TEIL 1

Modellieren, aber richtig!

Über die Vorteile und den Nutzen der modellbasierten Software- und Systementwicklung

Whitepaper

Dr. Horst Kargl
SparxSystems Software GmbH
www.sparxsystems.de
vendor@sparxsystems.eu
© 2017

ENTERPRISE ARCHITECT

Einleitung

Die Entwicklung moderner Software und durch Software unterstützter Systeme wird immer aufwendiger und komplexer. Immer mehr Unternehmen entscheiden sich daher, ihren Entwicklungsprozess zu überarbeiten. Die Verwendung grafischer Modellierungssprachen und dazu passender Werkzeuge kann dabei helfen, die Komplexität in den Griff zu bekommen. Allerdings gilt es bei diesem Schritt einige grundsätzliche Dinge zu beachten, um den Nutzen von Modellen auch wirklich auszuschöpfen. In meinen Kursen stelle ich zu Beginn gerne die Frage: Warum modellieren wir eigentlich? Als Antworten höre ich z. B. „Weil wir es müssen“, „Unsere Software muss auch dokumentiert werden“ etc. Diese Antworten zeigen mir, dass die Vorteile eines Modellierungsansatzes (im Vergleich mit dem klassischen, textbasierten Ansatz) meist nicht umfassend gesehen werden. Daher habe ich zwei „White Paper“ erstellt, die die einzelnen Aspekte näher beleuchten:

- **Modellieren, aber richtig! Teil 1** - Über die Vorteile und den Nutzen der modellbasierten Software und Systementwicklung
- **Modellieren, aber richtig! Teil 2** - Wie Entscheider die richtige Modellierungsstrategie finden und im Unternehmen installieren können

Im hier vorliegenden ersten Teil gehe ich der Frage nach dem Vorteil von modellbasierten Ansätzen nach, die grundsätzlich einfach zu beantworten ist. Um über die Inhalte eines Dokuments diskutieren zu können, müssen wir bei einer textuellen Beschreibung viele Seite lesen. Haben wir darüber hinaus Bilder verwendet, die keinem Standard entsprechen, sind auch diese meist erklärungsbedürftig. Bei Modellen dagegen, die mit standardisierten Modellierungssprachen (z.B. UML/SysML) erstellt wurden, folgen die Symbole einer definierten Semantik. Der Begleittext ist kurz und knapp den betreffenden Modell-Elementen zu entnehmen und damit leicht lesbar. Ein (richtig erstelltes) Modell/ Diagramm ist nicht überfrachtet, beantwortet genau die Fragen einer bestimmten Personengruppe und dient als Grundlage für die Diskussion. Ein Modell kann also zur zentralen Informationsquelle ausgebaut werden, aus der sich unterschiedlichste Artefakte (Dokumente, Code, Schemata, etc.) automatisch generieren lassen. Das Modell dient als zentrale Informationsquelle und lässt sich automatisiert weiterverarbeiten.

Textuelle vs. Modellbasierter Spezifikationen

Warum modellieren wir eigentlich? Diese Frage stelle ich oft in meinen Kursen. Ich bekomme dabei die unterschiedlichsten Antworten: „Weil wir es müssen“, „Unsere Software muss auch dokumentiert werden“ etc.… Diese Antworten zeigen, dass der Mehrwert, der durch einen Modellierungsansatz generiert werden kann, noch nicht gesehen wird. Der Umstieg von einem Textuellen Ansatz zu einem Modellbasierten Ansatz scheint anfänglich einen gewissen Mehraufwand zu verursachen. Ob es wirklich einen Mehraufwand gibt, bzw. wo dieser Mehraufwand liegt, wird in nächsten Abschnitt erörtert.

Textuelle Ansätze vs. Modellbasierte Ansätze

Ist Modellierung aufwendig? Ich denke, diese häufig gestellte Frage ist nicht so einfach zu beantworten, da es mehrere Dimensionen gibt, die bei der Erstellung von Modellen berücksichtigt werden müssen. Vergleichen wir modellbasierte Ansätze mit traditionellen, textbasierten Ansätzen.

Die Arbeit mit Dokumenten

Es ist wahrscheinlich einfacher, ein Worddokument mit Diagrammen zu erstellen, vorausgesetzt wir können mit Word (dem Werkzeug) und unserer bevorzugten Sprache (Deutsch, Englisch, etc.) umgehen. Darüber hinaus benötigen wir ein Zeichenwerkzeug (z. B. PowerPoint, Visio, etc.) für die Erstellung der Bilder, das natürlich auch beherrscht sein will. Wenn wir einige Jahrzehnte zurück denken, war die Bedienung von Office Produkten die große Herausforderung. Nehmen wir also an, dass diese zwei bis drei Dimensionen kein Problem darstellen, haben wir trotzdem noch den Aufwand, uns Gedanken über den Aufbau und die Struktur des Dokuments zu machen: In welchem Kapitel wird welche Information beschrieben? Wie abstrakt bzw. mit wie vielen Details soll das System beschrieben werden? Sie meinen jetzt, dass diese Struktur ja schon von den KollegInnen vorgegeben ist!? Das ist gut und wichtig! Irgendjemand hat sich diese Struktur allerdings auch einmal gründlich überlegt, sie an die in der Firma gelebten Prozesse angepasst und gelernt, sie richtig zu befüllen. Diese Vorgaben werden in der Regel auch nicht an einem Tag erstellt, sondern über einen längeren Zeitraum immer wieder aktualisiert, bis man mit der Vorlage zufrieden ist.

Die inhaltliche Beschreibung will überlegt sein

Neben den erwähnten Punkten müssen wir uns natürlich auch Gedanken machen, wie wir die zu übermittelnden Informationen in Worte oder Bilder fassen. Personen, die das Dokument lesen, sollen ja auch alles so verstehen, wie es gedacht war. Wir müssen daher Überlegungen zur Abstraktion und zu den Details der Beschreibung anstellen, sowie geeignete Symbole für unsere Bilder finden.

Bei den heute verwendeten textuellen Ansätzen sind einige Hürden schon vor vielen Jahren genommen worden (Textverarbeitung, Deutsch, Englisch, etc.). All diese Themen lernen wir bereits in der Grundschule. Verwendet man allerdings keine standardisierten und gut dokumentierten Symbole für die grafische Darstellung, muss jede Abbildung hinterfragt werden. Auch lässt sich der Inhalt eines Dokuments nur schwer erfassen, wenn seine Struktur nicht bekannt ist. Damit wir das schnelle Nachschlagen erschwert.

Zusatzaufwand bei der Modellierung (Tool und Sprache)

Möchten wir von einer textuellen Beschreibung unseres Systems auf einen modellbasierten Ansatz wechseln, müssen wir die verwendete Modellierungssprache und das verwendete Werkzeug kennen. Dafür müssen wir die Modellierungssprache und den Gebrauch des Werkzeugs lernen. Warum sollten wir diesen Schritt überhaupt machen, wenn wir mit Word und PowerPoint/Visio bereits Erfahrung haben? Denn offenbar

ist es in beiden Fällen notwendig, den Inhalt eines Dokuments zu strukturieren und zu beschreiben.

Was ist der Vorteil von Modellen?

Die Frage nach dem Vorteil von modellbasierten Ansätzen ist einfach zu beantworten. Das System ist in Dokumenten beschrieben, die in sich konsistent gehalten werden müssen. Ändert man z. B. den Namen einer Schnittstelle, sind alle Dokumente manuell zu aktualisieren. „Suchen und Ersetzen“ hilft dabei nur bedingt. Es muss ja auch der Kontext einbezogen werden, um nicht das falsche Wort zu ersetzen. Eindeutige und unverwechselbare Namen erleichtern diese Aufgabe zwar, sind aber nicht immer möglich.

Automatismen sind in Dokumenten nur schwer anwendbar

Wurden die Änderungen in das Dokument übernommen, folgt anschließend die Prüfung auf Vollständigkeit und Konsistenz. Dazu reichen die derzeit verfügbaren Möglichkeiten künstlicher Intelligenz (KI) allerdings noch nicht aus. Auf eine automatische Konsistenzüberprüfung des Textdokuments müssen wir daher verzichten. (Wie wir auch auf andere Automatismen verzichten müssen, welche die Verwendung einer formaleren Beschreibung als Modell mit sich bringt!!)

Der Vorteil einer standardisierten grafischen Sprache

Eine standardisierte grafische Sprache ist dann besonders nützlich, wenn sie von allen beherrscht wird. Verwendet allerdings jeder seine eigenen Symbole, führt dies automatisch zu Missverständnissen und Klärungsbedarf. Auch mit einer standardisierten Sprache ist man nicht davor geschützt, diese falsch zu verwenden. Je länger man allerdings damit arbeitet, und je mehr Modelle wir bereits erstellt haben, desto geringer ist die Wahrscheinlichkeit, etwas falsch zu machen. Das verhält sich ganz ähnlich wie beim Erlernen einer Fremdsprache.

Der Aufwand beim Review

Um über die Inhalte eines Dokuments diskutieren zu können, müssen wir bei einer textuellen Beschreibung viele Seite lesen. Haben wir darüber hinaus Bilder verwendet, die keinem Standard entsprechen, sind auch diese meist erklärungsbedürftig. Bei Modellen dagegen, die mit standardisierten Modellierungssprachen (z.B. UML/SysML) erstellt wurden, folgen die Symbole einer definierten Semantik. Der Begleittext ist kurz und knapp den betreffenden Modell-Elementen zu entnehmen und damit leicht lesbar. Ein (richtig erstelltes) Modell/Diagramm ist nicht überfrachtet, beantwortet genau die Fragen einer bestimmten Personengruppe und dient als Grundlage für die Diskussion.

Fazit

Sowohl bei textuellen als auch bei modellbasierten Ansätzen muss man sich identen Fragen stellen. Beim ersten Ansatz entfällt nur der Lernaufwand für die Modellierungssprache und das Modellierungswerkzeug. Verfügen wir allerdings über ein gewisses Grundverständnis einer standardisierten Modellierungssprache und des gewählten Modellierungswerkzeuges, dann bietet der Modellierungsansatz wesentlich mehr Möglichkeiten als ein rein textbasierter. Ein Modell kann zur zentralen Informationsquelle ausgebaut werden, aus der sich unterschiedlichste Artefakte (Dokumente, Code, Schemata, etc.) automatisch generieren lassen. Das Modell dient als zentrale Informationsquelle und lässt sich automatisiert weiterverarbeiten.

◆

Vom Nutzen der Modelle

Wenn wir von Modellen und Modellierungssprachen sprechen, haben viele (je nach Erfahrung) gleich ein bestimmtes Bild im Kopf. Verfügt man über keine einschlägigen Vorkenntnisse, besteht dennoch oft eine Vermutung, wozu sich grafische Modelle verwenden lassen.

Die Verwendungsmöglichkeiten und der Zweck von grafischen Modellen sind nach meiner Erfahrung aber oft sehr unterschiedlich. Die meisten der bestehenden Vorstellungen von grafischen Modellen sind zwar zutreffend, beschreiben aber oft nur einen bestimmten Aspekt und übersehen dabei viele weitere Nutzungsmöglichkeiten. Modelle können also verwendet werden, um:

- Systemanforderungen mithilfe von Modellen zu beschreiben
- Unternehmensarchitekturen darzustellen
    - Geschäftsprozesse
    - Unternehmensdaten
    - Systeme
    - Technische Dienste (Services)
    - Rollen und Aufgaben
- Eine Systemarchitektur aufzuzeigen, d. h. die Bausteine und deren Abhängigkeiten
- die Schnittstellen des Systems zu definieren
- Datenstrukturen zu definieren
- das Systemverhalten zu beschreiben
- die Implementierung zu beschreiben
    - Die Struktur der Implementierung
    - Das Verhalten der Implementierung
- vorhandenen Code darzustellen (Reverse Engineering).
- etc.

Die aufgelisteten Punkte beschreiben die Informationen, die möglicherweise in einem Modell enthalten sein können. Je nachdem, wer das Modell lesen soll, werden jedoch jeweils andere Informationen erwartet. Dieser Umstand führt uns direkt zu den zwei wichtigsten Fragen, die bei der Einführung von Modellierung zu stellen sind:

Wer liest das Modell?

Was soll aus dem Modell gelesen werden?

Sie haben sicher gleich gemerkt, dass diese Fragen nicht nur für die Modellierung gelten, sondern auch bei einer textuellen Beschreibung gestellt werden sollten. Leider werden sie allerdings bei der Modellierung fast nie gestellt, da (wie oben beschrieben) meist eine bestimmte Vorstellung vorhanden ist, was ein Modell können soll, bzw. wofür ein eingesetzt wird Eine detaillierte (oder gar vollständige) Beschreibung des Systems liegt allerdings in den wenigsten Fällen vor.
Können wir diese zwei grundsätzlichen Fragen beantworten, wissen wir bereits, welche Information, auf welcher Abstraktionsebene und mit wie vielen Details im Modell vorhanden sein muss. Wir wissen allerdings noch nicht, wie wir das im Detail mit UML/SysML oder einer anderen Modellierungssprache im gewünschten Werkzeug abbilden können.
Je mehr Antworten wir zu den zwei Fragen finden, desto klarer wird das Bild unseres Modells. Die Fragen könnten für unterschiedliche Rollen z. B. so formuliert sein:

Requirements-Manager:

- Wo wird diese Anforderung umgesetzt?
- Welche Anforderungen sind noch nicht
- berücksichtigt? ...

Unternehmens-Modellierer:

- Welcher Geschäftsprozess wird von welchen Systemen umgesetzt?
- Welche Geschäftsdaten werden von welchen Systemen verwaltet?
- Welche Rollen sind in welchen

- Geschäftsprozessen beteiligt?
- Was passiert, wenn ich den Server X abschalte?....

Software-Architekt:

- Wo wird ein bestimmtes Signal verschickt?
- Wer empfängt dieses Signal?
- Wer verwendet eine bestimmte Komponente?
- Was muss ich ändern, um die neue Anforderung umzusetzen?
- Wie gestalten sich die Abhängigkeiten zwischen den Systemteilen? ....

Software-Entwickler:

- Welche Anforderungen/Schnittstellen muss ich umsetzen?
- Was muss ich ändern, um die neue Anforderung umzusetzen?
- Was für eine Auswirkung hat es, wenn ich diese Klasse ändere?
- Was muss ich alles ändern, weil sich diese Anforderung geändert hat? ....

Software-Tester:

- Welche Anforderungen muss ich testen?
- Wie sieht mein Testaufbau aus?
- Was ist das erwartete Ergebnis bzw. der erwartete Ablauf? ...

Diese Fragen sind Beispiele, die in vielen Firmen so oder in ähnlicher Weise auch gestellt werden. Jedoch existieren verschiedene Abhängigkeiten zwischen den Fragen sowie Unterschiede in der erwarteten Abstraktion und den Details der Antworten. Es kann also nicht EIN Modell für EINE Problemstellung geben, sondern nur Lösungsvorschläge, die in der Praxis erprobt sind und von den Unternehmen oft adaptiert werden.
Wäre die Lösung immer eindeutig und klar, würde es genau einen Modellierungsansatz mit einer Methode geben. Eventuell könnten wir dann noch unterschiedliche Ansätze in unterschiedlichen Domänen differenzieren. Je nachdem, welche Aufgabe nun die Firma oder Abteilung hat, könnten wir ebenfalls noch differenzieren….

Allgemein gilt also: Im Grunde kann man jeden Modellierungsansatz auf einige wenige Aspekte zurückführen, die immer gleich sind. Im Detail finden sich aber immer mehr Unterschiede, jedes Modell sieht anders aus. Erfahrung hilft dabei, einen allgemeinen Lösungsansatz an die eigenen Bedürfnisse anzupassen. Es muss letztlich ein Weg gefunden werden, um die gewünschte Information abzubilden. Dafür sollten standardisierte Modellierungssprachen verwendet werden. Dabei soll darauf geachtet werden, dass die Sprachelemente semantisch richtig eingesetzt werden. Fehlen in der verwendeten Modellierungssprache bestimmte Ausdrucksmöglichkeiten, kann darüber nachgedacht werden diese zu ergänzen.
Im ersten Schritt ist es also wichtig ein Beispielprojekt (Leuchtturm-Projekt) zu erstellen, mit dem alle gestellten Fragen beantwortet werden können. Dieses Beispielprojekt nennen wir „Rererenzmodell“.
Halten wir nochmals fest, was wir bisher erfahren haben:

- In die Erstellung einer textuellen Spezifikation fließt im Grunde derselbe Aufwand wie in eine modellbasierte Spezifikation.
- Zum Einstieg in die Modellierung sind die Hürden der Modellierungssprache und des gewählten Werkzeugs zu überwinden.
- Der Vorteil von Modellen liegt in den Automatismen, die darauf anwendbar sind.
- Es ist wichtig, sich zu Beginn einer modellbasierten Entwicklung auf ein Referenzmodell zu einigen.

Die beiden ersten Punkte haben wir bereits behandelt. Daher wenden wir uns nun den Automatismen, die wir auf ein Modell anwenden können, sowie dem Referenzmodell zu.
Haben wir ein Modell erstellt, lassen sich folgende Automatismen darauf anwenden:

- Textuelle Spezifikation (Dokumente) aus dem Modell generieren
- Abstrakte Modelle in konkretere Modelle transformieren
- Code oder codeähnliche Artefakte wie Schemata etc. generieren

- Vorhandenen Code oder codeähnliche Artefakte wie Schemata etc. einlesen
- Verifikation der Modelle auf:
    - Vollständigkeit
    - Korrektheit
- Simulation von Modellen
- Änderungen automatisiert durchführen
- Testfälle aus Modellen generieren

Damit diese Automatismen funktionieren, sind allerdings Modellierungsrichtlinien notwendig. Sie helfen dabei, den großen Spielraum, der durch die Modellierungssprache und das verwendete Werkzeugs gegeben ist, einzuschränken und das Arbeiten einfacher und überschaubarer zu machen. Das Referenzmodell ist also die Grundlage, aus der wir durch Anwendung einer bestimmten Methode unsere Richtlinien ableiten. Auf deren Grundlage wird das Modell automatisch überprüft können, um anschließend alle anderen Artefakte ebenfalls automatisch und fehlerfrei aus dem Modell generieren zu können.

Worin liegen die Vorteile von Modellen?

Die Vorteile von Modellen ergeben sich aus der Ausdrucksmächtigkeit der Modellierungssprache und den Möglichkeiten des verwendeten Werkzeuges. Die Modellierungssprache selbst ist die gemeinsame Sprache, die wir gewählt haben. Ohne geeigneter Werkzeugunterstützung, ist deren Verwendung allerdings sehr aufwendig. Die oben erwähnten Automatismen lassen sich dann nicht oder nur sehr schlecht verwenden. Eine weitere Voraussetzung für diese Automatismen ist nämlich der Einsatz eines Modell-Repositoriums, das mit dem Werkzeug eng verbunden ist.

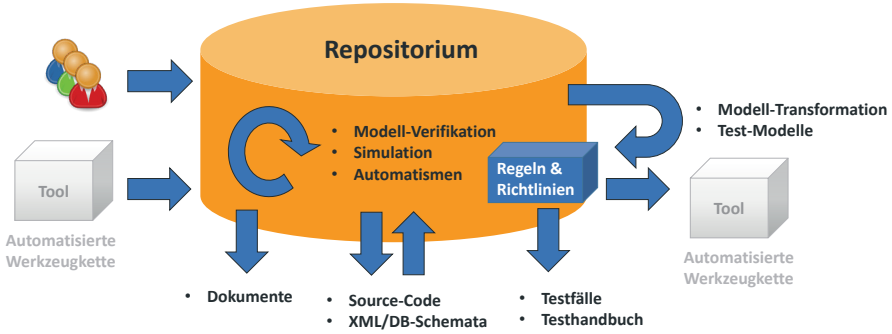
Auch Modelle können fehlerhaft sein.

Welchen Ansatz man auch immer verwendet, Fehler sind niemals ausgeschlossen. Etwa wenn die falschen Informationen mit den falschen Methoden in den falschen Werkzeugen abgebildet werden. Denn auch wenn die Modellierungssprache geeignet und das Werkzeug sehr mächtig ist, gilt die altbekannte Regel:

Garbage In

↓

Garbage Out



Wählt man aber den für das jeweilige Vorhaben passenden Ansatz, kann man von Modellen außerordentlich viel profitieren. Betrachten wir dazu einige der oben erwähnten Punkte genauer.

Dokumente generieren

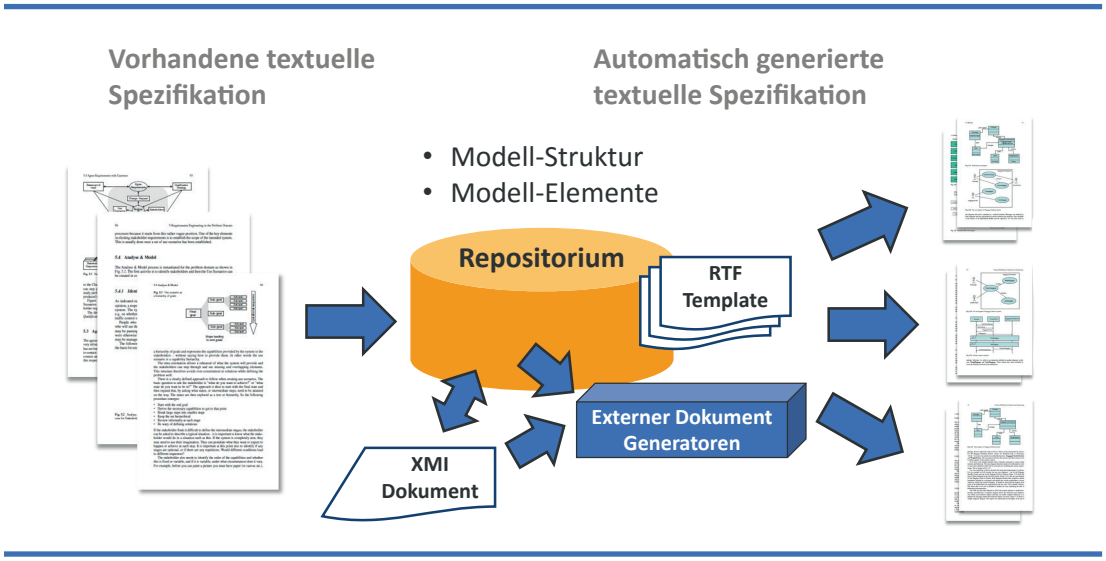
Der erste Punkt der obigen Liste („Textuelle Spezifikation (Dokumente) aus dem Modell generieren“) führt uns zurück zum Anfang unserer Betrachtungen und verdeutlicht einen grundlegenden Vorteil von Modellen. Anstatt die textuelle Spezifikation als reinen Text zu verfassen, beschreiben wir die Inhalte mit Modellen und generieren daraus elegant die textuelle Spezifikation. Natürlich werden wir die einzelnen Modellelemente auch mit Kommentaren – also Text – beschreiben, die anschließend automatisch in eine textuelle Spezifikation einfließen. So erhalten wir ein Dokument mit Diagrammen und der dazu passenden Beschreibung.

Eine mögliche Herangehensweise

Haben Sie bereits eine vorhandene textuelle Spezifikation, kann diese als Grundlage für den Aufbau der Modell-Struktur verwendet werden. Bei diesem Ansatz kennen wir die Information im Dokument und müssen uns nur noch eine geeignete Abbildung in einer Modellierungssprache überlegen. Mit dem geeigneten Werkzeug lässt sich anschließend das zuvor manuell geschriebene Dokument automatisch aus dem Modell generieren.

Durch die Verwendung unterschiedlicher RTF Templates, lassen sich sogar mehrere Dokumente mit jeweils unterschiedlichen Inhalten aus einem Modell generieren. Dafür müssen die verwendeten Dokument-Templates einmalig vorbereitet werden. Das Generieren der Dokumente kann durch einen einfachen Knopfdruck angestoßen werden und unter Umständen sogar zeitgesteuert angestoßen werden.

Wurde das Modell „richtig“ erstellt, sind die generierten Dokumente immer vollständig und in sich konsistent. Was heißt in dem Kontext nun „richtig“? Die Informationen müssen im Modell mit dem entsprechenden Detailierungsgrad vorliegen, um daraus ein Dokument generieren zu können, das wir auch manuell so geschrieben hätten. Wie schon beschrieben, gibt das Referenzmodell dafür die grundlegende Struktur vor. Mit den daraus abgeleiteten Regeln zum Aufbau des Modells lässt sich dieses auch verifizieren, wobei auf eventuelle Fehler oder Unvollständigkeiten hingewiesen wird.



Modelle transformieren

Aus einem Modell lassen sich nicht nur mehrere Dokumente generieren, sondern auch mehrere Ziel-Modelle. Das Ausgangsmodell ist dabei in der Regel abstrakter als das Zielmodell. So kann z. B. aus einem logischen Datenmodell ein physisches Datenmodell und daraus die DDL Skripte generiert werden, welche in die Datenbank eingespielt werden können. Neben Datenmodellen für Datenbanken können auch andere Modelle automatisch generiert werden, aus denen anschließend Code oder andere Schemata (XSD, RDF, JASON, etc.) generiert werden können. Grundsätzlich ist es auch möglich, direkt aus dem einen abstrakten logischen Modell, Code bzw. Code ähnliche Texte zu generieren. Der Vorteil der mehrstufigen Transformation besteht aber darin, dass jeder einzelne Schritt für sich weniger kompliziert ist als der große Schritt vom abstrakten Ausgangsmodell direkt zum Code. Zu bedenken ist auch, dass sich jedes generierte Modell mit zusätzlichen Informationen ergänzen lässt, bevor es weiter verarbeitet wird. Dadurch erhalten wir eine höhere Flexibilität.

Code generieren

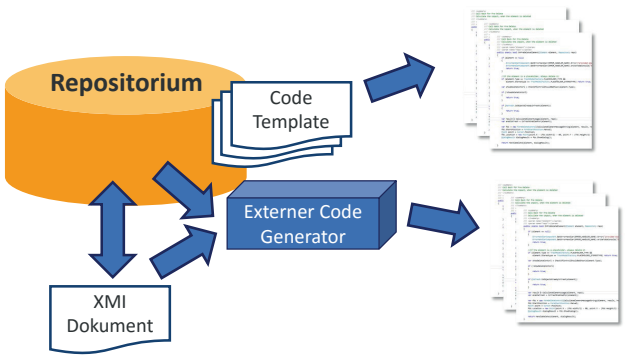
Wie schon beschrieben lässt sich aus einem ausreichend vollständigem und richtigem Modell Code generieren. Dabei werden zwei grundlegende Ansätze unterschieden.

1) *Generische Code-Generatoren*: Dafür benötigen wir ein Modell, das bereits sehr detailliert die Implementierung beschreibt und einen generischen Codegenerator, der aus jedem Modell dieser Abstraktionsebene Code generieren kann. Zum Beispiel modellieren wir eine Schleife, die im Code auch als Schleife generiert wird.

2) *Domänenspezifische Code-Generatoren*: Hier wirken ein Modell einer höheren Abstraktionsebene sowie ein spezieller Code-Generator zusammen, der für ein Modell-Element des abstrakten Modells mit ein paar Zusatzinformationen, Code generiert. Zum Beispiel modellieren wir eine Aktion (ein Schritt in einem Prozess), die einen Datenbankzugriff auf eine Datenbank beschreibt. Zusätzlich definieren wir die Datenbank für die der Code generiert werden soll, z. B. SQL-Server. Der Code-Generator enthält die Regeln, mit deren Hilfe aus der abstrakten Beschreibung der gesamte Zugriffs-Code für die gegebene Datenbank generieren werden kann.

Beim domänenspezifischen Code-Generator sind viele Architekturregeln fix im Codegenerator eingebaut (Code entsteht direkt aus einem abstrakten Modell). Beim mehrstufigen Modelltransformations-Ansatz würden diese Architekturregeln in einem oder mehreren Modelltransformationsschritten angewendet werden. Jedes Ergebnismodell zeigt diese Architektur und lässt sich zusätzlich anpassen.

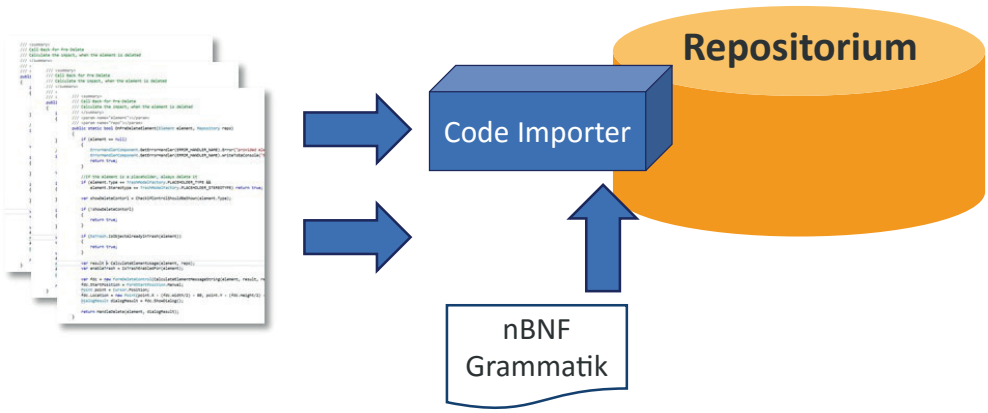
Da die Modelle in einer formalen Struktur vorhanden sind und auch programmatisch angesprochen werden können, ist es auch möglich eigene Code-Generatoren zu bauen oder Code-Generatoren anderer Firmen zu verwenden, um aus dem Modell den erwünschten Code zu generieren. Dabei kann direkt auf die Daten in der Datenbank zugegriffen werden oder über die mitgelieferte API. Werden die Modelle als XMI exportiert, können Code-Generatoren auch Modelle aus dem von der OMG standardisiertem Austauschformat lesen, um den Code zu generieren.



Vorhandenen Code einlesen

Enterprise Architect unterstützt auch das Einlesen von Code, damit kann Roundtrip Engineering etabliert werden. Änderungen sind dann im Code und/oder Modell durchführbar.

Es ist natürlich auch möglich, den vorhandenen Code lediglich einzulesen, um den aktuellen Implementierungsstand zu dokumentieren, ohne aus dem eingelesenen Modell jemals wieder Code zu generieren, bzw. den vorhandenen Code anzupassen. Der Code wird dabei nur im Code-File geändert, das Einlesen ist beliebig oft wiederholbar. Wurde der Code nun als Implementierungsmodell eingelesen, können abstraktere Modelle mit dem Implementierungsmodell verknüpft werden. Damit lässt sich die Nachvollziehbarkeit abstrakterer Modelle zur Implementierung erreichen, um die für den Leser des Modells relevanten Fragen beantworten zu können. Enterprise Architect kann mit einer Vielzahl von Programmiersprachen umgehen, die vom Code Importer eingelesen und als UML Modell dargestellt werden. Dieser Vorgang ist allerdings eine Blackbox und nicht anpassbar. Enterprise Architect bietet darüber hinaus die Möglichkeit, eigene Grammatiken beliebiger (Programmier-)Sprachen zu definieren. Mit diesem Ansatz lassen sich Programme von Sprachen einlesen, die das System sonst nicht unterstützt.

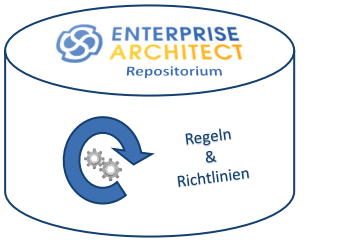


Verifikation der Modelle

Modellierungswerkzeuge bieten in der Regel die Möglichkeit, ein Modell zu überprüfen. Dabei unterscheiden wir die syntaktische Korrektheit der verwendeten Modellierungssprache und zusätzliche Regeln, mit denen das Modell nach eigenen Kriterien überprüft wird.

So lässt sich sicherstellen, dass der Inhalt des Modells zumindest den strukturellen Vorgaben eines Referenzmodells entspricht. Werden diese Richtlinien eingehalten, kann man davon ausgehen, dass alles, was aus dem Modell generiert wird, auch korrekt und vollständig ist.

Semantische Korrektheit ist natürlich nur bis zu einem gewissen Grad automatisch überprüfbar. Eine semantische Prüfung lässt sich durch aufwendigere Modellierungs-Regeln oder eine Modell-Simulation verifizieren.

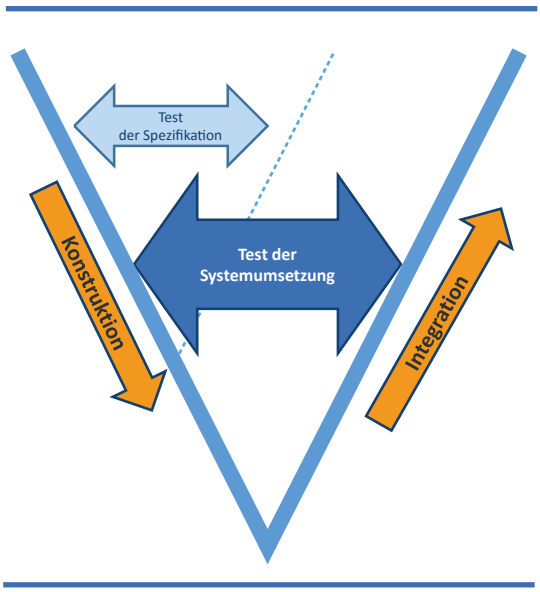


Simulation von Modellen

Die UML/SysML definiert neben der Syntax auch die Semantik, wie Verhaltensmodelle interpretiert werden. Enterprise Architect bietet die Möglichkeit, diese Modelle auch auszuführen. Dabei wird das Modell nach der gegebenen Sprachsemantik interpretiert. Findet sich ein Fehler im Modell, stoppt die Simulation an der Stelle und gibt eine Fehlermeldung aus.

Der Aufwand für simulierbare Modelle ist deswegen etwas größer, da man Vorgaben genauer einhalten muss. Der Vorteil dabei ist die Überprüfung der Spezifikation auf beliebiger Abstraktionsebene, was einer semantischen Verifikation entspricht.

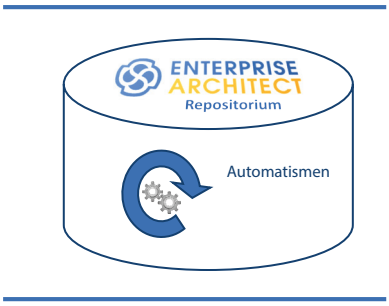
Im Sinne des V-Modells ist die Modell-Simulation die Verifikation der Spezifikation. Wir erhalten also ein kleines v in einem großem V.



Änderungen automatisch durchführen

Die UML/SysML bietet unterschiedliche Sichten zum Beschreiben eines Systems. Enterprise Architect unterstützt diese Sichten und bieten bereits eingebaute Automatismen, um die einzelnen Modelle in den unterschiedlichen Sichten synchron zu halten.

Je nach Einsatz der Modellierungssprache sind mehr oder weniger manuelle Schritte notwendig, um das gewünschte Modell zu erstellen und synchron zu halten.



Ist der manuelle Aufwand zu groß, um Modelle nach gewissen Regeln (dem Referenzmodell/Projekt-Metamodell) zu erstellen, können Automatismen in das Werkzeug eingebaut werden, die die Arbeit erleichtern. Dadurch erreicht man eine höhere Akzeptanz bei denjenigen, die das Modell erstellen und pflegen werden. Die Qualität der Modelle steigt dadurch automatisch, die nochmalige Bearbeitung fehlerhafter Modelle nach der Verifikation reduziert sich auf ein Minimum.

Testfälle automatisch generieren

Ein Modell kann die Struktur und Funktionsweise eines Systems beschreiben. Daher ist es naheliegend, dieses Modell nicht nur zur Erzeugung von Dokumenten, Code und anderen Artefakten zu verwenden, sondern auch zur Erstellung von Testfällen.

Bei der Erzeugung von Testfällen kommen oft nicht die konkreten Analyse-, Design- und Implementierungsmodelle zum Einsatz, sondern spezielle Testmodelle. Diese werden analysiert und mögliche Testfälle daraus generiert.

Enterprise Architect bietet derzeit keine der beschriebenen Möglichkeiten, mit Hilfe von Erweiterungen lassen sich aber automatisiert Testfälle aus einem Modell erzeugen.

Wo liegen die Hürden bei der Einführung modellbasierter Entwicklung?

Ein umfassendes Modellierungswerkzeug wie Enterprise Architect kann unterschiedlich verwendet werden, die Anzahl der Möglichkeiten wächst mit jeder neuen Version. Verschiedene Domänen nutzen das Werkzeug als ihre Plattform und entwickeln Erweiterungen, um die Anforderungen an einen domänenspezifischen, modellgetriebenen Ansatz besser zu bewältigen.
Aus welcher Domäne man nun kommt und wie auch immer man Modellierung mit Enterprise Architect einsetzen möchte: Es existieren zumindest vier bis fünf wichtige Themenbereiche, die in den nächsten Abschnitten genauer diskutiert werden.

- Die verwendete Modellierungssprache: Enterprise Architect beherrscht eine Vielzahl an unterschiedlichen Modellierungssprachen, die bekanntesten sind UML, SysML und BPMN. Der integrierte UML Profilmechanismus erlaubt es, eigene Modellierungssprachen zu entwickeln und dafür grafische Symbole, Toolboxen und Diagramm-Typen zu definieren. Jede Sprache hat allerdings ihre eigene Syntax und Semantik, die gelernt und beherrscht werden will. Ein Werkzeug wie Enterprise Architect kann bei dieser Aufgabe unterstützen.
- Das ausgewählte Werkzeug (Enterprise Architect): Enterprise Architect erlaubt es, verschiedene Modelle in verschiedenen Sprachen zu erstellen. Das lässt ihn auf den ersten Blick eher kompliziert erscheinen. Versteht man allerdings den Aufbau und die Struktur des Werkzeugs, fällt die Arbeit damit schnell sehr leicht.
- Die Vorgehensmethode: Eine Modellierungssprache gibt oft nicht vor, wie sie zu verwenden ist. UML/SysML zum Beispiel definieren lediglich die Syntax und die Semantik der zu verwendenden Symbole. Um ein Modell effizient erstellen zu können, sollte man eine definierten
- Struktur (Referenzmodell) folgen und Anfang und Ende des Modellierungsvorgangs festlegen. Dafür existieren Best-Practice Lösungen, aus denen sich der eigene Modellierungsansatz ableiten lässt.
- (Firmeninterne Prozesse): Neben den vier bereits skizzierten Themenbereichen spielen natürlich auch firmeninterne Prozesse eine Rolle. Zur Etablierung eines Modellierungsansatzes in der Firma müssen nämlich bestimmte Richtlinien und Bedingungen definiert und eingehalten werden. So sind zum Beispiel oft Anforderungen in einem bereits vorhandenen System erfasst und können von dort übernommen werden. Um einen Bruch in der Informationskette zu vermeiden ist darüber hinaus die nahtlose Integration zwischen den Werkzeugen erforderlich.

Die verwendete Modellierungssprache

Es existieren einige in der Praxis weit verbreitete Modellierungssprachen, die bekannteste ist sicherlich UML. Neben UML werden auch SysML und BPMN oft verwendet, UML ist allerdings die älteste und wohl allgemeinste Sprache. Mit UML lässt sich eigentlich schon „alles“ modellieren. Der Fokus von UML ist die Beschreibung von Software bzw. softwarenahen Systemen. Um den „Beigeschmack der Software“ aus UML herauszubekommen wurde SysML spezifiziert. SysML ist als spezielle Verfeinerung der UML schlanker, in einigen Bereichen allerdings konkreter und spezifischer.
UML und SysML sind also verwandte Sprachen, mit denen man verschiedene Aspekte eines (Software-) Systems beschreiben kann. Sie entspringen einer Sprachfamilie, die eine große Menge an Modell-Sichten zur Verfügung stellt. Die UML beinhaltet z. B. 14 Diagrammarten. Bei den neun SysML Diagrammarten sind sechs der UML-Diagramme weggefallen, dafür zwei neue hinzugekommen. Die 14. Diagrammart der UML ist der UML-Profilmechanismus, mit dem auch die SysML definiert wurde. UML und SysML stellen mehrere Modellarten zur Verfügung, meist bezeichnet man diese aber als Diagrammarten. Dieser kleine, aber feine Unterschied zeigt eine große Wirkung im Umgang mit einem Modellierungswerkzeug, da ein Diagramm

lediglich die grafische Darstellung des im Modell-Repositorium enthaltenen Modells ist.
Für eine Frage, die man mit dem Modell beantworten möchte, gibt es in der Regel mehrere mögliche Darstellungsformen. Ein Prozess lässt sich z. B. als Aktivitätsdiagramm, als Zustandsautomat oder als Sequenzdiagramm beschreiben oder mit der Sprache BPMN bzw. einer nicht so bekannten Modellierungssprache. Selbst wenn wir uns also auf die UML beschränken, haben wir drei Modellarten (Aktivitäten, Zustände, und Sequenzen), um Prozesse zu beschreiben. Um diesen großen Spielraum einzuschränken, müssen wir Regeln definieren, wann welches Modell für welchen Zweck zum Einsatz kommen soll.
Die UML bietet auch eine Menge an Beziehungen zur Verbindung der einzelnen Modellelemente. Einige sind von der Sprache so eingeschränkt, dass sie nur zwischen bestimmten Modellelementen erlaubt sind. Andere Beziehungsarten sind hingegen allgemeingültig und daher beliebig einsetzbar. Für die Abbildung einer Information im Modell sind also mehrere Möglichkeiten offen.
Betrachtet man ein Modell, so lässt sich die syntaktische Korrektheit (sogar automatisch) einfach überprüfen. Ob das Modell allerdings semantisch auch wirklich das wiedergibt, was sich der Ersteller gedacht hat, ist in der Regel nur durch einen Modell-Review überprüfbar. Um ein Modell als „gutes“ bzw. „passendes“ Modell einstufen zu können, muss man die Fragen kennen, die mit dem Modell beantwortet werden sollen.

Sprachelemente reduzieren und Verwendungsregeln definieren.

Viele der verwendeten Modellierungssprachen sind sehr umfangreich und bieten eine Fülle an Möglichkeiten, ein Problem zu beschreiben. Diese Möglichkeiten erschweren den täglichen Gebrauch der Modellierungssprache, vor allem wenn man im Team Modelle erstellt und austauscht. Daher sollte man sich möglichst rasch auf eine Sub-Menge an Sprachelementen der gewählten Modellierungssprache einigen und auch Regeln aufstellen, wann und wie sie verwendet werden dürfen. Der festgelegte Standard ist so gut wie möglich einzuhalten, lässt sich aber auch noch stärker einschränken. Darüber hinaus ist eine Anpassung möglich, wenn bestimmte Sprachelemente für die Beschreibung eines Problems oder einer Lösung fehlen. Mein Tipp: Anpassungen mit Bedacht durchführen bzw. nur wenn es unbedingt notwendig ist. Einschränkungen/Erweiterungen helfen jedenfalls dabei, die Komplexität der Modellierungssprache zu reduzieren und deren Verwendung zu vereinfachen, sowie das Verständnis der Inhalte der erstellten Modelle zu erhöhen.

Das ausgewählte Werkzeug

Neben der Modellierungssprache müssen wir uns auch mit dem gewählten Werkzeug auseinandersetzen, denn:

A fool with a tool is just a fool
↓
A genius without a tool is unproductive

In der Praxis ist man ohne Werkzeug einfach unproduktiv, sollte aber das für seinen Anwendungsfall am besten geeignete Werkzeug verwenden. Mit Enterprise Architect haben sie eine sehr gute Wahl getroffen, da er eine Plattform für unterschiedliche Anwendungsfälle ist. So können unterschiedliche Personengruppen dasselbe Werkzeug verwenden, um ihre Aufgaben zu erledigen. Lästige Brüche in der Werkzeugkette (Tool-Chain) entfallen. Die Daten im Projekt werden in einer Datenbank (Modell-Repositorium) gespeichert, neben Enterprise Architect sind auch andere Werkzeuge einsetzbar.
Natürlich ist Enterprise Architect nicht das einzige Werkzeug, das sie verwenden. Sie können ihn jedenfalls an ihre Bedürfnisse anpassen und in ihre Tool-Chain nahtlos einbetten. Viele Hersteller stellen für ihre eigenen Werkzeuge eine Integration von und

zum Enterprise Architect zur Verfügung. Eine einfache Web-Suche bringt schnell eine aktuelle Liste der verfügbaren Integrationen.
Das Ziel sollte sein, dass sie die grundlegende Struktur des Enterprise Architect kennen, um ihn ihrer Arbeitsweise bestmöglich anzupassen, damit:

- Oft benötigte Funktionen einfach auffindbar und leicht zu bedienen sind.
- Funktionen definiert werden, um die wichtigsten Fragestellungen die mit dem Modell beantwortet werden sollen, werkzeuggestützt und mit wenig Aufwand beantwortet werden können.

Die Vorgehensmethode

UML, SysML, BPMN und viele andere Modellierungssprachen definieren lediglich die Syntax und die Semantik der Sprache, aber nicht wie man mit ihnen Probleme lösen kann. Daher muss man sich auch mit der dafür anzuwendenden Methode auseinandersetzen.
In der Literatur findet sich eine Vielzahl an möglichen Vorgehensmethoden und Prozessen. In meiner langjährigen Praxis stellte ich aber fest, dass sich kein Unternehmen strikt an eine Vorgabe hält, unabhängig von den eingesetzten Methoden oder Prozessen. Methoden werden immer an firmeninterne Prozesse angepasst und durch eigene Erfahrungen weiter entwickelt. Ich vermute, dass es so viele Methoden wie innovative Köpfe gibt.
Bei der Analyse der einzelnen Methoden und Prozesse kristallisieren sich in der Regel viele Gemeinsamkeiten heraus. Mein Tipp: Gehen sie wieder von den zu beantwortenden Fragen aus und orientieren sie sich an einer bereits vorhandene Methode oder einem Vorgehens-Prozess. So finden sie die zu ihnen passende Vorgehensweise.
Grundsätzlich lassen sich zwei Arten von Methoden unterscheiden:

- Methoden, die das generelle Vorgehen beschreiben und definieren, wann welche Information in welcher Qualität zur Verfügung steht. So lassen sich Normen oft als Grundlage für eine Methode heranziehen, aber auch Agile Methoden wie SCRUM bzw. formalere Methoden wie der altbekannte RUP. Diese Methoden beschreiben oft nur global, welche Information vorhanden sein muss. Teils gehen sie aber auch bereits auf die Verwendung einer konkreten Modellierungssprache ein.
- Methoden für die Erstellung eines Modells, mit dem anschließend eine konkrete Frage beantwortet wird. Wie wir bereits sehen konnten ist die Beantwortung einer Frage ja oft durch verschiedene Modell möglich.

Im nächsten Kapitel gehen wir der Frage nach, wie man nun zur jeweils passenden Methode gelangt.

Die erforderliche Erfahrung

Eine Grundvoraussetzung für die Arbeit mit Modellen ist die Beherrschung des verwendetes Werkzeugs und der eingesetzten Modellierungssprache. Das alleine reicht aber noch nicht, um eine für die jeweilige Fragestellung optimale, modellbasierte Lösung zu erstellen. Erfahrung gewinnt man automatisch durch die Beschäftigung mit einem Thema und dem Lernen aus Fehlern. Es ist allerdings wesentlich effizienter, nicht nur aus den eigenen Fehler zu lernen, sondern auch aus den Fehlern der anderen. Wenn wir also von Erfahrung sprechen, dann sprechen wir von Best-Practice Lösungen.

Lerne aus Deinen Fehlern!

… noch besser: Lerne aus den Fehlern der anderen!