Plädoyer für ein einheitliches Grundgerüst bei der System- und Softwaremodellierung*

B. Paech
Institut für Informatik, Technische Universität München
D-80333 München, Germany
email: paech@informatik.tu-muenchen.de

Zusammenfassung

In diesem Beitrag stellen wir einen Grundgerüst zur Einordnung von Modellierungstechniken in der System- und Softwareentwicklung vor. Dieses Grundgerüst macht die Vielzahl der bei der Modellierung relevanten Fragestellungen deutlich, zeigt aber auch einen Weg um diese Vielfalt zu systematisieren. Mit diesem Beitrag soll keine endgültige Lösung für diese Systematisierung vorgestellt werden, sondern eine Diskussionsgrundlage für die Frage geschaffen werden, ob ein einheitliches Grundgerüst möglich und sinnvoll ist bzw. wie es aussehen sollte.

In diesem Workshop werden verschiedene Modellierungskonzepte und -vorgehensweisen verglichen. Angesichts der Fülle der unterschiedlichen Modellierungstechniken in der Literatur und der industriellen Praxis ist diese Gegenüberstellung dringend erforderlich. Dies wurde auch bei einer Erhebung unter 17 Industriepartnern im Rahmen des Bayerischen Forschungverbunds Software Engineering [BEF+96] bestätigt [DHP+98]. Dort wurde insbesondere eine klare Bewertung gefordert, welche Modellierungstechniken zu welchen Zweck verwendet werden können. Diese Bewertung erfordert allerdings einen Konsens darüber, was die wesentlichen Kriterien bei der Verwendung von Modellen in der System- und Softwaremodellierung sind. Ob dieser Konsens zu erreichen ist, wird sich im Verlauf des Workshops zeigen.
Im folgenden stelle ich ein Grundgerüst zur Einordnung der verschiedenen Modellierungstechniken vor. Dieses Grundgerüst macht die Vielzahl der bei der Modellierung relevanten Fragestellungen deutlich, zeigt aber auch einen Weg um diese Vielfalt zu systematisieren. Mit diesem Beitrag soll keine endgültige Lösung für diese Systematisierung vorgestellt werden, sondern eine Diskussionsgrundlage für die Frage geschaffen werden, ob ein einheitliches Grundgerüst möglich und sinnvoll ist bzw. wie es aussehen sollte. Die Kernpunkt dieses Grundgerüsts sind die folgenden drei Fragen:

- Was wird modelliert?
- Welche Aspekte werden modelliert?
- Wie werden die Modelle im Prozeß der System- und Softwareentwicklung verwendet?

Wir gehen nachfolgend auf diese Fragen näher ein. Dabei fokussieren wir auf graphische Modellierungstechniken, die typischerweise semi-formal sind (d.h. mit festgelegter Syntax, aber ohne festgelegte Semantik). Am Schluß des Beitrags diskutieren wir noch gesondert die Einbindung formaler Modellierungstechniken.

1 Was wird modelliert?

Im Lauf der System- und Softwareentwicklung werden typischerweise drei verschiedene Systeme betrachtet:

- das *Anwendungssystem*, d.h. das Umfeld des zu entwickelnden Softwaresystems - bei Informationssystemen typischerweise ein Unternehmensausschnitt,

*This paper originated in the ForSoft project supported by the Bayerische Forschungsstiftung.

- das *Nutzungssystem*, d.h. die Interaktion zwischen BenutzerInnen und Softwaresystem bei der Aufgabenerfüllung, und
- das *Softwaresystem* selbst.

Bei der Modellierung werden diese Systeme oft nicht klar getrennt. So wird z.B. die Modellierung der Aufgabenerfüllung im Anwendungssystem oft mit der Modellierung des Anwendungskerns im Softwaresystem vermischt.
Typische Modellierungstechniken für das Anwendungssystem sind heute Daten- bzw. Geschäftsprozeßmodelle. Letzter thematisieren vor allem die Beziehung zu den Kunden. Sozio-technische Ansätze [SN93] fokussieren dagegen mehr auf die Bedürfnisse der Mitarbeiterinnen und Mitarbeiter im Unternehmen.
Die Modellierung des Nutzungssystems wurde lange Zeit als eigenständiges Forschungsgebiet (Software-Ergonomie) angesehen, und wird erst in letzter Zeit zunehmend als Teil der Softwareentwicklung betrachtet. Ein wichtiger Beitrag dazu waren die Use Cases von Jacobson [Jac92]. Typischerweise können bei den Modellen im Nutzungssystem sehr viele verschiedene Abstraktionsebenen unterschieden werden: von den Arbeitsaufgaben über die Systemfunktionsaufrufe oder Dialoge bis hin zu einzelnen Mausklicks.
Auch bei der Modellierung des Softwaresystems sind verschiedene Abstraktionsebenen zu unterscheiden: von der konzeptuellen Beschreibung bis hin zu Code.
Da alle drei Systeme auch als informationsverarbeitende Systeme angesehen werden können, ist es sinnvoll die gleichen Modellierungstechniken für die verschiedenen Systeme einzusetzen. Allerdings ist sorgfältig abzuwägen, welche Aspekte wichtig sind, und welche Abstraktionsebene angemessen ist: so reicht es bei der Anwendungssystemmodellierung nicht aus, nur Daten und Verhalten zu modellieren. Sehr wichtig sind auch die Interessen und Ziele der menschlichen Akteure im Anwendungssystem (siehe z.B. [Yu97]). Weiterhin ist eine sehr detaillierte Verhaltensmodellierung des Anwendungssystems nicht sinnvoll, im Gegensatz zum Softwaresystem.
Eine erste Einordnung von Modellierungstechniken ist also möglich im Hinblick auf die Art des modellierten Systems. Eine detailliertere Betrachtung muß sich auf die modellierten Systemaspekte beziehen (siehe den nächsten Abschnitt). Um die Angemessenheit der Modellierungstechnik für das jeweilige System bewerten zu können, ist eine Priorisierung der Aspekte und der Abstraktionsebenen für die jeweiligen Systeme nötig.

2 Welche Aspekte werden modelliert?

Abbildung 1 zeigt die typischen Aspekte eines informationsverarbeitenden Systems in einem Metamodell. Anhand dieser Aspekte lassen sich viele verschiedene Modellierungstechniken einordnen. Das Systemmodell ist stark beeinflußt von Überlegungen zur formalen Fundierung von Modellierungstechniken mit einem mathematischen Systemmodell [BHH+97, Pae96]. Die hier angegebenen Beziehungen zwischen den Konzepten haben keine formale Semantik, sondern repräsentieren nur wichtige Zusammenhänge, die typischerweise mit den Modellierungstechniken erfaßt werden. Die Raute steht für die `besteht-aus`-Beziehung, der Pfeil für die `ist-ein`-Beziehung.
Im folgenden beschreiben wir kurz die einzelnen Aspekte. Die Beziehungen sind in der Erklärung kursiv hervorgehoben.

Akteur/System Akteure sind die wesentlichen Bestandteile eines Systems. Die Akteure kommunizieren miteinander, dazu ist die *Kenntnis*-Beziehung notwendig. Für die Modellierung der Kommunikation werden typischerweise verschiedene Annahmen getroffen: z.B. asynchroner oder synchroner Nachrichtenaustausch, gerichtete oder ungerichtete Nachrichten an einzelne oder mehrere Empfänger, oder Sicherheit des Kommunikationsmediums. Akteure können selbst wieder aus kommunizierenden Akteuren *zusammengesetzt* (und damit Systeme) sein.

Daten/Zustand Daten sind in den Akteuren *gekapselt*. Ihre Werte sind zeitlich variabel. Für andere Akteure sind die Daten nur über Kommunikation zugreifbar. Bei der Modellierung von Daten ist zu unterscheiden:

Datentypmodellierung beschreibt die Menge der möglichen Datenwerte, z.B. durch Abstrakte Datentypen.

statische Datenmodellierung beschreibt die Menge der möglichen Datenzustände eines Akteurs, typischerweise durch Entity-Relationship-Diagramme.

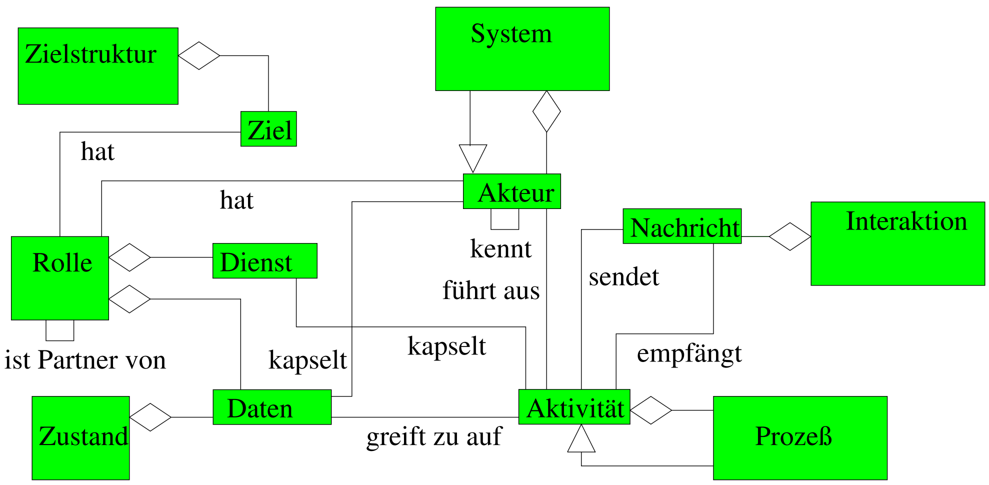

Abbildung 1: Das Systemkonzeptmodell

Datenverhaltensmodellierung beschreibt die Veränderung des Datenzustands über die Zeit, z.B. durch Zustandsübergangsdiagramme.

Aktivität/Prozeß Aktivitäten strukturieren das interne Verhalten eines Akteurs. Die Struktur wird bestimmt durch den Daten- und Steuerfluß zwischen den Aktivitäten und durch den Aufgabenkontext. Aktivitäten *greifen auf* Daten zu und *senden* und *empfangen* Nachrichten. Aktivitäten können selbst wieder aus Aktivitäten *zusammengesetzt* (und damit Prozesse) sein. Wieder können zwei Arten der Aktivitätsmodellierung unterschieden werden:

statische Aktivitätsmodellierung beschreibt die Menge der möglichen Datenflüsse zwischen den Aktivitäten, z.B. die Datenflußdiagramme der strukturierten Analyse.

Aktivitätsverhaltensmodellierung beschreibt die Abfolge der Aktivitäten, entweder vollständig oder als einzelne Prozesse. Dabei kommt zum Datenfluß Steuerfluß hinzu. Zur Darstellung komplexen Steuerflusses können Ereignisse hinzugenommen werden. Steht die Synchronisation der Aktivitäten auf dem gemeinsamen Datenraum im Vordergrund ist, die Hinzunahme von Zustandselementen wie in Petrinetzen sinnvoll.

Typischerweise werden Aktivitätenfolgen auch akteursübergreifend modelliert (z.B. Geschäftsprozeßdiagramme). Die Zuordnung der Aktivitäten zu Akteuren bzw. Rollen im Modell macht semantisch einen großen Unterschied: Ohne Rollen arbeiten die Aktivitäten auf einem gemeinsamen Datenraum, durch die Zuordnung zu Rollen haben die Aktivitäten nur Zugriff auf die Daten der eigenen Rolle.

Dienst Dienste strukturieren das externe Verhalten eines Akteurs. Sie *kapseln* eine von außen anstoßbare Folge von Aktivitäten. Bzgl. der Dienstausführung sind wieder für die Modellierung einige Annahmen zu treffen: z.B. werden beim Aufruf Daten übergeben, wartet der Aufrufer auf das Ergebnis, können in einem Akteur mehrere Dienste gleichzeitig aktiv sein. Wir unterscheiden wieder statische und dynamische Dienstmodellierung:

statische Dienstmodellierung beschreibt die Menge aller möglichen Dienstinstanzen eines Akteurs und die möglichen gegenseitigen Aufrufbeziehungen und Verschachtelungen. Diese Zusammenhänge werden typischerweise bei der prozeduralen Programmierung modelliert.

Dienstverhaltensmodellierung beschreibt die Ausführung eines Dienstes. Bei mehreren Threads ist insbesondere die Synchronisation der Dienste auf dem gemeinsamen Datenraum des Akteurs interessant. Für die Dienstverhaltensmodellierung können die gleichen Modellierungstechniken wie für Aktivitätsverhaltensmodellierung eingesetzt werden.

Ziel/Zielstruktur Ziele beschreiben selbstgesetzte Rahmenbedingungen *einer Rolle* bei der Aufgabenerfüllung. Ein Beispiel für eine statische Zielmodellierung wird in [Yu97] beschrieben. Dort werden die Abhängigkeiten zwischen Zielzuständen, Ressourcen, Kriterien und Aufgaben (auch verschiedener Akteure) modelliert. Spezielle Wege durch diese Zielstruktur entsprechen möglichen Wegen der Zielerreichung.

Nachricht/Interaktion Nachrichten sind die Grundlage der Kommunikation zwischen den Akteuren. Sie werden nur im Zusammenhang von Interaktion zwischen einzelnen Akteuren bzw. Rollen modelliert. Wie beim Datenfluß kann man die Menge der möglichen Nachrichtenflüsse (die Kanäle) oder spezielle Abfolgen (z.B. Sequenz- und Kollaborationsdiagramme in UML [BRJ97]) modellieren.

Rolle Rollen charakterisieren einen zusammenhängenden Aufgabenbereich *eines Akteurs* durch Angabe von Daten, der auf ihnen operierenden Dienste und der *Kommunikationspartner*. Klassen in der objektorientierten Modellierung erlauben nur eine eingeschränkte Rollenmodellierung, in der jeder Akteur (d.h. jedes Objekt) nur eine Rolle besitzt. Bei der

statischen Rollenmodellierung werden die einzelnen Rollen mit ihren Daten, Diensten und Partnern beschrieben. Ein typisches Beispiel sind die Objektmodelle der objektorientierten Methoden.

Rollenverhaltensmodellierung beschreibt das Zusammenspiel der Dienste einer Rolle. Im sequentiellen Fall (wie bei den typischen objektorientierten Programmiersprachen) wird dazu ein Zustandsübergangsdiagramm verwendet, dessen Transitionen mit Dienstaufrufen beschriftet sind. Sind die Dienstinstanzen innerhalb eines Akteurs nebenläufig, so können sie nicht mehr als atomar betrachtet werden. [PR97] zeigt einen Ansatz, wie man in diesem Fall das Rollenverhalten aus den einzelnen Dienstbeschreibungen ableiten kann.

Durch die obige Diskussion wird deutlich, daß Diagramme mit der gleichen Syntax zu sehr verschiedenen Zwecken eingesetzt werden können (z.B. Zustandsübergangsdiagramme für Rollenverhalten, Dienstverhalten und Datenverhalten). Umgekehrt können kleine notationalle Erweiterungen starke Auswirkungen auf die Semantik haben (z.B. Aktivitätsfolgen mit und ohne Rollen). Der Wunsch nach mglst. wenig verschiedenen Modellierungstechniken steht also im Widerstreit mit der visuellen Unterscheidungen verschiedener Modellierungskonzepte.

3 Wie werden die Modelle im Prozeß des System- und Softwareentwicklung verwendet?

Modelle werden in der System- und Softwareentwicklung innerhalb einer Methode eingesetzt. Eine Methode dient zur Erlangung von Erkenntnissen und Ergebnissen. Die Frage ist also, welche Erkenntnisse sich durch die Modelle gewinnen lassen und welchen Stellenwert sie im Hinblick auf die Produkt der Methode haben.

Erkenntnisse Modelle dienen zur Wissens- und Konsensbildung zwischen den unterschiedlichen Beteiligten, zur Untersuchung von Eigenschaften des modellierten Systems, und als Vorgabe für ein zu realisierendes System. Je nach Einsatzzweck ergeben sich unterschiedliche Anforderungen an die Anschaulichkeit und Genauigkeit der Modelle. Die Wichtigkeit der Erkenntnisse im Gesamtprozeß gibt einen Rahmen für den Modellierungsaufwand vor.

Produkte Ein System wird typischerweise durch drei Produkte beschrieben: die externe Aufgabenspezifikation, die interne Aktivitätenspezifikation und der Entwurf. Bei den typischen Softwareentwicklungsmethoden wird die externe Aufgabenspezifikation meist nur für das Softwaresystem angegeben, die interne Aktivitätenspezifikation für das Anwendungssystem, und für das Nutzungs- und Softwaresystem nur der Entwurf. Diese Produkte lassen sich aber jeweils für alle drei Systeme aufstellen. Aus den Erfahrungen mit den strukturierten und objektorientierten Methoden wurde deutlich, daß typischerweise eine Modellierungstechnik nicht für alle drei Produkte geeignet ist. Stattdessen werden heute oft für die interne Aktivitätenspezifikation daten- und prozeßorientierte Modellierungstechniken eingesetzt, während ein Entwurf sich gut durch rollen- und interaktionsorientierte Modellierungstechniken beschreiben läßt. Für die externe Aufgabenspezifikationen sind Ziel- und Dienstbeschreibungen wichtig. Durch die Beschreibung des gleichen Systems mit verschiedenen Modellen entsteht auch die Frage nach der Beziehung zwischen den Modellen.

Neben der Ausdrucksmächtigkeit sind also die mit einer Modellierungstechnik anwendbaren methodischen Schritte und der Zusammenhang von Modellen innerhalb einer Methode zu bewerten.

4 Formalität

Wir haben bisher vor allem semi-formale Beschreibungstechniken diskutiert. Je größer die Wichtigkeit von qualitativen und quantitativen Analysen der Modelle und eine Überprüfung der Beziehungen zwischen verschiedenen Modellen ist, desto wichtiger ist eine formale Semantik. Dabei ist aber zwischen einem direkten Gebrauch des Formalismus, bei dem die Entwickler eine formale Spezifikationssprache verwenden, d.h. die Details der formalen Semantik verstehen müssen, und einem indirekten Gebrauch zu unterscheiden, bei dem der Formalismus nur als semantische Grundlage für ein Regelwerk bzw. Werkzeug dient, das die Entwickler verwenden können ohne die semantischen Details zu kennen [Huß97, Pae95]. Im letzteren Fall müssen nur die Methoden- bzw. Werkzeugentwickler die Details der Semantik kennen. [BHH+97] gibt ein Beispiel für solch eine Semantik. Da eine formale Semantik für komplexe System meist sehr aufwendig ist, scheint ein direkter Gebrauch im gesamten Entwicklungsprozeß nicht möglich. Interessant ist der indirekte Gebrauch, der eine stellenweise Hinzunahme formaler Spezifikationssprachen wie Prädikatenlogik erlaubt, ohne sie für die ganze Systementwicklung zu fordern. Erste Ansätze für eine solche Werkzeugunterstützung bietet z.B. Autofocus [HSE97].

Literatur

[BEF+96] M. Broy, J. Eberspächer, G. Färber, G. Reinhart und H. Wildemann. Bayerischer Forschungsverbund Software - Engineering, Antrag auf Einrichtung an die Bayerische Forschungsstiftung. http://www.forsoft.de, 1996.

[BHH+97] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe und V. Thurner. Towards a Formalization of the Unified Modeling Language. In *ECOOP, LNCS 1241*, Seiten 344–366, 1997.

[BRJ97] G. Booch, J. Rumbaugh und I. Jacobson. The Unified Modeling Language for Object-Oriented Development, Version 1.1, 1997.

[DHP+98] B. Deifel, U. Hinkel, B. Paech, P. Scholz und V. Thurner. Die Praxis der Softwareentwicklung: Eine Erhebung. submitted to publication, 1998.

[HSE97] F. Huber, B. Schätz und G. Einert. Consistent Graphical Specification of Distributed Systems. In J. Fitzgerald, C. B. Jones und P. Lucas, Hrsg., *FME '97, LNCS 1313*, Seiten 122 – 141. Springer, 1997.

[Huß97] H. Hußmann. *Formal Foundations for Software Engineering Methods, LNCS 1322*. Springer, 1997.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering*. Addison-Wesley, 1992.

[Pae95] B. Paech. A Methodology Integrating Formal and Informal Software Development. In M.Wirsing, Hrsg., *ICSE-17 Workshop on Formal Methods Application in Software Engineering Practice*, Seiten 61–68, 1995.

[Pae96] B. Paech. Algebraic View Specification. In M.Wirsing, Hrsg., *AMAST'96, LNCS 1101*, Seiten 444–457, 1996.

[PR97] B. Paech und B. Rumpe. State Based Service Description. In H. Bowman und J. Derrick, Hrsg., *FMOODS, Volume 2*, Seiten 293–304. Chapman and Hall, 1997.

[SN93] D. Schuler und A. Namioka. *Participatory Design - Principles and Practices*. Lawrence Erlbaum Associates, 1993.

[Yu97] E.S.K. Yu. Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. In *Symposium on Requirements Engineering*, Seiten 226–235. IEEE, 1997.