Entwicklung zuverlässiger Informationssysteme

S. Conrad, G. Denker, M. Gogolla, R. Herzig, N. Vlachantonis und H.-D. Ehrich
Technische Universität Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D-38023 Braunschweig, Germany

1 Einleitung

Die zunehmende Komplexität von Softwaresystemen führt dazu, daß Fragen der Zuverlässigkeit und Korrektheit solcher Systeme eine immer zentralere Bedeutung erlangen. Ein Beleg hierfür sind die verschiedenen nationalen und internationalen Bemühungen um Zertifikationsverfahren für Software [Krü93, Rom93]. Insbesondere unter wirtschaftlichen Gesichtspunkten erweist es sich als sinnvoll, schon in den frühen Phasen geeignete Gütekriterien zu berücksichtigen, um späteren, meist kostenintensiven Wartungs– und Anpassungsaufwand in engen, kalkulierbaren Grenzen halten zu können.

Darüber hinaus zeigen verschiedene Bemühungen um eine Standardisierung des Softwareentwurfsprozesses (z.B. das V-Modell [BD93]), daß in Zukunft auch die Anforderungen der Auftraggeber den Nachweis einer durchgängigen Entwicklungsmethodik und den Einsatz formaler Methoden notwendig machen werden. Insbesondere in Hinblick auf sicherheitskritische Eigenschaften wird bald deren formaler Nachweis Vertragsgrundlage werden.

Diese sich abzeichnende Entwicklung hat uns dazu veranlaßt, die formalen Aspekte bei der Entwicklung von Informationssystemen (als eine große Klasse von Softwaresystemen) näher zu untersuchen. Dies geschah im wesentlichen im Rahmen des Forschungsprojektes KorSo[BJ93][1], in dem die Konsolidierung formaler Methoden im Mittelpunkt stand. Verschiedene Gruppen arbeiteten dabei zusammen an folgenden Schwerpunkten: Methodik, Beschreibung und Modellierung, Fallstudien und Werkzeuge.

Die Arbeiten unserer Gruppe dabei lassen sich wie folgt zusammenfassen: Ausgehend von einem einfachen objektorientierten Ansatz zur Modellierung von Objektgesellschaften als Informationssysteme wurden insbesondere eine geeignete Entwicklungsumgebung konzipiert, der Bereich Validation von Objektspezifikationen mittels Animation untersucht und ein Beweisunterstützungssystem entworfen, das formale Nachweise von Objekteigenschaften ermöglicht. Ziel ist die Entwicklung von Informationssystemen auf der Grundlage von objektorientierten Datenbanksystemen [KM93, Pis93].

Im folgenden werden wir zunächst die Objektbeschreibungssprache TROLL *light* vorstellen, die als Grundlage der in Angriff genommenen Aufgaben essentielle Konzepte zur Informationssystem–Modellierung mittels interagierender Objekte bietet. Dann geben wir eine kurze Übersicht über die zugehörige Entwicklungsumgebung bevor wir näher auf zwei wesentliche Komponenten, den Animator und das Beweisunterstützungssystem, eingehen.

2 Konzepte von TROLL *light*

TROLL *light* [CGH92, GCH93, HCG94] ist eine Sprache zur Beschreibung von strukturellen und dynamischen Eigenschaften von Objekten. Die Beschreibung von strukturellen Eigenschaften orientiert sich in weiten Teilen an semantischen Datenmodellen.

- Die in einem Zustand beobachtbaren Eigenschaften eines Objektes werden durch Attribute beschrieben. Einfache Attribute sind daten– oder objektwertig. Darüber hinaus erlauben vordefinierte Sortenkonstruktoren wie *set* oder *tuple* auch die Spezifikation komplexer Attributbereiche.
- TROLL *light*–Objekte sind in Objekthierarchien, die sich aus Unterobjektbeziehungen ergeben, organisiert. Eine Unterobjektbeziehung

[1] BMFT-Verbundprojekt KorSo(= KORrekte SOftware), Förd.Nr. (IT) 01 IS 203 D, 0kt.'91 – Juni'94.

ist hierbei als eine exklusive „Teil von“– bzw. Komponentenbeziehung zu verstehen.

- Normalerweise werden Attributwerte direkt durch das Eintreten gewisser Ereignisse bestimmt. Daneben ist es auch möglich, abgeleitete Attribute zu spezifizieren, d.h. Attribute, deren Inhalte sich aus anderer gespeicherter oder abgeleiteter Information bestimmen. Zur Formulierung von Ableitungsregeln stellt TROLL *light* einen SQL–ähnlichen Anfragekalkül zur Verfügung.
- Der Anfragekalkül von TROLL *light* unterstützt auch die Spezifikation statischer Integritätsbedingungen.

Die Beschreibung von dynamischen Eigenschaften basiert auf der Spezifikation von Ereignissen. Ereignisse sind Abstraktionen von zustandsverändernden Operationen auf Objekten.

- Objektereignisse werden durch eine endliche Menge von Ereignisgeneratoren beschrieben. Jeder Ereignisgenerator kann mit einer Liste von Parametersorten versehen sein. Ein Ereignisgenerator mit aktuellen Parameterwerten liefert ein Ereignis.
- Die Auswirkung von Ereignissen auf Attribute werden durch Auswertungsregeln beschrieben.
- Ereignisse in verschiedenen Objekten können durch Interaktionsregeln synchronisiert werden.
- Die möglichen Ereignisfolgen können mit Hilfe CSP-ähnlicher Prozeßbeschreibungen auf zulässige Folgen eingeschränkt werden.

Den Rahmen zur Beschreibung all dieser Objekteigenschaften bilden *Templates*. Als Beispiel geben wir in Abb. 1 die Beschreibung von Objekten an, die Autoren in einem Bibliotheksinformationssystem repräsentieren sollen.

Die so beschriebenen Autoren–Objekte haben einen Namen, ein Geburtsdatum und Anzahlen verkaufter Bücher als Attribute, wobei für jedes Jahr eine Anzahl verkaufter Bücher gespeichert werden kann. Ereignisse, die im Leben eines Autoren–Objektes eintreten können, sind sein Geburtsereignis, eine Namensänderung, das Abspeichern von Verkaufszahlen und schließlich ein Todesereignis. Im `VALUATION`–Abschnitt ist beschrieben, welche Auswirkungen das

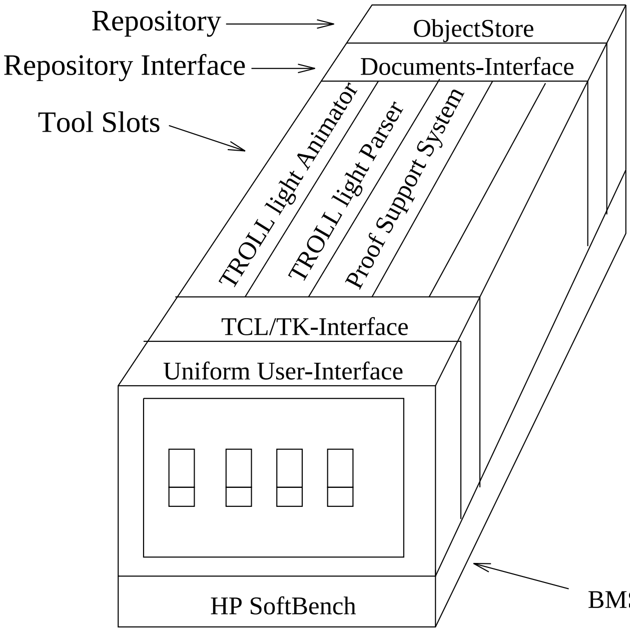

Abbildung 2: Architektur der TROLL *light*–Entwicklungsumgebung.

Eintreten eines Ereignisses auf Attribute hat. Der Verhaltensteil legt die zulässigen Lebensläufe von Autoren–Objekte fest.

3 Die TROLL *light*–Entwicklungsumgebung

Die Entwicklung großer Software– und Informationssysteme ist ein aufwendiger Prozeß, der verschiedene Methoden und Techniken zur Verminderung des Zeitbedarfs und der Kosten erfordert. Um das zu unterstützen, sind Software–Werkzeuge (Tools) notwendig (z.B. CASE–Werkzeuge wie in [KEK+94]). Um effektiv arbeiten zu können, sollten diese Werkzeuge innerhalb einer Software–Entwicklungsumgebung integriert werden. Kern unserer Entwicklungsumgebung ist die zuvor skizzierte Sprache TROLL *light*.

Die TROLL *light*–Entwicklungsumgebung ist ein offenes System, das von seinen Anwendern an neue Anforderungen angepaßt werden kann und in das neue Tools integriert werden können [VHG+93]. Bedingungen für die Integration betreffen u.a. die Anpassung neuer Tools, d.h. diese müssen in der Lage sein, von vorhandenen Tools erzeugte Daten und

```
TEMPLATE Author
  DATA TYPES   String, Date, Nat;
  ATTRIBUTES   Name:string; DateOfBirth:date;
               SoldBooks(Year:nat):nat;
  EVENTS       BIRTH create(Name:string, DateOfBirth:date);
                     changeName(NewName:string);
                     storeSoldBooks(Year:nat, Number:nat);
               DEATH destroy;
  VALUATION    [create(N,D)] Name=N, DateOfBirth=D;
               [changeName(N)] Name=N;
               [storeSoldBooks(Y,NR)] SoldBooks(Y)=NR;
  BEHAVIOR     PROCESS AuthorLife1 =
                 ( storeSoldBooks -> AuthorLife1 |
                   changeName -> AuthorLife2 |
                   destroy );
               PROCESS AuthorLife2 =
                 ( storeSoldBooks -> AuthorLife2 |
                   destroy );
               ( create -> AuthorLife1 );
END TEMPLATE;
```

Abbildung 1: Objektbeschreibung für Autoren.

Dokumente (Spezifikationen, Quelltexte, etc.) weiterzuverarbeiten. Das ist ein sehr wichtiger Aspekt, da Daten und Dokumente strukturiert in einem Datenbanksystem (Repository) abgespeichert werden.

Um die Entwicklungsumgebung weitestgehend offen für Erweiterungen und Anpassungen zu halten, müssen die Tools möglichst unabhängig voneinander sein. Daher unterstützt die TROLL *light*–Entwicklungsumgebung eine lose Kopplung der Tools. Das wird durch eine Tool–Kommunikation erreicht, die auf dem Austausch von Nachrichten (im Prototyp realisiert durch die HP Softbench) und auf der Speicherung der Daten in einem zentralen Repository basiert [DHS+91].

Nachrichten sind hierbei kein Mittel, um Daten auszutauschen, sondern dienen dem Austausch integritätserhaltender Informationen und der Steuerung der Tools [VH93]. Verändert beispielsweise ein Tool ein Dokument im Repository, so verschickt es eine Nachricht darüber an die Umgebung, um andere Tools darüber zu informieren. Der Austausch von Nachrichten wird durch den *Broadcast Message Server* (BMS) verwaltet, der Nachrichten selektiv nur an betroffene Tools weiterleitet. Neue Tools können integriert werden, ohne daß existierende angepaßt werden müssen. Im allgemeinen ist nur eine Aktualisierung der Informationen des BMS über die Weiterleitung von Nachrichten nötig.

Ferner ist die TROLL *light*–Entwicklungsumgebung eine Instantiierung des *ECMA*–Referenzmodells für Entwicklungsumgebungen (siehe Abb. 2). Sie ist auf den dargestellten Integrationsrahmen BMS, ObjectStore [LLOW91] und TCL/TK (User-Interface-Manager) [Ous93] aufgebaut.

Zwei wichtige Komponenten der TROLL *light*–Entwicklungsumgebung sind der Animator und das Beweisunterstützungssystem. Der Animator dient dem Prototyping einer mit TROLL *light* spezifizierten Objektgesellschaft und das Beweisunterstützungssystem hilft beim formalen Nachweis bestimmter Eigenschaften spezifizierter Objekte. Eine andere geplante, hier nicht näher vorgestellte Komponente ist ein Tool, das den Objektansatz von TROLL *light* mit der etablierten Entity-Relationship–Modellierungstechnik verbindet [GHC+94].

4 Der Animator

Motivation. Zwischen der angestrebten Formalität und der Verständlichkeit eines konzeptionellen Schemas besteht oftmals ein Zielkonflikt. Man vergleiche zum Beispiel Datenflußdiagramme als intuitive Darstellungen mit vager Semantik („Was ist ein Daten- bzw. Kontroll-, Informations-, Materialfluß?") mit algebraischen Spezifikationen als Beschreibungen mit klar definierter Semantik, aber aus Benutzersicht deutlich verminderter Zugänglichkeit. Ein formal beschriebenes konzeptionelles Schema muß deshalb zur Validierung dem Auftraggeber oder späteren Anwender eines zu entwickelnden Systems in irgendeiner Weise *vermittelt* werden. Hierzu gibt es verschiedene Techniken.

Eine weitverbreitete Vorgehensweise besteht in der *Umwandlung* eines konzeptionellen Schemas in leichter verständliche Darstellungen. Hierbei kommt insbesondere die Anwendung graphischer Darstellungsmittel oder die Rückübersetzung einer Spezifikation in eine natürlichsprachliche Form in Frage. Ein anderer Ansatz basiert auf der formalen *Analyse* konzeptioneller Schemata. Durch logisches Schließen können aus einem gegebenen Schema bestimmte Folgerungen abgeleitet und mit dem Anwender diskutiert werden. Ein Weg, der in besonderer Weise dazu geeignet ist, eine Modellbildung mit informellen Benutzervorstellungen abzugleichen, besteht im *experimentellen Prototyping* wichtiger Schemabestandteile [LL93].

Das Prototyping funktionaler Spezifikationen erfolgt in der Regel durch die Anwendung spezifizierter Funktionen auf konkrete oder symbolische Eingabewerte. Beim Testen nichtdeterministischer Systeme müssen dahingegen im allgemeinen bestimmte Ereignisse vorgegeben werden. Dies kann unter Umständen durch die Vorgabe gewisser Auftretenswahrscheinlichkeiten simuliert werden. Überläßt man dem Anwender die Vorgabe von Ereignissen zum Austesten bestimmter Szenarios, spricht man auch von *Animation* [CRB93].

Animation von Objektbeschreibungen. Die Animation von Objektbeschreibungen durch Vorgabe von Mengen parallel eintretender Ereignisse kann dazu dienen, das spezifizierte Verhalten konzeptioneller Objekte mit dem beabsichtigten Verhalten zu vergleichen. Dabei ist zu beachten, daß die Validierung einer Spezifikation den gleichen Einschränkungen unterliegt wie das Testen eines Programms. Die Feststellung, daß das spezifizierte Verhalten in den getesteten Fällen dem geforderten Verhalten entspricht, kann lediglich dazu dienen, das Vertrauen in eine Spezifikation zu erhöhen. Solange es aber nicht getestete Szenarios gibt, welche das Gegenteil zeigen könnten, bleibt die Korrektheit einer Spezifikation ungewiß.

Architektur des Animationssystems. Die Animation von Objektbeschreibungen sollte innerhalb einer Entwicklungsumgebung softwaremäßig unterstützt werden. Ein entsprechendes Werkzeug muß zum einen die Untersuchung aktueller Zustände einer Objektgesellschaft ermöglichen, zum anderen das Einleiten von Zustandsübergängen durch die Spezifikation von Mengen gleichzeitig eintretender Ereignisse erlauben.

Die Struktur des TROLL *light*-Animationssystems ist in Abb. 3 dargestellt (Pfeile drücken „benutzt"-Beziehungen aus). Das System basiert zunächst auf einem Speicher zur strukturierten Ablage von Objektbeschreibungen (engl. *template dictionary*). Dieser Speicher ist nicht exklusiver Bestandteil des Animationssystems, sondern zentrale Komponente der ganzen Entwicklungsumgebung, welche zum Beispiel auch vom Parser und dem Beweisunterstützungssystem benutzt wird. Die zweite grundlegende Komponente des Animationssystems stellt eine Objektbank dar, welche als persistenter Speicher für Objektzustände dient.

TROLL *light*-Objektbeschreibungen umfassen Terme (und Formeln) eines SQL-ähnlichen Anfragekalküls. Zur Auswertung dieser Terme ist ein eigenes Softwaremodul zuständig. Die Auswertung von Termen hängt generell vom aktuellen Zustand einer Objektgesellschaft ab, weshalb der Termauswerter Zugriff auf die Objektbank haben muß. Das Ausführungsmodul stellt das Herz des Animationssystems dar: Zu einem aktuellen Zustand und einer gegebenen Ereignismenge bestimmt das Ausführungsmodul einen möglichen Folgezustand bzw. berichtet über eventuell auftretende Fehlersituationen. Schließlich bildet die Benutzeroberfläche die Schnittstelle zu Anwendern des Animationssystems.

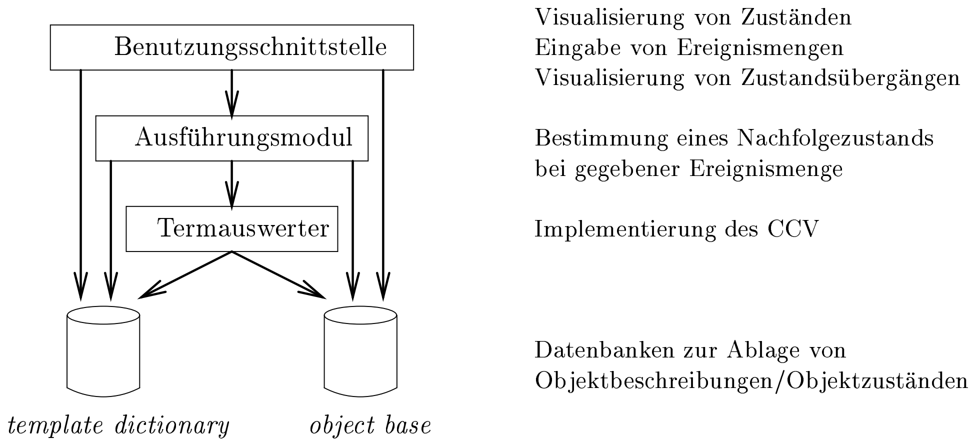

Abbildung 3: Architektur des TROLL *light*–Animationssystems.

Die Benutzeroberfläche wurde mit Hilfe von TCL/TK realisiert. Objektzustände werden im Rahmen von Fenstern zu Objekten visualisiert, welche unter anderem eine Darstellung der aktuellen Unterobjekte und Attributwerte eines Objekts umfassen (s. Abb. 4). Zusätzlich enthalten Objektfenster eine Liste der spezifizierten Ereignisgeneratoren. Durch Mausklick und Eingabe eventuell verlangter Ereignisparameter können zur Ausführung anstehende Ereignisse in eine spezielle *event collection box* übertragen werden, von wo aus sie schließlich dem Ausführungsmodul zur Berechnung eines möglichen Zustandsübergangs übergeben werden.

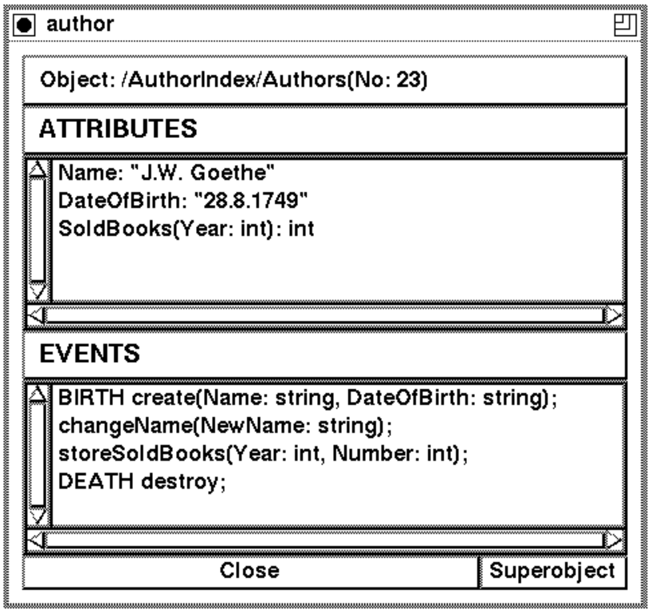

Abbildung 4: Visualisierung von Objektzuständen.

5 Das Beweisunterstützungs–system

Motivation. Mit Hilfe der Beweisunterstützungssystem soll es möglich sein, Eigenschaften von Objekten zu formulieren und diese dann unter Verwendung ihrer Spezifikation zu beweisen. Benötigt wird dazu ein Kalkül, in dem solche Eigenschaften ausgedrückt werden können. Ferner müssen TROLL *light*-Spezifikationen in diesen Kalkül übersetzt werden können [Con94]. Der für diese Zwecke entwickelte Verifikationskalkül berücksichtigt alle Konzepte von TROLL *light*, so daß beliebige TROLL *light*-Spezifikationen in Formelmengen dieses Kalküls übersetzt werden können. Um ein möglichst einfaches Ableitungssystem zu erhalten, beschränken wir uns auf eine bestimmte Formelstruktur. Dieser Kalkül ist als Grundlage für weitere Arbeiten entworfen worden. Daher ist er sehr elementar gehalten und umfaßt in der hier vorgestellten Fassung keine über TROLL *light* hinausgehenden Konzepte.

Im folgenden werden wir den Kalkül kurz und informell skizzieren. Anschließend präsentieren wir die Verwendung des Beweisunterstützungssystems, indem wir den von uns angestrebten Arbeitsablauf beim Verifizieren von Eigenschaften darstellen.

Verifikationskalkül. Um einen Eindruck von dem Verifikationskalkül zu vermitteln, stellen wir die Hauptkonzepte anhand von Beispielen vor. Das zentrale Konzept ist das Konzept der Formeln. Wie bereits erwähnt, erlauben wir nur eine spezielle Form von Klauseln als Formeln: $P_1, \ldots, P_n \to Q_1, \ldots, Q_m$ wobei die P_i und Q_j sogenannte Propositionen (Aussagen) sind. Eine solche Formel ist folgendermaßen zu verstehen: In jedem Zustand (der Objektgesellschaft), in dem alle Propositionen $P_1, \ldots, P_n$ erfüllt sind, ist auch mindestens eine der Propositionen $Q_1, \ldots, Q_m$ erfüllt.

Als nächstes müssen wir nun Propositionen näher betrachten. Elementare Propositionen sind Prädikatausdrücke $p(t_1, \ldots, t_n)$ (mit einem Prädikatsymbol p und geeigneten Termen t_i). Darüber hinaus erlauben wir Negation von Propositionen (also $\neg P$) und die Verwendung positionaler Operatoren (d.h., $[t_o.t_e]P$, wobei t_o ein Term ist, der ein Objekt bezeichnet, und t_e ein Term, der ein Ereignis für dieses Objekt beschreibt). Eine Proposition $[t_o.t_e]P$ kann wie folgt gelesen werden: „Wenn das nächste im Objekt t_o auftretende Ereignis t_e ist, dann gilt danach P". Wir unterscheiden zwischen zwei Arten von Prädikatsymbolen: zustandsabhängige und zustandsunabhängige. Beispielsweise sind Prädikatsymbole aus einer Datensignatur zustandsunabhängig (z.B. $\leq$ für ganze Zahlen), während Attribute von Objekten durch zustandsabhängige Prädikate dargestellt werden. Darüber hinaus gibt es zustandsabhängige Prädikate, um die Zulässigkeit (*enable*) und das tatsächliche Auftreten (*occur*) von Ereignissen behandeln zu können.

Die folgenden Formeln beschreiben Eigenschaften von Objekten, die Personen modellieren. Als Attribute haben diese Objekte einen Namen ($Name$) und einen Familienstand ($Status$). Als Ereignis betrachten wir hier die Namensänderung ($changeName$).

$$\begin{array}{l} Status(P, ledig) \ \to \\ \quad [P.changeName(N)]Status(P, verheiratet) \end{array}$$

$$\to \ [A.changeName(N)] \neg\, enable(A.changeName(N))$$

Die erste Formel gibt an, daß das Attribut $Status$ eines Personenobjektes P nach Eintreten eines $changeName$–Ereignisses für dieses Objekt den Wert $verheiratet$ annimmt, vorausgesetzt dieses Attribut hatte zuvor den Wert *ledig*. Die zweite Formel beschreibt, daß nach dem Eintreten eines $changeName$–Ereignisses in einem Personenobjekt P ein nochmaliges Eintreten eines $changeName$–Ereignisses für dieses Objekt nicht erlaubt ist.

Eine kurze Bemerkung zu der Frage, welche Arten von Eigenschaften mit diesem Verifikationskalkül bewiesen werden können, ist hier angebracht. In der hier vorgestellten Form ist die Ausdrucksfähigkeit des Kalküls (mit dem dazugehörigen Ableitungssystem) beschränkt. Wir können z.B. nicht beliebige temporale Eigenschaften für Objekte formulieren, da wir nur die positionalen Operatoren zur Verfügung haben, die in ihrer Mächtigkeit in etwa mit dem `next`–Operator temporaler Logiken vergleichbar sind. Folglich können wir z.B. auch nicht Lebendigkeitsbedingungen ausdrücken. Wir müssen uns gegenwärtig auf einzelne Zustandsübergänge oder endliche Folgen davon beschränken. Auf die Möglichkeit der späteren Erweiterungen, insbesondere um temporale Operatoren, haben wir jedoch schon bei der Basisversion dieses Kalküls geachtet.

Verifikation von Eigenschaften. Nachdem wir nun den Verifikationskalkül kurz vorgestellt haben, können wir auch einen Eindruck davon geben, wie das Beweisunterstützungssystem in die Entwicklungsumgebung eingebettet ist. Dies kann auf einfache Weise von der beabsichtigten Benutzung des Beweisunterstützungssystems geschlossen werden. Prinzipiell stellen wir uns den folgenden Arbeitsablauf vor, den wir auch in Abb. 5 dargestellt haben:

1. Die Transformation einer TROLL *light*–Spezifikation liefert eine entsprechende Template–Signatur zusammen mit einer Menge von Formeln, die eine logische Beschreibung der spezifizierten Objekte mit Mitteln des Verifikationskalküls darstellt. Die Ergebnisse solcher Trans-

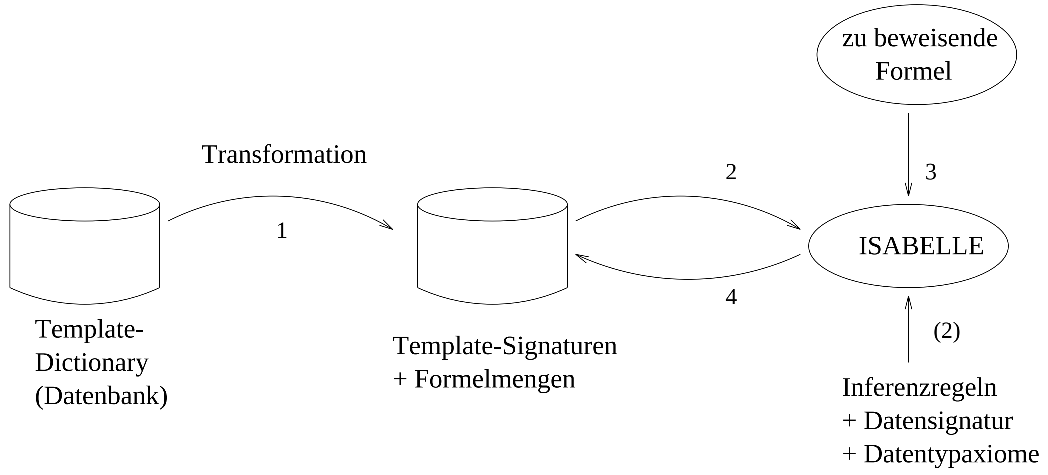

Abbildung 5: Arbeiten mit dem Beweisunterstützungssystem.

formationen werden wiederum in einer Datenbank gespeichert.

2. Liegt das Ergebnis der Transformation einer TROLL *light*-Spezifikation vor, kann der generische Theorembeweiser Isabelle [Pau90] aktiviert werden. Um Isabelle als Beweiser für den Verifikationskalkül einsetzen zu können, müssen zunächst die Syntaxdefinitionen und Inferenzregeln des Verifikationskalküls und eine vorgebene Datentypsignatur mit zugehörigen Axiomen geladen werden. Als nächstes müssen die Signatur und die Formeln geladen werden, die aus der Transformation in Schritt 1. erhalten wurden.

3. Nun können wir eine zu beweisende Eigenschaft als Formel des Verifikationskalküls formulieren und den Beweiser darauf anwenden. Der Beweis kann interaktiv gesteuert werden. Wir können den Beweis Schritt für Schritt durchführen, indem wir einzelne Inferenzregeln anwenden. Wir können aber auch Isabelle durch Wahl geeigneter Beweistaktiken veranlassen, Teile des Beweises selbständig zu führen.

4. Falls der Beweis erfolgreich durchgeführt wurde, kann die bewiesene Formel in der Datenbank abgespeichert werden. Damit kann diese Formel später wiederverwendet werden, um weitere Eigenschaften derselben Objekte zu beweisen.

Der hier vereinfacht dargestellte Ablauf kann als Grundlage für ein komfortables Beweisunterstützungssystem verwendet werden. Der realisierte Prototyp umfaßt nur die elementaren Funktionalitäten. Eine Reihe von Verbesserungen, z.B. hinsichtlich der Darstellung von Beweisen, wird notwendig sein, damit nicht nur wenige Spezialisten damit arbeiten können werden.

6 Schlußbemerkungen

Die Sprache TROLL *light* wurde in mehreren, verschieden großen Fallstudien eingesetzt [Ehr93], z.B. für die Spezifikation einer Fertigungszelle [HV94]. Hieraus läßt sich ableiten, daß unsere Sprache nicht nur für Informationssysteme geeignet ist. Die entworfene Entwicklungsumgebung ist in ihren Grundzügen prototypisch implementiert worden. Die näher vorgestellten Werkzeuge, das Animationssystem für TROLL *light*–Objektgesellschaften und das Beweisunterstützungssystem für die Verifikation von Objekteigenschaften, wurden bereits an verschiedenen kleineren Fallstudien gestestet.

Auch wenn das Ziel unserer Arbeit kein marktreifes Produkt war, zeigen die erzielten Ergebnisse mit den realisierten Prototypen bereits, daß eine entsprechende Umsetzung für den realen Einsatz prinzipiell schon möglich und sinnvoll ist. Erforderlich sind dann neben einer effizienten Implementierung noch einige Verbesserungen, z.B. eine komfortablere Benutzerschnittstelle für das Beweisunterstützungssystem.

Die Ergebnisse und Erfahrungen dieses Projektes haben uns davon überzeugt, daß der objektorientierte Entwurf auch für sicherheitskritische Anwendungen geeignet ist. Die theoretische Fundierung und die Bereitstellung von Validations- und Verifikationswerkzeugen wird dabei in Zukunft eine immer größere Rolle spielen, da die immer stärker werdenden Anforderungen an Zuverlässigkeit und Korrektheit eines Informationssystems eine Verlagerung des Enwicklungsaufwandes zugunsten der frühen Entwicklungsphasen erforderlich machen.

Das objektorientierte Paradigma erlaubt eine natürliche Modellierung des zu betrachtenden Weltausschnittes und kann so dazu beitragen, komplexe Strukturen und Zusammenhänge von Objekten in Informationssystemen besser in den Griff zu bekommen. Durch Animation von spezifizierten Objektgesellschaften läßt sich unmittelbar die Adäquatheit der Spezifikation prüfen. Erforderliche formale Nachweise von Objekteigenschaften lassen sich ebenfalls frühzeitig durchführen, da — wie wir gezeigt haben — geeignete Verifikationsansätze auch im objektorientierten Bereich zur Verfügung stehen.

In diesem Beitrag haben wir nur einen kurzen Überblick unserer Arbeit geben können und wollen. Ausführlichere Darstellungen der hier vorgestellten Aspekte und weiterer, hier nicht berücksichtigter Aspekte finden sich in der angegebenen Literatur.

Literatur

[BD93] A.-P. Bröhl und W. Dröschel. *Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden.* Oldenbourg-Verlag, München, 1993.

[BJ93] M. Broy und S. Jähnichen. Das BMFT–Verbundprojekt „Korrekte Software (KORSO)“. *Informatik — Forschung und Entwicklung*, 8(3):152–167, 1993.

[CDG+93] S. Conrad, G. Denker, M. Gogolla, R. Herzig, N. Vlachantonis und H.-D. Ehrich. Zur Entwicklung zuverlässiger Informationssysteme in KorSo. In H. Reichel, Herausgeber, *Informatik — Wirtschaft — Gesellschaft, Proc. 23. GI-Jahrestagung (GI'93)*, S. 464–469. Springer, Informatik aktuell, 1993.

[CGH92] S. Conrad, M. Gogolla und R. Herzig. TROLL *light*: A Core Language for Specifying Objects. Informatik-Bericht 92–02, TU Braunschweig, 1992.

[Con94] S. Conrad. On Certification of Specifications for TROLL *light* Objects. In H. Ehrig und F. Orejas, Herausgeber, *Proc. 9th Workshop on Abstract Data Types – 4th Compass Workshop (ADT'92)*, S. 158–172. Springer, LNCS 785, 1994.

[CRB93] R. Croshere, R. van de Riet und A. Blom. An Animation Facility to Simulate an Information and Communication System. In C. Rolland, F. Bodart und C. Cauvet, Herausgeber, *Advanced Information Systems Engineering, Proc. 5th CAiSE'93*, S. 547–568. Springer, Berlin, LNCS 685, 1993.

[DHS+91] S. Dewal, H. Hormann, L. Schöppe, U. Kelter, D. Platz und M. Roschewski. Bewertung von Objektmanagementsystemen für Software-Entwicklungsumgebungen. In H.-J. Appelrath, Herausgeber, *Proc. Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'91)*, S. 404–411. Springer, Berlin, Informatik-Fachberichte 270, 1991.

[Ehr93] H.-D. Ehrich, Herausgeber. *Beiträge zu* KorSo*– und TROLL light– Fallstudien.* Technische Universität Braunschweig, Informatik-Bericht, 93–11, 1993.

[GCH93] M. Gogolla, S. Conrad und R. Herzig. Sketching Concepts and Computational Model of TROLL *light*. In A. Miola, Herausgeber, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO'93)*, S. 17–32. Springer, Berlin, LNCS 722, 1993.

[GHC+94] M. Gogolla, R. Herzig, S. Conrad, G. Denker und N. Vlachantonis. Integrating the ER Approach in an OO Environment. In R. Elmasri, V. Kouramajian und B. Thalheim, Herausgeber, *Proc. 12th Int. Conf. on the Entity-Relationship Approach (ER'93)*, S. 382–395. Springer, Berlin, LNCS, 1994.

[HCG94] R. Herzig, S. Conrad und M. Gogolla. Compositional Description of Object Communities with TROLL *light*. In C. Chrisment, Herausgeber, *Proc. Basque Int. Workshop on Information Technology (BIWIT'94)*, S. 183–194. Cépaduès-Éditions, Toulouse, 1994.

[HV94] R. Herzig und N. Vlachantonis. TROLL light — Specification with a Language for the Conceptual Modelling of Information Systems. In C. Lewerentz und T. Lindner, Herausgeber, *Case Study "Production Cell": A Comparative Study in Formal Specification and Verification*, S. 231–239. FZI-Publication 1/94, Forschungszentrum Informatik, Karlsruhe (Germany), 1994.

[KEK+94] K. Kurbel, S. Eicker, F. Kersten, Th. Schneider und A. Teubner. I-CASE bei der Entwicklung eines großen Informationssystems: eine Information-Engineering-Fallstudie. *Wirtschaftsinformatik*, 36(2):130–144, 1994.

[KM93] A. Kemper und G. Moerkotte. Basiskonzepte objektorientierter Datenbanksysteme. *Informatik Spektrum*, 16(2):69–80, 1993.

[Krü93] F. Krückeberg. Zertifizierung von Software. *Wirtschaftsinformatik*, 35(2):183–186, 1993.

[LL93] V. Lalioti und P. Loucopoulos. Visualisation for Validation. In C. Rolland, F. Bodart und C. Cauvet, Herausgeber, *Advanced Information Systems Engineering, Proc. 5th CAiSE'93*, S. 143–164. Springer, Berlin, LNCS 685, 1993.

[LLOW91] C. Lamb, G. Landis, J. Orenstein und D. Weinreib. The ObjectStore Database System. *Communications of the ACM*, 34(10):50–63, 1991.

[Ous93] J.K. Ousterhout. *Tcl and the Tk Toolkit*. Computer Science Division, University of California at Berkeley, 1993.

[Pau90] L.C. Paulson. Isabelle: The Next 700 Theorem Provers. In P. Odifreddi, Herausgeber, *Logic and Computer Science*, S. 361–385. Academic Press, 1990.

[Pis93] P. Pistor. Objektorientierung in SQL3: Stand und Entwicklungstendenzen. *Informatik Spektrum*, 16(2):89–94, 1993.

[Rom93] H.D. Rombach. Software-Qualität und Qualitätssicherung. *Informatik Spektrum*, 16(5):267–272, 1993.

[VH93] N. Vlachantonis und P. Hartel. An Approach towards the Conceptual Modelling of the Tool Integration Aspect. In J.-C. Rault, Herausgeber, *Proc. 6th Int. Conf. Software Engineering & Its Applications (SEA'93)*, S. 543–553. EC2, Nanterre, France, 1993.

[VHG+93] N. Vlachantonis, R. Herzig, M. Gogolla, G. Denker, S. Conrad und H.-D. Ehrich. Towards Reliable Information Systems: The KORSO Approach. In C. Rolland, F. Bodart und C. Cauvet, Herausgeber, *Proc. 5th Int. Conf. on Advanced Information Systems Engineering (CAiSE'93)*, S. 463–482. Springer, Berlin, LNCS 685, 1993.