mehr zum thema:
**www.sophist.de/sophist-blog**

# STRUKTURIERTES MODELLIEREN: EIN MODELL IST MEHR ALS DIE SUMME SEINER DIAGRAMME

In Entwicklungsprojekten dienen Modelle in erster Linie der Kommunikation und Dokumentation. Daher sind die Lesbarkeit und Nachvollziehbarkeit von Entscheidungen und Informationen grundlegende Qualitätsfaktoren des Modells. Das in diesem Artikel vorgestellte systematische Vorgehen für die Strukturierung des Modells und die Verlinkung der Modellelemente hilft dabei, Informationen nachvollziehbar zu dokumentieren. Die aus der Systematik gewonnenen Regeln ermöglichen eine objektive und automatisierbare Qualitätskontrolle

die autoren

Anja Ranft
(E-Mail: Anja.Ranft@sophist.de)
ist bei der Sophist GmbH als Beraterin in den Bereichen Systemanalyse und Systemarchitektur tätig. In Projekten arbeitet sie an der Entwicklung und Modellierung großer technischer Systeme.

André Pflüger
(E-Mail: Andre.Pflueger@sophist.de)
ist für die Sophist GmbH als System-Engineer unterwegs und unterstützt die Kunden im Bereich der Anforderungs- und Systemanalyse sowie der Systemarchitektur.

## Warum Modelle eine feste Struktur benötigen

Die Zeiten loser Blattsammlungen und inkonsistenter Freihand-Zeichnungen in der modellgestützten Systementwicklung sind vorbei. Modelle sind heute weit mehr als nur ein paar grobe Skizzen, die überall auf dem Projektlaufwerk verteilen liegen. Wer bereits mit Modellen gearbeitet hat, weiß, wie umfangreich und aufwändig diese werden können. In der Analyse wird die Problemstellung in Form von Funktionen und benötigtem Wissen durchleuchtet, während im Design die Lösung – mit all ihren statischen und dynamischen Aspekten – möglichst umfassend wiederzugeben ist. Nebenbei dürfen auch die Verbindungen der Modellelemente untereinander und zu anderen Artefakten nicht fehlen. Mit diesem hohen Anspruch wachsen selbstverständlich auch die Komplexität und der Umfang der Modelle und es gilt, diese Komplexität zu beherrschen. Wem nutzt schon ein Modell mit tausenden von Elementen und hunderten von Diagrammen, das wegen mangelnder Struktur und Nachvollziehbarkeit niemand mehr verwenden kann und möchte?

Ein Modell soll uns zum einen als Qualitätsnachweis dienen und zum anderen eine Basis für spätere Änderungen, Erweiterungen und Fehlerlokalisierungen bieten. Diesen Zweck erfüllt es nicht, wenn Struktur und Nachvollziehbarkeit fehlen, denn dann produzieren Sie lediglich ein Wegwerf-Produkt. Um Qualität nachzuweisen, müssen wir zunächst Regeln definieren, die wir später – idealerweise automatisiert – prüfen können. Beispiele für solche Regeln sind:

- Jede Anforderung muss mindestens einer Komponente zugewiesen sein.
- Jede Komponente muss auf mindestens eine (funktionale oder nicht-funktionale) Anforderung zurückzuführen sein.

In unserem Beitrag geben wir praktische Tipps und Tricks für den Aufbau und die Pflege eines Analyse- und Design-Modells. Wir zeigen, wie Sie Ihr Modell stabil und einheitlich strukturieren, wie Sie durch Verlinkung der Diagramme und der Modellelemente das Modell für alle Beteiligten nachvollziehbar und lesbar gestalten. Zu guter Letzt geben wir noch einen Ausblick, wie Sie durch die Definition solcher festen Modellierungsregeln eine Grundlage für automatische Auswerte- und Prüfmöglichkeiten schaffen.

Neben den von uns behandelten methodischen Punkten spielen bei der Modellierung auch immer soziale Aspekte und Managemententscheidungen eine Rolle. Folgende Stolperfallen sind uns in Projekten des Öfteren begegnet:

- Das verwendete Vorgehensmodell fordert die Anforderungs- und Architekturdokumentation in einer anderen Form. Hier bereitet eine doppelte Pflege sowohl dieser Artefakte als auch des Modells meist zu viel Aufwand.
- Das Management ist nicht bereit, die benötigten Ressourcen freizugeben.
- Nicht alle an der Entwicklung beteiligten Personen stehen hinter dem modellbasierten Ansatz und sind bereit, mit dem Modell zu arbeiten.

Um ein böses Erwachen zu verhindern, sollten Sie daher unbedingt das Einverständnis und die Unterstützung aller Beteiligten und Betroffenen einholen, bevor Sie mit der Modellierung beginnen.

## Beispiel

Unser Vorgehen zum Aufbau eines Modells, zur Verlinkung von Modellelementen und zur Definition von prüfbaren Qualitätsregeln möchten wir im Folgenden anhand eines kleinen Beispiels erläutern.

Das zu entwickelnde System besteht aus Software- und Hardwareanteilen. Als Beispiel verwenden wir einen Audio/Video-Konverter, der zur Anzeige und Aufnahme von analogen Bild- und Tonsignalen, z. B. einem Fernsehsignal auf einem PC, verwen-

det werden kann. Dieser liegt als externes Hardwareelement vor, digitalisiert zunächst die eingehenden Daten und führt anschließend eine verlustbehaftete Konvertierung des digitalen Signals durch. Beispiele für diese Konvertierungen sind das MP3-Format für Ton- und das MPEG2-Format für Bildsignale. Der Endanwender kann mit Hilfe eines solchen Konverters zum Beispiel ein Fernsehsignal direkt als eine DVD-kompatible Datei abspeichern, ohne dass die CPU des PCs mit der Konvertierung der Signale belastet wird.

Für die Modellierung verwenden wir die *Unified Modelling Language (UML)* (vgl. [Rup07]).

## Abgrenzung: Modell und Diagramm

Um ein Modell verständlich und überschaubar zu gestalten, ist ein systematischer Aufbau unerlässlich. Eine Sammlung sämtlicher Diagramme in einem Tool macht noch kein Modell aus – erst wenn wir die Diagramme und die zugehörigen Elemente in einer nachvollziehbaren Struktur anlegen, kann man von einem Modell sprechen. Ein Modell ist die Summe aller Elemente innerhalb der Diagramme und deren Abhängigkeiten untereinander. Ein Modell:

- bildet einen Ausschnitt der realen Welt ab.
- vereint die Informationen aus allen Diagrammen.
- enthält weitere Informationen, die nicht in Diagrammen sichtbar sind (Eigenschaften von Modellelementen, Notizen, Regeln usw.).
- muss redundanzfrei und konsistent sein (jedes Modellelement darf nur einmal existieren).
- muss strukturiert und übersichtlich sein.

Modellierung bedeutet also immer Abstraktion und Ausschnittbildung. Daher ist eine grundlegende Frage, welche Aspekte im Modell abgebildet werden sollen. Diagramme hingegen:

- stellen verschiedene Sichten auf das Modell dar.
- enthalten immer nur Teilinformationen.
- sind aus Elementen des Modells aufgebaut (Klassen, Komponenten, Akteuren, Use-Cases usw.).
- können redundante Informationen enthalten (für eine bessere Lesbarkeit und Verständlichkeit).

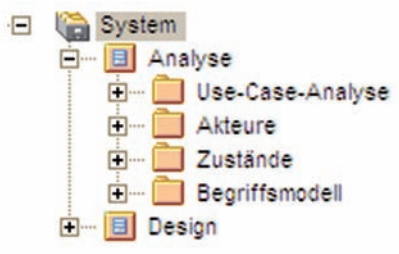


*Abb 1: Modellstruktur der Analyse.*

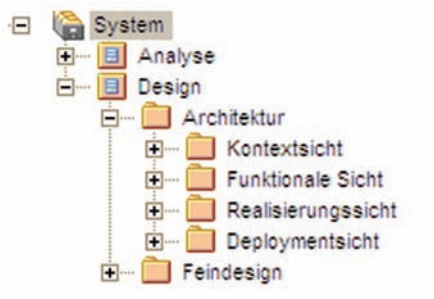


*Abb 2: Modellstruktur im Design.*

Vor der Erstellung eines Diagramms ist es wichtig zu fragen, welche Informationen darin gezeigt werden sollen (weniger ist manchmal mehr). Außerdem sollte man sich darüber Gedanken machen, wer das Diagramm lesen soll. Ein Use-Case-Diagramm für den Fachbereich ist anders zu gestalten als ein Komponenten-Diagramm für den Entwickler.

## Grundgerüst eines Modells

Kommen wir nun zu der Frage, wie Sie Ihr Modell strukturieren können. Gute Modellierungswerkzeuge bieten Ordner oder Pakete an, mit denen Sie eine Struktur – ähnlich den Verzeichnissen eines Dateisystems – anlegen können. Im ersten Schritt sollten Sie unbedingt die Analyse vom Design trennen. In der Analyse werden alle Anforderungen an das System definiert, im Design dann deren Umsetzung. Eine Trennung der Bereiche ist für die spätere Nachvollziehbarkeit grundlegend.

Ein Modell sollte das System immer aus mehreren Blickwinkeln betrachten. Im Systems-Engineering bezeichnen wir das als „Sichten". Sichten helfen Ihnen dabei, sich auf verschiedenen Aspekte (Dynamik, Statik, Zustände, Aktivitäten usw.) zu konzentrieren und mögliche Lücken in der Spezifikation zu entdecken. Innerhalb der Modellbereiche Analyse und Design sollten die verschiedenen Sichten voneinander getrennt dokumentiert werden. In der Analyse könnten das sein:

- eine funktionsorientierte Sicht (z. B. Use-Cases und Aktivitäten oder Interaktionen zu deren Verfeinerung)
- eine zustandsorientierte Sicht (z. B. Zustandsautomaten oder Timing-Diagramme)
- eine begriffsorientierte Sicht (z. B. ein Klassendiagramm als Fachmodell)

Ein Beispiel für eine solche Struktur in der Analyse zeigt Abbildung 1.

Im Design unterscheidet man zwischen der Architektur und dem Feindesign. Während die Architektur den groben Bauplan des Systems vorgibt, wird im Feinddesign die Realisierung der in der Architektur gefundenen Komponenten dargestellt. In der Architektur gibt es die verschiedensten Ansätze zur Sichtenbildung (vgl. z. B. [Sta02], [Reu09]). Auch hier sollte man die verschiedenen Sichten trennen und die Ordner auch innerhalb jeder Sicht nutzen, um die Diagramme und Modellelemente zu gruppieren. Ein möglicher Aufbau der Architektur ist:

- *Systemkontext:* Einbettung des Systems in seine Umgebung, Darstellung der Nachbarsysteme und Akteure des Systems und deren Abhängigkeiten.
- *Funktionale Sicht:* Zerlegung des Systems aus rein funktionaler Sicht, d. h. unabhängig von Implementierung und Hardware, Abbildung der Aufgaben aus der Analyse (Aktivitäten und Use-Cases) auf die funktionalen Komponenten.
- *Realisierungssicht:* Verteilen der funktionalen Komponenten auf reale Komponenten (ausführbare Einheiten).
- *Deployment-Sicht:* Darstellung der Hardwarelandschaft und der Verteilung des Systems. Abbildung 2 zeigt eine mögliche Struktur für das Design.

Zum Abschluss noch ein paar praktische Tipps zur Pflege eines Modells: Halten Sie in Ihrem Modell penible Ordnung. Das heißt, räumen Sie alle Modellelemente sofort auf und löschen Sie nicht verwendete Elemente sofort aus dem Diagramm bzw.

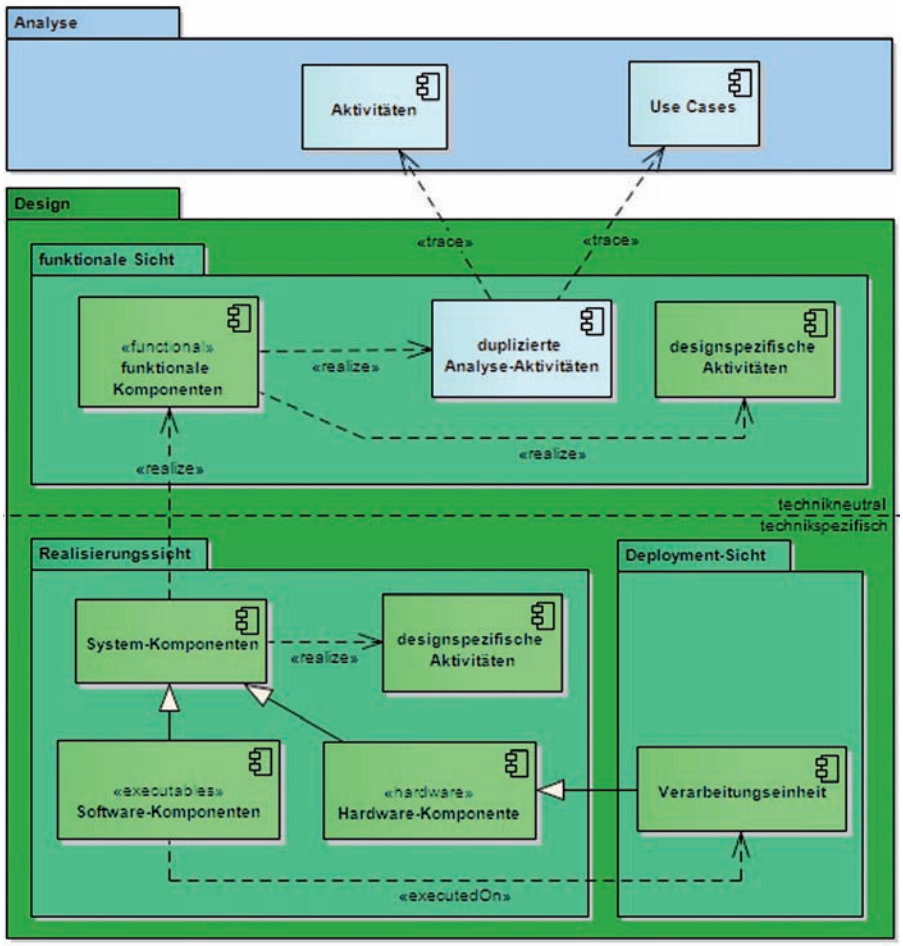


*Abb 3: Übersicht der Verbindungen im Modell.*

dem Modell. Dabei ist es ein großer Unterschied, ob man ein Element aus einem Diagramm oder aus dem Modell löscht. Löschen Sie ein Element nur aus einem Diagramm, so bleibt es im Modell bestehen, auch wenn es nirgendwo mehr verwendet wird. Aus dem Modell entfernte Elemente werden aus allen Diagrammen entfernt, in denen sie enthalten sind. Ein sehr hilfreiches Mittel, um die Navigation im Modell zu erleichtern, sind die von manchen Tools ermöglichten Hyperlinks. Mit diesen kann man zwischen Diagrammen, die eng miteinander verbunden sind, springen.

## Nachvollziehbarkeit von Informationen

Die Grundstruktur verbindet die Modellelemente mit den Bereichen Analyse und Design sowie den darunter liegenden Sichten. Darüber hinaus empfehlen wir Ihnen, die Elemente untereinander zu verknüpfen und somit die bestehende Struktur um eine semantische Struktur zu erweitern. Das ermöglicht es Ihnen, die im Modell abgelegten Informationen über die verschiedenen Bereiche und Sichten hinweg zu verfolgen und eine automatisierte Prüfung der oben angesprochenen Regeln durchzuführen. Einige Auswertungsmöglichkeiten stellen wir im Folgenden vor.

Wir konzentrieren uns in diesem Artikel auf folgende Verknüpfungsmöglichkeiten:

- *Verfeinerung von Modellelementen:* Innerhalb eines Bereichs oder über einen Bereich hinaus.
- *Übergang zwischen Analyse und Design:* Verbindung von Aktivitäten und funktionalen Komponenten oder Aktivitäten und Soft-/Hardware-Komponenten.
- *Verbindung von Modellelementen aus verschiedenen Sichten:* Funktionale Sicht und Realisierungssicht oder Realisierungs- und Deployment-Sicht.

Abbildung 3 zeigt die Verknüpfungen in den Bereichen Analyse und Architektur schematisch. Das Feindesign, das die Realisierung der Softwarekomponenten durch Softwareklassen darstellt, haben wir aus Gründen der Übersichtlichkeit weggelassen.

## Verfeinerung von Modellelementen

Bei der Verfeinerung von Modellelementen unterschieden wir die beiden folgenden Fälle:

### 1. Verfeinerung von Modellelementen innerhalb eines Bereichs

Die Verfeinerung von Modellelementen innerhalb eines Bereichs wird von guten Modellierungswerkzeugen dadurch unterstützt, dass ein Modellelement mit einem entsprechenden Diagramm verknüpft wird, das eine detailliertere Beschreibung enthält. Beispiele hierfür sind die Verknüpfung einer Aktivität oder eines Use-Cases mit einem Aktivitätsdiagramm, einer Systemkomponente durch Software- und Hardwarekomponenten oder einer Softwarekomponente durch ein Klassendiagramm im Feindesign. Die Verknüpfung wird durch das Modellierungswerkzeug hergestellt. Wie Sie diese Elemente im Modell ablegen können, zeigt Abbildung 4.

In unserem Beispiel des Audio-/Video-Konverters wird der Use-Case „analoges Signal konvertieren" durch ein Aktivitätsdiagramm verfeinert. Legen Sie für das Elternelement, in diesem Beispiel der Use-Case „analoges Signal konvertieren", einen Ordner an, in dem folgende Elemente abgelegt werden:

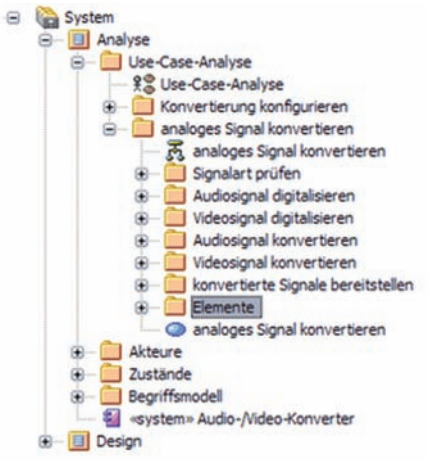


*Abb 4: Bereichsinterne Verfeinerung einer Aktivität.*

- das Elternelement selbst
- das neue Diagramm
- die dazugehörigen Elemente

Die dazugehörigen Elemente bestehen in diesem Beispiel aus sechs Aktivitäten (Kindelemente) und den nicht weiter verfeinerbaren Modellelementen, beispielsweise Start-, End- oder Entscheidungsknoten. Letztere können Sie in einem Sammelordner ablegen. In unserem Beispiel ist dies der Ordner „Elemente". Unser Tipp: Legen Sie für jedes Kindelement – auch für die nicht verfeinerten – einen Ordner an. Das fördert ein durchgängiges, einheitliches Erscheinungsbild im Modell.

**2. Verfeinerung von Modellelementen über einen Bereich hinaus**

Die vorgestellte Art der Verknüpfung eignet sich für Verfeinerungen innerhalb eines Bereichs oder einer Sicht, da die Elemente in der Hierarchiestruktur des Modells unterhalb des verfeinerten Elementes abgelegt werden. Sie ist aber ungeeignet, um beispielsweise Elemente aus der Analyse im Design zu verfeinern, ohne nachträglich die Ergebnisse der Analyse zu beeinflussen. Für diesen Fall empfehlen wir die Duplizierung der betroffenen Elemente. Das Duplikat kann dann unabhängig von der Analyse verfeinert werden. Eine Veränderung des Ursprungselements bleibt auf die Analyse beschränkt. Dadurch sind unterschiedliche Modellierungen in der Analyse und im Design möglich. Da die Verfeinerung designspezifisch ist, wird das Designelement auch im Bereich Design abgelegt. Die semantische Verbindung zwischen diesen beiden Elementen wird über eine *Trace*-Beziehung hergestellt.

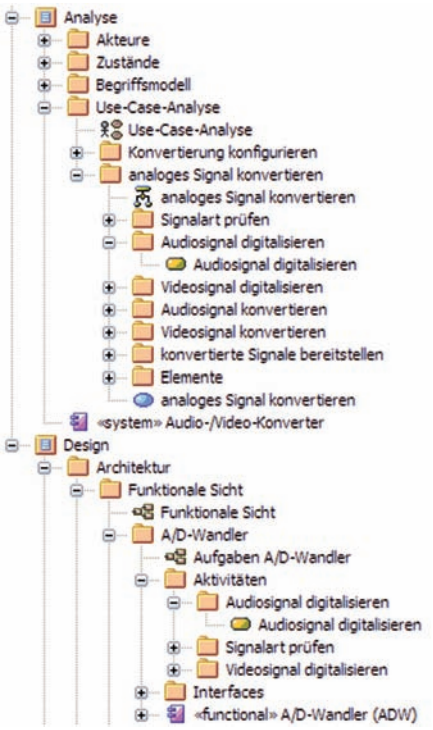


*Abb 5: Bereichsübergreifende Verfeinerung einer Aktivität.*

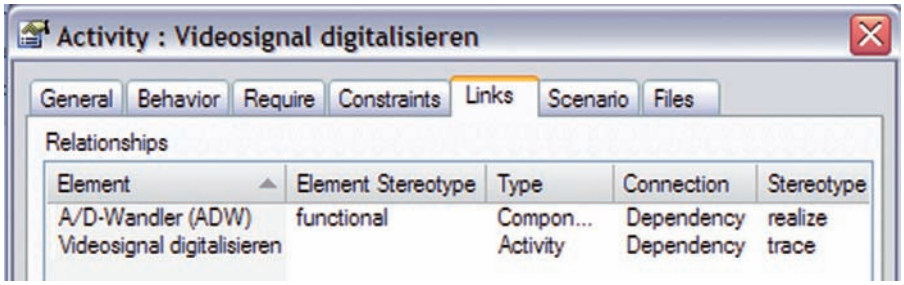


*Abb 6: Sichtbarkeit der Trace-Beziehung in den Eigenschaften des Elements.*

Um diese Art der Verbindung anhand unseres Beispiels zu erläutern, stellen wir uns vor, wir haben in der Analyse eine Aktivität „Audiosignal digitalisieren" definiert. Diese muss anschließend in der Designphase weiter verfeinert werden, um beispielsweise zu entscheiden, welche Komponenten an der Realisierung der Aktivität beteiligt sind oder welche Hardware zur Umsetzung benötigt wird. Wir duplizieren also die Analyseaktivität „Audiosignal digitalisieren" und legen das Duplikat sowie die Verfeinerung im Designordner „A/D-Wandler" (siehe Abbildung 5) ab. Außerdem verbinden wir das Duplikat mit dem Original über eine *Trace*-Beziehung. Die *Trace*-Beziehung muss übrigens nicht in einem Diagramm sichtbar sein. Es genügt, wenn diese über die Elementeigenschaften nachvollziehbar ist (siehe Abbildung 6).

## Übergang zwischen Analyse und Design

Bei der Erstellung einer Systemarchitektur können Sie zunächst eine funktionale Sicht erarbeiten. Das ist hilfreich, wenn beispielsweise die zu verwendende Hardware noch nicht bekannt ist oder festgelegt werden kann. Das System wird dabei in funktionale Komponenten aufgeteilt, die auch weiter verfeinert werden können. Damit Sie nachvollziehen können, welche funktionale Komponente welche Aufgaben aus der Analyse realisiert, empfiehlt es sich, zwischen den Elementen eine *Realize*-Beziehung zu erstellen.

In unserem Beispiel des Audio-/Video-Konverters existiert eine funktionale Komponente „A/D-Wandler". Diese hat die Aufgabe, die Aktivitäten „Audiosignale digitalisieren" und „Videosignale digitalisieren" zu realisieren (siehe Abbildung 7). Wie Sie die Elemente in Ihrer Modellstruktur ablegen können, zeigt Abbildung 8. Wir empfehlen, für jede funktionale Komponente einen Ordner mit folgendem Inhalt anzulegen:

- die funktionale Komponente selbst
- ein Diagramm „Aufgaben <Name der funktionalen Komponente>"
- ein Ordner für Aktivitäten der Komponente
- ein Ordner für Interfaces

Der Ordner „Aktivitäten" enthält alle Aufgaben, die die funktionale Komponente realisieren muss. Dies können sowohl die oben angesprochenen Duplikate aus der Analyse und deren Verfeinerungen sein, als auch durch den Realisierungsaspekt neu hinzukommende Anforderungen (in Abbildung 3 als „Designspezifische Aktivitäten" bezeichnet).

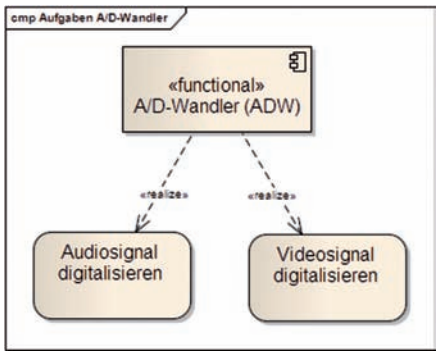


*Abb 7: Zuweisung von Aufgaben an eine funktionale Komponente.*

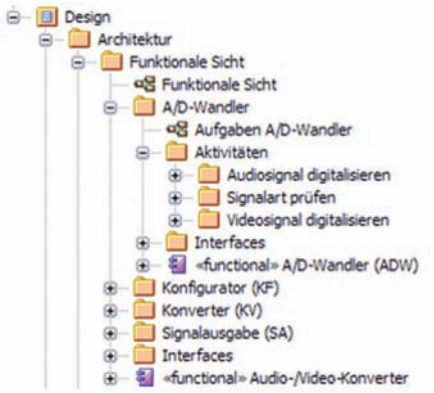


*Abb 8: Modellstruktur der funktionalen Sicht.*

Wenn Sie bei der Erarbeitung der Systemarchitektur keine funktionale Sicht benötigen, können Sie das an den funktionalen Komponenten vorgestellte Prinzip auch auf die Software- und Hardwarekomponenten übertragen.

## Verbindung von Modellelementen in verschiedenen Sichten

Wie im normalen Leben gibt es auch beim Modellieren eines Systems verschiedene Sichten auf einen bestimmten Aspekt. Um diesen richtig und vollständig verstehen zu können, sind neben den Sichten auch die Zusammenhänge zwischen diesen von großer Bedeutung. Am Beispiel der Architektur wollen wir zeigen, wie man die Elemente der verschiedenen Sichten miteinander verbinden und somit von der Analyse bis zur Deployment-Sicht verfolgen kann.

### Verbindung zwischen funktionaler Sicht und Realisierungssicht

In der funktionalen Sicht werden funktionale Komponenten mit den ihnen zugewiesenen Aufgaben über eine *Realize*-Beziehung verbunden. Das gleiche Prinzip können wir auch auf die Verknüpfung der funktionalen Sicht mit der Realisierungssicht anwenden. In der Realisierungssicht werden Komponenten definiert, die die funktionalen Komponenten realisieren. In unserem Beispiel können das sowohl Softwarekomponenten (so genannte *executables*), als auch Hardwarekomponenten sein. Die Komponenten aus der Realisierungssicht werden über eine *Realize*-Beziehung mit den funktionalen Komponenten verbunden.

In unserem Beispiel wird die funktionale Komponente „A/D-Wandler" von einer gleichnamigen Hardwarekomponente realisiert (siehe Komponentendiagramm in Abbildung 9). Darüber hinaus realisiert die Hardwarekomponente die designspezifische Aktivität „Digitalisierung konfigurieren", die auf den Use-Case „Konverter konfigurieren" zurückzuführen ist. Diese Aktivität hat sich erst durch die Auswahl der Hardware ergeben und ist deshalb keiner funktionalen Komponente zugeordnet. Die Modellstruktur in der Realisierungssicht ähnelt der in der funktionalen Sicht. Wir empfehlen, für jedes realisierende Element einen Ordner anzulegen, der das Element selbst und ein Diagramm „Aufgaben <Name der Komponente>" enthält. Die auch auf dieser Ebene möglicherweise hinzukommenden designspezifischen Aktivitäten und die Interfaces werden in den jeweiligen gleichnamigen Ordnern abgelegt. Die Modellstruktur der Realisierungssicht zeigt Abbildung 10.

### Verbindung zwischen Realisierungs- und Deployment-Sicht

In der Realisierungssicht haben wir die Zerlegung des Systems in Hardware- und Softwarebestandteile definiert und diesen ihre Aufgaben zugewiesen. In der Deployment-Sicht müssen wir nun noch die Verteilung der Softwarekomponenten auf die Verarbeitungseinheiten darstellen. Hierfür verwenden wir erneut die Abhängigkeitsbeziehung, dieses Mal mit dem Stereotyp *executedOn*. Alle Softwarekomponenten müssen also über eine *executedOn*-Beziehung mit der zugehörigen Hardwarekomponente verbunden werden.

Abbildung 11 zeigt das entsprechende Komponentendiagramm für den Audio/-Video-Konverter. Hier werden die beiden Softwarekomponenten „Konfigurator" und „Signalausgabe" auf die Hardwarekomponente „Power PC" abgebildet, die beiden Komponenten „Audiokonverter" und „Videokonverter" hingegen auf den *Field Programmable Gate Array (FPGA)*.

## Auswertungsmöglichkeiten

Die vorgestellte Modellstruktur bildet die Grundlage für ein überschaubares und strukturell nachvollziehbares Modell. Die Verknüpfung der Modellelemente fügt eine semantische Struktur hinzu. Durch das darunter liegende Metamodell der UML lassen sich die Modellelemente mit ihren Beziehungen untereinander automatisiert auswerten. Im Folgenden stellen wir drei Auswertungsmöglichkeiten beispielhaft vor.

### Konsistenzprüfung

Bei der Festlegung von Schnittstellen empfehlen wir, zu jedem *Required Interface* direkt ein *Provided Interface* zu modellieren. Das Fehlen eines Teils dieses Paares führt zu einer inkonsistenten Schnittstellenbeschreibung. Das Phänomen der „einsamen Schnittstellen" beobachten wir sowohl in großen als auch in kleineren Projekten.

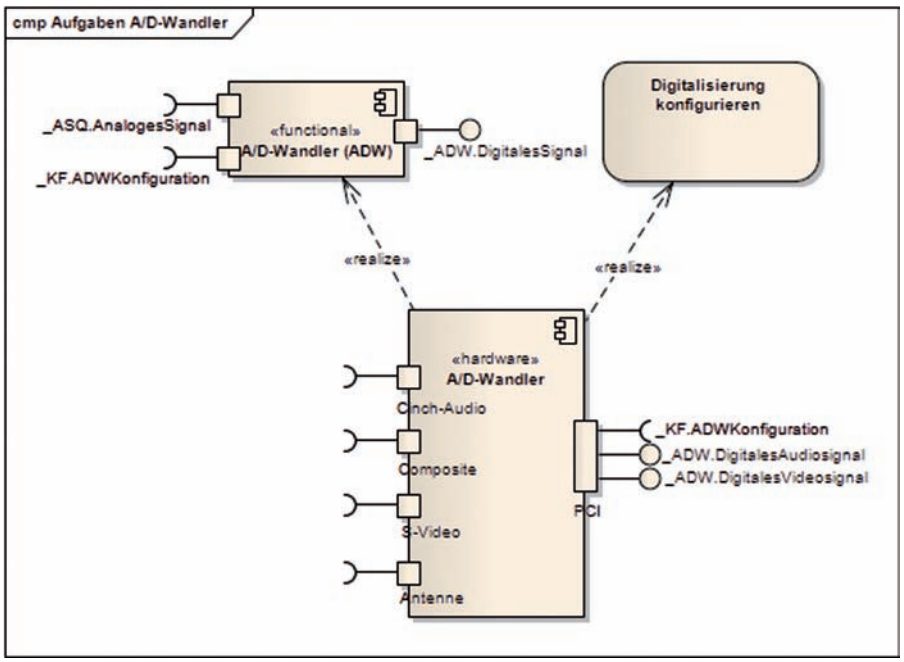


*Abb 9: Verbindung zwischen funktionaler Sicht und Systemsicht.*


*Abb 10: Modellstruktur der Realisierungssicht.*

Schnittstellen gehören zusammen, wenn ihre Namen identisch sind. Die Modellierung einer Abhängigkeitsbeziehung vom *Required Interface* zum *Provided Interface* ist hilfreich, aber optional. Sie erleichtert die manuelle Suche nach Inkonsistenzen, führt aber auch zu Unübersichtlichkeit in großen Diagrammen und wird deshalb gerne weggelassen.

Des Weiteren empfehlen wir, beim Anlegen von Modellelementen direkt die semantische Struktur mit aufzubauen bzw. bei Änderungen anzupassen. Ansonsten entstehen selbst bei kleinen Projekten schnell „verwaiste" Elemente, also Elemente ohne Verbindung zur semantischen Struktur. Sie vermeiden dadurch beispielsweise:

- funktionale Komponenten oder Systemkomponenten ohne Funktionalität
- im Design vergessene Analyseelemente
- Softwarekomponenten, die keiner Verarbeitungseinheit zugeordnet sind

Eine manuelle Konsistenzprüfung ist in größeren Projekten mit hohem Aufwand verbunden. Dieser lässt sich durch ein Tool verringern, das ein Modell auf Basis des UML-Metamodells nach „einsamen" Schnittstellen oder verwaisten Elementen durchsucht. Einige Modellierungswerkzeuge haben solche Tools integriert. Wenn Ihnen deren Möglichkeiten nicht ausreichen, können Sie auch externe Werkzeuge, wie beispielsweise „OCL" (vgl. [OCL10]) oder Xpand (vgl. [Xpa10]) von Eclipse verwenden.

Die automatisierte Konsistenzprüfung bietet den Vorteil, dass Sie die Qualität Ihres Modells deutlich steigern. Durch den wiederholten Einsatz lässt sich diese Qualität auch mit geringem Aufwand beibehalten. Die zuvor investierte Zeit zur Erstellung des Prüfungswerkzeugs amortisiert sich nach unseren Erfahrungen bereits nach wenigen Einsätzen.

**Abdeckung der Anforderungen**

Wir greifen an dieser Stelle das in Abbildung 3 gezeigte Übersichtsbild wieder auf.

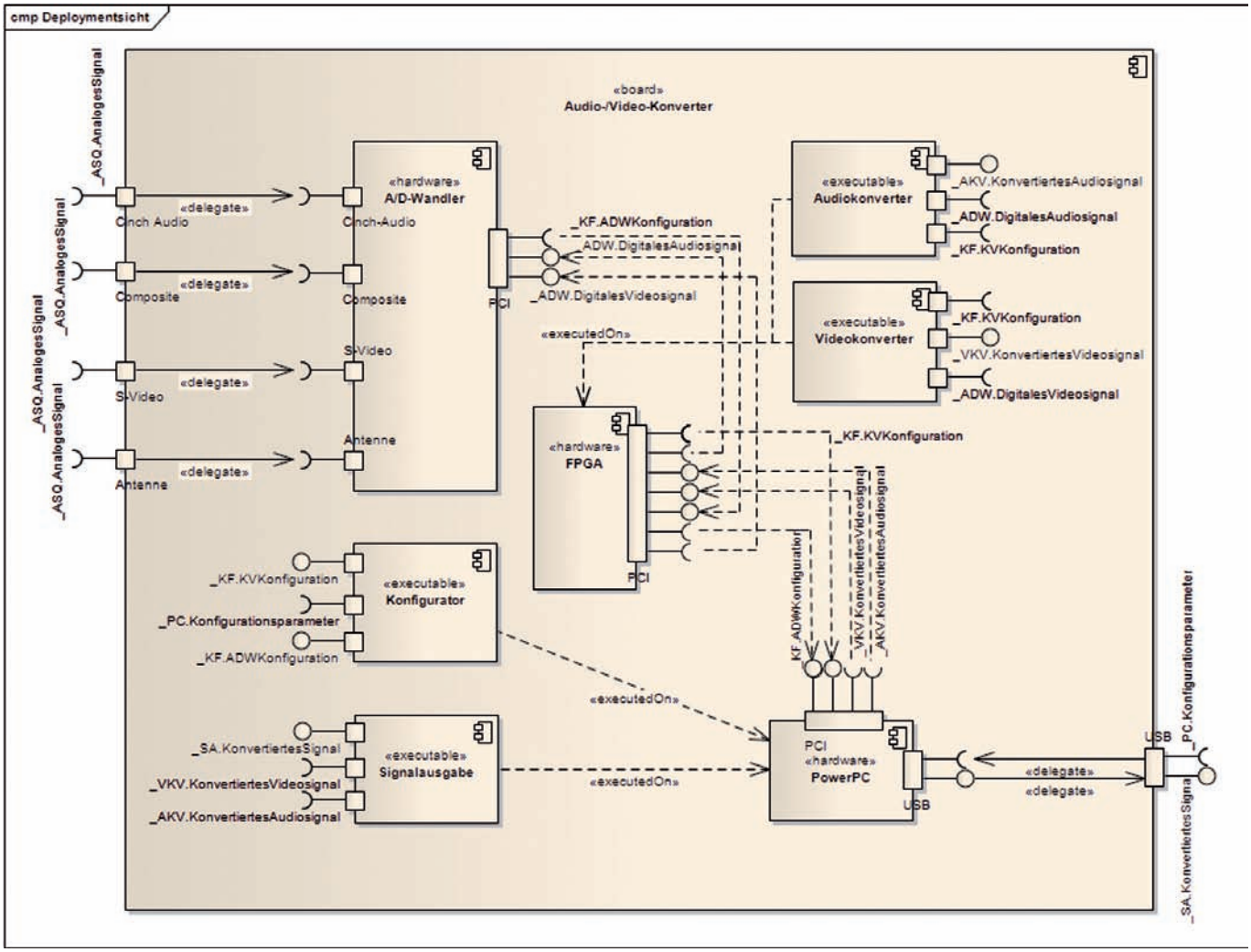


*Abb 11: Verbindung von Realisierungssicht und Deployment-Sicht.*

In dieser Abbildung sehen Sie die zwei Bereiche Analyse und Design sowie die Sichten des Designs mit den dazugehörigen Modellelementen. Außerdem sehen Sie die vorgestellten Verknüpfungsmöglichkeiten, die es ermöglichen, jedes Modellelement von oben nach unten zu verfolgen.

Durch eine Verlinkung der Elemente aus dem Analysebereich im Modell mit ihren natürlich-sprachlichen Anforderungen lässt sich nachvollziehen, welche Anforderungen im Modell berücksichtigt sind. Auf diese Weise lassen sich sowohl beim Modellieren vergessene Anforderungen als auch Modellelemente ohne zugeordnete Anforderungen identifizieren. Letzteres sollte in der Analyse nicht vorkommen und kann ein Hinweis auf unvollständige Anforderungen oder Goldrand-Lösungen sein. Architekturspezifische Aktivitäten hingegen können in Abhängigkeit vom Detaillierungsgrad der Anforderungen auch schon mal „vom Himmel fallen", also keine zugeordnete Anforderung haben.

Des Weiteren lässt sich über die Zuordnung der Analyseelemente zu den Designelementen prüfen, ob jede definierte Aufgabe auch einer Komponente zugewiesen ist, ob also bei der Umsetzung auch jede Aufgabe berücksichtigt wurde.

**Unterstützung bei der Impact-Analyse**

Ein Projekt wird selten so umgesetzt, wie es der Kunde bei der ersten Festlegung der Anforderungen spezifiziert. Im Laufe der Systementwicklung ergeben sich immer wieder Anforderungsänderungen. Sie müssen dann abschätzen, wie sich diese Änderungen auf das System auswirken. Die vorgestellte Modellstruktur unterstützt dabei, indem sie zeigt, mit welchen Modellelementen eine geänderte Anforderung verbunden ist. Dadurch lässt sich sowohl auf der Analyse- als auch auf der Designebene jedes Modellelement ermitteln, das von der Änderung betroffen ist. Mit Hilfe eines Tools lässt sich diese Aufgabe ebenfalls automatisieren.

## Fazit

Wir hoffen, Ihnen in diesem Artikel ein paar gute Tipps für die Erstellung Ihres nächsten Modells gegeben zu haben. Darüber hinaus gibt es selbstverständlich noch viele weitere Aspekte, die es bei der Modellierung zu berücksichtigen gilt und von denen wir hier abstrahiert haben. Beispiele hierfür sind die Verbindung zu modellexternen Artefakten, wie beispielsweise Anforderungen, Testfälle, Schnittstellenbeschreibungen und Änderungsanforderungen.

Der genaue Aufbau des Modells und die benötigten Beziehungen zwischen den Modellelementen hängen immer von den individuellen Projektrandbedingungen und dem zu entwickelnden Produkt ab. Trotzdem können die hier vorgestellte grundlegende Struktur und Verbindungen unserer Erfahrung nach in den meisten Fällen eingesetzt und bei Bedarf angepasst werden. Sie helfen, die Komplexität und den Umfang der Modelle zu beherrschen, indem sie Übersichtlichkeit und Nachvollziehbarkeit gewährleisten. ■

## Literatur & Links

**[OCL10]** Webseite des Eclipse OCL-Projekts, siehe: http://www.eclipse.org/modeling/mdt/?project=ocl (Zugriff im Juli 2010)

**[Reu09]** R. Reussner, W. Hasselbring, Handbuch der Software-Architekturen, dpunkt.verlag 2009

**[Rup07]** C. Rupp, S. Queins, B. Zengler u.a., UML 2 glasklar, Hanser 2007

**[Sta02]** G. Starke, Effektive Software-Architekturen, Hanser 2002

**[Xpa10]** Website des Eclipse Xpand-Projekts, siehe: http://www.eclipse.org/modeling/m2t/?project=xpand (Zugriff im Juli 2010)