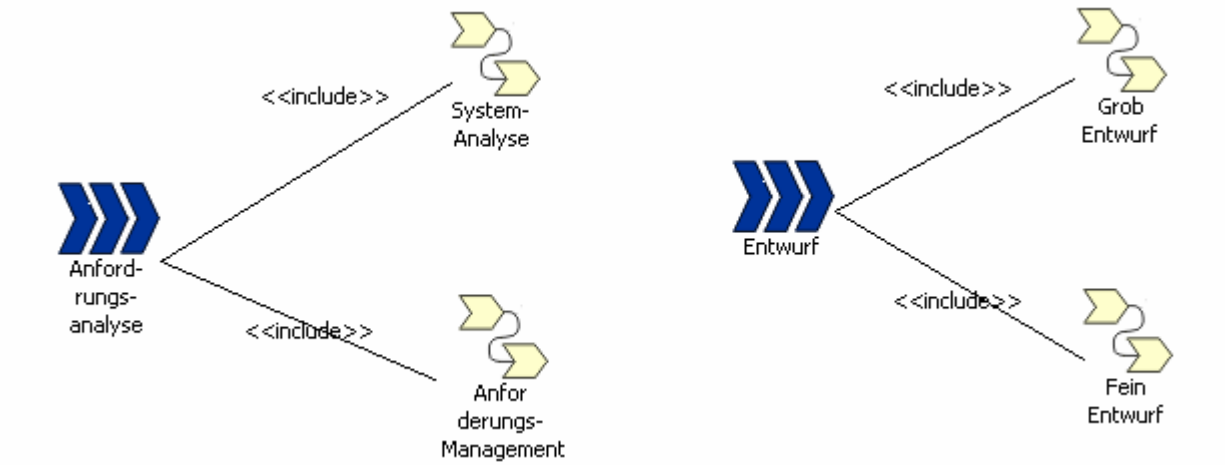


Abbildung 31: Use-Case-Beispiel Include-Beziehungen

Diese Abbildung wird in Prolog zu:

include (anforderungsanalyse,anforderungsmanagement).
include (anforderungsanalyse,systemanalyse).
include (entwurf,grob_entwurf).
include (entwurf,fein_entwurf).

Analog wird die Extends-Abhängigkeit zu dem Prädikat extends/2

3.2.3. SPEM Klassendiagramm

Klassendiagramme in SPEM beschreiben die Beziehungen der einzelnen Elemente zueinander. In der nächsten Abbildung werden die Prozesserzeugnisse in einem Klassendiagramm dargestellt. Der Kunde erstellt das Lastenheft, das in die funktionalen Anforderungen gehört. Ein Use-Case-Element ist ein UML Diagramm und wird durch die UML und durch Templates unterstützt (Abbildung 32). Mittels dieser Abbildung soll die Umwandlung des Klassendiagramms erklärt werden.

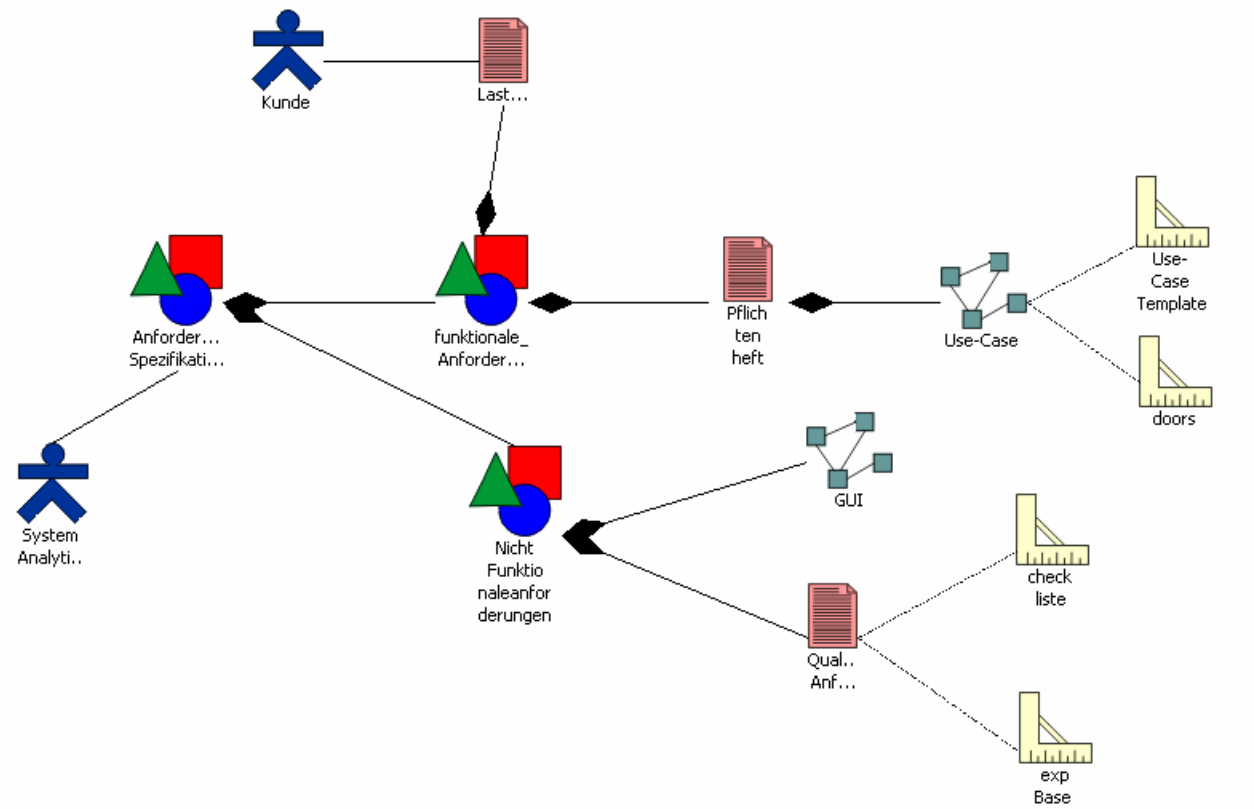


Abbildung 32: Klassendiagramm Softwareprojekt

In einem SPEM-Klassendiagramm können Rollen mit Prozessartefakten interagieren oder auch Prozessartefakte mit anderen Elementen in Beziehung stehen. Die Beziehung zwischen Rollen und Prozessartefakten wird in Prolog durch ein zweistelliges Prädikat beschrieben. In Abbildung 33 wird eine einfache Verbindung verwendet um den Arbeitsprodukten oder Dokumenten die Rolle zu zuteilen, die es erstellt.

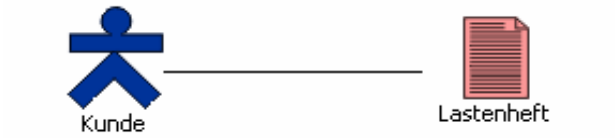


Abbildung 33: einfache Beziehung

Diese Abbildung übersetzt in Prolog wird zu:

responsibilty_role (kunde, lastenheft)

Dieses Prädikat besagt, dass der Kunde die verantwortliche Rolle für das Lastenheft ist.

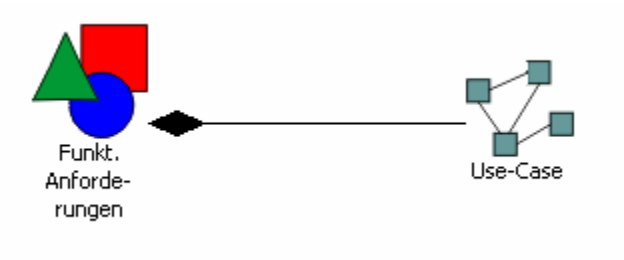


Abbildung 34: Komposition

Wenn ein Prozessartefakt aus einem anderen Prozessartefakt besteht (siehe Abbildung 34), wird dies in Prolog durch das zweistellige Prädikat consists_of dargestellt.

Die Abbildung 34 übersetzt in Prolog ergibt:

consists_of (funktionale_Anforderungen,use-case)

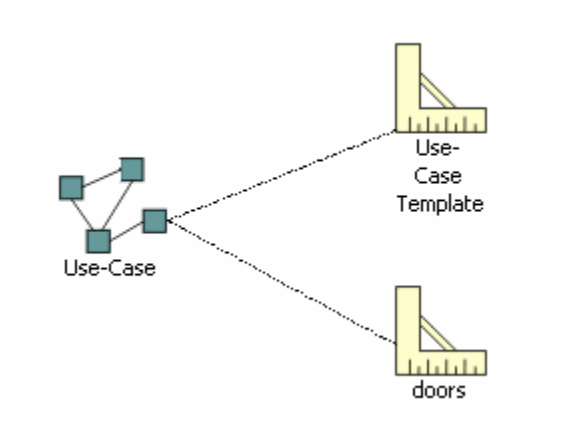


Abbildung 35: Guidance

In dem Beispiel Abbildung 35 wird ein Prozessratefakt von zwei Prozessartefakten unterstützt. In Prolog entstehen daraus zwei zweistellige Prädikate:

guidance_of (use-case, use_case_template).
guidance_of (use-case,doors).

Damit wird beschrieben, dass das UML-Profile Use-Case von der Guidance Use-Case Template und doors unterstützt wird.

3.3. Tabellen der Umwandlungsschritte

SPEM Sterotypen	Transformiert in Prolog
	workdefinition (Workdefinition_Name)
	phase (Phase_Name)
	activity (Activity_Name)
Iteration	iteration (Iteration_Name)
Step	step (Step_Name)
Lifecycle	lifecycle (Lifecycle_Name)

Tabelle 1 Arbeitsbeschreibungen

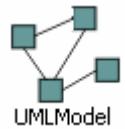
	workprodukt(Workprodukt_Name)
	document(Document_Name)
	uml_model (uml_model_Name)
	guidance (guidance_Name)
	process (Process_Name)

Tabelle 2 Prozessartefakte

	role (Name)
	performer(Name)

Tabelle 3 Prozessrollen

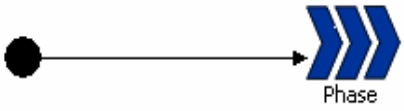
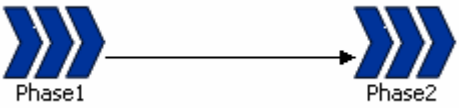
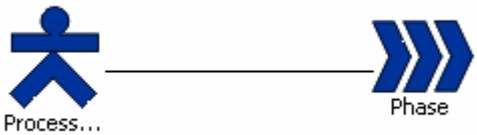
Verbindungen im Aktivitätsdiagarmm	**Übersetzt in Prolog**
	startphase(Phase).
	endphase(Phase).
	follow_phase (Phase1,Phase2)
	phase_input (Phase,Document)
	phase_output (Phase,WorkProdukt).
	role_phase (Role,Phase)

Tabelle 4 Aktivitätsdiagramm

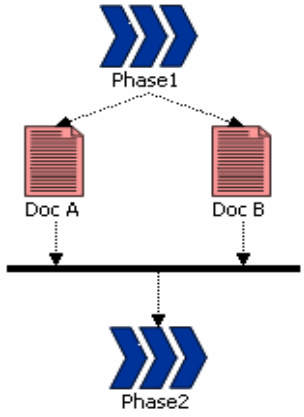
	follow_phase(Phase1,Phase2)
	follow_phase (Phase1,Phase2). phase_input (Phase2,Doc_A) phase_input (Phase2,Doc_B)

Tabelle 5 Followphase

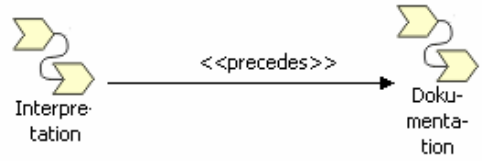
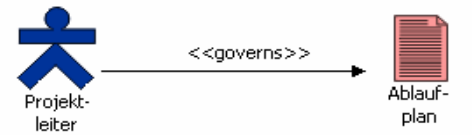
Abhängigkeiten in SPEM	Übersetzt in Prolog
	categorizes (anforderungen,systemanalyse).
	impacts (lastenheft,pflichtenheft).
	precedes (interpretation,dokumentation).
	governs (projektleiter,ablaufplan).

Tabelle 6 Abhängigkeiten

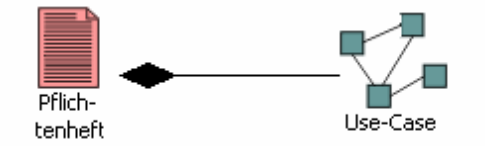
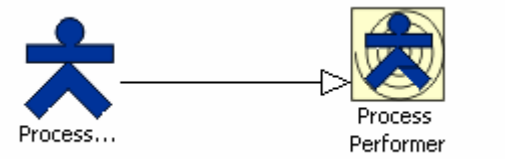
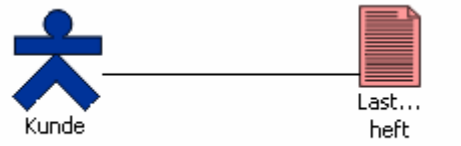
Verbindungen im Klassendiagramm	Übersetzt in Prolog
	consists_of (pflichtenheft,use_case).
	guidance (qualitätsanforderungen,checklist).
	inheritance (prozessrole, prozessperformer).
	responsibility_role (kunde,lastenheft).

Tabelle 7 Klassendiagramm

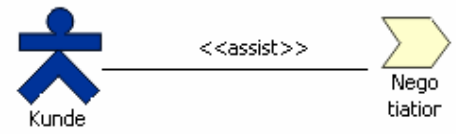
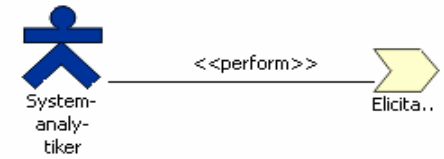
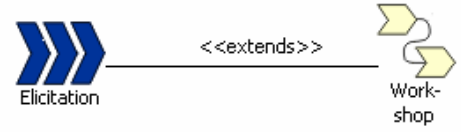
Transitionen im Use-Case-Diagramm	Übersetzt in Prolog
	assist (kunde,negotiation).
	perform (system_analytiker,elicitation).
	include (anfoderungsnalyse,systemanalyse).
	extends (elicitation,workshop).

Tabelle 8: Use-Case-Diagramm

3.4. Umwandlung der Annotation

In dem SPEM Editor von Andelco Jovancevic [5] kann man zu jedem Element eine oder mehrere Annotationen einfügen. In Abbildung 36 werden zwei Phasen des Softwareprojekts in einem Aktivitätsdiagramm dargestellt. Dieses Beispiel soll verdeutlichen, dass man in SPEM Prozessartefakte und Arbeitsbeschreibungen bewerten kann. Wenn man ein Merkmal nicht bewertet, bleibt dieser unbeschriftet wie z.B. bei der Entwurfspezifikation aus Abbildung 36.
Da nicht die Bewertung, sondern die Transformation der Bewertung ein Ziel dieser Arbeit ist, werde ich nicht darauf eingehen wie man eine Bewertung abgeben kann.
Vielmehr werde ich erklären wie eine Bewertung in SPEM umgewandelt in Prolog aussieht. Die Abbildung soll dabei helfen die Idee der Umwandlung leichter zu verstehen. Der oben genannte Editor enthält neben der Bewertung auch andere Annotationsdaten, die auf der graphischen Oberfläche nur dann zusehen sind, wenn man auf das jeweilige Element klickt und die Annoatationsdaten auswählt. Da sie Informationen über die Elemente und damit über das Projekt liefern und für die Prozessoptimierung verwendet werden können (mehr dazu im nächsten Kapitel) müssen diese ebenfalls mit übersetzt werden. Die Bedeutung der Annotationsdaten, insbesondere die Bewertungsmerkmale werden in Kapitel 4 „ Einordnung in den Prozess-Verbesserungs-Kreislauf erläutert. An dieser Stelle wird erstmal die Technische Umwandlung beschrieben.

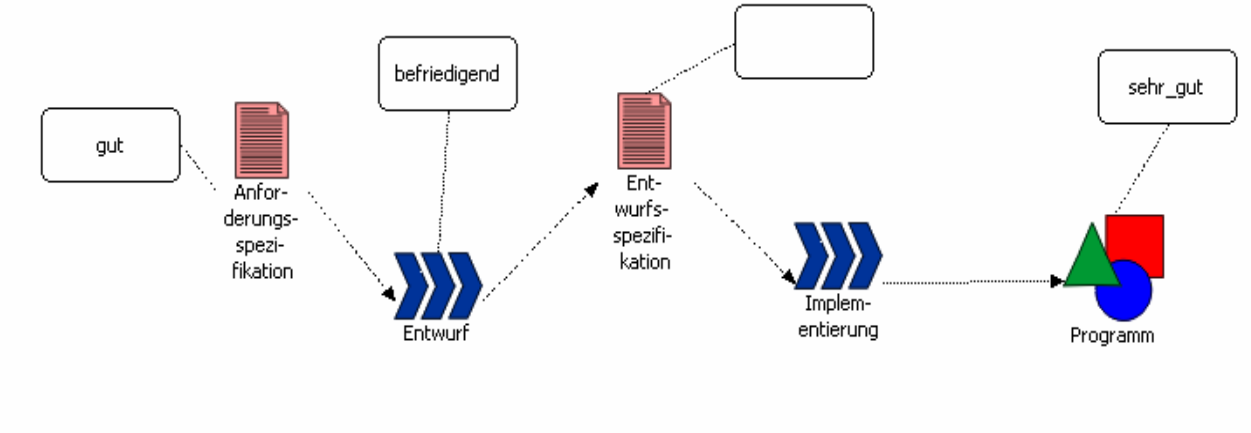


Abbildung 36: Beispiel für eine Annotation

Diese Abbildung stellt das Softwareprojekt mit den beiden Phasen Entwurf und Implementierung dar. Entwickler und Projektleiter haben die Möglichkeit ihre Prozesse zu bewerten und mit verschiedenen Merkmalen zu kennzeichnen. Dabei wählen Sie zwischen verschiedenen vordefinierten Merkmalen, die zu ihrer Bewertung passenden aus. In Abbildung 37 wird ein Prozessartefakt aus dem obigen Modell genau bewertet und in Prolog transformiert. Zuerst ist wichtig wie die ausgewählten Merkmale in Prolog aussehen sollen und wie man die nicht ausgewählten handhabt.

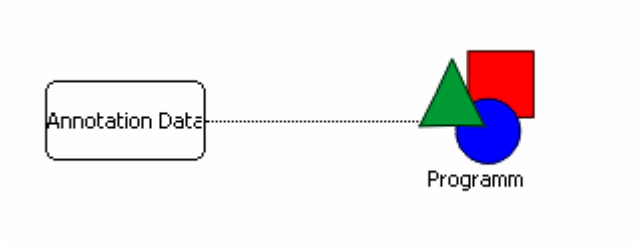
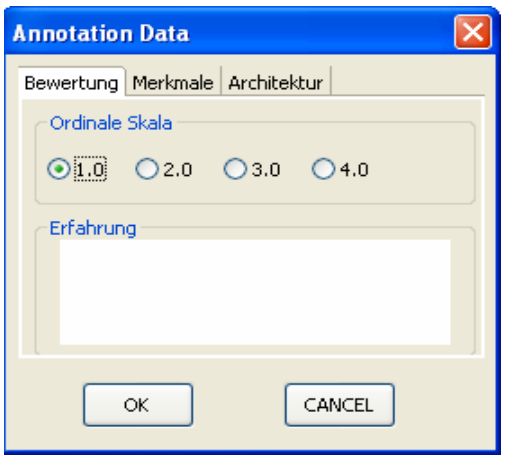


Abbildung 37: Annotation

Die Abbildung 38 und Folgende beschreiben die Bewertung des WorkProducts „Programm“.



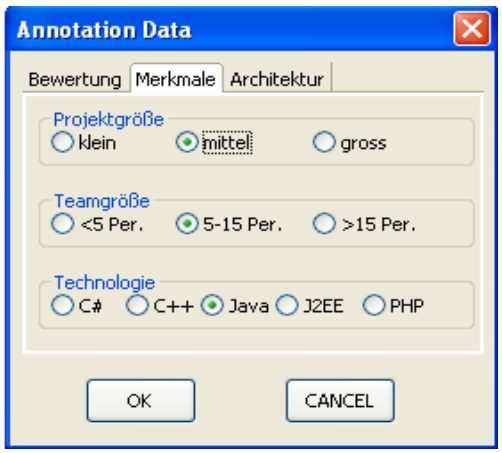
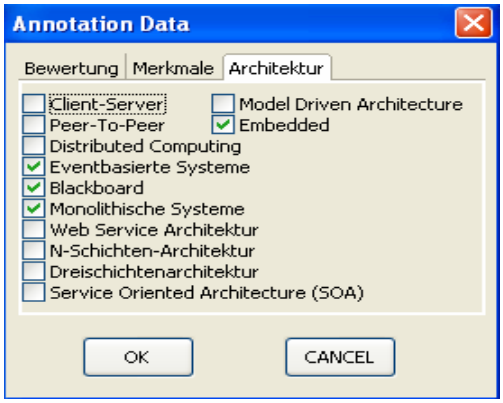


Abbildung 38 und 39: Annotationsdaten Bewertung und Merkmale



Nur bei der Architektur hat man die Möglichkeit mehrere Architekturen gleichzeitig auszuwählen. Diese müssen dem entsprechend alle in Prolog überführt werden. Wurde in dieser Arbeit dadurch gelöst, dass alle ausgewählten Architekturen durch Unterstriche voneinander getrennt wurden.

Abbildung 40: Annotationsdaten Architektur

Wie Man auch aus den Abbildungen 38 bis 40 entnehmen kann hat man sechs verschiedene Annotationskriterien.

- Bewertung: Ordinale Skala zwischen 1 und 4
- Erfahrung: individuelle Erfahrung

- Projektgroesse: Anzahl der Projektteilnehmer
- Teamgroesse: Anzahl der Personen in einer Gruppe
- Technologie: Programmierumgebung
- Architekturen: die verendeten Architekturen

Wobei man im Attribut Erfahrung seine Erfahrungen selbst beschreiben kann. Die anderen Annotationssdaten muss man nur auswählen. Da die Erfahrungen in langen Sätzen beschrieben werden kann, kann dies dazu führen, dass die Prädikate schnell unübersichtlich werden. Aus diesem Grund ist jedem selbst überlassen ob er eine Erfahrung mit übersetzt oder diese Stelle nicht angibt. Die Architekturattribute dagegen werden durch unterstriche zusammengeführt. Wird eine Rubrik nicht ausgewählt oder beschrieben, so wird dies in dem Prädikat an der entsprechenden Stelle mit „nicht angegeben“ verdeutlicht.
Transformiert man eine Annotation in Prolog so entstehen für jede Bewertung 6 dreistellige Prädikate.

- Das Prädikat wird nach dem Typ des Elements genannt
- In der ersten Stelle steht der Name des Elements
- Die zweite Stelle gibt das Bewertungsmerkmal an
- und in der dritten Stelle befinden sich die ausgewählten Annotationdaten

Das Beispiel aus den Abbildungen 38 bis 40 übersetzt in Prolog sieht wie folgt aus:

workprodukt (programm, teamgroesse, personen_5-15).

workprodukt (programm, erfahrung, nicht_angeben).

workprodukt (programm, projektgroesse, mittel).

workprodukt (programm, architektur,
eventbasierte_systeme_blackboard_monolithische_systeme_ embedded).

workprodukt (programm, bewertung,sehr-gut).

workprodukt (programm, technologie, java).

Analog würde eine Phase übersetzt so aussehen
phase(entwurf, teamgroesse, personen_5-15). usw.

4. Einordnung in den Verbesserungskreislauf der Prozesse

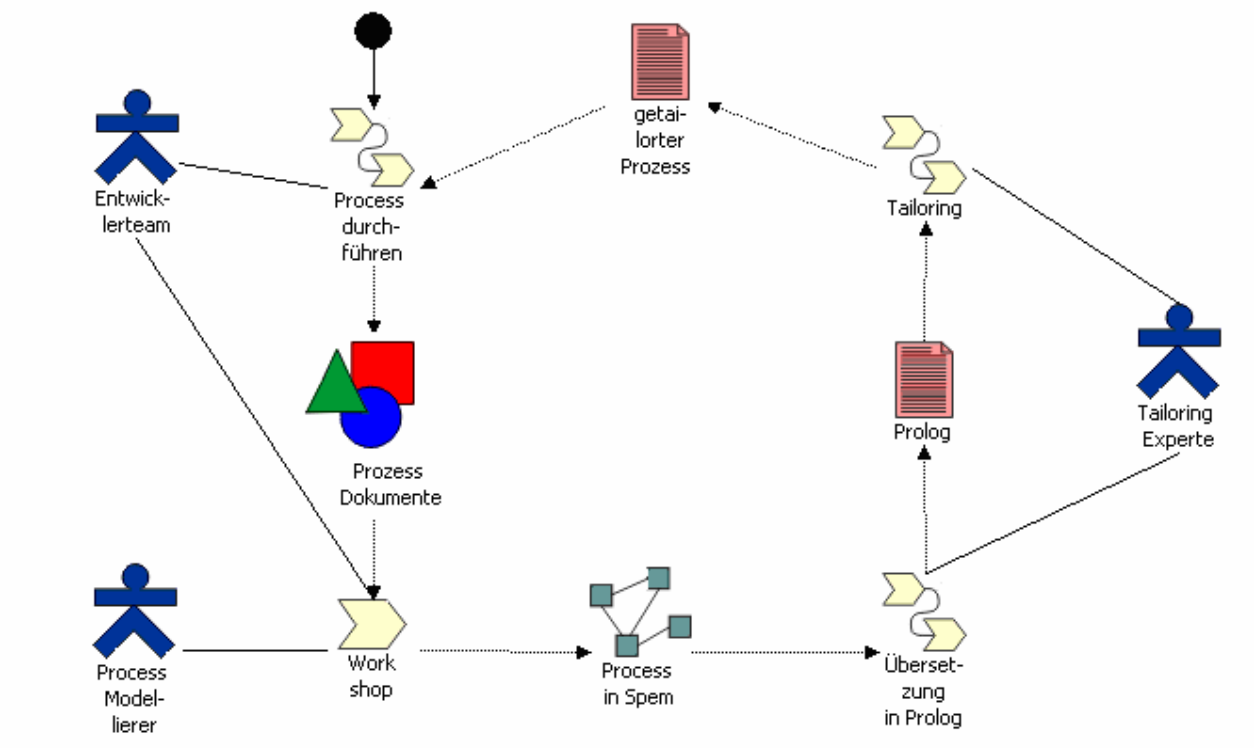


Abbildung 41: Verbesserungskreislauf der Prozesse

Die Abbildung 41 zeigt den Prozessverbesserungskreislauf mit den entsprechenden Arbeitsbeschreibungen. Der Kreislauf beginnt mit der Durchführung des Prozesses. Nach Abschluss des Prozesses treffen sich die beteiligten Personen mit dem Prozess-Modellierer in einem Workshop zusammen und analysieren den Prozess. Der Prozess wird von dem Prozess-Modellierer in SPEM zusammen mit den Beteiligten aufgezeichnet und von den Beteiligten bewertet. Da alle Beteiligten unterschiedliche Sichten auf die Software haben, existieren mehrere verschiedene Bewertungen für ein Prozesselement dass man ein Prozess aus verschiedenen Sichten optimieren kann. Der Manager z.B. vertritt das Interesse die Fertigstellung einer Software zu beschleunigen und die Kosten sowie den Zeitaufwand für Wartung und Weiterentwicklung dauerhaft zu senken.

Nach dem man den Prozess modelliert und bewertet hat beginnt die eigentliche Aufgabe der Prozessverbesserung, die Übersetzung in Prolog und das Tailorn. Der Tailoring-Experte versucht den Prozess durch Tailoring-Regeln in Prolog zu optimieren. Nach dem der Prozess in Prolog getailort wurde, wird der Kreislauf geschlossen und eventuell erneut fortgesetzt bis man den optimalen Prozess bestimmt hat.

Die Modellierung von Softwareprozessen unterstützt deren Qualität erheblich. Systematisches Vorgehen bei der Softwareentwicklung senkt das Risiko des Scheiterns und erleichtert das Finden von Fehlern. Ziel jeder Softwarenetwicklung ist es den Entwicklungsprozess zu optimieren. Um Softwareprozesse zu optimieren, muss man sie modellieren und bewerten.

4.1 Modellierung

Es ist mittlerweile kein Geheimnis mehr, dass die Modellierung von Software-Prozessen deren Qualität steigert. Die Modellierung einer Software soll die Effizienz des Entwicklungsprozesses erhöhen, indem mit der Implementierung erst dann begonnen wird, wenn die Planung zur Implementierung abgeschlossen ist.

Es existieren mehrere Prozessmodellierungssprachen wobei keines dieser, eine allgemeine Notation verwendet. Ziel war es eine Prozessmodellierungssprache zu entwickeln die zu einem Industriestandard wird, so dass sie alle bekannten Modellierungssprachen zu einer vereint.

Genau an dieses Prinzip setzt das Software Process Engineering Metamodell (kurz SPEM) an und ist wie bereits im zweiten Kapitel beschrieben nur eine kleine Erweiterung der UML. Es erbt die meisten Konzepte der UML und wendet es an die Domäne der Softwaremodellierung. SPEM ist eine sehr neue Modellierunksprache und soll dazu dienen, dass eine große Gemeinde von Softwareentwicklern sich damit besser verständigen kann. In wieweit SPEM dieser Rolle gerecht wird oder bisher wurde ist nicht Ziel dieser Arbeit.

Es soll hier nur als ein Werkzeug zur Modellierung von Softwareprozessen dienen und eine Aufgabe in dem Prozessverbesserungskreislauf erfüllen. Der Vorteil an SPEM bzw. an dem SPEM Editor der in dieser Arbeit verwendet wurde ist die Annotation. Diese soll in einem Workshop dazu verwendet werden

1. den Prozess grafisch darzustellen
2. Ihn mit Hilfe der Annotationsdaten zu bewerten

Hierbei stellt sich die Frage wer nimmt an so einem Workshop teil? Wer modelliert und wer übernimmt die Bewertung? Bei einem großen Projekt über 100 Personen ist es unmöglich, dass an einem Workshop so viele Personen teilnehmen können. So begrenzt man sich nur auf die wichtigsten Personen darunter überwiegend Projektleiter, Manager und einige wenige Entwickler, die dabei die Sicht der Entwickler repräsentieren. Wobei es sich für die Qualität der Bewertung ohne Frage eher empfiehlt, alle Gruppen zu vertreten.

Der Prozessmodellierer, der nicht zum Leiter- und Entwicklerteam gehört, hat die Aufgabe den Prozess in diesem Fall mit dem SPEM-Editor zu modellieren. Als Modellierungsexperte hilft er auch den Entwicklern, den Sie betreffende Teile des Prozesses grafisch darzustellen und anschließend zu bewerten. Der SPEM-Editor ermöglicht dem Modellierer jedes Prozessartefakt und jede Tätigkeit einzeln zu bewerten und diese mit zusätzlichen Informationen zu versehen. Dadurch kann man die Elemente untereinander vergleichen. Wichtig dabei ist es vorher zu definieren was die möglichen Bewertungsmerkmale bedeuten. Bewertet man z.B. ein Template für eine Anforderungsspezifikation mit befriedigend, so muss vorher klar sein welche Auswirkung dies auf den Prozess haben soll und was es bedeuten würde wenn man es mit gut bewertet. Eine mögliche Variante wäre: ein Template, das mit sehr gut bewertet wird, kann für den Prozess bedeuten, dass sehr viel Wert darauf gelegt wird. Ein mit ausreichend Bewertetes dagegen stellt keine wichtige Rolle für den Prozess dar und wird im nächsten Projekt nicht verwendet.

4.2 Prozesse analysieren und optimieren

Der nächste Schritt in dem Prozess-Verbesserungs-Kreislauf aus Abbildung 41 ist es, den in SPEM modellierten Prozess zu optimieren. Um eine Optimierung durchzuführen muss der Prozess mit Hilfe der Bewertung in einem Workshop analysiert werden. Dazu werden die erfassten Haupt- und Teilprozesse sowie Aktivitäten einzeln auf ihre Notwendigkeit und ihren Beitrag zur Wertschöpfung untersucht. Nicht wertschöpfende Aktivitäten werden gekennzeichnet. Auf diese Weise werden sämtliche Aktivitäten entlang der Prozesskette genauestens analysiert. Die Ergebnisse werden in einem Abschlussbericht zusammengefasst, in dem auch sämtliche Optimierungspotenziale enthalten sind.

Die Optimierung der Prozesse führt ein Tailoring-Experte außerhalb dieses Workshops durch. Die Rolle „Tailoring-Experte“ beschreibt hier den Experten (oder mehrere), dem Techniken und Werkzeuge bekannt sind mit denen er den Prozess in erster Linie verändern und somit verbessern kann. Der Tailoring-Experte führt keine Bewertung der Prozesse durch und ist nicht zwingend beteiligt an dem Workshop, in der diese modelliert und analisiert werden. Mit Hilfe des Abschlussberichts soll er im nächsten Schritt die Prozessoptimierung durchführen.

4.2.1. Optimierung

Die Optimierung von Prozessen soll in dieser Arbeit mit Hilfe von Prolog ermöglicht werden. An dieser Stelle beginnt die Aufgabe, des in dieser Arbeit entwickelten Programms. Der in SPEM dargestellte Prozess wird mit Hilfe dieses Programms in Prolog überführt. Auch Prolog ist nur ein Werkzeug wie viele Andere, mit denen Prozesse verbessert werden können.

Nachdem der Tailoring-Experte den Prozess in Prolog überführt hat, beginnt die Phase der Prozessverbesserung. Eine Möglichkeit der Prozessverbesserung ist das Tailorn von Prozessen.

4.2.2. Tailoring

Der englische Begriff „Tailoring“ bzw. „tailor“ heißt übersetzt zurechtschneiden. In Bezug auf die Prozessverbesserung bedeutet dies Anpassung des Softwareentwicklungsprozesses. Das Ziel des Tailoring ist also den gegebenen Prozess an die Bedürfnisse und Einschränkungen eines konkreten Projektes anzupassen [7].

Das „Tailorn“ von Softwareprozessen wird hier an Hand von so genannten „Tailoring-Regeln“ ermöglicht.

Die Tailoring-Regeln kommen z.B. aus [7]:

- Vorgaben der Organisation

- aus Erfahrungen aus anderen Projekten

- aus Gesetzen oder Normen (z.B.: manche Aktivitäten dürfen nicht verändert werden).

Man kann Softwareprozesse auf unterschiedlichen Oberflächen und Programmier-Sprachen tailorn. Hier soll das Tailoring als ein Teil des Prozess-Verbesserungs-Kreisslaufs in Prolog durchgeführt werden. In diesem Zusammenhang stellt sich die durchaus berechtigte Frage: „Warum Prolog?“

Prolog eignet sich gut Anfragen an die Wissensbasis zu stellen. Durch mehrfache Anwendung von Anfragen kann man den optimalen Prozess bestimmen. Das einzige Problem, dass hierbei entsteht, ist die mögliche Fülle und dadurch die Unübersichtlichkeit der Prädikate. Ich habe dies nach Absprache mit meinem Betreuer versucht zu reduzieren, indem ich Informationen aus SPEM, die für die Anfrage in Prolog nicht notwendig waren, herausgefiltert habe.

Eine andere Möglichkeit Softwareprozesse durch Anfragen zu tailorn, wäre in SQL gegeben. Leider ist die Umwandlung von SPEM in eine Datenbanksprache ein wenig aufwändiger als die Umwandlung in Prolog. Die Umwandlung von SPEM in Prolog wird in dieser Arbeit durch XSLT Styleheeets realisiert. Die Umwandlung von SPEM in eine Datenbanksprache verlangt neben der Programmierung von XSLT-Stylesheets noch die Generierung der Datenbank.

4.2.2.1. Wie tailort man Prozese in Prolog

Nun möchte ich den übersetzten Prozess tailorn und brauche dazu Tailoring-Regeln in Prolog. Bei der Erstellung der Tailorin-Regeln sind zwei Punkte von entscheidender Bedeutung.

1. Fragestellung
2. Zielsetzung

Fragestellung:

- Welche Aktivitäten sind für die Durchführung eines Projekts erforderlich?

- Welche Produkte müssen im Rahmen der Projektabwicklung erzeugt werden?

Zielsetzung

- Vermeidung unnötiger Aktivitäten

- Vermeidung sinnloser Dokumente

- Vermeidung des Fehlens wichtiger Dokumente und Aktivitäten

Aus den obigen Punkten entstehen die ersten Tailoring-Regeln. Sie erzeugen oder entfernen Aktivitäten, die für die Optimierung der Prozesse überflüssig oder dringend notwendig sind. Sie fügen Aktivitäten zwischen bestimmten Aktivitäten oder verändern diese. Dokumente oder andere Aktivitätseingaben werden entfernt oder neue werden hinzugefügt. Eine Rolle wird entfernt, neu definiert oder einer Arbeitbeschreibung zugeordnet. Des Weiteren werden Regeln definiert, die die Prozesse nach bestimmten Merkmalen tailorn. Betrachtet man dabei die Merkmale aus dem SPEM-Editor, hat man die Möglichkeit Prozesse nach den Annoattionsdaten zu tailorn. Dies erreicht man indem man Anfragen nach bestimmten Merkmalen stellt oder Prozessartefakte nach diesen Merkmalen entfernt oder hinzufügt.

Eine mögliche Anfrage könnte so aussehen:

- alle Prozessartefakte aufzulisten, die einer bestimmten Bewertung entsprechen, z.B. alle WorkProdukts mit Bewertungsattribut sehr gut: workproduct (X, bewertung, sehr-gut).

Diese Idee des Tailoring wurde in der Bachelorarbeit von Daniel Eggert [3] realisiert. Es wurde eine Tailoring-Wissensbasis erstellt, in denen Regeln für die Manipulation von Fakten und Prädikaten definiert sind. Damit die Prozessmodelle aus dem SPEM-Editor in der Applikation getailort werden kann, habe ich all die Prädikate an die Bezeichnung dieser Arbeit angepasst. Man kann einfach die aus SPEM übersetzten Prädikate in die entsprechende Wissensbasis einfügen und den „wizard“ ausführen (siehe Abbildung 42 und Folgende)

Abbildung 42: Prolog Wizard

Der Wizard stellt den Prozess in Baumstruktur dar und ist somit übersichtlicher als die reinen Anfragen. Nachdem man die Wissensbasen eingefügt hat, wählt man die Merkmale nach denen getailort werden soll und die GUI Applikation erstellt den Prozessbaum (Abbildung 43).

Abbildung 43: Prolog Wizard

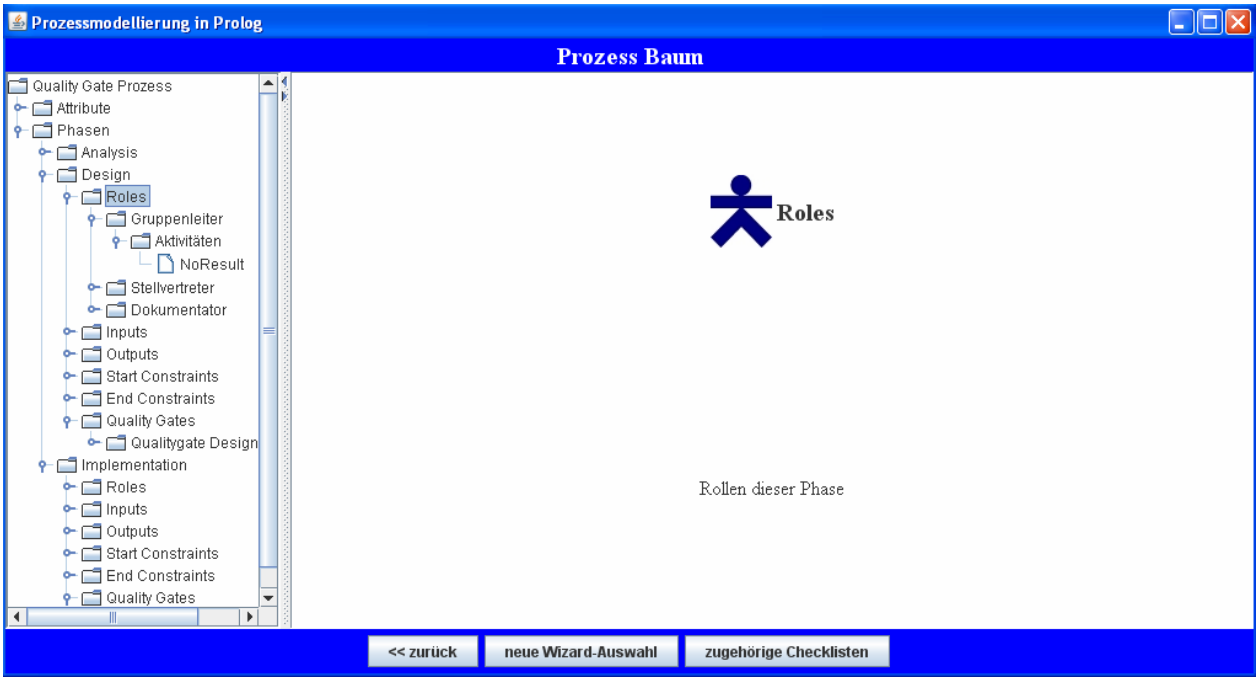


Abbildung 44: Prolog Wizard

Leider beschränkt sich diese Applikation nur auf Aktivitätsdiagramme und unterstützt nicht dieselben Bewertungen wie die aus dem SPEM-Editor. Damit man auch die Use-Case Diagramme und auch Klassendiagramme tailorn kann, müssen neue Tailoring-Regeln und neue Attribute für Bewertungen definiert werden. Hat man diese definiert, so kann man durch mehrfache Anfragen aus einem gegebenen Prozess einen optimierten Prozess herausholen. Dies soll im nächsten Kapitel in einem kleinen Kontext verdeutlicht werden. Da nicht das primäre Ziel dieser Arbeit das Tailoring ist und ein umfangreiches Beispiel den Rahmen dieser Arbeit

sprengen würde, wird hier nur ein Modellierter Workshop in Prolog optimiert. Dabei werden die Prologregeln die das Tailoring ermöglichen, als gegeben angenommen.

4.2.2.2. Tailoring-Beispiel

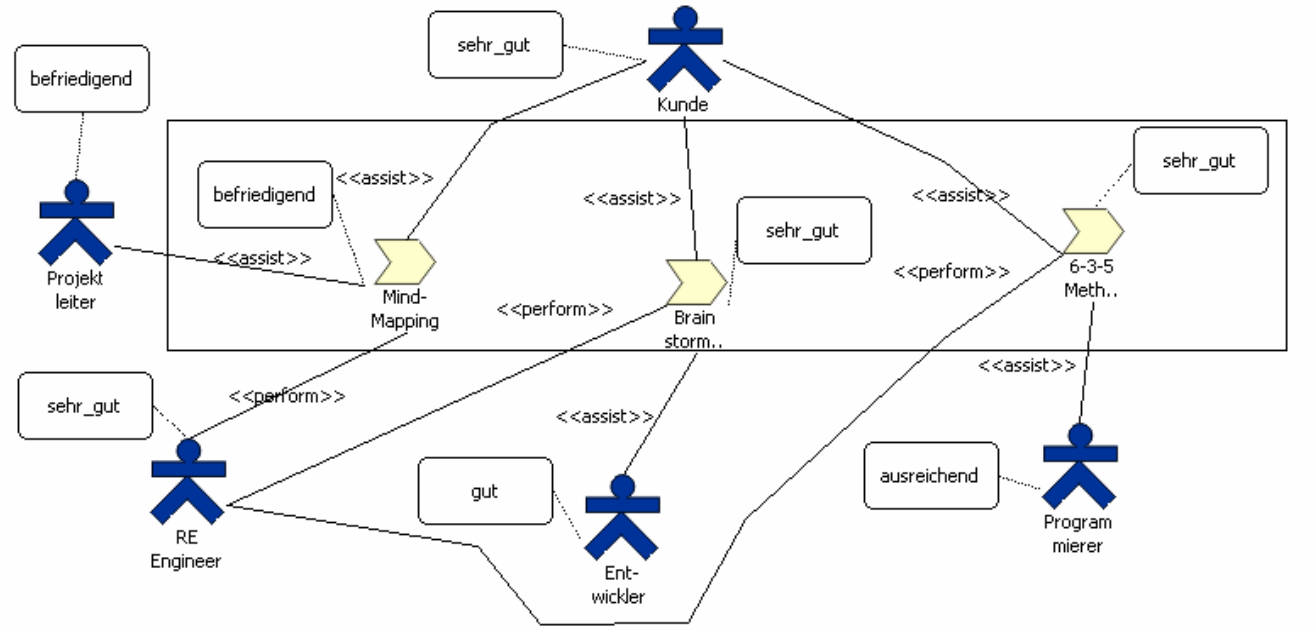


Abbildung 45: Workshop Anforderungsermittlung

Die Abbildung 45 stellt einen Workshop für die Anforderungsanalyse in einem Use-Case-Diagramm dar. Es ist nur eine von vielen Modellierungsmöglichkeiten und vertritt bewusst nicht die optimalen Rollen und Aktivitäten für solch einen Workshop. An diesem Modell soll das Tailoring angewendet werden, um den Leser eine mögliche Anwendung dieser Arbeit vorzustellen. Mit Hilfe der Bewertung soll dieses Beispiel optimiert werden. Die Aktivitäten in dem Modell stellen mögliche Methoden dar um in einem Workshop Anforderungen zu ermitteln. Jeder der die Vorlesung „Anforderungen und Entwurf“[8] gehört hat weiß was mit diesen Methoden gemeint ist. Die Qualität der Methoden hängt von den Personen die daran beteiligt sind und von dem Problem, für das es angewendet werden soll. Ein Projekt z.B. das Softwareprojekt wurde in einem Workshop wie im vorherigen Kapitel verdeutlicht von den Beteiligten mit Hilfe des Prozessmodellierers in SPEM modelliert. Eines dieser Modelle könnte wie das Use-Case-Diagramm aus der Abbildung 45 aussehen. Alle Prozesselemente wurden von den Beteiligten in diesem Workshop bewertet. Die anderen Annotationsdaten wurden nicht ausgewählt und sind für dieses Beispiel außer Bedeutung.

Anschließend wird dieses bewertete Prozessmodell von einem Tailoring-Experten mit dem Programm aus dieser Arbeit in Prolog überführt. Die Abbildung 46 stellt den
Workshop in Prolog dar. Jedoch wurden diesem Modell die nicht ausgewählten Annotationsdaten aus der Wissensbasis entfernt.

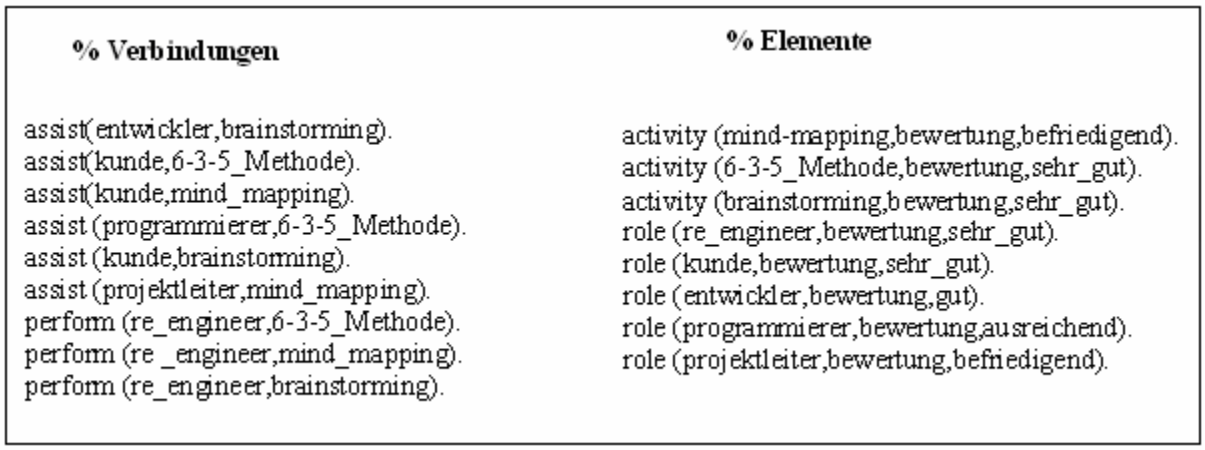


Abbildung 46: Prolog Wissensbasis

Der Tailoring-Experte soll nun alle Aktivitäten entfernen die nicht mit sehr gut bewertet wurden. Die Bewertung „sehr_gut“ bedeutet hierbei dass diese Methode für die Anforderungsermittlung in einem Workshop optimal geeignet ist. Die mit „gut“ und „befriedigend“ bewerteten Aktivitäten eignen sich zum Teil und nur bedingt für die Ermittlung von Anforderungen. Wobei die mit „ausreichend“ bewerteten Methoden gar nicht geeignet sind. Analog behandelt man die Prozessrollen. Denn nicht jede Rolle ist für jede Aktivität geeignet. Bei einem Workhop, der nur auf eine bestimmte Kapazität begrenzt ist, muss man gezielt auswählen, wer daran teilnehmen soll.

Da nur die wichtigsten und sinnvollsten Personen an diesem Workshop teilnehmen sollen, werden an dem obigen Beispiel auch alle Prozessrollen entfernt, die nicht mit „sehr_gut“ oder „gut“ bewertet wurden. Die Rolle Programmierer und Projektleiter ist für ein Workshop um Anforderungen zu ermitteln nicht notwendig. „sehr_gut bedeutet, diese Rolle ist zwingend notwendig für diesen Workshop oder für die Aktivität an der sie beteiligt ist. Die mit „gut“ bewerteten Rollen sind sinnvoll aber nicht zwingend Notwendig. Alle anderen Rollen stören oder sind nur zusätzliche Kosten.

Hierbei sollte man beachten dass jeder Prozess, die Bewertungsmerkmale unterschiedlich definieren kann. Mit „befriedigend“ bewertete Elemente könnten für den einen Prozess genügend und für den anderen Prozess eher unakzeptabel sein. Deshalb ist es für einen Tailoring-Experten auch wichtig, dass die Interpretation der Bewertungsmerkmale eindeutig ist.

Des Weiteren kann man nach Bedarf neue Rollen und zuständige Aktivitäten in die Wissensbasis einfügen. Oder auch Aktivitäten definieren und Diesen, bereits vorhandene Rollen zuteilen. Ein Beispiel dafür wäre hier, eine andere Methode, die sich ebenfalls für eine Anforderungsanalyse sehr gut eignet (z.B. gezielte Fragen an den Kunden).

Eine mögliche Optimierung für das Beispiel aus Abbildung 46 könnte wie folgt aussehen:

```
% Verbindungen

assist (entwickler,brainstorming).
assist (kunde,6-3-5_Methode).
assist (kunde,brainstorming).
perform (re_engineer,6-3-5_Methode).
perform (re_engineer,brainstorming).

% Elemente

activity (6-3-5_Methode,bewertung,sehr_gut).
activity (brainstorming,bewertung,sehr_gut).
role (re_engineer,bewertung,sehr_gut).
role (kunde,bewertung,sehr_gut).
role (entwickler,bewertung,gut).
```

Abbildung 47: Optimierter Workshop in Prolog

In der Abbildung 47 wird eine mögliche Optimierung des Workshops aus Abbildung 46 in Prolog dargestellt. Die Rollen und deren Beziehungen, die nicht mit „ sehr_gut oder "gut" bewertet wurden, sind eliminiert. Außerdem wurden die nicht mit „sehr gut" bewerteten Aktivitäten entfernt. An dieser Stelle ist es wichtig zu erwähnen, dass die Optimierung nach dem Annotationstyp „Bewertung" erfolgt ist. Man kann Prozesse auch nach anderen Merkmalen wie z.B. Architektur oder Projektgröße tailorn. Da eine Bewertung eine individuelle Entscheidung ist, kann es für jedes Modell mehrere Optimierungsansätze geben. Ziel jeder Projektleitung ist es für ihr Projekt den ergiebigsten herauszufiltern.

Ein anderes Beispiel fürs Tailoring (Prozessanpassung) wäre z.B. das Entfernen oder Hinzufügen von Arbeitsprodukten nach bestimmten Merkmalen, in einem Aktivitätsdiagramm. Oder definieren von neuen Abhängigkeiten in einem Klassendiagramm. Schließlich gibt es viele Möglichkeiten um Prozesse an zu passen. Das obige Beispiel sollte nur eine kleine Demonstration sein.

5. Zusammenfassung, Probleme und Ausblick

Ziel der Arbeit war es Prozessmodelle aus dem SPEM-Editor in Prolog zu überführen. Um dies zu realisieren musste man sich in SPEM, Prolog und XSLT einarbeiten. SPEM ist eine Erweiterung der UML und eignet sich gut für die graphische Darstellung von Softwareprozessen. Es soll insbesondere in der Industrieanwendung eine allgemeine Notation liefern. SPEM verwendet sechs Diagrammtypen aus der UML, wobei in dieser Arbeit nur die Use-Case-, Klassen- und die Aktivitätsdiagramme übersetzt wurden. Die Grund-Elemente für die Prozessmodellierung sind **Aktivitäten**, in denen **Arbeitsprodukte** erstellt werden und **Rollen**, die für die Aktivitäten verantwortlich sind. Einige Elemente sind Abstraktionen und Unterklassen dieser Elemente. Andere unterstützen diese und liefern zusätzliche Informationen.

Die Regeln zur Wohlgeformtheit definieren Einschränkungen zwischen den Beziehungen der SPEM-Elemente. Sie erlauben die Darstellung von Abhängigkeiten, die alle eine besondere Bedeutung haben.

Im Hauptteil der Arbeit wurde die Transformation an einem echten Projekt (das Softwareprojekt)

erklärt. Jedes SPEM-Element wurde in Prolog zu einem einstelligen Prädikat übersetzt. Mit dem Typ des Elements als Prädikatsnamen und der Bezeichnung des Elements als Prädikatswert.
(z.B. workprodukt(programm) oder phase (anforderungsanalyse) usw.
Alle Verbindungen wurden in zweistellige Prädikate transformiert und je nach Verbindungsart wurde diesem Prädikat ein Name zugeordnet. So wurden die Verbindung (gestrichelte Kante) z.B. zwischen einem Arbeitsprodukt und einer Phase in das Prädikat „phase_input (Phase, Arbeitsprodukt) und die Verbindung (durchgezogene Kante) zwischen zwei Phasen in „follow_phase (Phase1,Phase2).“ überetzt.

Die Verwendung der Arbeit wurde in Kapitel 4 erläutert. Mit diesem Programm können in SPEM modellierte Prozesse in Prolog überführt werden. Es bietet eine Schnittstelle zwischen dem Prozessmodellierer und dem Tailoring-Experten, der den Prozess im nächsten Schritt optimiert. Prolog dient hierbei als ein mögliches Werkzeug, um Prozessmodelle leicht zu verändern. Durch Tailoring-Regeln kann man z.B. Arbeitsbeschreibungen und Arbeitsprodukte, die keine Notwendigkeit oder Bedeutung für den Prozess heben, entfernen. Oder neue Rollen und Arbeitsbeschreibungen einfügen. Dabei spielt die Annotation eine wichtige Funktion. Sie ermöglicht es, das Prozessmodell mit Hilfe der Annotationsdaten zu filtern.

Test und Probleme

Das Programm wurde mit den Softwareprojekt-Beispielen (Abbildungen 19-32) und anderen kleinen Modellen getestet. Dabei bin ich so vorgegangen dass ich für jedes Element aus dem SPEM-Editor ein kleines Modell erstellt hab und es mit den entsprechenden Styleheets übersetzt hab. Die Schwierigkeit dabei lag eher daran, dass SPEM auch sehr komplizierte Modelle erlaubt, aber man kaum praktische Beispiele dafür findet. Die Übersetzung der drei Diagrammtypen, mit denen das Softwareprojekt in dieser Arbeit modelliert wurde befindet sich im Anhang.

Die Probleme der technischen Umsetzung dagegen lagen weniger an SPEM sondern vielmehr an dem ProFlow, das die Grundstruktur von dem SPEM-Editor bildet. Im Folgenden werde ich kurz darauf eingehen was die Probleme bei der Umwandlung waren und welche technischen Grenzen dabei auftraten.

Die Umwandlung von SPEM in Prolog wurde mittels XSLT-Stylesheets ermöglicht. Überraschender Weise gab es hier keine Probleme denn mit XSLT kann man wirklich fast alles realisieren. Die Probleme und Schwierigkeiten verursachten eher die technischen Grenzen von dem SPEM-Editor. Diese führten dazu, dass zwei entscheidende Merkmale der übersetzten Diagrammtypen nicht in Prolog überführt wurden.

1. Der SPEM-Editor bietet keine Darstellung von Swimlanes

2. Auch eine Decision ist aufgrund der ungeeigneten Entwicklung nicht korrekt übersetzbar

Die oben ernannten Probleme werden an Hand der Abbildung 48 erklärt. Hierbei geht es nicht um die inhaltliche Korrektheit des Modells, sondern vielmehr um die Notation. Das Beispiel ist

recht einfach und soll die Modellierung einer Entscheidung und dem Swimlane in SPEM verdeutlichen. Ein Swimlane teilt die Aktivitäten in Verantwortlichkeitsbereiche ein.

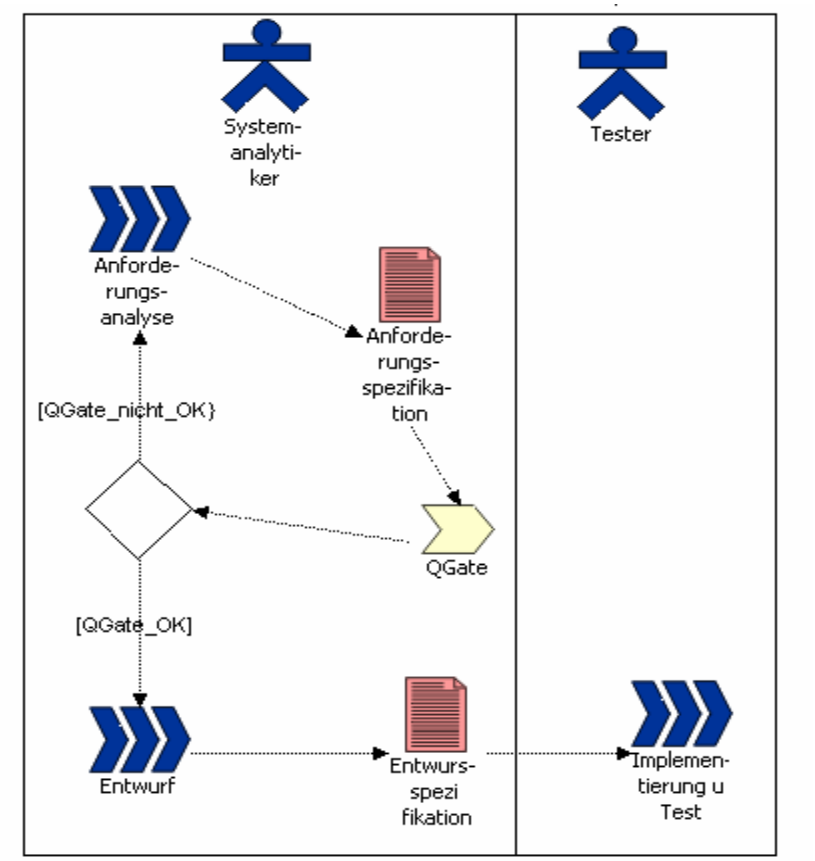


Abbildung 48: Decision und Swimlanes

In der Abbildung 48 wird eine Entscheidung getroffen und wenn diese Entscheidung gültig ist geht es in der nächsten Phase weiter. Ansonsten wird in die vorherige Phase zurückgekehrt.
Diese Art von Modellierung kann in dieser Arbeit nicht in Prolog übersetzt werden, weil in SPEM, die Entscheidung (Raute), die Bedingung und die Verbindung, einzelne Modelelemente sind und zwischen der Transition (Verbindung) und der Bedingung keine Beziehung besteht. Das bedeutet, dass die Aktivität und die Phase über eine Decision verbunden sind aber die Bedingung nicht direkt an den Verbindungen steht. So ist es nicht möglich eine Bedingung zu der Verbindung zu zuordnen, für die es verwendet wurde.

Auch die Swimlanes, die jeder Rolle seinen Verantwortlichkeitsbereich zuordnet, ist im SPEM-Editor entgegen der Abbildung nicht möglich und wurde deshalb ebenfalls nicht transformiert.

Ausblick

Die in dieser Arbeit entwickelten Stylesheets ermöglichen die vollständige Transformation der drei Diagrammtypen, die bereits bekannt sind. Die Prozessmodelle werden soweit sie für Prolog

notwendig sind komplett übersetzt. Anschließend kann man sie in Prolog weiter verwenden. Es gibt bestimmt einige Möglichkeiten, mit denen Prozessmodelle in Prolog bearbeitet werden können. Eines und die ursprüngliche Absicht dieser Arbeit war es die Prozessmodelle in Prolog zu überführen, damit sie dort von einem Tailoring-Experten optimiert werden. Leider ist ein Prozess sehr umfangreich und die Prozessmodelle dementsprechend sehr komplex und ausgedehnt. Ein Tailoring-Experte der nicht an der Entwicklung und Bewertung der Prozesse beteiligt ist müsste auch die Annotationsdaten kennen, damit er mit Diesen den Prozess tailort. Um die Verarbeitung der Prozessmodelle vor allem das Tailorn von Prozessen in Prolog zu erleichtern wäre eine entsprechende Applikation wie die aus der Abbildung 42 bis 44 sinnvoll. Eine Erweiterung dieser Arbeit wäre z.B. die Anpassung oder Erweiterung der Applikation in sofern, dass man dort die in dem SPEM-Editor verwendeten Annotationsdaten auswählen und dadurch den Prozess nach den Annotationdaten neu rekonstruieren kann. Dafür müssten an der Applikation aus der Arbeit von Daniel Eggert [3] die Annotationsdaten in sofern sie fürs Tailoring benötigt werden eingefügt und neue Tailoring-Regeln definiert werden.

6. Bedienungsanleitung

Die Transformation von SPEM in Prolog wird durch das ANT-Skript in Eclipse durchgeführt. Die Bedienung ist recht einfach wenn man zuvor Eclipse und den SPEM-Editor installiert hat. Weitere Informationen sind dafür aus der Bedienanleitung der Arbeit von Andleko Jovancevic [5] zu entnehmen. Für den Teil dieser Arbeit gilt:

Man öffnet ein neues Java-Projekt mit einem beliebigen Namen (z.B. Test). Anschließend drückt man mit der rechten Maustaste auf das so eben erzeugte Projekt und öffnet ein New File. Dieses File muss einen beliebigen Namen mit der Endung .xml haben (siehe Abbildung 49)
Hat man nun ein Namen angegeben und bestätigt, öffnet sich das Verzeichnis das eben erzeugt wurde. In diesem Fall „probe.xml“ dort schreibt man das Ant-Skript (wie in Abbildung 50) rein.

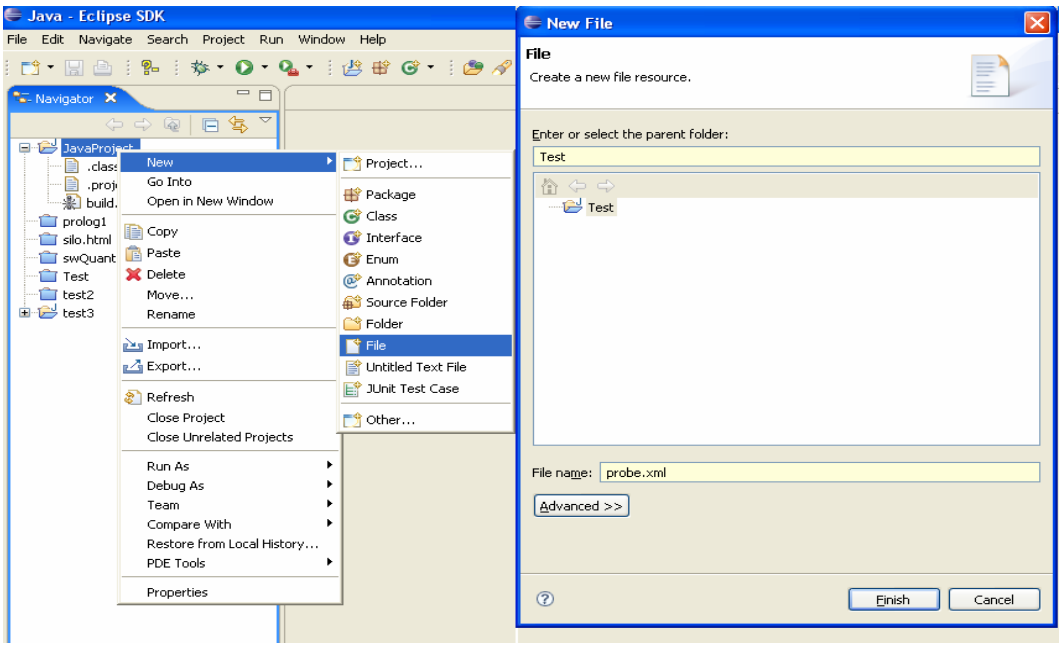
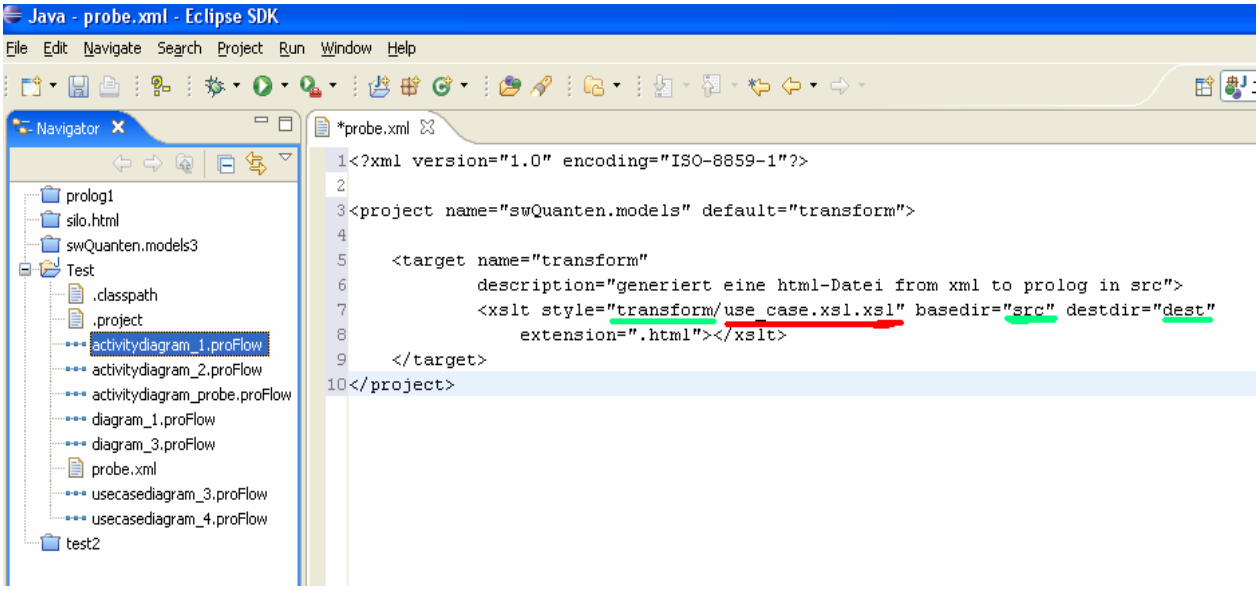


Abbildung 49: Bedienungsanleitung



Abbidung 50: Bedienungsanleitung

Anschließend erzeugt man drei neue Ordner mit den Namen die in der Abbildung 50 grün unterstrichen sind. In den Transform-Ordner speichert man das Stylesheet, das für die Transformation verwendet wird. Dieser Name muss mit dem in der Abbildung 50 rot unterstrichenen Namen übereinstimmen. In den Src-Ordner kommt die Quelldatei, die übersetzt werden soll. Nun kann die Übersetzung beginnen. Dafür muss man unter dem Projekt auf das erzeugte Ant-File (bei diesem Beispiel probe.xml) mit der rechten Maustaste drücken und über „Run As“ auf „Ant build „ klicken und die Übersetzung beginnen. Nach wenigen Sekunden bekommt man eine Bestätigung, dass die Übersetzung erfolgreich war und im dest-Ordner befindet sich dann die Ausgabe. Bei einer Fehlermeldung kann es sein, dass man sich vertippt oder ein falsches Stylesheet verwendet hat. In diesem Programm hat jeder Diagrammtyp ein eigenes Stylesheet. Damit auch die Wissensbasis überschaubar bleibt wurde in dieser Arbeit auch für die Annoation ein eigenes Stylesheet (eins pro Diagrammtyp) ertstellt.

Folgende Sytlesheets wurden für die Transformation erstellt und befinden sich im Ordner Sytlsheeets auf der CD, anbei an die Arbeit.

- Activity : für Aktivitätsdaiagramme ohne Annotation
- Use_case: für Use-Case-Diagramme ohne Annotation
- Klassen: für Klassendiagramme ohne Annotation

- Annotation_activity: für Annotationen im Aktivitätsdiagramm
- Annotation_use_case: für Annotationen im Use-Case-Diagramm
- Annotation_klassen: für Annotationen im Klassendiagramm

7. Quellenverzeichnis und Anhang

Quellenangaben

[1] [SPEM] Software Process Engineering Metamodel Specification Version 1.1
http://www.omg.org/technology/documents/formal/spem.htm
zuletzt online:15.05.2005

[2] SPEM-Paper:Robert Schuppenies, Sebastian Steinhauer
Hasso-Plattner-Institut, Prof.-Dr.-Helmert-Str. 2-3 D-14482 Pots{robert.schuppenies, sebastian.steinhauer}@student.hpi.unipotsdam.de

[3] Bachelorarbeit „Beschreibung und Tailoring von Softwareprozessen in Prolog“
von Daniel Eggert Fachgebiet Software Engineering Leibniz Universität Hannover

[4] SPEM-Tutorial
http://www.omg.org/docs/omg/03-09-04.ppt#304,2,Topics zuletzt online16.05.2005

[5] Bachelorarbeit „Ein grafischer Editor mit Annotationsmöglichkeit für SPEM-Modelle“ von Andelko Jovancevic Fachgebiet Software Engineering
Leibniz Universität Hannover

[6] Bachelorarbeit “Literaturrecherche und Aufbau von Beispielmodellen für Informationsfluss in Softwareprojekten“ von Marco Nötel Fachbereich Softwareengineering Leibniz Universität Hannover

[7] Bachelorarbeit „Integration von Tailoring in eine webbasierte Prozess-Beschreibung“
von Leif Singer Fachgebiet Software Engineering Leibniz Universität Hannover

[8] Vorlesung „Anforderungen und Entwurf“ Fachbereich Softwareengineering
Leibniz Universität Hannover

Abbildungen

Tabellen

Anhang

```
%alle Elemente

workdefinition(elicitation).
workdefinition(interpretation).
workdefinition(negotiation).
workdefinition(dokumentation).
workdefinition(valid_u_verifikation ).
role(kunde).
role(systemanalytiker).
role(protokollant).
role(entwicker).
role(tqa).

%alle Verbindungen

assist(tqa,dokumentation).
assist(protokollant,elicitation).
assist(kunde,negotiation).
assist(tqa,valid_u_verifikation ).
assist(tqa,interpretation).
assist(entwicker,interpretation).
assist(kunde,elicitation).
perform(systemanalytiker,dokumentation).
perform(systemanalytiker,interpretation).
perform(systemanalytiker,valid_u_verifikation ).
perform(systemanalytiker,negotiation).
perform(systemanalytiker,elicitation).
```

Übersetzung Abbildung 29

```
%alle Elemente

phase(anfordrungsanalyse).
phase(entwurf).
workdefinition(systemanalyse).
workdefinition(anforderungsmanagement ).
workdefinition(grob_entwurf).
workdefinition(fein_entwurf).

%alle Verbindungen

include(anfordrungsanalyse,anforderungsmanagement ).
include(entwurf,fein_entwurf).
include(anfordrungsanalyse,systemanalyse).
include(entwurf,grob_entwurf).
```

Übersetzung Abbildung 31

```
%alle Elemente

activity(QGate1).
activity(QGate2).
activity(QGate3).
document(Lasten- heft).
document(exp Base).
document(Nutz rechte Formular).
document(Anf. Spez.).
document(exp Base).
document(CL Fertig fÃ¼r Ent wurf).
document(Proto- koll QG 1).
document(Exp.Base).
document(Ent wurf).
document(Review Protokoll).
document(CL Fertig fÃ¼r Implementierung).
document(exp Base).
document(Protokoll QG2).
document(Exp Base).
document(Bedien- anleitung).
document(Code).
document(CL Fertig fÃ¼r Abnahme).
document(exp_Base).
document(Protokoll QG3).
phase(Anforde- rungsa- nalyse).
phase(Entwurf).
phase(implementierung).
phase(Abnahme).
processrole(QA/SE Berater).
processrole(Kunde).
processrole(Ent- wickler).
processrole(QA-FG).
processrole(qA/SE-Berater).
processrole(qA-FG).
processrole(Ent- wickler).
processrole(Kunde).
processrole(qA-FG).
processrole(Kunde).
processrole(Projekt manager).
workproduct(Proto- typ).
workproduct(Programm).
```

```
phase_input(implementierung,Protokoll QG2).
phase_output(Anforde- rungsa- nalyse,Nutz rechte Formular).
phase_output(implementierung,Programm).
phase_input(Anforde- rungsa- nalyse,Proto- koll QG 1).
activity_input(QGate2,CL Fertig fÃ¼r Implementierung).
verbunden(Ent wurf,fork_fulhph_1176417947915).
verbunden(fork_ecnqtl_1176419040356,QGate3)
activity_input(QGate3,).
phase_input(Anforde- rungsa- nalyse,exp Base).
activity_input(QGate1,exp Base).
verbunden(Code,fork_ecnqtl_1176419040356).
phase_input(implementierung,Exp Base).
phase_input(implementierung,Protokoll QG3).
activity_input(QGate3,exp_Base).
phase_input(Anforde- rungsa- nalyse,Lasten- heft).
verbunden(Anf. Spez.,fork_mvjbjn_1176416496257).
verbunden(fork_fulhph_1176417947915,QGate2)
activity_input(QGate2,).
activity_output(QGate2,Protokoll QG2).

phase_input(implementierung,Ent wurf).
phase_input(Entwurf,Protokoll QG2).
verbunden(Programm,fork_ecnqtl_1176419040356).
phase_input(Entwurf,Anf. Spez.).
activity_output(QGate3,Protokoll QG3).

phase_input(Abnahme,Bedien- anleitung).
phase_input(Entwurf,Exp Base).
verbunden(Bedien- anleitung,fork_ecnqtl_1176419040356).
phase_output(implementierung,Proto- typ).
phase_output(Entwurf,Review Protokoll).
verbunden(Nutz rechte Formular,fork_mvjbjn_1176416496257).
phase_output(implementierung,Bedien- anleitung).
responsibility(QA/SE Berater,Anforde- rungsa- nalyse).
responsibility(qA-FG,QGate2).
responsibility(qA-FG,QGate3).
responsibility(qA/SE-Berater,Entwurf).
responsibility(Projekt manager,Abnahme).
responsibility(Ent- wickler,Anforde- rungsa- nalyse).
responsibility(Kunde,Abnahme).
responsibility(Kunde,Anforde- rungsa- nalyse).
responsibility(Kunde,implementierung).
responsibility(Ent- wickler,implementierung).
responsibility(QA-FG,QGate1).
```

Übersetzung Abbildung 26