IEEE Transactions on Knowledge and Data Engineering,
Vol. 7, No. 3, pp. 505-508, IEEE, Los Alamitos, CA (1995)

A Development Environment for an Object Specification Language

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

Abstract

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. Two important ingredients of the environment to be described here in more detail are the animator and the proof support system.

Keywords: Information System Design, Semantic Data Model, Object Specification, Certification, Validation, Verification.

I. Introduction

Two important *phases* can be identified in the information system development process [1]: 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. We will concentrate in the following on the requirements engineering phase. This phase involves at least four important tasks [2]: (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* [3, 4], a dialect of OBLOG [5] and TROLL [6]. TROLL *light* is especially appropriate for information system design because it embodies ideas from data type specification [7], semantic data models [8], and process theory [9]. 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.

To summarize, we discuss the main motivation for our work which is without doubt improvement of software reliability and software quality. The way we want to achieve this is by formal specification, in particular with our specification language TROLL *light* which adds to classical semantic data modelling techniques ways to specify behavior. But TROLL *light* is not a stand alone language. Instead it highly supports the development process with modern graphical tools tackling both the formal and informal correctness problem. As far as we know, there are no other systems supporting formal object-oriented specification in the rigorous way as our environment does it. 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. Finally we give some concluding remarks.

II. 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 used templates
`SUBOBJECTS`	slots for subobjects

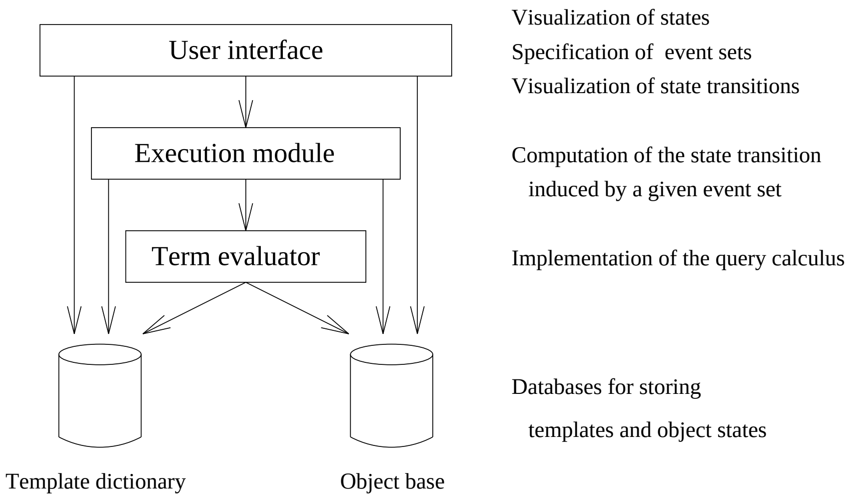

Fig. 1: Architecture of the TROLL *light* animation system

`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
`BEHAVIOR`	description of object behavior

`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. Due to space limitations we cannot explain the language features. Details concerning TROLL *light* can be found in [3, 4].

III. The Development Environment

The TROLL *light* development environment is an open system which allows developers to integrate new tools and to adapt and extend it to their own requirements. For preserving the openness of the environment, tools have to be as independent as possible. 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 [10] and on the other hand by an object-oriented documents repository (ObjectStore [11]) The latter supports storing design documents in a structured way whereas the former provides mechanisms for tool integration. The TROLL *light* development environment can be seen as an instantiation of the *ECMA* Reference Model for software development environments [12]. It is built on the depicted integration frames BMS, ObjectStore, and TCL/TK user interface manager [13]. A discussion of the architecture of the TROLL *light* development environment wrt. similar approaches can be found in [14].

A. The Animation System

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 subobject 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.

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.

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 subobject 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.

Architecture of the animation system. The structure of the TROLL *light* animation system is depicted in Fig. 1. 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. 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 state of an object community so that the term evaluator must be able to

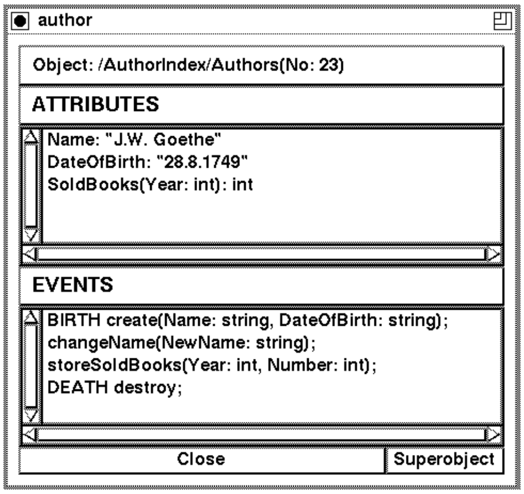

Fig. 2: Object window

access the object base. The *execution module* is the heart of the animation system. For a given set of event occurrences, its task is to compute 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.

Implementation aspects. A prototype version of the animation system covering basic requirements has been completed within our project. Implementation work was mainly done on the basis of diploma thesis.

The template dictionary and the object base were implemented using the OODBMS ObjectStore [11] 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. 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.

Finally, the user interface was realized by the help of TCL/TK [13]. In the prototype version of the animation system object states are visualized in so called object windows (see Fig. 2 for part of a library application). Object windows resemble the representation of pointers to current subobjects, the visualization of current attribute values as well as a 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.

B. The Proof Support System

Verification calculus. For pragmatic reasons, we only allow Gentzen-formulas with a conjunction of propositions as antecedent and a disjunction of propositions as conclusion as formulas in our calculus. The decision was motivated by the fact that thereby we can easily adopt existing proof systems, e.g., [15].

The most important concept of the verification calculus is the concept of formulas. We allow only a special form of clauses as formulas: $P_1, \ldots, P_n \to Q_1, \ldots, Q_m$ 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*.

IV. 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. A lot of work could be done here.

References

[1] 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.

[2] 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.

[3] 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.

[4] M. Gogolla, R. Herzig, S. Conrad, G. Denker, and N. Vlachantonis. Integrating the ER Approach in an OO Environment. In R. Elmasri, V. Kouramajian, and B. Thalheim, editors, *Proc. 12th Int. Conf. on the ER Approach (ER'93)*, pages 382–395. Springer, Berlin, LNCS 823, 1994.

[5] 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.

[6] G. Saake, R. Jungclaus, and T. Hartmann. Application Modelling in Heterogeneous Environments using an Object Specification Language. In M. Huhns et al., editors, *Int. Conf. on Intelligent & Cooperative Information Systems (ICICIS'93)*, pages 309–318. IEEE Computer Society Press, 1993.

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

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

[9] C.A.R. Hoare. Communicating Sequential Processes. *Communications of the ACM*, 21(8):666–677, 1978.

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

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

[12] 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.

[13] J.K. Ousterhout. TK: An X11 Toolkit Based on the TCL Language. Report, University of California at Berkeley, 1990.

[14] 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.

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