

# Vom UML-Modell zum Code mit objectiF und Eclipse

von Matthias Ehlert

## Modell = Code

Mit Eclipse steht eine mächtige, gleichzeitig sehr elegante und zudem kostenlose Entwicklungsumgebung für die Java-Entwicklung zur Verfügung. Die Mächtigkeit der IDE beruht auf ihrer quasi unbegrenzten Erweiterbarkeit über so genannte Plug-ins: Jeder Entwickler kann in Eclipse beliebig neue Funktionen integrieren und externe Anwendungen anbinden. Ob für Dialog- und Webseiten-Design, für die Entwicklung von Webanwendungen oder für O/R Mapping – Plug-ins gibt es inzwischen für jeden Zweck. Auch das UML-Tool objectiF bietet ein Eclipse-Plug-in, dessen Entwurfsmöglichkeiten die Implementierungsstärken von Eclipse im Entwicklungsprozess ergänzt.

### Unterstützung in jeder Entwicklungsphase

objectiF unterstützt den Entwickler bei der fachlichen Spezifikation sowie dem technischen Entwurf seiner Anwendung. Eclipse bietet alles, was man für das effiziente Implementieren, Kompilieren, Debuggen und Testen der Anwendung braucht. Mit der Kombination von objectiF und Eclipse steht in jeder Entwicklungsphase ein passendes Tool zur Verfügung.

Die Funktionalität von objectiF geht dabei über die grafische Modellierung hinaus – das Tool unterstützt den Ansatz Modell = Code. Oder anders ausgedrückt: Die implementierte und die dokumentierte Architektur einer Anwendung sind mit objectiF über den gesamten Application Life Cycle synchron.

Um die Vorteile beider Werkzeuge nutzen zu können, müssen die Eclipse-Projekte und das UML-Modell in objectiF miteinander verknüpft werden. Dabei spielt es keine Rolle, ob man mit der Modellierung in objectiF begonnen hat oder bereits Projekte in Eclipse existieren. Sobald die Tools integriert sind, generiert objectiF aus dem UML-Modell Code für die IDE oder analysiert, bereits vorhandenen Sourcecode und erzeugt daraus UML-Diagramme. Alle Änderungen werden ins jeweils andere Tool übernommen.

Die Kopplung der Werkzeuge erfolgt dialoggesteuert, objectiF hat hierfür entsprechende Menüfunktionen. Technisch ist die Verknüpfung eines Eclipse-Projekts mit einem UML-Modell folgendermaßen

realisiert: In einem ersten Schritt wird der Workspace, in dem das Projekt liegt, mit dem objectiF Repository verknüpft. Diese Funktion kann von jedem Package in objectiF aufgerufen werden. Workspaces sind Verzeichnisse, in denen Eclipse Projekte – in der Regel in Form von Unterverzeichnissen – verwaltet. Projekte können dabei beliebige Dateien enthalten. objectiF speichert das UML-Modell in einem Repository. Neben den UML-Diagrammen enthält das UML-Modell weitere Spezifikationen.

In einem zweiten Schritt muss jetzt noch die Verknüpfung mit den Projekten in Eclipse vorgenommen werden. In der UML gibt es aber keine Projekte. Daher werden in objectiF jedes UML Package einem Eclipse-Projekt zugeordnet und die Zugehörigkeit von Klassen zu Projekten damit bestimmt. Die Klassen können – wenn gewünscht – auch einzeln Projekten zugeordnet werden.

### Modell = Code, Code = Modell

Modellbasiert entwickeln bedeutet, solange immer wieder zwischen Modell und Code zu wechseln, bis beides stabil ist. Ist ein UML Package einmal einem Eclipse-Projekt zugeordnet, kann man frei zwischen den in diesem Package und allen Subpackages enthaltenen Modellelementen des UML-Tools und dem Quellcode in der IDE navigieren.

Man kann in allen objectiF-Diagrammen direkt von einer Klasse oder deren Methoden in den Implementierungscode dieser Klasse bzw. der Methoden springen – und zwar direkt an die gewünschte Stelle. Für die Navigation vom Code ins Modell generiert objectiF Verknüpfungen auf alle UML-Diagramme in den jeweiligen Package-Ordner in der Ressourcenperspektive von Eclipse. Mit einem Doppelklick auf eine Verknüpfung öffnet sich das gewünschte UML-Diagramm.

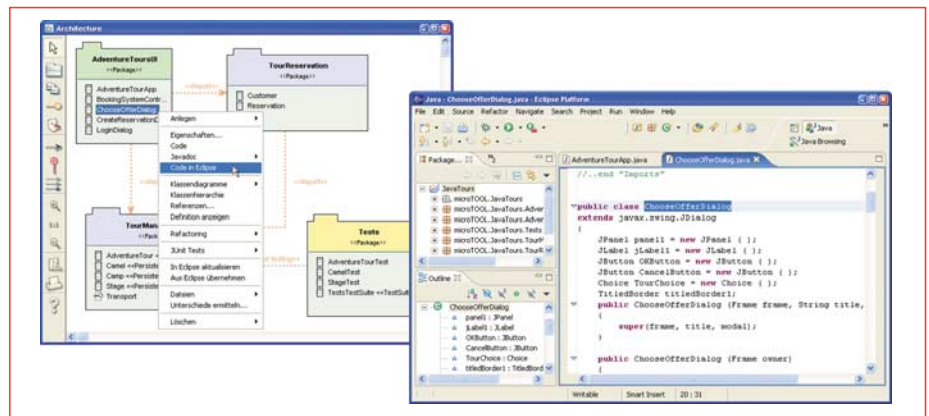


Abb. 1: Synchrone Navigation – vom Modell zum Code und umgekehrt

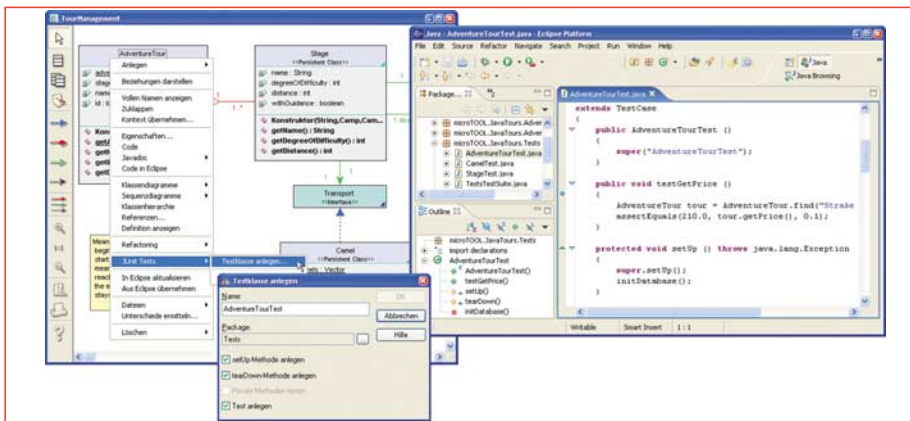


Abb. 2: Testen mit objectiF und JUnit

Bei Änderungen im Modell oder in der Implementierung lässt sich die jeweils andere Seite komfortabel aktualisieren. Änderungen im Code wirken sich in der Regel nur lokal aus. Das objectiF-Eclipse-Plugin bietet deshalb die Option, Änderungen in Eclipse beim Speichern der Dateien automatisch ins Modell zurückzuführen – vorausgesetzt, sie können fehlerfrei kompiliert werden. objectiF stellt auf diese Weise sicher, dass Sie nur syntaktisch korrekten Code ins Modell übernehmen. Wird das Modell geändert, wirkt sich das meistens auf mehrere Klassen oder auch Packages aus. Die Aktualisierung aus dem Modell in Richtung Eclipse muss daher immer explizit angestoßen werden. Bevor eine Datei in Eclipse aus dem Modell heraus überschrieben wird, wird die aktuelle Version in der lokalen Historie von Eclipse gesichert. Die komfortablen Synchronisationsfunktionen gewährleisten, dass Modell und Code auch über mehrere Releasezyklen hinweg immer zueinander passen. So können die beiden Tools optimal eingesetzt werden.

### Refactoring und Test

Refactoring ist eine Software-Engineering-Technik, die die Restrukturierung vorhandener Anwendungen erleichtert. Dabei wird das Design der Software verbessert und ihre Lesbarkeit erleichtert, ohne die bestehende Funktionalität zu beeinträchtigen. Gutes Design erlaubt die einfache Wiederverwendung von Komponenten und hilft, Zeit und Geld zu sparen. Eclipse bietet eine Vielzahl von Refactoring Patterns an: von einfachen Patterns auf Methodenebene bis hin zu Patterns, die die Struktur des Entwurfs verändern. Wann

aber macht es Sinn, ein solches strukturveränderndes Refactoring Pattern anzuwenden? Und welche Auswirkungen zieht eine Strukturänderung nach sich?

Antwort auf diese Fragen gibt das UML-Modell: Es macht Abhängigkeiten auf einem überschaubaren Abstraktionslevel sichtbar. Da objectiF bei jeder Änderung im Modell oder Code automatisch alle implementierungsrelevanten Abhängigkeiten mitpflegt, lassen sich vorhandene Abhängigkeiten aktuell und übersichtlich in den Package-Diagrammen darstellen. Auf Wunsch zeigt das Tool für ausgewählte Elemente alle Referenzen bis ins letzte Implementierungsdetail an. Mit objectiF können Refactorings also sicher geplant und durchgeführt werden. Das Tool stellt eine Reihe von Refactoring Patterns auf Architekturebene zur Verfügung. Extrahieren von Superklassen und Interfaces, Verschieben oder Umbenennen von Elementen oder auch das Ersetzen von Typen können Sie im Tool per Drag & Drop beziehungsweise auf Mausklick durchführen. Alle Änderungen werden an allen Verwendungsstellen im Code und Modell automatisch übernommen.

Kein Entwickeln ohne Testen. Um Fehler und unerwünschte Seiteneffekte schnell erkennen und beheben zu können, sollten Tests so früh wie möglich in den Entwicklungsprozess integriert werden. Mit objectiF kann man Tests bereits beim Entwurf der Anwendung planen, und man legt Testklassen, Testmethoden und Testsuites für das JUnit-Test-Framework an. Die JUnit-Tests können anschließend in Eclipse implementiert und ausgeführt werden. Die Testsuites werden von objectiF automa-

tisch gepflegt. Kommt ein neuer Test hinzu, wird die Suite entsprechend ergänzt.

### Eigene Funktionalität integrieren

objectiF ist genau wie Eclipse erweiterbar. Das Tool lässt sich um projektspezifische Entwurfsmethoden ergänzen. Die benötigten Informationen können über hierarchische Stereotypen und Tagged Values im UML-Modell abgelegt werden. Ebenso lässt sich die Codegenerierung benutzerspezifisch anpassen und erweitern. Dabei können alle Elemente des UML-Modells ausgewertet werden. Das heißt, auch Verhaltensdiagramme wie Aktivitäts- oder Zustandsdiagramme für die Codegenerierung werden verwendet. Dadurch wird es möglich, aus dem UML-Modell neben reinen Java-Klassen auch XML-Dateien wie z.B. XML Schemata oder Konfigurationsdateien für Struts zu generieren.

### Zusammenfassung

Eclipse bietet alles, um Anwendungen effizient zu implementieren, zu erstellen, zu debuggen und zu testen. objectiF ist ein leistungsfähiges UML-Tool. Es unterstützt dabei, Softwarearchitekturen mit den Mitteln der UML sichtbar zu machen. objectiF kann einen technischen Entwurf direkt in Code umsetzen, aber auch aus dem bestehenden Code die zugrunde liegende Struktur ermitteln und visualisieren.

Eclipse und objectiF arbeiten gut zusammen. Das UML-Modell bildet die Basis für die Implementierung in Eclipse. Die aus dem Modell generierten Dateien sind dabei nicht auf reine Java-Klassen beschränkt. Die Synchronisation zwischen beiden Tools stellt sicher, dass das Modell auch in der Implementierungsphase aktuell bleibt. Durch die aktuelle und übersichtliche Darstellung von Abhängigkeiten und die Refactoring Patterns in objectiF ist das Modell die geeignete Wahl, wenn größere Änderungen bzw. Erweiterungen geplant und durchgeführt werden. ■

*Matthias Ehlert ist Senior Consultant bei microTOOL. Er unterstützt Kunden beim Einsatz der UML und der Einführung von Entwicklungsprozessen.*

### Links & Literatur

[1] objectiF Eclipse Edition: [www.microtool.de/objectif/de/prod\\_eclipse.aspx](http://www.microtool.de/objectif/de/prod_eclipse.aspx)