White Paper

Impact Control

Offene Integrationsplattform

Metamodellbasierte Toolkopplung

Werkzeugübergreifende Traceability und Baselining

Und mehr

Offene Integrationsplattform Impact Control

Anforderungen bezüglich der Abbildung von Prozessen

Unabhängig davon ob eher agile oder formale Prozesse unterstützt werden sollen, ist für jedes Software entwickelnde Unternehmen ein Ansatz zu finden, der die richtigen Methoden, die richtige Werkzeugunterstützung und die richtige Einführungsstrategie liefert.

Ziel ist in jedem Fall eine durchgehende Tool-Unterstützung des gesamten Entwicklungsprozesses, beginnend beim Anforderungsmanagement, über die Designphase, die Planung der Umsetzung, die eigentliche Softwareproduktion bis zum Test gegen die ursprünglich definierten Anforderungen.

Kriterien für eine optimale Prozessabbildung, die jeweils verschiedene Ebenen betreffen, können bspw. wie folgt lauten:

- *Offener, herstellerunabhängiger Ansatz*
- *Inkrementell und flexibel erweiterbar bei hohem Integrationsgrad*
- *Traceability und Baselining über den gesamten Prozess*
- *Auf aktueller Open Source-Technologie basierend*
- *Anerkannten Prozess- und Technologiestandards genügend*

Grundsätzlicher Ansatz: Requirements Driven

Mit einem anforderungsgetriebenen Ansatz kann man per se einige der oben genannten Kriterien erfüllen, d. h. die Anforderungen „fließen" mit voller Transparenz durch den Prozess, von der Entstehung bis zur Auslieferung (egal ob neue Anforderung, Änderungsanforderung oder gemeldeter Fehler).

Es muss immer klar sein:

- Wo kommt die Anforderung her?
- In welchem Status befindet sie sich und wer hat sie bisher bearbeitet?
- Wie ist sie im System umgesetzt, in welcher Version?
- Welche Anforderungen gehören zusammen bzw. sind voneinander abgeleitet?

Unabhängig vom Prozessmodell sieht ein Entwicklungsprozess in etwa so aus:

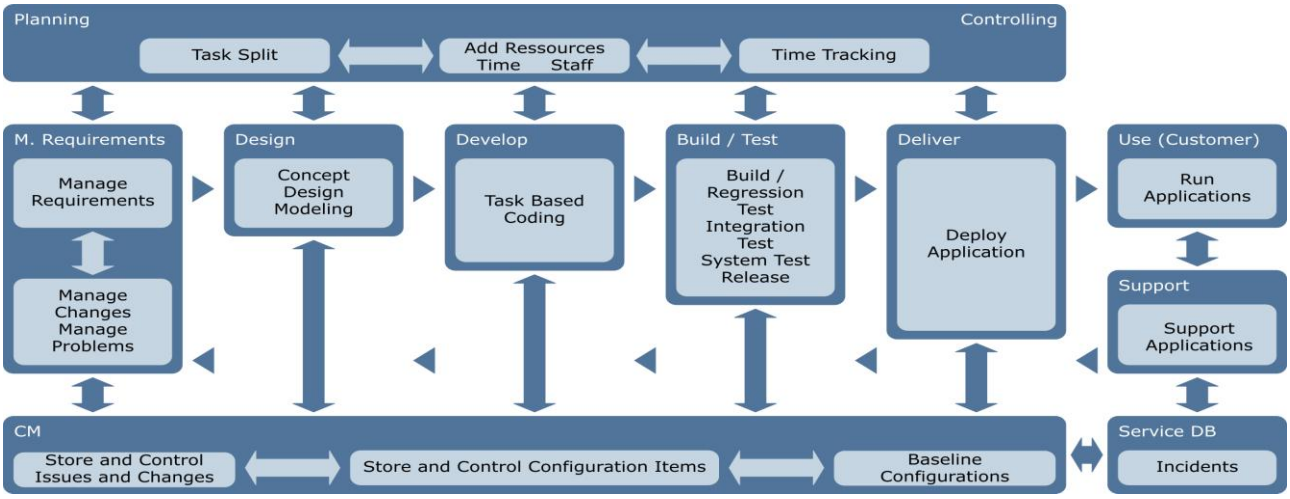

Abbildung 1: Softwareentwicklungsbausteine Prozesssicht

High Level Use Cases via Impact Control realisiert

Create Project Structure

- Anlage der Projektstruktur (Prozess-/ Entwicklungsstruktur) -> Templates definierbar

Requirements Management

- Requirements importieren/ exportieren
- Requirements mit externem Werkzeug synchronisieren (bidirektional)
- Bei Bedarf Requirements über internen Editor anlegen und bearbeiten

Design/ Modeling

- UML-Diagramme mit externem Werkzeug synchronisieren (bidirektional)
- Bei Bedarf UML-Diagramme über Eclipse-basierten internen Editor anlegen und bearbeiten

Automation of Modeling and Development Steps

- Mittels QVT (Query View Transformation) Modellierungs- und Sourcecodeartefakte nach vordefinierbaren Transformationsregeln automatisch generieren, bspw. generiere zu jedem Requirement einen Use Case und umgekehrt

Consistency Checks

- Mittels OCL (Object Constraint Language) Konsistenzregeln definieren und prüfen, bspw. prüfe, ob für jedes Requirement mind. 1 Use Case vorhanden ist

Traceability Management

- Setzen von Traceability-Links zwischen Artefakten
- Visualisieren der Traceability zwischen beliebigen Artefakten über Matrix
- Korrigieren bzw. Setzen von Traceability-Beziehungen in Matrix

Change Management

- Change Requests mit externem Issue Tracker synchronisieren
- Change Requests über Task-basierten Commit mit bearbeiteten Artefakten verlinken

Impact Analysis

- Direkte Abhängigkeiten darstellen
- Indirekte Verbindungen darstellen (bis Prozessanfang/ -ende)

Baseline-/ Release Management

- Baselines anlegen
- Artefakte zu Baselines zuordnen
- Status für Baseline vergeben, bspw. "planned", "test", "invalid", "released"
- Existierende Baselines und ihren Status anzeigen
- Artefakte zur Baseline anzeigen
- Zwei Baselines bzgl. der Change Requests an Artefakten (inkl. Modellen) und ihrer Beziehungen zueinander vergleichen
- Durch Baseline beschriebene Konfiguration aus Repository auschecken
- Baselines einzelner Komponenten zu einer Produkt-Baseline zusammenstellen (Hierarchische Baselines)

Version Control

- Alle relevanten Artefakte (Configuration Items) über zentrales Repository persistieren

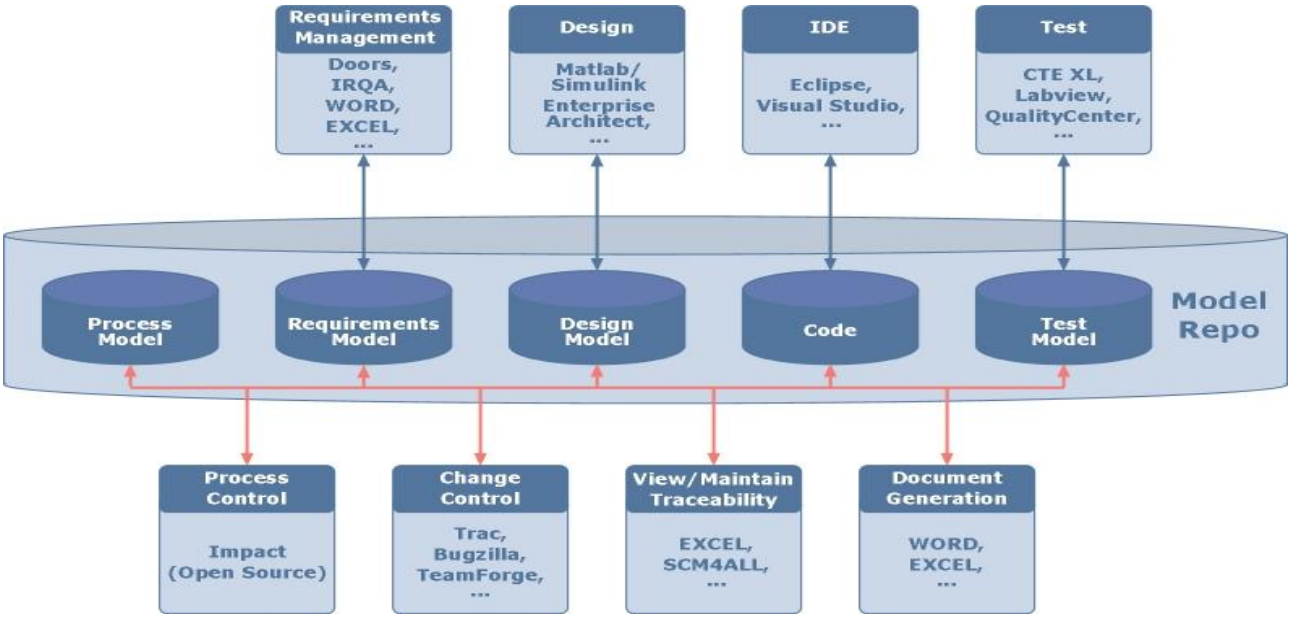

Abbildung 2: zentrales Repository

Software Development

- Task-orientiert arbeiten
- Sourcecode hinsichtlich Cyclomatic Complexity analysieren
- Reverse engineeren

Parallel Development of Artefacts incl. Models

- Konkurrierende Änderungen auf einer Entwicklungslinie vornehmen
- Modelle vergleichen (Differenzen visualisieren)
- Modelle mergen (Modelle selektiv zusammenführen)

Build-/ Test Management

- Build und Test über integrierte Open Source und/ oder kommerzielle Build- und Test Werkzeuge vornehmen

Project Management

- Tasks mit externem Projektmanagement-Werkzeug zur Iterations- und Release-planung synchronisieren

Document Generation

- Informationen über Artefakte aus den einzelnen Prozessphasen in vorbereitete Templates generieren, so dass mit Dritten Requirements, Design-Dokumente, etc. ausgetauscht werden können

Code-/ Configuration Generation

- Applikationen auf Basis der in früheren Prozessphasen vorgenommenen Modellierung und des Inputs bzgl. Konfigurationen und Varianten generieren

Metamodellbasierte Toolkopplung

Prinzip und Überblick

Üblicherweise sind im Softwareentwicklungsprozess viele Akteure involviert, die mit verschiedenen Tools auf unterschiedlichen Abstraktionsebenen an der Softwareerstellung mitwirken. Jedes Tool verarbeitet durch seinen verantwortlichen Akteur Informationen vorgelagerter Teilprozesse und erzeugt Arbeitsprodukte/ -ergebnisse für nachgelagerte Aktivitäten. In der Praxis kommt es in diesem Kontext zu natürlichen Brüchen, die z. B. aus folgenden Gründen auftreten:

- Tools sind untereinander oft inkompatibel, da Informationen nur bedingt und/ oder unvollständig ausgetauscht werden können.
- Die Ausführung von Teilaktivitäten kann u. U. nicht nachvollzogen werden, da der Informationsaustausch manuell erfolgt.
- Tools verwalten ihre Arbeitsprodukte in eigenen, von der Außenwelt gekapselten Repositorien (z. B. Datenbanken, Konfigurationsdateien, interne Dateiformate etc.) über verschiedene Betriebssystemplattformen hinweg. Die Beziehungen zu den Arbeitsprodukten anderer Tools (werkzeugübergreifende Traceability) können nicht abgebildet werden.

Zur Überwindung obiger Probleme wird eine Metamodell-basierte Toolintegration favorisiert. Im Kern geht es dabei um die Abbildung toolspezifischer Informationen auf ein vereinheitlichtes Metamodell, das alle erforderlichen Sprachkonstrukte zur prozess-übergreifenden Darstellung des Entwicklungsprozesses bereitstellt.

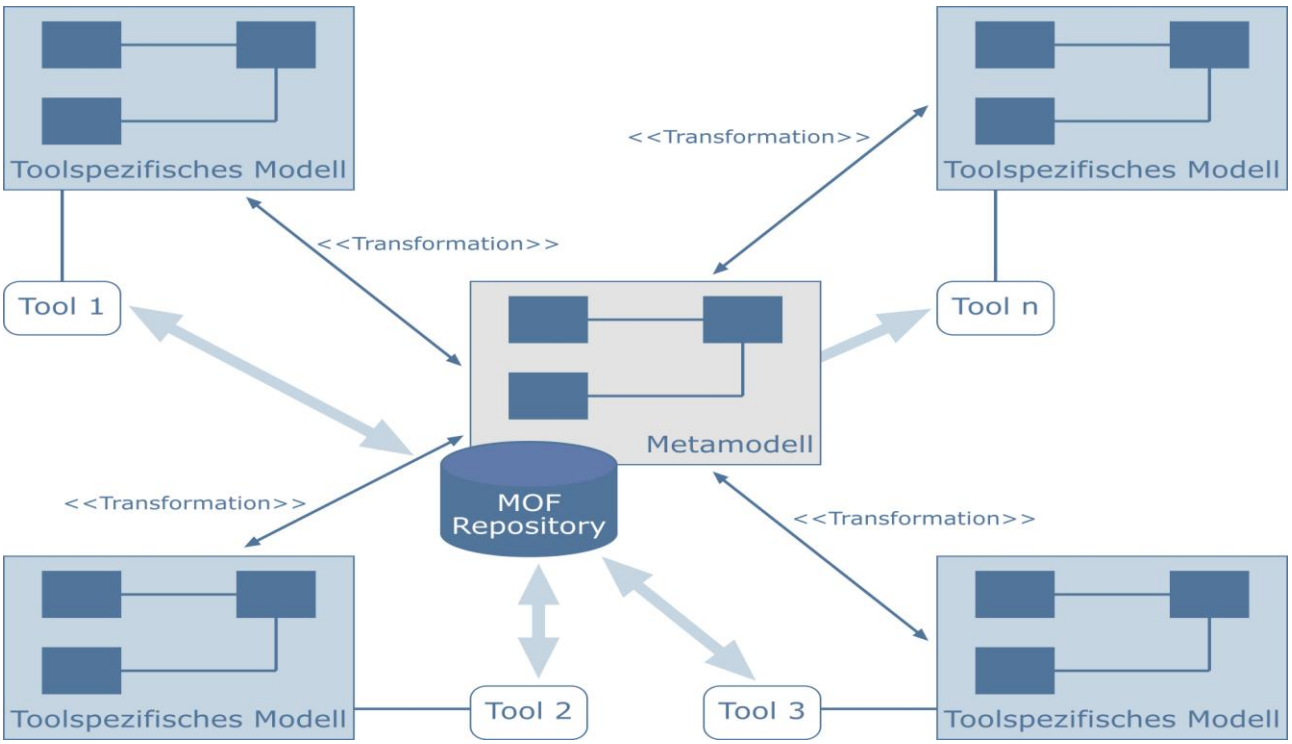

Abbildung 3: Prinzip der Toolkopplung über Metamodelle

Neben den verschiedenen Arbeitsprodukten (Modellinstanzen) der einzelnen Tools werden im Repository auch die Beziehungen (Links) untereinander persistent gespeichert. Vorraussetzung ist, dass Tools ihre spezifischen Informationen über Import-/ Export-Schnittstellen verfügbar machen können.

Im Kontext der Softwareentwicklung bietet sich als Metamodell die UML an, die bewährte Konzepte zur Beschreibung von Softwaresystemen bereitstellt. Dieses Metamodell wird um weitere Konstrukte (Requirements, Issues, Test Cases, etc.) erweitert, die die Arbeitsprodukte der einzelnen Entwicklungsphasen repräsentieren.

Je nach erforderlichem Detaillierungsgrad werden die Elemente feingranular oder in gröberen Blöcken modelliert. Die Beziehungen zwischen den Modellelementen werden durch UML-Assoziationen/ Links oder eigenen spezifischen Typen realisiert.

Im folgenden Diagramm sind exemplarisch einige Tools aufgeführt, die typischer Weise in den verschiedenen Entwicklungsphasen benutzt werden.

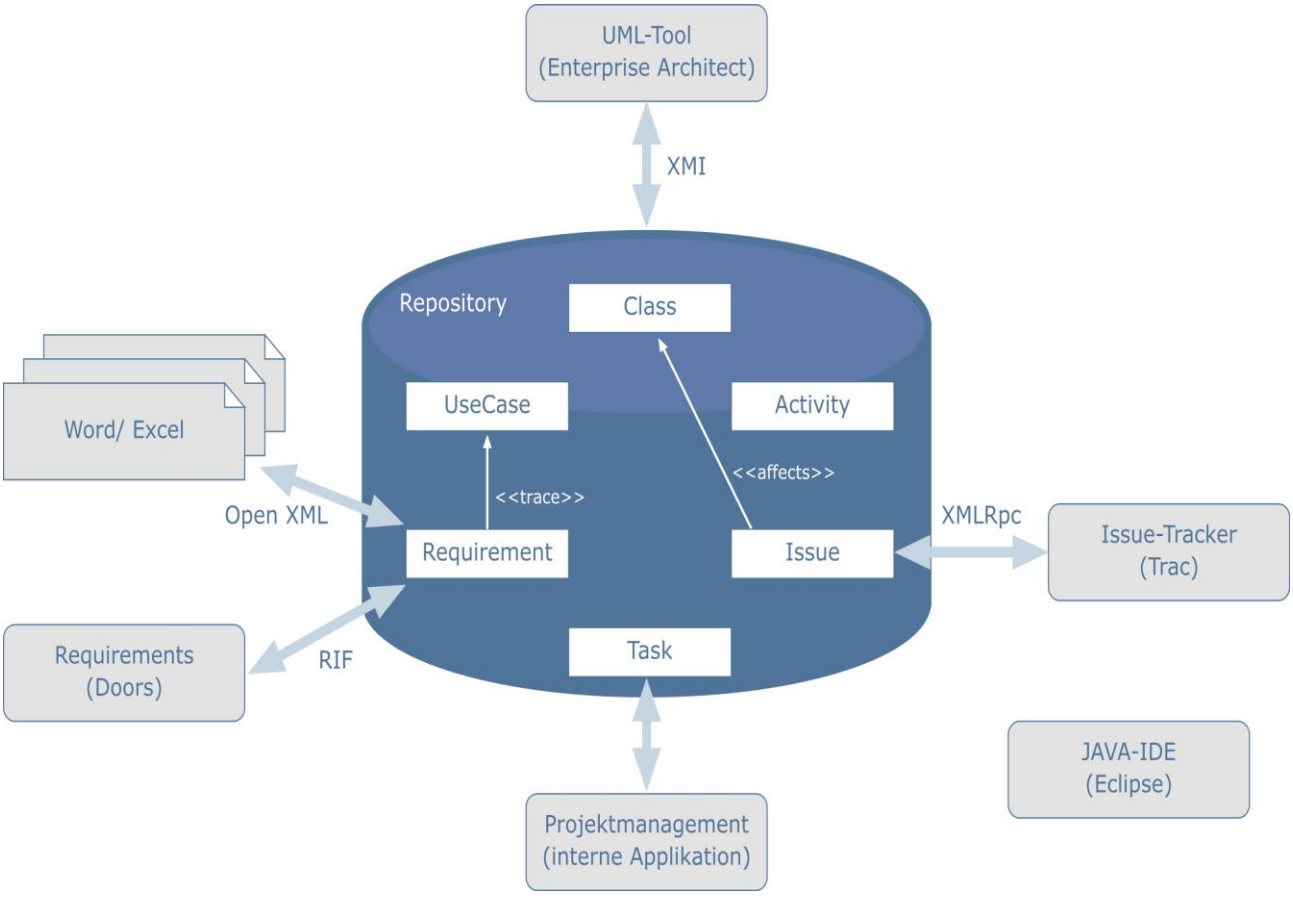

Abbildung 4: Toolkopplung über Metamodelle beispielhaft

Im Rahmen der Anforderungsanalyse werden funktionale und nicht-funktionale Requirements erstellt, die die Eigenschaften des zu entwickelnden Systems spezifizieren. Die Anforderungen werden in einem nächsten Entwicklungsschritt durch Use Cases und Aktivitätsdiagramme verfeinert. Ein in sich geschlossener Prozess erfordert die Nachvollziehbarkeit dieser Aktivität, d. h. zu einem späteren Zeitpunkt muss feststellbar sein, welche Anforderung welchen Use Case (und umgekehrt) initiiert hat.

Durch den Metamodell-basierten Ansatz lassen sich diese Informationen sicher und persistent verwalten, denn neben den Arbeitsprodukten werden auch die Beziehungen als Modellinstanzen gespeichert. Darüber hinaus lässt sich die Erstellung von Use Cases aus Requirements durch Modelltransformationen automatisieren.

Bekannte Metamodelle wie die UML oder CWM basieren ihrerseits auf dem MOF (Meta Object Facility) Meta-Metamodell, das von der OMG entwickelt wurde.

Die *Object Management Group (OMG)* ist ein 1989 gegründetes Konsortium, das sich mit der Entwicklung von Standards für die herstellerunabhängige systemübergreifende Objektorientierte Programmierung beschäftigt. Der OMG gehörten zur Gründung 11 Firmen, darunter IBM, Apple und Sun an. Mittlerweile hat sie über 800 Mitglieder und entwickelt international anerkannte Standards.

Zu den Meta Object Facilities gehören weitere Spezifikationen wie die QVT (Modelltransformationen) und OCL (Invarianten und Bedingungen). Mit diesen Technologien lassen sich Modellelemente durch Transformation aus anderen Elementen automatisiert generieren und auf ihre Modellkonsistenz überprüfen.

Am Beispiel der Anforderungsanalyse werde die hier in Word erstellten Requirements in das Repository über die OpenXML-Schnittstelle importiert. Durch Modelltransformationen wird zu jeder Anforderung ein Use Case (z. B. mit gleichem Namen) erzeugt. Die generierten Use Cases werden dann über das XMI-Austauschformat in einem UML-Tool weiter verfeinert. Die Invariante „jedes Requirement wird durch einen Use Case präzisiert" kann zu jedem Zeitpunkt durch eine Modellabfrage geprüft werden. Modellinkonsistenzen lassen sich dadurch schnell erkennen und frühzeitig beheben.

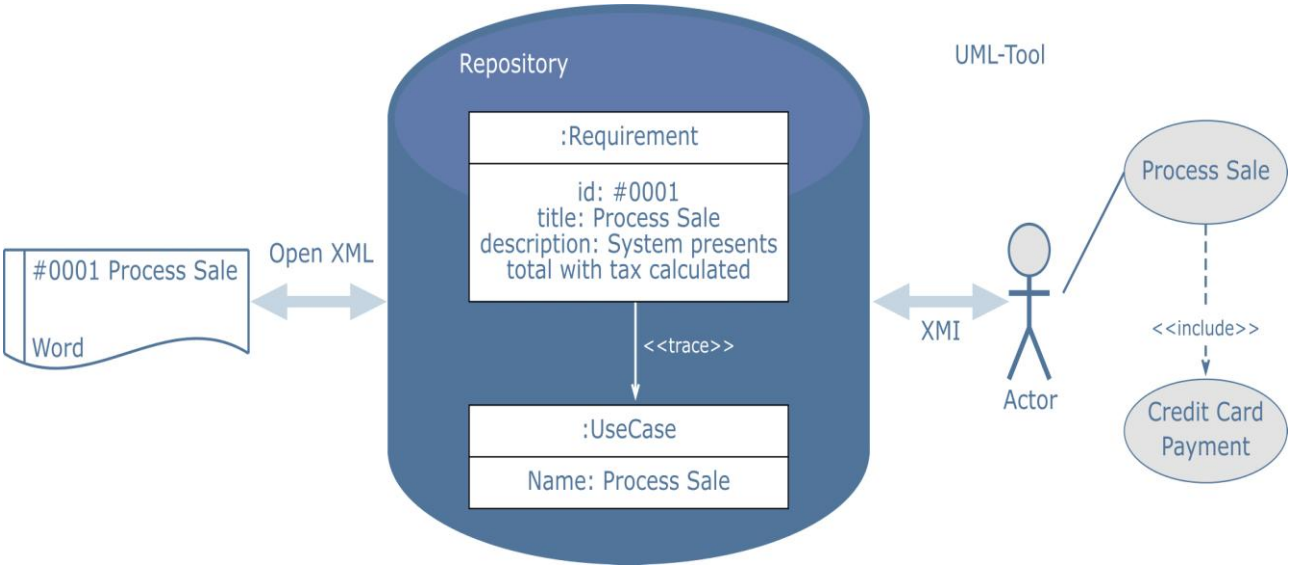

Abbildung 5: Import und Storage von Requirements via Metamodell

Dieses Konzept lässt sich zur Beschreibung aller weiteren Entwicklungsphasen durchgängig anwenden. Mit jedem Entwicklungsschritt wird das Repository um neue Modelelemente und Beziehungen stetig und konsistent erweitert.

Die Verwaltung des Repositories erfolgt über das Impact Control Center (ICC). Dieses ICC realisiert und nutzt die Schnittstellen zu externen Tools und stellt Funktionen für die Modelltransformation/ -abfragen bereit.

Metamodell: Softwareentwicklungsprozess

Für die einzelnen Entwicklungsphasen wird ein Metamodell konzipiert, das die typischen Artefakte, Arbeitsprodukte und Beziehungen der Aktivitäten abbildet.

Das folgende konzeptionelle Metamodell stellt beispielhaft die Verknüpfung zwischen den Bereichen Requirement, Design, Implementierung und Test dar.

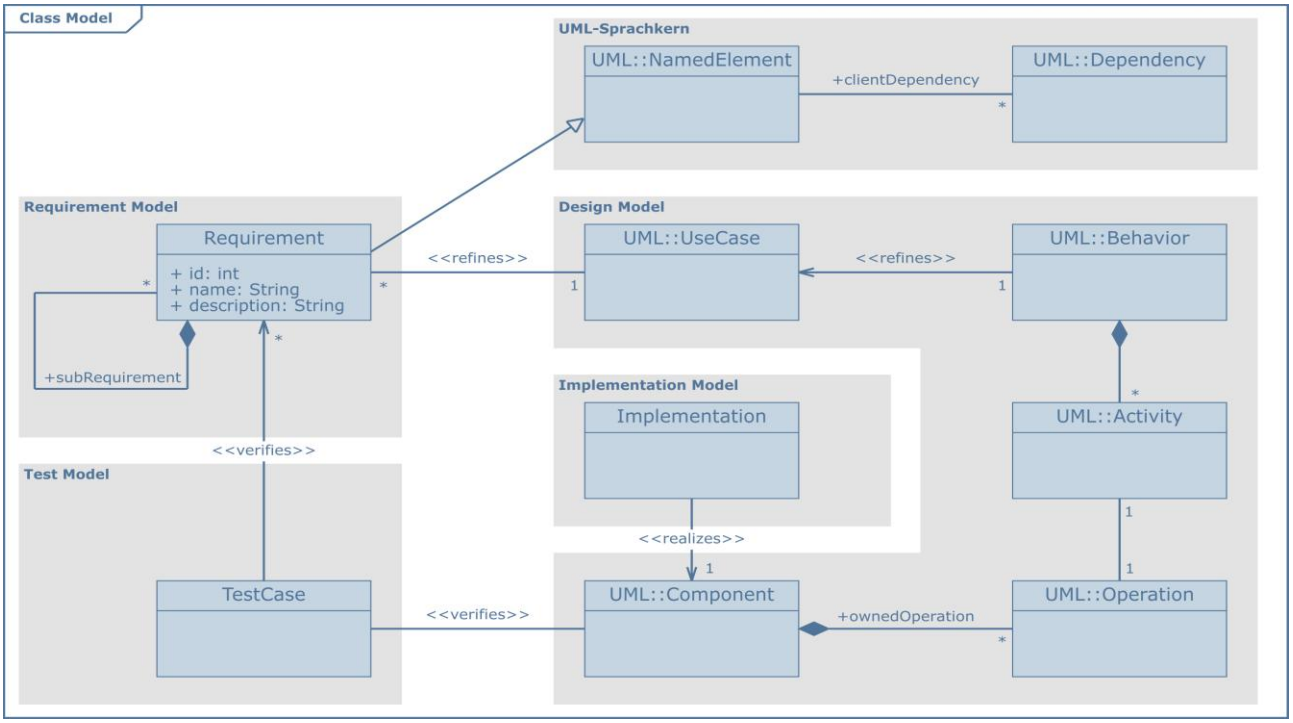

Abbildung 6: Modellierung der Traceability im Metamodell

Durch die übergreifende Modellierung von Abhängigkeiten zwischen den einzelnen Elementen ergeben sich „Regeln", die für den gesamten Entwicklungsprozess eingehalten werden müssen und die Durchgängigkeit sicherstellen:

- Jedes Requirement wird durch einen Use Case präzisiert
- Das Verhalten eines Use Case-Szenarios wird durch Aktivitäten (Aktivitätsdiagramm) beschrieben
- Aktivitäten werden auf die Operationen einer Komponente abgebildet
- Jede Entwurfskomponente wird durch ihre konkrete Implementierung realisiert
- Anforderungen und Komponenten werden durch einen geeigneten Testfall verifiziert

Eine modellierte Komponente die z. B. über keinen Test Case verifiziert wird, darf nicht für die Implementierung freigegeben werden.

Option: Model Driven Engineering

Das Repository enthält ein plattformunabhängiges Modell (PIM) des zu entwickelnden Softwareproduktes. Das System wird z. B. durch ein UML-Komponentenmodell spezifiziert. Die plattformabhängige Realisierung (PSM) erfolgt z. B. im Application Server-Umfeld durch Java Enterprise Beans. Mit der QVT können große Teile des Java Codes aus dem Komponentenmodell durch Standardtransformationen automatisiert generiert werden.

Technische Realisierung: Eclipse

Eclipse ist weltweit mittlerweile wohl das größte Open Source-Projekt. Es integriert viele Frameworks, die nahezu alle Bereiche der Softwareentwicklung abdecken.

Für die konkrete Realisierung der Integrationsplattform und des modellbasierten Repositories stehen in diesem Umfeld alle erforderlichen Technologien zur Verfügung.

- Umsetzung der MOF-Spezifikation durch das Eclipse Modeling Framework (Ecore, EMF, GMF, MDT, QVT, OCL)
- Applikationsentwicklung mit Rich Client Plattforms (RCP, SWT)
- Java-IDE für die Entwicklung von Tool-Adaptern.
- Erweiterbarkeit durch Eclipse-PlugIns
- UML 2.1 PlugIn zur Abbildung des Design-Modells

Das Impact Control Center ist als RCP-Anwendung realisiert und lässt sich als Eclipse-PlugIn auch direkt in die Eclipse-IDE einhängen. Die grafische Oberfläche (s. Standard Widget Toolkit) visualisiert die Modellelemente des Repositories und bietet verschiedene Navigationssichten auf das Gesamtmodell. Die Teilmodelle beruhen auf Ecore (Eclipse MOF) und können dadurch mit dem EMF-Framework nahtlos in die Architektur eingebunden werden.

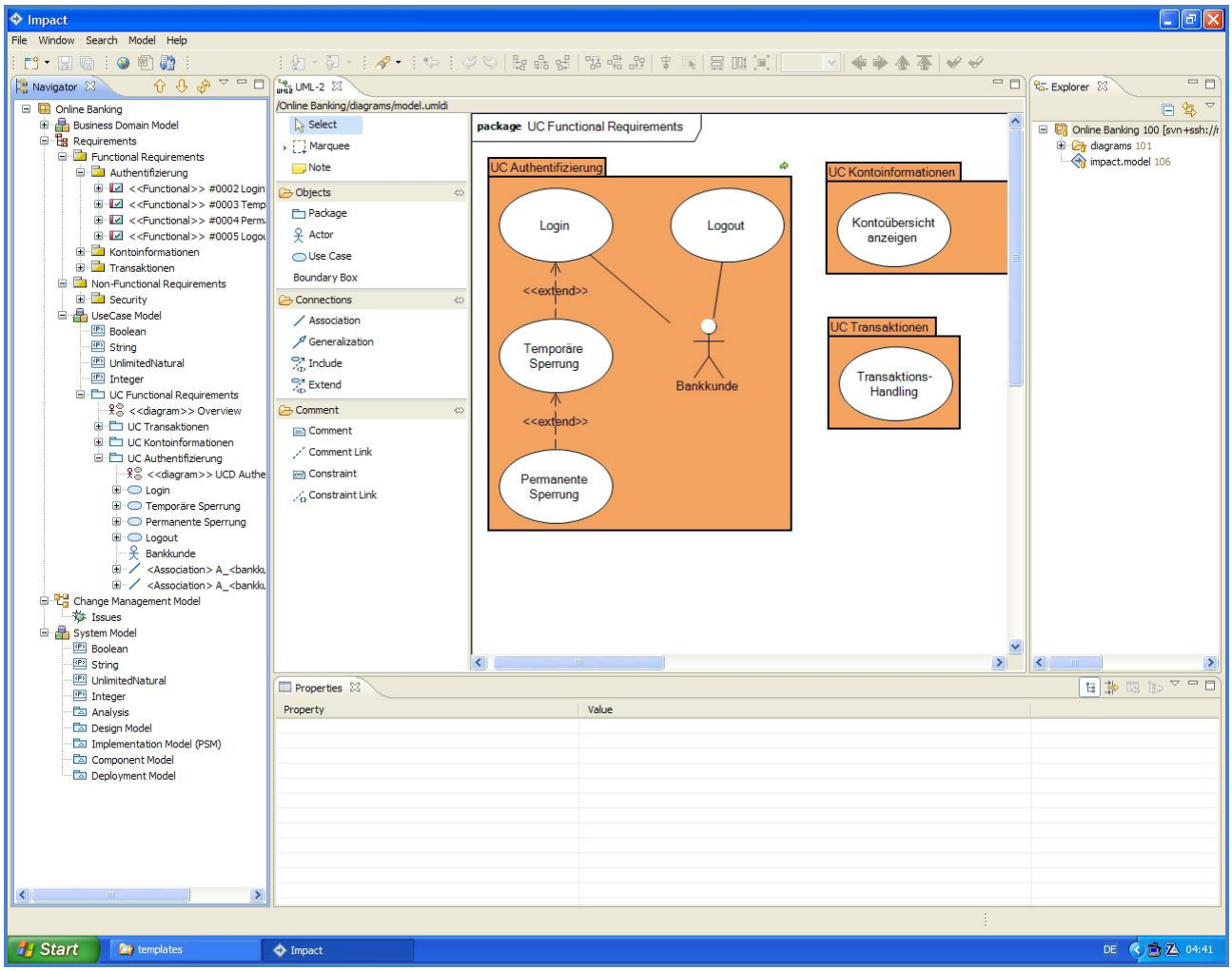

Abbildung 7: Screenshot: Sichten

Die Navigation durch das Repository erfolgt über eine Baumstruktur, die entsprechend den Projektanforderungen in logische Teilmodelle und Pakete strukturiert ist.

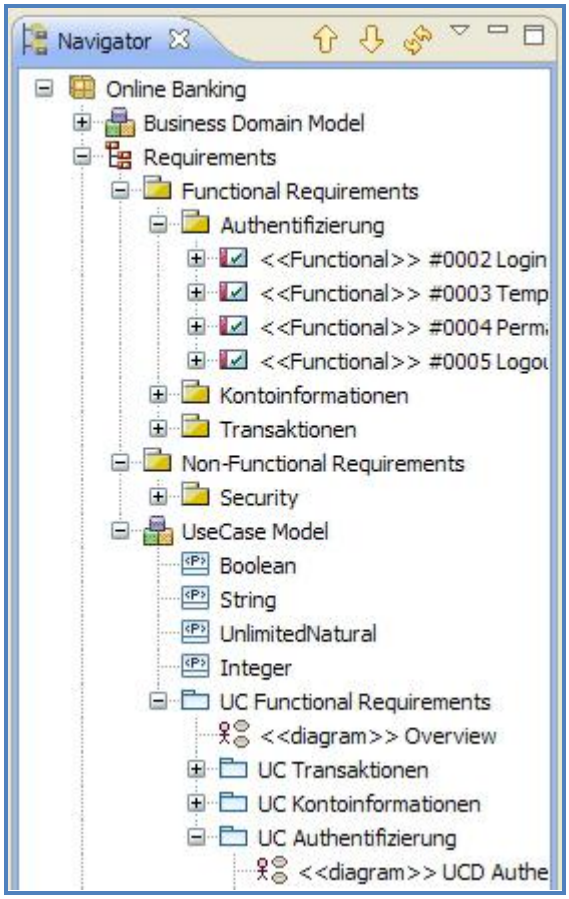

Abbildung 8: Screenshot: Navigation/Repository

Die Abhängigkeiten der Modellelemente lassen sich z. B. durch spezialisierte Sichten auf das Modell jederzeit nachvollziehen.

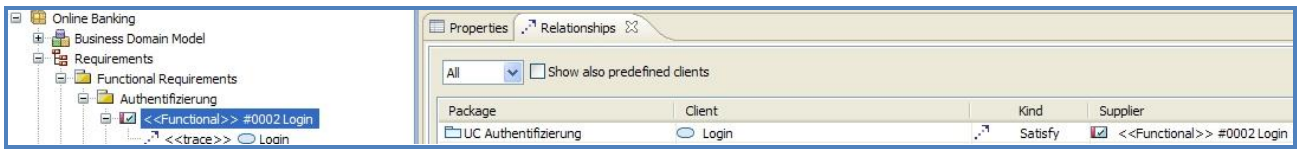

Abbildung 9: Screenshot: Relationships (z. B. Abhängigkeit Requirement -> Use Case)

Für die Modellvalidierung (OCL) und –transformation (QVT) sind im Eclipse-Umfeld ausgereifte Frameworks vorhanden. Modellinkonsistenzen können über integrierte Menüs jederzeit ermittelt, visualisiert und den einzelnen Prozessphasen zugeordnet werden.

Die Versionierung des Modells erfolgt über die Einbindung eines geeigneten Team-PlugIns (Subclipse, Subversive, etc.). Zukünftige Entwicklungen der Eclipse-Community in den

Bereichen Modeling und MDA können mit relativ geringem Anpassungsaufwand integriert werden und den Softwareentwicklungsprozess um neue sinnvolle Features erweitern.

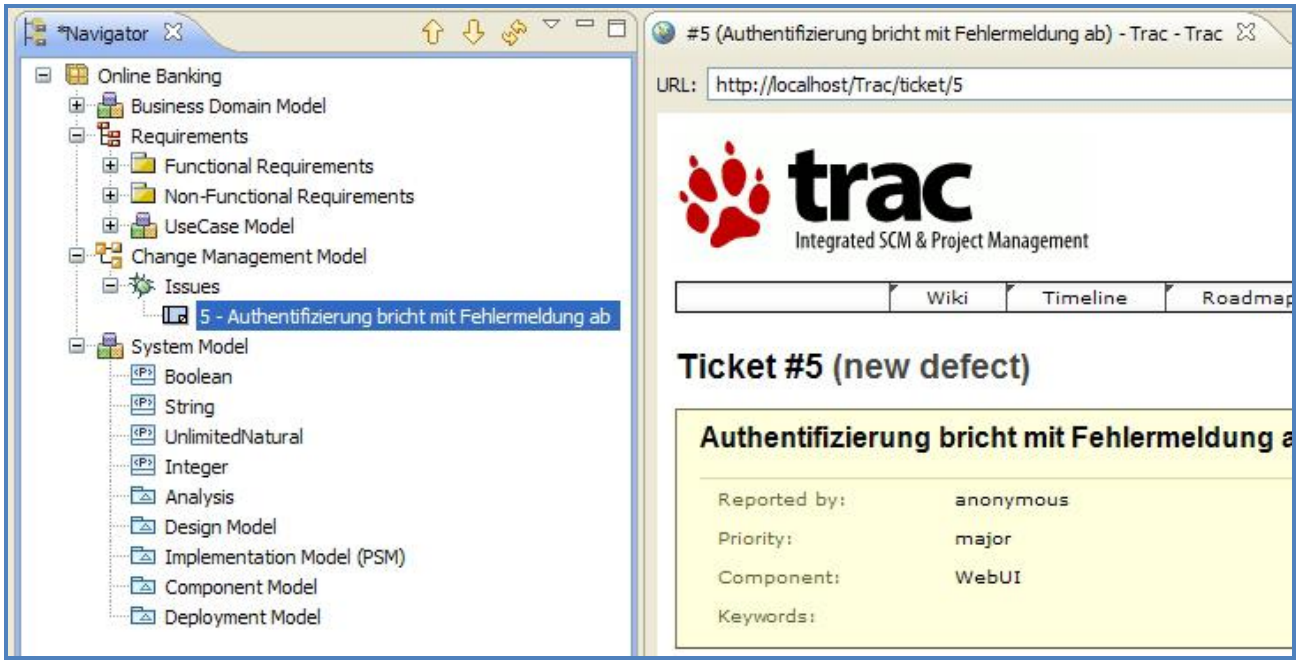

Abbildung 10: Screenshot: Toolintegration/z. B. Issue-Tracker

Die Anbindung der verschiedenen Werkzeuge erfolgt über Adapter. Webbasierte Issue Tracker wie Trac oder Bugzilla realisieren z. B. die XMLRpc-Schnittstelle. Der Adapter importiert über diese Schnittstelle die toolspezifischen Daten und überführt sie in eine äquivalente Ecore-Darstellung. Die Einführung eines neuen Tools erfordert lediglich die Entwicklung eines neuen Adapters. Der Modellkern bleibt davon unberührt.

Fazit

Zusammenfassend lässt sich feststellen, dass das Konzept des Metamodell-basierten Repositories auf Basis von Eclipse bedingt durch die Ecore-Standardisierung und die offenen Erweiterungsmöglichkeiten eine echte Alternative zu monolithischen „All-In-One"-Tools darstellt.