



orchideo

Die effiziente Art, Software zu entwickeln

Version 1.4

Inhaltsverzeichnis

Wunsch und Wirklichkeit der Softwareentwicklung	4
Die Vision: Ideale Softwareentwicklung	4
Die Herausforderung: Komplexität	4
Die Historie	5
Die Mittel zum Paradigmenwechsel	6
Abstraktion	6
Strukturierung	6
Methodik	8
Tools	8
Die Lösung	9
orchideo	10
Was ist orchideo?	10
Der Kern: orchideo engine	11
Objekte managen: orchideo objects	12
Modellierung mit IDE Komfort	12
Features für POJOs	13
Benutzeroberflächen gestalten: orchideo views	13
Das orchideo views Baukastensystem	14
Konzeptionen in orchideo views	16
Dokumente erzeugen: orchideo documents	18
Strukturierung der Dokumentationsgenerierung	18
Zielformate	18
Infrastruktur und Architektur	19
Prozess der Dokumentationsgenerierung	20
Definition einer Dokumentation	20
So einfach wie möglich	21
Oberflächen beschreiben: orchideo ui-designer	22
Komplexität beherrschen mit orchideo	23

Liebe Leserin, lieber Leser,

Sie kennen das sicher auch, es gibt Dinge, die werden immer knapper. Beispielsweise die Zeit und das Budget für die Entwicklung von Software. Hinzu kommen steigende Anforderungen. Software soll von Anfang an fehlerfrei erstellt werden, die Entwicklung des Systems möglichst lange flexibel bleiben und die Anwendung ein möglichst langes Leben haben, allen nachträglichen Änderungen zum Trotz.

Mit der traditionellen Vorgehensweise der Softwareerstellung, bei der große Teile des Codes manuell erstellt werden, können wir diesen Herausforderungen nicht mehr begegnen. Sie ist in weiten Bereichen nicht ausreichend strukturiert und befasst sich mit immer wiederkehrenden technischen Problemstellungen auf zu niedriger Abstraktionsebene.

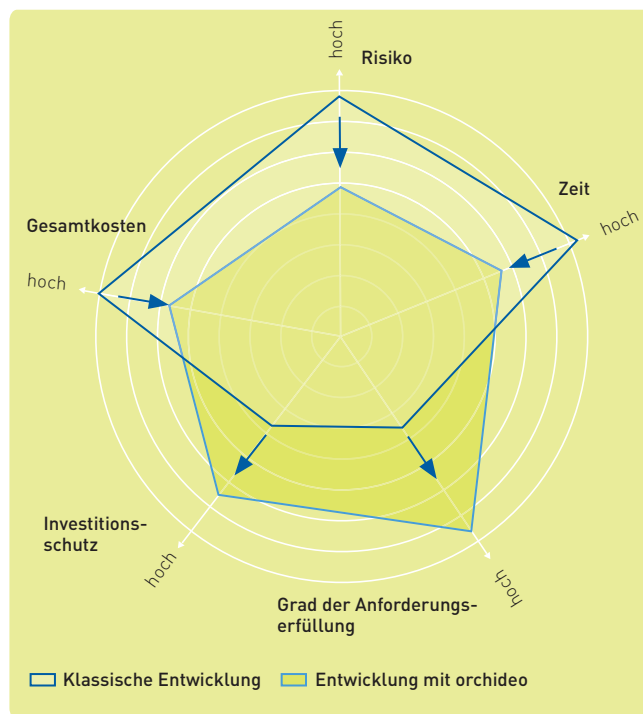
Mit orchideo haben wir einen Weg beschritten, der die innovativen Trends der Softwareentwicklung aufgreift und fortschreibt. orchideo ist unsere Strategie zu unserer Vision: der idealen Softwareentwicklung.

Wunsch und Wirklichkeit der Softwareentwicklung

Die Vision: Ideale Softwareentwicklung

Software wird immer wichtiger. Sie steuert Beschaffungs- und Produktionsprozesse, unterstützt die Produktentwicklung, bestellt, fakturiert, kommuniziert und vieles mehr. Die ökonomischen Konsequenzen nicht funktionierender Software sind Existenz gefährdend. Aber auch funktionsfähige, jedoch schlichtweg nicht den Bedürfnissen entsprechende Software, behindert Unternehmen und Menschen in ihrer Arbeit. Die Forderung ist einfach: Unternehmen brauchen die ideale Software, um effizient und wirtschaftlich erfolgreich arbeiten zu können.

Unsere Vision ist es, mit der idealen Softwareentwicklung unseren Kunden diese ideale Software zur Verfügung zu stellen. Dabei müssen unter den gegebenen Rahmenbedingungen von Zeit und Kosten die bestehenden Anforderungen möglichst exakt umgesetzt werden. Die entstehende Lösung muss dabei zukunftsfähig sein, um auch kommende Änderungen integrieren zu können. In allen Projektphasen muss dabei das Risiko stets minimal und kontrollierbar bleiben.



Effizienz in der Softwareentwicklung

Reifegrad von Softwareentwicklung

Als Maßstab für den Reifegrad (und damit die Qualität) der Softwareentwicklung dienen folgende Kriterien:

Zeit: Wie viel Zeit muss aufgewendet werden, um das gewünschte System zu erstellen?

Kosten: Welche Kosten müssen für die Erstellung und Wartung eines Systems aufgewendet werden?

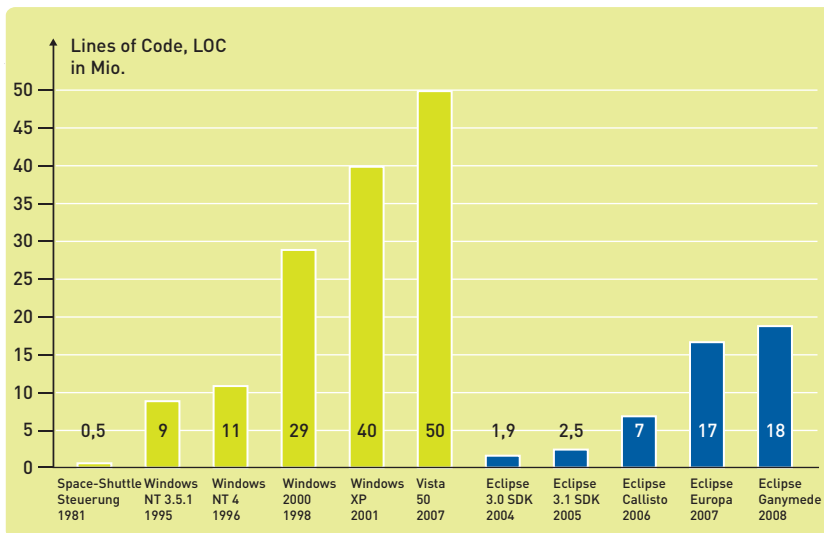
Investitionssicherheit: Wie viel Investitionssicherheit bietet die Lösung? Investitionssicherheit definiert sich über die Lebensdauer einer Software, die maßgeblich von Faktoren wie Erweiterbarkeit und Wartbarkeit abhängt. Erweiterbarkeit und Wartbarkeit korrelieren meist direkt mit der Güte der Struktur („inneren Ordnung“) einer Software.

Grad der Anforderungserfüllung: Wie hoch ist der Grad der Anforderungserfüllung? Je höher der Grad der Anforderungserfüllung ist, desto gewinnbringender kann die Software die Prozesse unterstützen, für die sie entwickelt wurde. Relevant sind dabei funktionale wie nicht-funktionale Anforderungen.

Risiko: Mit welcher Wahrscheinlichkeit wird ein hoher Grad an Anforderungserfüllung im Zeit- und Kostenrahmen erzielt?

Die Herausforderung: Komplexität

Die immer wichtiger und geschäftskritischer werdende Software birgt eine inhärente Herausforderung in sich: Sie wird immer komplexer. Sei es durch wachsenden Funktionsumfang, hohe Ansprüche an Komfort und Bedienbarkeit, Integration mit anderen Systemen, Einbettung in immer komplexer und dynamischer werdende Geschäftsprozesse oder eher eine Kombination aus allen Faktoren – die Software muss mithalten. Gleichzeitig werden die Zeitrahmen, die zur Realisierung dieser stets komplexer werdenden Softwaresysteme zur Verfügung stehen, immer knapper.



Größe von Software-Systemen

Nüchterne Zahlen belegen das ungeheure Wachstum sehr eindrucksvoll: So flog das Space Shuttle 1983 noch mit 500.000 Zeilen Code (Lines of Code, LOC) ins All. Mit Windows Vista sind wir heute bei 50 Millionen LOC angelangt. Selbst im Open Source Bereich zeigt sich ein immenses Wachstum, was am Beispiel des eclipse Ökosystems sehr deutlich wird: Von 2,5 Millionen LOC des eclipse 3.0 SDKs im Jahre 2004 bis zu 18 Millionen LOC des eclipse Ganymede Releases 2008.

Die Historie

Nach der maschinennahen Programmierung kam bald die strukturierte Programmierung. In den 80er Jahren begann die Entwicklung von OO, die ihren Durchbruch in den 90ern hatte. Jeder dieser Paradigmenwechsel brachte Verbesserungen in der Strukturierbarkeit von Softwaresystemen.

Am Anfang jedes Wechsels standen Weiterentwicklungen im Bereich der Programmiersprachen. Wichtigstes Merkmal dabei: Jede Evolution führt in den Sprachen zu höheren Abstraktionsebenen. Statt Maschinenregistern verwendet der Entwickler Variablen. Statt konkreten Einsprung- und Rücksprungadressen ruft er Methoden auf und verwendet IF, FOR und WHILE Ausdrücke. Statt Zeiger auf Adressbereiche verwendet er Datenstrukturen und Objekte. Jüngste Errungenschaft: Statt expliziter Speicherverwaltung verwendet er automatisches Garbage Collecting.

Doch die Sprache alleine schafft den Paradigmenwechsel nicht. Ausgehend von der Sprache müssen Methodiken und Werkzeuge entstehen, die die neue Technologie begreifbar und beherrschbar machen. So entstanden aus der OOP die OOA und die OOD, oder allgemein die OOSE, mit fundierten Konzepten zur besseren Beherrschbarkeit von Software. Parallel dazu erleichtern Tools die Arbeit im neuen Paradigma, seien es IDEs mit ausgefeilten Code-Editoren und Klassenbrowsern oder grafische Werkzeuge für SA/SD in den 80ern und die CASE Tools der 90er.

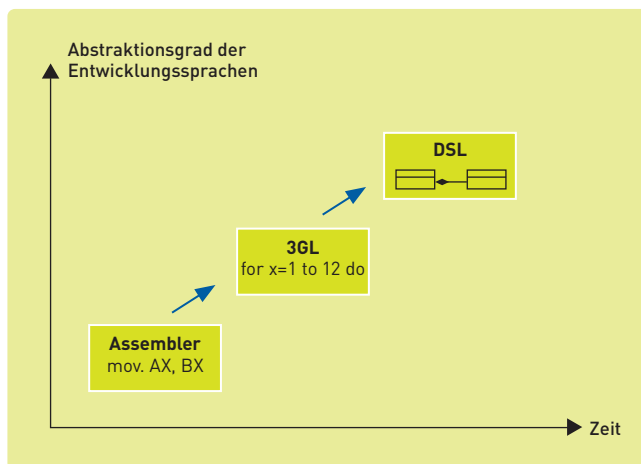
Die Voraussetzungen für einen Paradigmenwechsel sind demnach:

- > Höhere Abstraktion der Sprache
- > Bessere Strukturierbarkeit von Softwaresystemen
- > Entwicklung formaler Methodiken
- > Werkzeugunterstützung

Die Mittel zum Paradigmenwechsel

Abstraktion

Das Prinzip ist einfach: Mit Abstraktion lassen sich Details reduzieren. Man konzentriert sich auf das Wesentliche und kann mit geeigneten Konstrukten die Kerninformationen klarer ausdrücken. Die Vorteile: Effizienzgewinn durch Weglassen unnötiger Details und Qualitätssteigerung durch Verlagerung der tieferen Abstraktionsebenen in eine strukturierte, stabile Umgebung.



Abstraktion der Sprachen

Im Bereich der Programmiersprachen ist das Prinzip der Abstraktion offensichtlich. Programmierte man zu Beginn noch in Assembler und Maschinensprache direkt mit CPU-Befehlen, so waren erste Hochsprachen wie PL/I, COBOL und FORTRAN bereits losgelöst von Maschineninstruktionen und ein Compiler übernahm die Übersetzung des verständlicheren Programm-codes. Java, .NET und viele andere neuen Programmiersprachen befreien den Programmierer mittlerweile durch Garbage Collection von manuellem Speichermanagement.

Die Modellierungssprache UML hat sich im Analysebereich stark durchgesetzt. Allerdings treten mit zunehmendem Verbreitungsgrad die Schwachstellen der Übermächtigkeit einerseits und fehlender Detailtiefe andererseits zu Tage. In jüngster Zeit etablieren sich nun Domain Specific Languages (DSLs) am Markt. Domänenspezifische Sprachen sind genau für eine Domäne,

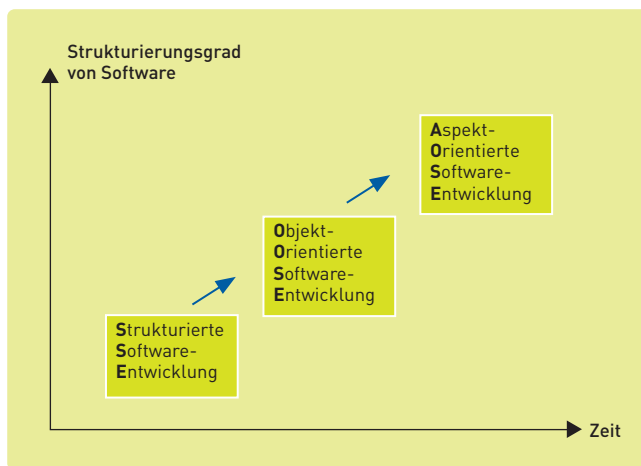
ein Anwendungsgebiet oder einen Zweck entworfene, formale Sprachen. Sie können die notwendigen Beschreibungen exakt ausdrücken und dabei gleichzeitig alle nicht relevanten Teile abstrahieren. Das Gegenteil sind universell einsetzbare Programmiersprachen wie C und Java oder eine universell einsetzbare Modellierungssprache wie UML. DSLs sind heute im Fokus der modellgetriebenen Softwareentwicklung (model-driven software development, MDS), bei der ein Modell der Software mit Hilfe von DSLs erstellt wird und dann automatisiert durch Compiler, Generatoren oder Interpreter in lauffähige Software transformiert wird.

Strukturierung

Mit Abstraktion allein lassen sich komplexe Softwaresysteme nicht beherrschen. Software benötigt eine innere Struktur. Sie muss in kleinere Teile zerlegt werden, welche für sich genommen überschaubar und beherrschbar bleiben. Jeder dieser Teile sollte dabei eine klar definierte Aufgabe haben. Ein anderer Begriff für dieses Konzept der Strukturierung ist Modularisierung, und in jüngster Zeit hat man den von Informatikgröße Edsger Dijkstra bereits 1974 geprägten Terminus Separation of Concerns (etwa Trennung der Belange) wieder entdeckt. Hinter allen Bezeichnungen steckt das gleiche Ziel: Das Aufteilen der Software in kleinere Blöcke, die überschaubar sind, parallel entwickelt werden können, langfristig wartbar bleiben und im Idealfall auch wieder verwendbar sind.

Ähnlich wie bei der Abstraktion hat die Informationstechnologie auch im Bereich der Strukturierung eine lange Entwicklung hinter sich. Nach der Pionierzeit Mitte des letzten Jahrhunderts hat sich schnell die strukturierte Programmierung durchgesetzt, bei der Konstrukte wie Schleifen, Blöcke und Funktionen eingeführt wurden. Die objekt-orientierte Softwareentwicklung hat mit ihren Konstrukten wie Klassen, Vererbung und Beziehungen neue Dimensionen der Strukturierung von Software eröffnet und mit dem Konzept der Kapselung vormals verworrene und verstreute Elemente der Software neu abgegrenzt. Doch die Objektorientierung stößt bei der Größe und Komplexität heutiger Softwaresysteme an ihre Grenzen.

Die vorherrschende Lösung für diese Herausforderung ist die Bildung von größeren, klar definierten Komponenten und Services. OSGi und SOA sind aktuelle Ausprägungen dieses Konzepts. Doch egal, ob auf Objekt- oder Komponentenebene modularisiert wird, es gibt Teile der Software, die sich der Modularisierung mit herkömmlichen Mitteln widersetzen. Dabei handelt es sich um Querschnittsfunktionalitäten, die mehrere Module betreffen. Typische Beispiele für diese Art von Funktionalitäten sind Logging, Fehlerhandling oder Transaktionssteuerung. Ihre Realisierung ist über diese anderen Module



Strukturierung von Software

verteilt – das Lokalkriterium für modularen Code wird massiv verletzt. Die jeweiligen Module wiederum sind übersät mit Code aus Querschnittsbelangen, der die Lesbarkeit und Verständlichkeit der eigentlichen Funktionalität des Moduls erschwert. Ganz offensichtlich sind Änderungen an solchen Querschnittsbelangen nur sehr aufwändig umzusetzen. Neue Querschnittsbelange in ein bestehendes System einzufügen ist ebenfalls nur mit großem Aufwand möglich.

Ein Kandidat, der bereits im Namen seinen Anspruch auf einen Paradigmenwechsel kundtut, ist die Aspektorientierung. Die Aspektorientierung erschafft dabei neue Dimensionen für die Strukturierung von Software, indem Querschnittsfunktionalitäten und übergreifende Teile in Aspekten modularisiert werden können. Zugehörige Werkzeuge verbinden dann die Aspekte mit den anderen Modulen und erzeugen somit die lauffähige Anwendung. Dieser Prozess wird „verweben“ („weaving“) genannt.

Die Aspektorientierung hat zum Ziel, die „Separation of Concerns“, also die Modularisierung von Software zu verbessern. Sie erreicht dies durch die Einführung neuer Konstrukte und von Regeln, wie diese Konstrukte miteinander zu verbinden sind. Die Vorteile dieses Ansatzes liegen auf der Hand: Eine optimale Modularisierung erlaubt optimale Parallelisierung der Softwareentwicklung, erzeugt überschaubare, gut lesbare und klar abgegrenzte Funktionseinheiten der Software und resultiert in leichter wieder verwendbaren und austauschbaren Modulen. Der Ansatz, neue Konzepte als Erweiterung einer bekannten Mainstream-Sprache zu platzieren, ist durchaus sinnvoll, wie der Erfolg von C++ auf Basis von C gezeigt hat.

Methodik

Die modellgetriebene Softwareentwicklung beschreibt als eines ihrer Grundkonzepte den stetigen Rückgriff auf geeignete Modelle, anstatt die Informationen während des Entwicklungsprozesses fortlaufend einer Replikation, Umformung und Verteilung zu unterziehen (DRY-Prinzip - Don't-Repeat-Yourself). Bei MDSD drückt man die fachlichen Aspekte der zu erstellenden Software mit einer DSL aus. Technische Aspekte werden hier weitestgehend nicht abgebildet, sondern werden durch die automatisierte Transformation Teil der Software. Dieses Vorgehen erlaubt es insbesondere, die fachlichen Aspekte durch Änderung der Transformation des Modells auf andere technische Umgebungen zu portieren. Die Investition in die fachlichen Aspekte der Anwendung bleibt dabei geschützt. Die Trennung von Fachlichkeit und Technologie ist ein wesentlicher Investitionsschutz.

DSLs haben meist eine einfach verständliche, grafische Notation, die sich oft an die bekannte (aber ungleich komplexere) UML-Notation anlehnt. Anders als bei klassischer Programmierung, bei der die Struktur und das Verhalten der Software lediglich aus kryptischem Programmcode ablesbar sind, bietet die modellgetriebene Softwareentwicklung die Möglichkeit, die Konzepte der Software in Bildern und Diagrammen darzustellen. Diese können nicht zuletzt auch mit dem Fachbereich erstellt und diskutiert werden. Die verbesserten Kommunikationsmittel zwischen IT und Fachabteilungen resultieren in einer deutlich verbesserten Erfassung und Abdeckung der Anforderungen.

A propos Sicht: DSLs eignen sich ebenso gut für die grafische Visualisierung wie für die textliche Darstellung. Entsprechende Werkzeuge und Bibliotheken wie EMF, GMF und Xtext machen diesen Ansatz bereits jetzt praxisreif.

Die im MDSD Ansatz verwendeten Generatoren, Interpreter und Frameworks für die Zielumgebungen bieten Funktionalitäten, die in sehr vielen Teilen des Entwicklungsprojekts Verwendung finden. Im Idealfall können die Komponenten in mehreren Projekten oder einer Softwaresystemfamilie weiterverwendet werden, was ein offensichtlich hohes Potenzial an Einsparung in sich birgt. Man gewinnt Effizienz und Qualität. Diese Form der Wiederverwendung kann schon innerhalb eines Projektes tragen.

Die automatische Transformation der Modelle in lauffähige Anwendungen durch Generatoren und Interpreter erspart dem Entwickler einen großen Teil manueller Programmierarbeit. Das führt zu höherer Effizienz bei gleichzeitiger Steigerung der Qualität, da sich durch Reduktion manueller Programmierung auch die Anzahl der entstehenden Fehler reduziert.

Gleichwohl zeigt die Praxis, dass es aufgrund der unterschiedlichen Granularitäten und schnell unübersichtlich werdenden Darstellungen nur selten Sinn macht, sich auf eine 100%ige Modellierung zu fokussieren. Dies betrifft besonders die Domäne der GUI-Gestaltung, da hier das vollständige Universum der verschiedenen Ausdrucksformen die Mittel eines sinnvoll beherrschbaren Modells bei weitem übersteigt. Es gilt also je nach Problemdomäne eine ideal ausbalancierte Mischung zwischen Modell und Implementierung nutzen zu können. Die Frage nach dem idealen Modellierungsgrad adressiert sowohl die Domäne der Fachlichkeit als auch die ergänzenden, technischen Bereiche. Anschließend gilt es in der Entwicklung, die modellierte Information nahtlos mit der Implementierung zu integrieren.

Tools

Software ist aus keinem Unternehmen wegzudenken, sie unterstützt und integriert alle wichtigen Unternehmensprozesse – so auch die Entwicklung und Fertigung. Ein Unternehmen, das selbst Software fertigt, benötigt genauso die geeignete Software, um ihre Produkte / Kundenlösungen effizient zu erstellen. Beim Einsatz von neuen Technologien, Methoden und Sprachen lässt die geeignete Toolunterstützung oft zu wünschen übrig. Dieser Umstand ist eine Hürde für deren wirtschaftlichen Einsatz. Wie unsere Kunden Softwarelösungen zur effizienteren Prozessunterstützung erstellen lassen, müssen auch wir unsere Tools zur effizienten Softwareentwicklung erstellen. Der Entwickler, der auf Basis eines MDSD-Frameworks entwickelt, darf von den unterschiedlichen Umsetzungsschritten (Modellierung, Implementierung) in lauffähige Software nichts merken. Er hat eine klar abstrahierte und strukturierte Sicht der Dinge vor sich und kann den Komfort à la eclipse IDE voll nutzen. Das Ideal ist eine durchgängige Toolchain, die sich als Einheit darstellt.

Die Lösung

Für sich alleine gesehen ist keines der vorgestellten Konzepte zum jetzigen Zeitpunkt fähig, einen Paradigmenwechsel herbeizuführen. Dem einen fehlt Struktur, dem anderen Abstraktion, dem dritten Methodik oder Tools. Eine vielversprechende Lösung steckt hinter der Kombination mehrerer Konzepte. Warum nicht den Strukturgewinn von Aspektorientierung mit der Abstraktion von DSLs verbinden? Warum nicht das Ganze in eine kombinierte Modellierungs- und Entwicklungsumgebung bringen?

Unsere Antwort auf diese Herausforderung ist orchideo. Wofür steht das? Nun, die Orchidee ist ein Hybrid. Hybride verbinden eine Reihe von Einzelkomponenten und Bestandteilen zu einem größeren Ganzen mit Mehrwert. Innovation lebt von Kreativität und der Realisierung von neuen Ideen. Unsere Vision ist die ideale Softwareentwicklung. Unser Sinnbild ist orchideo.

Mit orchideo haben wir eine Umgebung geschaffen, die es ermöglicht, die Anforderungen von Kundenlösungen exakt und verständlich zu beschreiben. Exakt, weil wir damit Wissen in Form von Modellen für die Transformation und Versorgung von implementierungsnaheren orchideo Komponenten haben. Verständlich, weil unsere Softwarelösungen nur so gut sein können, wie wir die Belange unserer Kunden verstanden und rückgekoppelt haben. Wir bauen Individuallösungen, das sind Maßanzüge – keine Konfektionsware.

Aspektorientierung

Die Aspektorientierung ist ein noch sehr junges Paradigma in der Informatik und als Ergänzung – nicht als Ersatz – für die Objektorientierung zu betrachten. Die Einführung der Aspektorientierung macht derzeit eine ähnliche Entwicklung durch, wie die der Objektorientierung Ende der 80er Jahre:

Das Konzept erlangt Aufmerksamkeit durch die Vermischung mit populären Programmiersprachen (AspectJ und Java vs. C++ und C). Es entstehen theoretischere Abhandlungen über die Thematik und man beginnt deren volle Auswirkungen zu verstehen. Man gelangt von aspektorientierter (vs. objektorientierter) Programmierung zu aspektorientierter (vs. objektorientierter) Softwareentwicklung. Die Debatte über das neue Paradigma wird oft ablehnend und emotional geführt durch die (unnötige) Angst der Ablösung der „alten“ Konzepte der Objektorientierung (vs. strukturierter Programmierung). Es gibt „early adopter“ in der Entwicklergemeinschaft und solche, die den Paradigmenwechsel nicht mitmachen wollen oder können.

Wir sind überzeugt davon, dass die Aspektorientierung ein fester Bestandteil der zukünftigen Informationstechnologie sein wird. Es ist unser professioneller Anspruch, diese neuen Entwicklungen zu beobachten, Risiken abzuschätzen und dort anzuwenden, wo sie unsere Vision der idealen Softwareentwicklung voranbringt.



Was ist orchideo?

Unser Ansporn ist die ideale Kundenlösung und unsere Vision die ideale Softwareentwicklung, welche es uns erlaubt, diese ideale Kundenlösung zu realisieren. orchideo ist unser Ansatz, um dem Problem der immer komplexer werdenden Software zu begegnen und der Vision der idealen Softwareentwicklung näher zu kommen.

Wir verbinden die Konzepte von modellgetriebener und aspektorientierter Softwareentwicklung in einem Satz von Werkzeugen, Frameworks und Bibliotheken mit dem Ergebnis, optimal strukturierte Software auf hohem Abstraktionsniveau entwickeln zu können. Unser Fokus liegt dabei auf klassischen Business Applications, also Anwendungen zur Verwaltung von Daten, die sowohl interaktiv als auch im Batch verarbeitet werden und die sich in die IT-Landschaft eines Unternehmens eingliedern und mit anderen Anwendungen kommunizieren.

Verschiedene, spezialisierte Aufgabenbereiche verlangen spezialisierte Lösungen. Daher ist orchideo keine monolithische Alles-Oder-Nichts Gesamtlösung, sondern eine Suite aus Einzelkomponenten. Diese Aufteilung ermöglicht uns eine flexible Weiterentwicklung der Einzelteile mit agiler Releaseplanung und schneller Reaktion auf das sich rasch ändernde IT-Umfeld. Dennoch legen wir besonderes Augenmerk auf eine enge Integration der Komponenten, denn auch hier gilt, dass das Ganze mehr ist als die Summe seiner Teile.

Das Gesamtpaket von orchideo besteht aus nachstehenden Komponenten, welche im Folgenden näher beschrieben werden:



orchideo|engine: Der aspektorientierte Microkernel, der die Grundlage zur flexiblen Kombination und Erweiterung von Features darstellt.



orchideo|objects: Übernimmt das Management von persistenten Daten als Abstraktionsschicht für die Geschäftslogikebene.



orchideo|views: Beschäftigt sich mit dem Management von Benutzeroberflächen als Abstraktionsschicht für die Präsentationsebene.



orchideo|documents: Begleitet das Projekt von der Analysephase bis zum Betrieb durch automatisierte Erzeugung von Dokumenten aus den Modellen in allen Projektphasen.



orchideo|ui-designer: Beschreibt Aufbau und Inhalte von GUIs und ermöglicht es, diese mit dem Geschäftsmodell von orchideo|objects zu verbinden.

Der hier gebotene Überblick über orchideo zeigt die aktuell verfügbaren Komponenten. In nächster Zeit kommen weitere Komponenten wie z. B. orchideo|workflows und orchideo|services zur Abbildung von Geschäftsprozessen und Integration von SOA Services hinzu. Bitte fragen Sie uns über Details zum aktuellen Stand oder informieren sie sich auf unserer Webseite!

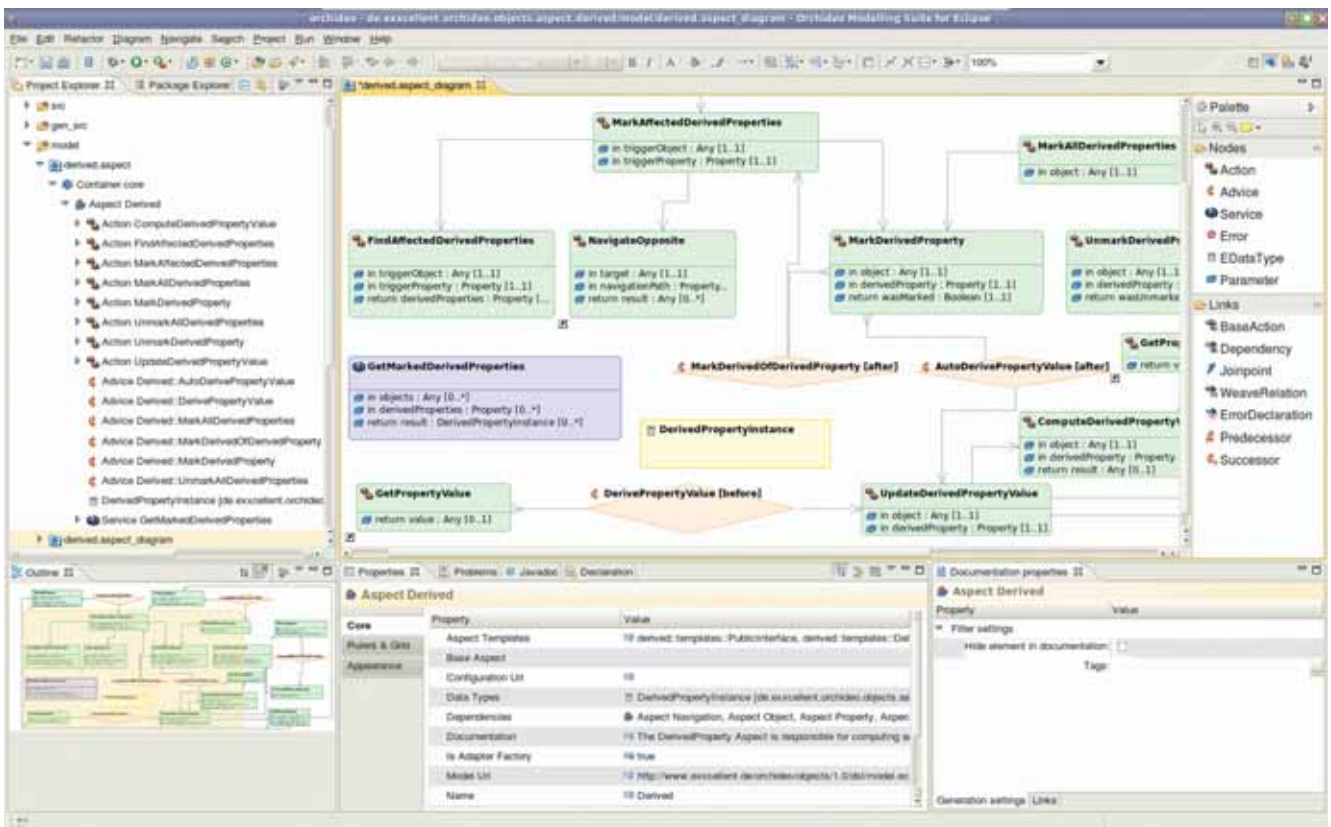
Der Kern: orchideo|engine

Das Prinzip der orchideo|engine beruht darauf, dass Funktionen innerhalb der Applikation als Aspekte definiert werden. Welche Features das sind, ist zunächst einmal offen. orchideo|objects bietet Features wie Persistenz, Berechnung abgeleiteter Attributwerte und Prüfung von Geschäftsregeln auf beliebigen Objekt-Netzen. In Kundenprojekten realisieren wir zusätzlich z. B. ein Journal-Feature, das Datenänderungen durch Benutzer-eingaben mitprotokolliert. Jedes dieser Features wird als Aspekt formal in einem Modell definiert. Dazu dient unser grafischer Aspekteditor auf eclipse EMF/GMF Basis, aus dem heraus auch gleich automatisch die Grundimplementierung der Aspekte generiert wird.

Kernbestandteil der Aspekte sind Actions und Advices. Actions werden als Rechteck symbolisiert und stellen eine konkrete ausführbare Einheit dar, die ein Aspekt als Funktionalität anbietet, z. B. für das Setzen eines Attributwertes oder das Löschen eines Objektes.

Advices werden als Raute dargestellt und verbinden Actions nach dem Muster „Immer wenn Action A passiert, soll (vorher oder nachher) Action B ausgeführt werden.“ Damit kann man Actions aus verschiedenen Aspekten miteinander in Beziehung setzen, sodass z. B. nach jedem Setzen eines Attributwertes die Geschäftsregeln für dieses Attribut validiert werden. Das Setzen des Attributwertes muss dabei nicht wissen, dass und ob überhaupt eine Validierung statt findet. Jeder Aspekt kann unabhängig und eigenständig definiert und implementiert werden.

Die orchideo|engine erlaubt eine flexible Kombination und Konfiguration von Aspekten und deren Advices. Damit kann jedes Projekt seinen individuellen Satz an Funktionalitäten zusammenstellen und dabei neue Aspekte ergänzen oder auch die Funktionsweise bereits vorhandener Aspekte durch Vererbung an die eigenen Bedürfnisse anpassen. Diese flexible Kombination und Erweiterung von Features ist ein Kriterium, das ein herkömmliches, nicht aspektorientiertes Framework, kaum ermöglichen kann. In einer dynamischen und variablen IT-Landschaft ist aber genau sie die Voraussetzung, um agil auf individuelle und sich ändernde Anforderungen eingehen zu können.



Trotz dieser Mächtigkeit bleibt das komplexe Ganze in seiner Summe einfach und beherrschbar, da Aspekte lediglich in einem einzigen Konstrukt (den Actions) untereinander interagieren können, die zudem noch als formale Modelle mit einfacher graphischer Visualisierung vorliegen.

Neben der eclipse-basierten Toolunterstützung arbeitet die orchideo|engine größtenteils zur Laufzeit, indem sie Actions ausführt, Advices aufruft und damit andere Actions verwebt. Alle Aktionen können dabei mit einem Undo vollständig und stufenlos rückgängig gemacht werden. Der Betrieb ist dabei sowohl in einer OSGi/Equinox Umgebung als auch standalone möglich.

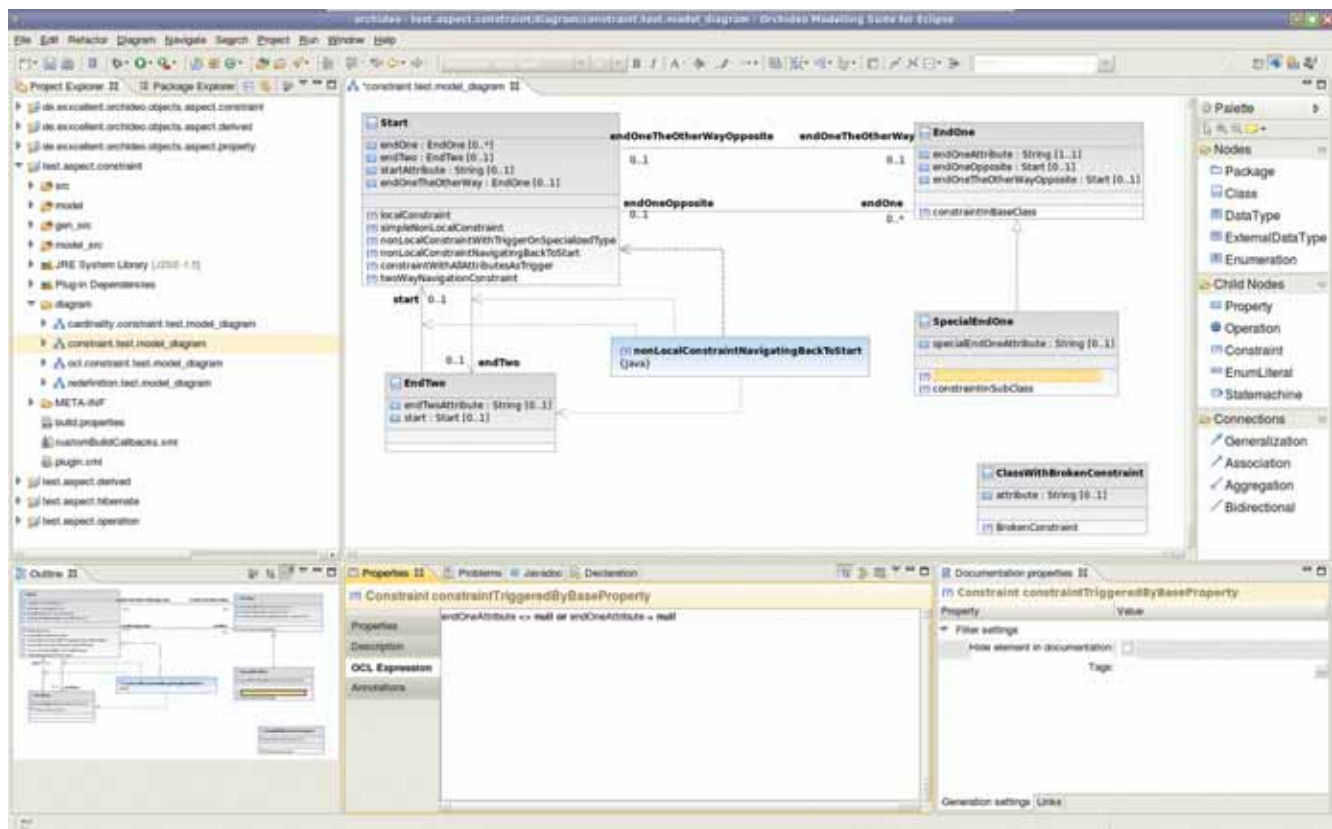
Objekte managen: orchideo|objects

Die Flexibilität der orchideo|engine haben wir genutzt, um orchideo|objects zu erstellen. Entstanden aus unserem pleXX Business Object Framework können wir die dort gebotenen

Features nun als frei kombinierbare und erweiterbare Aspekte zur Verfügung stellen. In Verbindung mit einem auf Effizienz ausgelegten grafischen Modelleditor bauen wir damit auf eine zukunftsfähige Plattform, mit der wir den hohen Anforderungen an unsere kundenspezifischen Businesslösungen gerecht werden können.

Modellierung mit IDE Komfort

Modellgetriebene Softwareentwicklung bedeutet, die Arbeit der Softwareingenieure vom Code-Editor mehr und mehr hin zum Modelleditor zu verlagern. Aus Management- und Projektleitersicht liegen die Vorteile auf der Hand. Jedoch begegnen viele Entwickler, trotz aller Effizienzsteigerung, der Arbeit mit dem Modell eher mit Skepsis. Die bisherigen Erfahrungen mit Modellierungswerkzeugen, manuellen und langwierigen Generierungsprozessen und umständlicher Fehlerdiagnose führen zu einer „gefühlten Ineffizienz“ des einzelnen Entwicklers. Es ist uns daher ein besonderes Anliegen, den Komfort, den moderne IDEs im Code-Editor bieten, auch in der Modellierung zur Verfügung zu stellen. Im Idealfall sollte kein Unter-



schied mehr zwischen der Arbeit im Code und der Arbeit im Modell zu spüren sein. Der Wechsel vom Modell zum Code und zurück muss fließend sein.

Diese Überlegungen flossen in den grafischen Modelleditor von orchideo|objects ein. Das Bearbeiten von Attributen und Methoden wird durch Content-Assist und Auto-Complete unterstützt. Mit gleichem Komfort können Geschäftsregeln und Berechnungsregeln für Attribute in einem OCL-Editor definiert werden. Fehler werden sofort im Diagramm angezeigt und führen nicht zu kryptischen Meldungen im Generator oder gar zu unkompilebarem Code. Der Generator selbst wird automatisch beim Speichern des Modells aufgerufen, ähnlich wie der Java-Compiler automatisch beim Speichern von Source Dateien aufgerufen wird. Er läuft im Hintergrund, sodass die Arbeit des Entwicklers nicht durch manuelles Starten und Warten auf den Generator blockiert wird. Ähnlich effizient erfolgt der Weg vom Code zurück in das Modell. Mit einem „Open Model Element“ Dialog lässt sich jedes Modellelement genau-so einfach auffinden, wie es der „Open Type“ Dialog für Java Typen ermöglicht.

Features für POJOs

Die Aspekte von orchideo|objects kümmern sich nach der Modellierung um die Verwaltung der Objekte zur Laufzeit. Für den Entwickler unterscheiden sich diese zunächst einmal kaum von herkömmlichen „Plain Old Java Objects“ (POJOs). Lediglich im Hintergrund sorgt orchideo|objects dafür, dass jedes Lesen und Setzen von Attributwerten und jeder Aufruf von Methoden über die Basisaspekte laufen. Darauf aufbauend lassen sich dann Objektbeziehungen effizient verwalten, Geschäftsregeln prüfen, abgeleitete Attributwerte berechnen, Objekte persistent speichern und vieles mehr. Alles durch einzeln an- und abschaltbare Aspekte. Diese umfangreichen Funktionalitäten ebnen den Weg für applikationsspezifische Aspekte. Dadurch wird die optimale Balance aus Wiederverwendung und Individualität erreicht.

Benutzeroberflächen gestalten: orchideo|views

Benutzeroberflächen stellen in datenzentrierten Anwendungen einen Großteil der Lösung dar. Durch hohen Bedienungskomfort und komfortable Benutzerführung werden sie zunehmend komplexer. Oberflächen müssen für Poweruser effizient und für den weniger versierten Nutzer intuitiv sein. Technologischer Fortschritt erlaubt maßgeschneiderte Benutzerinteraktion – von der simplen Stammdatenerfassung bis zur Bearbeitung von komplexen fachlichen Vorgängen. Von der Massendatenverarbeitung bis zur detaillierten Einzelfallbeurteilung. Die Oberfläche ist die Repräsentation zum Nutzer – sie bestimmt die Qualität und Zielerreichung der Lösung wesentlich. Daher ist ein wichtiger Baustein in einem erfolgreichen Softwareentwicklungsprozess die Entwicklung geeigneter Methoden, um für die fachlichen Anforderungen effizient, passgenaue Anwendungsoberflächen zu entwickeln.

Die Vielfalt der Präsentations- und Benutzerinteraktionsmöglichkeiten verlangt eine besonders hohe Agilität bei der Gestaltung und Weiterentwicklung von Oberflächen.

orchideo|views ist ein hoch-flexibles, plattformunabhängiges Baukastensystem für die effiziente Entwicklung der GUIs von datenzentrierten Enterprise-Applikationen.

Es adressiert die gesamte Bandbreite der Oberflächenentwicklung: von der nahezu codierfreien GUI-Modellierung für das Rapid Prototyping bis hin zur umfassenden Individual-Lösung mit spezifischem Backend, maßgeschneiderten Darstellungsformen und Fachmodellen.

Durch sein einzigartiges, mehrschichtiges Baukastensystem vereint orchideo|views die Vorteile eines hohen Grades an Wiederverwendung mit gleichzeitig größtmöglicher Flexibilität. Konkrete Technologien und Modelle werden durch die Auswahl von Bausteinen festgelegt und optimal unterstützt.

orchideo|views fokussiert sich darauf, Mittel und Wege anzubieten, um verschiedenste Modelle effizient integrieren zu können. Darauf aufbauend können für den GUI-Bereich geeignete Abstraktionsschichten genutzt werden.

Dieser Frameworkansatz umfasst die gesamte Spannweite von der Integration in geeignete Modelleditoren bis hin zur individuell ausgeprägten Programmierumgebung (API).

Als bewährte Modell-Inhalte zur direkten Nutzung innerhalb der GUI erweisen sich in der Praxis unter anderem:

- > Meta-Informationen zum Fachmodell (Namen, Beschreibungen, Semantik)
- > Fachliche Prüfbedingungen für die Datenerfassung
- > Regeln für Prozess- und Dialogabläufe
- > Beschreibungen von Rollen, Sichtbarkeiten und Zugriffsrechten
- > Fachliche Struktur und Inhalte von Dialogen speziell für explorative Prototypen

Die Aufgabe der Anbindung der Modellfeatures übernehmen austauschbare Bausteine, welche wir als nächstes näher erläutern.

Das orchideo|views Baukastensystem

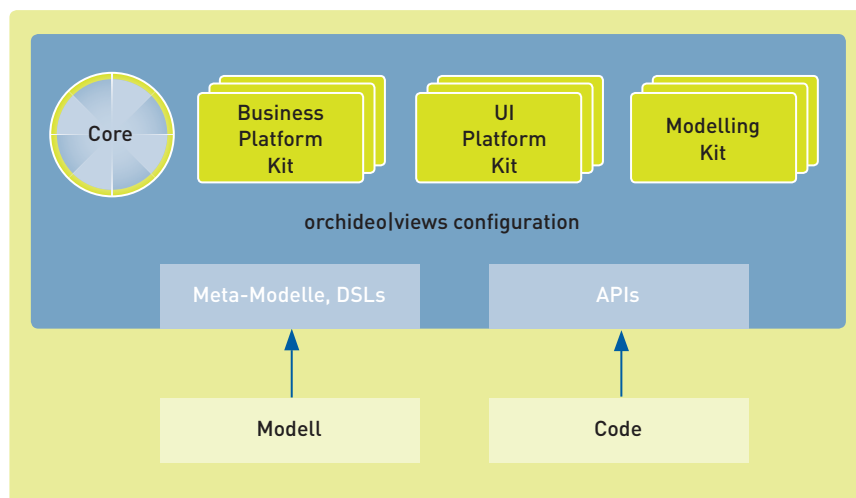
Die Benutzeroberflächen typischer Enterprise Applikationen bestehen oft aus einer Kombination von vielen einfachen und

funktionsseitig sich ähnelnden Teilen, gepaart mit individuellen und oft komplexen Lösungsteilen. Großes Einsparungspotential besteht in der Bereitstellung von wiederverwendbaren und einfach auszutauschenden Lösungs-Bausteinen.

Für maximale Effizienz und Flexibilität bietet der Baukasten von orchideo|views neben Fertigteilen auch anpassbare Halbprodukte und Vorlagen. Diese kommen typischerweise in höheren Schichten zum Einsatz, da dort ein sehr hoher Spezialisierungsgrad erforderlich ist.

Zu den Schichten gehören die unterschiedlichen Ausprägungen der **orchideo|views configuration**. Eine orchideo|views configuration stellt die oberste Framework-Schicht innerhalb einer Anwendung dar. Gegen diese Schicht arbeiten der Generator (DSL-abhängig) und die Entwickler. orchideo|views Konfigurationen definieren damit für die jeweilige Anwendungsdomäne das jeweils spezifisch gehaltene Framework.

Diese hoch spezialisierte orchideo|views configuration in Kombination mit den abstrakt gehaltenen tieferen Ebenen **orchideo|views core** und **orchideo|views kits** erlauben eine



orchideo|views Baukastensystem

bislang unerreichte Kombination an Flexibilität, Detailkontrolle und Wiederverwendung. Allgemein gültige Teile arbeiten gegen die plattformunabhängigen, tieferen Schichten. Dem Entwickler und DSL Generator stehen für die jeweilige Problemdomäne optimierte Schnittstellen mit voller Detailkontrolle zur Verfügung.

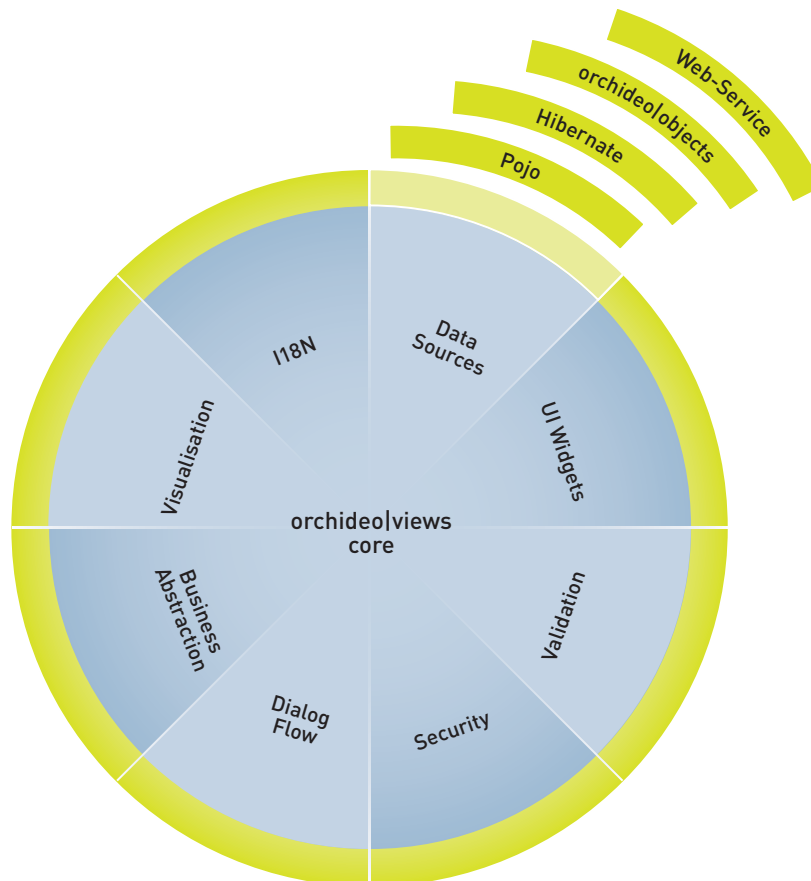
orchideo|views configurations sind Zusammenstellungen gewählter orchideo|views kits und Modellierungsplattformen kombiniert mit einer dazu passenden, domänenspezifischen API. Sie nehmen Querschnittseinstellungen in Bereichen wie Dialogflußsteuerung, Layout, Datenanbindung, Internationalisierung, Validierung und weiteren vor und lassen die domänenspezifisch getroffenen Annahmen in ihre vereinfachte API einfließen. Damit stellt orchideo|views configuration die vom Entwickler vorrangig verwendete Schnittstelle dar, kapselt die für das Projekt getroffenen Gemeinsamkeiten und bildet somit das Bindeglied zwischen der anwendungsspezifischen Problemdomäne und der flexiblen Frameworkstruktur.

orchideo|views core

Der Kern orchideo|views core umreißt den für alle Applikationen gemeinsamen Rahmen und stellt eine plattformunabhängige Basis für alle weiteren, darauf aufbauenden Teile dar. Sein Inhalt umfasst die Definition der gemeinsamen Grundlagen für die Oberflächentechnologie (Web/Desktop), die Backend-Anbindung, die Ablaufsteuerung bzw. das zugrunde liegende Objektmodell und Querschnittsfunktionalitäten wie Internationalisierung, Objektvalidierung und -formatierung.

orchideo|views kits

Zur praktischen Verwendung wird der orchideo|views core um ausgewählte Kits ergänzt, welche sowohl Adapter für die unterschiedlichste Basis- und Backendtechnologien als auch Lösungsbausteine für wiederkehrende, fachliche Problemstellungen bereitstellen.



orchideo|views core Übersicht

Eines der grundlegendsten Bauteile stellt das **UI Platform Kit** dar. Es bildet die Brücke zur GUI Basistechnologie. Neben den bestehenden Lösungen für Desktop- und Web-Umgebungen können auch neue Plattformen jederzeit über neue Kits angebunden werden.

Ein weiterer, wichtiger Baustein ist das **Business Platform Kit**. Es ist das zentrale Bindeglied, um das über eine DSL oder einer anderen Sprache abgebildete Fachwissen für die GUI-Ebene verfügbar zu machen. Es ermöglicht die Bezugnahme auf das fachliche Geschäftsmodell sowie die Integration der ergänzenden Aspekte wie definierten Gültigkeitskriterien, Sicherheitsrichtlinien und einzuhaltende Prozessabläufe.

Konzeptionen in orchideo|views

MDSD in der Oberflächenentwicklung

Massives Potential zur Effizienzgewinnung in der Oberflächenentwicklung eröffnet sich durch den Zugriff auf Abstraktion mittels DSLs. Das Ziel dieser Abstraktion wurde in den vorangegangenen Ausführungen umfassend beschrieben.

Die modellgetriebene Softwareentwicklung stand bisher vorrangig im Fokus der Modellierung von Datenstrukturen und fachlichen Abläufen. Im Bereich der GUI stellt sich somit natürlicherweise die Frage nach geeigneten Formen der Modellierung.

Bisherige Ansätze zur vollständigen Beschreibung einer GUI samt Darstellung, Verhalten und Funktionalität über ein Modell sind gescheitert. Dies gründet vorrangig in der großen Bandbreite an variablen Aspekten einer GUI, welche die Findung einer allgemein gültigen Abstraktion und damit eines abgeschlossenen Modells für die GUI verhindern.

Bereits die Themen Layout und Darstellung bilden für sich ein eigenes, von der Fachdomäne völlig unabhängiges Lösungsuniversum. Modellierung ist hierfür auf Grund der unterschiedlichen technologischen Umfeldern nur schwer umsetzbar. Ein modellgetriebener Ansatz für die GUI-Entwicklung muss sich auf die elegante und einfache Integration der Fachmodelle konzentrieren. Gleichzeitig müssen Freiheiten in der Gestaltung und der technologischen Plattform gegeben sein. Es gilt also, auf Basis der zugrunde liegenden, spezifischen Fachmodelle, schnell individuelle Darstellungssichten aufbauen und realisieren zu können.

Datenanbindung und Validierung

Für eine GUI müssen Daten aus den verschiedensten Quellen ausgelesen, in die passende Form transformiert und dargestellt werden. Der gleiche Weg, lediglich in gegenläufiger Richtung und erweitert um zusätzliche Eingabeprüfungen, ist für bearbeitende Sichten notwendig. Da die Benutzeroberflächen von Enterprise-Applikationen in der Regel stark datenzentrischen Charakter besitzen, stellt dieser Themenkomplex eines der wichtigsten Kernfunktionalitäten dar.

orchideo|views selbst stellt keine einschränkenden Anforderungen an die Inhalte und Quellen der zu realisierenden Benutzeroberflächen. Über die in den Kits gelieferten Adapter lässt sich das gesamte Spektrum vom orchideo|objects, O/R-Mappern (Hibernate, Toplink, etc.) über Webservices bis hin zu reinen Java-Objekten (POJOs) nahtlos integrieren.

Das Auslesen, die Aufgliederung und Transformation der erhaltenen Quelldaten in die einzelnen Bestandteile eines Dialogs (z. B. eines Textfeldes) erfolgt über modellbasierte Abbildungsregeln. Über dieses Regelwerk weiß jeder Teil der GUI, von woher er seine Daten beziehen, wie diese darzustellen und gegebenenfalls zurückzuschreiben sind, ohne Details der Backendtechnologie zu kennen. Durch den enthaltenen Modellbezug hat sie auch vollständigen Zugriff auf alle anderen Inhalte des Modells und kann somit Beschriftungen, Eingabe- und Berechtigungsprüfungen und andere Aspekte automatisch berücksichtigen.

Strukturierung

Eine klare Strukturierung und damit Trennung der Geschäftslogik von der Präsentationslogik sind Grundvoraussetzung für eine gute Wartbarkeit der Oberflächen. orchideo|views baut auf eine strikte Trennung der prozess- und businessrelevanten Teile von den darstellungsrelevanten Teilen.

Die Implementierung der Fachlogik in einer orchideo|views Oberfläche hat keinerlei direkten Zugriff auf die Präsentation. Die Fachlogik hält alle dargestellten Daten und empfängt von der Präsentationslogik die ausgelösten, fachlichen Ereignisse. Über technologieunabhängige Kommandos kann sie wiederum allgemeine Operationen auf der Präsentationsschicht auslösen und kontrolliert damit sowohl den einzelnen Dialog als auch die Dialogablaufsteuerung auf Prozessebene. Somit hält die Geschäftslogik sämtliche geschäfts- und prozessrelevanten Zügel in der Hand und ist dennoch unabhängig von den Details der Präsentation.

Die Präsentationsschicht baut wiederum auch auf einer weiteren, klaren Strukturierungsebene auf. Anstatt direkt gegen die Mittel der Oberflächentechnologie zu arbeiten, nutzt sie die von der orchideo|views Konfiguration angebotenen, teils domänenspezifischen Mittel zur Oberflächengestaltung. Über deklarative Mittel werden Struktur, Inhalt und Funktionalität eines Dialogs näher beschrieben. Die Sichtbarkeit, Layout und Darstellung einzelner GUI-Elemente werden über diese Regeln automatisch angesteuert. Diese Infrastruktur gliedert sich in einen vom Framework gesteuerten Lebens- und Interaktionszyklus, welche als Vermittler zwischen den verschiedenen Framework-Services (Validierung, Security, Datenanbindung, ...) und der realisierten Implementierungslogik auftritt. Trotz dieses hohen Grades an Abstraktion und Vereinfachung steht dem Entwickler zur Realisierung flexibler Detaillösung jederzeit der Durchgriff auf die Mittel der Oberflächentechnologie offen.

Prozessbasierte Ablaufsteuerung

Die Dialog-Ablaufsteuerung einer orchideo|views Applikation obliegt vollständig der darstellungsunabhängigen Prozessschicht. Sie bindet sich nahtlos in die von der Fachlichkeit beschriebenen Prozesse ein. Je nach Anforderung kommen verschiedene Steuerungsimplementierungen zum Zuge und bilden von einfachen, expliziten Abläufen über Wizard-Steuerungen bis hin zu komplett modellbasierten Ablaufsteuerungen die unterschiedlichsten Strategien und Anwendungsverhalten ab.

Dokumente erzeugen: orchideo|documents

Die Dokumentationsgenerierung aus UML Modellen hat bei eXXcellent solutions eine langjährige Tradition. In der Vergangenheit war die Generierung auf UML Modelle aus einem proprietären Modellierungswerkzeug beschränkt. Es hat sich gezeigt, dass dieser Rahmen für die Zukunft zu eng ist und wir eine deutlich breitere Palette an Quellmodellen und Zieldokumentationen für die Generierung benötigen. Beispielsweise fallen durch den zunehmenden Einzug der Modellierung in die Entwicklung eine Reihe von zusätzlichen Dokumentationen, Benutzerhandbücher, Änderungsdokumentationen usw. an.

Die Herausforderung ist, einen Dokumentationsgenerator zu schaffen, der beliebige EMF-basierte Modelle in wohlstrukturierte Dokumente überführen kann, ohne sich dabei auf ein bestimmtes (eclipse-basiertes) Tool festlegen zu müssen.

Fast alle Modellierungstools bieten die Möglichkeit, deren Modellinhalte in Form von Reports in lesbare Dokumente zu exportieren. Unser Anspruch an Dokumente geht aber deutlich über die Inhalte eines Reports hinaus. Es geht nicht nur darum, Modellinhalte vollständig auszugeben. Die Herausforderung besteht vor allem darin, dies in einer verständlichen, zielgruppenbezogenen und für den Leser ansprechenden Form zu tun.

Die folgenden Kriterien sind uns bei der modellbasierten Generierung von Dokumentation besonders wichtig:

- > Möglichkeit der vollständigen Verwendung von Texten und Grafiken aus den Modellen
- > Einbringen von ergänzenden, nicht aus den Modellen stammenden Texten an definierten Stellen im Dokument
- > Wohlstrukturierte Darstellung der Modellinhalte
- > Einfache, schnelle und reproduzierbare Generierung der Dokumente
- > Umfangreiche Konfigurationsmöglichkeiten
- > Ansprechendes Layout mit sauberer Formatierung
- > Möglichkeiten, die Dokumentenstruktur individuell anzupassen

Strukturierung der Dokumentationsgenerierung

Durch den zunehmenden Einzug von DSLs in der Analyse und Entwicklung werden zur Beschreibung von Software immer mehr unterschiedliche Modelle und DSLs verwendet. Ein Dokument kann damit eine Komposition der Inhalte verschiedener Modelle sein. Die Modelle können zudem unterschiedliche Meta-Modelle besitzen.

Der Transformationsprozess von den Ausgangsmodellen in die Dokumentation umfasst deshalb mehrere Schritte. Bei den Zwischenschritten entstehen i.d.R. Zwischenmodelle, die als konsolidierte Grundlage für weitere Verarbeitungsschritte dienen.

Bei der Nutzung von Zwischenmodellen wird ein Xtext basierter Template-Mechanismus, der somit ebenfalls modellbasiert ist, zur weiteren Überführung verwendet. Dieser Ansatz wurde gewählt, um eine saubere Trennung und damit Strukturierung der einzelnen Überführungsschritte zu gewährleisten.

Der Transformationsprozess zur Generierung von Dokumenten besteht aus einzelnen austauschbaren Komponenten, die jede für sich abgegrenzte Funktionen im Generierungsprozess erfüllen. Die Komponenten sind aktive Elemente und je nach Funktion sind unterschiedliche Eingriffs- und Konfigurationsmöglichkeiten vorhanden.

Zielformate

Genau wie die flexible Möglichkeit der Dokumentationsgestaltung ist es notwendig, unterschiedliche Zielformate für unterschiedliche Einsatzzwecke zu bedienen. Mit dem Erzeugen des Zieldokuments ist für den Benutzer der Prozess oft noch nicht beendet. Er möchte das Dokument zum Teil einer größeren Dokumentation machen, zusätzliche Artefakte bzgl. des Dokuments erzeugen oder andere Schritte ausführen. Für die weitere Verarbeitung in einem Textverarbeitungsprogramm erzeugen wir ODT, PDF, HTML oder DocBook XML. Natürlich ist die Bedienung der Zielformate erweiterbar, so dass auch Formate wie OOXML generiert werden könnten.

Infrastruktur und Architektur

Die Plattform eclipse stellt eine Basisinfrastruktur zur Verfügung, um eigene Funktionen als Plugin zu integrieren bzw. vorhandene Funktionen zu erweitern. Über Extension-Points, OSGI-Services und die eclipse Plugin-Struktur können über individuelle Modelladapter verschiedenste unabhängige Modelle als Quellen zur Dokumentationserstellung herangezogen werden.

Quellen der Dokumentation

Die zu generierende Dokumentation bezieht ihre Inhalte aus drei möglichen Quellen:

- > domainspezifisches EMF-Modell
- > GMF-Diagramm (als spezielles EMF-Modell) für Diagramme
- > Zusätzliche Dokumentationseinstellungen für einzelne Modellelemente (die intern ebenfalls als EMF-Modell abgelegt werden)

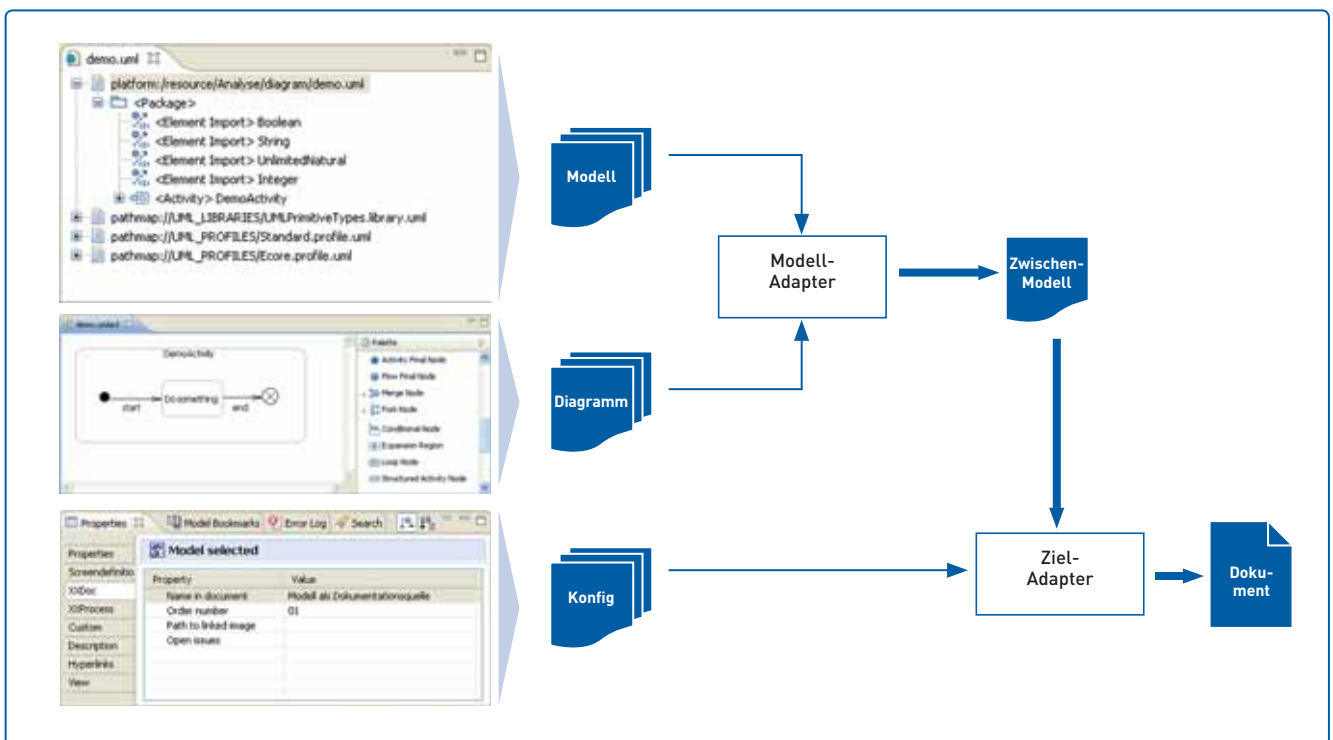
Das domainspezifische EMF-Modell stellt den primären Content der Dokumentation dar. GMF- oder andere grafische Editoren beinhalten die Diagramme zum Modell. Über einen dem Quellmodell (EMF/GMF) entsprechenden Modelladapter wird der

Inhalt in ein einheitliches Zwischenformat überführt. Neben den fachlichen Inhalten enthalten die Quellmodelle auch orchideo|documents spezifische Properties. Diese sind Parameter zur Dokumentationssteuerung, um auf feingranularer Ebene den Erstellungsprozess beeinflussen zu können. Diese zusätzlichen Properties werden zentral vom Dokumentationsgenerator verwaltet und in die unterstützten Modelle eingebunden. Wichtig für die Abbildung der Modellelemente ist die Möglichkeit, dass eigene Modelladapter erstellt bzw. vorhandene adaptiert werden können.

Modelladapter werden für folgende Anwendungsfälle benötigt:

- > Anbindung von beliebigen EMF-Modellen
- > Anbindung von beliebigen GMF-Diagrammen
- > Anbindung von beliebigen DSLs (z. B.: eclipse UML2, BPMN, GUI Prototype Diagramme, orchideo|objects, weitere eigene DSLs...)
- > Interpretation der Verzeichnisstruktur von eclipse-Projekten

Die folgende Grafik illustriert die Zusammenhänge der unterschiedlichen Komponenten für die Dokumentationsgenerierung.



Komponenten für die Dokumentengenerierung

Prozess der Dokumentationsgenerierung

Die Dokumentationsgenerierung besteht aus drei Schritten:

1. Parametrierung eines Generierungsvorgangs
2. Transformieren der EMF-Modelle in das DocBook kompatible Zwischenmodell
3. Erzeugen des Zielformats aus dem Zwischenmodell

Bevor aus den Modellen Dokumente erzeugt werden können, muss definiert werden, welche Modellbereiche in je ein Dokument überführt werden sollen und welche Inhalte aus dem verwendeten Modellbereich im Dokument erscheinen sollen. Diese Parametrierungen werden pro generiertem Dokument getroffen und in einem sogenannten „Set“ verwaltet.

Nach dem Start der Dokumentengenerierung wird als erstes die Projektstruktur ausgewertet und die enthaltenen Modelle mit dem jeweiligen entsprechenden Modell-Adapter in das Zwischenmodell überführt. Der jeweilige Modell-Adapter transformiert sein EMF-Modell in das DocBook-Zwischen-Modell. Hierbei werden bereits Filter angewandt, so dass der Umfang des DocBook Zwischen-Modells dem Umfang des Zieldokuments entspricht. Durch die Einbettung beliebiger EMF-Modelle in das Projekt-Modell können Modelle geschachtelt und in beliebiger gemischter Anzahl verarbeitet werden. Als Ergebnis liegt ein DocBook-kompatibles Modell vor, in dessen Struktur alle zu dokumentierenden EMF-Modelle überführt wurden. Anschließend wird aus dem Zwischenmodell über einen Target-Adapter (ist spezifisch für ein spezielles Zielformat, z. B. ODT, HTML, PDF, ...) das Zieldokument im entsprechenden Format erzeugt.

Das Generieren eines Dokuments kann auch ohne GUI im Batch erfolgen. Damit kann die Aktualisierung der Dokumentation z. B. in einen Buildprozess integriert werden.

Definition einer Dokumentation

Um eine bestimmte Dokumentation zu erhalten, definiert der Benutzer ein Set, das alle Eigenschaften beschreibt. Solch ein Set von Einstellungen umfasst u.a. folgende typische Angaben:

- > Der Name der Dokumentation. Dieser bestimmt auch den Namen des Zielartefakts
- > Das Ablageverzeichnis für die Dokumentation
- > Den Scope der Generierung. Dies kann ein beliebiger Einstiegspunkt in ein spezielles DSL-Modell oder das Projekt-Dokumentationsmodell (z. B. ein Verzeichnis) sein
- > Die Auswahl der zu verwendenden Modell-Adapter für die einzelnen Modelltypen
- > Die Konfiguration von Filtern, z. B. dass nur bestimmte Diagramme dokumentiert werden
- > Das Image-Format, wenn das Zielformat Bilder unterstützt
- > Die im generierten Dokument verwendete Sprache
- > Zusätzliche, für die verwendeten Modell-Adapter spezifische Optionen
- > Weitere zielformatspezifische Optionen

Konfiguration

Es ist unbedingt erforderlich, dass der Anwender an möglichst vielen Stellen eingreifen kann, um die eigenen Anforderungen umsetzen zu können. Die Dokumentationsgenerierung bedingt mehrere Einstellungen, damit das gewünschte Resultat erzielt werden kann. Diese können diverse Verarbeitungsschritte bei der Generierung und natürlich den Inhalt und das Aussehen des Dokuments betreffen.

- > Für jedes Modellelement muss definiert werden können, ob es dokumentiert werden soll oder nicht. Diese Information muss während der Erarbeitung des Modells definierbar sein. Diese steuernden Angaben werden strikt von den eigentlichen Modellinformationen getrennt. Das Modell enthält fachliche Informationen, die Steuerinformationen für die Dokumentationsgenerierung dagegen sind rein technisch. Auch hier halten wir uns an den Grundsatz der separation of concerns.

- > Bestimmte Dokumentationsbestandteile können nur zur Laufzeit durch komplexen Code bestimmt werden. Dies ist über eine API zur Einbindung von Java-Aufrufen realisiert (z. B. rückwärtiges Navigieren von unidirektionalen Beziehungen, präventives Sammeln von Modellinhalten für eine spätere Ausgabe, ...).
- > Es können dedizierte, einzelne Bereiche der Dokumentationsgenerierung an individuelle Bedürfnisse bzgl. Inhalt und Struktur angepasst werden. Hierfür sind explizite Erweiterungspunkte vorgesehen, die vom Anwender unter Verwendung einer einfachen Xtext basierten Templatesprache benutzt werden können, um die generierten Dokumente an die eigenen Bedürfnisse anzupassen.

Filterung von Modellelementen

Aus einem Modell können verschiedene Dokumente erzeugt werden. Diese können sich in Inhalt und Umfang unterscheiden. Es ist möglich, dass der Benutzer Markierungen (Tags) an Modellelementen vergibt, die er bei der Erstellung eines Dokumentations-Sets als Filterkriterium verwendet. Zu den unterstützten Filtern gehören:

- > Nur Elemente dokumentieren, die bestimmte Eigenschaften besitzen (z. B. nur Attribute von Klassen und keine Operationen, keine Elemente, die das Tag „HideInDoc“ besitzen, nur Diagramme, die das Tag „Architekturdiagramm“ besitzen, ...)
- > Filter, der nach einer bestimmten Strategie Modelle und/oder Diagramme auswählt (z. B.: nur orchideo|objects Modelle, nur UML2 Tools UseCase Diagramme, ...)
- > Namepattern-Filter mit dem Elemente ausgeschlossen werden können, die ein bestimmtes Namensschema erfüllen, z. B. alles aus *impl* Verzeichnissen

Konfigurationseigenschaften von Modellelementen und Diagrammen

Es gibt einige für alle Modellelemente und Diagramme gültige Eigenschaften, die das Verhalten der Dokumentengenerierung steuern. Dazu gehören:

- > Der Name, der für das Element im Dokument verwendet werden soll
- > Die Reihenfolge des Elements in der Dokumentation
- > Tags

- > Eine (Kurz-)Beschreibung für Modellelemente, die keine eigene Property dafür definieren
- > Ein Flag für Diagramme (GMF-Editor), ob das Diagramm dokumentiert werden soll
- > Ein Flag, ob das Element in einem Diagramm dokumentiert werden soll.

So einfach wie möglich

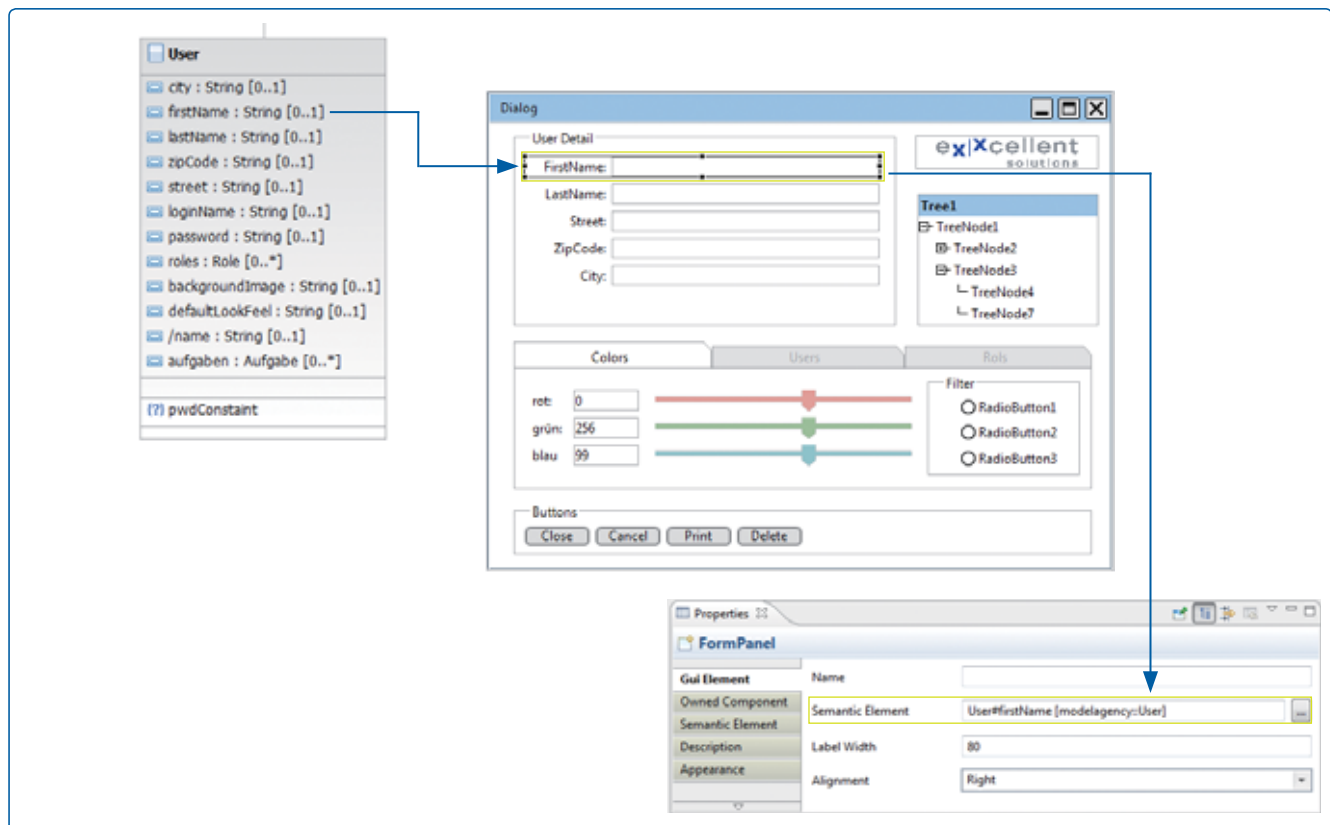
Unser Ziel ist es, die Dokumentationsgenerierung so einfach wie möglich zu gestalten. Sie muss zu jeder Zeit und von jedem Projektbeteiligten ohne großen Aufwand durchgeführt werden können. Aus unserer Erfahrung heraus wissen wir, dass in einem Projekt typischerweise mehrere unterschiedliche Teildokumente generiert werden müssen, die im Nachgang zu einem oder mehreren Gesamtdokumenten zusammengeführt werden. Wir lösen diese Aufgabenstellung, indem wir für jedes Teildokument eine eigene Konfiguration bzw. ein Konfigurations-Set pflegen. Die unterschiedlichen Sets werden mit dem Projekt versioniert und passen somit zu jeder Zeit zum Versionsstand der Modellierung. Über Batches oder über eine komfortable GUI kann die Dokumentationsgenerierung simultan für mehrere Sets durchgeführt werden. Ein spezielles Wissen hierzu ist nicht erforderlich.

Oberflächen beschreiben: orchideo|ui-designer

Wie viele andere Softwarelieferanten stehen auch wir vor der Aufgabe, Grafische User Interfaces (GUIs) zu erstellen, deren Layout mit dem Kunden abgestimmt und deren Funktionalität und Anbindung an das verwendete Datenmodell dokumentiert ist. Leider bietet weder die UML Spezifikation noch die bestehenden Modellierungstools die Möglichkeit, derartige Definitionen in einem Modell zu treffen.

Unsere Erfahrung aus Kundenprojekten hat gezeigt, dass sich derartige Entwürfe nicht sinnvoll textuell in UseCases beschreiben lassen. Auch macht es keinen Sinn, die Entwürfe der GUIs in weiteren Tools zu erstellen, da sich die Pflege der Beschreibung über die Projektlaufzeit unnötig verkompliziert.

Diese und weitere Gründe haben uns dazu bewogen, einen eigenen GMF Diagrammtyp zu erstellen, mit dem sich GUI Definitionen sehr einfach und effizient erstellen lassen. Attribute aus dem orchideo|objects Modell lassen sich über Drag & Drop in GUI Diagramm visualisieren und sind somit automatisch mit dem orchideo|objects Modell verknüpft. Die Controls des GUI Prototype Diagramms lassen sich (im Gegensatz zu anderen GUI Modellen) über Properties des Modell beschreiben. Durch den generischen Ansatz von orchideo|documents ist es nun ein leichtes, diese qualitativ hochwertige GUI-Beschreibung in den generierten Dokumenten mit auszugeben.



Beispiel eines UI-Diagramms

Komplexität beherrschen mit orchideo

Mit orchideo haben wir eine Umgebung geschaffen, die es uns ermöglicht, die Komplexität moderner Softwaresysteme zu beherrschen. Wir setzen dabei auf die modellgetriebene Softwareentwicklung, die es uns erlaubt, auf hoher Abstraktionsebene effizient zu arbeiten. Dabei legen wir großen Wert auf die enge Integration der modellgetriebenen Entwicklung mit der „herkömmlichen“ Entwicklung mit Programmiersprachen, welche weiterhin Bestandteil der Softwareentwicklung ist und es auf absehbare Zeit auch bleiben wird.

Wir setzen ferner auf die Konzepte der Aspektorientierung, um eine optimale Strukturierung der Software zu gewährleisten.

Nur mit dieser ausgeprägten Modularisierung können wir flexibel auf die aktuellen und kommenden Anforderungen an Softwaresysteme eingehen und dabei gleichzeitig die stetig wachsende Komplexität beherrschen.

Wir bündeln unser Know-how in einem Satz von Werkzeugen, Frameworks und Bibliotheken, den wir für jedes Einsatzgebiet maßschneidern können. Denn jeder Kunde mit seinem Kerngeschäft, jede Applikation und jedes Projekt sind verschieden. Es ist unser Anspruch, dieser Individualität mit unseren Produkten und Lösungen zu begegnen, um so mit idealer Software zum Erfolg unserer Kunden beizutragen.



eXXcellent solutions gmbh
www.exxcellent.de

Beim Alten Fritz 2
D - 89075 Ulm

Phone +49 (0) 731 - 55026-0
Fax +49 (0) 731 - 55026-99