TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

INFORMATIK-BERICHTE

94-06

The KORSO Approach to the Development of Reliable Information Systems[1]

01.10.1991 – 30.06.1994

M. Gogolla, N. Vlachantonis, R. Herzig, G. Denker, S. Conrad, H.-D. Ehrich

TU Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D-38023 Braunschweig, Germany

July 1, 1994

[1]Das diesem Bericht zugrundeliegende Vorhaben wurde mit Mitteln des Bundesministers für Forschung und Technologie (Förderkennzeichen 01 IS 203 D) gefördert. Die Verantwortung für den Inhalt dieser Veröffentlichung liegt bei den Autoren.

Abstract

KORSO was a joint project of fourteen university partners and one industrial partner, sponsored by the German Ministry of Research and Technology (BMFT). The project aimed at the consolidation of existing and the investigation of new methods for the development of correct software. Within this framework the focus of the KORSO group at the Technical University of Braunschweig was on the improvement of techniques for the development of reliable information systems. Our work focused on the specification language TROLL *light* which allows to describe the part of the world to be modeled as a community of concurrently existing and communicating objects. Our specification language comes along with an integrated, open development environment. The task of this environment is to give support for the creation of correct information systems. This report summarizes the results achieved by the Braunschweig KORSO group.

Contents

Chapter 1

Introduction

The information system development process can be roughly split up into two important phases. The aim of the requirements engineering (or specification) phase is to obtain a first formal description of the system in mind. Since this formal description should still abstract from most implementation details it is usually called a conceptual schema. Based on the conceptual schema and by further consideration of nonfunctional requirements a working system is developed in the design engineering (or implementation) phase [Lou92]. We will concentrate in the following on the requirements engineering phase. This phase involves at least four important tasks [DDP93]: (1) Find the users' demands on the system in mind (elicitation), (2) describe a conceptual model of the system in mind (modeling), (3) test whether the conceptual model satisfies formally described quality criteria (analysis), (4) test whether the conceptual model meets the informal user requirements (validation).

As a first formal description of real-world entities we start with the object specification language TROLL *light* [CGH92, GCH93, HCG94], a dialect of OBLOG [SSE87] and TROLL [JSHS91]. TROLL *light* is especially appropriate for information system design because it embodies ideas from data type specification [EM85, EGL89, EM90, Wir90], semantic data models [STW84, HK87, PM88], and process theory [Mil80, Hoa85, Hen88, BW90]. But an attractive language for information system design must be completed by specification tools. Therefore the object description language TROLL *light* comes along with a development environment offering special tools for verification and validation purposes in order to support the user during the design process according to tasks (3) and (4) from above:

- For us, the analysis task (3) consists in verifying properties, i.e., tackling the *formal correctness problem*. But in contrast to program verification where a program, i.e., an implementation, is proved to satisfy its specification, we want to verify properties of objects at the level of specification. This is necessary in order to check whether the specification meets the intended requirements. Verifying such properties directly from the specification can help to avoid misdevelopments based on inadequate specifications.

  In order to support this kind of verification the TROLL *light proof support* *system* solves verification tasks given by the user. It checks whether the desired properties are fulfilled by the specified TROLL *light* object descriptions.

- A specification of a formal conceptual schema must be validated against the informal system requirements in order to meet task (4). This is known as the *informal correctness problem*. One possible way to assure informal correctness of a conceptual schema consists in rapid prototyping which means to construct an experimental version of a system on a quick and cheap basis. Hence a prototype will often illustrate only some important aspects of a required behavior, thereby neglecting others like questions of performance or security. Nevertheless, by observing the behavior of a prototype the clients of a system can better judge the usefulness of a conceptual schema than by reading specification texts only. Typical tools supporting prototyping may be screen painters, report generators, program generators, animation systems, etc.

  The TROLL *light animator* is designed to simulate the behavior of a specified object community. By this the informal view of the real-world fragment to be modeled is validated against the current specification.

In the following we present our system by going into some details concerning the concepts of TROLL *light*, giving comments on the idea of an open development system and presenting the two specific tools for animation and verification. We concentrate on these topics because case studies and application domains are discussed in [Ehr94]. Finally we give some concluding remarks.

Chapter 2

The Language TROLL *light*

TROLL *light* is a language for describing static and dynamic properties of objects. This is achieved by offering language features to specify object structure as well as object behavior. The main advantage of following the object paradigm is the fact that all relevant information concerning one object can be found within one single unit and is not distributed over a variety of locations. As in TROLL object descriptions are called templates in TROLL *light*. Because of their pure descriptive nature templates may roughly be compared with the notion of class found in object-oriented programming languages. In the context of databases, however, classes are also associated with class extensions so that we used a different notion. Templates show the following structure.

`TEMPLATE` name of the template	
`DATA TYPES`	data types used in current template
`TEMPLATES`	other templates used in current template
`SUBOBJECTS`	slots for sub-objects
`ATTRIBUTES`	slots for attributes
`EVENTS`	event generators
`CONSTRAINTS`	restricting conditions on object states
`VALUATION`	effect of event occurrences on attributes
`DERIVATION`	rules for derived attributes
`INTERACTION`	synchronization of events in different objects
`BEHAVIOR`	description of object behavior by a CSP-like process
`END TEMPLATE`	

Roughly speaking, the `DATA TYPES` and `TEMPLATES` sections are the interfaces to other templates, the `SUBOBJECTS`, `ATTRIBUTES`, and `EVENTS` sections constitute the template signature, and in the remaining sections axioms concerning static (`CONSTRAINTS` and `DERIVATION`) and dynamic (`VALUATION`, `INTERACTION`, and `BEHAVIOR`) properties are specified.

To give an example for templates let us assume that we have to describe authors. For every author the name, the date of birth, and the number of books sold by year have to be stored. For the dynamic part we require that an author may change her name only once in her life. An appropriate TROLL *light* specification would hence be:

```
TEMPLATE Author
  DATA TYPES   String, Date, Nat;
  ATTRIBUTES   Name:string;BirthDate:date;SoldBooks(Year:nat):nat;
  EVENTS       BIRTH create(Name:string, BirthDate:date);
                     changeName(NewName:string);
                     storeSoldBooks(Year:nat, Number:nat);
               DEATH destroy;
  VALUATION    [create(N,D)] Name=N, BirthDate=D;
               [changeName(N)] Name=N;
               [storeSoldBooks(Y,NR)] SoldBooks(Y)=NR;
  BEHAVIOR     PROCESS authorLife1 =
                 ( storeSoldBooks -> authorLife1 |
                   changeName -> authorLife2 |
                   destroy -> POSTMORTEM );
               PROCESS authorLife2 =
                 ( storeSoldBooks -> authorLife2 |
                   destroy -> POSTMORTEM );
               ( create -> authorLife1 );
END TEMPLATE;
```

2.1 Data types section

Data types are assumed to be specified with a data type specification language. In the KorSo project we use SPECTRUM [BFG+93] as a reference language, but other proposals like ACT ONE [EFH83], PLUSS [Gau84], Extended ML [ST86], or OBJ3 [GW88] will do their job just as well. With the `DATA TYPES` section the signature of data types is made known to the current template. For example, referring to `Nat` means that the data sort `nat`, operations like `+ : nat x nat -> nat`, and predicates like `<= : nat x nat` are visible in the current template definition. Note that we employ a certain convention concerning the naming of data types, templates and associated sorts. We use names starting with an upper case letter to denote data types and templates whereas the corresponding sort names start with a lower case letter.

2.2 Attributes section

Attributes denote observable properties of objects. They are specified in the `ATTRIBUTE` section of a template by $a\,[\,(s_1,\ldots,s_n)\,]\!:\!d$, where a is an attribute name generator, d is a sort expression determining the range of an attribute, and $s_1,\ldots,s_n$

($n \geq 0$) denote optional parameter sorts (data or object sorts). The sort expression d may be built over both data sorts and object sorts by applying predefined type constructors. We have decided to include the type constructors `tuple`, `set`, `bag`, `list`, and `union`. Of course other choices can be made. Thereby, complex attributes can be specified, e.g., data-valued, object-valued, multi-valued, composite, alternative attributes, and so on. The interpretation of all sort expressions contains the undefined element $\perp$, and therefore all attributes are optional by default. Attribute names may be provided with parameters. For example, by the declaration `SoldBooks(Year:nat):nat` a possibly infinite number of attribute names like `SoldBooks(1994)` is introduced. To underline the meaning of attribute parameters, optional parameter names like `YEAR` in `SoldBooks` may be added. We demand that in a given state of an object only a finite number of attributes takes values different from $\perp$ such that only these attributes have to be stored. Therefore, a parametrized attribute $a(s_1, \ldots, s_n) : s$ can also be viewed as an attribute a : $\texttt{set}(\texttt{tuple}(s_1, \ldots, s_n, s))$, but clearly the formerly inherent functional dependency would have to be described by an explicit constraint. The same works for parametrized sub-object constructors to be discussed later.

2.3 Events section

Incidents possibly appearing in an object's life are modeled by events. Event names are given in the `EVENT` section by e[($s_1, \ldots, s_n$)], where e denotes an event generator and $s_1, \ldots, s_n$ ($n \geq 0$) represent optional parameter sort expressions. Event parameters are used to define the effect of an event occurrence on the current state of an object (see the explanation of the `VALUATION` section below), or they are used to describe the transmission of data during communication (see the explanation of the `INTERACTION` section below). Special events in an object's life cycle are the `BIRTH` and `DEATH` events with which an object's life is started or ended. Several birth or death events may be specified for the same object. A template may have no death event, but we require it to have at least one birth event.

2.4 Valuation section

The effect of events on attribute values is specified in the `VALUATION` section of a template definition by valuation rules, having the following form: [`{`*precond*`}`] `[`*event_descr*`]` *attr_term* `=` *term*. Such a rule says that immediately after an event occurrence belonging to the event term *event_term*, the attribute given by the attribute term *attr_term* has the value determined by destination *term*. The applicability of such a rule may be restricted by a formula *precond*, i.e., the valuation rule is applied only if the precondition is true. The precondition as well as both terms are evaluated in the state before the event occurrence. Thereby all these items may have free variables which however must appear in the event term. An event term

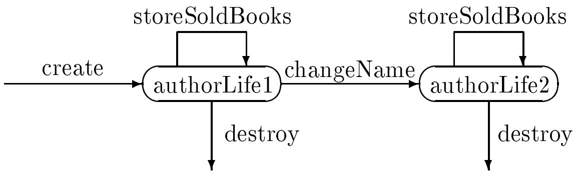

Figure 2.1: Behavior of authors

consists of an event generator and a (possibly empty) list of variables representing the parameters of an event. By a concrete occurrence of such an event, these variables are instantiated with the actual event parameters in such a way that the precondition and the other terms can be evaluated.

To summarize we can say that a valuation is a proposition stating that after the occurrence of a certain event some attributes shall have some special values. The following frame rule is assumed: Attributes which are not caught by a suitable valuation rule of a given event remain unchanged. Before the birth event, all attributes of an object are assumed to be undefined. Thus if the event in question is a birth event, some attributes may remain undefined. An attribute can only be affected by local events, i.e., events which are specified in the same template in which the attribute is specified.

2.5 Behavior section

In the `BEHAVIOR` section the possible event sequences in an object's life are restricted by means of behavior patterns. Roughly speaking, behavior patterns determine a finite non-deterministic machine called o-machine (o for object) in the sequel. However, regarding an object together with its attributes we shall in general get an infinite number of object states and not a finite one. By behavior patterns event sequences, which an object must go through, can be specified as well as event dependent branchings. We have used the keyword `POSTMORTEM` within the behavior patterns to denote that the object vanishes. In Fig. 2.1 the behavior of authors is visualized by the corresponding o-machine representation. Within an object description the behavior section may be missing. In that case event sequences are unrestricted, i.e., it is only required for event sequences to start with a birth event and possibly end with a death event. In addition to the mentioned possibilities, behavior patterns may include preconditions for event occurrences referring to the objects' attributes and sub-objects in order to restrict the possible event sequences further. Thus, edges in the o-machine may not only be labeled by event names but should also mention the appropriate precondition; in our example these preconditions are all `true`.

After dealing with the TROLL *light* features for simple objects we now turn to composite objects. In order to combine several authors in a higher-level object, classes are usually introduced as containers in object-oriented databases. In TROLL *light* an explicit class concept is not needed. Classes are viewed as composite objects instead, and therefore classes are described by templates as already mentioned before. However, means of describing the relationship between a container object and the contained objects must be added. This is done by introducing sub-object relationships denoting (exclusive) part-of relationships. The following example gives the format of container objects for authors.

```
TEMPLATE AuthorClass
  DATA TYPES    String, Date, Nat;
  TEMPLATES     Author;
  SUBOBJECTS    Authors(No:nat):author;
  ATTRIBUTES    DERIVED NumberOfAuthors:nat;
  EVENTS        BIRTH createClass;
                      addObject(No:nat,Name:string,BirthDate:date);
                      removeObject(No:nat);
                DEATH destroyClass;
  CONSTRAINTS   NumberOfAuthors<10000;
  DERIVATION    NumberOfAuthors=CNT(Authors);
  INTERACTION   addObject(N,S,D) >> Authors(N).create(S,D);
                removeObject(N) >> Authors(N).destroy;
END TEMPLATE;
```

2.6 Templates section

Within the `TEMPLATES` section, other (existing) templates can be made known to the current template. We assume templates to induce corresponding object sorts. Hence referring to `Author` means that the object sort `author`, and the attributes and the event generators of `Author` are visible in `AuthorClass`.

2.7 Subobjects section

An object of sort `authorClass` will hold finitely many author objects as private components or `SUBOBJECTS`. In order to be able to distinguish several authors from a logical point of view, an explicit identification mechanism is needed. One way to install such a mechanism would be to assign a unique name for each sub-object, e.g., `MyAuthor`, `YourAuthor`, ... Indeed, such a name allocation could be expressed in TROLL *light* as follows

```
SUBOBJECTS  MyAuthor:author; YourAuthor:author; ...;
```

But clearly, in the case of a large number of authors such a procedure is not practicable. A solution is given by parametrized sub-object constructors as shown in the example. As with parametrized attributes, a possibly infinite number of logical sub-object names like `Authors(42)` are defined by the sub-object name declaration for `Authors`, but not the author objects themselves. The declaration only states that in context of an object of sort `authorClass`, author objects are identified by a natural number. In semantic data models the parameter `No` would be called a key. But the parameters need not be related to any attributes of the objects they identify. Each logical sub-object name corresponds to an object of the appropriate object sort. Analogously to attributes, we demand that in each state only a finite number of defined sub-objects exists.

2.8 Interaction section

Interaction rules are specified in the `INTERACTION` part of the template definition. During the lifetime of an author class there will be events concerning the insertion or deletion of authors. In `AuthorClass`, the insertion of an author should always be combined with the creation of a new author object. In the template definition this is expressed by an event calling rule `addObject(N,S,D) >> Authors(N).create(S,D)`, where `N` denotes a natural number, `S` a string, and `D` a date. The general event calling scheme is expressed as [`{`*precond*`}`] [*src_obj* .]*src_event* `>>` [*dest_obj* .]*dest_event*. Such a rule states that whenever an event belonging to the source event *src_event* occurs in the source object denoted by *src_obj*, and the formula *precond* holds, then the event denoted by the destination event *dest_event* must occur simultaneously in the destination object denoted by *dest_obj*. If one of the object terms is missing, then the corresponding event term refers to the current object. The source object term is not allowed to have free variables, but the destination object term, the destination event term, and the precondition are allowed to have free variables, which however have to occur in the source event term. As already mentioned in the explanation of valuation rules, these free variables are instantiated by an event occurrence corresponding to the event term in such a way that the precondition and the other terms can be evaluated. In almost all cases the objects to be called for must be alive. The one and only exception is when a parent object calls for the birth of a sub-object. In this case it is even a requisite that the destination object does not exist already.

2.9 Constraints and derivation sections

In addition to the mentioned language features TROLL *light* offers some further, more sophisticated concepts which will be mentioned only briefly: Possible object states can be restricted by explicit constraints, which are specified in the `CONSTRAINTS` section of a template. Constraints are given by closed formulas in the TROLL *light* query calculus (cf. [BNPS92]), an offspring of an extended Entity-Relationship calculus [GH91]. For example, an author container of sort `authorClass` may only be populated by less than 10000 authors (as required above). Derived attributes can be specified by stating `DERIVATION` rules which, in fact, are closed terms of the TROLL *light* query calculus. In the example we declared the attribute `NumberOfAuthors` of `authorClass` as derived. We employed the function `CNT` which counts the elements of a multi-valued term. In the derivation rule `Authors` is regarded as a term of sort `set(tuple(nat,author))`. So `NumberOfAuthors` does not need to be set by explicit valuation rules.

Up to now we have only dealt with the description of objects (i.e., templates). In order to obtain object instances we must choose one template as the schema template, and one fixed object of the corresponding object sort as the schema object. All other objects in an object community will be sub-objects of this distinguished schema object. More details concerning TROLL *light* can be found in [CGH92, GCH93, HCG94].

Chapter 3

The Development Environment

The TROLL *light* development environment[Vla93] is an open system which allows developers to integrate new tools and to adapt and extend it to their own requirements. Integration requirements concern the adaptation of tools, i.e., new or adapted tools should be able to work with documents (specifications, source code, etc.) created by other tools [VH93]. This is a very important aspect because documents have certain structure and semantics and are stored in a structured way in a repository.

For preserving the openness of the environment tools have to be as independent as possible[SB93, Kro93, ESF91]. For this the TROLL *light* development environment supports a loose coupling of tools. This is achieved on the one hand by a kind of tool communication which is based on interchanging messages and notifications (HP SoftBench [Cag90]) and on the other hand by an object-oriented documents repository (ObjectStore [LLOW91]). The latter supports the possibility of storing design documents in a structured way whereas the former provides mechanisms for tool integration[TN92, Wyb91].

The core of the HP SoftBench is the *Broadcast Message Server (BMS)* responsible for the indirect message exchange protocol. Tools can decide themselves whether they react to certain messages. During the starting phase they inform the BMS about the messages they are interested in. The BMS forwards messages only to interested tools. If, for instance, the TROLL *light* editor closes the creation of a specification it will inform its environment through a notification message about this event. The environment, i.e., other interested tools, may then decide on their own what to do next. The TROLL *light* parser could start parsing the new specification, and after this it could inform its environment about the results. Again other tools may take actions on their own. Since tools do not depend on each other it is quite easy to add, replace, or remove a tool from the environment.

The TROLL *light* development environment can be seen as an instantiation of the *ECMA* Reference Model for software development environments [Ear90] (see Fig. 3.1). It is built on the depicted integration frames BMS, ObjectStore, and Tcl/Tk user interface manager [Ous93]. A further discussion of the TROLL *light* development environment can be found in [VHG$^+$93].

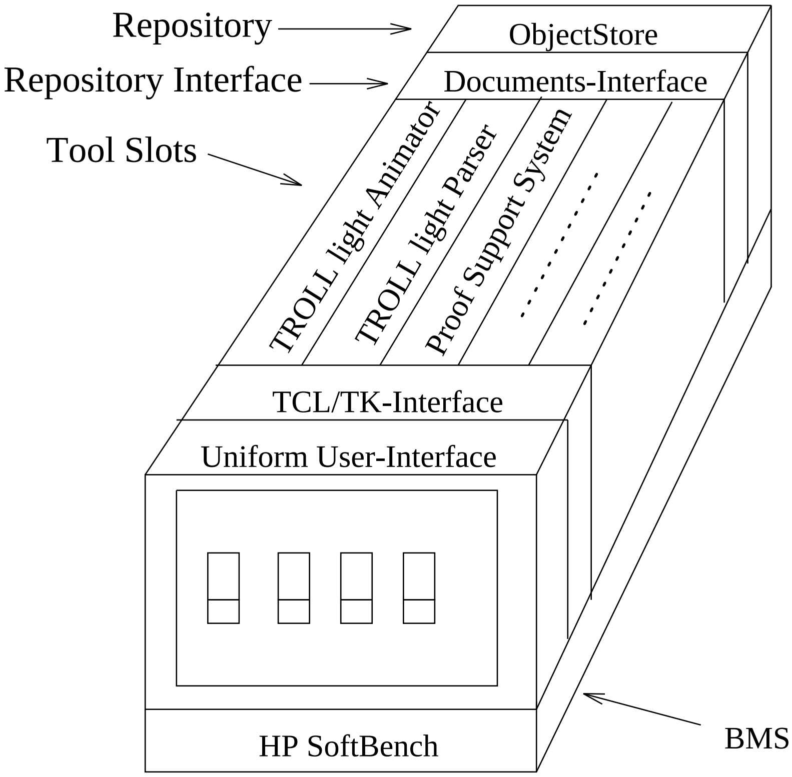

Figure 3.1: Architecture of the TROLL *light* development environment

3.1 The Animation System

3.1.1 Animating templates

A template generally describes structural and dynamic aspects of a prototypical object. Structural properties are centered around the specification of possible attribute states, dynamic properties around the specification of possible event sequences. Looking at a template with sub-object slots the prototypical object described by this template is in fact an object community where events in different objects may be synchronized by interaction rules.

Speaking in more technical terms the model of a template is a state transition system in which a state transition is accompanied by a finite set of event occurrences which has to be closed against synchronization. TROLL *light* object descriptions abstract from the causality or initiative of event occurrences. Hence making a state transition system to move means to indicate certain event occurrences from the outside of the system. This is what we call *animation* of templates (cf. [CRB93, DDD93, HG94b]).

Animation of templates may help to assure that the specified behavior of objects or object communities matches the required behavior. Of course, animation of a conceptual model shows the same problems like testing an implementation. From observations that the observed behavior agrees with the requirements we would like to draw the conclusion that a conceptual schema (an implementation) is correct with respect to the requirements (the conceptual schema). This, however, cannot be done, because there still may be some traces in the animation (in an implementation) which have not been tested and which may show the opposite effect. Hence animation (testing) is only useful to falsify informal correctness [Wie93].

3.1.2 Requirements for a tool supporting animation

A software system with the aim to support the animation of templates should meet the following requirements. It should support (1) the exploration of actual states of an object community, (2) the specification of event occurrences for initiating state transitions, and (3) the visualization of state changes.

To be more precise exploration of states should be assisted by means to illustrate the actual global structure of an object community, to show the attribute state (and optionally the behavior state) of a single object, to traverse sub-object relationships or object-valued attributes, and eventually to formulate ad hoc queries against object states.

With respect to the specification of event occurrences it must be possible to indicate event occurrences for possibly more than one object at a time. The system should help to find event sets being closed against synchronization. Hence, when a specified event occurrence provokes a second event occurrence in another object, this event occurrence should be added automatically to the current event set.

State changes provoked by a given closed event set, as there might be insertions and deletions of objects or attribute updates, should be made visible to the user, for example by appropriate messages in a specific window. When a desired state transition cannot be carried out, for instance because the resulting state violates some integrity constraints, these conditions should also be reported to the user.

3.1.3 Architecture of the animation system

The structure of the TROLL *light* animation system is depicted in Fig. 3.2. First of all the animation system consists of a *template dictionary* which is a persistent store for object descriptions. The template dictionary is not an exclusive part of the animation system but an integral part of the whole development environment, for instance shared by the parser and the proof support system. The second basic component of the animation system is the *object base* which is a persistent store to hold object states. In principal, the object base could also be a transient store but in the system used for implementation there was only a slight distance between transient and persistent structures so that we preferred persistent structures. Hence it is possible to stop a current animation session at one time and start it again later on.

Object descriptions contain terms and formulas of the TROLL *light* query calculus which are evaluated by the third component of the animation system, the *term evaluator*. The evaluation of terms and formulas generally depends on the current

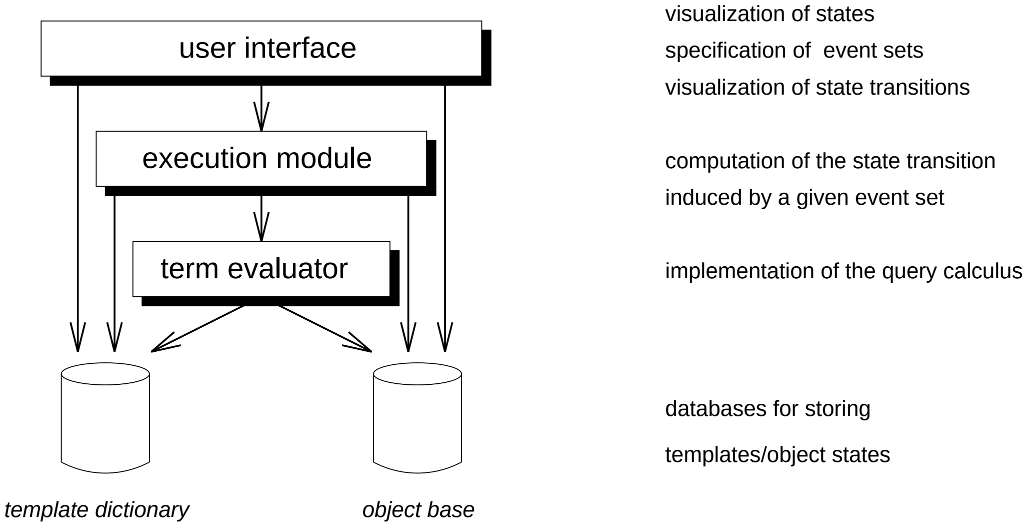

Figure 3.2: Architecture of the TROLL *light* animation system

state of an object community so that the term evaluator must be able to access the object base. The *execution module* is the heart of the animation system. Its task is to compute for a given set of event occurrences a successor state or to report errors if such a state cannot be determined for different reasons. Finally the *user interface* establishes communication with the human operator.

3.1.4 Implementation aspects

A prototype version of the animation system covering basic requirements has been completed with the end of the KORSO project. Work was done on the basis of three diploma thesis.

The template dictionary and the object base were implemented using the OODBMS ObjectStore [LLOW91] on the basis of C++. The object base is designed for storing values of any complexity, even a whole NF^2 or complex object model database [SS86, AFS89].

The term evaluator and the execution module were implemented in C++. Term evaluation generally results in a complex value which is stored using the same structures as found in the object base. It is noteworthy that terms that are to be evaluated are not limited concerning their complexity. For instance, it has been shown that the term evaluator was even capable of evaluating 1000 nested select-from-where queries.

Finally the user interface was realized by the help of Tcl/Tk [Ous93]. In the prototype version of the animation system object states are visualized in so called *object windows* (see Fig. 3.3). Object windows resemble the representation of pointers to current sub-objects, the visualization of current attribute values as well as a

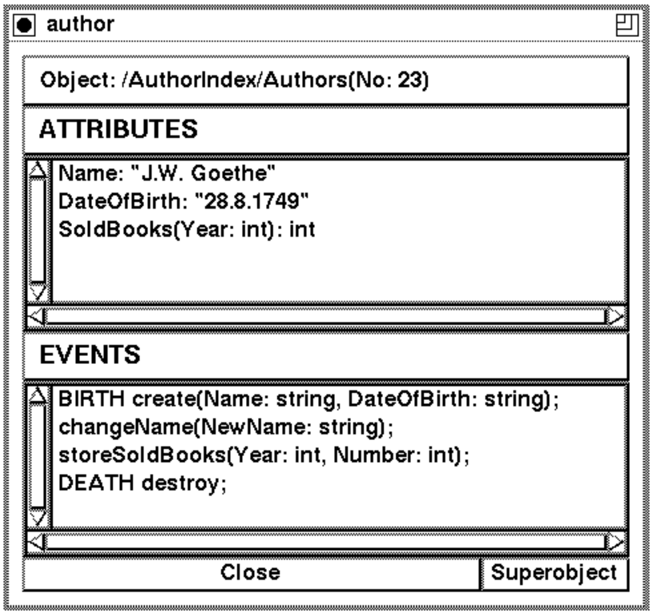

Figure 3.3: Object window

depiction of the list of possible event generators. Event occurrences can be specified by marking an event generator and optionally entering required parameter values. Specified event occurrences are collected in a separate window from which a set of event occurrences may be sent to the execution module.

3.2 The Proof Support System

3.2.1 Verification calculus

The development of the verification calculus was driven by pragmatic considerations. Since the simplicity of the proof system was of great importance for our purposes (i.e., verifying object properties), we decided to choose a simple structure of formulas. We only allow Gentzen-formulas with a conjunction of propositions as antecedent and a disjunction of propositions as conclusion, i.e., formulas do not have other formulas as subformulas. The decision for this kind of formulas was motivated by the fact that thereby we can easily adopt existing proof systems of various sequent calculi, e.g., [Pae88, FM91, Gal93]. Furthermore, we have found a generic theorem prover (ISABELLE [Pau90]) which allows for a realization of such sequents.

In order to give an impression of the verification calculus we briefly present the main concepts followed by an example. More details on the calculus, its proof system, and the translation of TROLL *light* object descriptions into formulas of this calculus can be found in [Con93b, Con94b, Con94a].

The most important concept is the concept of formulas. As already mentioned we allow only a special form of clauses as formulas: $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ where P_i and Q_j are so-called propositions. Such a formula must be understood as follows: in each state (of an object community) in which all the propositions $P_1, \ldots, P_n$ are fulfilled there is at least one of the propositions $Q_1, \ldots, Q_m$ which is also fulfilled in that state.

Next, we have to introduce our notion of proposition. Propositions are mainly given by predicate expressions $p(t_1, \ldots, t_n)$ (with p a predicate symbol and t_i appropriate terms). Furthermore we allow negation (i.e., $\neg P$) and the use of positional operators (i.e., $[t_o.t_e]P$, where t_o is a term denoting an object and t_e a term describing an event for this object). A proposition $[t_o.t_e]P$ could be read as "if the event occurring in object t_o is t_e then P holds afterwards". We distinguish between two kinds of predicate symbols: rigid and non-rigid ones (i.e., state-independent and state-dependent predicate symbols, resp.). For instance, rigid predicate symbols are given by the data type specification used (e.g., $\leq$ for integers), whereas attributes of objects are modeled by non-rigid predicates. Furthermore we have non-rigid predicates for dealing with enabledness and occurrence of events, namely the predicates *enable* and *occur*.

The following formulas express properties of `authorClass` and `author` objects as described above:

$$\begin{aligned} &NumberOfAuthors(AC, C) \ \rightarrow \\ &\qquad [AC.addObject(N, S, D)]NumberOfAuthors(AC, C+1) \\ &\rightarrow \ [A.changeName(N)]\neg\, enable(A.changeName(N)) \end{aligned}$$

The first formula states that given a value C for the attribute `NumberOfAuthors` of an `authorClass` object AC this attribute value is increased after the occurrence of an `addObject` event in the object AC. The second formula describes that after the occurrence of a `changeName` event in an author object A a further occurrence of a `changeName` event in the same object is not allowed.

Properties to be proved must be expressed as formulas as well. For instance, the formula

$$\rightarrow \ [A.changeName(N_1)]\neg\, occur(A.changeName(N_2))$$

can be proved from the set of formulas obtained by transforming the specification of `authorClass` and `author` objects into formulas of our calculus.

3.2.2 Limitations of the calculus

Of course, the question arises which kind of properties can be proved by means of this calculus. At the time being, the expressive power of the calculus and the power of the proof system is quite limited, although we only need skolemization for transforming arbitrary formulas of first-order predicate logic into formulas of our calculus. But we cannot express arbitrary temporal properties for objects because there are only positional operators which can be compared with the next operator of temporal logics. In consequence, we cannot reason about, e.g., liveness properties.

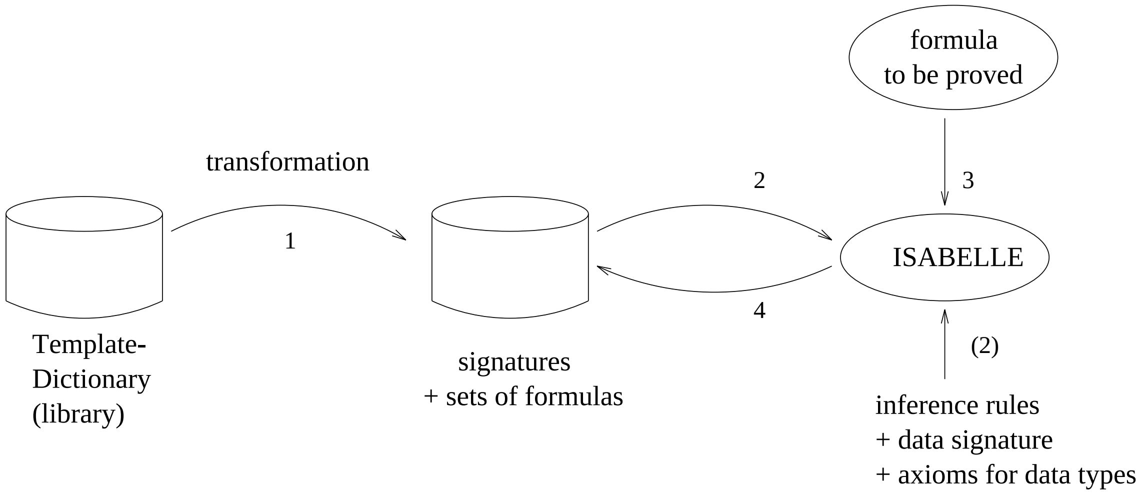

Figure 3.4: Working with the proof support system

We can only consider single state transitions or finite sequences of events. However, later extensions of our calculus are not excluded. Especially the introduction of additional temporal operator for the future like "always" and "sometime" seems to be advisable.

We have been aware of these limitations from the beginning. But our aim has been to develop a basic calculus which can be used as an experimental basis for modifications and extensions. In consequence, this calculus should only cover inevitable aspects, and we have taken care that the calculus goes together with our specification language TROLL *light*.

3.2.3 Verifying properties

After having introduced the verification calculus we can give a brief impression of how the proof support system is embedded in the development environment. In principle we have the following steps in mind which are also depicted in Fig. 3.4:

1. The (automatic) transformation of a TROLL *light* specification yields a corresponding template signature together with a set of formulas giving a logical description of the specified objects by means of the verification calculus. The results of such transformations are stored in a database.

2. When the transformation of a TROLL *light* specification has been completed, the generic theorem prover ISABELLE [Pau90] can be activated. In order to use the theorem prover for the verification calculus its syntax and inference rules must be loaded into the prover (in fact we can generate an instance of ISABELLE in which these parts are already included). Next, we have to include the signature and the formulas obtained from the TROLL *light* specification which shall be investigated.

3. Now we can formulate a property to be proven as a formula of the verification calculus. Given such a formula we can apply the prover. The proof can be controlled interactively, i.e., we can carry out the proof step by step by applying single inference rules or we can cause ISABELLE to automatically run parts of the proof by choosing suitable proof tactics.

4. If the proof of a property was successful the corresponding formula can be added to the database. Thereby, this formula can be reused later in proving further properties of the same objects.

Of course this is only a first, rough prototype of a proof support system for gaining experience in verifying object properties. This phase has just begun. First examples have already been carried out successfully, however, more and larger example are necessary to be able to judge the adequacy of the calculus and the conception of the proof support system. Afterwards this system can be revised in order to better meet pragmatic and practical requirements. At the time being a bothersome impediment in handling larger proofs is that nearly each proof step must be done by hand. By developing adequate proof tactics and adding them to the theorem prover we will perhaps be able to improve this situation.

Chapter 4

Activities of the Braunschweig KORSO Group

4.1 Abstracts of Papers

To give an overview on the scientific results we recite the abstracts of the papers written within the project ordered by the bibtex key. If a paper is written in German then also the abstract is given in German.

Zur Entwicklung zuverlässiger Informationssysteme in KorSo [CDG+93]

Stefan Conrad, Grit Denker, Martin Gogolla, Rudolf Herzig, Nikolaos Vlachantonis, Hans-Dieter Ehrich

Innerhalb des KorSo-Projektes verfolgt unsere Gruppe das Ziel, die Entwicklung zuverlässiger Informationssysteme auf der Basis formaler Spezifikationen zu unterstützen. Dazu konzentriert sich unsere Arbeit auf die Spezifikationssprache TROLL *light*, die es erlaubt, Teile der zu modellierenden Welt als eine Gemeinschaft nebeneinander bestehender und miteinander kommunizierender Objekte zu beschreiben. Auf diese Weise bestimmen wir sowohl die Struktur als auch das Verhalten der konzeptionellen Objekte. Unsere Spezifikationsumgebung für TROLL *light* erlaubt die Animation von Spezifikationen ebenso wie das Beweisen von Eigenschaften der Spezifikationen unter Verwendung von Theorembeweisern.

Dieser kurze Artikel gibt einen Überblick über die Arbeit im Braunschweiger Teilprojekt. Dazu wird kurz die Sprache TROLL *light* skizziert. Nach Diskussion der Bedeutung von Zertifikation, also Validation und Verifikation von Spezifikationen, wird die TROLL *light* Entwicklungsumgebung für den Entwurf von Informationssystemen vorgestellt. Abschließend wird der Stand der Implementierung dieser Entwicklungsumgebung dargelegt.

An Elementary Logic for Object Specification and Verification [CE94]

Stefan Conrad, Hans-Dieter Ehrich

In this paper we sketch an elementary logic for specifying information systems as systems of dynamic objects. First, we informally present the ingredients which should be included in a logic for abstract object specifications. Afterwards we turn to a calculus which realizes a basic part of such a logic. For this calculus a simple proof system is provided which permits to gain experience in verifying object properties. We conclude by briefly pointing out future work.

An Annotated Bibliography on Object-Orientation and Deduction [CG92]

Stefan Conrad, Martin Gogolla

This note tries to briefly survey research activities and results on the integration of object-oriented concepts and deductive database languages.

There are several efforts creating new deductive languages and systems extending conventional deductive database languages by involving concepts of higher order logics, especially the treatment of sets. In contrast to these value-based languages, object logics try to integrate the most important object-oriented concepts. Thereby, object identities and inheritance are available in these languages. Nevertheless, the ways to integrate object identities, classes and inheritance are quite different.

Besides these efforts in designing object-oriented deductive languages, the theoretical foundations of object-oriented, of deductive and of object-oriented and deductive systems are investigated. Whereas several well known and overall accepted semantics for logic programming exist, there is no clear consensus about what an object-oriented system is. Therefore, in this area a lot of work still has to be done.

This survey can only be a snapshot of a very lively research area. Up to now a lot of excellent results have been presented. However, there still are some important steps to go on the way towards a full integration of object-orientation into deductive database languages.

TROLL *light*: A Core Language for Specifying Objects [CGH92]

Stefan Conrad, Martin Gogolla, Rudolf Herzig

TROLL *light* is a language for conceptual modeling of information systems. It is designed to describe the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects. TROLL *light* objects have observable properties modeled by attributes, and the behavior of objects is described by events. Possible object observations may be restricted by constraints, whereas event occurrences may be restricted to specified life cycles. TROLL *light* objects are organized in an object hierarchy established by subobject relationships. Communication among objects is supported by event calling. Apart from introducing the various possibilities for the syntactical description of objects, we aim to describe how the state of an object community may be changed by event occurrences.

Einbindung eines bestehenden Datenbanksystems in einen formalen Software-Entwicklungsprozeß [Con93a]

Stefan Conrad

Dieser Beitrag beschäftigt sich mit dem Problem, wie ein bestehendes Datenbanksystem innerhalb eines formalen Software-Entwicklungsprozeß berücksichtigt werden kann. Konkret betrachten wir den Fall, daß ein relationales Datenbanksystem vorliegt. Dazu ist zunächst ein relationaler Datenbankentwurf durchzuführen, der die formal spezifizierte Welt repräsentiert. Ein Schwerpunkt der weiteren Betrachtungen ist die Behandlung von Integritätsbedingungen, die in der formalen Spezifikation formuliert wurden.

On Certification of Specifications for TROLL *light* Objects [Con93b]

Stefan Conrad

In this paper we informally present a certification calculus for the object specification language TROLL *light*. The language supports the state- and behavior-oriented specification of information systems. It allows orthogonal construction of large systems from subsystems. The certification calculus provides a basis for verifying properties of specified objects. Besides an introduction to this calculus we show how formulae of this calculus can be derived from TROLL *light* specifications. The transformation from specification language into calculus shall later be done automatically. Furthermore, we demonstrate proving of object properties by means of an example.

Spezifikation eines vereinfachten Datenbanksystems — ein Beitrag zur HDMS-A-Fallstudie [Con93c]

Stefan Conrad

Eine Möglichkeit bestehende Teilsysteme innerhalb eines formalen Software-Entwicklungsprozeß zu berücksichtigen ist, sie formal nachzuspezifizieren. Hier führen wir dies examplarisch für ein relationales Datenbanksystem vor. Allerdings werden wir nur einige Teile formal mit TROLL *light* beschreiben. Dies reicht aber schon aus, um die wesentlichen Probleme eines solchen Vorgehens erkennen und diskutieren zu können.

Zur Verifikation von Objekteigenschaften [Con93d]

Stefan Conrad

Der Beitrag geht kurz auf die Konzepte ein, die die Spezifikationsprache TROLL *light* bietet. Anschließend präsentieren wir informell unseren Verifikationskalkül, für den wir eine modelltheoretische Semantik auf der Basis von modifizierten Kripke-Strukturen skizzieren. Dabei sollen vor allem Aspekte der Kompositionalität behandelt werden.

A Basic Calculus for Verifying Properties of Synchronously Interacting Objects [Con94a]

Stefan Conrad

We introduce a basic calculus for expressing and proving properties of synchronously interacting objects. The objects considered have dynamic behavior and are organized into object communities in a hierarchical way. We present syntax and semantics of the calculus in detail. Moreover, a set of basic inference rules is given. Essential properties of the calculus are discussed, especially the question of compositionality. The calculus developed constitutes a basis for investigating issues of verifying properties of objects and object communities. In consequence, we have focused on a small number of essential concepts. We hope that this calculus can contribute to gain experience in verifying objects. Therefore, a prototype implementation of the calculus using a generic theorem prover has been developed.

Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte [Con94b]

Stefan Conrad

In dieser Arbeit wird ein Basiskalkül für die Verifikation von Objekteigenschaften vorgestellt. Die frühzeitige Verifikation wichtiger Eigenschaften stellt eine formale Unterstützung des Entwurfes von Informationsystemen dar und kann dazu beitragen, Kosten für spätere Fehlersuche und -behebung zu vermeiden. Insbesondere bei sicherheitsrelevanten Anwendungen ist der formale Nachweis kritischer Eigenschaften oft erforderlich.

Gegenstand der Betrachtung sind Objekte in Informationssystemen, die z.B. mit Hilfe der Sprache TROLL *light* spezifiziert werden können. Diese Objekte haben Attribute und atomare Ereignisse, die Zustandsänderungen verursachen. Die Objekte in einem Informationssystem können über Ereignisaufrufe, die eine synchrone Ausführung der beteiligten Ereignisse bewirken, miteinander kommunizieren und interagieren.

Der hier entwickelte Kalkül kann als Basis für verschiedene Objektbeschreibungssprachen eingesetzt werden, da er in einfacher Weise um weitere Konzepte erweitert werden kann. Die Verwendung einer einfachen Formelstruktur ermöglicht ein überschaubares und somit leicht verwendbares Ableitungssystem. Mit Hilfe dieses Basiskalküls und seiner Realisierung auf einem generischen Beweissystem können nun die Besonderheiten der Verifikation von Eigenschaften miteinander kommunizierender Objekte in "Reinkultur" untersucht werden.

Im Gegensatz zu anderen Ansätzen wurde von Anfang an darauf geachtet, daß die entstehenden Formeln sowohl objektlokal als auch systemglobal gültig sind. Damit werden semantische Probleme beim Zusammensetzen von Spezifiktionen vermieden und kompositionales Beweisen ermöglicht. Hierzu wird eine Lösung für ein Problem angeboten, das bislang oft vernachlässigt wurde. Es handelt sich dabei um eine adäquate Behandlung der verschiedenen objektlokalen Begriffe des "nächsten" Zustandes. Dies ist für die Nutzbarkeit eines solchen Kalküls in der Praxis von besonderer Bedeutung – und dort insbesondere für die Wiederverwendung von Spezifikationen, die das zeitliche Verhalten der beteiligten Objekte umfassen.

Towards Semantic Foundation of Basic TROLL *light* Constructs [Den92]

Grit Denker

The specification language TROLL *light* is designed for the conceptual modeling of information systems. TROLL *light* specifications describe the Universe of Discourse as a system of concurrently existing and interacting objects. The state of a system is given by the states of all objects, i.e., their attributes values and their process state, which are currently part of the system. System evolution is caused by a set of events which are to be executed concurrently and give rise to changes of several objects.

Assuming this interpretation of systems and their evolution we adopt an approach by Meseguer which deals with rewriting systems and concurrency. This approach provides a more general interpretation of conditional rewriting logic, in the sense of a logic of change and action, to come up with a model of concurrency. The approach covers a lot of already known models, and especially it can be applied to the object-oriented subject.

Translating TROLL *light* Concepts to Maude [DG93]

Grit Denker, Martin Gogolla

The specification language TROLL *light* is designed for the conceptual modeling of information systems. Maude is a logic programming language, which unifies the two paradigms of functional and concurrent object-oriented programming. Because of the very similar features offered by both languages, we present a translation from TROLL *light* concepts into the Maude language in order to compare the languages. Apart from presenting the translation, the languages are briefly described and illustrated by examples.

Entwicklung zuverlässiger Informationssysteme - Stand der Arbeiten im Januar'93 [EGC+93]

Hans-Dieter Ehrich, Martin Gogolla, Stefan Conrad, Grit Denker, Rudolf Herzig, Nikolaos Vlachantonis

Innerhalb des KorSo-Projektes befaßt sich unsere Gruppe damit, die Entwicklung zuverlässiger Informationssysteme auf der Basis formaler Spezifikationen zu unterstützen. Dazu konzentriert sich unsere Arbeit auf die Spezifikationssprache TROLL *light*, die es erlaubt, Teile der zu modellierenden Welt als eine Gemeinschaft nebeneinander bestehender und miteinander kommunizierender Objekte zu beschreiben. Auf diese Weise bestimmen wir sowohl die Struktur als auch das Verhalten konzeptioneller Objekte. Unsere Spezifikationsumgebung für TROLL *light* erlaubt die Animation von Spezifikationen ebenso wie das Beweisen von Eigenschaften der Spezifikationen unter Verwendung von Theorembeweisern.

Entwicklung zuverlässiger Informationssysteme - Stand der Arbeiten im Januar'94 [EGC+94]

Hans-Dieter Ehrich, Martin Gogolla, Stefan Conrad, Grit Denker, Rudolf Herzig, Nikolaos Vlachantonis

Um die Entwicklung großer Software- und Informationssysteme zu unterstützen, sind Software-Werkzeuge notwendig, die, um effektiv arbeiten zu können, innerhalb einer Software-Entwicklungsumgebung integriert werden müssen. Kern unserer Entwicklungsumgebung ist die Spezifikationssprache TROLL *light*, die es erlaubt, den zu modellierenden Weltausschnitt, der in einem Informationssystem gespeichert werden soll, als eine Gemeinschaft nebeneinander bestehender und miteinander kommunizierender Objekte zu beschreiben. 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 der Spezifikation.

Beiträge zu KorSo- und TROLL *light* Fallstudien [Ehr93a]

Hans-Dieter Ehrich (Ed.)

In diesem Bericht sind einige Beiträge der Braunschweiger KorSo-Gruppe zusammengestellt, die innerhalb von KorSo-Fallstudien oder als interne Arbeiten entstanden sind. Die ersten beiden Beiträge sind im Zusammenhang mit der HDMS-A-Fallstudie entstanden. Der dritte gibt eine Modellierung der Fertigungszelle unter Verwendung der Objektbeschreibungssprache TROLL *light* an. Im vierten Beitrag wird der Einsatz von TROLL *light* zur Spezifikation eines Template-Dictionary gezeigt, das zur strukturierten Ablage von Objektbeschreibungen innerhalb einer Entwicklungsumgebung dient. Abschließend wird noch eine Sammlung kleinerer TROLL *light* Spezifikationen vorgestellt. Es handelt sich dabei zumeist um Beispiele, die in der Literatur für verschiedene andere Spezifikationsformalismen verwendet werden.

Entwicklung zuverlässiger Informationssysteme [Ehr93b]

Hans-Dieter Ehrich

Dieser kurze Artikel gibt einen Überblick über die Arbeit im Braunschweiger Teilprojekt. Dazu wird kurz die Sprache TROLL *light* skizziert. Nach Diskussion der Bedeutung von Zertifikation, also Validation und Verifikation von Spezifikationen, wird die TROLL *light* Entwicklungsumgebung für den Entwurf von Informationssystemen vorgestellt. Abschließend wird der Stand der Implementierung dieser Entwicklungsumgebung dargelegt.

An Object-Oriented Design for the ACT ONE Environment [GC93]

Martin Gogolla, Ingo Claßen

The overall aim of this paper is to stabilize and strengthen the algebraic specification method to software engineering and development. We do not introduce new theoretical results, but define a conceptual model, i.e., an information system schema, for the well-established algebraic specification language ACT ONE and its accompanying specification environment.

This paper gives a formal description of suitable database support for an environment supporting the interactive development of ACT ONE specifications. The object-oriented data model of TROLL *light*, a language developed recently within the KorSo project, is used to present the design of such an environment and its dynamic behavior. However, the concepts used are general enough to support other specification and even programming languages as well. Therefore, we feel the design of a conceptual schema for ACT ONE is mainly a case study in employing an object-oriented data model for database support of specification or programming languages. It is therefore a proposal for the consolidation of environments for algebraic specification languages.

Sketching Concepts and Computational Model of TROLL *light* [GCH93]

Martin Gogolla, Stefan Conrad, Rudolf Herzig

The specification language TROLL *light* is intended to be used for conceptual modeling of information systems. It is designed to describe the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects, i.e., an object community.

The first part of the present paper introduces the various language concepts offered by TROLL *light*. TROLL *light* objects have observable properties modeled by attributes, and the behavior of objects is described by events. Possible object observations may be restricted by constraints, whereas event occurrences may be restricted to specified life-cycles. TROLL *light* objects are organized in an object hierarchy established by sub-object relationships. Communication among objects is supported by event calling.

The second part of our paper outlines a simplified computational model for TROLL *light*. After introducing signatures for collections of object descriptions (or templates as they are called in TROLL *light*) we explain how single states of an object community are constructed. By parallel occurrence of a finite set of events the states of object communities change. The object community itself is regarded as a graph where the nodes are the object community states reachable from an initial state and the edges represent transitions between states.

Integrating the ER Approach in an OO Environment [GHC+94]

Martin Gogolla, Rudolf Herzig, Stefan Conrad, Grit Denker, Nikolaos Vlachantonis

We translate Entity-Relationship (ER) schemas into the object-oriented specification language TROLL *light*. This language describes the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects, i.e., an object community. Thereby two essential aspects, structure and behavior, are integrated in one formalism. By doing the translation from ER to TROLL *light* we preserve the visual advantages of the former and receive a formalism through the latter which can be mapped to an adequate object-oriented database system. Proceeding this way we hope our proposal for transforming ER schemas into TROLL *light* specifications provides a valuable link between structural and dynamic modeling.

Some Examples for TROLL *light* Templates [Gog93]

Martin Gogolla

We introduce some small examples for templates which are given in the language TROLL *light* intended for the specification of conceptual objects in information systems. The spectrum of our examples ranges however from rather untypical information systems like clocks, counters, or flipflops over stacks and trees to more specific information system applications like libraries.

Compositional Description of Object Communities with TROLL *light* [HCG94]

Rudolf Herzig, Stefan Conrad, Martin Gogolla

TROLL *light* is a language for the modeling of information systems. It is designed to describe the Universe of Discourse (UoD) as a system of concurrently existing and interacting objects. TROLL *light* objects have observable properties modeled by attributes, and the behavior of objects is described by events.

Possible object observations may be restricted by constraints, whereas event occurrences may be restricted to specified life cycles. TROLL *light* objects are organized in an object hierarchy established by subobject relationships. Communication between objects is supported by event calling. Apart from introducing the various possibilities for the syntactical description of objects, we aim to describe how the state of an object community may be changed by event occurrences.

Spezifikation der abstrakten Syntax von TROLL *light* [Her93]

Rudolf Herzig

Einen wichtigen Bestandteil der TROLL *light* Entwicklungsumgebung bildet das Template-Dictionary. Dies ist ein persistenter Speicher zur strukturierten Ablage von TROLL *light* Objektbeschreibungen. Als Vorbereitung der Implementierung eines solchen Speichers wird die abstrakte Syntax der Sprache TROLL *light* mit TROLL *light* selbst spezifiziert.

Zur Spezifikation von Objektgesellschaften mit TROLL *light* [Her94]

Rudolf Herzig

Objektorientierung präsentiert sich als das Schlagwort der neunziger Jahre. Allerdings existiert bis heute noch kein Konsens darüber, was Objektorientierung eigentlich auszeichnet. In dieser Arbeit wird eine spezielle Perspektive von Objektorientierung verfolgt, nämlich die objektorientierte Spezifikation von Informationssystemen als ein Ansatz zur integrierten Beschreibung von Struktur und Verhalten konzeptioneller Objekte. Als Beschreibungssprache wird TROLL *light* vorgestellt. TROLL *light* ist eine schlanke, übersichtliche Sprache, welche in Form sogenannter Templates einen einheitlichen Rahmen zur Beschreibung sowohl einfacher Objekte als auch ganzer Objektgesellschaften als Hierarchien parallel existierender, synchron interagierender Objekte bereithält.

Zur semantischen Fundierung der Sprache werden zunächst die Sprachkonzepte von TROLL *light* in mathematisch-abstrakter Form definiert und dann Objektgesellschaften als Zustandsübergangssysteme beschrieben. Die formalen Überlegungen bildeten die Grundlage zur Implementierung eines Werkzeugs, welches über die prototypische Animation von Objektbeschreibungen die Validierung von Objektspezifikationen unterstützt. Es wird über die Konzeption dieses Werkzeugs und spezielle Implementierungsaspekte berichtet.

TROLL *light* enthält als Teilsprache einen Anfragekalkül, welcher innerhalb von Templates beispielsweise die Beschreibung abgeleiteter Attribute und statischer Integritätsbedingungen unterstützt, ansonsten aber auch für SQL-artige Ad-hoc-Anfragen an Objektgesellschaften zur Verfügung steht. Die Definition dieses Kalküls erfolgt unabhängig von TROLL *light* im Rahmen eines allgemeinen Modells komplexer Werte, so daß der Kalkül sowohl als Basis relationaler als auch objektorientierter Anfragesprachen verwendet werden kann.

Aus dem Blickwinkel der Vorbereitung einer entsprechenden Spezifikationsmethodik wird objektorientierte Spezifikation in ein Schichtenmodell für Informationssysteme eingeordnet, welches eine Datenschicht zur Beschreibung konzeptionell zustandsloser Datenwerte und eine Objektschicht zur Beschreibung von Dingen mit Zustand unterscheidet. Vorbereitet durch einen Vergleich bekannter Beschreibungstechniken zur Funktions-, Struktur- und Verhaltensmodellierung, werden schließlich die Sprachkonzepte von TROLL *light* kritisch bewertet und eventuell notwendige Erweiterungen diskutiert.

Zur Gestaltung der Benutzeroberfläche bei der Animation von Objektspezifikationen [HFG94]

Rudolf Herzig, Heiko Fischer, Martin Gogolla

Wir diskutieren Animation als einen möglichen Weg zur Validierung von Objektspezifikationen und stellen die Benutzungsschnittstelle des TROLL light Animationssystems vor. Im Mittelpunkt stehen Fragen der Visualisierung des Zustands einer Objektgesellschaft sowie der Einleitung von Zustandsübergängen durch die Vorgabe endlicher Ereignismengen.

Transforming Conceptual Data Models into an Object Model [HG92]

Rudolf Herzig, Martin Gogolla

In this paper a conceptually simple structural object model focusing on object types, attributes and ISA relationships is introduced. The model is derived mainly from an extended Entity-Relationship approach, but concepts from other semantic and object-oriented models have influenced its features. It is shown how high-level conceptual data models can be mapped to this model, and to what extent the object model subsumes classical modeling paradigms.

A SQL-like Query Calculus for Object-Oriented Database Systems [HG94a]

Rudolf Herzig, Martin Gogolla

Currently much effort is being spent on providing object-oriented databases with ad hoc query facilities. In this paper we present a SQL-like query calculus whose major contribution lies in its inherent orthogonality and rigorous mathematical foundation. The calculus is essentially a calculus of complex values but it is defined independently of any concrete database model. The calculus can be used to formulate queries in value-based and object-based data models. Moreover it provides a general facility for the manipulation of complex values.

An Animator for the Object Specification Language [HG94b]

Rudolf Herzig, Martin Gogolla

Techniques for the development of reliable information systems on the basis of their formal specification are the main concern in our project. Our work focuses on the specification language TROLL *light* which allows to describe the part of the world to be modeled as a community of concurrently existing and communicating objects. Our specification language comes along with an integrated, open development environment. The task of this environment is to give support for the creation of correct information systems. One important ingredient of the environment to be described here in more detail is the TROLL *light* animator.

Spezifikation einer Fertigungszelle in TROLL *light* [HV93]

Rudolf Herzig, Nikolaos Vlachantonis

Im Rahmen des KorSo-Verbundprojektes werden Fallbeispiele für die praxisnahe Untersuchung der theoretischen Ergebnisse der einzelnen Partner herangezogen. In dieser Arbeit wird TROLL *light* zur Spezifikation einer Fertigungszelle benutzt. Ein Modell dieser Fertigungszelle wird vom FZI Karlsruhe zur Verfügung gestellt.

Specification of a Production Cell with TROLL *light* [HV94]

Rudolf Herzig, Nikolaos Vlachantonis

In the KorSo project several case studies are made to test the practicability of certain methods. In this report the language TROLL *light* is used for the specification of a production cell. A model of the production cell is provided by the FZI Karlsruhe.

An Approach towards the Conceptual Modelling of the Tool Integration Aspect [VH93]

Nikolaos Vlachantonis, Peter Hartel

In this paper we describe a conceptual model which is purposed for the modelling of the control integration aspect of Software Engineering Environments (SEE). We present a graphical notation for this model under the notion of Tool/Document (T/D) model which is based on predicate/transition Petri nets. It has therefore a formal semantics and can be further processed for reasons of validation and implementation. Moreover we embed this model methodically in the development process of SEEs by taking also into account that such systems must be integrated into companies with different structures, processes, and requirements. This is one of the reasons why we think that it is necessary to first model and validate relevant parts of a SEE in order to minimize costs and risks. As a case study for this model we use parts of the object-oriented CASE tool OBLOG-CASE.

Towards Reliable Information Systems: The KorSo Approach [VHG+93]

Nikolaos Vlachantonis, Rudolf Herzig, Martin Gogolla, Grit Denker, Stefan Conrad, Hans-Dieter Ehrich

Within the compound project KorSo our team is concerned with the research on techniques and methods for the development of reliable information systems on the basis of formal specifications. Our work focuses on the specification language TROLL *light* which allows to describe the part of the world which is to be modeled as a community of concurrently existing and communicating objects by determining their structure as well as their behavior. Moreover we develop and implement a computer aided specification environment for TROLL *light* which permits a prototyping animation as well as the proof of properties of specifications.

Ideen zur Architektur einer offenen und integrierten Entwurfsumgebung für Informationssysteme [Vla92]

Nikolaos Vlachantonis

Der Entwurf und die Implementierung von Informationssystemen ist ein sehr komplexer Arbeitsprozeß, der von einem integrierten Softwaresystem unterstützt werden sollte, um in vielerlei Hinsicht effizient und korrekt durchgeführt werden zu können. Zu diesem Zweck werden in dieser Arbeit erste Überlegungen zu den Anforderungen, zum möglichen konzeptionellen Aufbau und zu den dazugehörenden Komponenten eines

CASE (Computer Aided Software Engineering) Systems für diese Aufgaben präsentiert. Die Konzeption dieses CASE Systems basiert auf der Idee voneinander unabhängiger Tools (Softwarewerkzeuge), die aber bzgl. der Aspekte Kommunikation, Datenbank (Entwurfsdokumente) und Benutzeroberfläche eng miteinander integriert sind. Im besonderen stehen die Ziele Austauschbarkeit von Tools und Erweiterbarkeit des CASE Systems im Mittelpunkt.

A Case Study for an Open CASE System: The TROLL *light* Development Environment [Vla93]

Nikolaos Vlachantonis

In this paper, the TROLL *light* development environment is described as a special computer aided software engineering (CASE) system. In particular, it serves for the formal development of information systems on the basis of the object-oriented specification language TROLL *light* which in turn allows to describe the Universe of Discourse (UoD) as a community of concurrently existing and communicating objects. TROLL *light* is especially suited for the specification of information systems but is not restricted to this application field. Within the compound project KorSo our team is concerned with the research on techniques and methods for the development of reliable information systems on the basis of formal specifications.

Logische Modellierung der Steuerungs-Integration in offenen Softwareentwicklungssystemen [Vla94]

Nikolaos Vlachantonis

Die Entwicklung von komplexen Software-Systemen kann heutzutage nur noch unter Einsatz von Hilfsmitteln stattfinden. Diese Hilfsmittel, die sogenannten Tools (Werzkeuge), sind wiederum selbst Software-Systeme. Sie unterstützen die verschiedenen Arbeitsphasen der Software-Entwicklung, indem sie den Software-Entwicklern als “Werkzeuge” bei der Durchführung ihrer Aufgaben dienen. Sie vereinfachen die Erstellung komplizierter Entwurfs- und Designdokumente, führen Konsistenztests durch, können Transformationen von Dokumenten vornehmen, etc. Meist wird eine Vielzahl von Tools benötigt, da Software-Entwicklung i.a. ein sehr umfangreicher Prozeß ist, der viele unterschiedliche Teilaufgaben umfaßt.

Beim Einsatz vieler unterschiedlicher Tools entsteht das Problem ihrer korrekten Handhabung. Die Software-Entwickler müssen die Tools in einer korrekten Abfolge anwenden und müssen dafür Sorge tragen, daß alle Ergebnisse konsistent sind und fehlerhafte nicht weiter benutzt werden. Sie sind somit nicht mehr nur mit der Software-Entwicklung selbst beschäftigt sondern auch mit organisatorischen Aufgaben. Dieser Sachverhalt führt erfahrungsgemäß sehr leicht zu Fehlern, die sich auf die Qualität der erstellten Software niederschlagen.

Ein weiteres Problem ist die Kompatibilität der Tools untereinander. Das beudeutet, daß die mit Hilfe eines Tools erstellten Dokumente (Ergebnisse) oft nicht ohne manuelle Anpassungen und Änderungen von anderen dafür vorgesehenen Tools weiterverarbeitet werden können. In solchen Fällen sind die Tools nicht vernünftig "integriert", und es ergeben sich weitere potentielle Fehlerquellen bei der Software-Entwicklung.

Um diese Probleme zu vermeiden, werden CASE-Systeme (Computer Aided Software Engineering) bzw. Software-Entwicklungssysteme eingesetzt, die die unterschiedlichen Tools integrieren und organisatorische Arbeiten übernehmen. Leider haben sich Software-Entwicklungssysteme bisher als nicht sonderlich erfolgreich erwiesen. Die Gründe dafür sind u.a. eine mangelnde Integrationsfähigkeit für die unterschiedlichen Tools, eine schlechte Erweiterbarkeit um neue Tools und nur wenige Möglichkeiten zur Anpassung an die lokalen Gegenbenheiten des Einsatzortes. Daher werden derzeit verstärkt Möglichkeiten zur Verbesserung der Qualität von Software-Entwicklungssystemen erforscht.

Die vorliegende Arbeit ist diesem Forschungsbereich zuzuordnen. Es wird ein Verfahren mit der zugehörigen Technik (Tool/Dokumenten-Modell) entwickelt, das die logische und implementierungsunabhängige Modellierung eines Integrationsaspekts, der Steuerungs-Integration von Tools, innerhalb von Software-Entwicklungssystemen ermöglicht. Damit läßt sich das Zusammenspiel der Tools vorab beschreiben und durch Simulationstechniken auf Schwachpunkte hin untersuchen. Ferner wird dadurch die Anschaffung und Anpassung existierender und die Entwicklung neuer Tools erleichtert.

Das genannte Verfahren wird darüberhinaus methodisch in die Konzeption von Software-Entwicklungssystemen eingebettet. Dazu wird es in ein Vorgehensmodell zur Entwicklung offener Software-Entwicklungssysteme integriert, und es wird gezeigt, wie der Dokumentenaustausch zu den benachbarten Entwicklungsphasen durchgeführt werden kann.

4.2 List of Talks

We also list the talks which have been given by members of our project ordered by date.

- H.-D. Ehrich. *Konzepte der Objekt-Orientierung.* Hauptvortrag auf dem 2. Workshop Informationssysteme und Künstliche Intelligenz, FAW Ulm, 24.2.92.
- M. Gogolla. *Towards TROLL light — An Object-Oriented Specification Language.* COMPASS Subgroup Meeting on Object-Orientation and Concurrency, TU Berlin, 26.-27.3.1992.
- G. Denker. *Conditional Term Rewriting — A Model of Concurrency.* COMPASS Subgroup Meeting on Object-Orientation and Concurrency, TU Berlin, 26.-27.3.1992.
- M. Gogolla. *Überblick über Konzepte von TROLL light.* 2. KORSO-Workshop, Meckenheim, 30.3.-1.4.92.
- M. Gogolla. *TROLL light — Ein Ansatz zur objektorientierten Spezifikation.* KORSO-Kolloquium, TU Berlin, 18.5.1992.
- G. Denker. *Towards Semantic Foundation of Basic TROLL light Constructs.* 4. GI-Workshop "Grundlagen von Datenbanken", Barsinghausen, 9.-12.6.1992.
- N. Vlachantonis. *Architektur einer integrierten Entwurfsumgebung für Informationssysteme.* 4. GI-Workshop "Grundlagen von Datenbanken", Barsinghausen, 9.-12.6.1992.
- R. Herzig. *Spezifikation einer Fertigungszelle in TROLL light.* Treffen der KORSO-AG Fallstudien, TU Berlin, 6.7.92.
- H.-D. Ehrich. *Begriffe und Konstruktionen für Objekt-Orientierte Informationssysteme.* Kolloquium, TU München 21.7.92.
- H.-D. Ehrich. *Designing Information Systems as Communities of Objects.* Hauptvortrag auf der 3rd Int. Conf. on Database and Expert Systems Applications DEXA'92, Valencia 2.9.92.
- M. Gogolla. *Braunschweiger Aktivitäten im KORSO Projekt.* Treffen der südostniedersächsischen Datenbankgruppen, Clausthal-Zellerfeld, 8.9.92.
- S. Conrad. *Zur Entwicklung eines Verifikationskalküls für Objektspezifikationen.* Treffen der südostniedersächsischen Datenbankgruppen, Clausthal-Zellerfeld, 8.9.92.
- R. Herzig. *Transformation konzeptioneller Datenmodelle in ein Objektmodell.* Treffen der südostniedersächsischen Datenbankgruppen, Clausthal-Zellerfeld, 8.9.92.

- S. Conrad. *Zur Entwicklung eines Verifikationskalküls für TROLL light.* 3. KORSO-Workshop, Gaggenau, 14.-16.9.92.
- R. Herzig. *Spezifikation von Objekten in TROLL light.* 3. KORSO-Workshop, Gaggenau, 14.-16.9.92.
- G. Denker. *Towards a Semantical Foundation of TROLL light.* IS-CORE Subgroup Meeting, Lissabon, 5.-6.10.1992.
- R. Herzig. *Transforming Conceptual Data Models into an Object Model.* 11th Int. Conf. on the Entity-Relationship Approach (ERA), Karlsruhe, 8.10.92.
- S. Conrad. *On Transforming Object Specifications into a Certification Calculus.* 9th ADT-Workshop, Caldes de Malavella, 26.-30.10.92.
- G. Denker. *Translating TROLL light Concepts to Maude.* 9th ADT-Workshop, Caldes de Malavella, 26.-30.10.92.
- M. Gogolla. *Entwicklung zuverlässiger Informationssysteme.* Informatik-Kolloquium, Universität Karlsruhe, 23.11.1992.
- N. Vlachantonis. *Instanziierung des KORSO-Methodikrahmens anhand der Methodik der TROLL light Entwicklungsumgebung.* Treffen der KORSO-AG Methodik, Bremen, 26.11.92.
- M. Gogolla. *Ein Ausführungsmodell für TROLL light.* Treffen der KORSO-AG Sprachen, Braunschweig, 2.12.92.
- R. Herzig. *Ein Anfragekalkül für TROLL light.* Treffen der KORSO-AG Sprachen, Braunschweig, 2.12.92.
- M. Gogolla. *Towards Semantics of TROLL light on the Basis of Graphs.* Graph-Grammatik Kolloquium, TU Berlin, 19.1.1993.
- S. Conrad. *Die Datenbankkomponente am Beispiel eines vereinfachten, relationalen Datenbanksystems.* KORSO-Arbeitstreffen zur HDMS-A-Fallstudie, TU Berlin, 20.1.93.
- S. Conrad. *Zur Modelltheorie des TROLL light Verifikationskalküls.* 4. KORSO-Workshop, München, 17.3.-19.3.93.
- H.-D. Ehrich. *Specifying Information Systems as Object Communities.* Leibniz Workshop on Specification Technology and Information Systems, The Hebrew University, Jerusalem, 22.3.93.
- H.-D. Ehrich. *Constructing Systems as Object Communities.* TAPSOFT'93 (Invited Lecture), Paris-Orsay, 14.4.93.

- M. Gogolla. *TROLL light — A Core Language for Specifying Objects.* Dagstuhl-Seminar on Formal Aspects of Object-Based Dynamics, 26.4.1993.
- M. Gogolla. *A Computational Model for TROLL light.* Dagstuhl-Seminar on Specification and Semantics, 24.5.1993.
- S. Conrad. *Zur Verifikation von Objekteigenschaften.* 5. GI-Workshop "Grundlagen von Datenbanken", Graal-Müritz, 1.-4.6.93.
- N. Vlachantonis. *Towards Reliable Information Systems: The KORSO Approach.* Int. Conf. on Advanced Information Systems Engineering (CAISE), Paris, 8.-11.6.93.
- M. Gogolla. *Operational Semantics for TROLL light.* 4. COMPASS Subgroup Meeting on Object-Orientation and Concurrency organized together with COMPUGRAPH, L'Aquilla, 14.6.1993.
- N. Vlachantonis. *Case Study for an Open CASE System: The TROLL light Development Environment.* Int. Conf. on Database and Expert Systems Applications (DEXA), Prag, 6.-8.9.93.
- M. Gogolla. *Sketching Concepts and Computational Model of TROLL light.* Int. Conf. on Design and Implementation of Symbolic Computation Systems (DISCO), Gmunden, 15.-17.9.93.
- G. Denker. *On Reification of Object-Oriented Specifications.* 5th COMPASS Workshop, Dresden, 20.9.-23.9.93.
- R. Herzig. *Zur Entwicklung zuverlässiger Informationssysteme in KORSO.* 23. GI-Jahrestagung, Dresden, 29.9.-1.10.93.
- M. Gogolla. *Die TROLL light Entwicklungsumgebung: Stand der Arbeiten im Oktober'93.* 5. KORSO-Workshop, Gosen, 13.-15.10.93.
- H.-D. Ehrich. Formale *Spezifikation von Informationssystemen.* Kolloquium, Universität Bremen, 15.12.93.
- M. Gogolla. *Integrating the ER Approach in an OO Environment.* Int. Conf. on the Entity-Relationship Approach (ERA), Arlington (Texas), 15.-17.12.93.
- R. Herzig. *Compositional Description of Object Communities with TROLL light.* Int. Basque Workshop on Information Technology (BIWIT): Information Systems Design and Hypermedia, Biarritz, 8.2.94.
- S. Conrad. *Proof Support for TROLL light.* COMPASS/IS-CORE-Workshop on Models and Specification Formalisms for Objects and Processes, Lissabon, 21.-23.02.94.

- G. Denker. *Transactions in Object Oriented Specifications.* COMPASS/IS-CORE-Workshop on Models and Specification Formalisms for Objects and Processes, Lissabon, 21.-23.02.94.
- R. Herzig. *Zur Implementierung des TROLL light-Animationssystems.* 6. KORSO-Workshop, Bremen, 28.02.-02.03.94
- N. Vlachantonis. *Spezielle Aspekte der TROLL light-Entwicklungsumgebung.* 6. KORSO-Workshop, Bremen, 28.02.-02.03.94
- R. Herzig. *Zur Gestaltung der Benutzungsschnittstelle bei der Animation von Objektspezifikationen.* GI-Workshop "Benutzungsschnittstellen für Datenbanken", Frühjahrstreffen der GI-Fachgruppe Datenbanksysteme, Kassel, 17.3.94.
- M. Gogolla. *An Animator for the Object Specification Language TROLL light.* Colloquium on Object Orientation in Databases and Software Engineering, Université du Quebéc à Montréal, 16.5.94.
- G. Denker. *Modules as Units For Reification.* ISCORE Working Meeting on Modularization, Orsay, 7.4.-8.4.94.
- S. Conrad. *An Elementary Logic for Object Specification and Verification.* GI-Workshop "Formale Grundlagen für den Entwurf von Informationssystemen", Ev. Akademie Tutzing, 26.5.94.
- G. Denker. *Action Reification in the Context of Concurrency Control.* 10th Workshop on Abstract Data Types (ADT), Genua, 30.5.-3.6.94.
- M. Gogolla. *An Algebraic Semantics for the Object Specification Language TROLL light.* 10th Workshop on Abstract Data Types (ADT), Genua, 30.5.-3.6.94.

4.3 Abstracts of Students' Theses

Students played an important part in the project. They did their theses in cooperation with project members. All theses are written in German. Therefore the abstracts are given in German as well.

Umsetzung von TROLL *light* Spezifikationen nach C++ am Beispiel einer Fertigungszelle, Studienarbeit, Dezember 1992

Niels Abert, Jens Siemer

Ziel dieser Arbeit ist es, Methoden zur Umsetzung von TROLL *light* Spezifikationen in C++-Programme am Beispiel einer Fertigungszelle zu entwickeln. TROLL *light* dient der formalen und konzeptionellen Modellierung von Informationssystemen. Dabei werden statische und dynamische Aspekte des zu modellierenden Weltausschnitts in einer abstrakten Spezifikation beschrieben. Um den Zusammenhang zwischen konzeptionellem Modellieren und Implementieren von Informationssystemen aufzuzeigen, wird TROLL *light* mit der objektorientierten Programmiersprache C++ syntaktisch und semantisch verglichen. Dazu wird C++ in seinen wichtigsten Merkmalen beschrieben, die eine Ähnlichkeit zu TROLL *light* erkennen lassen, um Ansatzpunkte für die Umsetzung von TROLL *light* Spezifikationen in C++-Programme zu finden.

Übersetzung von TROLL *light* Objektbeschreibungen in Formeln eines Verifikationskalküls, Diplomarbeit, März 1994

Niels Abert

Ziel dieser Arbeit ist es, Transformationsvorschriften zur Übersetzung von TROLL *light* Objektbeschreibungen in Formeln des TROLL *light* Verifikationskalküls zu implementieren. Dazu ist der Entwurf von Methoden zur Implementierung der Vorschriften notwendig. TROLL *light* dient zur konzeptionellen Modellierung von Informationssystemen. Mit Hilfe einer abstrakten Spezifikation beschreibt man die statischen und dynamischen Aspekte des zu modellierenden Weltausschnittes. Der betrachtete Verifikationskalkül dient zur Beschreibung bzw. zum Herleiten von Objekt-Eigenschaften mit Hilfe von Formeln. Die Formeln werden durch Transformationsvorschriften aus der gegebenen Objektbeschreibung gewonnen. Die Implementierung dieser Vorschriften erfolgt mit Hilfe der C++-Klassen des objektorientierten Datenbankmanagementsystems ObjectStore und den Prolog-Prädikaten des QuintusProlog-Entwicklungssystems.

Implementierung eines Term- und Formelauswertes für TROLL *light* Spezifikationen, Diplomarbeit, September 1993

Michael Alexander

In dieser Arbeit wird die Implementierung eines Term- und Formelauswerters dargestellt. Die Terme und Formeln sind Teil eines Anfragekalküls für komplexe Werte. Der Kalkül ist wiederum Teil der objektorientierten Spezifikationssprache für Informationssysteme TROLL *light*. Neben den üblichen Operationen und Prädikaten stellt der Kalkül mit select-Termen eine SQL-ähnliche, vollkommen orthogonale Anfragemöglichkeit zur Verfügung. Bei einer prototypischen Ausführung der TROLL *light* Spezifikationen können sowohl ad hoc- als auch vordefinierte Anfragen an Objektgesellschaften gestellt werden. Diese Anfragen werden mit dem Term- beziehungsweise Formelauswerter berechnet. Für die Auswertung ist ein rekursiver Algorithmus implementiert worden. Auf Grund der Orthogonalität des Kalküls ist die Bildung von beliebig komplexen Termen und Formeln nur durch die Systemressourcen beschränkt. Bei der Implementierung wurde insbesondere auf die Funktionalität und weniger auf Effizienz Wert gelegt.

Spezifikation der elektronischen Patientenakte HDMS-A: Eine TROLL *light* Fallstudie, Studienarbeit, Dezember 1993

Elke Becker

Bei der Entwicklung von Informationssystemen werden verschiedene Entwurfsphasen durchlaufen. Zu diesen Entwurfsphasen gehören im Datenbankentwurf Requirements-Engineering und Design-Engineering. Requirements-Engineering dient der Ermittlung der Anforderungen an das zu entwickelnde Produkt. Als Ergebnis oder Produkt des Requirements-Engineering erhält man ein konzeptionelles Modell einer Anwendung. Weitere Entwicklungsschritte fallen unter Design-Engineering, wo die Implementierung eines konzeptionellen Modells festgelegt wird. Das Ergebnis dieser Entwicklungsphase ist ein arbeitsfähiges System. Dieser Studienarbeit liegt die Aufgabe zugrunde, ein konzeptionelles Modell einer Datenbankanwendung zu spezifizieren. Insbesondere wird hier der Entwurf einer elektronischen Patientenakte betrachtet. Dabei wird in objektorientierten Entwicklungsschritten vorgegangen. Als Ergebnis dieser Entwicklungsschritte erhält man die Spezifikation einer elektronischen Patientenakte in der Sprache TROLL *light*. Berücksichtigt wird dabei nicht nur der formale Aufbau der Akte, sondern auch die Anwendungen, die auf dieser Akte stattfinden können.

Vergleichende Betrachtung von Verfeinerungsbegriffen im Bereich formale Spezifikation, Diplomarbeit, Februar 1993

Katja Beermann

Bei der Entwicklung von Softwaresystemen ist es von entscheidender Bedeutung, daß die erzeugten Programme exakt die Aufgabenstellung erfüllen. Hierzu werden formale Methoden benötigt, die das Problem erfassen und die Anforderungen genau definieren. Diese formale Beschreibung der Anwendung ist in der Spezifikation festgehalten, die durch Verfeinerung in ein ausführbares Programm transformiert wird. Drei verschiedene Verfeinerungstechniken sind Gegenstand der vorliegenden Betrachtung. Zunächst wird die grundlegende Idee der jeweiligen Arbeit aufgezeigt. Die Arbeit schließt mit einem anhand eines gemeinsamen Beispiels gestellten Vergleich dieser unterschiedlichen Verfeinerungsbegriffe.

Implementierung des TROLL *light* Ausführungsmodells, Diplomarbeit, September 1993

Jens Brinckmann

TROLL *light* ist eine Sprache, mit der ein Ausschnitt der realen Welt modular als eine Objektgesellschaft beschrieben werden kann. Anhand von Ereignissen werden Zustandsveränderungen der Objekte und hiermit der Objektgesellschaft, in der sie sich befinden, bewirkt. Es können gleichzeitig in mehreren, paarweise verschiedenen Objekten Ereignisse stattfinden, die miteinander synchronisiert sein können. Die Ausführung und die Zulässigkeit von Ereignissen regelt das TROLL *light* Ausführungsmodell. Bestandteil der TROLL *light* Entwicklungsumgebung ist ein Animator, mit dem Objektspezifikationen prototypisch ausgeführt werden können. Diese Arbeit beschreibt die Implementierung von Teilen des Animators zur Realisierung des TROLL *light* Ausführungsmodells. Die Einheiten sind eine Bibliothek zur strukturierten Speicherung und Verwaltung von Objektbeschreibungen, eine Bibliothek zur strukturierten Speicherung, Verwaltung und Manipulation einer Objektgesellschaft und ein Modul zur Ausführung von Objektereignissen.

Kontextsensitive Analyse von TROLL *light* Objektbeschreibungen, Diplomarbeit, Juni 1994

Matthias Brocks

In dieser Arbeit wird die Implementierung eines Übersetzers dargestellt. Zur Speicherung von Objektbeschreibungen der Spezifikationssprache TROLL *light* existieren eine Datenbankschnittstelle, die sich an der konkreten Syntax orientiert, sowie eine, die an der abstrakten Syntax angelehnt ist. Die Übersetzung der konkreten Darstellung in die abstrakte ist mit einer kontextsensitiven Analyse verbunden.

Vergleich von Entwicklungsumgebungen für algebraische Spezifikationen, Diplomarbeit, Oktober 1993

Rainer Conrad

Algebraische Spezifikation ist eine der am besten vorangetriebenenn formalen Spezifikationstechniken. Die ersten Entwicklungsumgebungen wurden in den späten Siebziger Jahren entwickelt. In dieser Arbeit werden die fünf Systeme OBJ3, ACT, ASSPEGIQUE+, RAP und OBSCURE miteinander verglichen. Die Hauptziele dieses Vergleichs sind die Betrachtung der Ausdrucksfähigkeit der algebraischen Spezifikationssprachen, der Komfort, d.h. die Funktionalität der Umgebungen, die Effizienz und die Benutzerfreundlichkeit. Ohne diese Punkte ist eine Umgebung nicht für wirkliche Spezifikationen nutzbar. Wir mußten an jedem der fünf Systeme Kritik anbringen. Die Ergebnisse dieser Arbeit könnten einerseits benutzt werden, um die bestehenden Systeme zu verbessern und andererseits, um ein neues System zu entwickeln.

Implementierung der graphischen Animation von TROLL *light* Objekten, Diplomarbeit, März 1994

Heiko Fischer

Mit der Spezifikationssprache TROLL *light* können konzeptionelle Objekte einer Objektgesellschaft formal in ihrer Struktur und in ihrem Verhalten zueinander beschrieben werden. Bildet die Objektgesellschaft die Grundlage für ein aufwendiges Informationssystem, ist es wichtig, diese zuvor prototypisch auszutesten. Dadurch wird zwar keine vollständige Korrektheit der Spezifikation gewährleistet, aber es kann doch zumindest eine gewisse Validierung erreicht werden. Beim Testen sollte der Anwender Ereignisse in seinem Modell initiieren und die Reaktionen darauf beobachten können. Man spricht hierbei auch von Animation. Ein die

Animation unterstützendes Software-Werkzeug wird als Animationssystem oder kurz Animator bezeichnet. In dieser Arbeit wird für einen Animator eine graphische Fensteroberfläche als Benutzungsschnittstelle zwischen Anwender und formaler Spezifikation entworfen und implementiert.

Eine Fallstudie zur Übersetzung einer TROLL *light* Spezifikation nach L&O, Studienarbeit, April 1994

Sedat Güler

Ziel dieser Studienarbeit ist es, die Übertragbarkeit der Konzepte von TROLL *light* nach L&O zu prüfen. L&O ist quasi ein objektorientiertes Prolog. Vorteil von Prolog ist, daß diese Sprache sich besonders gut für die Übertragung abstrakter Formalismen in eine Programmiersprache eignet. Im ersten Kapitel werden zunächst die Erweiterungen von L&O zum konventionellen Prolog vorgestellt sowie die Konzepte von TROLL *light* näher beschrieben. Inhalt des zweiten Kapitels ist die Übersetzung eines TROLL *light* Beispiels nach L&O und die Beschreibung der dabei auftretenden Probleme.

Ansätze zur Spezifikation von reaktiven Systemen – Eine vergleichende Betrachtung, Diplomarbeit, März 1994

Adnaan Gorial

Die Arbeit befaßt sich mit einem Vergleich von Ansätzen zur Spezifikation reaktiver Systeme. Nach einer Einführung in die Arbeitsweise reaktiver Systeme werden einige Ansätze detailliert betrachtet. Dabei handelt es sich um CCS, CSP, ACP, Ströme sowie den polyadischen π-Kalkül aus dem Bereich der algebraischen Ansätze, Statecharts und Petrinetze aus dem Bereich der graphischen Ansätze sowie UNITY und eine Form der temporalen Logik aus dem Bereich der Ansätze in temporler Logik. Im Anschluß an die Darstellung der verschiedenen Ansätze findet sich ein Vergleich der Ansätze, der sich schwerpunktmäßig auf die Bereiche Prozeßdarstellung, Kommunikation, Nichtdeterminismus, Verklemmungsverhalten und Abstraktion bezieht.

Implementierung einer algebraisch spezifizierten Beispielprogrammiersprache in PROLOG, Studienarbeit, April 1993

Kais Gulidjan

Diese Studienarbeit stellt die Implementierung einer mittels einer algebraischen Spezifikation beschriebenen, einfachen Beispielprogrammiersprache vor. Als Implementierungssprache wurde die logische Programmiersprache PROLOG gewählt. Dies ist insofern besonders effektiv, da im wesentlichen nur jede Gleichung in eine äquivalente Klausel übertragen werden muß. Das entstehende PROLOG-Programm terminiert für alle Eingaben, da die zugrundeliegende algebraische Spezifikation die Finite-Church-Rosser-Eigenschaft besitzt.

Objektorientierte Datenbanksysteme — Übersicht und Vergleich, Diplomarbeit, Januar 1993

Hartwig Hake

Die Aufgabenstellung zu dieser Diplomarbeit war, zunächst einen Überblick über objektorientierte Datenbanksysteme zu geben und anschließend diese zu vergleichen. Dabei sollten gleichermaßen kommerzielle Systeme als auch experimentelle Systeme berücksichtigt werden. In dieser Arbeit werden vor allem kommerziell verfügbare Systeme betrachtet: O_2, GemStone, Itasca (ORION), ObjectStore, ONTOS, POET und Iris. Zusätzlich werden drei experimentelle Systeme (MANDRILL, JASMINE und ODIN) kurz behandelt. Alle behandelten Systeme sind auf der Grundlage von Veröffentlichungen in internationalen Journalen und auf Konferenzen und — soweit zugänglich — von Produktinformationen der vertreibenden Firmen beschrieben worden. Die Arbeit beginnt mit einem kurzen Überblick über die verwendeten Begriffe, da insbesondere im Bereich der Objektorientierung viele Begriffe sehr uneinheitlich verwendet werden. Es folgt die Beschreibung der oben aufgezählten, kommerziell verfügbaren objektorientierten Datenbanksysteme. Es wurde hier eine möglichst einheitlich strukturierte Beschreibung der Systeme angestrebt, um den späteren Vergleich zu erleichtern. In der Regel sind für jedes System die Architektur, das Datenmodell, Anfragesprachen, Programmiersprachen, Transaktionsverwaltung und die zugehörige Entwicklungsumgebung beschrieben worden. Daneben sind die Besonderheiten, die einzelne Systeme bieten, auch berücksichtigt worden. Ein kurzer Überblick über drei experimentelle Systeme schließt die Beschreibung von objektorientierten Datenbanksystemen ab. Nach einem Vergleich der beschriebenen Systeme anhand der oben aufgeführten Merkmale, wird die Arbeit mit einer Zusammenfassung und einer Einschätzung der zukünftigen Entwicklung objektorientierter Datenbanksysteme beendet.

Konzeption und Implementierung einer Benutzeroberfläche für die TROLL *light* Entwicklungsumgebung, Studienarbeit, Juli 1993

Oliver Hellmold, Lars Schulz

In dieser Studienarbeit wird eine dynamische und integrierende Benutzerschnittstelle für die TROLL *light* Entwicklungsumgebung konzipiert und implementiert. Hierzu erfolgt unter Berücksichtigung ergonomischer Aspekte die Gestaltung einer graphischen Benutzeroberfläche, in die der Benutzer die von ihm gewünschten Werkzeuge der TROLL *light* Entwicklungsumgebung einbinden und während der Laufzeit bedienen kann. Die Implementierung wird unter Zuhilfenahme des Toolkits Tk sowie der ihm zugrundeliegenden Programmiersprache Tcl realisiert.

Konzepte und Methoden in der objektorientierten Analyse und Modellierung, Diplomarbeit, November 1993

Hartmut Herde

Diese Diplomarbeit ist inhaltlich einzuordnen im Bereich der Methodologie des Software Engineering bzw. Requirements Engineering. Sie beinhaltet eine exemplarische Darstellung und einen Vergleich von drei ausgewählten Methoden zur Verwendung in der ersten Phase des Software-Lebenszyklus, der Anforderungsanalyse. Die behandelten Methoden sind die Strukturierte Analyse, die Object Modeling Technique und Object Oriented Software Engineering. Als methodische Basis wird ein umfassender Kriterienkatalog entwickelt, der geeignet ist, die übersichtliche und umfassende Darstellung der Methoden, ihrer Sprachen, Inhalte und Vorgehensweisen zu ermöglichen. Die Kriterien umfassen Gesichtspunkte der verwendeten Notationen und ihrer Konzepte, die zugrundeliegenden Vorgehensmodelle und anwendungsorientierte Gesichtspunkte wie Dokumentationsumfang und Werkzeugunterstützung. Auf Fallstudien zur Beschreibung der Methoden baut ein Vergleich auf, der Stärken und Schwächen der bearbeiteten Ansätze aufzeigt.

Der Theorembeweiser Tatzelwurm, Studienarbeit, August 1992

Thorsten Lüdtke

In der Programmverifikation werden Theoreme erzeugt, die für die Korrektheit eines Programms hinreichend sind. Im allgemeinen sind diese Theoreme sehr umfangreich oder ihr Beweis ist sehr lang. Deshalb sind automatische Hilfen notwendig, um diese Theoreme zu erzeugen und zu beweisen. Der Tatzelwurm enthält einen Beweiser für die Prädikatenlogik erster Stufe, der auf der Methode des Analytischen Tableaus basiert. Die Arbeit ist folgendermaßen aufgebaut: Nach einer kurzen Präsentation der Terminologie wird die Arbeit des Beweisers erläutert. Anhand von Beispielen wird erklärt, wie der Beweiser während des Beweises eines Theorems kooperierende Entscheidungsprozeduren nutzt. Schließlich folgen eine kurze Systembeschreibung und einige Beweise.

Entwicklung einer Transformation von TROLL *light* Formeln in einen Zertifikationskalkül, Studienarbeit, April 1993

Matthias Meyer

Im Rahmen dieser Arbeit erfolgte die Entwicklung eines Regelsystems, das es ermöglicht, gegebene logische Formeln in sogenannte Gentzen-Formeln zu überführen. Logische Formeln dienen der Formulierung z.B. von Constraints, die Objekte einer Spezifikation in der Spezifikationssprache TROLL *light* erfüllen sollen. Es werden verschiedene Kalküle der Prädikatenlogik erster Stufe in Betracht gezogen, die sich als Grundlage für eine schrittweise automatische Transformation eignen. Angestrebtes Ziel ist ein Algorithmus, der das Grundgerüst für eine Implementierung ist.

Integration des Theorembeweisers ISABELLE in die TROLL *light* Entwicklungsumgebung, Diplomarbeit, März 1993

Nils Notbohm

Ein wesentlicher Punkt bei der Entwicklung von Software-Entwicklungssystemen (CASE-Systemen) ist der Wunsch, die einzelnen Tools, die die verschiedenen Phasen des Entwicklungsprozesses unterstützen, durch Toolintegration zu verbinden. In der vorliegenden Diplomarbeit wird die Integration des Theorembeweisers Isabelle in die TROLL *light* Entwicklungsumgebung beschrieben. Das Ziel der Arbeit ist dabei, Erfahrungen bei der Integration eines Tools in eine Entwicklungsumgebung zu gewinnen, die auf der HP SoftBench basiert. Die HP SoftBench stellt ein Messagekonzept zur Verfügung, mit dem die Kommunikation der verschiedenen Tools realisiert werden kann. Die TROLL *light* Entwicklungsumgebung ist ein Beispiel für ein CASE-System. Sie unterstützt die Entwicklung von Informationssystemen auf der Basis von formalen Spezifikationen in TROLL *light*. Die Spezifikation in TROLL *light* bietet die Möglichkeit, bestimmte Eigenschaft des zu spezifizierenden Informationssystems zu beweisen. Diese Eigenschaften sollen in der TROLL *light* Entwicklungsumgebung mit Hilfe des Theorembeweisers Isabelle formal bewiesen werden. Die Integration von Isabelle in die TROLL *light* Entwicklungsumgebung setzt sich aus drei Aspekten zusammen. Es ist zum einen über die HP SoftBench die Kommunikation mit den anderen Tools der Entwicklungsumgebung zu realisieren. Des weiteren ist von Isabelle auf die gemeinsame Datenbasis, die durch das objektorientierte Datenbankmanagementsystem ObjectStore verwaltet wird, zuzugreifen. Als letzter Aspekt der Integration wird Isabelle mit einer auf dem OSF/Motif-Standard basierenden Benutzerschnittstelle versehen.

Konsistenzprüfungen von Objektspezifikationen in TROLL *light* mit Hilfe eines Tableaubeweisers, Studienarbeit, August 1992

Carsten Paasch

Der Zweck einer objektorientierten Spezifikationssprache wie TROLL *light* besteht darin, eine Möglichkeit zu schaffen, die reale Welt in ein von Computern erkenn- und bearbeitbares Modell zu überführen, indem Eigenschaften und Verhalten von Objekten aus der realen Welt sowie deren Interaktionen untereinander modelliert werden. TROLL *light* ist dabei so schlicht gehalten, daß Überprüfungen von Objektspezifikationen automatisiert erfolgen können. Eine Art solcher Überprüfungen ist die Konsistenzprüfung von Objektspezifikationen, die feststellen soll, ob Spezifikationen in sich logisch widerspruchsfrei sind, und die auftretende Widersprüche gegebenenfalls dem Benutzer mitteilt. Eine solche Konsistenzprüfung wird im Rahmen dieser Studienarbeit mit Hilfe des

Tableaubeweisers Tatzelwurm durchgeführt. Dazu müssen die zu untersuchenden logischen Formeln zunächst nach bestimmten Regeln zusammengefaßt und in ein syntaktisches Format gebracht werden, welches der Tableaubeweiser Tatzelwurm verstehen kann. Nachdem der Tableaubeweiser Widersprüche in der Spezifikation aufgedeckt hat, müssen diese dem Benutzer mitgeteilt werden. Die Programme für die Konvertierung der Formeln aus den TROLL *light* Objekten und ihre Übergabe an den Tatzelwurm sowie für die Bearbeitung der Ausgaben aus dem Tatzelwurm wurden in der Programmiersprache C unter UNIX implementiert.

Das AGG System: Test und Bewertung beim Einsatz zur graphischen Darstellung von TROLL *light* Spezifikationen, Studienarbeit, Juli 1993

Markus Paeselt

AGG ist ein an der TU Berlin entwickeltes algebraisches Graphgrammatiksystem. Es ist im Rahmen des Projektes COMPUGRAPH (Computing by Graph Transformation) entstanden, welches sich mit dem Einsatz von Graphgrammatiken und Graphtransformationen im Bereich der Informatik beschäftigt. Ziel der vorliegenden Arbeit ist es, daß AGG System auf seine Funktionen hin zu testen und zu bewerten inwieweit man es zur graphischen Darstellung von TROLL *light* Spezifikationen einsetzen kann. Hierzu werden zunächst einmal die verschiedenen Funktionen des Systems untersucht und anhand von Beispielen erläutert. Anschließend werden die wesentlichen Konzepte der objektorientierten Spezifikationssprache TROLL *light* erklärt und exemplarisch in Graphen mit Hilfe von AGG umgesetzt. Es wird ein Algorihmus entwickelt, der eine teilweise automatische Umsetzung von Spezifikationen in Graphen beschreibt. Abschließend wird eine Bewertung gegeben, inwieweit das AGG System zur graphischen Darstellung von TROLL *light* Spezifikationen geeignet ist.

Erfahrungsbericht zur Anwendung des generischen Beweisers Isabelle, Studienarbeit, Januar 1994

Manfred Rose

Mit dem generischen Beweiser Isabelle kann man ein weites Spektrum von Theorien implementieren und in den so entstehenden Systemen formale Beweise führen. Bedingung für die Umsetzbarkeit einer Theorie ist, daß sie sich auf eine bestimmte Variante des getypten λ-Kalküls reduzieren läßt, in der für Konstanten polymorphe Typen zugelassen sind, deren Typvariablen an geordnete Gültigkeitsbereiche gebunden werden. Die Beschreibung einer Theorie umfaßt die Angabe ihres Typsystems, die Auflistung ihrer Konstanten, die Definition ihrer Syntax sowie die Aufzählung ihrer Axiome. Definiert werden Theorien als Erweiterungen oder Vereinigungen bestehender Theorien, wobei die Metalogik von Isabelle als einzige fest installierte Theorie Elementarbaustein jedes Theoriesystems ist. Mit ihren Konstanten werden Definitionen und Schlußregeln formuliert. Ihrem Axiomensystem, einem Ausschnitt aus der intuitionistischen Logik höherer Stufe, liegt eine sehr allgemeine Vorstellung vom Beweisen zugrunde. Hauptbeweisprinzip ist die Resolution, eine Verallgemeinerung der Transitivitätsregel für die Implikation. Sie fügt zwei Schlußregeln nach vorheriger Unifikation der Konklusion der ersten Regel mit einer Prämisse der zweiten Regel zusammen. Beweise können mit Hilfe der Resolutionsregel in zwei Richtungen geführt werden. In besonderem Maße wird der Rückwärtsbeweis unterstützt. Dabei reduziert man die Konklusion des zu beweisenden Theorems durch Resolution mit Axiomen, Annahmen und abgeleiteten Theoremen auf einfache Tautologien oder bereits bewiesene Theoreme. Für die Beweisführung stehen dem Anwender eine interaktive Beweisumgebung, einige allgemeine Beweispakete und eine Vielzahl von Hilfen für die Programmierung von Beweisprozeduren, die Taktiken genannt werden, zur Verfügung. Insbesondere gibt es vordefinierte elementare Taktiken und Kontrollstrukturen realisierende Funktionen, mit denen Taktiken zu neuen Taktiken zusammengesetzt werden können. Zudem sind Theorien verschiedener Logikschulen vordefiniert, wie z.B. die klassische Prädikatenlogik erster und höherer Stufe, eine Modallogik und die konstruktive Typentheorie. Das gesamte System wurde in der funktionalen Programmiersprache ML geschrieben, die auch für den Dialog mit dem Benutzer verbindlich ist. Die vorliegende Arbeit beschreibt anhand von Beispielen Konzepte und Handhabung der Version Isabelle-91 des Beweisers. Erklärt und vorgeführt wird, wie man Theorien definiert, Beweise führt, Beweisprozeduren programmiert und die bereits vorhandenen Beweispakete nutzt.

Eine Schnittstelle zu ObjectStore für die strukturierte Speicherung von TROLL *light* Spezifikationen, Studienarbeit, Dezember 1992

Frank Ruloffs

In dieser Studienarbeit wird eine abstrakte Datenstruktur entwickelt, die es erlaubt Nutzinformationen aus Spezifikationstexten strukturiert abzuspeichern. Diese Datenstruktur wird für die Spezifikationssprache TROLL *light* implementiert. Die persistente Speicherung der Datenstruktur wird mit Hilfe des OODBMS ObjectStore durchgeführt. Außerdem wird eine Schnittstelle entwickelt, die die Zugriffe auf die persistente Datenstruktur verwaltet.

Implementierung eines Parsers für TROLL *light* Spezifikationen, Studienarbeit, Dezember 1992

Frank Schulten

Im Rahmen dieser Studienarbeit wurde ein Parser für die abstrakte Spezifikationssprache TROLL *light* erstellt. Der geparste Spezifikationstext wird im objektorientierten Datenbanksystem ObjectStore abgelegt. Der Parser wurde mit Hilfe der UNIX-Tools YACC und Lex erstellt. Ferner wird eine Einführung in TROLL *light* gegeben.

Realisierung des TROLL *light* Verifikationskalküls mit Hilfe des generischen Theorembeweisers Isabelle, Diplomarbeit, April 1994

Jens Siemer

TROLL *light* dient zum konzeptionellen Entwurf von Objektgesellschaften. Dabei können die Objekte in ihrer Struktur und ihrem Verhalten beschrieben werden. Um die Korrektheit dieses Entwurfs bezüglich des zu modellierenden Weltausschnitts zu gewährleisten, ist es nötig, Eigenschaften überprüfen zu können. Zu diesem Zweck werden TROLL *light* Objektspezifikationen in den TROLL *light* Verifikationskalkül transformiert. Diese Formeln können anschließend mit Hilfe eines geeigneten Beweisers getestet werden. Gegenstand dieser Arbeit ist die Realisierung eines solchen Beweisers für den TROLL *light* Verifikationskalkül. Als Basis wird der generische Theorembeweiser Isabelle verwendet.

Chapter 5

Conclusion

We have presented a sketch of our object description language TROLL *light* and its accompanying environment. The environment supports the development of correct information systems in (1) validating the informal view of the real-world fragment to be modeled against the current conceptual schema and (2) in checking whether desired properties are fulfilled by the specified object descriptions.

However, the system integration and the implemented tools are far from being perfect. For example, the animator does not support user-defined data types, ad-hoc queries, graphical visualization of sub-object relationships, and enhanced visualization of object behavior. The proof support system could be improved by allowing for instance temporal logic formulas to be checked against the specification, and implementation on the basis of ISABELLE would profit from including more proof tactics in order to speed up the inference mechanism. Up to now only the first steps towards integration of the different tools have been implemented and much more work could be done here.

Acknowledgements: Our research activities have been constructively criticized by our Braunschweig and KORSO colleagues. Most of the implementation work has been done by students. Therefore we would like to thank especially N. Abert, M. Alexander, E. Becker, J. Brinckmann, M. Brocks, R. Conrad, H. Fischer, S. Güler, H. Herde, O. Hellmold, M. Meyer, N. Notbohm, C. Paasch, M. Rose, F. Ruloffs, F. Schulten, L. Schulz, J. Siemer, U. Trenkner.

Bibliography

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

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hussmann, D. Nazareth, F. Regensburger, and K. Stølen. The Requirement and Design Specification Language SPECTRUM — An Informal Introduction (2 parts). Technical Report TUM–I9311/9312, TU München, 1993.

[BNPS92] E. Bertino, M. Negri, G. Pelagatti, and L. Sbattella. Object-Oriented Query Languages: The Notion and the Issues. *IEEE Trans. on Knowledge and Data Engineering*, 4(3):223–237, 1992.

[BW90] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambridge University Press, 1990.

[Cag90] M.R. Cagan. The HP SoftBench Environment: An Architecture for a New Generation of Software Tools. *Hewlett Packard Journal*, 41, 1990.

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

[CE94] S. Conrad and H.-D. Ehrich. An Elementary Logic for Object Specification and Verification. In U. Lipeck and G. Vossen, editors, *Workshop Formale Grundlagen für den Entwurf von Informationssystemen, Tutzing*, pages 197–206. Technical Report Univ. Hannover, No. 03/94, 1994.

[CG92] S. Conrad and M. Gogolla. An Annotated Bibliography on Object-Orientation and Deduction. *ACM SIGMOD RECORD*, 21(1):123–132, 1992.

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

[Con93a] S. Conrad. Einbindung eines bestehenden Datenbanksystems in einen formalen Software-Entwicklungsprozeß. In Ehrich [Ehr93a], pages 1–14.

[Con93b] S. Conrad. On Certification of Specifications for TROLL *light* Objects. In H. Ehrig and F. Orejas, editors, *Proc. 9th Workshop on Abstract Data Types – 4th Compass Workshop (ADT'92)*, pages 158–172. Springer, Berlin, LNCS 785, 1993.

[Con93c] S. Conrad. Spezifikation eines vereinfachten Datenbanksystems — ein Beitrag zur HDMS-A–Fallstudie. In Ehrich [Ehr93a], pages 15–26.

[Con93d] S. Conrad. Zur Verifikation von Objekteigenschaften. In B. Thalheim, editor, *5. GI Workshop Grundlagen von Informationssystemen*, pages 40–44. Universität Rostock, Fachbereich Informatik, Bericht 3–93, 1993.

[Con94a] S. Conrad. A Basic Calculus for Verifying Properties of Synchronously Interacting Objects. Informatik-Bericht 94–05, Technische Universität Braunschweig, 1994.

[Con94b] S. Conrad. *Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte*. PhD thesis, Technische Universität Braunschweig, Naturwissenschaftliche Fakultät, 1994.

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

[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.

[DDP93] E. Dubois, P. Du Bois, and M. Petit. O-O Requirements Analysis: An Agent Perspective. In O.M. Nierstrasz, editor, *Proc. European Conf. on Object-Oriented Programming (ECOOP'93)*, pages 458–481. Springer, Berlin, LNCS 707, 1993.

[Den92] G. Denker. Towards Semantic Foundation of Basic TROLL *light* Constructs. In U.W. Lipeck and R. Manthey, editors, *4. GI Workshop Grundlagen von Datenbanken*, pages 22–26. ECRC, München, Technical Report ECRC-92-13, 1992.

[DG93] G. Denker and M. Gogolla. Translating TROLL *light* Concepts to Maude. In H. Ehrig and F. Orejas, editors, *Proc. 9th Workshop on Abstract Data Types – 4th Compass Workshop (ADT'92)*, pages 173–187. Springer, Berlin, LNCS 785, 1993.

[Ear90] A. Earl. A Reference Model for Computer Assisted Software Engineering Environment Frameworks. Technical report, Hewlett-Packard Laboratories, Bristol, England, 1990. Version 4.0 ECMA/TC33/TGRM/90/016.

[EFH83] H. Ehrig, W. Fey, and H. Hansen. ACT ONE: An Algebraic Specification Language with Two Levels of Semantics. Technical Report 83-03, Technische Universität Berlin, 1983.

[EGC+93] H.-D. Ehrich, M. Gogolla, S. Conrad, G. Denker, R. Herzig, and N. Vlachantonis. Entwicklung zuverlässiger Informationssysteme - Stand der Arbeiten im Januar'93. In M. Broy and S. Jähnichen, editors, *Korrekte Software durch formale Methoden*, pages 50–56, 1993. Broschüre herausgegeben anläßlich des Industrietages auf dem 4. KORSO-Workshop.

[EGC+94] H.-D. Ehrich, M. Gogolla, S. Conrad, G. Denker, R. Herzig, and N. Vlachantonis. Entwicklung zuverlässiger Informationssysteme - Stand der Arbeiten im Januar'94. In M. Broy and S. Jähnichen, editors, *Korrekte Software durch formale Methoden*, pages 82–94, 1994. Broschüre herausgegeben anläßlich des Industrietages auf dem 6. KORSO-Workshop.

[EGL89] H.-D. Ehrich, M. Gogolla, and U.W. Lipeck. *Algebraische Spezifikation abstrakter Datentypen – Eine Einführung in die Theorie.* Teubner, Stuttgart, 1989.

[Ehr93a] H.-D. Ehrich, editor. *Beiträge zu KORSO– und TROLL* light*–Fallstudien.* Technische Universität Braunschweig, Informatik-Bericht Nr. 93–11, 1993.

[Ehr93b] H.-D. Ehrich. Entwicklung zuverlässiger Informationssysteme. *Informatik — Forschung und Entwicklung*, 8(3):160–161, 1993. In: Das BMFT-Verbundprojekt "Korrekte Software (KORSO)", M. Broy, S. Jähnichen.

[Ehr94] H.-D. Ehrich. KORSO Reference Languages: Concepts and Application Domains. In M. Broy and S. Jähnichen, editors, *KORSO*. Springer, Berlin, LNCS, 1994.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Springer, Berlin, 1985.

[EM90] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 2: Modules and Constraints.* Springer, Berlin, 1990.

[ESF91] *The ESF Software Bus – An Overview.* Hohenzollerndamm 152, Berlin, 1991.

[FM91] J. Fiadeiro and T. Maibaum. Temporal Reasoning over Deontic Specifications. *Journal of Logic and Computation*, 1(3):357–395, 1991.

[Gal93] J. Gallier. Constructive Logics — Part I: A Tutorial on Proof Systems and Typed λ-Calculi. *Theoretical Computer Science*, 110(2):249–339, 1993.

[Gau84] M.-C. Gaudel. A First Introduction to PLUSS. Technical Report, Université de Paris-Sud, Orsay, 1984.

[GC93] M. Gogolla and I. Claßen. An Object-Oriented Design for the ACT ONE Environment. In M. Nivat, C. Rattray, T. Rus, and G. Scollo, editors, *Proc. 3rd Int. Conf. on Algebraic Methodology and Software Technology (AMAST'93)*, pages 361–368. Springer, London, Workshops in Computing, 1993.

[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.

[GH91] M. Gogolla and U. Hohenstein. Towards a Semantic View of an Extended Entity-Relationship Model. *ACM Trans. on Database Systems*, 16(3):369–416, 1991.

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

[Gog93] M. Gogolla. Some Examples for TROLL *light* Templates. In Ehrich [Ehr93a], pages 51–62.

[GW88] J.A. Goguen and T. Winkler. Introducing OBJ3. Research Report SRI-CSL-88-9, SRI International, 1988.

[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.

[Hen88] M. Hennessy. *Algebraic Theory of Processes*. MIT Press, Cambridge (MA), 1988.

[Her93] R. Herzig. Spezifikation der abstrakten Syntax von TROLL *light*. In Ehrich [Ehr93a], pages 43–49.

[Her94] R. Herzig. *Zur Spezifikation von Objektgesellschaften mit TROLL* light. PhD thesis, Technische Universität Braunschweig, Naturwissenschaftliche Fakultät, 1994.

[HFG94] R. Herzig, H. Fischer, and M. Gogolla. Zur Gestaltung der Benutzeroberfläche bei der Animation von Objektspezifikationen. *Datenbankrundbrief*, 13:43–45, 1994. Proc. Workshop der GI-Fachgruppe Datenbanken.

[HG92] R. Herzig and M. Gogolla. Transforming Conceptual Data Models into an Object Model. In G. Pernul and A M. Tjoa, editors, *Proc. 11th Int. Conf. on the Entity-Relationship Approach (ER'92)*, pages 280–298. Springer, Berlin, LNCS 645, 1992.

[HG94a] R. Herzig and M. Gogolla. A SQL-like Query Calculus for Object-Oriented Database Systems. In E. Bertino and S. Urban, editors, *Proc. Int. Symp. on Object-Oriented Methodologies and Systems (ISOOMS), Palermo (Italy)*, 1994.

[HG94b] 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*, pages 4–17. Université du Quebéc à Montréal, 1994.

[HK87] R. Hull and R. King. Semantic Database Modelling: Survey, Applications, and Research Issues. *ACM Computing Surveys*, 19(3):201–260, 1987.

[Hoa85] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Englewood Cliffs (NJ), 1985.

[HV93] R. Herzig and N. Vlachantonis. Spezifikation einer Fertigungszelle mit TROLL *light*. In Ehrich [Ehr93a], pages 27–41.

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

[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.

[Kro93] K. Kronlöf, editor. *Method Integration: Concepts and Case Studies.* Wiley & Sons, 1993.

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

[Lou92] P. Loucopoulos. Conceptual Modeling. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases, and CASE: An Integrated View of Information Systems Development*, pages 1–26. John Wiley & Sons, New York, 1992.

[Mil80] R. Milner. *A Calculus of Communicating Systems.* Springer, Berlin, 1980.

[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.

[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.

[PM88] J. Peckham and F. Maryanski. Semantic Data Models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[SB93] D. Schefström and G. van den Broek, editors. *Tool Integration: Environments and Frameworks.* Wiley & Sons, 1993.

[SS86] H.-J. Schek and M.H. Scholl. The Relational Model with Relation-Valued Attributes. *Information Systems*, 2(2):105–133, 1986.

[SSE87] A. Sernadas, C. Sernadas, and H.-D. Ehrich. Object-Oriented Specification of Databases: An Algebraic Approach. In P.M. Stoecker and W. Kent, editors, *Proc. 13th Int. Conf. on Very Large Databases VLDB'87*, pages 107–116. VLDB Endowment Press, Saratoga (CA), 1987.

[ST86] D.T. Sannella and A. Tarlecki. Extended ML: An Institution-Independent Framework for Formal Program Development. In *Proc. Workshop on Category Theory and Computer Programming*, pages 364–389. Springer, LNCS 240, 1986.

[STW84] M. Schrefl, A M. Tjoa, and R.R. Wagner. Comparison Criteria for Semantic Data Models. In C.V. Ramamoorthy, editor, *Proc. Int. Conf. on Data Engineering*, pages 120–125. IEEE, Silver Spring (MD), 1984.

[TN92] I. Thomas and B.A. Nejmeh. Definitions of Tool Integration for Environments. *IEEE Software*, pages 29–35, March 1992.

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

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

[Vla92] N. Vlachantonis. Ideen zur Architektur einer offenen und integrierten Entwurfsumgebung für Informationssysteme. In U.W. Lipeck and R. Manthey, editors, *4. GI Workshop Grundlagen von Datenbanken*, pages 157–161. ECRC, München, Technical Report ECRC-92-13, 1992.

[Vla93] N. Vlachantonis. A Case Study for an Open CASE System: The TROLL *light* Development Environment. In V. Mařík, J. Lažanský, and R.R. Wagner, editors, *Proc. 4th Int. Conf. on Database and Expert Systems Applications (DEXA'93)*, pages 673–676. Springer, Berlin, LNCS 720, 1993.

[Vla94] N. Vlachantonis. *Logische Modellierung der Steuerungs-Integration in offenen Softwareentwicklungssystemen.* PhD thesis, Technische Universität Braunschweig, Naturwissenschaftliche Fakultät, 1994.

[Wie93] R.J. Wieringa. *Information System Development and Conceptual Modeling: A Comparative Study.* Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1993.

[Wir90] M. Wirsing. Algebraic Specification. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science, Vol. B*, pages 677–788. North-Holland, Amsterdam, 1990.

[Wyb91] N. Wybolt. Perspectives on CASE Tool Integration. *ACM SIGSOFT Software Engineering Notes*, 16(3):56–60, 1991.