Zur Gestaltung der Benutzungsschnittstelle bei der Animation von Objektspezifikationen*

Rudolf Herzig, Heiko Fischer, Martin Gogolla
TU Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D-38023 Braunschweig
e-mail: herzig@idb.cs.tu-bs.de

1 Kontext

Im Rahmen des BMFT-Verbundprojekts KORSO [BJ93] beschäftigt sich die Braunschweiger Gruppe mit Aspekten der Entwicklung zuverlässiger Informationssysteme. Der Schwerpunkt liegt dabei auf Fragen der Spezifikation, Analyse und Validierung konzeptioneller Schemata. Als Spezifikationssprache wurde zu Beginn des Projekts die Objektbeschreibungssprache TROLL *light* [CGH92, HCG94], ein Dialekt der Sprache TROLL [JSHS91], bestimmt.

Die Sprache TROLL *light* unterstützt die integrierte Beschreibung statischer und dynamischer Eigenschaften konzeptioneller Objekte (s. Anhang A bezüglich einer Beispielspezifikation). Als Modell einer TROLL *light*-Spezifikation ergibt sich ein Zustandsübergangssystem. Der Zustand eines solchen Systems setzt sich aus den Attribut- und Verhaltenszuständen einzelner Objekte zusammen, welche in einer Objekthierarchie angeordnet sind. Zustandsübergänge werden durch das Eintreten einer endlichen, bezüglich Synchronisation abgeschlossenen Menge von Ereignissen ausgelöst [GCH93].

Die hinter einem konzeptionellen Schema stehende Modellbildung muß mit den informalen Vorstellungen der Auftraggeber bzw. späteren Anwender eines zu entwickelnden Systems abgeglichen werden. Formale Spezifikationen bergen dabei die Gefahr, daß sie vom Anwender entweder gar nicht oder nur unzureichend verstanden werden. Eine wichtige Aufgabe der Validierung konzeptioneller Schemata besteht deshalb in einer geeigneten Vermittlung der Modellbildung. Mögliche Vorgehensweisen bestehen in der Verwendung graphischer Darstellungsmittel, der Rückübersetzung in natürlichsprachliche Beschreibungen [Dal92] oder der Diskussion möglicher Schlußfolgerungen aus einem konzeptionellen Schema [Coh84]. Ein Weg, der in besonderer Weise dazu geeignet ist, eine Modellbildung zu verdeutlichen, besteht im *experimentellen Prototyping* [Ala84, Flo84] wichtiger Schemabestandteile.

Hinsichtlich des letzten Punktes ist zu beachten, daß nicht jede Spezifikation direkt ausführbar sein muß. Unter Umständen kann aber durch eine Transformation in eine spezielle Prototypingsprache eine Ausführbarkeit zumindest in Teilen realisiert werden. Das Testen funktionaler Spezifikationen erfolgt in der Regel durch die Vorgabe konkreter oder symbolischer Eingabewerte [Kne89]. Bei der Spezifikation nichtdeterministischer Systeme, insbesondere bei Verhaltensmodellierungen, kann das Auftreten bestimmter Ereignisse unter Umständen durch die Vorgabe von Auftretenswahrscheinlichkeiten simuliert werden. Überläßt man dem Anwender die Vorgabe von Ereignissen zum Austesten bestimmter Szenarios, spricht man auch von *Animation* [CRB93, DDD93].

*Die vorliegende Arbeit wurde vom Bundesministerium für Forschung und Technologie unter der Förderungsnummer 01 IS 203 D (KorSo = Korrekte Software) unterstützt.

Zu TROLL *light* wurde eine Entwicklungsumgebung konzipiert, deren wesentliche, bisher implementierte Bestandteile ein Parser zur syntaktischen Analyse, eine Struktur zur persistenten Ablage, ein Beweisunterstützungssystem zum Nachweis spezieller Eigenschaften und ein Animationssystem zur Validierung von Objektspezifikationen sind [CDG+93, VHG+93]. Wir diskutieren im folgenden Abschnitt einige Anforderungen, die an ein Werkzeug zur Unterstützung der Animation von TROLL *light*-Objektspezifikationen zu stellen sind, und gehen im dritten Abschnitt darauf ein, in welcher Weise diese Anforderungen in der aktuellen Implementierung der Benutzungsschnittstelle des TROLL *light*-Animationssystems [Fis94] realisiert worden sind.

2 Anforderungen an das TROLL *light*-Animationssystem

Ein Werkzeug zur Animation von Objektbeschreibungen dient dazu, Eigenschaften eines spezifizierten Zustandsübergangssystems zu verdeutlichen. Zu diesem Zweck muß das System sowohl Hilfen geben zur Erkundung des Zustands einer Objektgesellschaft als auch zur Durchführung von Zustandsübergängen.

2.1 Exploration des Zustands einer Objektgesellschaft

Eine Objektgesellschaft präsentiert sich im Kontext von TROLL *light* als eine Objekthierarchie. Wie oben erwähnt, besteht der Zustand eines einzelnen Objekts einer solchen Hierarchie aus einem Attributzustand und einem Verhaltenszustand.

Der Attributzustand ist im Allgemeinfall gegeben als ein Tupel einzelner Attributwerte, wobei Attribute prinzipiell beliebig komplex im Sinne von Nichtstandarddatenmodellen [AFS89] sein können. Genauer gesagt sind mehrwertige (set-, list-, bag-, map-wertige), zusammengesetzte (tuplewertige) und alternative (union-wertige) Attribute möglich, deren Wertebereiche sowohl über Standarddatentypen (z. B. int, string) als auch über speziellen benutzerdefinierten Datentypen (z. B. bitmap) konstruiert sein können. Bei der Visualisierung von Attributen ist man daher zum einen mit dem Problem der generischen Darstellung beliebig komplexer Werte konfrontiert, zum anderen kann es sein, daß man darüber hinaus hinsichtlich benutzerdefinierter Datentypen auf spezielle Darstellungsformen zu achten hat (vgl. O_2-LOOKS [D+90]).

Der Verhaltenszustand eines Objekts entspricht im wesentlichen dem Zustand eines Automaten. Die Darstellung eines aktuellen Verhaltenszustands könnte graphisch im Rahmen eines Verhaltensdiagramms erfolgen oder als Marke in textuellen Prozeßbeschreibungen.

Bezüglich der Visualisierung einer gesamten Objektgesellschaft bietet sich die Darstellung in Form eines Baumes an, allerdings kann bei größeren Objektgesellschaften eine solche Darstellung schnell unübersichtlich werden. In jedem Fall sollte es aber möglich sein, von einem Objekt aus Unterobjektbeziehungen zu verfolgen bzw. zu einem Objekt das Oberobjekt zu ermitteln.

Die Erkundung des Zustands einer Objektgesellschaft sollte durch eine Möglichkeit zur Formulierung von Ad-Hoc-Anfragen im Sinne eines SQL-ähnlichen Anfragekalküls ergänzt werden.

2.2 Durchführung von Zustandsübergängen

Zum Einleiten von Zustandsübergängen muß eine endliche Menge von Ereignissen angegeben werden. Dies geschieht durch die Wahl spezifizierter Ereignisgeneratoren. Bei parametrisierten Ereignisgeneratoren müssen zusätzlich aktuelle Parameterwerte angegeben werden. Hierbei kann es sich wiederum um beliebig komplexe Werte handeln, welche auch Objektreferenzen enthalten können.

Eine Ereignismenge muß bezüglich Synchronisation abgeschlossen sein. Nicht abgeschlossene Ereignismengen können unter Umständen vom Animationssystem durch die Auswertung von Interaktionsregeln in Objektspezifikationen automatisch zu abgeschlossenen Ereignismengen ergänzt werden.

Zustandsübergänge müssen in geeigneter Form visualisiert werden. Hierzu gehört die Darstellung von Attribut- und Verhaltenszustandsänderungen und ein Bericht über eingefügte und gelöschte Unterobjekte. Gegebenenfalls sollten diese Darstellungen um Angaben durchgeführter Ereignissynchronisationen ergänzt werden.

Nicht jede Ereignismenge ist mit einem erlaubten Zustandsübergang verbunden. Falls ein Zustandsübergang zum Beispiel wegen verletzter Integritätsbedingungen nicht möglich ist, muß diese Fehlersituation in geeigneter Weise dargestellt werden.

3 Zur Benutzungsschnittstelle des TROLL *light*-Animationssystems

Die Implementierung der Fensteroberfläche des Animators erfolgte mit Hilfe von Tcl/TK [Ous93]. Mit Tcl/TK können in einfacher Weise typische Elemente einer Fensterumgebung wie beispielsweise *buttons*, *listboxes*, *scrollbars*, *entries* u. a. dargestellt und in sogenannten *frames* in fast beliebiger Weise angeordnet werden. Die Definitionen der einzelnen Fensterelemente liegen in textueller Form als Script-Dateien vor, die erst zur Laufzeit des Systems eingelesen und interpretiert werden. Dies hat den Vorteil, daß die Fensterumgebung unabhängig von der eigentlichen Applikation in gewissen Grenzen änderbar ist und somit leicht individuellen Benutzerwünschen angepaßt werden kann.

Ein einzelner Objektzustand wird in einem *Objektfenster* angezeigt (s. Anhang B), wobei die Objektsorte den Titel des Fensters bildet. Um die Herkunft des Objektes in der Objekthierarchie zurückverfolgen zu können, wird am Kopf des Objektfensters der Objektname als Pfadausdruck dargestellt. Darunter befinden sich die spezifizierten Unterobjektslots, Attributslots und Ereignisgeneratoren in jeweils voneinander getrennten Listboxen. Einfache Attributwerte wie Werte der Sorten int oder string werden direkt in Objektfenstern dargestellt, wohingegen die Darstellung komplexer Attribute oder die Verfolgung von Objektreferenzen, z. B. bei Unterobjektslots, durch Mausklick angefordert werden muß. Bei komplexen Werten öffnet sich dabei eine sogenannte *select box*, bei Objektreferenzen entsprechend ein neues Objektfenster.

In einer *select box* (s. Anhang B) werden Werte dargestellt, die in einem zugehörigen Objektfenster aufgrund ihrer Komplexität nicht befriedigend dargestellt werden können. Der Bezug wird über die Felder *Object* und *S-Slot* (für einen Unterobjektslot) und *A-Slot* (für einen Attributslot) hergestellt. Bei einer Kollektion wie einer Menge, einer Liste, einem Bag oder einem Map werden die einzelnen Elemente zeilenweise bereitgestellt, und können, sofern sie ebenfalls komplexwertig sind, weiter ausgewählt werden. Es ist dabei zu beachten, daß parametrisierten Unterobjektslots jeweils Map-Werte zugeordnet werden. Einen Spezialfall stellen Tupelwerte mit mehr als einer komplexen Komponente dar. Wird ein solcher Wert angeklickt, kommt man ebenfalls in eine *select box*, wo man diejenige Tupelkomponente aktivieren muß, für die eine weitere Ausgabe stattfinden soll.

Die gegenwärtige Version des Animationssystems unterstützt noch nicht die Darstellung von Verhaltenszuständen. Außerdem ist es noch nicht möglich, Ad-hoc-Anfragen an Objektgesellschaften zu stellen. Dies könnte in einfacher Weise geschehen, indem man in Objektfenstern Anfrageterme eingibt. Bislang ist aber noch kein Parser für solche Terme geschrieben worden.

Ereignisse werden durch das Anklicken von Ereignisgeneratoren in Objektfenstern ausgewählt. Handelt es sich dabei um einen parameterlosen Ereignisgenerator, wird dieser sofort in eine sogenannte *event collection box* (s. Anhang C) übertragen, in der zur Ausführung anstehende Ereignisse gesammelt werden. Bei parametrisierten Ereignisgeneratoren öffnet sich vorher eine *event parameter specification box* (s. Anhang C). In einer solchen Box werden Ereignisparameter in Form von Termen angegeben. Die Eingabe von Objektreferenzen kann entweder über einen entsprechenden Objektpfad oder über ein Makro, das die Auswahl des gewünschten Objektes aus einer *select box* heraus ermöglicht, erfolgen. Eine syntaktische und logische Überprüfung der Terme findet erst nach Anwahl des *ok buttons* statt. Eingabefehler werden kommentiert; die bisher erfolgten Eingaben können weiter editiert werden. Ansonsten wird das Ereignis direkt an die *event collection box* übergeben. Mit dem *reset button* werden bisher durchgeführte Änderungen und mit dem *cancel button* die Übernahme des gesamten Ereignisses verworfen.

Mit dem *execute button* werden die in der *event collection box* gesammelten Ereignisse dem Ausführungsmodul übergeben, welches nach dem „Alles-oder-Nichts-Prinzips“ einen möglichen Nachfolgezustand bestimmt. Die Ergebnisse einer Zustandsänderung werden in einer *update report box* (s. Anhang C) dokumentiert. Genauer gesagt werden in der gegenwärtigen Version des Animationssystems die Pfadausdrücke der neu geschaffenen oder geänderten Objekte dargestellt sowie die Anzahl der gelöschten Objekte bilanziert. Ein detaillierterer Änderungsbericht, welcher zum Beispiel auf die tatsächlich eingetretenen Ereignisse Bezug nimmt, ist für zukünftige Versionen des Animationssystems geplant. Falls ein Zustandsübergang aufgrund verletzter Integritätsbedingungen nicht möglich ist, wird die Fehlersituation in einer *alert box* dargestellt.

Literatur

[AFS89] S. Abiteboul, P.C. Fischer, and H.J. Schek (eds.), *Nested Relations and Complex Objects in Databases*, Springer, Berlin, LNCS 361, 1989.

[Ala84] M. Alavi, *An Assessment of the Prototyping Approach to Information Systems Development*, Communications of the ACM **27** (1984), 556–563.

[BJ93] M. Broy and S. Jähnichen, *Das BMFT-Verbundprojekt "Korrekte Software (KORSO)"*, Springer, Informatik Forschung & Entwicklung **8** (1993), no. 3, 152–165.

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

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

[Coh84] D. Cohen, *A Forward Inference Engine to Aid in Understanding Specifications*, Proc. Am. Assoc. Artificial Intelligence Conference (AAAI), 1984.

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

[D+90] O. Deux et al., *The Story of O_2*, IEEE Trans. on Knowledge and Data Engineering **2** (1990), no. 1, 91–108.

[Dal92] H. Dalianis, *A Method for Validating a Conceptual Model by Natural Language Discourse Generation*, Proc. 4th Conf. on Advanced Information Systems Engineering (CAiSE) (P. Loucopoulos, ed.), Springer, Berlin, LNCS 593, 1992, pp. 425–444.

[DDD93] E. Dubois, P. Du Bois, and F. Dubru, *Animating Formal Requirements Specifications of Cooperative Information Systems*, Internal report, Inst. d'Informatique, Facultés Universitaires de Namur, 1993.

[Fis94] H. Fischer, *Implementierung der Benutzungsschnittstelle zur Animation von TROLL light–Objektbeschreibungen*, Diplomarbeit, TU Braunschweig, 1994.

[Flo84] C. Floyd, *A Systematic Look at Prototyping*, Approaches to Prototyping, Proc. of the Working Conference on Prototyping, Namur 1983 (R. Budde, K. Kuhlenkamp, L. Mathiassen, and H. Züllighoven, eds.), Springer, Berlin, 1984, pp. 1–19.

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

[HCG94] R. Herzig, S. Conrad, and M. Gogolla, *Compositional Description of Object Communities with TROLL light*, Proc. Basque Int. Workshop on Information Technology (BIWIT): Information Systems Design and Hypermedia (C. Chrisment, ed.), Cépaduès-Éditions, Toulouse, 1994, pp. 183–194.

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

[Kne89] R. Kneuper, *Symbolic Execution as a Tool for Validation of Specifications*, Tech. Report UMCS-89-7-1, Department of Computer Science, University of Manchester, 1989.

[Ous93] J.K. Ousterhout, *Tcl and the TK Toolkit*, Computer Science Division, University of California at Berkeley, 1993, preprint, to be published by Addison-Wesley.

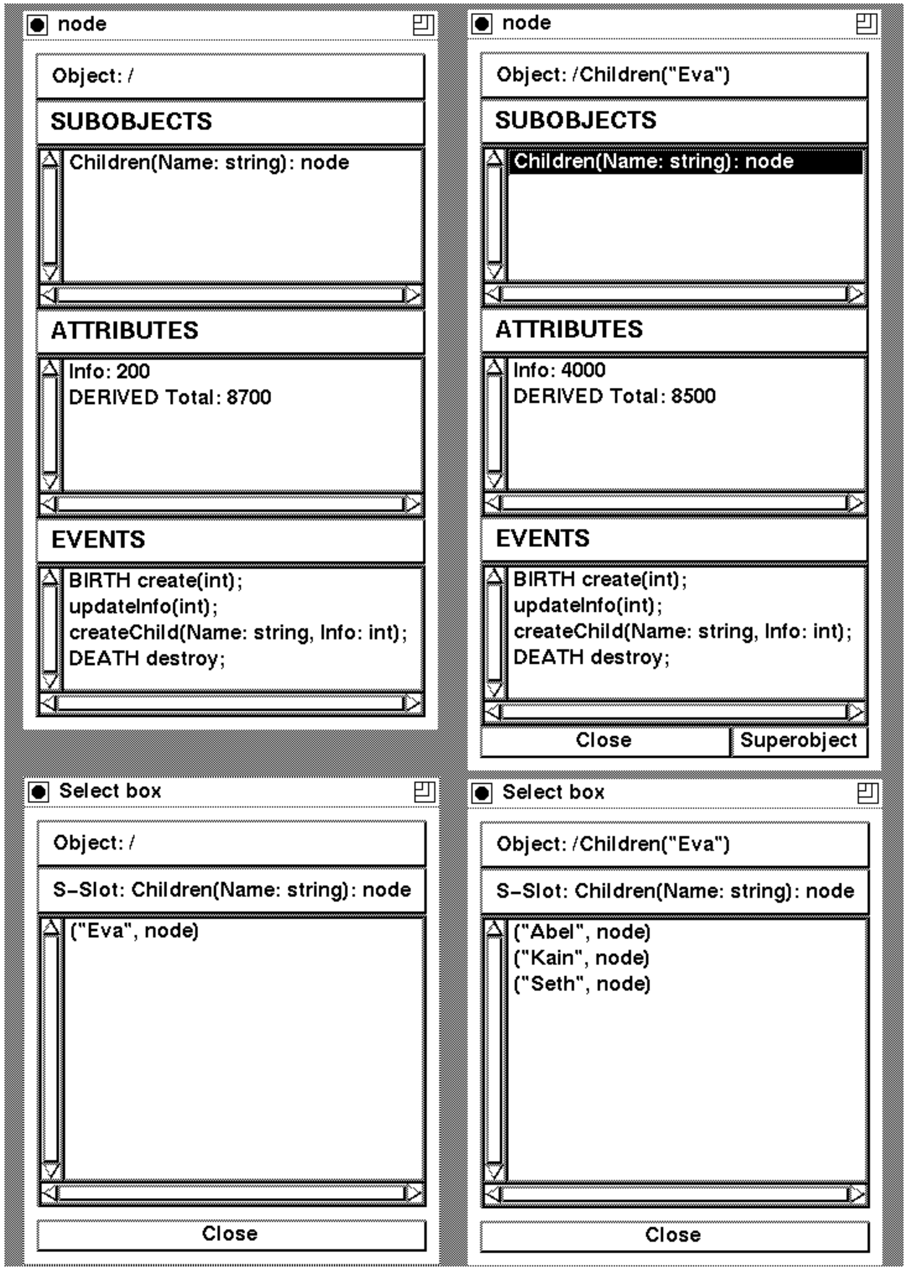
[VHG+93] N. Vlachantonis, R. Herzig, M. Gogolla, G. Denker, S. Conrad, and H.-D. Ehrich, *Towards Reliable Information Systems: The KORSO Approach*, Advanced Information Systems Engineering, Proc. 5th CAiSE'93 (C. Rolland, F. Bodart, and C. Cauvet, eds.), Springer, Berlin, LNCS 685, 1993, pp. 463–482.

A TROLL *light*-Beispielspezifikation

```
TEMPLATE Node
  DATA TYPES    String, Nat;
  SUBOBJECTS    Children(Name:string):node;
  ATTRIBUTES    Info:nat;
                DERIVED Total:nat;
  EVENTS        BIRTH create(nat);
                      updateInfo(nat);
                      createChild(Name:string, Info:nat);
                DEATH destroy;
  CONSTRAINTS   CNT(Children) < 5;
  VALUATION     [create(I)]     Info = I;
                [updateInfo(I)] Info = I;
  DERIVATION    Total = Info + SUM(SELECT Total(x)
                                   FROM x IN Children);
  INTERACTION   createChild(N,I) >> Children(N).create(I);
  BEHAVIOR      PROCESS Node =
                  ( create -> Nodelife );
                PROCESS Nodelife =
                  ( updateInfo -> Nodelife  |
                    createChild -> Nodelife |
                    {CNT(Children)=0} destroy );
END TEMPLATE;
```

Es werden Bäume beschrieben. Jedem Baumknoten ist eine natürliche Zahl als Knoteninformation zugeordnet. Ein Knoten kann maximal vier Unterknoten haben, welche jeweils über eine Zeichenkette identifiziert werden. Das abgeleitete Attribut Total gibt zu jedem Knoten n die Summe der Knoteninformationen des Teilbaums an, dessen Wurzel durch n gegeben ist (*bill of material*-Problem). Als mögliche Ereignisse wurden neben den Geburts- und Todesereignissen die Änderung der Knoteninformation und das Hinzufügen eines Unterknotens spezifiziert. Ein Knoten darf nur gelöscht werden, falls er keine Unterknoten hat.

B Beispielfenster zur Visualisierung von Objektzuständen



C Weitere Fenstertypen