

Modellgetriebene Softwareentwicklung

Ein Bild sagt mehr als tausend Zeilen Code

Model Driven Software Development (MDS) ist eine Technologie, die den Prozess der Softwareentwicklung enorm beeinflusst. Die Verwendung von Modellen in einem Softwareentwicklungsprojekt ist an sich nichts neues, MDS führt jedoch eine neue Art der Verwendung des Modells ein – aus dem Modell wird der ausführbare Code generiert und als Basis des Projekts verwendet. Viele Entwicklungswerkzeuge beherrschen die Technik in einem begrenzten Umfeld allerdings schon seit Jahren, der Unterschied liegt viel mehr in der Sicht auf Prozesse und Organisation.

Ein Modell kann beliebig definiert werden. In diesem Artikel möchten wir uns allerdings primär den UML-Modellen widmen. UML definiert mehrere Diagramme, die jeweils einen Teil der gesamten UML-Spezifikation abbilden.

MDS ermöglicht die Generierung des Programmcodes aus diversen Diagrammen. Der Klassiker ist selbstverständlich das Klassendiagramm – hat man die Klassen in der objektorientierten Welt in ein Diagramm modelliert, so kann man daraus einen Code erzeugen. Durch die Eigenschaft der Klassendiagramme, die Struktur und nicht das Verhalten zu modellieren, ist auch das Ergebnis bedingt – der generierte Code besteht aus einem "Skelett" der Klassen: Klassendefinition, Attribute, Methodendefinition. Damit ist das Generat "nur" halbfertig.

Aus diesem Grunde können in einem Softwareentwicklungsprojekt auch Bereiche identifiziert werden, die mehr oder weniger gut für die Generierung des Codes aus Klassendiagrammen geeignet sind. So können in einer 3-Schichten-Architektur mit einem vertretbaren Aufwand (!) ca. 90 % des Programmcodes in der Persistenzschicht generiert werden. Das ist leicht nachvollziehbar, wenn man bedenkt, dass die Persistenzschicht meistens aus persistierbaren Klassen (in vielen Technologien auch Entitäten genannt), evtl. aus Data-Access-Objekten (DAO) und wenig anderen Artefakten besteht. Die persistierbaren Klassen zeichnen sich gerade dadurch aus, dass diese hauptsächlich die Struktur der Business-Objekte vorgeben und relativ wenig Verhalten beinhalten. Die DAO-s haben zwar nur Verhalten und sehr wenig Struktur, aber dafür ist das Verhalten meistens standardisiert und besteht aus CRUD- (CREATE, READ, UPDATE, DELETE) und "Finder"-Operationen. Die restlichen Artefakte, wie z.B. technologiespezifische Deskriptoren stellen auch überwiegend die Struktur dar und können generiert werden.

Anders sieht es in den übrigen zwei Schichten einer 3-Schichten-Anwendung aus. Die Business-Schicht hat weniger Struktur und viel Verhalten, die Präsentationsschicht meistens gleich viel von beidem. Selbstverständlich kann man ein Klassendiagramm mit einem Aktivitätsdiagramm kombinieren und dadurch sowohl Struktur, als auch das Verhalten abbilden. Die komplexen Berechnungen innerhalb der Operationen könnte man sogar durch Nassi-Schneidermann-Diagramme beschreiben und die Klassendiagramme dadurch erweitern (wenn das auch nicht mehr im Sinne der UML ist).

Die Strukturen der Präsentationsschicht lassen sich auch sehr detailliert modellieren: Jede Schaltfläche, jedes Ein- und Ausgabefeld, jedes Optionswahlfeld kann in ein Diagramm modelliert und zum Generieren des Codes benutzt werden. So könnte man wieder auf die gleichen 90 % des gesamten Programmcodes kommen, wie in der Persistenzschicht – es ist aber leicht zu erkennen, dass die Komplexität steigt. Und Komplexität ist genau das, was man durch Einsatz der Modelle vermeiden oder reduzieren wollte. Neben dem Klassendiagramm gibt es auch andere Diagrammtypen, die sehr gut für die Generierung des Codes geeignet sind. So z.B. die Zustandsautomaten – sie sind strikt definiert und, was sehr wichtig ist, sie beschreiben im Idealfall nur einen kleinen Ausschnitt des Gesamtmodells.

Die Vorteile der Code-Generierung in einem großen Softwareentwicklungsprojekt liegen auf der Hand:

- Der Code ist mit dem Modell synchron und entspricht (meistens) der Dokumentation.
- Der Code ist standardisiert und damit viel einfacher zu verstehen und zu warten.

Die Code-Generierung bringt aber auch Nachteile mit sich:

- Es besteht die Gefahr, dass die Modelle zu komplex werden.
- Der Entwickler ist nicht mehr so flexibel und kann nicht immer nach Belieben den Code ändern.
- Es werden zusätzliche Regeln eingeführt, die den Prozess der Softwareentwicklung komplexer machen.

Doch wie soll es in einem Softwareentwicklungsprojekt aussehen, wer erstellt die Modelle und wer verwendet diese?

Es ist heutzutage üblich, den gesamten Softwareentwicklungsprozess in Disziplinen aufzuteilen. Meistens sind das Analyse, Architektur, Design, Entwicklung, Deployment und Test. Die "Analyse" (Anforderungs- und Systemanalyse) beschäftigt sich mit der Entwicklung der Fachkonzepte und ist im Idealfall die einzige Schnittstelle zu dem Kunden oder der Fachabteilung. Die "Architektur" legt die groben Vorgaben des Projektes fest. Die Disziplin "Design" erarbeitet die Feinstruktur des Anwendungssystems. In der Disziplin "Entwicklung" wird das Anwendungssystem ausprogrammiert. Deployment beschäftigt sich mit den Fragen der Installation und die Disziplin "Test" beschäftigt sich mit der Prüfung des Produkts.

Im Idealfall werden die Modelle in der Analyse erstellt. Dadurch wird gewährleistet, dass die Ergebnisse der Analyse durch die Modellspezifikation formalisiert und weitestgehend interpretations- und widerspruchsfrei sind. Diese Modelle werden dann sowohl für die Entwicklung, als auch für die Tests als Grundlage genommen. Die Analyse erstellt ein Modell, welches frei von den technischen Aspekten ist – ein Platform Independent Model (PIM). Die technischen Aspekte werden erst in der Architektur festgelegt. Durch die Kombination des PIM und der Ergebnisse, die in der Disziplin Architektur erarbeitet werden, entsteht ein Platform Specific Model (PSM).

Das PSM wird in Design weiter ausgearbeitet, so dass aus dem Modell der Code generiert werden kann. Sind die Modellübergänge von Analyse zu Design und von Design in den Code korrekt definiert und ausgeführt, so sollte der Code auch vollständig dem Analysemodell entsprechen.

Die Übergänge selbst werden Modelltransformation genannt und beinhalten auch eine bestimmte Logik, welche die Vorgaben der Architektur und ausgewählte Design-Patterns umsetzen kann. So z.B. kann die Modelltransformation bestimmen, ob ein DAO-Pattern oder Service-Locator verwendet wird...

MDSO gewinnt immer mehr an Sympathie und Vertrauen - besonders auch beim Unternehmens-Management. Denn was könnte für künftige Projekte besser sein, als Analyse, Entwicklung und Test im Einklang zu halten und den Code in der Entwicklung automatisch zu generieren?

Doch die Technologie birgt auch Gefahren, die ein Projekt in Zeitnot bringen können. Vor allem ist es wichtig das richtige Maß an Modellierung und an der Logik bei den Modelltransformationen zu definieren, denn der "Job" den Programmcode zu schreiben gehört dem Programmierer und der ist bestimmt nicht vom Aussterben bedroht...

Kontaktinformationen

Isento GmbH
IT-Beratung & Services

Venusweg 1a
90763 Fürth

Telefon 0911 21 77 38 70
Fax 0911 21 77 38 77
E-Mail info@isento.de
Web www.isento.de