Modellbasierte Softwareentwicklung

Wer den Einstieg in "Modellbasierte Softwareentwicklung" über eine Internetrecherche sucht, kann ganz schnell frustriert sein. Vieles davon ist "abstrakte Kunst". Im schlimmsten Fall landet man bei Metamodellen zu deren Verständnis man ein Studium beginnen kann. Dabei ist in der Praxis alles viel einfacher. Überall wo bereits jetzt Code-Generatoren eingesetzt werden liegt modellbasierte Softwareentwicklung vor. Allerding ist man sich dessen meistens nicht bewusst, weil primär der generierte Sourcecode im Vordergrund steht und nicht die Daten woraus dieser Sourcecode generiert wird. Dabei bilden genau diese Daten das Input-Modell.

Um Klarheit zu haben brauchen wir eine Definition für ein "Modell".

Ein Modell ist eine wohl strukturierte Menge von Daten, die alle Informationen beinhalten, um daraus eine wohl strukturierte Menge von Daten erzeugen zu können!

Damit ist implizit ganz nebenbei auch der Begriff "Modelltransformation" erklärt, denn er meint nichts anderes als das Erzeugen eines Modells aus einem anderen Modell. In der Praxis ist es natürlich Sinn der Sache, dass eine Modelltransformation mittels Parser und Generator automatisch durch geführt werden kann, denn das ist der eigentliche Benefit, den man von der modellbasierten Softwareentwicklung hat.

Bleibt noch eines zu klären: Was meint in obiger Definition der Begriff "wohl strukturiert"? Nun, wenn die Modelltransformation automatisch durchführbar sein soll, muss es eine eindeutige Zuordnung der Daten aus dem Input-Modell zu den Daten aus dem Output-Modell geben und es muss eindeutig festgelegt sein, wie im Input-Modell der Wert für ein ganz bestimmtes Datum ermittelt werden kann und wie und an welchen Stellen diese in das Output-Modells einfließen. Dies alles wird in der Modelltransformation beschrieben! Aber eine Modelltransformation kann einfach oder kompliziert sein und um es vorweg zu nehmen, bei der modelbasierten Softwareentwicklung ist es genau das "das Kunststück"; nämlich die Struktur eines Modells gerade so zu wählen, dass die benötigten Modelltransformationen möglichst einfach beschrieben und möglichst effizient umsetzbar sind.

Soweit so gut! Aber irgendwo muss man doch mal anfangen? Genau - Modelle werden entwickelt. Und es beginnt bei dem Sourcecode der generiert werden soll. Auf die Idee einen Sourcecode-Generator zu schreiben kommt man meist dann, wenn man als Entwickler immer wieder ähnlichen Sourcecode schreiben muss. (Das kann sich auf ganze Dateien oder auch nur auf sich vielmals wiederholenden Sourcecode-Fragmente in einer Datei beziehen.) Ab dem Moment ist einem bereits bewusst, dass der zu erstellende Sourcecode in zwei Teile zerfällt, nämlich in konstanten Code, der bei allen Varianten identisch ist und in variablen Code-Stücke, die in den konstanten Code eingebettet sind. Im einfachsten Fall können die variablen Code-Stücke durch einen Platzhalter

ersetzt werden, der dann bei der Generierung wiederum durch einen konkreten Wert bzw. Text ersetzt wird. Als klassisches Instrument wird dazu ein Präprozessor verwendet, der genau dieses macht und vor der Kompilierung die definierten Platzhalter durch aktuelle Werte setzt. Spätestens aber dann, wenn im Sourcecode tabellenartige Strukturen aufgebaut werden müssen, ist es mit dem reinen Ersetzen von Platzhaltern nicht mehr getan - ein richtiger Codegenerator wird benötigt.

Einen Codegenerator für selbst geschrieben, individuellen Sourcecode kann man natürlich nicht fertig kaufen - denn muss man selber erstellen. Und hier beginnt das Dilemma! Denn einen Codegenerator programmiert man nicht einfach mal so nebenbei. Denn er muss nicht nur Dateien generieren können, sondern auch Input-Dateien parsen können, um die relevanten Daten zu extrahieren. Im allgemeinen müssen diese auch noch transformiert werden, bevor sie zur reinen Generierung verwendet werden können. Schnell sind da etliche Personenmonate verbraucht. Na, lohnt ich das dann wirklich? Lieber doch keine modellbasierte Softwareentwicklung?

Generator-Generatoren helfen weiter. Nichts ist naheliegender als zum Erstellen des Sourcecodes für einen Source-Generators einen Source-Generator zu verwenden? Den Satz muss man sich erst nochmal in anderer Form durchlesen: " ...Sourcecode für einen Source-Generators mit einen Source-Generator generieren". das klingt doch gut! Und in der Tat, schon seit den Anfängen der Informatik gibt es Parser-Generatoren, die aus einer Beschreibung im EBNF-Format den Sourcecode für einen Parser generieren. Zum Generieren der Output-Dateien, suchen wir uns dann irgendwas mit Template-Programmierung, die beiden kombinieren wird dann und verbinden sie mit den notwendigen Tansformationen, die wird dann selber hinzu programmieren. Dann müssen wir nur noch ein paar Fehler rausmachen - aus dem verflixten Ding - schon sind wird fertig - nach ein paar Wochen. Tja, das war das Ende einer guten Idee.

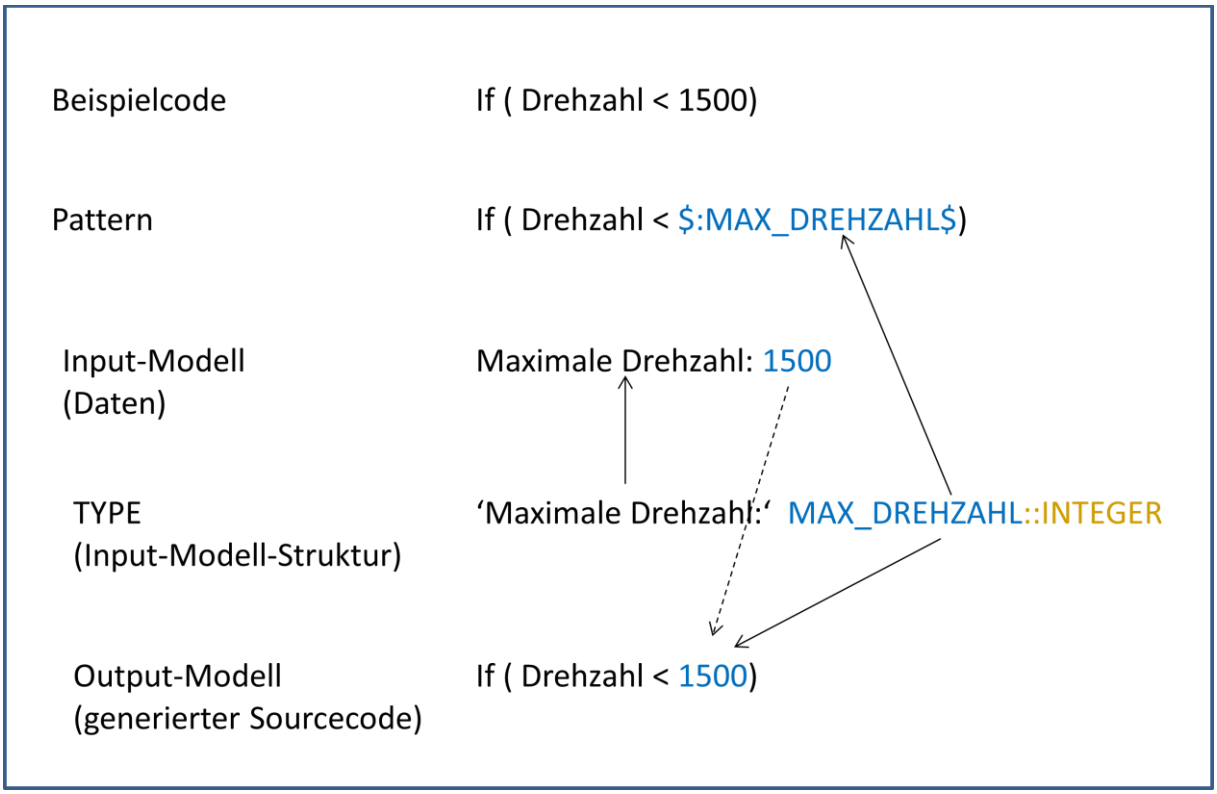
Die Idee "Modellbasierte Softwareentwicklung" scheiterte in der Vergangenheit meist daran, dass der Aufwand den notwendigen Generator zu erstellen den Aufwand überstieg, den man hat, wenn man den Sourcecode immer wieder händisch anpasst. Letzteres ist zwar nervig und impliziert in der Regel aufwendige nachfolgende Fehlersuche, aber führt in einem gesicherten Zeitrahmen ans Ziel.

Jetzt noch einmal und ganz anders: Was wäre wenn wir ein Tool hätten, das wir binnen weniger Stunden so konfigurieren könnte, dass es sich selber in den Sourcecode-Generator umwandelt, den wir brauchen? Nein, das ist kein Traum sondern Realität. (Siehe http://x2x.sss.de)

x2xGen ist ein Interpreter der mittels drei verschiedenartiger Konfigurationsdateien so instruiert werden kann, dass er einen beliebigen Sourcecode-Generator realisiert, in dem er intern einen Parser, einen Transformator und einen Generator nachbildet und diese selber zu einer ausführbaren Einheit kombiniert.

Dabei spiegeln die drei verschiedenen Arten von Konfigurationsdateien genau die Grundbestandteile der modellbasierten Softwareentwicklung wider. Mit TYPE-Dateien wird die Struktur des Input-Modells beschrieben. PATTERN-Dateien beschreiben die Struktur der zu generierenden Dateien des Output-Modells nach dem WYSIWYG-Prinzip und in SCRIPT- Daten werden schließlich die notwendigen Modelltransformationen festgelegt. Hinzu kommt noch eine MAIN-Datei die im Sinne einer Projektdatei fungiert und dem Interpreter mitteilt, wo er welche Dateien findet bzw. ablegen soll.

Wenn in der Praxis ein Sourcecode-Generator erstellt werden soll, liegt immer schon entsprechender handgeschriebener Beispielcode vor. Dabei gibt es jetzt zwei grundverschiedene Situationen: Entweder die variablen Bestandteile des zu generierenden Sourcecode sind von anderer Seite vorbestimmt (z.B. in Dateien die von extern geliefert werden) oder es gibt noch keine Vorgaben für das Input-Modell. Ist letzteres der Fall, kann das Input-Modell auf einfache Weise aus dem zu generierenden Beispielcode entwickelt werden. Dabei startet man den Beispielcode als PATTERN, in dem Schritt für Schritt Platzhalter konkrete Werte ersetzen, wobei diese wiederum gleichzeitig in die Dateien des Input-Modell aufgenommen werden und der Name des Platzhalters in der die TYPE-Datei übernommen wird, die die Struktur des Input-Modells beschreibt. Um ein effizientes Parsen des Input-Modells zu ermöglichen wird noch dem Wert eines Platzhalters ein entsprechendes Schlüsselwort vorangestellt.



In diesem Beispiel soll das Sourcecode-Fragment "If (Drehzahl < 1500)" erstellt werden, wobei die Zahl "1500" variabel ist und über das Input-Modell bestimmt wird. Im PATTERN wird diese Zahl durch den Platzhalter $:MAX_DREHZAHL$ ersetzt und der Platzhalter als vom Type ::INTEGER in den TYPE aufgenommen, wobei das Schlüsselwort 'Maximale Drehzahl' vorangestellt ist. Diese weist den Parser an, dass er in den Input-Daten den Wert für den Platzhalter MAX_DREHZAHL nach dem Literal 'Maximale Drehzahl' findet.

Neben dem Einfachen verwenden von Platzhaltern führen Listen im Input-Modell und Schleifen-Anweisungen zu Ziel, wenn sich wiederholende Sourcecode-Fragmente generiert werden müssen, wie zum Beispiel Initialwerte für konstante Arrays. Dabei soll an dieser Stelle aber nicht weiter ins Detail gegangen werden, denn das würde den Rahmen dieses Artikels sprengen. Konkrete Beispiele dazu findet man aber unter **http://x2x.sss.de** .

Ergibt es sich beim Erarbeiten des Input-Modells, dass dieses stark tabellen-lastig ist, so ist es naheliegend diese Tabellen mit einem Tabellenkalkulationsprogramm wie z.B. Excel zu erfassen und im CSV -Format zu exportieren. In diesem Fall hat der Anwender zwar die freie Wahl darin wie er seien Tabellen aufbaut, jedoch ist die Struktur für das Input-Modell des Sourcecode-Generators durch das CSV-Format vorbestimmt. Der TYPE für das Input-Modell muss also genau so definiert werden, dass die Input-Werte aus der CSV-Datei richtig erfasst werden. Da CSV-Dateien eine besonders einfache, zeilenorientierte Struktur haben, stellt das Erstellen des TYPEs keine besonderen Ansprüche. Andererseits wird man durch die Verwendung des Tabellenkalkulationsprogramms die Input-Daten besonders anschaulich darstellen können. Dieses Beispiel zeigt auch, dass es oft günstig ist Modelltransformationen in mehreren Schritten durchzuführen, denn der Export in eine CSV-Datei ist nichts anderes als eine Modelltransformation.

Zurück zu dem Fall, dass das Input-Modell von extern vorgegeben ist. Hier empfiehlt es sich eine Modell-Transformation dazwischen zu schalten. D.h. für das Input-Modell und das Output-Modell zwei zunächst unabhängige Strukturen bzw. TYPEs festzulegen und dann diese mittels einer Modell-Transformation ineinander zu überführen.

Ist erst ein solcher Generator im Einsatz, kommt man schnell auf den "Geschmack" und es werden weitere folgen. Neben den Sourcecode-Generatoren kommen Dokumenten-Generatoren zum Einsatz, die nach gleichen Schema erstellt werden. In den Make-Prozess integriert bilden sie eine Kette von Modelltransformationen, die modellbasierte Softwareentwicklung zum Alltag machen und so für hochqualitative Softwareprodukte sorgen.