# Spezifikation und Ausführung von Objektinteraktionen in TROLL *

Thorsten Hartmann

TU Braunschweig, Abteilung Datenbanken, Postfach 3329, D-38023 Braunschweig

E-mail: hartmann@idb.cs.tu-bs.de

**Zusammenfassung**

Der Entwurf von Informationssystemen beginnt in der Regel mit der konzeptionellen Modellierung. Da diese Entwurfsphase entscheidend für den späteren Erfolg des erstellten Systems ist, sollte diese Phase mit formalen Modellen durchgeführt werden, die Konsistenzüberprüfung und durchgängige Werkzeugunterstützung ermöglichen. Für die spätere Akzeptanz und Güte der erstellten Spezifikation ist ein frühes Prototyping unerläßlich. Formale Spezifikationssprachen sind jedoch durch ihren deklarativen Charakter allenfalls teilweise und nur nach entsprechender Transformation ausführbar. Der Artikel skizziert einen Ansatz zur Analyse und nachfolgenden Generierung von Ausführungsplänen für in der Sprache TROLL spezifizierte zusammengesetzte Zustandsübergänge.

## 1 Einleitung

Als Informationssysteme werden interaktive Softwaresysteme bezeichnet, die einen Ausschnitt der *realen Welt* abbilden. Dazu haben Informationssysteme Massendaten zu verwalten und Werkzeuge zu deren Aufbereitung zur Verfügung zu stellen. Informationssysteme sind in der Regel langlebig, dürfen jedoch nicht starr sein. Aus dieser kurzen Charakterisierung leiten sich spezielle Anforderungen für den Entwurf von Informationssystemen her:

- Als Grundlage aller weiteren Entwurfsschritte muß ein von (späteren) Anwendern und Entwerfern[1] gut verstandenes, konsistentes und (im Rahmen der Anwendung) vollständiges Modell erstellt werden.
- Für die Datenverwaltungskomponente sollten Methoden aus dem Bereich des Datenbankentwurfs verwendet werden (vergl. z.B. [Saa93]), da einerseits von konkreten Zielsystemen abstrahiert werden soll (physische Datenunabhängigkeit), andererseits jedoch auch zukünftige Änderungen der Diskurswelt antizipiert werden müssen (logische Datenunabhängigkeit).
- Durch die interaktive Natur von Informationssystemen muß die Benutzerschnittstelle des Systems in frühen Phasen des Entwurfs zunächst bezüglich ihrer Funktionalität spezifiziert werden [SHJ93]. Eine ergonomische Gestaltung kann dann in die späteren Entwurfsphasen verlagert werden.

*Diese Arbeit wurde gefördert von der Deutschen Forschungsgemeinschaft unter Sa 465/1-3 und teilweise von der CEC unter ESPRIT WG 6071 ISCORE II (Information Systems – COrrectness an REusability) bzw. ESPRIT WG 8319 ModelAge (A Common Formal MODEL of AGEnts)

[1] Wenn wir im folgenden männliche Formen benutzen sind immer auch Entwerferinnen gemeint.

Informationssysteme sind somit mehr als Datenbanken mit einer Sammlung von Applikationsprogrammen, statt dessen müssen sie als integrierte Systeme angesehen werden, die ihrerseits wieder in komplexe Strukturen der realen Welt eingegliedert sind. Sie müssen daher bezüglich ihrer Fehlerfreiheit schon aus rechtlichen Gründen oft besonderen Anforderungen genügen. Die Notwendigkeit des Einsatzes von formalen Methoden für die frühen Entwurfsphasen von Informationssystemen ist daher allgemein anerkannt [BM92].

Der Anwendung von formalen Methoden in kommerziellen Umgebungen stehen aber immer noch diverse Punkte entgegen. So erscheint deren Aufwand in frühen Entwurfsphasen immer noch erheblich größer als bei traditionellen Vorgehensweisen. Einen Ausweg aus dieser Problematik können Ansätze aufzeigen, die die Vorteile verschiedener Paradigmen des Softwareentwurfs in einem einheitlichen formalen Rahmen vereinen. Dazu gehören mehr informelle graphische Ansätze, semantische Datenmodellierung, Spezifikation von Prozessen, Spezifikation von abstrakten Datentypen und objekt-orientierte Modellierung. Der Ansatz einer formalen Spezifikationssprache wie TROLL [HSJ+94], zusammen mit einer geeigneten Werkzeugunterstützung (TBENCH), die graphische [WJH+93] wie textuelle Spezifikationen erlaubt und diverse Analyse- und Transformationswerkzeuge bereitstellen wird, geht diesen Weg.

Problematischer als die Hemmschwelle, die es bei der Anwendung von formalen Methoden zu überwinden gilt, ist jedoch die Frage, ob die *modellierte* Welt auch die *reale* Welt hinreichend genau widerspiegelt. Anders als bei formalen Beweis- und Analysemethoden zur Konsistenzsicherung kann hier nur das Experiment Aufschluß geben. Ein Prototyping-Ansatz für TROLL [BP92], der in eine Entwicklungsumgebung integriert sein wird, ist daher zwingend erforderlich [Har93]. Wir werden im folgenden ein spezielles Thema im Bereich dieser Problematik skizzieren: Ein Ausführungsmodell für deklarativ spezifizierte Zustandsübergänge.

# 2 Das Troll Objektmodell

Um die im folgenden dargestellte Thematik verständlich zu machen, wollen wir hier zunächst einen Überblick über das zugrundeliegende Objektmodell von TROLL geben. Für detailliertere Darstellungen sei der interessierte Leser auf den Sprachreport [HSJ+94], oder auch auf semantische Grundlagenarbeiten wie [SSE87, ES91] verwiesen.

TROLL-Objekte sind *eingekapselte* Einheiten im Sinne einer Menge von *Ereignissen*, die auf einem nicht sichtbaren lokalen Zustand arbeiten. Sichtbare Eigenschaften von Objekten werden als datenwertige Attribute modelliert, deren Wert *nur* vom internen Objektzustand abhängig ist. *Objekte* sind damit *beobachtbare Prozesse*. Wir betrachten *Sequenzen* von Ereignissen und *Beobachtungen* über Anfangsstücken solcher Sequenzen.

Ein wesentlicher Teil einer TROLL-Spezifikation besteht aus einer wahlweise deklarativen oder operationalen Prozeßbeschreibung, die mögliche Sequenzen von Ereignissen festlegt. Ein weiterer Teil besteht aus der Beschreibung der aus den Prozessen abgeleiteten Attributbelegungen – mit anderen Worten – der *Zuweisung* von Datenwerten an Attribute bei Eintreten von Ereignissen.

Die Kommunikation von Objekten wird als *Synchronisation* von Objektprozessen modelliert. Kommunikation in TROLL basiert auf einem *synchronen* Nachrichtenaustausch. Eine in manchen Anwendungen natürlicher erscheinende *asynchrone* Kommunikation, wie sie in *Actor*-Formalismen verwendet wird [AWY93], wurde nicht in TROLL eingeführt. Für eine Spezifikationssprache, die formale Beweise über erfolgreiche und nicht erfolgreiche Kommunikationswünsche von Objekten ermöglichen soll, ist synchrone Kommunikation zunächst die natürlichere Wahl. Asynchrone Kommunikation wird hier als ein Implementierungswerkzeug gesehen, mit dem synchrone Kommunikation realisiert werden kann.

Objekte in TROLL kommunizieren durch den *Aufruf von Ereignissen*. Formal wird dazu bei Eintreten eines Ereignisses in einem Objekt die Menge der (transitiv) aufgerufenen Ereignisse berechnet. Diese *Ereignismenge* definiert einen Synchronisationspunkt für verschiedene Objektlebensläufe mit jeweils lokalen Zustandsübergängen. Für die *operationale Ausführung* einer solchen Ereignismenge müssen wir hingegen zusätzlich *Parameter* von Ereignissen betrachten. Mit anderen Worten, die Kontrollstruktur, die durch die *operationale* Spezifikation von Ereignisaufrufen den *Ereignismengen* aufgeprägt ist (Ereignismengen sind partiell geordnet [HS93]), muß für deren Ausführung daher weiter verfeinert werden. Eine *konzeptionell parallele* Ereignisausführung wird dabei in eine *sequentielle* Transaktion umgeformt.

Syntaktisch werden die skizzierten Aspekte soweit als möglich in *Ereignisspezifikationen* zusammengefaßt: Ein Ereignis hat Vorbedingungen, induziert neue Attributbelegungen, bestimmt Antwortwerte an aufrufende Ereignisse und ruft andere Ereignisse auf. Wir werden aus Platzgründen hier keine TROLL Sprachbeschreibung geben, sondern im folgenden nur ein kurzes, selbsterklärendes Beispiel eines Ereignisaufrufes verwenden.

# 3 Ausführung von Ereignismengen

Das im letzten Abschnitt informell eingeführte Modell der Objektkommunikation mit Hilfe von synchronisierten Ereignissen soll nun am Beispiel genauer dargestellt und eine mögliche Formalisierung skizziert werden. Die Formalisierung dient dabei nicht direkt einer *Ausführung* von Ereignismengen, gibt jedoch Anhaltspunkte für die *Analyse* von Spezifikationen zur nachfolgenden Erstellung bzw. Auswahl von *Ausführungsplänen*.

Als Beispiel geben wir folgende (konstruierte) Aufgabe an: In einer zu spezifizierenden „Bankenwelt“ soll von einem Konto ein prozentualer (als Eingabe zur Verfügung gestellter) Betrag des einen Kontos auf das andere transferiert werden. Das initiierende Objekt dieser Aktion (die `Bank`) wird davon unterrichtet, ob das Quell-Konto den zur Befriedigung der Abbuchung benötigten Betrag enthält. Das Bankobjekt enthält dazu eine Ereignisbeschreibung für ein Ereignis `Ueberweisung`:

```
events
  variables UeberweisungsBetrag:money ;
  Ueberweisung(Q:|Konto|,Z:|Konto|,P:real,!Ok:bool)
    calling
      Konto(Q).ProzentVomSaldo(P,UeberweisungsBetrag) ;
      Konto(Q).Abbuchen(UeberweisungsBetrag,Ok) ;
      { Ok = true } Konto(Z).Einzahlen(UeberweisungsBetrag) .
```

`Ueberweisung` hat als Parameter Objektidentitäten von Quelle- und Ziel-Konto `Q` und `Z` (zu Objektidentitäten in TROLL siehe [HSJ$^+$94]), den Prozentsatz `P` sowie den *Rückgabewert* `Ok`. Der Rückgabewert wird in der Signatur des Ereignisses mit einem „*!*“ gekennzeichnet. Er darf bei Aufruf des `Ueberweisung`-Ereignisses nicht instantiiert sein. Das Ereignis `Ueberweisung` initiiert in den beteiligten Kontenobjekten ein Ereignis zur Bestimmung des zu übertragenden Betrages, sowie Ereignisse zur Durchführung der Ueberweisung. Das `Abbuchung`-Ereignis bestimmt dabei den Rückgabewert, das `ProzentVomSaldo`-Ereignis den abzubuchenden Betrag. Sie sind (für Kontenobjekte) wie folgt spezifiziert:

```
events
  Abbuchen(Betrag:money,!Ok:bool)
    changing { Saldo>=Betrag }  Saldo := Saldo - Betrag
    binding Ok = (Saldo >= Betrag) .
  Einzahlen(Betrag:money)
    changing Saldo := Saldo + Betrag .
  ProzentVomSaldo(P:real,Betrag:money)
    binding Betrag = (Saldo * P / 100) .
```

Bei Eintreten des `Abbuchen`-Ereignisses wird eine konkrete Änderung des (sichtbaren) Objektzustandes nur dann durchgeführt, wenn genug Geld vorhanden ist (*changing*). Die *binding*-Spezifikation instantiiert den Parameter `Ok`. Es soll an dieser Stelle bemerkt werden, daß in TROLL auch das Ereignis `Abbuchen` – bei nicht genügender Deckung des Kontos – durch eine Vorbedingung verboten werden kann. Die Modellierung wäre so natürlicher, da bei ungenügender Deckung das *Ereignis* `Abbuchen` nicht stattfinden kann. Hier sollte jedoch zur Verdeutlichung ein lokal bestimmter Parameter (traditionell ein Rückgabewert) modelliert werden.

Auf der Spezifikationsebene finden diese vier Ereignisse *synchron* statt. Sie definieren zusammen einen komplexen Zustandsübergang der drei beteiligten Objekte Konto `Q`, Konto `Z` und `Bank`. Die *Ausführung* eines solchen Zustandsüberganges durch ein Prototypingwerkzeug hingegen muß eine mögliche *Reihenfolge* der Ereignisse unter Berücksichtigung des *Kontroll-* sowie des *Datenflusses* berücksichtigen.

Aus operationaler Sicht ist intuitiv in diesem Beispiel zu sehen, daß das `ProzentVomSaldo` Ereignis *vor* den Ereignissen `Abbuchen` und `Einzahlen` stattfinden muß, sowie daß das Ereignis `Ueberweisung` zunächst in reduzierter Form (ohne den `Ok` Parameter) und frühestens nach dem `Abbuchen`-Ereignis vollständig stattfinden kann.

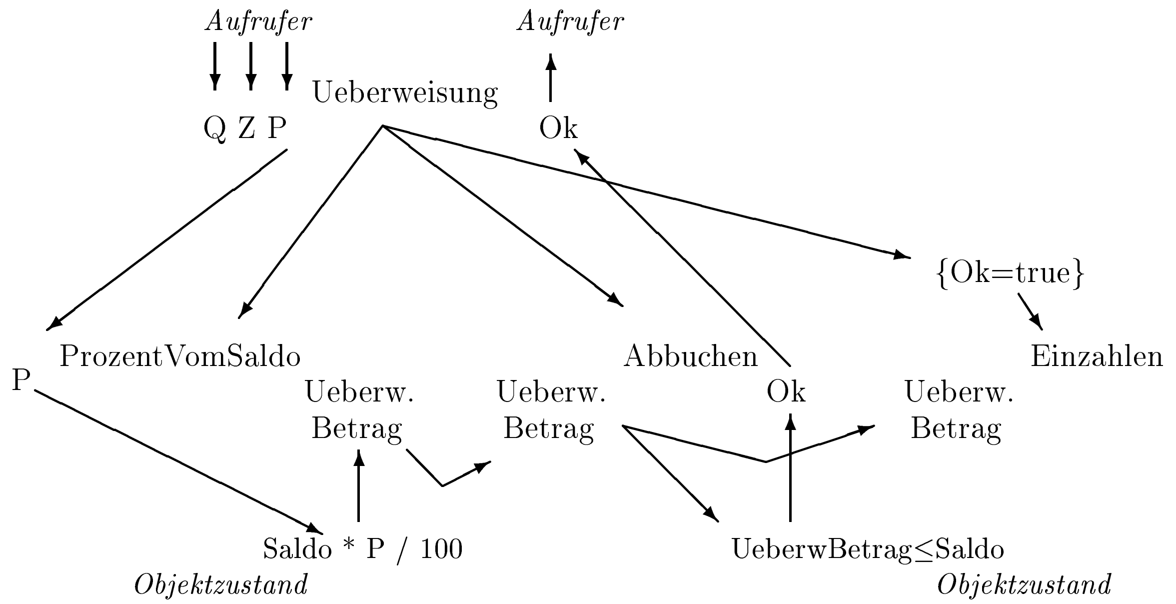
Formal haben die so spezifizierten Aufrufegeln Ähnlichkeiten mit *Regeln* in Logikprogrammen sowie mit *Produktionen* von (kontextfreien) Grammatiken. `ProzentVomSaldo` kann in ersterem Sinne als *Anfrage* betrachtet werden, die den freien Parameter `UeberweisungsBetrag` instantiiert. Betrachtet man kontextfreie Grammatiken mit *Attributen* für Symbole, die konkrete Instantiierungen in Ableitungsbäumen näher beschreiben, kommt man zu *Attributierten Grammatiken*. In einer attributierten Grammatik werden zu jeder Produktion Berechnungsregeln für Attribute der beteiligten Symbole angegeben, die sich auf die Attribute der sonstigen Symbole der Regel beziehen können (eine formale Darstellung findet sich in [Knu68]). Werte können auf diese Weise zwischen den beiden Seiten der Produktion, d.h. im Ableitungsbaum von unten nach oben (*berechnet*) bzw. umgekehrt (*ererbt*), sowie auch zwischen den Ableitungsbäumen, die durch die rechten Seiten repräsentiert werden, weitergegeben werden.

TROLL Aufrufregeln können mit geringen Modifikationen in attributierte Grammatiken transformiert werden [HS93]. Eine ähnliche Vorgehensweise wurde in [Tsa89] formal ausgearbeitet. Dort werden Eigenschaften attributierter Grammatiken zur Optimierung von Prologprogrammen benutzt. Für Aufrufspezifikationen werden Ereignissymbole mit Nicht-Terminal-Symbolen (kurz Symbole) der Grammatik und Ereignisparameter mit Attributen für Symbole identifiziert. Für attributierte Grammatiken existieren Algorithmen zur Analyse von Abhängigkeiten zwischen Attributen. Auf diese Weise können *Klassen* von Grammatiken bestimmt werden, die für bestimmte Attribut-Auswertungsalgorithmen notwendig sind [Wil79]. Hier soll diese Information jedoch nicht zur Attribut-Auswertung, sondern zur Bestimmung von Ausführungsplänen genutzt werden.

Mit Aufrufspezifikationen ist die Reihenfolge der aufgerufenen Ereignisse bei der späteren Ausführung zunächst nicht festgelegt. Eine Grammatik legt dagegen in der traditionellen Interpretation die möglichen Sequenzen von Terminalzeichen fest. Da wir in der skizzierten Transformation keine Terminalzeichen definiert haben, wird hier die Reihenfolge der *Unterbäume* des Ableitungsbaumes von links nach rechts zugrundegelegt. Die Transformation von Aufrufregeln in eine attributierte Grammatik ist daher nicht eindeutig, sondern erzeugt eine *Menge* möglicher attributierter Grammatiken (Permutationen der rechten Seiten der Aufrufspezifikationen) beispielsweise die folgende (siehe auch [HS93]):

```
Grammatik G:
  V={Ueberweisung,ProzentVomSaldo,Abbuchen,Einzahlen}, v0={Ueberweisung},
  P={p0: Ueberweisung → ProzentVomSaldo Abbuchen Einzahlen}
Attribute (Für Symbole in V):
  I(Ueberweisung)={Q,Z,P},            S(Ueberweisung)={Ok},
  I(ProzentVomSaldo)={P},             S(ProzentVomSaldo)={UeberweisungsBetrag},
  I(Abbuchen)={UeberweisungsBetrag},  S(Abbuchen)={Ok},
  I(Einzahlen)={UeberweisungsBetrag}, S(Einzahlen)=∅
Auswertungsregeln der att. Grammatik (für p0):
  Ueberweisung.Q, Z und P := ``vom Aufrufer zu liefern'',
  ProzentVomSaldo.P := Ueberweisung.P,
  ProzentVomSaldo.UeberweisungsBetrag := ``Saldo'' * ProzentVomSaldo.P / 100,
  Abbuchen.UeberweisungsBetrag := ProzentVomSaldo.UeberweisungsBetrag,
  Abbuchen.Ok := Abbuchen.UeberweisungsBetrag ≤ ``Saldo'',
  Einzahlen.UeberweisungsBetrag := Abbuchen.UeberweisungsBetrag,
  Ueberweisung.Ok := Abbuchen.Ok .
```

Die diversen Abhängigkeiten (Daten- wie Kontrollfluß) in der Beispielspezifikation sind in folgendem Bild nochmals verdeutlicht. Attribute der Ereignissymbole sind jeweils rechts (berechnet) und links (ererbt) unter den Symbolen notiert:




*Aufrufer* bezieht sich dabei auf ein nicht weiter spezifiziertes Objekt (durch calling), oder auf den Benutzer eines Prototypingsystems, der testweise das Ereignis `Ueberweisung` auszulösen versucht. *Objektzustand* skizziert die Umgebung, in der die Ereignisse ausgeführt werden und bezieht sich in diesem Beispiel auf das Quellkonto (`Saldo`).

Eine bestimmte Grammatik kann nun als ein möglicher *Ausführungsplan* für Ereignismengen angesehen werden, wenn für jedes Ereignis in der so erhaltenen Reihenfolge jeweils alle Parameter bekannt sind, das Ereignis also vollständig instantiiert ist. Grammatiken mit dieser Eigenschaft werden als *L-attributiert* bezeichnet [Wil79]. Für Anwendungen in formalen Sprachen besagt diese Eigenschaft, daß die Attribut-Auswertung mit der von links nach rechts im Wort fortschreitenden Syntaxanalyse – ohne Knoten des Baumes mehrfach zu besuchen – durchgeführt werden kann.

In unserem Kontext müssen Attributberechnungen *entgegen* den durch die Aufrufrelation bestimmten Kontrollfluß jedoch noch einer weiteren Transformation unterworfen werden. Wenn der Rahmen von Ereignissen als unteilbaren Einheiten nicht verlassen werden soll, müssen solche Ereignisse wie `Ueberweisung` im o.g. Beispiel in zwei neue Ereignisse umgewandelt werden [HS93]. Die bei der Analyse der generierten Grammatiken gewonnene Information kann dazu Anhaltspunkte liefern. Es wird in diesem Zusammenhang nochmal darauf hingewiesen, daß nicht an die vollständige Transformation von Aufrufregeln in attributierte Grammatiken gedacht ist, jedoch theoretische Ergebnisse teilweise für die Abhängigkeitsanalyse genutzt werden können.

# 4 Zusammenfassung und Ausblick

Prototyping von Spezifikationen für Informationssysteme wurde als ein Mittel zur *Validierung* von Entwürfen kurz skizziert und als ein entscheidendes Mittel zur Qualitätssicherung und Akzeptanzsteigerung des später zu implementierenden Systems dargestellt (siehe auch [Har93, BP92]). Die deklarative Natur von Spezifikationen in Troll steht einer direkten Ausführung, z.B. von komplexen Zustandsübergängen einer Objektgesellschaft, entgegen. Zur (prototypischen) Ausführung müssen daher zunächst Ausführungspläne erstellt werden. Es wurde die Nutzung eines Formalismus – attributierte Grammatiken – vorgeschlagen, der die für die Erstellung von Ausführungsplänen notwendigen theoretischen Ergebnisse und Algorithmen bereitstellt. Die Ausarbeitung von Kriterien für die Analyse von aus Aufrufspezifikationen erzeugten Abhängigkeitsgraphen bzw. den dazu äquivalenten Grammatiken zur Generierung von Ausführungsplänen ist gegenwärtig daher ein konkreter Gegenstand weiterer Arbeiten. Außerdem muß die hier aus Platzgründen nicht betrachtete Zyklenfreiheit von Parameterabhängigkeitsgraphen überprüft werden. Auch für dieses Problem existieren Lösungen aus dem Gebiet der attributierten Grammatiken. Nach der Erstellung von Ausführungsplänen kann dann die Ausführung der einzelnen Ereignisse – gekapselt als eine Transaktion – durchgeführt werden. Da Aufrufmengen zustandsabhängig sind [HSJ+94], muß geprüft werden, inwieweit die skizzierten Methoden zur Spezifikationszeit und wann sie zur Laufzeit durchgeführt werden müssen. Zur Zeit wird eine eingeschränkte Implementierung des skizzierten Ausführungsmodells in *C++* im Rahmen einer Diplomarbeit [Sch94] durchgeführt. Nebenbedingung dieser Arbeit ist eine verteilte Umgebung. Einschränkungen wurden daher in Bezug auf die allgemeine Parameterübergabe vorgenommen. Ergebnisse des hier skizzierten Analyseprozesses sollen dann in die Troll–*C++* Übersetzung eingearbeitet werden.

## Danksagung

Wir danken allen Kollegen und Kolleginnen in Braunschweig für Diskussionen über die Thematik dieser Arbeit. Besonderer Dank gilt hier Gunter Saake für Anregungen bezüglich der Analyse von Spezifikationen.

## Literatur

[AWY93] Agha, G.; Wegner, P.; Yonezawa, A. (Hrsg.): *Research Directions in Concurrent Object-Oriented Programming.* MIT Press, 1993.

[BM92] Barroca, L.M.; McDermid, J.A.: Formal Methods: Use and Relevance for the Development of Safety-Critical Systems. *The Computer Journal*, Band 35, Nr. 6, 1992, S. 579–599.

[BP92] Bischofberger, W.; Pomberger, G.: *Prototyping-Oriented Software Development.* Springer, Berlin, 1992.

[ES91] Ehrich, H.-D.; Sernadas, A.: Fundamental Object Concepts and Constructions. In: Saake, G.; Sernadas, A. (Hrsg.): *Information Systems – Correctness and Reusability.* TU Braunschweig, Informatik Bericht 91-03, 1991, S. 1–24.

[Har93] Hartmann, T.: Operationalisierung von Objektspezifikationen. In: Thalheim, B. (Hrsg.): *Tagung des GI-Arbeitskreises: Grundlagen von Informationssystemen.* Univ. Rostock, 1993, S. 64–68.

[HS93] Hartmann, T.; Saake, G.: Abstract Specification of Object Interaction. Informatik-Bericht 93–08, Technische Universität Braunschweig, 1993.

[HSJ+94] Hartmann, T.; Saake, G.; Jungclaus, R.; Hartel, P.; Kusch, J.: Revised Version of the Modelling Language TROLL. Informatik-Bericht 94-03, TU Braunschweig, 1994.

[Sch94] Scholz, T.: Ansätze zur Implementierung von TROLL Objektinteraktionen in C++, 1994. Diplomarbeit, *In Vorbereitung.*

[Knu68] Knuth, D.E.: Semantics of Context Free Languages. *Math. Systems Theory*, Band 2, 1968, S. 127–145.

[Saa93] Saake, G.: *Objektorientierte Spezifikation von Informationssystemen.* Teubner, Stuttgart/Leipzig, 1993. Habilitationsschrift.

[SHJ93] Saake, G.; Hartmann, T.; Jungclaus, R.: Objektspezifikation von Benutzerschnittstellen in TROLL. In: Mayr, H.C.; Wagner, R. (Hrsg.): *Objektorientierte Methoden für Informationssysteme.* Springer, Berlin, Informatik aktuell, 1993, S. 173–186.

[SSE87] Sernadas, A.; Sernadas, C.; Ehrich, H.-D.: Object-Oriented Specification of Databases: An Algebraic Approach. In: Stoecker, P.M.; Kent, W. (Hrsg.): *Proc. VLDB'87.* VLDB Endowment Press, Saratoga (CA), 1987, S. 107–116.

[Tsa89] Tsakpinis, A.: *Werkzeuge zur Optimierung von Logischen Programmen: Erkenntnisse aus einem Vergleich mit Attributierten Grammatiken.* Dissertation, Univ. Erlangen, 1989.

[Wil79] Wilhelm, R.: Attributierte Grammatiken. *Informatik-Spektrum*, Band 2, 1979, S. 139–131.

[WJH+93] Wieringa, R.; Jungclaus, R.; Hartel, P.; Hartmann, T.; Saake, G.: OMTROLL – Object Modeling in TROLL. In: Lipeck, U.W.; Koschorreck, G. (Hrsg.): *Proc. IS-CORE '93, Univ. of Hannover, Informatik Bericht 01/93*, 1993, S. 267–283.