

Ein Metamodell zur architekturorientierten Beschreibung komplexer Systeme

Peter Tabeling

HASSO-PLATTNER-INSTITUT für Softwaresystemtechnik
Prof. Dr. Helmert - Straße 2-3
14482 Potsdam
Peter.Tabling@hpi.uni-potsdam.de

Zusammenfassung: Dieser Beitrag stellt einen Ansatz zur Beschreibung von Softwaresystemen vor, der insbesondere zur Erfassung der Architektur komplexer Systeme dient. Er zielt daher nicht auf die Darstellung von Programmstrukturen ab, sondern auf die Beschreibung von Systemmodellen, die der Mensch zu unterschiedlichsten Zeitpunkten der Softwareentwicklung erstellt. Er eignet sich zur Beschreibung sowohl übergeordneter als auch realisierungsnaher Modelle und hat sich in einer Reihe industrieller und universitärer Projekte bewährt. Der Ansatz ist nicht an einem bestimmten Programmierparadigma ausgerichtet, sondern an den Bedürfnissen des Menschen nach Anschaulichkeit. Trotzdem können technische Merkmale komplexer Systeme wie Verteilung, Nebenläufigkeit, Transaktionen oder ein dynamisch veränderlicher Aufbau aus Komponenten gut erfasst werden. Der Schwerpunkt dieses Beitrags liegt nicht auf der verwendeten Notation, sondern der Diskussion des zugrunde liegenden begrifflichen Metamodells.

1 Einführung

1.1 Hintergrund

Die Entwicklung komplexer Softwaresysteme kann wegen des großen Umfangs der zu erstellenden Software nur arbeitsteilig erfolgen. Diese arbeitsteilige Entwicklung erfordert einen intensiven Gedankenaustausch der beteiligten Personen über das zu erstellende System. Oft sind dabei grundsätzliche Entscheidungen bzgl. der zu wählenden Systemarchitektur zu diskutieren, wobei z.B. die Festlegung auf die später zu verwendenden Programmiersprachen oder Softwarekomponenten noch gar nicht erfolgt ist. Desweiteren kommt es vor, dass Kommunikationspartner gar nicht direkt an der Softwareerstellung beteiligt sind und daher nur konstruktive Merkmale des Systems verstehen müssen - man denke z.B. an die Kommunikation zwischen Entwicklern und höheren Vorgesetzten oder zwischen Systemhersteller und Auftraggeber. Im skizzierten Kontext geht es also primär darum, Wissen über gedachte Systemstrukturen zu vermitteln, und nicht um die Software, die zur Realisierung dieser gedachten Strukturen erstellt wird.

1.2 Zielsetzung

Hauptgedanke des vorzustellenden Beschreibungsansatzes war somit die Schaffung einer begrifflichen Basis sowie zugehöriger Notationen, die optimal zur zwischenmenschlichen Kommunikation über gedachte Systemstrukturen geeignet sind. Die Wahl der Beschreibungsmittel musste sich daher an den Bedürfnissen des Menschen und nicht an bestimmten Programmierparadigmen orientieren. Damit unterscheidet sich der Ansatz z.B. von der Unified Modeling Language [BRJ99], die “sowohl für den Mensch als auch die Maschine nutzbar” sein soll¹ und einen engen Bezug zu objektorientierten Programmiersprachen aufweist.²

Einerseits galt es also, ein Metamodell zu entwickeln, das auf möglichst wenigen Grundbegriffen basiert und eine gewisse Anschaulichkeit der Modelle erlaubt. Da der gesuchte Ansatz vor allem zur Beschreibung großer Systeme geeignet sein sollte, musste das Metamodell andererseits ausreichende Mächtigkeit aufweisen, um Vorteile bei der Beschreibung komplexer Architekturen zu bieten. Es galt also, typische Merkmale wie Verteilung, Nebenläufigkeit, Transaktionen oder einen dynamisch veränderlichen Aufbau des Systems zu erfassen. Der objektorientierte Ansatz ist zu diesem Zweck nur eingeschränkt geeignet, da der Objektbegriff zu allgemein ist und sich Systemarchitekturen schlecht durch Objektstrukturen beschreiben lassen [KI99].

2 Der Ansatz

Der vorgestellte Ansatz beruht auf Arbeiten von Wendt [We82a][We82b] und wurde in verschiedenen Arbeiten [Zu90][Bu98][Ta00a] weiterentwickelt. Bei komplexen Systemen sind viele Systemmodelle zu unterscheiden, die einerseits in sich strukturiert sind und andererseits untereinander in Beziehung stehen. Aus dieser Sichtweise resultieren bestimmte Strukturtypen, die bei komplexen Systemen stets zu erfassen sind. Dies sind zum einen die Strukturen *innerhalb* eines Modells, nämlich Aufbau-, Ablauf- und Wertestrukturen. Zum anderen sind es die modellübergreifenden Strukturen, nämlich Elemente unterschiedlicher Modelle und deren Implementierungsbeziehungen. Da eine Vorstellung der Darstellungskonzepte den Rahmen dieses Beitrags sperren würde, werden im Folgenden primär die grundlegenden Begriffe und deren Beziehungen erläutert.

2.1 Strukturen innerhalb eines Modells

Zunächst werden die Strukturen *eines* Modells betrachtet, d.h. diejenigen Systemstrukturen die der Mensch bei Beschränkung auf *eine* bestimmte Abstraktionsebene vor dem geistigen Auge „sieht“ - siehe auch Abbildung 1.³

¹ siehe [Ob01]: “[UML was meant to be] a modeling language usable by both humans and machines”

² siehe [Ob01]: “UML does have a tight mapping to a family of OO languages.”

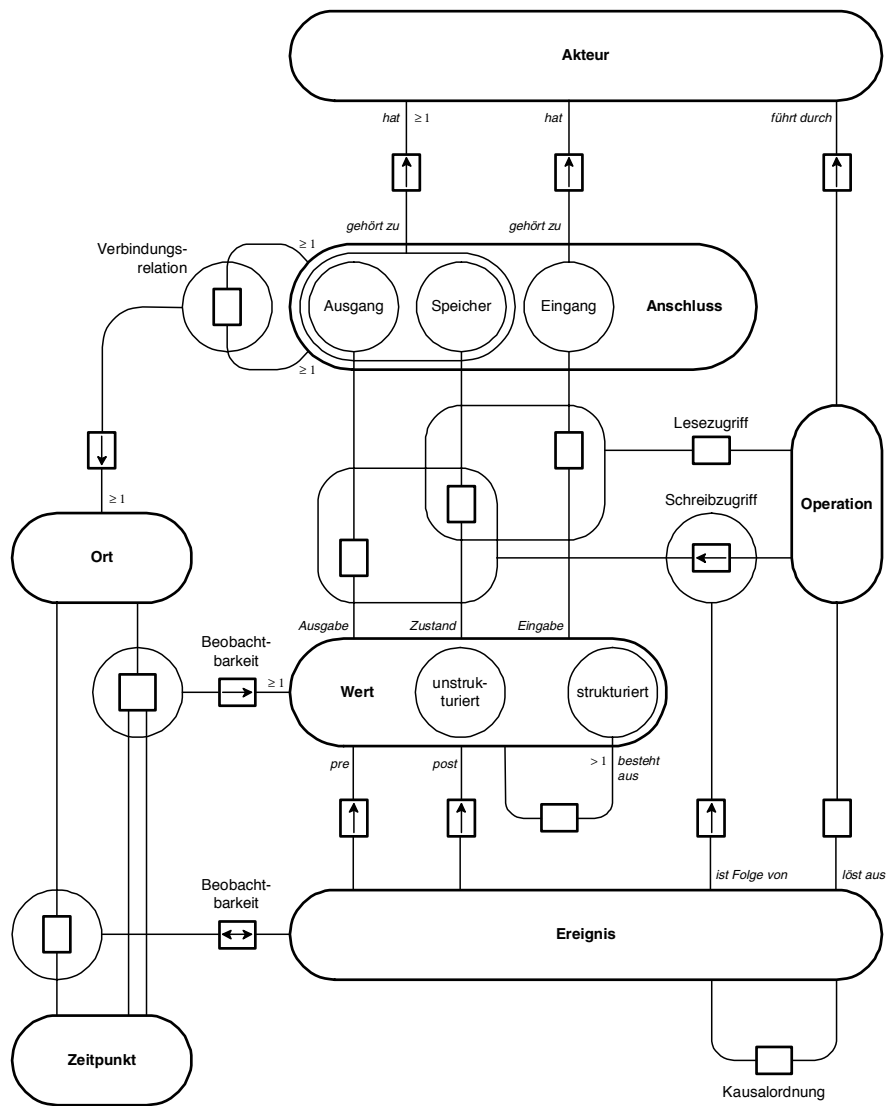


Abbildung 1: Grundlegende Entitäten und Relationen eines Modells

3 Es handelt sich um ein E/R-Diagramm, bei dem abgerundete Knoten für Entitäten und rechteckige für Relationen stehen. Einfache Pfeile weisen auf Funktionen hin, wobei der Pfeil vom Definitions- zum Wertebereich weist. Doppelpfeile identifizieren 1:1-Abbildungen. Rechtecke in Rundknoten stellen objektifizierte Relationen dar, d.h. Relationen, deren Elemente an einer weiteren Relation teilnehmen (siehe z.B. Schreibzugriff). Partitionen werden durch Rundknoten innerhalb eines Rundknotens dargestellt (siehe z.B. Partitionierung der Werte in unstrukturierte und strukturierte Werte).

Aufbaustruktur. Kerngedanke des Ansatzes ist die Vorstellung, dass jedes System als Gebilde aus agierenden Komponenten - sprich *Akteuren* - veranschaulicht werden kann, wobei sich das Systemverhalten aus dem Zusammenwirken der Akteure ergibt [We89]. Bei einem aus mehreren Akteuren bestehenden System wird hier die Vorstellung vorausgesetzt, dass die Akteure unabhängig von der aktuellen Einbindung in den Aufbau des Systems existieren und „bausteinartig“ kombiniert werden können. Dies setzt voraus, dass Verbindungen zwischen Akteuren erzeugt und aufgetrennt werden können, ohne dass dies einen Eingriff in die Akteure selbst erfordert. Daher müssen Akteure über abstrakte *Anschlüsse* verfügen, über die Verbindungen zwischen Akteuren hergestellt werden. Je nachdem, ob ein Akteur auf einen Anschluss ausschließlich lesend oder schreibend zugreift, handelt es sich um einen *Ein-* bzw. *Ausgang*. Ein zustandsbehafteter Akteur muss über wenigstens einen *Speicher* zur „Aufbewahrung“ seines Zustandes verfügen. Da andere Akteure möglicherweise mit diesem Speicher verbunden werden können, werden Speicher hier zu den Anschlüssen gezählt.

Ein Systemaufbau entsteht erst, wenn auf Basis der Anschlüsse *Verbindungen*¹ von Akteuren definiert werden. Eine Menge untereinander verbundener Anschlüsse stellt eine *Schnittstelle* zwischen Akteuren dar.² Schnittstellen und Speicher stellen (fiktive) *Orte*³ im Aufbau dar, auf dem Werte im System beobachtbar sind. Ein Aufbau aus Akteuren und Orten ist *nicht* notwendigerweise als Hinweis auf den *physikalischen* Aufbau des Systems zu deuten, sondern stellt - insbesondere auf höheren Betrachtungsebenen - eine rein gedankliche Aufteilung des Systems dar. Abbildung 2 zeigt ein Beispiel, bei dem drei

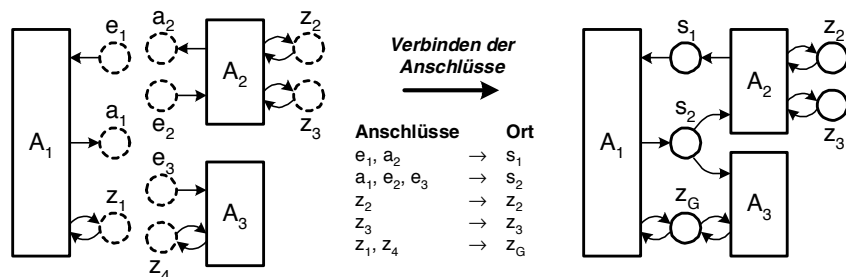


Abbildung 2: Anschlussbasierter Aufbau aus Akteuren

Akteure A_1 , A_2 und A_3 (links im Bild) über ihre Anschlüsse verbunden werden, sodass eine Aufbaustruktur mit fünf Orten entsteht (rechts im Bild). Wie das Beispiel zeigt, können nicht nur einfache Speicher (z_2, z_3) und Schnittstellen zur Verbindung zweier Akteure (s_1) erfasst werden, sondern auch weitergehende Verbindungen wie gemeinsam genutzte Speicher (z_G) und Broadcast-Schnittstellen (s_2). Da der vorgestellte Ansatz weder Beschränkungen bzgl. der Anschlusszahl eines Akteurs noch bzgl. der Anzahl untereinander verbindbarer Anschlüsse vorsieht, können auch komplexe Aufbaustrukturen problemlos beschrieben werden.

1 Verbindungen können als Äquivalenzrelation auf der Menge der Anschlüsse dargestellt werden.
 2 Es ist mind. ein Eingang und mind. ein Ausgang oder Speicher unter den verbundenen Anschlüssen.
 3 Ein Ort entspricht dabei einer Äquivalenzklasse der Verbindungsrelation.

Wertestrukturen. Zu den in einem Zeitpunkt gegebenen Strukturen zählen neben dem Aufbau des Systems die Werte, die auf den Orten des Systems beobachtbar sind. Dabei kann es sich um *unstrukturierte Werte* wie Integer-Zahlen oder auch um beliebig *strukturierte Werte* wie z.B. eine ganze Datenbank handeln. Diese sich aus dem jeweiligen Anwendungsgebiet ergebenden Strukturen werden beim vorgestellten Ansatz durch weiterentwickelte E/R-Diagramme dargestellt, die den Wertebereich der entsprechenden Speicher bzw. Schnittstellen sowie deren Typbeziehungen beschreiben. (Die Darstellung des Metamodells in Abbildung 1 stellt ein Beispiel dieses Diagrammtyps dar.)

Ablaufstrukturen. Die Dynamik eines Systems wird erst „sichtbar“, wenn man die Betrachtung über die Zeit ausdehnt. Dann sind einerseits *Ereignisse* gegeben, d.h. Wertewechsel (pre→post), die an einem bestimmten Ort und zu einem bestimmten *Zeitpunkt* beobachtbar sind.¹ Andererseits sind auch Intervalle zwischen Zeitpunkten gegeben, in denen unveränderte Werte auf einem Ort beobachtbar sind.²

Während Ereignisse und Werte elementar sind bzgl. der *Beobachtbarkeit* durch Akteure, sind *Operationen* elementar bzgl. der *Aktivität* von Akteuren, d.h. alle Aktivitäten der Akteure sind auf Operationen rückführbar. Eine Operation ist definiert als eine Elementaraktivität, bei der ein Akteur einen Wert (das Operationsergebnis) auf einem Ort erzeugt, wobei dieser erzeugte Wert von Werten abhängt, welche der Akteur von einem oder mehreren Orten liest [Ta00a]. Dabei findet auf jedem gelesenen Ort ein *Lesezugriff* und auf dem geschriebenen Ort ein *Schreibzugriff* statt. Betreffen ein Lese- und ein Schreibzugriff den gleichen Ort, dann bilden diese einen *modifizierenden Zugriff*. Operationen können von Ereignissen ausgelöst werden und können Ereignisse zur Folge haben, wodurch kausale Abhängigkeiten von Ereignissen entstehen.³

An dieser Stelle mag es als zweckmäßige Abstraktion erscheinen, „Operation“, „Ereignis“ und „Zustandsübergang“ als Synonyme zu betrachten. Anhand einiger Beispiele lässt sich zeigen, dass dies eine unzulässige Vereinfachung darstellen würde. Betrachtet man das Beispiel a) in Abbildung 3, so sieht man, dass im Falle eines reinen Zustandsübergangs der gelesene und der beschriebene Ort identisch sind: es ist der Speicher, auf dem das Tupel (a,b,c) als strukturierter Wert liegt. Bei der Operation findet somit ein modifizierender Zugriff (M) statt, der aus einem Lesezugriff (gestricheltes Oval L) und ein Schreibzugriff (schraffierter Übergang S) besteht. Man sieht leicht, dass Ereignisse zwar die Folge von Operationen (genauer: des jeweiligen Schreibzugriffs) sind⁴, aber es kann durchaus passieren, dass eine Operation kein Ereignis zur Folge hat, da das Schreiben eines Wertes nicht unbedingt eine Wertänderung bedeuten muss (im betrachteten Beispiel wäre dann $a=a'$).

Während bei a) die Operation einem (modifizierenden) Zugriff entspricht, ist bei b) zusätzlich ein Lesezugriff gegeben, bei dem c als Eingabe gelesen wird. Beim Beispiel c) sind

1 „Beobachtbarkeit von Ereignissen“ ist darstellbar als 1:1-Abbildung: (Orte \times Zeitpunkte) \leftrightarrow Ereignisse

2 „Beobachtbarkeit von Werten“ ist darstellbar als Funktion: (Orte \times Zeitpunkte²) \rightarrow Werte

3 Kausale Abhängigkeit von einem nicht operationsauslösenden Ereignis e kann dadurch entstehen, dass der post-Wert von e bei einer Operation gelesen wird.

4 im betrachteten Beispiel hat der Zugriff *zwei* Ereignisse zur Folge, nämlich die Wertewechsel (a,b,c)→„NICHTS“ und „NICHTS“→(a',b,c).

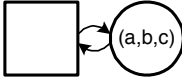
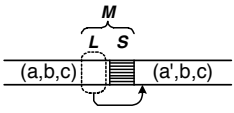
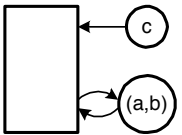
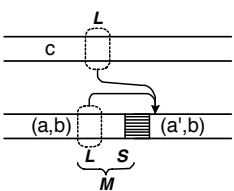
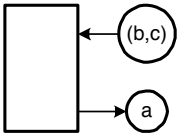
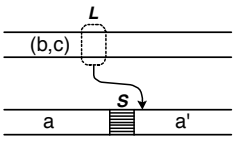
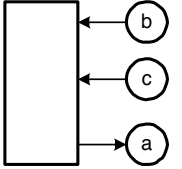
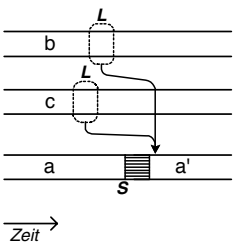
Systemaufbau (Ausschnitt)	Operation $a := f(b,c)$	Operationstyp
a) 		Zustandsübergang
b) 		Zustandsübergang mit Eingabeverarbeitung
c) 		Zuordnerschritt mit <i>temporal</i> konsistenter Eingabe
d) 		Zuordnerschritt mit <i>kausal</i> konsistenter Eingabe

Abbildung 3: Beispiele von Operationen

zwar wie in a) ein Lese- und ein Schreibzugriff gegeben - diese bilden aber keinen modifizierenden Zugriff, da unterschiedliche Orte betroffen sind. Während das Paar (b,c) im Beispiel c) in einem Lesezugriff gelesen wurde, sind dazu in Beispiel d) wegen der Aufteilung auf zwei Eingänge zwei Lesezugriffe nötig.

Konsistenzannahmen im Hinblick auf verteilte Systeme. Der vorgestellte Operationsbegriff basiert *nicht* auf der Annahme eines beobachtbaren globalen Systemzustandes, sondern der Vorstellung, dass die im Rahmen einer Operation gelesenen Werte lediglich aus Sicht des operationsdurchführenden Akteurs "gleichzeitig gegeben" sind. Somit muss man bei jedem Lesezugriff eine implementierungsbedingte individuelle Verzögerung annehmen, die sich in Abbildung 3 durch einen entsprechenden Versatz der Lesezugriffe untereinander äußert. Die Möglichkeit, einen *temporal konsistenten* Wert (also einen Wert, der tatsächlich in einem Zeitpunkt vorgelegen hat) zu beobachten, ist somit auf einzelne Orte beschränkt. Dagegen weisen die im Rahmen *einer* Operation von *verschiedenen* Orten gelesenen Werte i.A. nur *kausale Konsistenz* auf - sie entsprechen also einem

mente aus M_1 auf implementierende Modellelemente aus M_2 abgebildet werden. Derartige Abbildungen sind in allen drei Strukturbereichen - Aufbau-, Ablauf- und Wertestruktur - möglich. Im Folgenden werden Grundüberlegungen zu diesen Abbildungen vorgestellt - eine ausführliche Diskussion findet man in [Ta00a].

Implementierungsbeziehungen bei Wertestrukturen. Der einfachste Fall der Implementierungsbeziehung ist gegeben, wenn ein Wertebereich eines höheren Modells durch einen Wertebereich auf tieferer Ebene implementiert wird - man denke z.B. an die („Unicode“-) Kodierung von Buchstaben durch 16-stellige Binärtupel. Aufbau- bzw. Ablaufstrukturen sind von solchen Wertebereichsabbildungen nicht betroffen.¹

Implementierungsbeziehungen bei Ablaufstrukturen. Oft sind höhere Operationen auf Basis eines Repertoires einfacherer Operationen zu implementieren - siehe z.B. die Umschreibung der Operation „ $a := a/b + b$ “ durch zwei Operationen „ $a := a/b$ “ und „ $a := a + b$ “, in Abbildung 5. An diesem Beispiel wird ersichtlich, dass die Implementierung einer Ope-

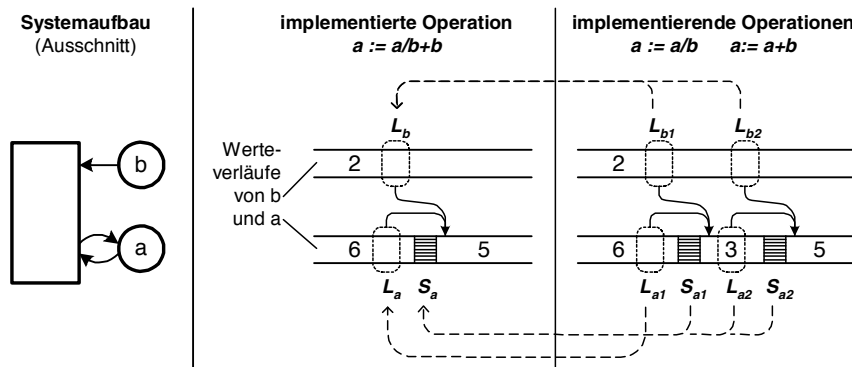


Abbildung 5: Beispiel zur Operationsimplementierung (1)

ration letztlich die Implementierung der entsprechenden Zugriffe erfordert: Der Lesezugriff L_b wird durch die Lesezugriffe $\{L_{b1}, L_{b2}\}$ implementiert, der modifizierende Zugriff $M_a = \{L_a, S_a\}$ wird durch die Zugriffe $\{L_{a1}, S_{a1}, L_{a2}, S_{a2}\}$ implementiert. (L_{a2} fällt dabei auf höherer Ebene in den Schreibzugriff S_a .)

Operationsimplementierungen sind jedoch nicht auf den Fall beschränkt, dass eine Operation durch eine Sequenz von Operationen implementiert wird. Eine Operation kann auch durch eine Menge nebenläufiger Operationen implementiert werden. Dies ist dann der Fall, wenn z.B. ein an einem Ort O befindlicher strukturierter Wert (a, b) auf tieferer Ebene durch zwei unstrukturierte Werte a und b implementiert wird, die an verschiedenen Orten O_a und O_b liegen, sprich ortsverteilt sind - siehe Abbildung 6. In diesem Fall muss z.B. die Operation „ $(a, b) := (2a, 3b)$ “ durch zwei nebenläufige Operationen „ $a := 2a$ “ und „ $b := 3b$ “ realisiert werden.²

1 wenn man davon absieht, dass die Funktionen, gemäß derer Operationsergebnisse von gelesenen Werten abhängen, ebenfalls abzubilden sind.

2 a und b seien ganze Zahlen.

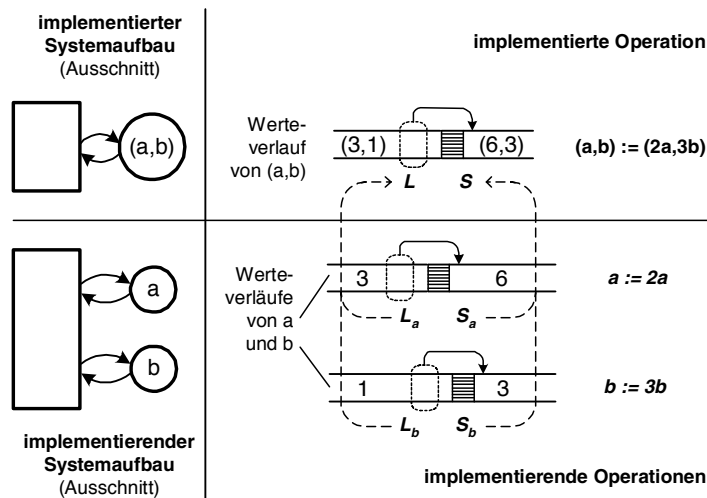


Abbildung 6: Beispiel zur Operationsimplementierung (2)

Auch in diesem Beispiel lassen sich die entsprechenden Zugriffe abbilden: Der modifizierende Zugriff $M=\{L,S\}$ wird durch die Zugriffe $\{L_a,L_b,S_a,S_b\}$ implementiert.

Mit der Abbildung von Schreibzugriffen erfolgt gleichzeitig auch die Abbildung von Ereignissen. So wird z.B. das Ereignis "Verschwinden des Wertes (3,1)" (markiert den Beginn des Schreibzugriffes S in Abbildung 6) auf zwei nebenläufige Ereignisse (Beginn der Schreibzugriffe S_a bzw. S_b) abgebildet.

Implementierungsbeziehungen bei Aufbaustrukturen. Ein Ort kann durch mehrere Orte implementiert werden - man denke z.B. an die Replikation, bei der ein Speicher durch mehrere Speicher realisiert wird. Ein Akteur kann durch mehrere Akteure und evtl. mehrere Orte implementiert werden. Ein einfaches Beispiel dafür stellt die Realisierung eines Akteurs durch eine Aufbaustruktur aus implementierenden Akteuren und Speichern dar. Interessante Abbildungen sind z.B. gegeben, wenn die Implementierungsbeziehungen zeitvariant sind. Ein Beispiel dafür wären Akteure, die zwecks Lastverteilung zeitabhängig auf verschiedenen Rechnern realisiert werden.

Werden Akteure bzw. Orte bei Implementierungsschritten aufgeteilt, so erfordert dies i.d.R. auch die Implementierung von Operationen, da einzelne Operationen dann durch Operationen unterschiedlicher Akteure bzw. Zugriffe durch mehrere Zugriffe auf unterschiedliche Orte realisiert werden müssen.

Modellierung von Transaktionen. Der vorgestellte Ansatz erlaubt eine neue und "natürlichere" Deutung des Transaktionsbegriffs [GR93]. Danach stellt eine Transaktion eine fehlertolerante Implementierung eines modifizierenden, lesenden oder schreibenden Zugriffs eines Akteurs auf einen Ort dar [Ta00a]: Wird bei einem Implementierungsschritt ein Zugriff auf mehrere Zugriffe aufgeteilt, so stellen diese Zugriffe erst dann eine fehlertolerante Zugriffsimplementierung dar, wenn sie die ACID-Eigenschaften aufweisen. Ato-

micity, Consistency und Isolation leiten sich dabei aus der Zugriffsaufteilung ab, während sich Durability erst aus der Forderung nach Fehlertoleranz ergibt.

Transaktionen können also als Ergebnisse von Implementierungsschritten in Ablauf- oder Aufbaustrukturen verstanden werden, sie ergeben sich dabei *implizit* aus der Abbildung von Operationen und/oder Orten. Dabei lassen sich auch die unterschiedlichen Transaktionstypen in einfacher Weise einordnen: Ist eine Zugriffsaufteilung in einer *Operationsaufteilung* begründet, so erhält man eine *flache* Transaktion. (Die Zugriffsmengen $\{L_{a1}, L_{a2}, S_{a1}, S_{a2}\}$ und $\{L_{b1}, L_{b2}\}$ in Abbildung 5 stellen entsprechende Beispiele dar.) Eine Zugriffsaufteilung wegen *Ortsaufteilung* führt dagegen zu einer *verteilten* Transaktion (siehe z.B. die Zugriffsmenge $\{L_a, L_b, S_a, S_b\}$ in Abbildung 6). Eine *mehrstufige* Zugriffsaufteilung führt zu einer *geschachtelten* Transaktion.

3 Abschließende Bemerkungen

Praktische Erfahrungen. Nach dem vorgestellten Ansatz wurden Projekte unterschiedlicher Größe und Art modelliert. Es handelte sich dabei nicht nur um universitäre Projekte (siehe z.B. [Ka95]), sondern auch um nichtuniversitäre Projekte wie z.B. die Dokumentation des Systems R/3 der Firma SAP AG oder die Modellierung des Apache Webservers [Gr01]. Dabei haben speziell die aufbauorientierte Sichtweise sowie das durchgängig verwendete Metamodell sehr zur Anschaulichkeit der Darstellungen beigetragen.

Softwarestruktur vs. Systemarchitektur. Ziel des Ansatzes ist die Erfassung der Systemarchitektur und nicht der Softwarestruktur. Daher ist *deren* Festlegung - also z.B. die Festlegung der Modularisierung - als (teilweise) losgelöste Aufgabe zu betrachten, bei der jedoch ein gutes Verständnis der Systemarchitektur eine wertvolle Ausgangsbasis bildet. In [K199] sind z.B. Abbildungsmöglichkeiten zwischen Systemarchitektur und objektorientierten Modellen beschrieben, wonach ein Objekt z.B. einem Ort (Speicher oder Schnittstelle), einem Akteur oder einem Wert entsprechen kann.

Ausblick. Das vorgestellte Metamodell kann als Grundlage zur Gestaltung einer „architekturorientierten“ Programmiersprache dienen. Eine derartige Sprache würde dem Entwickler angemessenere Sprachmittel zur Programmierung komplexer Systeme bieten. Darüberhinaus könnte das zugehörige Laufzeitsystem in transparenter Weise Transaktionen durchführen sowie bestimmte Speicherkonsistenzmodelle unterstützen [Ta00a].

Literaturverzeichnis

- [BRJ99] Booch, Rumbaugh, Jacobson: The Unified Modeling Language User Guide. Addison-Wesley 1999
- [Bu98] Andreas Bungert: Beschreibung programmierter Systeme mittels Hierarchien intuitiv verständlicher Modelle. Shaker Verlag Aachen 1998

- [CL85] Chandy, Lamport: Distributed Snapshots - Determining Global States of Distributed Systems. ACM Transactions on Computer Systems, Vol. 3, No. 1, February 1985
- [Gr01] Gröne, Knöpfel, Kugel, Schmidt: The Apache Modelling Project. Hasso-Plattner-Institut für Softwaresystemtechnik, Potsdam, www.hpi.uni-potsdam.de/apache, 2001
- [GR93] Gray, Reuter: Principles of Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993
- [Ka95] Mark Kappel: Entwurf und Implementierung eines Simulators für Register-Transfer-Netze. Diplomarbeit, Lehrstuhl für Dig. Systeme der Universität Kaiserslautern, 1995
- [K199] Wolfram Kleis: Konzepte zur verständlichen Beschreibung objektorientierter Frameworks. Shaker Verlag Aachen 1999
- [K186] Dietrich Klugmann: Ein Beitrag zur Anwendung des Begriffs der Strukturvarianz in programmierten Systemen. Dissertation, Universität Kaiserslautern 1986
- [Ob01] Object Management Group: What Is OMG-UML And Why Is It Important? www.omg.org/gettingstarted/what_is_uml.htm
- [Ta00a] Peter Tabeling: Der Modellhierarchieansatz zur Beschreibung nebenläufiger, verteilter und transaktionsverarbeitender Systeme. Shaker Verlag, Aachen 2000
- [Ta00b] Peter Tabeling: Der Modellhierarchieansatz zur Beschreibung nebenläufiger und verteilter Systeme. Beitrag zum GI-Workshop „Visuelle Verhaltensmodellierung verteilter und nebenläufiger Systeme“ Münster, 2000
- [We82a] Siegfried Wendt: Einführung in die Begriffswelt allgemeiner Netzsysteme. Regelungstechnik, 30. Jahrgang 1982 Heft 1
- [We82b] Siegfried Wendt: Der Kommunikationsansatz in der Software-Technik. data report 17 (1982) Heft 4
- [We89] Siegfried Wendt: Nichtphysikalische Grundlagen der Informationstechnik - Interpretierte Formalismen. Springer Verlag, Heidelberg 1989
- [Zu90] Wolfgang Zuck: Ein Beitrag zur konsistenten Mitdokumentation von Systementwürfen auf der Basis von Strukturplänen. Dissertation, Universität Kaiserslautern 1990