PROPEL - Eine Sprache zur Beschreibung von Prozessmustern

Mariele Hagen, Volker Gruhn

Lehrstuhl für Angewandte Telematik / e-Business
Institut für Informatik
Universität Leipzig
Klostergasse 3, D-04109 Leipzig
{hagen,gruhn}@ebus.informatik.uni-leipzig.de

Abstract: Prozessmuster ermöglichen die modulare Modellierung und flexible Anwendung von Softwareprozessen. Gegenwärtige Beschreibungen von Prozessmustern weisen jedoch Mängel wie uneinheitliche und uneindeutige Beschreibungsformen und fehlende Beziehungsdefinitionen auf. Diese Mängel wirken sich nachteilig auf den effektiven Einsatz von Prozessmustern aus. In dieser Arbeit stellen wir die Sprache PROPEL (Process Pattern Description Language) vor, die Konzepte zur Beschreibung von Prozessmustern und Beziehungen zwischen Prozessmustern bereitstellt. Mit Hilfe von PROPEL können einzelne Prozessmuster modelliert und durch Definition von Beziehungen zu komplexeren Prozessen zusammengesetzt werden. Durch die Darstellung eines Prozessmuster-Katalogs in verschiedenen Sichten können die Prozessmuster und ihre Beziehungen übersichtlich dargestellt werden. Ein Beispiel illustriert, wie ein Prozessmuster-Katalog und die zugehörigen Prozessmuster modelliert werden.

1 Einleitung

Ein Prozessmuster repräsentiert einen erprobten (Muster-)Prozess, der ein häufig auftretendes Problem löst (s. [Co96] für eine Einführung). Das Problem verkörpert eine konkrete Situation, die während der Softwareentwicklung eintreten kann. Der Prozess beschreibt eine Menge von Aktivitäten, die zur Lösung des Problems ausgeführt werden können. Prozessmuster bieten folgende Vorteile [DGH02], [St00], [BRS98]:

- **Dokumentation erprobten Wissens.** Prozessmuster dienen dazu, Wissen über Prozesse zu dokumentieren, welches sich bewährt hat. Da die Autoren nur nützliches Prozesswissen dokumentieren, können sich die Anwender darauf verlassen, dass der Prozess auch wirklich „funktioniert“ und damit wiederverwendbar ist.
- **Basis für Kommunikation.** Prozessmuster dienen als Kommunikationsgrundlage für alle an einem Prozess beteiligten Personen. Prozessmuster sind hierfür besonders gut geeignet, da sie einen Prozess und das von ihm zu lösende Problem kompakt darstellen.

- **Abstraktion von Details.** Prozessmuster lassen stets bestimmte Details unberücksichtigt. Hierdurch kann ein Prozessmuster in vielen Situationen eingesetzt („instantiiert“) werden. Einer Prozessmuster-Instanz kann also - wenn erforderlich - weitere Detailinformationen hinzugefügt werden.
- **Flexible Prozessunterstützung.** Durch den Einsatz von Prozessmustern wird nur jeweils die notwendige Prozessunterstützung vom Anwender angefordert. Tritt ein Problem auf, wird ein entsprechendes Prozessmuster - wenn vorhanden - eingesetzt. Der Anwender kann ggf. unter verschiedenen Varianten auswählen oder auch Verfeinerungen eines Prozessmusters einsetzen. Alternativ kann er auch einen individuellen Prozess durchführen. Durch die Angabe von Beziehungen kann der Anwender bei jedem Prozessmuster erkennen, welche Prozessmuster er z.B. als nächstes anwenden kann oder welche Prozessmuster im Rahmen eines übergeordneten Prozessmusters angewendet werden können. Hierdurch können Prozessmuster, obwohl klar getrennte Prozessfragmente, zu komplexeren Prozessen miteinander verknüpft werden. Prozessmuster können entsprechend der jeweiligen Erfordernis im Projekt dynamisch ausgewählt und angewendet werden. Aus diesem Grund werden Prozessmuster auch als Alternative zu herkömmlichen Vorgehensmodellen wie Rational Unified Process [RUP04], V-Modell [VM97], Objectory Process [Ja92] usw. betrachtet, wenn es darum geht, den Softwareprozess an die jeweilige Projektsituation anzupassen [BRS98].

Obwohl wie oben erläutert das Konzept der Prozessmuster sehr vielversprechend ist, ist es noch keineswegs ausgereift [Ha02]. Bisherige Arbeiten beschäftigen sich eher mit der Identifikation neuer Prozessmuster statt mit der geeigneten Beschreibung oder verfolgen eine andere Zielrichtung. Das Forschungsgebiet „Workflow Patterns“ zielt z.B. darauf ab, implementierungsnahe Muster für Entwurf, Realisierung und Vergleich von Workflowmanagementsystemen zu identifizieren [Aa02]. Derzeitige Beschreibungen von Prozessmustern weisen noch folgende Mängel auf:

- **Keine einheitliche Beschreibung.** Es gibt mittlerweile eine Reihe von Beschreibungsschemata für Patterns wie das GOF-Schema [Ga95] oder das Alexanderian-Schema [Al77]. Da aber jedes Beschreibungsschema von einer anderen Domäne herrührt (GOF aus der Software-Design-Domäne, Alexanderian aus der Architektur-Domäne), ist jedes Schema auch speziell auf diese Domäne abgestimmt und dafür geeignet. Für Autoren und Anwender von Patterns führt dies zu einigen Irritationen, da nicht klar ist, wie die Patternelemente zu beschreiben und zu interpretieren sind.
- **Mangelnde Beschreibung der Prozessmuster-Beziehungen.** Bisherige Prozessmuster vernachlässigen die Beschreibung ihrer Beziehungen zu anderen Prozessmustern. Prozessmuster können jedoch ihre volle Kraft erst im Zusammenspiel mit anderen Prozessmustern entfalten [Co96]. Durch die Kombination einzelner Prozesse können somit komplexere Prozesse modelliert werden. Diese Information muss durch eine explizite Beschreibung der Prozessmuster-Beziehungen mitgeliefert werden. Gegenwärtige Beschreibungen von Beziehungen sind jedoch nicht formal und geben keine Auskunft über ihre syntaktische oder semantische Bedeutung (z.B. „Pattern X uses Pattern Y“ or „Pattern A can be combined with pattern B“).

- **Keine präzise Beschreibung.** Prozessmuster wurden bislang auf informelle Weise durch natürliche Sprache beschrieben. Dieses Vorgehen ist einerseits ein Vorteil, da für das Verstehen eines Patterns kein Wissen über Notation, Syntax und Semantik erforderlich ist. Natürliche Sprache ist andererseits ein informelles Ausdrucksmittel und produziert mehrdeutige und unpräzise Ausdrücke (vgl. z.B. die Beziehungen „is prerequsite of“ und „may contain“ in [St00]). Neben der unpräzisen Beschreibung von Beziehungen sind erstaunlicherweise die Beschreibungen der Prozesse informell. Eine schrittweise Beschreibung des Prozesses fehlt oft, und es gibt sogar Prozessmuster, die gar keinen Prozess als Lösung enthalten (s. z.B. [Co94]).

Aus den vorgenannten Gründen haben wir die Sprache PROPEL (Process Pattern Description Language) zur Beschreibung von Prozessmustern entwickelt (s. [Di02] für eine detaillierte Übersicht). Diese Sprache basiert auf der Unified Modeling Language (UML) [UML03]. Hierdurch sollen möglichst viele bewährte und verbreitete Modellierungskonzepte der „Lingua Franca“ des Software Engineering wiederverwendet werden. Die UML besitzt bereits Konzepte zur Modellierung von Prozessen mit Hilfe von Aktivitätsdiagrammen, die wir uns für die Modellierung des Prozesselements eines Prozessmusters zunutze machen. PROPEL propagiert ein einheitliches Beschreibungsschema für Prozessmuster und definiert Beziehungen zwischen Prozessmustern. PROPEL erlaubt hiermit die semiformale Beschreibung (u.a. durch OCL-Constraints) von Prozessmustern und deren Beziehungen.

In Kapitel 2 präsentieren wir die grundlegenden Konzepte von PROPEL. Anschließend stellen wir in Kapitel 3 eine Auswahl an Prozessmustern und Problemen vor, die mit Hilfe von PROPEL modelliert wurden. In Kapitel 4 fassen wir unsere Ergebnisse zusammen und geben einen Ausblick auf zukünftige Arbeiten.

2 Grundlegende Konzepte der Process Pattern Description Language PROPEL

Die Erweiterung der UML zur Process Pattern Description Language PROPEL findet auf der Metamodellebene (Schicht M2) statt (s. Abbildung 1). Die Schicht M2 enthält Konzepte in Form von Metaklassen und Metaassoziationen zur Modellierung der Prozessmuster. Die Schicht M1 beschreibt Prozessfragmente in Form von Prozessmustern. Die Entitäten dieser Schicht sind Instanzen der Elemente von Schicht M2. Die zuunterst liegende Schicht M0 von PROPEL enthält instantiierte, d.h. zur Laufzeit eines Projekts angewendete Prozessmuster. Die Entitäten dieser Schicht sind Instanzen der Elemente von Schicht M1.

Analog zur UML-Spezifikation haben wir die neu eingeführten Konzepte in den Metamodell-Paketen „Process Pattern Package“ (s. Abschnitt 2.1), „Relationship Package“ (s. Abschnitt 2.2) und „Process Pattern Graph Package“ (wird hier nicht weiter betrachtet, eine Auswahl der Diagrammtypen ist jedoch im Rahmen des Beispiels in Kapitel 3 ersichtlich) gebündelt (s. Abbildung 1, Metamodell-Ebene M2).

Für jedes Paket erläutern wir die abstrakte Syntax von PROPEL für jede Metaklasse. Aus Platzgründen verzichten wir an dieser Stellen auf die Angabe der OCL-Constraints. In [Di02] können diese nachgelesen werden.

Referenzieren wir ein Metamodell-Element, so machen wir dies durch Fettschrift kenntlich (z.B. **ProcessPattern** für die Metamodell-Klasse). Das jeweilige Assoziationsende geben wir in Kursivschrift an (z.B. *solves*). In den folgenden Abbildungen werden von uns neu eingeführte Metaklassen schattiert dargestellt. Alle Metaklassen sind, wenn nicht anders angegeben, Subklassen der Metaklasse **ModelElement**.

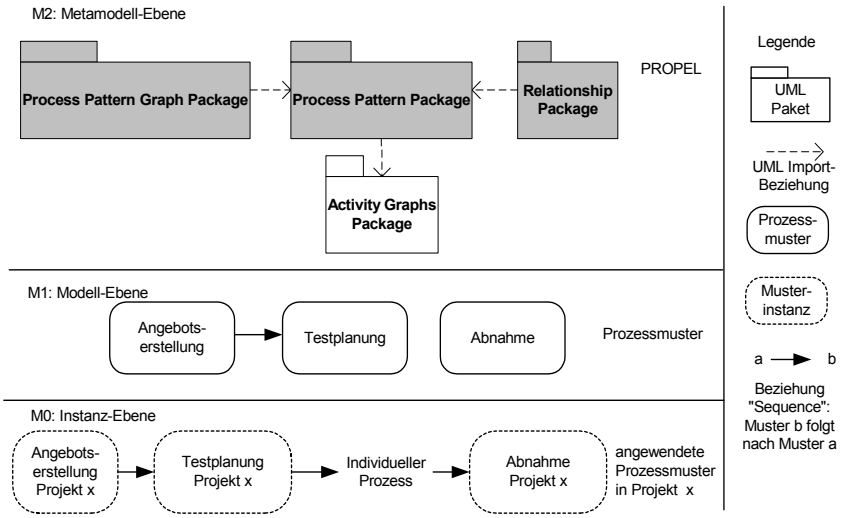

Abbildung 1: Schichten der Process Pattern Description Language PROPEL

2.1 Process Pattern Package

Das Process Pattern Package beinhaltet alle notwendigen Konzepte, um ein einzelnes Prozessmuster beschreiben zu können.

ProcessPattern (s. Abbildung 2): Ein Prozessmuster beschreibt einen bewährten Prozess, um ein Problem, das in einem bestimmten Kontext wiederholt aufgetreten ist, zu lösen. Das Problem repräsentiert eine Situation, die im Rahmen eines Softwareentwicklungsprojekts auftreten kann (*problem*). Der Kontext spezifiziert Objekte und Ereignisse, die vor und nach der Ausführung des Prozessmuster vorliegen (*initial* und *resulting*). Der Prozess spezifiziert Prozessschritte, die notwendig sind, um das Problem zu lösen (Assoziation *process*).
Ein Prozessmuster ist Teil des Aggregats **ProcessPatternCatalogue**, d.h. ein Prozessmuster wird stets einem Prozessmuster-Katalog zugeordnet (*catalog*). Ein Prozessmuster besitzt ferner beliebig viele Assoziationen mit verschiedenen Beziehungen, die das Pro-

zessmuster mit einem anderen Prozessmuster verknüpft. Das Prozessmuster kann ferner thematisch eingeordnet werden (*aspect*).

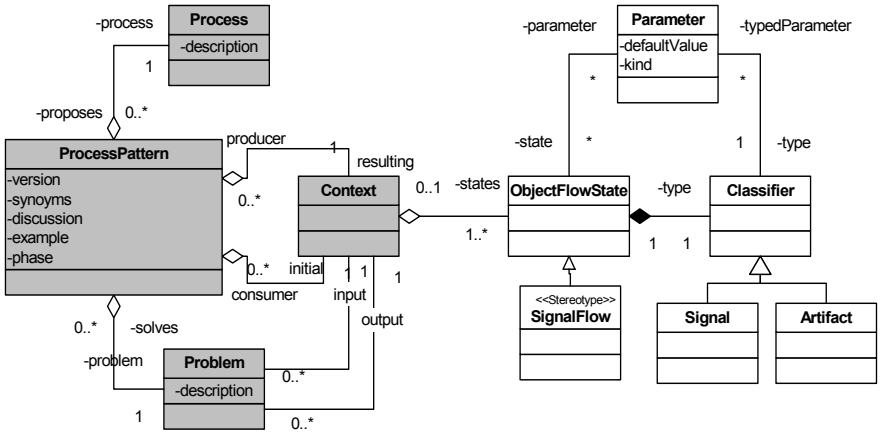

Abbildung 2: Abstrakte Syntax des Process Pattern Packages - Teil 1

Problem (s. Abbildung 2): Ein Problem beschreibt die Problemstellung bzw. die Ziele, die mit dem Einsatz eines Musters gelöst bzw. erreicht werden sollen. Im PROPEL-Metamodell wird diese Forderung durch die Metaklasse **Problem** beschrieben. Probleme werden durch einen Input und einen Output spezifiziert (*input* und *output*). Input und Output sind jeweils Mengen von Objekten und Ereignissen (**ObjectFlowState**). Ein Problem kann thematisch eingeordnet werden (*aspect*). Die Assoziation *solves* beschreibt, dass es für ein Problem beliebig viele Lösungen (d.h. Prozessmuster) geben kann. Die ausführliche textuelle Beschreibung des Problems wird durch das Attribut *description* ausgedrückt.

Context (s. Abbildung 2): Der Kontext eines Prozessmusters definiert die Bedingungen, die vor und nach Anwendung des Prozessmusters erfüllt sein müssen. Ein Kontext ist Teil eines Prozessmusters. Er wird entweder von einem Prozessmuster konsumiert (consumer) oder produziert (producer). Ein Kontext ist eine Menge von Objekten und und Ereignissen (states). Ein Objektzustand wird durch die UML-Metaklasse ObjectFlowState repräsentiert, dem als Classifier (d.h. als Typ) ein Artefakt (Artifact) zugeordnet wird. Ein Ereignis wird ebenfalls durch das Stereotyp SignalFlow repräsentiert, der Typ des Classifiers ist dann ein Signal (s. [UML03], S. 2-180: „For applications requiring that actions or activities bring about an event as their result, use an object flow state with a signal as a classifier.“).

Abbildung 3 zeigt Beispiele für Prozessmuster, Problem und Kontext.

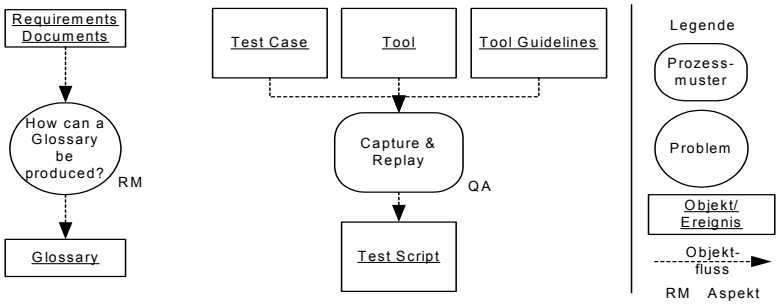

Abbildung 3: Beispiele für Problem (links) und Prozessmuster (rechts), jeweils mit Kontext

Process (s. Abbildung 2 und Abbildung 4): Ein Prozess kann Bestandteil beliebig vieler Prozessmuster sein (*proposes*, Abbildung 2). Für die Modellierung von Prozessen greifen wir auf UML-Aktivitätsdiagramme (**ActivityGraph**) zurück. Ein Aktivitätsdiagramm ist eine spezielle Zustandsmaschine, wobei Zustände (**State**) als Aktivitäten (**ActionState**) interpretiert werden. Ein Aktivitätsdiagramm beschreibt den Kontroll- und den Objektfluss zwischen seinen Aktivitäten. Obwohl Aktivitätsdiagramme eigentlich dazu konzipiert sind, das Verhalten von Software zu beschreiben, werden sie zunehmend zur Modellierung von organisatorischen Prozesse eingesetzt (vgl. [UML03], S. 2-172). Im Metamodell ist ein Prozess eine Spezialisierung der Metaklasse **ActivityGraph** (s. Abbildung 4). Der Prozess erläutert, wie die vom Prozessmuster adressierte Problemstellung gelöst wird.

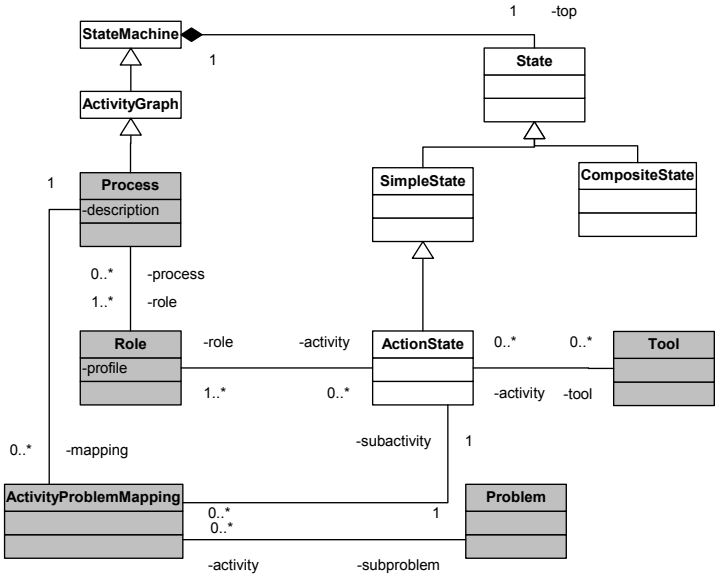

Abbildung 4: Abstrakte Syntax des Process Pattern Packages- Teil 2

Für die Ausführung des Prozesses ist eine Menge von Rollen verantwortlich (*role*, s. Abbildung 4). Die ausführliche textuelle Beschreibung der Lösung wird durch das Attribut *description* ausgedrückt.

Die hierarchische Komposition von Prozessen wird durch die Abbildung von Problemen auf Aktivitäten gelöst. Den Aktivitäten innerhalb eines Prozesses können passende Probleme zugeordnet werden (*mapping*). Diese Probleme sind somit Subprobleme des Problems des übergeordneten Patterns. Dies weist darauf hin, dass ggf. weitere Prozessmuster zur Lösungs des Teilschritts vorhanden sind. Die Existenz einer oder mehrer Zuordnungen von Problemen zu Aktivitäten ist also Voraussetzung für die hierachische Komposition (s. Beziehung Use in Abschnitt 2.2).

ActivityProblemMapping (s. Abbildung 4): Ein **ActivityProblemMapping** erlaubt es, einer Aktivität (*subactivity*) ein Problem (*subproblem*) zuzuordnen. Zu diesen Problemen können dann wiederum Prozessmuster zugeordnet werden.

Role (s. Abbildung 4): Eine Rolle beschreibt die notwendigen Erfahrungen, Kenntnisse und Fähigkeiten, um Aktivitäten durchzuführen. In der UML-Spezifikation gibt es bereits die ähnlich anmutende Metaklasse **Actor**. Die Zuordnung von dieser Metaklasse zu anderen Modellelementen ist jedoch auf das zu entwickelnde System beschränkt, d.h. es sind nur Assoziationen zu Uses-Cases, Subsystemen und Klassen erlaubt. Wir benötigen jedoch für die Modellierung von Prozessen einen weiter gefassten Begriff und haben deshalb die Metaklasse **Role** eingeführt. Rollen werden Aktivitäten (Assoziation *activity*) und Prozessen (Assoziation *process*) zugeordnet. Die Tätigkeiten und Aufgaben einer Rolle können über ein Profil (s. Attribut *profile*) detailliert beschrieben wird.

Tool (s. Abbildung 4): Ein Werkzeug unterstützt die Ausführung einer Aktivität. Bei Werkzeugen handelt es sich meistens um Softwaresysteme. Mit dem Einsatz eines Werkzeugs können verschiedene Ziele verknüpft sein, diese Ziele können sich von Aktivität zu Aktivität unterscheiden. Ein Werkzeug kann mit Aktivitäten assoziiert werden.

Abbildung 5 zeigt ein Prozess-Beispiel.

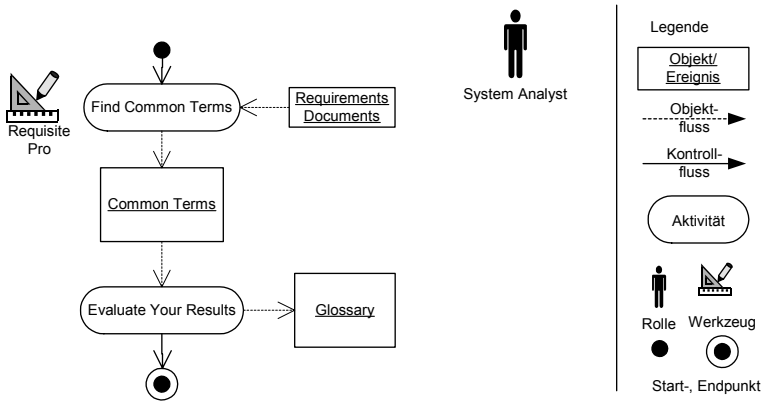

Abbildung 5: Beispiel für Prozess

2.2 Relationship Package

Das Relationship Package beinhaltet alle notwendigen Konzepte, um die Beziehung zwischen Prozessmustern beschreiben zu können (Übersicht auf Abbildung 7).

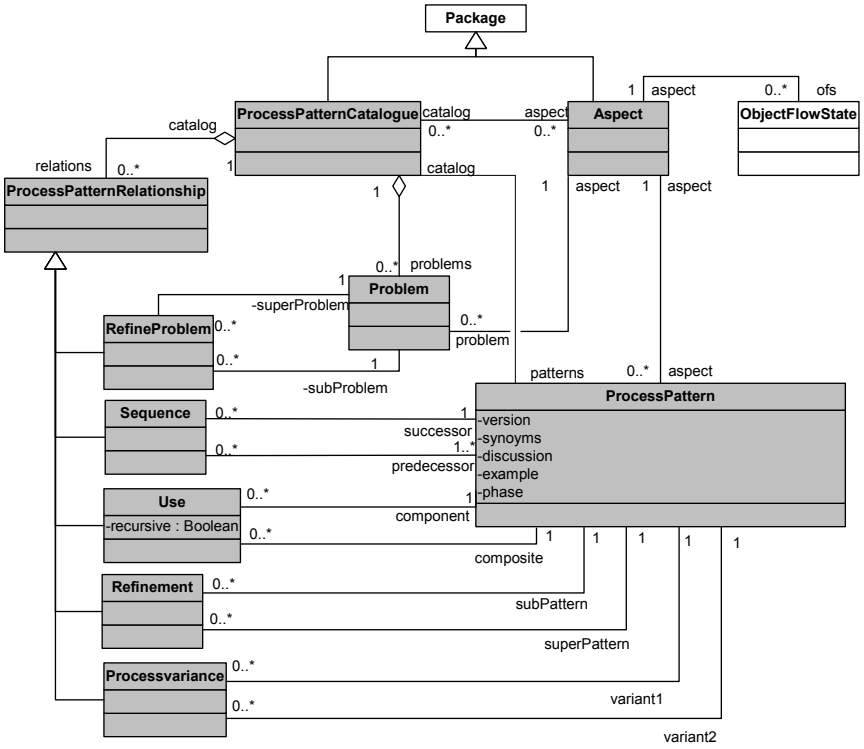

Abbildung 6: Abstrakte Syntax des Process Pattern Relationship Packages

ProcessPatternRelationship (s. Abbildung 6): Diese Metaklasse ist ein Supertyp für alle Prozessmuster-Beziehungen. Eine Beziehung wird einem Prozessmuster-Katalog **(ProcessPatternCatalogue)** zugeordnet (*catalog*).

Sequence (s. Abbildung 6): Die Sequence-Beziehung verknüpft ein oder mehrere vorausgehende Prozessmuster (*predecessor*) und ein nachfolgendes Prozessmuster (*successor*). Zwischen mehreren Prozessmustern besteht eine solche Beziehung, wenn die vorausgehenden Prozessmuster zusammen alle Objekte und Ereignisse produzieren, die das nachfolgende Prozessmuster für seine Ausführung benötigt. Die Verknüpfung zu den Prozessmustern einer Sequence-Beziehung wird durch die beiden Assoziationen *predecessor* und *successor* realisiert.

Use (s. Abbildung 6): Die Use-Beziehung setzt jeweils zwei Prozessmuster (Kompositmuster und Komponentenmuster) zueinander in Beziehung. Zwischen zwei Prozessmus-

tern besteht eine solche Beziehung, wenn das Komponentenmuster einen Teilprozess des Kompositmusters (d.h. einen Teil der Lösung) beschreibt. Die Use-Beziehung beschreibt keine Teil-Ganzes-Beziehung: Ein Kompositmuster kann auch ohne Zuhilfenahme eines vorhandenen Komponentenmusters angewendet werden. Komponentenmuster verkörpern also stets die Option, eine bestimmte Aktivität des Kompositmusters detaillierter ausführen zu können. Die Wahrnehmung diese Option liegt jedoch beim Anwender. Die Verknüpfung zu den beiden Prozessmustern einer Sequence-Beziehung wird durch die beiden Assoziationen *composite* und *component* realisiert.

Processvariance (s. Abbildung 6): Die Processvariance-Beziehung setzt jeweils zwei Prozessmuster zueinander in Beziehung. Zwischen zwei Prozessmustern besteht eine solche Beziehung, wenn die beiden Varianten zwar das gleiche Problem lösen, dafür aber unterschiedliche Lösungen anbieten. Die Verknüpfung zu den beiden Prozessmustern einer Processvariance-Beziehung wird durch die beiden Assoziationen *variant1* und *variant2* realisiert.

Refinement (s. Abbildung 6): Prozessmuster können auf verschiedenen Abstraktionsstufen existieren. Prozessmuster hoher Abstraktionsstufen besitzen weniger Details als Prozessmuster niedriger Abstraktionsstufen. Stellt ein Prozessmuster eine Abstraktion eines anderen Prozessmusters dar, so nennen wir diese Beziehung Refinement-Beziehung. Die Refinement-Beziehung setzt jeweils zwei Prozessmuster zueinander in Beziehung. Sie bedeutet, dass das detailliertere Muster das abstraktere Muster verfeinert, d.h. ein Pattern wird als Spezialisierung des anderen, allgemeineren Musters aufgefasst. Das Superpattern enthält ein abstrakteres Problem und einen abstrakteren Prozess zur Lösung dieses Problems als das Subpattern. Die Refinement-Beziehung impliziert also, dass die Probleme der betroffenen Prozessmuster in einer RefineProblem-Beziehung stehen (s. weiter unten). Die Verknüpfung zu den beiden Prozessmustern einer Refinement-Beziehung wird durch die beiden Assoziationen *superPattern* und *subPattern* realisiert.

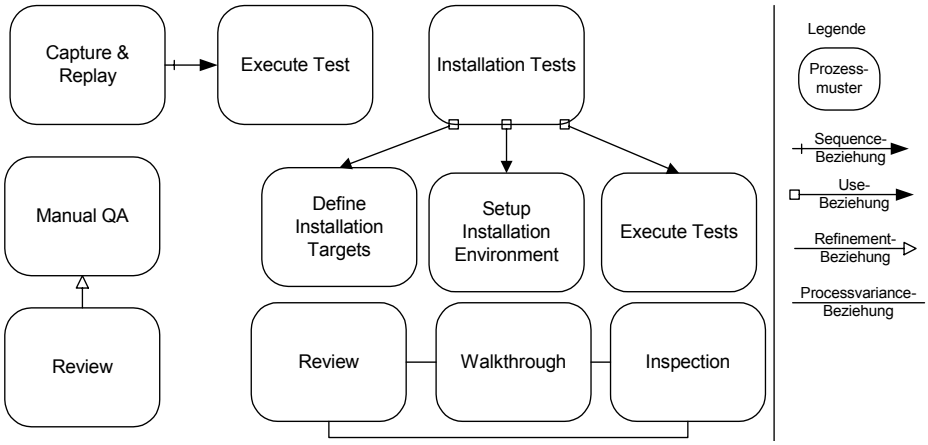

Abbildung 7: Beispiele für Beziehungen zwischen Prozessmustern und Problemen

RefineProblem (s. Abbildung 6): Die RefineProblem-Beziehung setzt jeweils zwei Probleme (Superproblem und Subproblem) zueinander in Beziehung. Zwischen zwei Proble-

men besteht eine solche Beziehung, wenn das Subproblem das Superproblem verfeinert. Im Rahmen der Verfeinerung wird der Input und der Output des Superproblems jeweils durch Input und Output des Subproblems verfeinert (s. z.B. Objekt „Requirements Documents“, welches im Input des Subproblems verfeinert wird; das Objekt „Glossary“ wird nicht verfeinert, ist also sowohl im Super- als auch im Subproblem vorhanden). Die Verknüpfung zu den beiden Problemen einer RefineProblem-Beziehung wird durch die beiden Assoziationen *superProblem* und *subProblem* realisiert. Ein Beispiel für eine RefineProblem-Beziehung zeigt Abbildung 8.

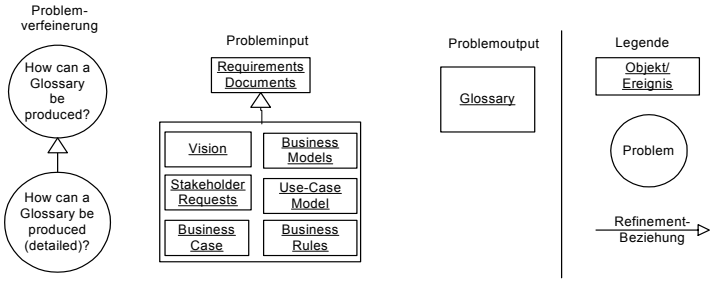

Abbildung 8: Beispiel für RefineProblem-Beziehung

ProcessPatternCatalogue (s. Abbildung 6): Ein Prozessmuster-Katalog repräsentiert eine Menge von Prozessmustern und Beziehungen, die Prozessmuster miteinander verknüpfen. Ein Prozessmuster-Katalog ist eine Subklasse von **Package**.

Aspect (s. Abbildung 6): Der Aspekt ist Subklasse von **Package** und bündelt eine Menge von Prozessmustern, Problemen, Objekten und Ereignissen. Der Aspekt gibt Aufschluss über die thematische Ausrichtung dieser Elemente. Beispiele für Aspekte sind „Requirements Management (RM)“ und „Quality Assurance (QA)“ (s. Abbildung 3).

3 Beispiel

Für die Validierung der Konzepte von PROPEL haben wir den Prozessmuster-Katalog CADS (Catalog for the Development of Software) entwickelt, der die Anwendung von PROPEL veranschaulicht. Den Katalog haben wir vom Rational Unified Process (RUP) abgeleitet ([Kr00], [RUP04]). Auf diese Weise konnten wir in der Praxis bewährte und eingesetzte Prozesse verwenden und uns auf die Form der Beschreibung konzentrieren.

Das Beispiel gliedert sich in zwei Teile, in die Darstellung der Katalogsichten (Abschnitt 3.1) und in die Darstellung eine ausgewählten Problems und eines Prozessmusters (Abschnitt 3.2). Die Katalogsichten dienen dazu, wie in einem Inhaltsverzeichnis sich zunächst einen Überblick auf vorhandene Prozessmuster und deren Beziehungen verschaffen zu können. Interessiert sich der Anwender für ein bestimmtes Problem oder Prozessmuster, kann er sich dies dann im Detail ansehen.

Die Darstellung der in diesem Kapitel aufgeführten Prozessmuster beschränkt sich auf Prozessmuster, die der Disziplin bzw. dem Aspekt „Requirements Management“ angehören. Prozessmuster, die anderen Aspekten angehören (z.B. „Business Modeling“), werden durch ein entsprechendes Kürzel gekennzeichnet (z.B. „BM“ für „Business Modeling“). Aus Platzgründen können hier nur eine Auswahl von Prozessmuster gezeigt werden, die Darstellung ist also nicht vollständig in Bezug auf den Katalog, aber vollständig in Bezug auf den gewählten Ausschnitt.

3.1 Sichten des Prozessmuster-Katalogs

Nachfolgend zeigen wir den von uns gewählten Ausschnitt (Aspekt „Requirements Management“) des Prozessmuster-Katalogs mit Hilfe verschiedener Sichten. Jede Sicht vermittelt eine Übersicht über die Beziehungen eines bestimmten Typs. Prinzipiell können auch alle Sichten in einer Sicht (das entspräche dann der Darstellung des Gesamtkatalogs) vereinigt werden, was jedoch zu Lasten der Übersichtlichkeit geht.

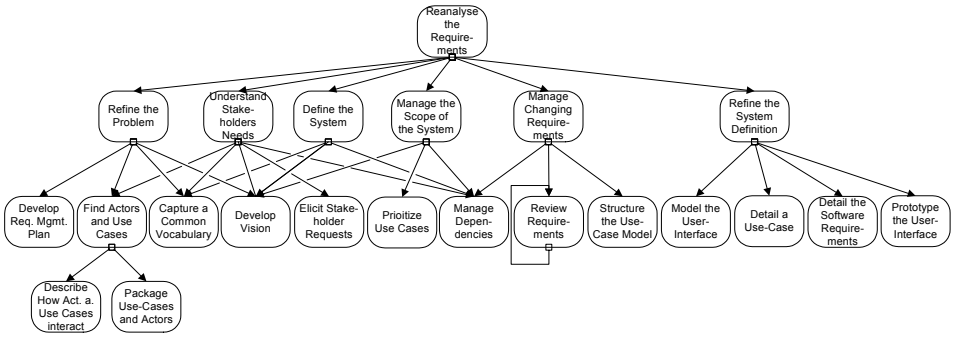

Abbildung 9: Katalogdiagramm - Sicht „Use-Beziehung“

Abbildung 9 zeigt in einem Ausschnitt den hierarchischen Aufbau des Katalogs CADS mit Hilfe der Use-Beziehung. Dieser hierarchische Aufbau wurde weitestgehend aus dem RUP übernommen (da dieser bereits hierarchisch in Disciplines, Workflowdetails und Activities vorstrukturiert ist). Anhand der Abbildung wird deutlich, dass einige Prozessmuster mehrfach genutzt werden, wie z.B. das Prozessmuster „Develop Vision“. Das Prozessmuster „Review Requirements“ steht mit sich selbst in einer (rekursiven) Use-Beziehung. Eine solche hierarchische Übersicht ist im RUP nicht vorhanden (vgl. mit [He01]: „The RUP does not offer appropriate support for structuring complex software processes. It ignores most powerful mechanisms of computer science for mastering complexity: hierarchy, recursion and orthogonality.“).

Abbildung 10 zeigt eine Auswahl der existierenden Sequence-Beziehungen. Die Sequence-Beziehung bedeutet eine Abfolge von Prozessmustern, d.h. dass ein Prozessmuster nach einem oder mehreren Prozessmustern ausgeführt werden kann. Diese Sicht hat den Vorteil, dass man auf einen Blick erkennen kann, welche Prozessmuster vor oder nach einem Prozessmuster angewendet werden können oder müssen. Um beispielweise

das Prozessmuster „Find Actors and Use Cases“ ausführen zu können, können die Prozessmuster „Elicit Stakeholder Requests“, „Find Business Workers and Entities“ usw. vorher ausgeführt werden (s. Abbildung 10). Diese Art von Information ist im RUP nur implizit vorhanden.

Die Anzahl der Sequence-Beziehungen mit einem vorausgehenden Prozessmuster ist in diesem Beispiel eher gering. Dies liegt darin begründet, dass der RUP in neun verschiedene Disziplinen aufgeteilt ist, aber meistens Prozesse (d.h. Workflowdetails oder Aktivitäten) mehrerer Disziplinen zusammenwirken, um ein Objekt oder ein Ereignis zu produzieren. Wegen dieser engen Verknüpfung der Prozesse ist das Vorkommen von Sequence-Beziehungen mit mehr als einem vorausgehenden Prozessmuster viel höher (s. Abbildung 10).

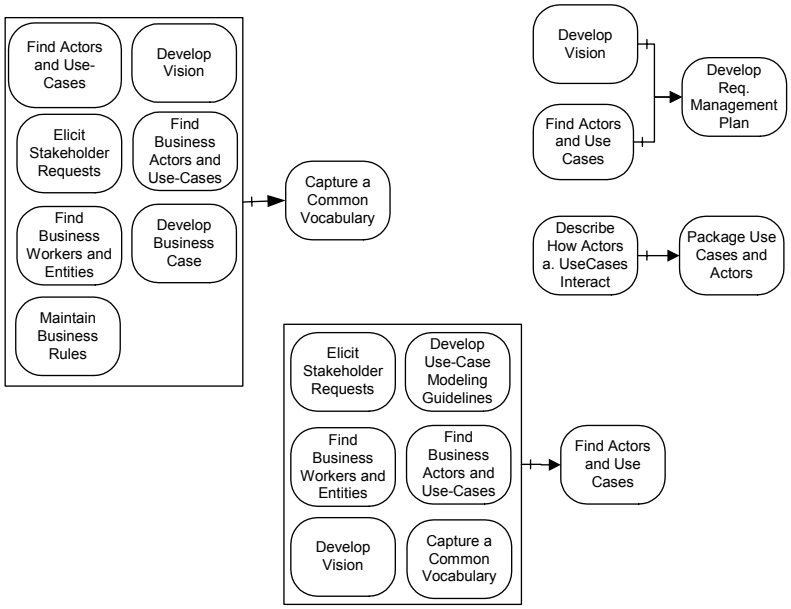

Abbildung 10: Katalogdiagramm - Sicht „Sequence-Beziehung“

Die Darstellung der Sequence-Beziehung ist ein Vorteil gegenüber der Darstellung des RUPs, in der nur jeweils die Input- und die Outputartefakte eines Workflowdetails bzw. einer Aktivität angegeben werden, aber nicht mögliche Reihenfolgen. Gerade weil im RUP die Aktivitäten, Workflowdetails und Disziplinen eng miteinander verzahnt sind, ist eine solche informative Darstellung hilfreich.

Abbildung 11 zeigt eine Auswahl von Refinement-Beziehungen. Z.B. existiert zu dem Prozessmuster „Capture a Common Vocabulary“ die verfeinernden Prozessmuster „Vocabulary by Project Team“ und „Vocabulary by System Analyst“. Diese beiden Prozessmuster beschreiben detaillierter als das abstrakte Pattern, wie ein Projektglossar erstellt

werden kann. Detaillierter sind hierbei jeweils Kontext und Prozess der verfeinernden Prozessmusters. Der Anwender kann sich also bei Vorliegen einer Refinement-Beziehung entscheiden, wieviel Unterstützung (Superpattern - weniger Unterstützung, Subpattern - mehr Unterstützung) er zur Lösung des Problems benötigt. Der RUP bietet dagegen keine Möglichkeiten zur Beschreibung von Prozessverfeinerungen an.

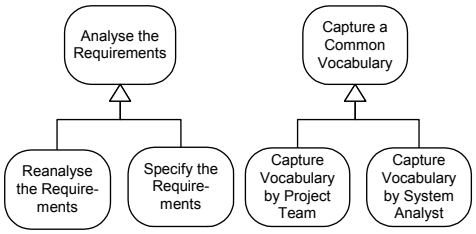

Abbildung 11: Katalogdiagramm - Sicht „Refinement-Beziehung“

Abbildung 12 zeigt in einer Übersicht eine Auswahl der Processvariance-Beziehungen. Prozessvariante Pattern lösen jeweils das gleiche Problem. Hierzu gehören beispielsweise die drei Prozessmuster „Review Requirements“, „Walkthrough Requirements“ und „Inspect Requirements“. Diese drei Prozessmuster zeigen, wie auf unterschiedliche Art eine manuelle Prüfung durchgeführt werden kann. Der Anwender kann sich also bei Vorliegen einer Processvariance-Beziehung entscheiden, wie er das Problem lösen möchte. Auf variante Prozesse wird im RUP nicht eingegangen. Dies ist ein Nachteil für den Anwender, da er somit stets auf ein bestimmtes Vorgehen festgelegt ist. Ggf. kennt er auch eine variante Vorgehensweise, die er der Prozessbeschreibung des RUPs gerne hinzufügen würde. Dies ist jedoch nicht möglich.

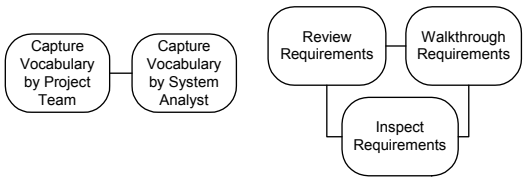

Abbildung 12: Katalogdiagramm - Sicht „Processvariance-Beziehung“

3.2 Problem und Prozessmuster

Innerhalb eines Projekts soll ein Projekt-Vokabular entwickelt werden, um die Kommunikation zwischen den Mitarbeitern zu verbessern und eindeutige Formulierungen in den Dokumenten zu erreichen. Diese Aufgabe wird durch das Problem „How can a Glossary be produced?“ repräsentiert (s. Abbildung 13). Dort sehen wir links das zu lösende Problem mitsamt Input und Output, rechts die lösenden Prozessmuster abgebildet. Das Problem wird durch das Prozessmuster „Capture a Common Vocabulary“ gelöst. Zu diesem

Prozessmuster gibt es zwei Verfeinerungen, Prozessmuster „Capture Vocabulary by System Analyst“ und Prozessmuster „Capture Vocabulary by Project Team“.

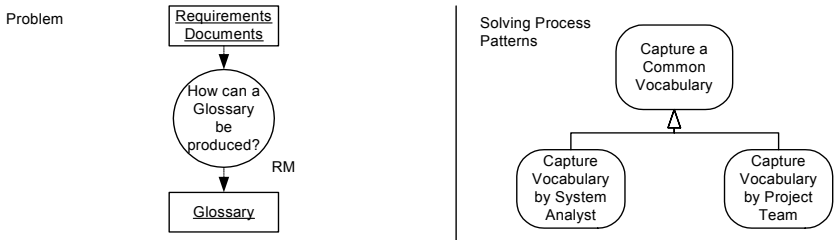

Abbildung 13: Problem „How can a Glossary be produced?“

Abbildung 14 zeigt die Beschreibung des Prozessmusters „Capture a Common Vocabulary“. Neben der Darstellung des zu lösenden Problems (oben) werden die Beziehungen des Prozessmusters (Mitte) und der Prozess (unten) des Prozessmusters dargestellt. Die einzelnen Prozessmusterelemente wurden bereits in Abschnitt 2 eingeführt.

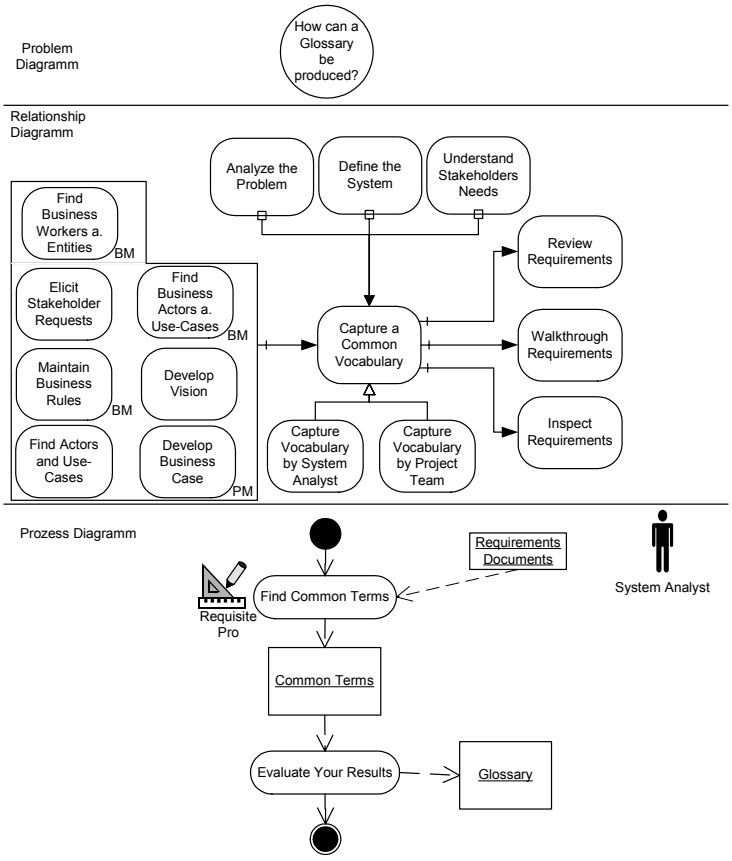

Abbildung 14: Prozessmuster Diagramm „Capture a Common Vocabulary“

An diesem Beispiel wird deutlich, dass ein komplexes Vorgehensmodell wie der RUP mit Hilfe von Prozessmustern beschrieben werden kann. Die Beschreibung durch Prozessmuster hat mehrere Vorteile: Durch die Darstellung von Prozessmustern eines Katalogs in Sichten kann sich der Pattern Anwender schnell einen Überblick über alle vorhandenen Prozessmuster verschaffen. Durch die Identifikation und Formulierung der Prozessmuster Beziehungen erhält er weitere Hinweise für die Anwendung von Patterns. Hat er z.B. das Prozessmuster „Capture a Common Vocabulary“ angewendet, ist erkennbar, dass anschließend das Prozessmuster „Review Requirements“, „“Walkthrough Requirements“ oder „Inspect Requirements“ angewendet werden kann (s. Abbildung 14, Sequence-Relation zwischen diesen Mustern). Durch die explizite Formulierung von Problemen können prozessvariante Prozessmuster identifiziert und damit dem Anwender Alternativen vorgeschlagen werden.

4 Zusammenfassung und Ausblick

Die in dieser Arbeit vorgestellten Process Pattern Description Language PROPEL besitzt alle notwendigen Mittel, um Prozessmuster in einer einheitlichen Form zu beschreiben. Hierzu gehört die Beschreibung von Problemen, Kontexten, Objekten und Ereignissen, Rollen und Werkzeugen. Besonders hervorzuheben ist die Möglichkeit, Beziehungen zwischen Prozessmustern und damit zwischen Prozessen zu beschreiben. Die meisten Vorgehensmodelle besitzen keine Möglichkeit zur Darstellung von Prozess-Beziehungen. Die semiformale Darstellung von Prozessen erhöht die Genauigkeit und lässt gleichzeitig zu, Regeln für die hierarchische Struktur von Prozessmustern aufzustellen.

Mit den Mitteln der Process Pattern Description Language PROPEL haben wir einen Prozessmuster Katalog basierend auf Prozessen des Rational Unified Process’ beschrieben. Der Katalog präsentiert sowohl die Prozessmuster als auch die Beziehungen zwischen Prozessmustern mit Hilfe verschiedener Sichten. Die Katalogsichten dienen dazu, wie in einem Inhaltsverzeichnis sich zunächst einen Überblick auf vorhandene Prozessmuster und deren Beziehungen verschaffen zu können.

Um die semiformale Darstellung von Prozessmustern in ein formale Darstellung umzuwandeln, entwickeln wir gegenwärtig die formale Semantik von PROPEL durch Abbildung der Syntax auf die semantische Domäne der Petri-Netze. Für eine weitere verbesserte Handhabbarkeit von Prozessmustern arbeiten wir an der „Process Pattern Workbench“, einer elektronischen Plattform zur Dokumentation und Verwaltung von Prozessmustern. Details über einen Prototypen sind in [Sc03] verfügbar. Die derzeitigen Arbeiten an der „Process Pattern Workbench“ haben zum Ziel, die Anwendung von Prozessmustern aufzuzeichnen, um Projektmitarbeitern Auskunft über die anzuwendenden Prozesse zu geben, aber auch um statistisch auszuwerten, welche Abfolgen von Prozessmustern häufiger angewendet werden als andere (sog. Prozessmuster Sequenzen). Ferner steht noch die Entwicklung der Methodik des „Process Pattern Managements“ aus, mit deren Hilfe Prozessmuster systematisch identifiziert, dokumentiert, ausgewählt und eingesetzt werden können.

Literatur

[Al77] Alexander, C.; Ishikawa, S.; Silverstein, M. et al.: A Pattern Language. Oxford University Press, 1977.

[Aa02] Aalst, W.; Barros, A.; ter Hofstede, A. et.al.: Workflow Patterns. In: QUT Technical report. FIT-TR-2002-02, Queensland University of Technology, Brisbane, 2002.

[BRS98] Bergner, K.; Rausch, A.; Sihling, M.: A Component Methodology based on Process Patterns. TUM-I9823, Universität München, 1998.

[Co94] Coplien, J.: A Development Process Generative Pattern Language. In: Proceedings of PLoP 94, 1994.

[Co96] Coplien, J.: Software Patterns. SIGS Book & Multimedia, 1996.

[DGH02] Dittmann, T., Gruhn, V., Hagen, M.: Improved Support for the definition and usage of process patterns. 1st Workshop on Process Patterns, OOPSLA 2002, Seattle, 2002.

[Di02] Dittmann, T.: PPDL - Eine Beschreibungssprache für Process Patterns, Diplomarbeit, Universität Dortmund, 2002.

[Ga95] Gamma, E.; Helm, R.; Johnson, R.; et.al.: Design Patterns. Addison-Wesley, 1995.

[Ha02] Hagen, M.: Support for the definition and usage of process patterns. Position Paper, EuroPloP 2002, http://www.haase-consulting.com/workshops/FgEuroplop02/position_papers.html.

[He01] Hesse, W.: Dinosaur Meets Archaeopteryx? Seven Theses on Rational's Unified Process (RUP). Proc. CAiSE'98/LFIP 8.1 Int. Workshop on Evaluation of Modelling Methods in System Analysis and Design (EMMSAD'01), Interlaken, 2001.

[Ja92] Jacobson, I.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison Wesley Publishing Company, 1992.

[Kr00] Kruchten, P.: The Rational Unified Process, Addison-Wesley Professional, 2000.

[RUP04] Rational Unified Process, IBM, http://www-306.ibm.com/software/awdtools/rup/index.html, 2004.

[Sc03] Schröder, J.: Die Process Pattern Workbench, Diplomarbeit, Universität Dortmund, 2003.

[St00] Störrle, H.: Models of Software Architecture. Design and Analysis with UML. Dissertation, Universität München, 2000.

[UML03] Object Management Group: OMG Unified Modeling Language Specification, March 2003, Version 1.5, formal/03-03-01. http://www.omg.org/docs/formal/03-03-01.pdf.

[VM97] V-Modell ’97: Entwicklungsstandard für IT-Systeme des Bundes. BWB IT 15, 1997.