# PESOA

**Process Family Engineering in Service-Oriented Applications**

BMBF Project

# Qualitätsverbesserungen

Autoren:
Cord Giese
Marco Heidenwolf
Frank Puhlmann
Arnd Schnieders
Andrej Werner
Mathias Weske

PESOA-Report Nr. 30/2007
Februar 2007

PESOA is a cooperative project supported by the federal ministry of education and research (BMBF). Its aim is the design and prototypical implementation of a process family engineering platform and its application in the areas of e-business and telematics.
The project partners are:

DaimlerChrysler AG
Delta Software Technology GmbH
ehotel AG
Fraunhofer IESE
Hasso-Plattner-Institute
University of Leipzig

PESOA is coordinated by
Prof. Dr. Mathias Weske
Prof.-Dr.-Helmert-Str. 2-3
D-14482 Potsdam

www.pesoa.org

# Zusammenfassung

In der PESOA-Projektphase 5 wurden Aspekte der Qualitätssicherung für die Prozessfamilienentwicklung untersucht. Der Prozessfamilienentwickler soll durch die erarbeiteten Techniken bei der Qualitätssicherung für die Mitglieder einer Prozessfamilie explizit unterstützt werden. Dieser Bericht fasst die Arbeitsergebnisse des PESOA-Konsortiums in der Projektphase 5 zusammen.

# Inhalt

# 1 Einleitung

In der PESOA Projektphase 5 wurden Aspekte der Qualitätssicherung für die Prozessfamilienentwicklung untersucht. Der Prozessfamilienentwickler soll durch die erarbeitenden Techniken bei der Qualitätssicherung für die Mitglieder einer Prozessfamilie explizit unterstützt werden.

In Abschnitt 2.1 sind die Ergebnisse des Projektpartners Delta Software Technology zusammengefasst. Die Firma Delta beschäftigte sich in der Projektphase 5 mit der Qualitätsverbesserung durch Automatisierung. Es wurde dabei die Korrelation zwischen Automatisierung und der Qualität in den Werkzeugketten untersucht. Ziel von Delta war zudem eine Erhöhung des Automatisierungsgrads in den Werkzeugketten. Die Firma ehotel AG arbeitete in der Projektphase 5 am Entwurf einer erweiterten Modellkonfigurationsstrategie für die E-Business Domäne. Diese soll verstärkt Variationspunkte in User Interfaces berücksichtigen. Die Ergebnisse der Firma ehotel sind in Abschnitt 2.2 zusammengefasst. Mit dem Thema der Qualitätsverbesserung durch Korrektheitsüberprüfungen in variantenreichen Prozessen beschäftigte sich das Hasso-Plattner-Institut in der Projektphase 5. Hierbei wurde eine erweiterte formale Darstellung von Prozessmodellen zum Zweck der Korrektheitsprüfung entwickelt. Es wurde zudem eine prototypische Werkzeugkette entwickelt, die die Untersuchung der Korrektheit abgeleiteter Prozessmodellvarianten erlaubt. Eine Zusammenfassung der Ergebnisse des Hasso-Plattner-Instituts in der Projektphase 5 findet sich in Abschnitt 2.3. Die Projektphase 5 wurde von der Universität Leipzig genutzt, um Vorschläge zur Verbesserung einzelner Prozessschritte des PESOA Vorgehens auf Basis des Evaluationsberichts aus der Projektphase 4 zu erarbeiten. Die Ergebnisse sind in Abschnitt 2.4 zusammengefasst.

# 2 Ergebnisse

In diesem Kapitel werden die Arbeitsergebnisse der Projektpartner in der Projektphase 5 zusammengefasst.

## 2.1 Delta Software Technology GmbH

In den vorangegangenen Projektphasen wurden Werkzeugketten für die Anwendungsbereiche e-Business und Automotive realisiert. Im Rahmen des Arbeitspakets „Qualitätsverbesserung durch Automatisierung“ hat Delta Software Technology (Delta) die Möglichkeiten untersucht, in diesen Werkzeugketten durch zusätzliche Automatismen Qualitätsverbesserungen zu erreichen. Schließlich sind auf dieser Basis Anpassungen der Werkzeuge erfolgt, die anhand eines konkreten Beispiels validiert wurden.

### 2.1.1 Erfahrungen aus den Fallstudien

Den Ausgangspunkt für die Analysearbeiten bildeten die beiden Fallstudien „Magrathea“ [GGH06] und „Windshield Wiper“ [BFG06]. Die jeweils realisierten Werkzeugketten sind durch erhebliche Unterschiede gekennzeichnet: Sie sind in sehr unterschiedlichen Anwendungsbereichen (Magrathea: e-Business, Windshield Wiper: Automotive) angesiedelt, sie basieren auf verschiedenen Prozessmodellnotationen (Magrathea: BPMN [BPMN], Windshield Wiper: UML [UML]), und es sind unterschiedliche Generierungskonzepte implementiert worden (Magrathea: generischer Ansatz, Windshield Wiper: spezifischer Ansatz, vgl. dazu [GOB05, Kap. 3.2]).

Trotz dieser Unterschiede gab es in beiden Fallstudien wichtige gemeinsame Erfahrungswerte: Zum einen lag der Aufwand im Bereich des *Domain Engineering* deutlich über demjenigen für das *Application Engineering*. Dies entspricht der Intention von Produktlinien, die konkrete Applikation schneller und billiger zu produzieren, indem zuvor in eine entsprechende Infrastruktur investiert wird. Zum anderen stieg jedoch innerhalb des *Domain Engineering* der Aufwand von Phase zu Phase deutlich an, d.h. das Domänen-Design war aufwändiger als die Domänenanalyse, und die Domänen-Implementierung aufwändiger als das Domänen-Design. Somit gab es insbesondere für die Domänen-Implementierung den größten Bedarf an zusätzlicher Automatisierung.

### 2.1.2 Quality of Service

Neben der wirtschaftlichen Motivation, den Aufwand zur Erstellung einer Applikation zu minimieren, gibt es eine Reihe von applikationsbezogenen Gründen für einen höheren Automatisierungsgrad. Speziell im Bereich Automotive gibt es sehr spezifische Anforderungen an die Qualität der erzeugten Applikation.

**Performance.** Performance-Anforderungen sind vorwiegend motiviert durch Speicher-Beschränkungen (z.B. Scheibenwischer-Controller: 4KB SRAM). Hier führt eine domänenspezifische Codegenerierung generell zu besseren Ergebnissen als „all-in-one“-Lösungen wie z.B. MATLAB/Simulink oder UML-basierte Generatoren.

**Reliability.** Zuverlässigkeits-Anforderungen unterliegen in führenden Embedded-Bereichen, z.B. in der Medizin, aber auch im Automotive-Bereich, einer weit fortgeschrittenen Standardisierung. Ein Beispiel ist der Standard (A)SIL ((Automotive) Safety Integrity Level, IEC 61508), der ein Maß für die Zuverlässigkeit liefert. Hier ist eine Zertifizierung üblich. In Bezug auf das PESOA-Konzept ist zum einen festzustellen, dass Modellierung, die Wiederverwendung von Assets, und eine automatische Codeproduktion generell zu höherer Software-Qualität führen. Dies wird insbesondere an einer geringeren Zahl an Fehlern sichtbar, die dann, wenn sie auftreten, leichter zu erkennen sind. Zum anderen können einzelne Anforderungen eine semantische Ebene, d.h. die Domäne, betreffen. Ein Beispiel dafür ist, bezogen auf das Scheibenwischer-Beispiel aus [BFG06], ein Notfallprogramm, welches bei einem Ausfall der Verbindung zwischen dem zentralen Controller und dem Regensensor ein festes Wischintervall realisiert. Dies bedeutet, dass Zuverlässigkeits-Anforderungen, sofern sie variabel sind, zum Teil bereits in der Domänenanalyse berücksichtigt werden müssen. Es handelt sich dann nicht zwingend um Prozessvariabilität, aber um Problemraum-Variabilität im Sinne der generativen Programmierung [CE00].

**Real-time.** Speziell im Automotive-Bereich gibt es zahlreiche Echtzeitanforderungen. Insbesondere ist die Einhaltung von Zeitschranken wichtiger als eine allgemeine Schnelligkeit der Software [BEL04, Kap. 4.2.2]. Ein einfaches Beispiel ist ein Scheibenwischer-Regensensor: Es ist ausreichend, wenn das menschliche Auge die Antwortzeit als „sofort“ einstuft, was bei einer Bildfrequenz von mehr als 60 Hertz der Fall ist. Für die Sensor-Antwortzeit, einschließlich der zugehörigen Kontroll-Routine, resultiert daraus eine Zeitschranke von unter 1/60 Sekunde. Ein weiteres Beispiel ist eine Motorsteuerung mit festen Zeitpunkten für Zündung und Einspritzung. Innerhalb der PESOA- Produktlinienarchitektur können solche Anforderungen auf verschiedenen Ebenen berücksichtigt werden. Wenn die konkrete Echtzeitanforderung variabel ist, und somit generierter Code davon betroffen ist, kommt es darauf an, ob es sich um eine Lösungsraum-Variabilität oder eine Problemraum-Variabilität handelt. In letzterem Fall sind alle *Domain Engineering*-Phasen betroffen. Im Domänen-Design können Echtzeitanforderungen beispielsweise durch UML-Timing-Diagramme modelliert werden. Handelt es sich hingegen um eine reine Lösungsraum-Variabilität, so muss der in der Domänen-Implementierung entwickelte Generator dies berücksichtigen, und für seine Anwendung in der Applikations-Implementierung eine zusätzliche Konfigurationsmöglichkeit anbieten. Im Fall von HyperSenses™ würde dies durch ein Konfigurations-Rendering geschehen [GOB05, Kap. 3.3]. Ob eine konkrete Echtzeitanforderung im Problemraum oder im Lösungsraum anzusiedeln ist, hängt dabei vom Abstraktionsgrad der Domänenmodellierung ab.

Die aufgezählten Anforderungen sind typische „Quality of Service“ (QoS)-Anforderungen – ein Kontext in welchem das PESOA-Konzept ausführlich diskutiert wurde [BFL06]. Kennzeichnend ist auch bei diesen – Applikations- und Domänen-bezogenen – Anforderungen, dass in jedem Fall die Generierung der Software einen wesentlichen Beitrag zur Lösung liefert.

Insgesamt gibt es für die Phase Domänen-Implementierung den größten Bedarf für eine verbesserte Automatisierung.

### 2.1.3 Tool-Anpassungen

In der Domänen-Implementierung waren bereits in den genannten Fallstudien ein automatischer Import von Modelldaten sowie eine automatische Code-Produktion gegeben. Das Metamodell des Codegenerators wurde hingegen manuell erstellt. Hier ist grundsätzlich eine automatische Erzeugung von Metamodelldaten aus Artefakten der Domänenanalyse und des Domänen-Designs möglich. Viel versprechend ist dies vor allem für die Automotive-Werkzeugkette, da hier die betrachtete Domäne mit dem Beispiel „Windshield Wiper“ sehr eng gefasst wurde und ein domänenspezifisches Metamodell realisiert wurde. Für die e-Business-Werkzeugkette wurde hingegen ein generisches Metamodell (bezogen auf den definierten Ausschnitt der BPMN) realisiert, so dass dort diese Aufgabe entfällt.

In Bezug auf HyperSenses schuf Delta zunächst die technischen Voraussetzungen für die automatische Erzeugung von Metamodelldaten. Dies geschah zum einen durch Anpassungen in der Implementierung des „HyperSenses Model Interface“, einer Schnittstelle welche ursprünglich nur die Erzeugung von Modelldaten ermöglichte. Zum anderen wurde der Modell-Import so erweitert, dass er diese erweiterte Schnittstelle zum Import von Metadaten verwenden konnte. Sowohl der Modell-Import als auch das „HyperSenses Model Interface“ wurden in ANGIE implementiert, vgl. dazu [GGH06, Kap. 4.4.3].

### 2.1.4 MetaEdit+ und HyperSenses

Um den neuen Metamodell-Import zu testen, wurde eine Anbindung an das Werkzeug MetaEdit+ der Firma MetaCase implementiert. In der vorhandenen Automotive-Werkzeugkette hätte zwar das „Decision Model“ den benötigten Input für einen Metamodell-Import geliefert. Jedoch ließ sich anhand der MetaEdit-Anbindung außer dem Metamodell-Import auch die Konfigurierbarkeit und Neutralität des Modell-Imports überprüfen[1].

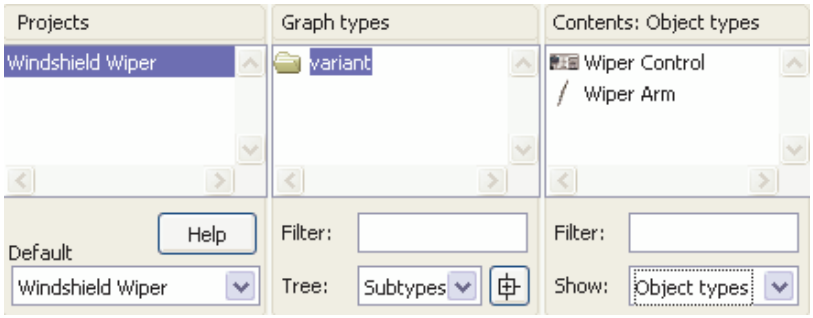


Abbildung 1: Metamodell-Browser in MetaEdit+

In MetaEdit+ wurden zunächst die Metadaten für die Scheibenwischersteuerung erfasst: Objekttypen, Attribute, Relationen, Bedingungen und grafische Symbole wurden definiert (Abbildung 1). Die von Delta verwendete Version 4.5 beta von MetaEdit+ bot die Möglichkeit eines XML-basierten Exports dieser Metadaten. Für dieses Format wurde der neue Metamodell-Import konfiguriert. Um diesen Import komfortabel aus der HyperSenses-Oberfläche aufzurufen, wurde ein entsprechendes „Tool-Skript“ auf Basis des „HyperSenses Tool Interface“ geschrieben [BFG06, Kap. 7.1]. Das im-

[1] Da der Projektpartner Fraunhofer IESE nicht mehr an Phase 5 teilnahm, war bzgl. der vorhandenen Werkzeugkette auch kein umfassender Support mehr gegeben.

portierte Metamodell entsprach nahezu vollständig dem zuvor definierten Metamodell aus der Automotive-Werkzeugkette. Geringe Abweichungen waren notwendig, da ein generisches Konzept zur Abbildung von MetaEdit-Metadaten auf HyperSenses-Metadaten definiert wurde. Daher war es möglich, die vorhandenen Code-Patterns unter minimalen Anpassungen zu übernehmen.

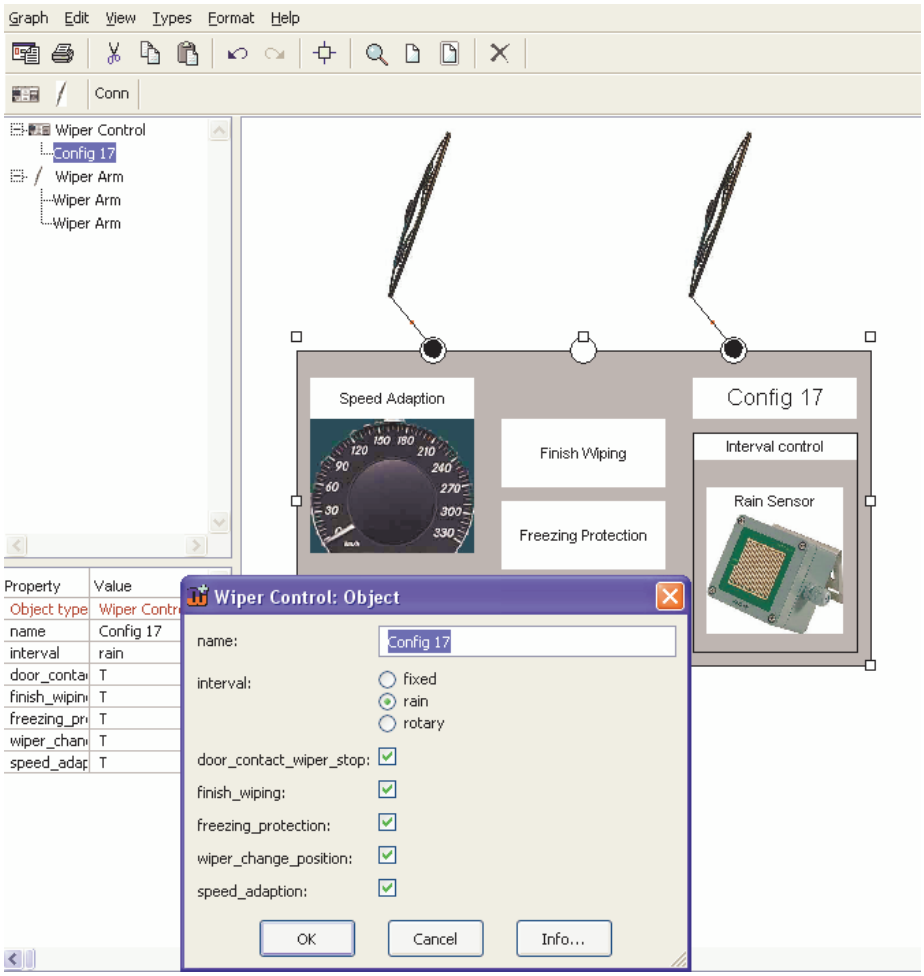


Abbildung 2: Modellierung einer Scheibenwischer-Variante mit MetaEdit+

Es wurden mehrere konkrete Beispiel-Varianten („Graphs“) in MetaEdit+ editiert (Abbildung 2) und in XML-Dateien exportiert. Der vorhandene Modell-Import wurde für das MetaEdit-spezifische Format und das zuvor importierte Metamodell konfiguriert. Letzteres geschah automatisch, als Resultat des zuvor durchgeführten Metamodell-Imports. Die Beispiel- Modelldaten wurden nach HyperSenses importiert, d.h. in HyperSenses-Modelle konvertiert (Abbildung 3).

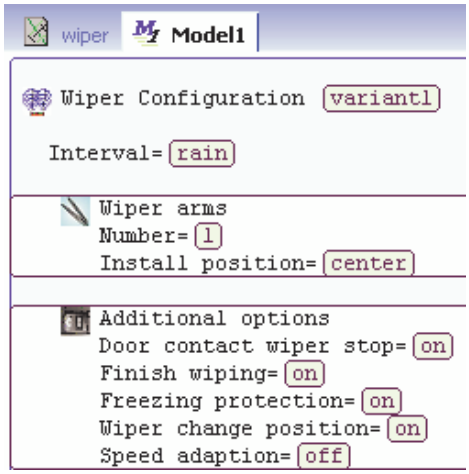


Abbildung 3: Importierte Modelldaten in HyperSenses

Die weiteren Schritte, die Code-Produktion mit Hilfe der „Produce Factory“, die Erstellung der ausführbaren Datei, sowie der Test auf dem Demonstrator, wurden wie in der Automotive-Fallstudie durchgeführt [BFG06, Kap. 7].

### 2.1.5 Zusammenfassung

Im PESOA-Prozess bietet die Phase Domänen-Implementierung das größte Potential für Verbesserungen durch einen höheren Automatisierungsgrad. Im Zentrum steht dabei die Entwicklung des domänenspezifischen Codegenerators. Die Realisierung der Werkzeuganbindung MetaEdit-HyperSenses durch zwei miteinander gekoppelte Import-Schritte zeigte, dass hier ein höherer Automatisierungsgrad machbar und praktikabel ist. Vor dem Hintergrund gerade domänenspezifischer Codegeneratoren, die für eine andere (verwandte) Domäne neu implementiert werden müssen, erscheint dies besonders vorteilhaft.

## 2.2 ehotel AG

Das prozessbasierte Produktlinienengineering hat zu einer besseren Strukturierung der aktuellen ehotel Softwareentwicklung beigetragen.

Erfahrungen in der Praxis haben allerdings gezeigt, dass sich die Variabilitätskonfiguration abweichend von der im PESOA Kontext fokussierten Bindung zur Build-time besser zu späteren Bindezeitpunkten durchführen lässt. Dies wird in der Projektphase 5 unter dem Ziel der Qualitätsverbesserung betrachtet.

### 2.2.1 Projektkontext

Ausgangspunkt innerhalb des PESOA Projekts ist die Fallstudie Magrathea. Der Softwaregenerator Magrathea generiert basierend auf einem Prozessmodell aus Implementierungsartefakten eine konkrete Implementierung der modellierten Applikation. Auch statische Konfigurationen, welche sich nicht im Prozessmodell widerspiegeln, werden in diesem Schritt vorgenommen. Die Beschreibung der Bindung von Variabilitäten umfasst sowohl den Zeitpunkt der Bindung als auch die Art der Realisierung der Bindung. Im Magrathea Ansatz handelt es sich um eine statische Build-time Bindung.

Die Erkenntnisse aus der PESOA Domänenanalyse flossen direkt in die Konzeption und Entwicklung der neuen ehotel-Buchungsplattform ein. Dies führte zu einer strukturierteren Systemarchitektur. Dies bedeutet in der Praxis eine generische Serverkomponente und ein umfangreiches Client Framework. Die Serverkomponente stützt sich auf den de-facto Brachenstandard der Open-Travel-Alliance, welcher vielfältigste Geschäftsprozesse abdeckt. Die Client Komponente setzt technisch auf dem GUI Framework Tapestry auf, das vorbildlich Modularisierung und Wiederverwendung unterstützt. Diese Komponenten werden innerhalb von Applikationsservern betrieben. Unter diesem Ansatz bietet es sich an, die Bindung der Komponenten auf den spätestmöglichen Zeitpunkt zu verlagern.

### 2.2.2 Zielsetzung

Im Rahmen des Process Family Engineerings kann die Variabilität in den Prozessen zu verschiedenen Zeitpunkten aufgelöst werden. Eine nachgelagerte Bindung kann während der Aktivierung bzw. zur Laufzeit erfolgen. Dies bedeutet, dass die Software entweder nur beim ersten Start oder bei jeder Startphase auf die spezifischen Anforderungen konfiguriert wird.

Motiviert aus Praxisanforderungen ist das Ziel die Bindung variabler zu gestalten und den Bindezeitpunkt zur Startzeit bzw. Laufzeit vorzunehmen. Dies vermeidet Code-Redundanz und kommt einem Framework Ansatz entgegen. Das Ziel ist eine lose Kopplung um Modularisierung, Testbarkeit und Skalierbarkeit zu erhöhen. In der Praxis bedeutet dies eine Auslagerung der Konfigurationsentscheidungen (Komponenten Bindung) an ein externes Framework.

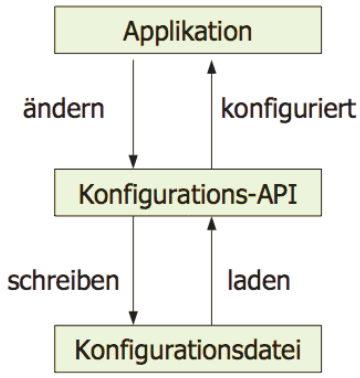


Abbildung 4: Konfiguration zur Start- oder Laufzeit mit Hilfe einer externen Konfigurationsdatei

### 2.2.3 Konzepte

**Dependency Injection**. Dependency Injection ist ein Entwurfsmuster und dient in einem objektorientierten System dazu, die Abhängigkeiten zwischen Komponenten oder Objekten zu minimieren. Dependency Injection ist eine Anwendung des Prinzips der Inversion of Control (IoC), bezieht sich aber nur auf die Erzeugung und Initialisierung von Objekten. Sie kann als Verallgemeinerung des Fabrikmusters verstanden werden. Die Funktionalität bleibt trotz dieser Kontrollumkehr als Einfügung enthalten. Dadurch ist es einfach möglich, Abhängigkeiten zu erkennen.

Der Begriff **Inversion of Control** (IoC) bezeichnet ein Umsetzungsparadigma, das in der Objektorientieren Programmierung Anwendung findet. Dieses Paradigma beschreibt die Arbeitsweise von Frameworks: eine Funktion eines Anwendungsprogramms wird bei einer Standardbibliothek registriert, und von dieser zu einem späteren Zeitpunkt aufgerufen. Statt dass die Anwendung den Kontrollfluss steuert und lediglich Standardfunktionen benutzt, wird die Kontrolle über die Ausführung bestimmter Unterprogramme an das Framework abgegeben.

In einem klassisch aufgebauten OO-System ist jedes Objekt selbst dafür zuständig, seine Abhängigkeiten, also benötigte Objekte und Ressourcen, zu erzeugen und zu verwalten. Dafür muss jedes Objekt einige Kenntnisse seiner Umgebung mitbringen, die es zur Erfüllung seiner eigentlichen Aufgabe eigentlich nicht benötigen würde. Insbesondere muss es, um die entsprechenden Objekte erzeugen zu können, ihre konkrete Implementierung kennen. Dependency Injection überträgt die Verantwortung für das Erzeugen und die Verknüpfung von Objekten an ein extern konfigurierbares Framework, entsprechend einem Komponentenmodell. Dadurch wird der Code des Objektes unabhängig von seiner Umgebung und von der konkreten Umsetzung der Klassen, die es benötigt. Das vermeidet unnötige Abhängigkeiten beim Kompilieren und erleichtert die Wartung und das Testen.

### 2.2.4 Verwertung

Im Kontext der ehotel Hotelbuchungs-Applikation kommen die Konzepte durch das HiveMind Framework zum Einsatz.

**Apache Hivemind**. Hivemind ist ein leichtgewichtiger Kompositionsmanager entstanden aus der Entwicklung des Web-Application Frameworks Tapestry. Hivemind bietet im Wesentlichen drei Funktionen an:

- die Komposition von komplexen Komponenten mittels Dependency Injection,
- die Verwaltung von Konfigurationsdaten zur internen Konfiguration von Komponenten und
- das Abfangen von Methodenaufrufen an Komponenten und das Einfügen von Querschnittsfunktionen in Form von allgemeinen Proxy-Objekten

Hivemind-Komponenten (Services) werden direkt über Java-Interfaces beschrieben, Java-Klassen implementieren als Hivemind-Komponenten diese Schnittstellen. Eine externe Konfiguration in XML-Dateien beschreibt die Verknüpfung von Implementierungen mit diesen Service-Schnittstellen. Abhängigkeiten löst Hivemind soweit möglich automatisch auf, es ist aber auch möglich, Abhängigkeiten explizit zu beschreiben. Um Abhängigkeiten zu erfüllen, können Komponenten ebenso wie sonstige externe Abhängigkeiten aus einem globalen Archiv geladen werden. Dabei zeichnet sich Hivemind durch eine gute Fehlerbehandlung aus. Fehler in der Konfiguration werden früh erkannt und mit aussagekräftigen Fehlermeldungen versehen. Dabei zeigten sich drei wesentliche Vorteile durch den Einsatz eines Kompositionsmanagers:

- Einsparung von Code

- Höhere Übersichtlichkeit des Kompositionscodes
- Größere Flexibilität durch Verlagerung von Konfigurationsentscheidungen in externe Konfigurationsdateien

### 2.2.5 Zusammenfassung

Dieses vom Magrathea Projekt abweichende Bindezeitmodell könnte eine alternative Strategie und in der ehotel Praxis eine klare Qualitätsverbesserung sein. Die Komponenten (Code-Fragmente) werden nicht mehr direkt per Generator-Software verbunden, sondern der Generator erzeugt aus einem Prozessmodell ein Set von Konfigurationsdateien. Dadurch wird eine weitere Abstraktion erreicht, eine Entkopplung der Komponenten und somit wird die Code Redundanz vermindert. Die damit erreichte Wiederverwendbarkeit der Komponenten und die flexible Konfiguration sind bei einem Produkt welches ständig weiter entwickelt wird, unverzichtbar.

## 2.3 Hasso-Plattner-Institut für Softwaresystemtechnik

In der Phase 5 des PESOA-Projekts wurde am HPI eine Werkzeugkette zur Korrektheitsüberprüfung konfigurierter Prozessmodelle entwickelt. Darüber hinaus wurden zuvor die konzeptionellen Grundlagen für diese Werkzeugkette geschaffen.

### 2.3.1 Soundness-Eigenschaft

Ein wesentliches Kriterium für den Einsatz einer Prozessfamilienentwicklung ist die hohe Qualität der erstellten Produkte. Durch den hohen Anteil von wieder verwertbaren Teilen, Core Assets, tritt eine geringe Redundanz auf. Die Qualitätssicherung muss sich somit nicht auf eine Vielzahl von ähnlichen, mit redundanten Artefakten ausgestatten Produkten befassen, sondern kann ihren Fokus auf die Qualität der Einzelkomponenten legen.

Im Bereich der Prozessfamilienentwicklung kommt jedoch ein weiteres Kriterium hinzu. Es müssen nicht nur die einzelnen Aktivitäten korrekt funktionieren, sondern auch das Zusammenspiel zwischen diesen. Im Besonderen soll ein konfiguriertes Prozessmodell keine Deadlocks, d.h. fehlerhafte Verknüpfungen zwischen Aktivitäten, enthalten. In [SP05] haben wir einen Ansatz zur statischen Prüfung von Deadlock-Freiheit für UML-Aktivitätsdiagramme vorgestellt. Dieser Ansatz basiert auf der Annahme, dass es Algorithmen zur Deadlock-Prüfung von Prozessmodellen auf Basis der Soundness-Definition von van der Aalst gibt [AHH94].

Wie sich jedoch während der Durchführung des PESOA-Projektes gezeigt hat, ist die Soundness-Definition nur bedingt für die im Projekt verwendeten Prozessmodelle verwendbar. Dies hat zwei wesentliche Gründe. Zum einen ist Soundness auf Basis von Workflow-Netzen [AH02] definiert. Zwar gibt es für UML-Aktivitätsdiagramme Abbildungen auf Workflow-Netze. Diese beziehen sich jedoch auf alte UML-Versionen [EsW01] oder bilden nur eine Teilmenge ab [Stö04], welche für den Einsatz im Projekt nicht ausreichend ist. Für den Bereich E-Business ist keine Abbildung von BPMN auf Workflow-Netze verfügbar. Zum zweiten unterstützt die Soundness-Definition nach van der Aalst keine verzögerten, engl. lazy, Aktivitäten. Verzögerte Aktivitäten treten immer dann auf, wenn die Rückgabe des Ergebnisses eines Geschäftsprozesses nicht mit dessen Terminierung übereintrifft. Gerade im Bereich E-Business ist eine Unterstützung für verzöger-

te Aktivitäten jedoch wichtig, da viele Buchhaltungstätigkeiten erst nach Lieferung an den Kunden, d.h. aus Sicht des Kunden hat der Geschäftsprozess sein Ziel erfüllt, möglich sind.

Um die oben genannten Nachteile der Soundness-Definition für Workflow-Netze zu umgehen, führen wir eine abgeschwächte Soundness-Variante ein. Im Rahmen dieses Berichtes beschränken wir uns auf eine informelle Darstellung, die technischen Details sind in [PW06] veröffentlicht. Für die Prüfung auf Deadlock-Freiheit eines konfigurierten Prozessmodells ist eine gewisse formale Struktur unabdingbar. Dieses ist durch folgende Definition gegeben:

*Ein Prozessmodell ist strukturell-sound, wenn es genau eine initiale und genau eine finale Aktivität besitzt. Alle anderen Aktivitäten müssen auf Pfaden zwischen diesen Aktivitäten liegen.*

Die initiale Aktivität stellt den Beginn des Prozesses dar und die finale Aktivität liefert das Ergebnis für den Kunden zurück. Die Einschränkung auf eine finale Aktivität anstelle einer Teilung in Ergebnislieferung und Terminierung ist nicht erforderlich, da dies durch gewisse Prozessmuster [AHKB03] ebenso auf der definierten Struktur erreicht werden kann. Basierend auf Struktureller-Soundness kann jetzt definiert werden, wann ein Prozessmodell Deadlock-frei ist:

*Ein Geschäftsprozess, welcher strukturell-sound ist, wird als lazy-sound bezeichnet, wenn er in jedem möglichen Ausführungsfall exakt ein Ergebnis zurückliefert.*

Durch die oben genannte Definition werden Ausführungspfade ausgeschlossen welche zu Deadlocks führen, da in diesem Fall kein Ergebnis zurückgeliefert wird. Eine formale Umsetzung von Lazy-Soundness wird mittels Bisimulationstechniken [Mil89] aus dem Bereich der Prozessalgebren ermöglicht. Dabei werden Geschäftsprozesse als mathematische Modelle von Graphen interpretiert, wobei jede Aktivität (inkl. Entscheidungen und Verknüpfungen) einem Prozessmuster zugeordnet wird. Basierend auf einer Formalisierung der Prozessmuster, welche im Rahmen des PESOA Projektes entwickelt wurde [PW05], werden die so genannten Prozessgraphen in formale Terme einer Prozessalgebra überführt. Die erzeugten Terme können mittels Bisimulation von Invarianten überprüft werden, welche formal die Beschreibung von Lazy-Soundness repräsentieren.

### 2.3.2 Technische Umsetzung

Die technische Umsetzung der Prüfung von konfigurierten Prozessmodellen auf Deadlock-Freiheit wird durch eine Werkzeugkette realisiert. Aufgrund der zeitlichen Rahmenbedingungen erfolgte dabei eine Beschränkung auf die E-Business Domäne. Die konzeptionelle Gestaltung ist jedoch so ausgelegt, dass eine Anpassung an andere Domänen mit geringem Aufwand möglich ist.

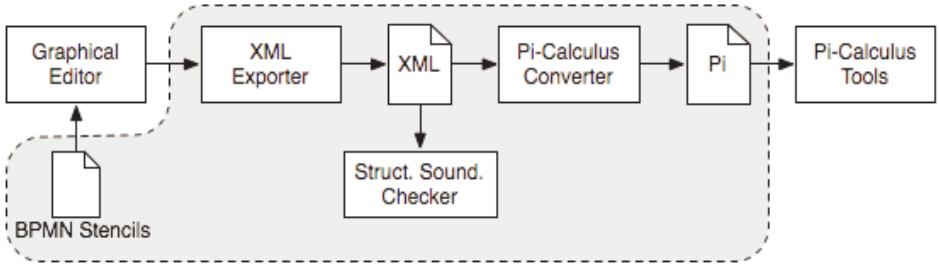


Abbildung 5: Werkzeugkette zur Korrektheitsprüfung konfigurierter Prozessmodelle

Abbildung 5 zeigt die Abhängigkeiten zwischen den Werkzeugen und Dokumenten in der Werkzeugkette. Werkzeuge und Skripte sind als Rechtecke und Dokumente als Notizen dargestellt. Die im Rahmen des PESOA-Projektes erstellten Komponenten sind dabei im hinterlegten Bereich dargestellt.

Als Grundlage zur Analyse der Deadlock-Freiheit wird eine XML-Darstellung des Prozessgraphen eines konfigurierten Prozesses verwendet. Die XML-Darstellung kann durch verschiedene Werkzeuge erstellt werden. In Abbildung 5 wird dabei ein grafischer Editor verwendet, wie er auch im Rahmen des PESOA-Projektes entwickelt wurde. Dieser Editor wird mit speziellen Schablonen für die jeweilige Prozessmodellierungssprache ausgestattet, in diesem Falle für BPMN. Die XML-Darstellung eines Prozesses kann durch den Structural Soundness Checker auf strukturelle Korrektheit überprüft werden. Nach diesem Schritt kann die XML-Darstellung in Terme einer Prozessalgebra überführt werden, wir verwenden den Pi-Calculus [SaW03]. Die dabei entstehende ASCII-Darstellung der Prozesse kann in existierende Bisimulations-Checker für den Pi-Calculus eingegeben werden. Mit Hilfe dieser kann Lazy Soundness für das ursprüngliche Prozessmodell entschieden werden. Eine ausführliche technische Beschreibung wurde in [Puh06] veröffentlicht.

### 2.3.3 Beispiel

Zur Illustration der Überprüfung eines Prozessmodells greifen wir auf das in den PESOA Berichten 8 und 17 vorgestellte Prozessmodell eines E-Business-Shops zurück. Eine konfigurierte Variante ist in Abbildung 6 dargestellt. Die entsprechende Beschreibung findet sich in den entsprechenden Berichten.

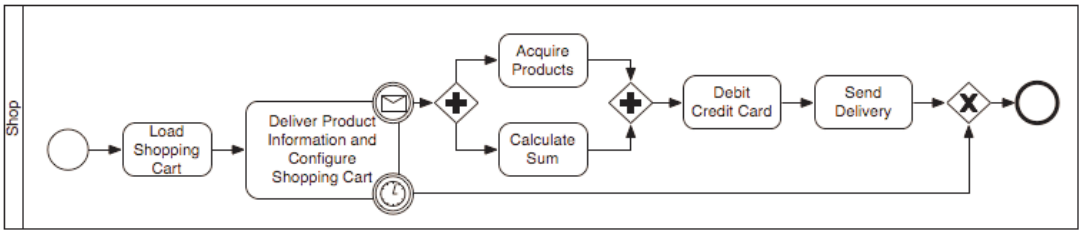


Abbildung 6: Beispiel für einen konfigurierten Prozess

Abbildung 6 zeigt das konfigurierte Prozessmodell des E-Business-Shops. Der Sub-Process Checkout wurde expandiert dargestellt (Acquire Pro-

ducts, Calculate Sum, Debit Credit Card). In einem ersten Schritt wird das grafische Modell nun in eine XML-Darstellung überführt:

```
<!-- Model exported by OmniGraffle BPMN Exporter (C) 2006 Frank
Puhlmann -->
<model>
   <process id="1" type="BPMN">
      <node id="737" type="XOR Gateway"/>
      <node id="570" type="Timer Intermediate Event" scope="919"/>
      <node    id="544"    type="Message    Intermediate    Event"
scope="919"/>
      <node id="538" type="End Event"/>
      <node id="928" type="Task" name="Send Delivery"/>
      <node id="923" type="AND Gateway"/>
      <node id="922" type="Task" name="Debit Credit Card"/>
      <node id="921" type="Task" name="Calculate Sum"/>
      <node id="920" type="Task" name="Acquire Products"/>
      <node id="717" type="AND Gateway"/>
      <node id="919" type="Task" name="Deliver Product Information
and Configure Shopping Cart"/>
      <node id="677" type="Task" name="Load Shopping Cart"/>
      <node id="534" type="Start Event"/>
      <flow id="942" type="Sequence Flow" from="737" to="538"/>
      <flow id="941" type="Sequence Flow" from="570" to="737"/>
      <flow id="940" type="Sequence Flow" from="928" to="737"/>
      <flow id="939" type="Sequence Flow" from="922" to="928"/>
      <flow id="938" type="Sequence Flow" from="923" to="922"/>
      <flow id="937" type="Sequence Flow" from="921" to="923"/>
      <flow id="936" type="Sequence Flow" from="717" to="921"/>
      <flow id="935" type="Sequence Flow" from="920" to="923"/>
      <flow id="934" type="Sequence Flow" from="717" to="920"/>
      <flow id="933" type="Sequence Flow" from="544" to="717"/>
      <flow id="932" type="Sequence Flow" from="677" to="919"/>
      <flow id="671" type="Sequence Flow" from="534" to="677"/>
   </process>
</model>
```

Das Root-Element der XML-Darstellung is <model>, welches verschiedene Prozessgraphen enthalten kann. Ein einzelner Prozess wird dabei durch das Element <process> gekennzeichnet. Die mathematische Struktur eines Prozesses setzt sich aus Knoten und Kanten zusammen, was durch die Elemente <node> und <flow> repräsentiert wird. Der Typ eines jeden Knoten ist lediglich als Text gegeben, um möglichst unabhängig von bestimmten Notationen zu bleiben. Eine Interpretation des Typs zu einem Prozessmuster erfolgt erst in der späteren Formalisierung. Basierend auf der XML-Darstellung kann bereits gezeigt werden, dass der konfigurierte Prozess strukturell-sound ist. Dazu dient das Ruby-Skript `struct_sound`:

```
localhost$ ruby struct_sound.rb omni-export.xml
Structural Soundness Checker for Business Process Diagrams (May,
24th 2006)
(c) 2006 Frank Puhlmann, puhlmann@hpi.uni-potsdam.de
Processing input file omni-export.xml
OK: The BPD has exactly one Start Event.
OK: The BPD has exactly one End Event.
Node with scope event found!
Node with scope event found!
OK: All nodes of the BPD are reachable from the Start Event.
OK: The End Event is reachable from all nodes of the BPD.
OK: The given BPD is structural sound!
```

Im Anschluss kann die XML-Darstellung in formale Repräsentation konvertiert werden. Dabei werden die Typen der Knoten sowie deren Abhängigkeiten interpretiert und entsprechende formale Terme erzeugt.

```
agent       N737(e941,e940,e942)=(e941.N737_1(e941,e940,e942)       +
e940.N737_1(e941,e940,e942))
agent N737_1(e941,e940,e942)=t.('e942.0 | N737(e941,e940,e942))
agent           N923(e937,e935,e938)=e937.e935.t.('e938.0           |
N923(e937,e935,e938))
agent N922(e938,e939)=e938.t.('e939.0 | N922(e938,e939))
```

```
agent N921(e936,e937)=e936.t.('e937.0 | N921(e936,e937))
agent N920(e934,e935)=e934.t.('e935.0 | N920(e934,e935))
agent    N717(e933,e936,e934)=e933.t.('e936.0    |    'e934.0    |
N717(e933,e936,e934))
agent
N919(e932,e941,e933)=(^check,true,false)e932.(N919_1(e932,e941,e933
,check,true,false) | N919_2(e932,e941,e933,check,true,false))
agent  N919_1(e932,e941,e933,check,true,false)='e941.'check<false>.0
+ 'e933.'check<false>.0 + 'check<true>.0
agent         N919_2(e932,e941,e933,check,true,false)=t.(0         |
N919(e932,e941,e933))
agent N677(e671,e932)=e671.t.('e932.0 | N677(e671,e932))
agent N534(e671,i)=i.t.'e671.0
agent N538(e942,o)=e942.t.'o.N538(e942,o)
agent N928(e939,e940)=e939.t.('e940.0 | N928(e939,e940))
agent
N(i,o)=(^e942,e941,e940,e939,e938,e937,e936,e935,e934,e933,e932,e67
1,true,false)(N737(e941,e940,e942)    |    N923(e937,e935,e938)    |
N922(e938,e939)    |    N921(e936,e937)    |    N920(e934,e935)    |
N717(e933,e936,e934)  |  N919(e932,e941,e933)  |  N677(e671,e932)  |
N534(e671,i) | N538(e942,o) | N928(e939,e940))
agent S_LAZY(i,o)=i.t.'o.0
```

Jeder Term der Darstellung wird durch `agent` eingeleitet. Der Hauptterm, welcher den Prozess repräsentiert, ist gegeben durch `N(i,o)`. Die Invarianten werden durch den Term `S_LAZY` gegeben. Mittels der formalen Darstellung kann entschieden werden, ob der formalisierte Prozess die gegebenen Invarianten erfüllt. Dies geschieht durch das existierende Tool Mobility Workbench [ViM94]:

```
The Mobility Workbench
 (MWB'99, version 4.136, built Fri Jun 30 10:59:18 2006)

MWB>input "/Users/frank/Documents/Diss/tools/agents.mwb"
MWB>weq N(i,o) S_LAZY(i,o)
The two agents are equal.
Bisimulation relation size = 56.
```

Da beide Terme äquivalent sind, ist der konfigurierte Prozess lazy-sound.

## 2.4 Universität Leipzig

Im Rahmen der Qualitätsverbesserung hinsichtlich des PESOA–Forschungsprojektes bezieht sich die Universität Leipzig im Großen auf den bereits veröffentlichten Evaluationsbericht [WeH06]. In diesem wurden in den jeweiligen Ergebnissen bereits verschiedene Vorschläge zur Verbesserung einzelner Prozessschritte im PESOA-Vorgehen genannt. Es folgt eine kurze Zusammenfassung der betrachteten PESOA-Prozessschritte sowie deren Ergebnisse aus der Evaluation [WeH06].

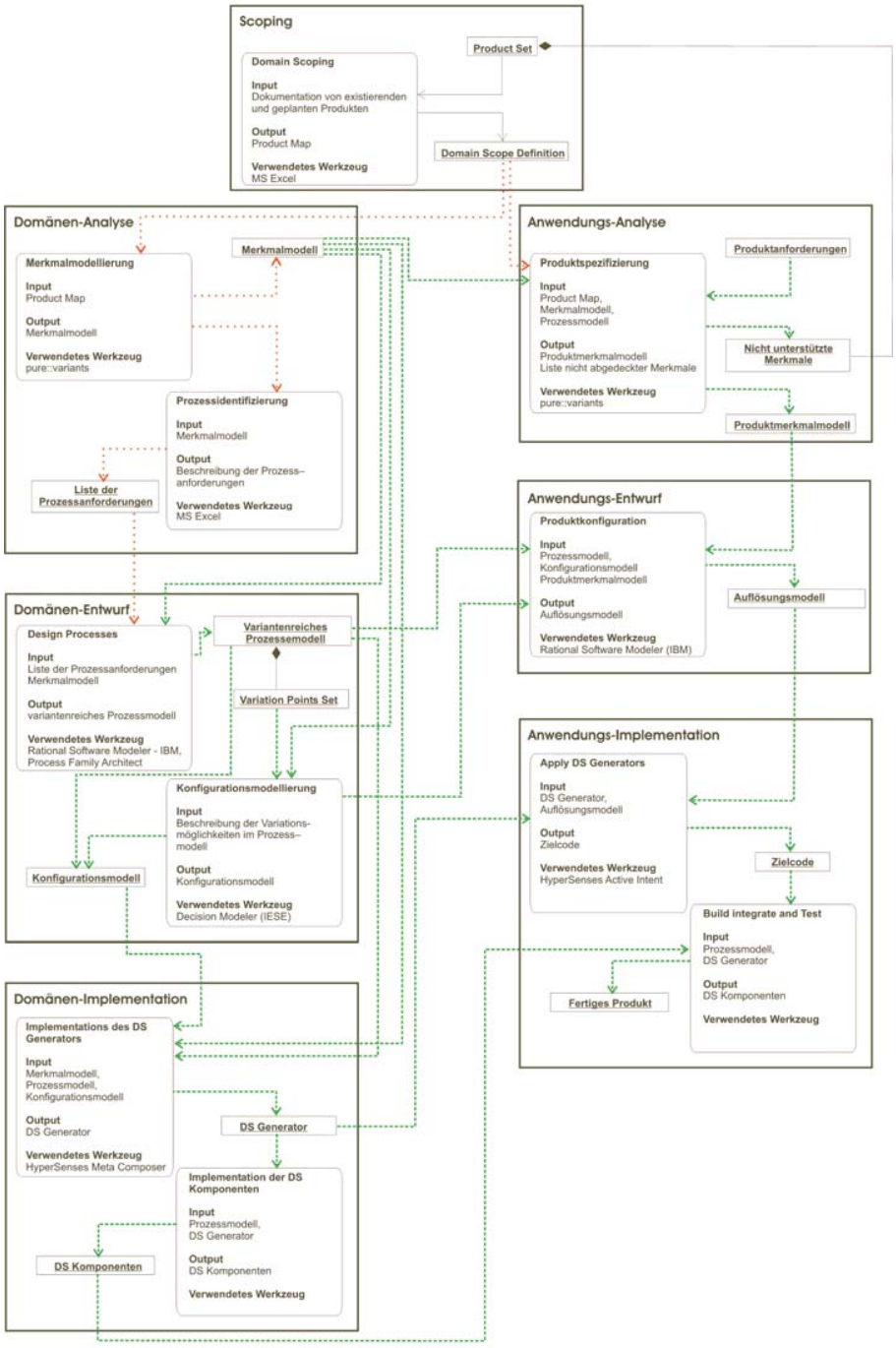


Abbildung 7: PESOA - Prozess mit eingesetzten Werkzeugen und farblicher Markierung von Qualitätsengpässen [WeH06]

### 2.4.1 Zusammenfassung der Ergebnisse der Evaluation anhand der PESOA- Prozessschritte

**Scoping**. Im Bereich des Scoping war MS Excel ausreichend, da es sich lediglich um die Darstellung der aus dem Product Line Mapping gewonnen Produkt- und Domainbeschreibungen handelte. Somit konnte mit Hilfe des PuLSE Eco Ansatzes die Identifizierung und Spezifizierung des Umfangs der Prozessfamilie, in diesem Fall des Scheibenwischersystems, gut durchgeführt und dargestellt werden.

**Domänenanalyse**. Merkmalmodelle und Event Lists verdeutlichten eine ganze Reihe von, für die Produktfamilie wesentlichen, Sachverhalten. Das daraus resultierende Wissen, welche Merkmale die Produktfamilie zur Verfügung stellt und welche Merkmale kundenspezifisch entwickelt werden müssen, hilft eine bessere Abschätzung bezüglich Zeit und Entwicklungskosten durchzuführen. Das deckt wiederum die Ziele der strategischen Unternehmensebene und dient der besseren Entscheidungsfindung des Kunden. Nachteilig ist, dass sämtliche im Scoping erarbeiteten Daten per Hand in das Merkmalmodell übernommen werden müssen.

**Anwendungsanalyse**. Genau wie im Prozessschritt der Domänenanalyse, werden auch an dieser Stelle die im Scoping erarbeiteten Daten benötigt. Da diese bereits in der Domänenanalyse im Merkmalmodell eingepflegt wurden und kein weiteres Werkzeug außer das bereits in der Domänenanalyse verwendete pure::variants verwendet wird, stellt das an dieser Stelle kein Problem dar und bedingt keine Verbesserungsvorschläge. Die unterschiedlichen Modelle und Sichtweisen von pure::variants lassen aus den variantenreichen Modellen konkrete Produkte ableiten. Zudem ist es grafisch darstellbar, welche Prozessvariationen harmonieren und welche sich gegenseitig ausschließen. Das Werkzeug erreicht folglich alle durch die Stakeholder gesetzten Ziele und wird somit den Anforderungen gerecht. Der in Abbildung 7 dargestellte kritische Übergang vom Scoping zur Anwendungsanalyse besteht demzufolge nur indirekt.

**Domänenentwurf**. Die beschriebenen Ziele wurden bis auf die Untersuchung des Einflusses der Variabilitätsmodellierung auf die strukturellen Korrektheitseigenschaften der Prozessmodellvarianten erfüllt. Grund dafür ist wieder die Datenvorlage im MS Excel Format, welche verantwortlich dafür ist, dass die Daten für das Prozessmodell momentan noch ausschließlich aus dem Merkmalmodell entnommen werden.

**Anwendungsentwurf**. Ein Werkzeug, das zur Konfiguration eines Produktes verwendet werden soll, muss sich nahtlos in den gesamten PESOA-Prozess und die entsprechende Werkzeugkette eingliedern lassen. Aufgrund der vorhandenen Schnittstellen des „Configure Product“–Werkzeuges zu den PESOA-Phasen „Specify Product“ (Anwendungs-Analysis), „Design Processes“ und „Model Configurations“ (Domänen-Entwurf) sowie „Apply DS Generator“ (Anwendungs-Implementation), gilt diese Integrationsnotwendigkeit insbesondere für Werkzeuge, die die genannten Phasen unterstützen.

**Domänenimplementierung**. Die HyperSenses-Werkzeuge wurden in erheblichem Umfang im Rahmen des PESOA Forschungsprojektes weiterentwickelt und den Bedürfnissen angepasst. Es war möglich den Generator und die Komponenten werkzeugunterstützt zu erstellen. Aus Sicht des Herstellers Delta Soft, strebt man lediglich nach einem weiteren Automatisierungsgrad. Sozusagen besteht der aktuelle Handlungsbedarf darin, die technischen Voraussetzungen eines Metamodell-Imports zu schaffen. Zudem befasst man sich mit der Möglichkeit, die HyperSenses-Tools in die Eclipse-Umgebung einzubinden. Zukünftig verspricht man sich ein besseres Zusammenspiel zwischen der Domänen- und Applikations-Implementation, da diese bislang zu wenig im PESOA-Prozess berücksichtigt wurden.

**Anwendungsimplementierung**. Das konfigurierte Produkt konnte werkzeuggestützt interaktiv erzeugt werden, was den Kern der Ziele der Stake-

holder im Wesentlichen trifft. Wünschenswert ist hier etwas mehr Komfort, insbesondere durch eine Integration in die durch den Rational Software Modeler gegebene Werkzeugumgebung.

**Gesamtergebnis**. Wie der Evaluation [WeH06] zu entnehmen ist, werden die Anforderungen bereits größtenteils abgedeckt. Betrachtet man die einzelnen Ergebnisunterpunkte, äußert sich in erster Linie der Wunsch, die Plattform als Ganzes zu integrieren um so zwischen den verwendeten Werkzeugen bessere Schnittstellen für die Datenweitergabe zu schaffen. Im Folgenden soll in einer Tabelle aufgelistet werden, an welchen Stellen Qualitätsverbesserungen angestrebt werden sollten.

| **PESOA - Prozess-schritt** | **Ziel / Anforderung** | **Erfüllungsgrad** |
|---|---|---|
| **Scoping** | Übersichtliche Darstellungsmöglichkeit | erfüllt |
| | Verwaltung der Produkteigenschaften | erfüllt (mit Einschränkungen) |
| | Datenexport der Produkteigenschaften | nicht erfüllt |
| **Domänen-Analyse** | Datenimport der Daten aus dem Product Map | nicht erfüllt |
| | Management der Variabilitäten | erfüllt |
| | Darstellbarkeit der gesamten Produktlinie einschließlich invarianter Bestandteile und Erweiterungspunkten | erfüllt |
| | Listenerstellung über die in den identifizierten Produkten ablaufenden Prozessen | erfüllt |
| **Anwendungs-Analyse** | Ableitungsmöglichkeit eines konkreten Produktes aus dem Merkmalmodell | erfüllt |
| **Domänen-Entwurf** | Prozessmodellierung auf Basis der Prozessliste und des Merkmalmodels | nicht erfüllt |
| | Modellierung des Kernprozesses einschließlich seiner Variationspunkte | erfüllt |
| | Aussagemöglichkeit über die Auswirkung der Auswahl von Eigenschaften | erfüllt |
| **Anwendungs-Entwurf** | Auflösung der Variationspunkte entsprechend der Produkteigenschaften zu einem gültigen Auflösungsmodell | erfüllt |
| **Domänen-Implementation** | Implementation eines domänenspezifischen Generators, der zur Erzeugung von Anwendungen in der Anwendungs-Implementation herangezogen werden kann | erfüllt |
| **Anwendungs-Implementation** | automatische Erzeugung eines fertigen Produktes | erfüllt |

**Empfehlung**. Aufgrund der Vielzahl an Werkzeugen besteht noch Handlungsbedarf. Sinnvoll wäre die Entwicklung einer integrierten Plattform, welche die Funktionalitäten der einzelnen Werkzeuge vereint, um die PESOA–Vorgehensprozesse optimal unterstützen zu können. Die Vereinigung der Logik bzw. der Funktionalität der einzelnen Werkzeuge in eine Plattform würde zu dem eine Verbesserung des Zeit- und damit auch des Kostenfaktor darstellen.

### 2.4.2 Qualitätsverbesserung

Verbesserung der Qualität wird dann erreicht, wenn Möglichkeiten geschaffen werden, die die an einen Sachverhalt gesetzten Erwartungen, verbessern. Im Allgemeinen betrifft das diejenigen Erwartungen, die Arbeitsschritte effizienter, schneller, besser und billiger werden lassen ohne dabei die Güte des Produktes zu verschlechtern, sondern vielmehr zu verbessern. Die bisher eingesetzten Werkzeuge decken zwar den gesamten Bedarf um fertige Anwendungen zu erzeugen, sie sind aber nicht in allen Schnittstellen miteinander kompatibel.

**Scoping – Domänenanalyse**. Betrachtet man die Schnittstelle vom Scoping zur Domänenanalyse wird deutlich, dass die im Product Map stehenden Daten alle im MS Excel Format .xls vorliegen. Diese müssen per Hand in das pure::variants Merkmalmodel eingepflegt werden. Dieser Übergang kostet unnötige Zeit und verursacht somit unnötige Kosten, er beinhaltet auch die Gefahr von Fehlern in der Überführung. Laut pure::systems besteht die Möglichkeit des Imports über Dateien im CSV Format. Sinnvoll wäre daher die Einbeziehung von Anforderungswerkzeugen, die es ermöglichen Produkteigenschaften komfortabel zu pflegen und im Repository Beziehungen zwischen den identifizierten Stakeholder-Zielen, den definierten Produktvarianten, den zugehörigen Produkteigenschaften und den domänenspezifischen und anwendungsfallspezifischen Entwicklungsartefakten herstellen zu können. Dieses wäre für die Traceability vorteilhaft. Als Vorschlage für ein solches Anforderungswerkzeug wäre das „Open Source Requirements Management Tool“ kurz [OSRMT] zu nennen. Dieses könnte in die PESOA-Plattform (Werkzeuge) integriert werden. Allerdings unterstützt dieses Werkzeug zum aktuellen Stand in der Version 1.3 nur die Ausgabe in XML.

**Domänenanalyse – Domänenentwurf**. Zum einen sollte der Datenimport der Prozessanforderungsliste automatisiert werden und zum anderen besteht noch in der Untersuchung des Einflusses der Variabilitätsmodellierung auf die Korrektheitseigenschaften der variantenreichen Prozessmodelle Handlungsbedarf. Es muss sichergestellt werden, dass bei der Konfiguration variantenreicher Prozessmodelle lediglich syntaktisch und strukturell korrekte Prozessmodellvarianten abgeleitet werden können.

Für die Verwaltung der Prozessanforderungen kann als Requirement Management Tool ebenfalls das oben vorgestellte OSRMT verwendet werden.

# 3 Literatur


[AH02] Wil van der Aalst and Kees van Hee. Workflow Management. MIT Press, 2002.

[AHH94] W. M. P. van der Aalst, K.M. van Hee, and G. J. Houben. Modeling and Analysing Workflow using a Petri-net based Approach. In G. De Michelis, C. Ellis, and G. Memmi, editors, Proceedings of the second Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms, pages 31–50, 1994.

[AHKB03] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow Patterns. Distributed and Parallel Databases, 14(1):5–51, 2003.

[BEL04] J. Bayer, M. Eisenbarth, T. Lehner, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland. Domain Engineering Techniques and Process Modeling. PESOA-Fachbericht Nr. 09/2004.

[BFG06] J. Bayer, T. Forster, C. Giese, T. Lehner, M. Schaude, A. Schnieders, P. Sommer, J. Weiland. Process Family Engineering in the Automotive Domain. PESOA-Fachbericht Nr. 25/2006.

[BFL06] J. Bayer, T. Forster, T. Lehner, C. Giese, A. Schnieders, J. Weiland. Process Family Engineering in Automotive Control Systems – A Case Study. GPCE4QoS-Workshop, GPCE, Oktober 2006.

[BPMN] Object Management Group. Business Process Modeling Notation (BPMN) Specification. OMG Final Adopted Specification, Februar 2006.

[CE00] K. Czarnecki, U. Eisenecker. Generative Programming – Methods, Tools, and Applications. Addison-Wesley, Boston, MA, 2000.

[EsW01] Eshuis, R.; Wieringa, R. A formal semantics for UML Activity Diagrams – Formalising workflow models, Technical Report CTIT-01-04, U. Twente, Dept. of Computer Science, 2001.

[GGH06] C. Giese, S. Golega, M. Heidenwolf, W. Lindenthal, H. Overdick, A. Schnieders, J. Schulz-Hofen. Process Family Engineering for E-Business Process Families: Case Study. PESOA-Report No. 26/2006, Delta Software Technology, Fraunhofer IESE, Hasso-Plattner-Institut, September 2006.

[GOB05] C. Giese, H. Overdick, W. Buhl. Realisierungsstrategien für Prozessfamilien. PESOA-Fachbericht Nr. 15/2005.

[Mil89] Robin Milner. Communication and Concurrency. Prentice Hall, New York, 1989.

[Puh06] Frank Puhlmann: A Tool Chain for Lazy Soundness. Demo Session of the 4th International Conference on Business Process Management, CEUR Workshop Proceedings Vol. 203, Vienna, Austria, pages 9-16, 2006.

[PW05] Frank Puhlmann, Mathias Weske: Using the Pi-Calculus for Formalizing Workflow Patterns. In W.M.P. van der Aalst et al. (Eds.): BPM 2005, volume 3649 of LNCS, Nancy, France, Springer-Verlag, pages 153-168, 2005.

[PW06] Frank Puhlmann, Mathias Weske: Investigations on Soundness Regarding Lazy Activities. In S. Dustdar, J.L. Fiadeiro and A. Sheth (Eds.): Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS, Vienna, Austria, Springer Verlag, pages 145-160, 2006.

[SaW03] Sangiorgi, D., Walker, D. The π-calculus: A Theory of Mobile Processes. Paperback edn. Cambridge University Press, Cambridge, 2003.

[SP05] Arnd Schnieders, Frank Puhlmann. Activity Diagram Inheritance. In Proceedings of the 8th International Conference on Business Information Systems BIS, Poznan, Poland, pages 20-22, 2005.

[Stö04] Störrle, Harald. Semantics of Control-Flow in UML 2.0 Activities, Intl. Symp. VisualLanguages/Human Computer Centered Systems 2004, Rome

[UML] Object Management Group. Unified Modeling Language: Superstructure. Version 2.0, August 2005.

[ViM94] Björn Victor and Faron Moller. The Mobility Workbench—a tool for the π-calculus. In David Dill, editor, CAV'94: Computer Aided Verification, volume 818 of Lecture Notes in Computer Science, pages 428–440. Springer-Verlag, 1994.

[WeH06] A. Werner, F. Huster. Evaluation der Process Family Plattform an einem Fallbeispiel. PESOA-Report No. 28/2006, Universität Leipzig, Dezember 2006.