AristaFlow: Komponentenbasierte Anwendungs-entwicklung, Prozesskomposition mittels Plug & Play und adaptive Prozessausführung

Colin Atkinson [1], Peter Dadam [2]
[1] Lehrstuhl für Softwaretechnik, Universität Mannheim
[2] Institut für Datenbanken und Informationssysteme, Universität Ulm

Abstract

Das AristaFlow-Projekt hat zum Ziel, den gesamten Lebenszyklus prozess-orientierter Anwendungssysteme zu unterstützen, d.h. von der Komponentenent-wicklung, über die komponentenorientierte Prozesskomposition bis hin zur adaptiven Prozessausführung. Im Beitrag wird vorgestellt, welche Anforderungen an eine "ideale" Entwicklungsumgebung für Anwendungskomponenten in diesem Zusammenhang gestellt werden, wie diese Komponenten bei der Prozesskomposition mittels Plug & Play verwendet werden und was die Laufzeitumge-bung (das ADEPT2-System) leisten muss, um eine adaptive und gleichzeitig auch robuste Prozessausführung zu unterstützen.

1 Einführung

Unter Schlagworten wie Workflow Management (WFM), Business Process Management (BPM), Enterprise Application Integration (EAI) oder Service-orientierte Architekturen (SOA) kommen derzeit vermehrt Technologien auf den Markt, welche die rechnerbasierte Unterstützung von Geschäftsprozessen im Fokus haben. Sie alle verfolgen das Ziel, die Effizienz und Anpassungsfähigkeit der Unternehmen hinsichtlich ihrer internen Geschäftsprozesse zu steigern sowie ihre Interaktion mit Kunden und Geschäftspartnern besser zu unterstützen.

Typisch für alle diese Ansätze ist, dass sie eine Trennung zwischen Prozess- und Anwendungslogik vornehmen bzw. voraussetzen. D.h. man geht davon aus, dass die heute noch vorherrschenden monolithischen, funktions- und datenzentrierten Anwen-dungssysteme durch Funktions- bzw. Service-Bibliotheken abgelöst werden, die es ermöglichen, die Anwendungsfunktionen relativ frei miteinander zu kombinieren. Die zu unterstützenden Prozesse werden getrennt davon in einem Prozess-Management-System beschrieben und hinterlegt. Dieses verknüpft dann die Anwendungsfunktionen entsprechend der hinterlegten Ablaufbeschreibung.

Damit diese Technologien auch tatsächlich den gewünschten Nutzen bringen, müssen sie zum einen eine sehr viel raschere Implementierung neuer Prozesse unterstützen und zum anderen Ad-hoc-Abweichungen vom vorgeplanten Ablauf ermöglichen. Beide Aspekte implizieren, dass systemseitig gewisse Zusicherungen übernommen werden müssen, dass die so erzeugten bzw. ad hoc modifizierten Prozesse

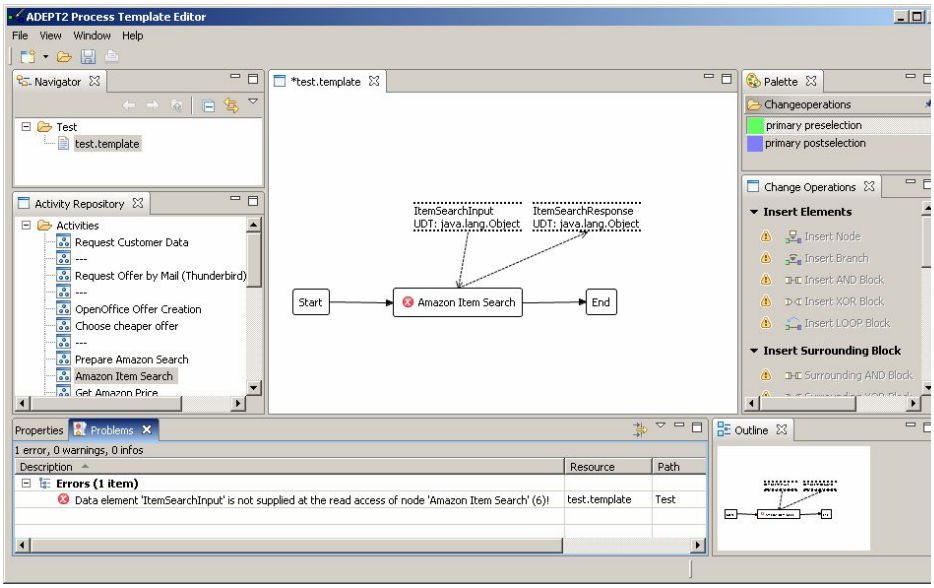

Abb. 1.: Einfügen einer Anwendungsfunktion aus dem Activity Repository

bzw. Prozessinstanzen keine Laufzeitfehler verursachen, etwa durch Aufruf von Anwendungsfunktionen mit unversorgten Aufrufparametern. Solche Zusicherungen sind aber nur dann möglich, wenn die Entwicklungsumgebung (für die Prozesskomposition) und die Laufzeitumgebung (im Falle von Ad-hoc-Abweichungen) über eventuell bestehende Abhängigkeiten zwischen den Anwendungsfunktionen Bescheid wissen. Hierzu gehören unter anderem das Wissen über Aufrufparameter und Rückgabewerte sowie eventuell zu beachtender Reihenfolge-Einschränkungen beim Aufruf von Anwendungsfunktionen, etwa wegen versteckter Datenflüsse zwischen Anwendungsfunktionen oder bei zustandsbehafteten Anwendungskomponenten. Nur so kann systemseitig durch entsprechende Analysen festgestellt werden, ob die vom Prozessmodellierer (zur Entwurfszeit) oder vom Anwender (im Falle von Ad-hoc-Änderungen auf Prozessinstanzebene) gewünschte Ausführungsreihenfolge der Anwendungsfunktionen realisierbar ist oder nicht.

Abb. 1 zeigt einen Screenshot des ADEPT2-Editors. Aus dem Activity Repository wurde die Anwendungsfunktion “Amazon Item Search” gewählt und per “Drag & Drop” in den Prozessgraph eingefügt und für deren Ein- und Ausgabeparameter (da noch nicht vorhanden) systemseitig nach Rückfrage geeignete Datenelemente (Prozessvariablen) angelegt. Da der Eingabeparameter im Prozessablauf bislang noch nicht „versorgt“ wird, wird im *Problems*-Fenster eine entsprechende Fehlermeldung ausgegeben. Eine Prozessvorlage kann erst dann an die ADEPT2-Laufzeitumgebung zur (späteren) Instanziierung übergeben werden, wenn diese keine Fehler mehr enthält.

Analoge Prüfungen finden zur Laufzeit im Kontext von Ad-hoc-Änderungen sowie bei Prozess-Schemaevolution [RiDa03, Rind04] statt. Die gewünschten Änderungen werden nur gewährt, wenn dies zu keinen Inkonsistenzen führt. – Eine ausführlichere Beschreibung dieser Aspekte sowie den resultierenden Anforderungen an die ADEPT2-Ausführungsumgebung (d.h. die „Prozessengine“), die an der Universität Ulm entwickelt wird, findet sich im Tagungsband zum doIT-Forschungstag 2005 (siehe [DRRA05]).

Der Schwerpunkt dieses Beitrags liegt auf der Konzeption und Realisierung der Entwicklungsumgebung für Anwendungsfunktionen. Wie bereits oben erläutert, müssen im Activity Repository Informationen über die Anwendungsfunktionen sowie eventuell zu beachtende Abhängigkeiten zwischen Anwendungsfunktionen abgelegt werden. Ziel des AristaFlow-Projektes ist es, diese Information möglichst ohne Zusatzaufwand für den Anwendungsentwickler aus dem Entwurfs- und Entwicklungsprozess heraus zu gewinnen bzw. abzuleiten.

Im folgenden Kapitel werden zunächst die Anforderungen an eine solche Entwicklungsumgebung (incl. der Prozesskomposition) aus Sicht des Entwicklers von Anwendungsfunktionen bzw. Anwendungskomponenten sowie des Prozessentwicklers beschrieben und das Zusammenspiel zwischen Entwicklungs- und Ausführungsumgebung erläutert. In Kapitel 3 werden verschiedene Aspekte und Teilbereiche der Entwicklungsumgebung diskutiert und in Kapitel 4 die Nutzung der Komponenteninformation durch das Prozess-Management-System erläutert. Kapitel 5 schließt mit einer Zusammenfassung und einem Ausblick.

2 Problemstellung

Die zunehmende Bedeutung von komponentenbasierten und service-orientierten Entwicklungsansätzen in der Softwaretechnik spiegelt sich in der Verbreitung modellgetriebener Verfahren wider [MuMi03]. Die modellgetriebene Softwareentwicklung ergänzt die traditionelle reine Quelltextbeschreibung von Softwaresystemen und Komponenten durch „plattformunabhängige“ Modelle, die Beschreibungen aus verschiedenen Sichtweisen auf die Komponenten liefern. Es gibt jedoch keinen (de jure oder de facto) Standard der beschreibt, wie die verschiedenen Sichtweisen und Diagrammtypen der modellgetriebenen Entwicklung (z.B. Klassendiagramme, Zustandsdiagramme, Constraints) benutzt werden sollen, um Komponenten oder die Kombination von Komponenten zu beschreiben.

In der Praxis benutzen Entwickler daher unstrukturierte Ansammlungen von Artefakten, die lose miteinander gekoppelt sind. Dies kann zu Konsistenzproblemen führen, da keine werkzeuggestützten Mechanismen zur Prüfung vorhanden sind, ob die verschiedenen Sichten wechselseitig konsistent sind. Außerdem können Verwaltungs- und Navigationsprobleme auftreten, da die Zuordnung von logischen Komponenten zu konkreten Sichten nicht explizit von Werkzeugen unterstützt wird. Mit der Unter-

stützung von Plug & Play für Prozess-Management-Systeme verstärken sich diese Probleme, etwa wenn für diese Systeme weitere Informationen zu der schon vorhandenen Ansammlung von Diagrammen und Sichten von Projekten hinzugefügt werden. Beispielsweise muss für Anwendungskomponenten sichergestellt werden, dass ihre Methoden beim Einsatz im Geschäftsprozess in der richtigen Reihenfolge ausgeführt werden und eventuell notwendige Vorbedingungen (z.B. Wertebereiche von Eingabeparametern) eingehalten werden. Bei „falschem“ Einsatz soll der Benutzer idealerweise schon bei der Modellierung eines Prozesses gewarnt werden. Zur Laufzeit müssen unerwünschte Methodenaufrufe vor Ausführung abgefangen werden und sinnvolle Rückmeldungen für Benutzer erzeugt werden.

Im Rahmen des AristaFlow-Projektes werden Konzepte zur Modellierung solcher Komponenten entwickelt. Es wird untersucht, welche Informationen von Prozess-Management-Systemen für Plug & Play benötigt werden und woher diese im Entwicklungsprozess kommen. Diese Informationen werden von ebenfalls entwickelten Verfahren genutzt, um den „richtigen“ Einsatz von Anwendungskomponenten in Prozess-Management-Systemen sicherstellen. Für die Komponentenmodellierung entsteht derzeit an der Universität Mannheim eine Entwicklungsumgebung, die

- ein systematisches, benutzerfreundliches Konzept und Rahmenwerk zur Definition und Navigation verschiedener Sichten auf Komponenten und/oder komponentenbasierten Systemen bietet,
- eine flexible Infrastruktur zur Verfügung stellt, welche die Erweiterung mit Werkzeugen zur Erzeugung von Sichten und zu Konsistenzprüfungen auf einfache und systematische Weise ermöglicht und
- ein einheitliches Metamodell bietet, welches die automatische Erzeugung von Sichten aus einer einzigen zugrunde liegenden Repräsentation eines komponentenbasierten Systems ermöglicht.

2.1 Von der Komponente zum fertigen Prozess

Im AristaFlow-Projekt wird das Spektrum von der Entwicklung einer Komponente bis zu ihrem Einsatz in einem Prozess-Management-System betrachtet (Abb. 2). Zunächst wird die Komponente von einem Komponenten-Entwickler modelliert und implementiert, um dann in einem öffentlichen Verzeichnis publiziert zu werden. Ein Administrator des Unternehmens, das Prozesse erstellen möchte, kann hierfür nun geeignete Komponenten in das unternehmenseigene Verzeichnis kopieren. Beim Import kann der Administrator den einzelnen Aktivitätenvorlagen Bearbeiterrollen zuordnen und außerdem Installationsinformationen angeben. Weiterhin können Taxonomieangaben angepasst werden und Parametern unternehmensweit eindeutige Bezeichner zugeordnet werden, welche das Prozess-Management-System später auswerten kann. Ein Prozess-Entwickler kann mit einem Editor Prozessvorlagen und Komponenten zu ausführbaren Prozess-Schemata kombinieren.

2.2 Zusammenspiel von Entwicklungs- und Ausführungsumgebung

Je mehr Informationen die Ausführungsumgebung für Prozesse von einer Komponente erhält, desto mehr Prüfungen können vorgenommen werden. Viele der Prüfungen werden bereits zur Modellierzeit der Prozess-Schemata durchgeführt, damit ungültige Schemata nicht zur Ausführung kommen können. Für die Fälle, dass Modellierzeitprüfungen unmöglich oder zu komplex sind, werden Prüfungen zur Laufzeit eingesetzt. Diese können unerwünschtes Verhalten einer Komponente verhindern, indem beispielsweise vor Aufruf einer Methode einer Komponente geprüft wird, dass die vom Komponentenentwickler festgelegten Vorbedingungen erfüllt sind.

Damit Plug & Play zuverlässig funktioniert, sollten so viele Informationen der Komponente wie möglich vom Prozess-Management-System ausgewertet werden. Dies darf aber nicht dazu führen, den Komponentenentwicklungsprozess unnötig zu erschweren. Daher müssen soweit wie möglich Artefakte wiederverwendet werden, die im Softwareentwicklungsprozess ohnehin anfallen. Beispielsweise können vom Entwickler festgelegte Constraints (Vor- und Nachbedingungen für Operationen [ocl06, WaKl03]) in die Komponentenbeschreibung aufgenommen und für Laufzeitprüfungen verwendet werden. Aus UML Diagrammen und dem Quelltext können Beschreibungen für Aktivitätenvorlagen erzeugt werden, welche dann für die Konstruktion von Prozessvorlagen verwendet werden.

Um das Programmieren herkömmlicher Komponenten zu unterstützen wird der Komponentenentwickler jedoch nicht verpflichtet, alle Möglichkeiten der

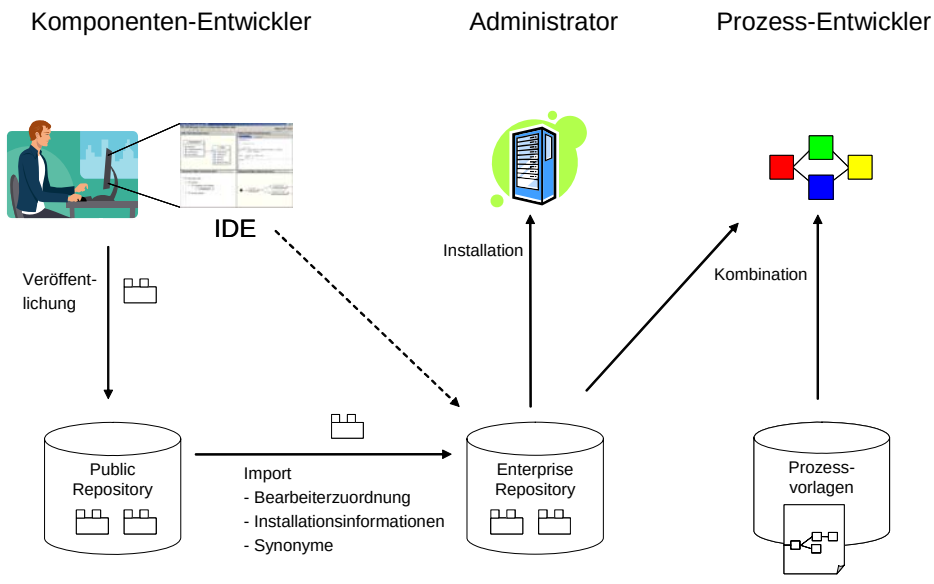

Abb. 2.: Entwicklungsprozess

Entwicklungsumgebung zu nutzen. Es gibt eine minimale Menge von Artefakten die für das Komponentenverzeichnis notwendig sind, etwa Angaben über ausführbare Methoden einer Komponente.

3 Entwicklungsumgebung für Komponenten

3.1 Metamodell

Während der Entwicklung einer Komponente fallen verschiedene Artefakte wie UML Klassendiagramme, UML Zustandsdiagramme und Quelltexte an. Alle repräsentieren verschiedene Sichten auf eine Komponente und stehen normalerweise in Beziehung zueinander. Mit einer zunehmenden Anzahl von Artefakten wird die Navigation unübersichtlich, ebenso wie die Erhaltung der Konsistenz verschiedener, miteinander in Beziehung stehender Sichten. Eine ideale Lösung hierfür wäre es, wenn jedes Werkzeug direkt auf einem einzigen gemeinsamen Metamodell arbeiten würde und Änderungen direkt mit diesem Modell synchronisiert werden würden (Abb. 3). So wäre die Konsistenz zwischen den Werkzeugen und damit den verschiedenen Sichten sichergestellt.

Die Universität Mannheim entwickelt derzeit ein einheitliches Modell für alle Arten von Artefakten. Das langfristige Ziel ist die Anbindung aller Komponenteneditoren der Entwicklungsumgebung an dieses Modell. Als pragmatische Lösung werden mittelfristig mehrere Formate existieren, welche durch die Entwicklungsumgebung synchronisiert werden. Die Entwicklungsumgebung ist flexibel konzipiert, so dass neue Editoren als Erweiterungen eingefügt werden können. Für das AristaFlow Projekt wurden Editoren entwickelt, die speziell dafür konzipiert sind, notwendige Informationen für das Komponentenrepository zu bearbeiten.

3.2 Sichtenkonzept

Die Navigation und Verwaltung von Artefakten kann für den Entwickler durch ein Sichtenkonzept und eine geeignete Werkzeugunterstützung wesentlich erleichtert werden. Komponentenmodellierungstechnologien wie KobrA [ABB+02] benutzen verschiedene Sichten auf Komponenten für verschiedene Aufgaben, um die Komplexität für den Modellierer beherrschbar zu machen. Dies ist auch ein Ziel der Entwicklungsumgebung für AristaFlow. Komponenten werden hier als mehrdimensionaler Würfel betrachtet, in dem eine Zelle einen Aspekt einer Komponente beschreibt (z.B. den Verhaltensaspekt einer Bank-Komponente). Eine Zelle des Würfels wird ausgewählt, indem ein Wert für jede Dimension gewählt (oder der voreingestellte Wert übernommen) wird (Abb. 4).

Ein Beispiel für eine Zellauswahl wäre die „strukturelle“ Projektion, die „Black Box“ Abstraktion und die Auswahl von „Bank“ als Komponente. Eine Zelle kann eine oder mehrere Perspektiven haben (z.B. UML Klassendiagramm oder eine Zuordnung

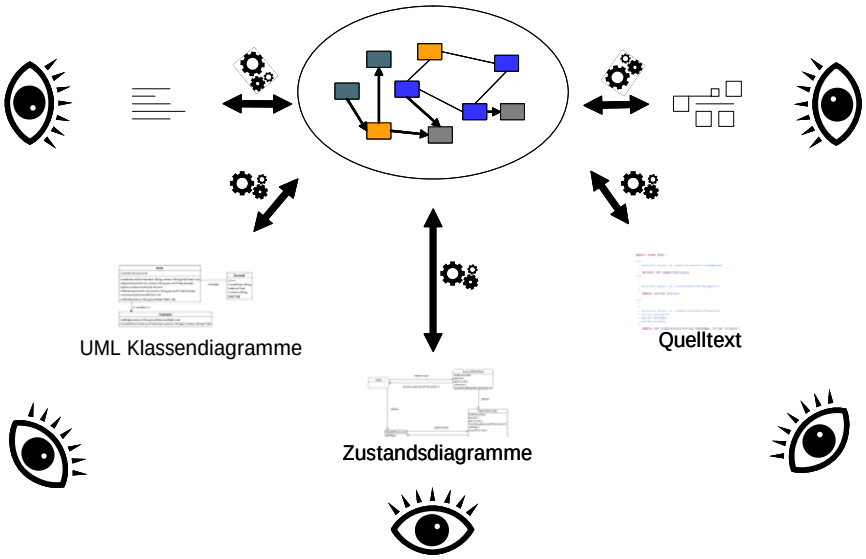

Abb. 4.: Ideale Verknüpfung von Sichten und Modell

von Taxonomien zu Klassen). Eine Perspektive kann vom Entwickler mit einem Editor bearbeitet werden.

Im Normalfall ist einer Perspektive ein Editor zugeordnet, z.B. einem UML Klassendiagramm das UML Werkzeug MagicDraw. Um größere Flexibilität zu ermöglichen, können einer Perspektive auch mehrere Editoren zugeordnet werden, etwa wenn zwei alternative Werkzeuge für UML Klassendiagramme vorhanden sind.

Auch die Dimensionen der Entwicklungsumgebung sind änderbar und um zusätzliche Dimensionen erweiterbar. Für die Entwicklung von Workflowkomponenten werden voreingestellte Dimensionen und Dimensionselemente zur Verfügung gestellt. Neben der Komponentendimension, die als Dimensionselemente die zu bearbeitenden

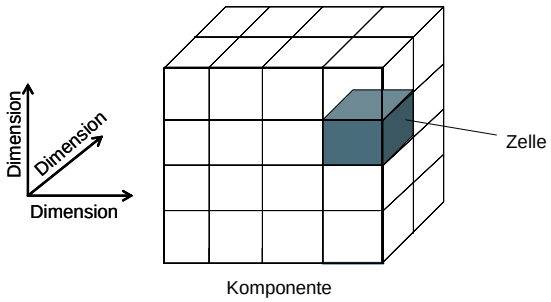

Abb. 3.: Komponentennavigation

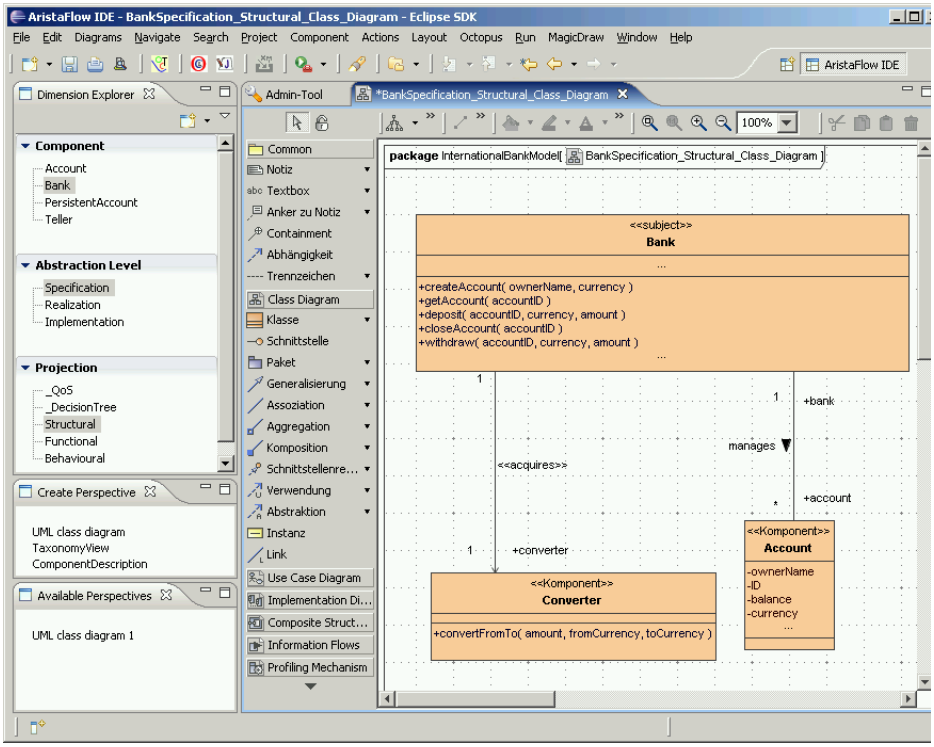

Abb. 5.: Entwicklungsumgebung mit dem UML Diagramm einer Bank

Komponenten oder Sub-Komponenten beinhaltet, stehen die Dimensionen Projektion und Abstraktion zur Verfügung.

Die Abstraktion gibt den Detaillierungsgrad einer Komponente an. Zur Auswahl stehen in der Voreinstellung die Elemente Spezifikation, Realisierung und Implementierung. Die Spezifikation stellt eine „Black Box“-Sicht auf die Komponente dar, während die Realisierung eine „White Box“-Sicht bezeichnet. Auf der Implementierungsebene kann der Quelltext bearbeitet werden.

Die Dimension Projektion besteht aus einer strukturellen, funktionalen und Verhaltensprojektion. Die strukturelle Projektion zeigt z.B. die Zusammenhänge zwischen Klassen, während die funktionale Projektion die Spezifikation von einzelnen Methoden ermöglicht. Erlaubte Aufrufreihenfolgen von Methoden können in der Verhaltensprojektion festgelegt werden. Abb. 5 zeigt, wie in der Entwicklungsumgebung gerade ein UML Diagramm der Bank-Komponente auf der „Black Box“ Abstraktionsebene und der strukturellen Projektion bearbeitet wird.

3.3 Erweiterungen

Neben der Möglichkeit, weitere Dimensionen und Dimensionselemente hinzuzufügen können ebenfalls eigene Editoren und Werkzeuge zur Konsistenzprüfung und Erzeugung von Sichten geschrieben und in die Entwicklungsumgebung eingefügt werden. Für die Projektion könnte etwa das Element „Quality of Service“ aufgenommen werden. Denkbare neue Dimensionen wären eine Versionsdimension mit verschiedenen Komponentenversionen als Dimensionselementen oder eine Dimension mit Produkten einer Familie für das Product Line Engineering. Eigene Werkzeuge oder Assistenten könnten Daten für Editoren erzeugen, neue Konsistenzprüfungen vornehmen oder zusätzliche Komponentenexportfunktionalitäten bereitstellen.

4 Nutzung von Komponenteninformationen in Prozess-Management-Systemen

Das Entwickeln von Anwendungskomponenten für Prozess-Management-Systeme soll möglichst wenig Mehraufwand im Vergleich zur Entwicklung herkömmlicher Komponenten bedeuten. Daher wird versucht, möglichst viele Informationen des normalen Softwareentwicklungsprozesses auszuwerten und möglichst automatisch in Formate zu übersetzen, welche von einem Prozess-Management-System ausgewertet werden können. Zudem müssen nicht alle Möglichkeiten der Entwicklungsumgebung genutzt werden, etwa um schnell „herkömmliche“ Komponenten zu implementieren.

Für die Prozessmodellierung benötigte Aktivitätenvorlagen können weitgehend automatisch aus dem Quelltext erzeugt werden. Stehen Parameter von Komponenten unterschiedlicher Hersteller im Zusammenhang (z.B. zwei unterschiedliche Parameter, die eine Kontonummer darstellen) ermöglicht eine Zuordnung von Bezeichnern die automatische Verknüpfung von Datenelementen und Parametern im Prozess-Management-System. Die Zuweisung einer Komponente oder einzelner Methoden zu einer oder mehreren Taxonomien ist ebenso möglich. Im Taxonomieeditor der Entwicklungsumgebung können eigene Taxonomien importiert und bearbeitet werden.

Gibt der Komponentenentwickler eine Verhaltensbeschreibung seiner Komponente an, z.B. als Zustandsdiagramm, dessen Übergänge durch Methodenaufrufe ausgelöst werden oder als regulären Ausdruck, ist beispielsweise ein Einsatz von Protokollprüfern möglich. So wird sichergestellt, dass der Prozessmodellierer die Aktivitäten einer Komponente im Rahmen der von ihrem Entwickler vorgesehenen Reihenfolgeeinschränkungen einsetzt.

Constraints, d.h. Vor- und Nachbedingungen für einzelne Operationen sowie Invarianten können für Laufzeitprüfungen verwendet werden. Bevor die Prozessausführungsumgebung z.B. eine Methode aufruft, kann die Vorbedingung geprüft werden. Der Code für die Überprüfung kann durch die Komponentenentwicklungsumgebung automatisch erzeugt werden und mit der Komponente geliefert werden. Eine nicht eingehaltene Vorbedingung, die zu teuren und schwer nachvollziehbaren

Laufzeitfehlern führen könnte, kann damit abgefangen und stattdessen eine für den Workflowadministrator sinnvolle Rückmeldung erzeugt werden.

5 Zusammenfassung und Ausblick

Das AristaFlow-Projekt verfolgt das Ziel, eine Technologiebasis zu entwickeln und prototypisch zu implementieren, welche den gesamten Lebenszyklus prozessorientierter Informationssysteme abdeckt: Von der zweckmäßigen Modellierung und Implementierung geeigneter Anwendungskomponenten, über die Prozesskomposition mittels „Plug & Play“ bis hin zur flexiblen und adaptiven Prozessausführung. Ein wesentliches Ziel hierbei ist, durch geeigneten Entwurf und Implementierung der Komponenten sowie darauf basierender Methoden bei der Prozesskomposition, spätere Laufzeitfehler weitgehend auszuschließen. Im vorliegenden Beitrag wurde schwerpunktmäßig der Aspekt des Entwurfs und der Implementierung der Anwendungskomponenten behandelt und gezeigt, welche Maßnahmen in diesem Zusammenhang ergriffen werden, um dieses Ziel zu erreichen.

Die auf diesen Konzepten basierende Entwicklungsumgebung, ein Repository für Komponenten sowie Konzepte zur Auswertung von Artefakten aus dem Softwareentwicklungsprozess für Prozess-Management-Systeme befinden sich in der Implementierung.

Danksagung

Wir danken den Mitarbeitern der Forschungsgruppe des AristaFlow Projektes Hilmar Acker, Kevin Göser, Martin Jurisch, Ulrich Kreher, Markus Lauer, Dietmar Stoll und den vielen Studenten für ihre wertvollen Beiträge. Weiterhin bedanken wir uns bei Matthias Gutheil, Markus Kalb, Thao Ly, Michael Predeschly, Manfred Reichert und Stefanie Rinderle ihre Unterstützung.

Literatur

[ABB+02] C. Atkinson, J. Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, J. Zettel: Component-Based Product Line Engineering with UML. Addison-Wesley Publishing Company, 2002.

[DRRA05] P. Dadam, M. Reichert, S. Rinderle, C. Atkinson: Auf dem Weg zu prozessorientierten Informationssystemen der nächsten Generation - Herausforderungen und Lösungskonzepte. In: D. Spath, K. Haasis, D. Klumpp (Hrsg.): Aktuelle Trends in der Softwareforschung - Tagungsband zum doIT Software-Forschungstag 2005, Karlsruhe, Juni 2005. Schriftenreihe zum doIT Software-Forschungstag, Band 3, MFG Stiftung, 2005, S. 47-67

[MuMi03] J. Mukerji, J. Miller: Overview and guide to OMG's architecture. http://www.omg.org/cgi-bin/doc?omg/03-06-01, Juni 2003

[ocl06] Object Management Group: Object Constraint Language Specification, Version 2.0. http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf, Mai 2006.

[Rind04] S. Rinderle: Schema Evolution in Process Management Systems. Dissertation, Universität Ulm, Fakultät für Informatik, Dezember 2004

[RiDa03] S. Rinderle, P. Dadam: Schemaevolution in Workflow-Management-Systemen ("Aktuelles Schlagwort"). Informatik-Spektrum, Band 26, Heft 1, Februar 2003, S. 17-19

[WaKl03] J. Warmer, A. Kleppe: The Object Constraint Language: Getting Your Models Ready for MDA. Second Edition, Addison Wesley, August 2003.