Unterstützung des korrektheitszentrierten Entwurfs von Informationssystemen

Stefan Conrad

Universität Magdeburg, Institut für Technische Informationssysteme,
Postfach 4120, D-39016 Magdeburg, Germany

E-mail: conrad@iti.cs.uni-magdeburg.de

Zusammenfassung Im vorliegenden Beitrag wird ein Ansatz zum Einsatz von Verifikationswerkzeugen in der Entwurfsphase für Informationssysteme vorgestellt. Basierend auf einer einfachen Beschreibungssprache für Struktur und Verhalten von Objekten wird ein geigneter Kalkül präsentiert, mit dem das Beweisen von Objekteigenschaften mittels Herleitung aus der Spezifikation möglich wird. Hiermit ist eine frühe formale Überprüfung des Entwurfes auf wichtige Eigenschaften möglich.

1 Einleitung

Für die Entwicklung von Informationssystemen spielt der konzeptionelle Entwurf eine zentrale Rolle. Fehler, die in dieser Phase gemacht werden, haben weitreichende und damit vor allem kostenintensive Folgen. Kommt zu dem Entwurf der Struktur des betrachteten Weltausschnittes auch noch die Beschreibung des Verhaltens hinzu, erreicht der Entwurf schnell eine Komplexität, die die Entdeckung von Modellierungsfehlern und Beschreibungsfehlern erheblich erschwert. Fehler in der Verhaltensbeschreibung können insbesondere in sicherheitskritischen Systemen hohe Kosten verursachen oder sogar Menschenleben gefährden.

Daher haben wir im Rahmen des BMFT–Verbundprojektes KorSo die korrektheitszentrierte Entwicklung von Informationssystemen untersucht und dabei drei Schwerpunkte gesetzt: die Konzeption einer geeigneten Enwicklungsumgebung, Animation und Verifikation. Als Grundlage unserer Arbeit verwenden wir eine vereinfachte Sprache zur Beschreibung von Objekten (TROLL *light* [CGH92]), die ein Dialekt der Sprache TROLL [JSHS91] ist.

Diese Bemühungen ordnen sich gut in die allgemeinen Bestrebungen ein, formale Methoden schon in den frühen Phasen der Entwicklung einzusetzen. Hiermit kann der immer stärker werdenden Forderung nach Zertifikation von Softwaresystemen Rechnung getragen werden (vgl. [Krü93, Rom93]). Aus unserer Sicht kann sich an die Erstellung einer formalen Spezifikation, ihrer Validierung und Verifikation eine Transformation auf ein bestehendes objektorientiertes Datenbanksystem (siehe [Heu92]) anschließen.

[0] Diese Arbeit wurde gefördert durch das BMFT im Projekt KorSo (= KORrekte SOftware, Förd. Nr. IT 01 IS 203 D) und durch die ESPRIT Basic Research Working Groups IS-CORE (No. 6071) and ModelAge (No. 8319).

In diesem Beitrag soll der Gesichtspunkt der Verifikation von Objektspezifikationen im Vordergrund stehen, d.h. die Bereitstellung eines geeigneten Kalküls, der in Form eines Beweisunterstützungssystems als Werkzeug dem Benutzer zur Verfügung gestellt wird. Der Aspekt der Animation von Objektspezifikationen ist z.B. in [HFG94, HG94] ausführlicher dargestellt.

Im folgenden werden wir zunächst kurz die Sprache TROLL *light* vorstellen. Anschließend geben wir eine informelle Einführung in den zugehörigen Basiskalkül, gehen kurz auf Beweisregeln und das Führen von Beweisen ein. Danach diskutieren wir die Funktionalität des Beweisunterstützungssystems.

2 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 organisiert, die sich aus Unterobjektbeziehungen ergeben. Eine Unterobjektbeziehung 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 Parametern versehen sein.
- 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.

```
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;
```

Abb. 1. Objektbeschreibung für Autoren.

Die so beschriebenen `Author`–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 `Author`–Objektes eintreten können, sind sein Geburtsereignis, eine Namensänderung, das Abspeichern von Verkaufszahlen und schließlich ein Todesereignis. Im `VALUATION`–Abschnitt ist beschrieben, welche Auswirkungen das Eintreten eines Ereignisses auf Attribute hat. Der Verhaltensteil legt die zulässigen Lebensläufe von `Author`–Objekte fest.

```
TEMPLATE AuthorContainer
  DATA TYPES   String, Int;
  TEMPLATES    Author;
  SUBOBJECTS   Authors(No:int):author;
  ATTRIBUTES   DERIVED  NumberOfAuthors:int;
  EVENTS       BIRTH  create;
                 addAuthor(No:int, Name:string, DateOfBirth:string);
                 removeAuthor(No:int);
               DEATH  destroy;
  DERIVATION   NumberOfAuthors=CNT(Authors);
  CONSTRAINTS  NumberOfAuthors<10000;
  INTERACTION  addAuthor(N,S,D) >> Authors(N).create(S,D);
               removeAuthor(N) >> Authors(N).destroy;
END TEMPLATE;
```

Abb. 2. Spezifikation eines zusammengesetzten Objektes.

In TROLL *light* können Objekte hierarchisch zusammengesetzt werden. Der dafür zur Verfügung gestellte Mechanismus sind Unterobjektbeziehungen. In Abb. 2 werden Objekte beschrieben, die `Author`–Objekte als Unterobjekte haben (im `SUBOBJECTS`–Abschnitt beschrieben). Im `INTERACTION`–Teil wird die Kommunikation zwischen dem „Oberobjekt" und seinen „Unterobjekten" festgelegt.

3 Beweisunterstützung

Testverfahren, wie z.B. die Animation einer Spezifikation, können viele Entwurfs– und Beschreibungsfehler entdecken. Allerdings reichen diese Verfahren i.allg. nicht für den Nachweis aus, daß bestimmte Eigenschaften gelten. Insbesondere in Hinblick auf das dynamische Verhalten der beschriebenen Objekte bleibt immer noch eine Unsicherheit darüber übrig, ob die Spezifikation korrekt ist, da immer nur eine endliche Anzahl von Abläufen getestet werden kann.

Daher halten wir ein Beweisunterstützungssystem für notwendig, das uns helfen soll, Eigenschaften einzelner Objekte, aber auch Eigenschaften einer ganzen Objektgesellschaft formal nachzuweisen. Dies erlaubt uns, Eigenschaften zu garantieren, die in allen möglichen Lebensläufen gelten. Allerdings gehen wir davon aus, daß in der Regel nicht jede Eigenschaft formal bewiesen werden sollte, da der Aufwand oft nicht in einem angemessenen Verhältnis zum Nutzen steht. Aber schon der Nachweis wesentlicher Eigenschaften (insbesondere sicherheitskritischer) kann die Zuverlässigkeit des zu entwickelnden Informationssystems wesentlich erhöhen. Eine Reihe von Anwendungsgebieten verlangt in immer größeren Maße verifizierte Software, vor allem wenn fehlerhafte Software große finanzielle Verluste verursachen kann oder sogar menschliches Leben gefährdet.

Auf einer sehr niedrigen Ebene können bereits automatische Werkzeuge eingesetzt werden, die Konsistenzüberprüfungen vornehmen, Mehrdeutigkeiten aufdecken und die Vollständigkeit einer Spezifikation überprüfen können. Auf einer höheren Ebene ist dann das formale Beweisen von Objekteigenschaften möglich. Hier kann das bereits erwähnte Beweisunterstützungssystem eingesetzt werden. Die nachzuweisenden Eigenschaften werden in einem einfachen Kalkül formuliert und können dann interaktiv bewiesen werden, wobei bis zu einem gewissen Grad Beweisabläufe auch automatisierbar sind.

Um eine schnelle Verfügbarkeit für den Einsatz in Fallstudien zu erreichen, haben die Formeln des Kalküls eine sehr einfache Struktur (sogenannte Gentzen-Klauseln). Dadurch läßt sich leicht ein Beweissystem durch Adaption von Inferenzregeln klassischer Sequenzenkalküle erhalten (vgl. [Gen35, Pae88]). Dies erlaubt eine schnelle prototypische Implementierung mit Hilfe eines generischen Theorembeweisers (z.B. ISABELLE [Pau90]).

3.1 Der Basiskalkül

Im folgenden werden anhand von Beispielen Formeln des Kalküls und Inferenzregeln vorstellen. Anschließend zeigen wir mittels eines kleinen Beispiels das Beweisen von Objekteigenschaften. Eine detaillierte Vorstellung und Diskussion des Kalküls ist in [Con94] zu finden.

Formeln des Kalküls sind im wesentlichen Gentzen-Klauseln über einfache Zustandsaussagen, wobei Zustandaussagen folgender Gestalt seien können:

- $t_1 = t_2$ ist eine Zustandaussage über die Gleichheit zweier Terme,
- $p(t_1, \ldots, t_n)$ ist eine Zustandsaussage, wenn p ein Prädikatsymbol ist und $t_1, \ldots, t_n$ Terme geeigneter Sorte sind,
- $\neg P$ ist eine Zustandsaussage, wenn P eine Zustandsaussage ist,
- $[t_o.t_e]P$ ist eine Zustandsaussage, wenn P eine Zustandsaussage ist, t_o ein Term ist, der ein Objekt bezeichnet, und t_e ein Term ist, der ein zu t_o gehöriges Ereignis beschreibt.

$[t_o.t_e]P$ wird gelesen als „wenn als nächstes das Ereignis t_e im Objekt t_o eintritt, dann gilt anschließend P".

Sind also $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ einfache Zustandsaussagen, dann ist

$$P_1, \ldots, P_n \ \rightarrow \ Q_1, \ldots, Q_m$$

eine Formel, die wie folgt zu lesen ist: sind P_1 und ... und P_n gleichzeitig erfüllt (in einem beliebigen Zustand der Objektgesellschaft), dann ist gleichzeitig auch Q_1 oder ... oder Q_m erfüllt (in demselben Zustand).

Beispiel:
Betrachten wir die in Abb. 1 gegebene Spezifikation von `Author`–Objekten, dann beschreiben die folgenden Formeln Eigenschaften solcher `Author`–Objekte:

$$(1) \rightarrow \ [A.changeName(\text{“J.W. Goethe”})]Name(A, \text{“J.W. Goethe”}),$$

$$(2) \neg\, enable(A.changeName(N)) \ \rightarrow$$
$$[A.storeSoldBooks(Y, C)]\neg\, enable(A.changeName(N)).$$

Formel (1) besagt, daß in jedem Zustand (bzw. zu jedem Zeitpunkt) wahr ist, daß nach dem Eintreten eines Ereignisses *changeName*("J.W. Goethe") für ein `Author`–Objekt A das Attribut $Name$ dieses Objektes den Wert "J.W. Goethe" hat. Dies gilt für jeden Zustand, weil keine einschränkende Vorbedingung auf der linken Seite des $\rightarrow$ gegeben ist. Formel (2) beschreibt, daß, wenn ein Ereignis $changeName(N)$ für ein `Author`–Objekt A nicht erlaubt ist, es auch nach dem Eintreten eines Ereignisses $storeSoldBooks(Y, C)$ für A nicht erlaubt ist.

Formeln des Kalküls dienen also dazu, Objekteigenschaften zu beschreiben. Damit kann eine ganze Objektgesellschaft durch eine Menge solcher Formeln beschrieben werden. Für die Objektspezifikationssprache TROLL *light* haben wir eine entsprechende Übersetzung entwickelt, die zu einer beliebigen TROLL *light* Spezifikation eine Menge von Kalkülformeln als logische Beschreibung der spezifizierten Objektgesellschaft erzeugt (siehe [Con94]).

Ausgehend von einer Menge solcher Formeln können wir nun unter Anwendung von Inferenzregeln weitere Formeln als logische Konsequenzen herleiten. Eine Zusammenstellung elementarer Inferenzregeln für unseren Kalkül ist in Abb. 3 dargestellt. Diese Regeln konnten im wesentlichen von klassischen Sequenzenkalkülen [Gen35] und einigen neueren Ansätzen für Objektlogiken [FM91] übernommen und auf unseren Kalkül angepaßt werden. Für die hier angegebenen Regeln

$$\frac{}{P \rightarrow P}\textbf{(axiom)} \qquad \frac{R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, \neg R}\textbf{(negR)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R'}\textbf{(ext)} \qquad \frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R}{\neg R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}\textbf{(negL)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R \qquad R, P'_1, \ldots, P'_k \rightarrow Q'_1, \ldots, Q'_l}{P'_1, \ldots, P'_k, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, Q'_1, \ldots, Q'_l}\textbf{(cut)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{s(P_1), \ldots, s(P_n) \rightarrow s(Q_1), \ldots, s(Q_m)}\textbf{(sub)} \qquad \frac{occur(E_1) \rightarrow occur(E_2)}{[E_2]P \rightarrow [E_1]P}\textbf{(occ)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{[E]P_1, \ldots, [E]P_n \rightarrow [E]Q_1, \ldots, [E]Q_m}\textbf{(nex)}$$

Abb. 3. Elementare Inferenzregeln.

können wir zeigen, daß sie korrekt sind [Con94]. Vollständigkeit des Beweissystems haben wir nicht angestrebt, weil zu vermuten ist, daß Vollständigkeit aufgrund der Ausdrucksmächtigkeit des Kalküls nicht erreichbar ist.

Eine andere wichtige Eigenschaft unseres Kalküls ist, daß Beweise kompositional sind. Dies bedeutet, daß wir Beweise von Eigenschaften so lokal wie möglich durchführen können (z.B. nur unter Verwendung der sich aus der Spezifikation des betroffenen Objektes ergebenden Eigenschaften). Haben wir beispielsweise eine Eigenschaft von `Author`–Objekten nur unter Verwendung ihrer Spezifikation bewiesen, bleibt dieser Beweis gültig, wenn wir den Kontext ändern, also wenn wir z.B. die Spezifikation einer Bibliothek betrachten, in der wir die Spezifikation der `Author`–Objekte verwenden.

Diese Art der Kompositionalität ist unverzichtbar, damit wir nicht dieselben Eigenschaften bei Kontexterweiterungen immer wieder neu beweisen müssen. Die hier angegebenen Inferenzregeln behandeln im wesentlichen den ausagelogischen Teil unseres Kalküls.

Die Eigenschaften, die den TROLL *light* Objekten inhärent sind, werden durch Axiome beschrieben. Zum Beispiel sind die folgenden Formeln Axiome:

(1) $occur(O.E) \;\rightarrow\; enable(O.E)$
(2) $occur(O.E_1),\; occur(O.E_2) \;\rightarrow\; O.E_1 = O.E_2$
(3) $Att(O, P_1, \ldots, P_n, V_1),\; Att(O, P_1, \ldots, P_n, V_2) \;\rightarrow\; V_1 = V_2$

Axiom (1) beschreibt, daß ein auftretendes Ereignis auch gleichzeitig erlaubt sein muß. Axiom (2) besagt, daß nicht zwei (oder mehr) verschiedene Ereignisse gleichzeitig in demselben Objekt auftreten können. (3) ist ein Axiomenschema und verlangt, daß für ein beliebiges Attribut *Att* der Attributwert zu jedem Zeitpunkt eindeutig sein muß.

In Fortsetzung des durchgängigen Beispiels werden wir den Beweis einer Objekteigenschaft exemplarisch vorführen. Die zu beweisende Eigenschaft ist dabei:

Nach dem Auftreten eines `addAuthor`–Ereignisses in einem `AuthorContainer`–Objekt hat das Attribut `Name` des neuen `Author`–Objektes den Wert, der als Parameter des `addAuthor`–Ereignisses aufgetreten ist:

$$\rightarrow [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)$$

Der Beweis basiert auf den beiden Formeln φ_1 und φ_2, die grundlegende Eigenschaften von `Author`– und `AuthorContainer`–Objekten beschreiben. φ_1 gibt dabei an, daß nach dem Auftreten eines *create*–Ereignisses in einem `Author`–Objekt das Attribut *Name* dieses Objektes den Wert hat, der als Parameter dem *create*–Ereignis mitgegeben wurde. Die Formel φ_2 besagt, daß das Auftreten eines *addAuthor*–Ereignisses in einem `AuthorContainer`–Objekt das gleichzeitige Auftreten eines *create*–Ereignisses in einem `Author`–Objekt zur Folge hat, das ein Unterobjekt des `AuthorContainer`–Objektes ist. Welches Unterobjekt betroffen ist, wird durch den aktuellen Wert des entsprechenden Parameters des *addAuthor*–Ereignisses festgelegt. Diese Eigenschaften können direkt in den Objektspezifikationen (siehe Abb. 1 und 2) wiedergefunden werden. Der Beweis sieht nun wie folgt aus:

$$\cfrac{\cfrac{\varphi_1}{\varphi_3}\,\textbf{(sub)} \qquad \cfrac{\varphi_2}{\varphi_4}\,\textbf{(occ)}}{\varphi_5}\,\textbf{(cut)}$$

mit

$$\begin{aligned}
\varphi_1 &: \rightarrow\ [A.create(S, S')]Name(A,\ S)\\
\varphi_2 &: occur(AC.addAuthor(Nr, S, S'))\\
&\qquad\quad \rightarrow\ occur(Authors(AC, Nr).create(S, S'))\\
\varphi_3 &: \rightarrow\ [Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\ S)\\
\varphi_4 &: [Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\ S)\\
&\qquad\quad \rightarrow\ [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)\\
\varphi_5 &: \rightarrow\ [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)
\end{aligned}$$

Beginnend mit φ_1 liefert die Substitution von A durch $Authors(AC, Nr)$ die Formel φ_3. Formel φ_4 wird durch Anwendung der Regel **(occ)** auf φ_2 erhalten. Schließlich können wir φ_3 und φ_4 unter Anwendung der Regel **(cut)** zusammenführen. Das Ergebnis ist φ_5, die Eigenschaft, die wir beweisen wollten.

3.2 Systemfunktionalität

Nach der kurzen Einführung in den Kalkül geben wir nun einen Überblick über die beabsichtigte Funktionalität des Beweisunterstützungssystems. Prinzipiell denken wir dabei an die folgenden Arbeitsschritte (siehe auch Abb. 4):

1. Die (automatische) Übersetzung einer TROLL *light* Spezifikation, die im Template–Dictionary abgespeichert ist, liefert eine zugehörige Template–Signatur zusammen mit einer Menge von Formeln, die die logische Beschreibung der spezifizierten Objekte in unserem Kalkül darstellen. Das Ergebnis dieser Transformation wird wiederum in einer Datenbank gespeichert.

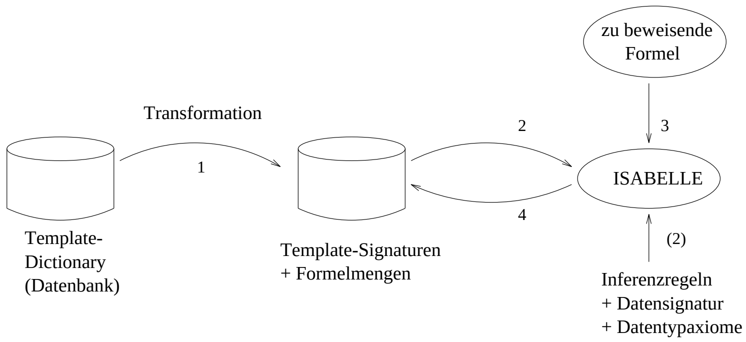

Abb. 4. Das Beweisunterstützungssystem.

2. Wenn die Transformation einer TROLL *light* Spezifikation abgeschlossen ist, kann der generische Theorembeweiser ISABELLE [Pau90] aktiviert werden. Um diesen Theorembeweiser für unseren Kalkül verwenden zu können, müssen die entsprechenden Syntaxdefinitionen und Inferenzregeln des Kalküls in den Beweiser geladen werden. Dann können die Template–Signatur und die Formeln aus dem vorherigen Schritt hinzugefügt werden.
3. Nun können wir Eigenschaften als Formeln des Kalküls formulieren, die dann bewiesen werden sollen. Auf jede solche Formel können wir dann den Beweiser anwenden. Der Beweis kann interaktiv gesteuert werden, d.h. wir führen den Beweis Schritt für Schritt durch Anwenden einzelner Inferenzregeln aus, oder wir lassen Teile des Beweises automatisch erzeugen, indem wir eine geeignete vordefinierte Beweistaktik auswählen.
4. Wenn der Beweis einer Eigenschaft erfolgreich war, kann die entsprechende Formel in die Datenbank eingefügt werden. Dadurch ist es möglich, diese Formel für andere Beweise einzusetzen (sozusagen als Lemma).

Natürlich ist dies nur ein erster, sehr grober Ansatz für den Einsatz eines Beweisunterstützungssystems. Jedoch zeigt dies bereits die grundlegenden Funktionalitäten: Transformation in Formeln des Kalküls, interaktive Verwendung eines (generischen) Theorembeweisers und Abspeichern bewiesener Formeln. Um ein komfortables Beweisunterstützungssystem zu erhalten, müssen weitere, mächtigere Funktionalitäten integriert werden. Dabei stellen wir uns insbesondere die folgenden vor:

Protokollieren der Beweisschritte: Damit kann eine Menge Arbeit erspart werden, wenn nach kleinen Spezifikationsänderungen Eigenschaften nochmal bewiesen werden müssen, da die meisten Beweisschritte sich erfahrungsgemäß nicht ändern.

Entwicklung adäquater Beweistaktiken: Anstelle der schrittweisen Durchführung von Beweisen erlauben es Beweistaktiken, Teile des Beweises automatisch durch den Theorembeweiser finden zu lassen. Hierzu ist es allerdings erforderlich, in größeren Fallstudien typische Beweissequenzen und –situationen herauszufinden und für diese dann entsprechende Taktiken zu entwickeln.

Komfortablere Benutzungsschnittstelle: Die aktuell verwendete Version von ISABELLE bietet nur eine sehr „unfreundliche" Benutzungsschnittstelle an. Hier besteht Bedarf an umfangreichen Verbesserungen, um eine höhere Akzeptanz zu erreichen. Insbesondere eine graphische Repräsentation der Beweise wäre sehr hilfreich.

Zunächst ist nur eine erste, prototypische Version des Beweisunterstützungssystems realisiert worden. Sie soll uns helfen, Erfahrungen beim Beweisen von Objekteigenschaften zu sammeln. Erste Beispiele konnten bereits erfolgreich durchgeführt werden. Es sind jedoch mehr und vor allem größere Fallstudien notwendig, bevor die Adäquatheit des Kalküls und die Konzeption des Beweisunterstützungssystems hinreichend beurteilt werden können. Danach könnte dann auf der Basis der erzielten Ergebnisse eine Überarbeitung des Kalküls und des Beweisunterstützungssystems durchgeführt werden.

4 Abschließende Bemerkungen

Das vorgestellte Beweisunterstützungssystem ist im Rahmen des KORSO–Projektes prototypisch realisiert worden und anhand einer Reihe von Beispielen ausgetestet worden. Eine Vervollständigung des Beweisunterstützungssystems, umfangreichere Fallstudien und eine daraus abzuleitende Integration der Verifikation auf Spezifikationsebene in eine geeignete Entwicklungsmethodik (z.B. OMT [RBP+91] oder das V-Modell [BD93]) bieten ausreichende und vielversprechende Ansatzpunkte für weitere Projekte.

Zusammen mit dem hier nicht näher behandelten, aber ebenfalls als Prototyp vorliegendem Animationssystem für TROLL *light*–Spezifikationen sind bereits wichtige Werkzeuge für eine integrierte Entwicklungsumgebung vorhanden.

Die Beschränkung von TROLL *light* in der Anzahl der angebotenen Konzepte war hilfreich, um die prinzipiellen Probleme zu identifizieren und geeignete Lösungen zu entwickeln. Damit bietet sich nun eine Erweiterung auf eine Sprache wie TROLL an, die eine größere Zahl von Konzepten zur Modellierung anbietet und es damit dem Entwerfer ermöglicht, möglichst wirklichkeitstreu modellieren zu können. Je mehr Konzepte eine Spezifikationssprache zur Modellierung anbietet, um so mehr stellt sich die Frage nach einer adäquaten Umsetzung all dieser Konzepte auf mögliche Zieldatenbanksysteme. Gegenwärtig gibt es hier noch eine gewisse Diskrepanz zwischen Spezifikationssprachen und existierenden (objektorientierten) Datenbanksystemen. Diese gilt es in der nächsten Zeit abzubauen, um so einen durchgängigen Einsatz formaler Mothoden bei der Entwicklung von Informationssystemen zu ermöglichen.

Danksagung: Besonderer Dank gilt meinen Kollegen G. Denker, H.-D. Ehrich, M. Gogolla, R. Herzig und N. Vlachantonis, mit denen ich in Braunschweig im KORSO–Projekt zusammengearbeitet habe. Darüber hinaus haben Partner aus dem KORSO–Projekt als auch aus den Esprit BRA Working Groups COMPASS und ISCORE in Diskussionen Beiträge zu unserer Arbeit geleistet.

Literatur

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

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

[Con94] S. Conrad. *Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte.* Fortschritt-Berichte Reihe 10, Nr. 295. VDI-Verlag, Düsseldorf, 1994.

[FM91] J. Fiadeiro and T. Maibaum. Towards Object Calculi. In G. Saake and A. Sernadas, editors, *Information Systems — Correctness and Reusability, Workshop IS-CORE '91, ESPRIT BRA WG 3023, London*, pages 129–178. Informatik-Bericht 91–03, Technische Universität Braunschweig, 1991.

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

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 1935.

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

[Heu92] A. Heuer. *Objektorientierte Datenbanken — Konzepte, Modelle, Systeme.* Addison Wesley, 1992.

[HFG94] R. Herzig, H. Fischer, and M. Gogolla. Zur Gestaltung der Benutzungsschnittstelle bei der Animation von Objektspezifikationen. In L. Wegner, editor, *Proc. GI-Workshop "Benutzungsschnittstellen für Datenbanken", Kassel, 17.-18.3.94*, pages 43–45. GI-Datenbankrundbrief 13, Mai 1994.

[HG94] R. Herzig and M. Gogolla. An Animator for the Object Specification Language TROLL *light*. In V.S. Alagar and R. Missaoui, editors, *Proc. Colloquium on Object Orientation in Databases and Software Engineering (COODBSE'94)*, pages 4–17. Université du Quebéc à Montréal, 1994.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91-04, TU Braunschweig, 1991.

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

[Pae88] B. Paech. Gentzen-Systems for Propositional Temporal Logics. In E. Börger, H. Kleine Büning, and M.M. Richter, editors, *2nd Workshop on Computer Science Logic*, pages 240–253. Springer, LNCS 385, 1988.

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

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, 1991.

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