ELECTRONIC WORKSHOPS IN COMPUTING

Series edited by Professor C.J. van Rijsbergen

Rainer Manthey and Viacheslav Wolfengagen (Eds)

Advances in Databases and Information Systems 1997

Proceedings of the First East-European Symposium on Advances in Databases and Information Systems, (ADBIS'97), St Petersburg, 2-5 September 1997

A Validation System for Object Oriented Specifications of Information Systems

Antonio Grau and Mojgan Kowsari

Published in collaboration with the British Computer Society

BCS

Springer

ISBN: 3-540-76227-2

A Validation System for Object-Oriented Specifications of Information Systems

Antonio Grau [*]
Inst. for Prog. Languages and Information Systems, Database Group, Technical University Braunschweig
Braunschweig, Germany

Mojgan Kowsari [†]
Inst. for Progr. Languages and Information Systems, Database Group, Technical University Braunschweig
Braunschweig, Germany

Abstract

In this paper, we present a set of software tools for developing and validating object oriented conceptual models specified in TROLL. TROLL is a formal object-oriented language for modelling information systems on a high level of abstraction. The tools include editors, syntax and consistency checkers as well as an animator which generates executable prototypes from the models on the same level of abstraction. In this way, the model behaviour can be observed and checked against the informal user requirements. After a short introduction in some validation techniques and research questions, we describe briefly the TROLL language as well as its graphical version OMTROLL. We then explain the system architecture and show its functionalities by a simplified example of an industrial application which is called CATC (Computer Aided Testing and Certifying).

1 Introduction

Modelling information systems is becoming more and more important as the size and complexity of systems are growing. Validating these models is an indispensable task with the aim of checking whether they correctly describe the informal user requirements. Animating specifications which means the automatic construction of executable specifications on the same level of abstraction is a validation technique that allows users and specifiers to execute prototypes of the models for checking them with respect to the intended functionality. These generated prototypes must not be confused with the system implementation, where factors of quality like efficiency, security, ergonomic user interfaces etc. are borne in mind. Our intention is to construct prototypes as quickly as possible for validating the specifications at the first steps of the system development.

In this paper we propose a validation system for the formal object oriented specification language TROLL [6]. The TROLL approach supports the declarative specification of conceptual models. It integrates concepts for modelling dynamic, structural and process aspects. We use TROLL to develop an information system in an industrial environment. The project is located in the Physical Technical Federal Board (PTB), a federal institute for science and technology and the highest technical authority for metrology and physical safety in Germany. The project is carried out by group 3.5. The objective of this project is to develop an information system that supports staff and operators which are concerned with testing and certifying of explosion proof electrical equipment. Such equipment is allowed to be set into hazardous areas because it has been approved and certified by staff in charge (about 100 employees) settled in three different labs of the group 3.5 manually and individually. On average, 1000 certificates a year are issued. Because of the huge amount of data a standardized archive and catalogue of all existing certificates of explosion proof equipment is planned which

[*] Work reported here was partially supported by Spanish Ministry Education and Science
[†] And the Physical Technical Federal Board, lab 3.51, project Computer Aided Testing and Certifying

will be integrated in a software package called CATC (Computer Aided Testing and Certifying). Since 1994, the design and modelling of CATC is the long-term aim of the cooperation between the PTB and the database group of the Technical University of Braunschweig [25, 24, 19, 33]. As now, we have not applied any tool support to the modelling process of our system. This lack of tools has entailed some problems. On one hand specifiers (most of them students in the last year of their graduation) had to write the system specifications by hand and to check themselves whether the syntax and semantics of the specifications were correct. This task was sometimes tiresome because of the large amount of specification code. On the other hand the validation process was not so successful as we would liked it. The domain specialists had difficulties understanding the models and were not prepared to learn new formalisms. This handicap made difficult to check whether the specification was a good description of the requirements that the software system should fulfil. We think the application of software tools in the specification process would help to solve some of such problems. In this paper we will present our validation system by a simplified example of CATC.

In the next section we give a short introduction in some validation techniques and research questions. In section 3 we introduce briefly the TROLL language and its graphical version OMTROLL. Section 4 describes the functionalities of the validation system and its architecture. We will show the validation system by means of an example in section 5. At last, section 6 contains an outlook of the project.

2 Model Validation

The validation task has the objective of checking whether the model correctly and adequately expresses the requirements informally stated by the users. Nowadays the use of formal methods at the specification level constitutes a great topic of research in the academic community and its application in industrial environments is being significantly encouraged [22, 1]. Formal specification languages aid in making specifications more concise and less ambiguous, avoiding possible false interpretations of them. Unfortunately, their mathematical rigour entails complex syntax and semantics and consequently understanding the specification requires a good knowledge about their backgrounds. Validating the model requires the involvement of people with different backgrounds and therefore the model should be described in an easy and understandable way. Formality versus comprehensibility seems to be a dilemma difficult to solve. How can user participation be eased in the validation process? Different validation techniques have been developed with the aim of making possible this user involvement. Among them there are:

- *Presentation of the models in a more easily understandable way*. The introduction, for instance, of graphical symbols or user-defined concepts make specifications more intuitive and accessible to the users. Another possibility is to paraphrase specifications into natural language allowing users to read them in a language with which they are familiar [31, 3].

- *Model execution*. This approach emphasizes on validating the dynamic properties of the model through its execution. In this way users can observe, experiment and test the dynamic properties of the model making easier its comprehension. Different techniques are given in this context depending on the aims we want to reach with the execution or the level of executability of the modelling language. Some of them are [17]: step by step or interactive execution [29], batch execution, programmed execution with break points, transformational prototyping [28] and animation [20, 8, 26].

- *Explanation generation*. This technique aims at improving the validation of specifications by means of giving explanations related to the different parts of the conceptual modelling. In [15] an explanation component, giving explanatory texts about the modelling language, the conceptual model and its execution, together with its integration in a CASE environment is described. For explanation generation about model execution see also [30].

In this paper we propose an animation system for the object-oriented specification language TROLL. For animating specifications we mean the process of identifying *scenarios*, where a scenario is a sequence of events which may occur in the domain of the system, and testing them against the specification to see whether the specification meets the intended user requirements involved in these scenarios. This task requires the automatic construction of executable specifications from conceptual model specifications on the same level of abstraction.

In relation to animation, some questions can be formulated:

- *Are specifications necessarily executable?* Specifications should be declarative. They must describe *what* the customer want to have and not *how* it can be realised. It has been argued that the demands for high expressiveness and executability exclude each other, and that executable specifications should be avoided [21]. Nevertheless, some kind of executability could be profitable in order to validate the model against the user requirements [12]. Since specification languages often have executable constructs, we could refine non-executable parts of the specification into executables expressions. If such transformation is not possible we could also warn the user of the animation system if (s)he tries to execute non-executable parts of the specification [29, 20].

- *Does the executable specification correspond with the declarative one?* Since a formal specification language has semantics formally defined, it could be possible to prove that a tool supporting animation for such language conforms its dynamic semantics. However, this does not seem very practical because of the size and complexity of proofs [27].

- *What functional and non-functional requirements should the animation tool have?* In relation to the functional requirements, some features to take into account could be, for example, whether the execution is carried out in a batch and/or interactive way, the volume of information acquired about the execution traces or the level of explanation shown when an event cannot be carried out in the current system state. Non-functional requirements, as for instance, the size of the specifications that can be animated and with which speed can be executed the events defined in the scenarios, should also be considered. It has to be remarked that we say efficiency with reference to the animation tool and not to the specifications. The temptation of optimising the specification in order to achieve efficient executions would limit its expressiveness and therefore it should be avoided.

- *What test cases (scenarios) should be tested with the animation tool?* Since the number of possible scenarios to be considered in an exhaustive animation would be incredibly large, they must be restricted to a representative set. Here, test case generators or research in the area of specification-based software testing would be helpful.

- *Who should use the animation tool?* An animation session should be carried out by specifiers together with domain specialists in order to observe and discuss whether the specification describe the user requirements that the future software system must fulfill. Nevertheless, it must be mentioned that the interface of the animation tool does not represent the eventual user interface of the system being developed.

- *What conclusions can be stated about the animated specification?* Once the specification has been animated and observed that its behaviour meets the user requirements in a set of scenarios, we would like to affirm that such specification is *correct* with respect to the requirements. Unfortunately, such correctness can only be assessed if the specification has been tested in all possible scenarios. Thus, animation can only detect *incorrectness* in relation to the user requirements but not the opposite [13].

3 TROLL

TROLL stands for “Textual Representation of an Object Logical Language”. It is a formal language for the specification of object systems on a high level of abstraction [6, 18]. TROLL defines an abstract model called the *Universe of Discourse* to cover all aspects which are relevant with respect to organisational activities in complex information systems. It includes the functional requirements of the later system and excludes non-functional requirements (like technology bindings of later implementations).

TROLL combines an intuitive diagrammatic notation called OMTROLL, which is similar to OMT [32] and adapted to TROLL [34, 23].

The basic ideas and concepts of TROLL can be summarised as follows:

- The basic building blocks of information systems are objects.
- Objects are classified in classes and described by a set of attributes and actions.
- Every object describes a set of *sequential life cycles*, i.e. sequences of *local actions* on the object.
- An object system is composed of a number of concurrent objects. These objects are the *nodes* of the system. Nodes usually have other objects as components. To establish global communication in an object system, nodes can be connected through global interactions.

The following are the basic features of the language:

- A *system specification* consists of a set of *data type* definitions, a set of *object class* specifications (prototypical object descriptions), and a number of *object declarations*.
- *Parameterised data types* allow for the construction of new data types based on a fixed universe of predefined data types.
- An object class specification is a set of *attributes*, *actions*, and *constraints*.
- Object classes may be constructed over other object classes (*aggregation*) to describe complex objects, i.e., objects which are composed of component objects.
- An object class may be the *specialisation* of another object class. The specialised class (*subclass*) may have properties in addition to those inherited from its *superclass*.
- Concurrent objects are declared over object classes. These declarations describe the potential objects in the system. Interactions (through *action calls*) between different objects describe the global synchronisation relations. All actions which are called within one *event* are understood to take place concurrently. Action parameters are exchanged through unification.

OMTROLL consists of four diagrams to illustrate the specification graphically:

1. The *Community Diagram* defines the static structure of the system. It consists of all object classes, their composition and inheritance hierarchies, specialisation and aggregation of object classes and is the first raw design of the system. As such, it provides a simple and intuitive means to illustrate the structure of the system. The notation is quite similar to OMT and was adapted to TROLL [23].
2. The next step is to define an *Object Class Declaration Diagram* for each object class of the community diagram in order to declare its actions and attributes. The attributes are declared by their signature, i.e., name plus optional parameters, and optional classifications (e.g. derived, history, optional, constant). The actions are declared by an identifier and a list of parameters.
3. With the *Object Behavior Diagram* one can define explicitly the life cycle of an object. The nodes represent the states of an object. The edges represent state transitions and are labelled with the action that causes the state transition. Additionally, constraints can be attached to the state transitions. The diagram is pretty much like state charts of Harel [16].
4. Then the *Object Communication Diagram* depicts the communication between object classes. A set of interactions may be declared for each action and constraints for each occurrence of an action. These diagrams can be compared to Fusion diagrams [2].
5. Finally the *Data Type Diagram* represents user defined data types over standard data types.

The case study which will be introduced in Sect 5 illustrates some of the language features and will be present by OMTROLL notation. Semantics are given to TROLL specifications using different techniques: the static structure of an object system is semantically described with algebraic methods, statements over object states are expressed with a logic calculus, and the dynamic structure of the system, i.e. the systems evolution, is reflected via a temporal logic which is interpreted in terms of event structures. For exhaustive description of the underlying theory, semantics, and logics see [11, 10, 9], whereby refinement issues have been treated in [5, 4] and asynchronous communication in [7].

4 The TROLL Validation System

The validation system is a set of integrated tools with the aim of helping the specification of conceptual models in TROLL and their later validation through animation. For this purpose, editors are available helping the introduction of the specifications. Syntax and consistency checkers check the syntax as well as the static semantic of the specifications. Then, executable prototypes are generated and are executed by the animator. In this way a simulation of the model behaviour is obtained and can be checked against the intended system.

4.1 Animating Object Societies

The object system can be animated through *event* occurrences indicated from the outside of the system. An event occurrence implies the synchronised execution of a finite set of actions and determine the new object states. Basically, the animation should support:

- The exploration of actual states of the object society, i.e. the visualisation of object interfaces (attributes and actions) as well as the possibility of navigating across complex objects and their components and specialisations.
- The execution of event occurrences by the users.
- The visualisation of state transitions produced by such events.
- When an event occurrence cannot be carried out its cause must be reported to the user. (for instance, either a precondition is not fulfilled or the new state violates an integrity constraint.)

4.2 System Architecture

Once both the syntax and the static semantic has been checked the specification is divided in two parts:

- Signatures are stored in the repository. Specification signatures include object declarations, object class declarations and data type definitions.
- Axioms are translated into C++ code. Axioms include the attribute valuations and action calls defined in the object actions, as well as constraints and rules defined for derived attributes.

By means of the information stored in the repository the instance schema generator generates database schemas for the system instances. The instance database will contain the actual states of the system objects. This construction is depicted in Fig. 1.

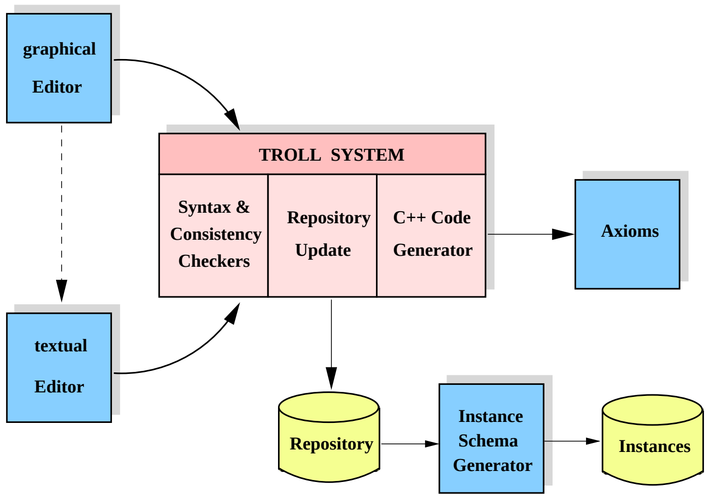


Figure 1 - System Architecture

In animation time (see Fig.2) the user interacts with the system through the user interface. Here the user can observe the system objects and their actual states, execute actions (for instance, create or delete objects executing birth or death actions resp.) and observe state transitions. For this task the animation manager reads in the repository the system descriptions, accesses the instances database in order to obtain information about the system objects and executes the code generated for the object axioms.

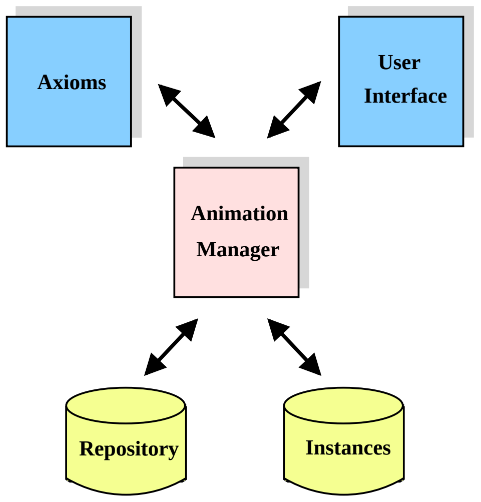


Figure 2 - Animation Architecture

Editors

In the TROLL graphical editor specifications are described using the OMTROLL notation. The graphical editor provides different diagrams to define object declarations as well as global structure and interactions. Operation definitions and axioms are introduced by adequate dialog boxes. Although the model can be specified entirely by means of the graphical editor, for refinement purposes would be more adequate the use of textual representation. For this reason the editor translates graphical notations into textual TROLL automatically.

For textual specifications a TROLL model language has been implemented in *Emacs* [14]. It constitutes the front end of the system. Here other system tools can be invoked providing a common interface. Among the new capabilities added to the *Emacs* Editor there are:

- support of specification projects (showing actual projects, adding or deleting files in a project etc.).
- automatic generation of patterns by defining macros.
- search for declarations through the project files.
- cross-referencing (for instance, between complex objects and their components).
- automatic generation of documentation files.
- different type styles for reserved words and comments.
- automatic indentation.
- invocation of the other system tools (parser, code generator, animator etc.).

Repository

Information about the system description is stored in the repository. The instance database generator reads from it the necessary information and generates the database schemas for the future system instances. In animation time the animation manager will access the repository in order to get information about the objects which are possible to create and about their object classes. The repository structure has been specified with TROLL itself.

Instance Database Generator

For the first prototype of the animator we have decided to use a relational database system due to its high performance and portability. Various techniques have been developed to translate TROLL objects into relational schematas. The instance database generator reads the specification description stored in the repository and generates a *sql* script which includes the necessary commands for creating the respective relational schematas.

Code Generator

The code generator translates the model behaviour into C++ code. The transformation strategy and execution mechanisms are being now studied. When an action **a** is called, it is executed as follows:

- an action set is constructed which contains all the actions called by **a**. This set includes transitive action calls and therefore the existence of cycles has to be checked.
- is checked if the preconditions of the actions in the set are fulfilled.
- is checked if there are conflicts in attribute assignments.
- the state transitions are temporary realised.
- is checked if the integrity constraints of the involved objects are fulfilled in the new state. If one or more constraints are violated the system returns to the old state.

Implementation Aspects

A prototype of the system is being now implemented with the following tools: C/C++ as programming languages, Tcl/Tk for the user interfaces and the graphical editor, Emacs for the textual editor and RDBMS Ingres for the databases. Current work focusses on the code generation as well as the database interfaces. As future work it is our intention to incorporate data distribution facilities.

5 The Case Study

In this section we present our case study. We focus on one part of the CATC system, namely the pressure test. We introduce briefly the CATC project, an informal description of the pressure test and its modelling. We exclude details of complex calculations, because it would go out of the scope of this paper.

The CATC Project

CATC has to support three different problem domains, which are based on certification of electrical equipment according to the European standards: *administration management, design approval,* and *experimental tests.*

The *administration management* includes the registration of applications by the manufacturer. It includes for example the company name, the staff name in this company and technical descriptions about the electrical equipment which has to be certified.

The subsystem dealing with *design approval* includes the assessment of design papers for the equipment based on descriptions and its accordance with the European standards.

The subsystem for the *experimental tests* performed by operators in the test lab stores all relevant data. There are different tests depending on the kind of the electrical equipment (for example temperature tests for heaters, experimental dynamic tests for motors). The European standards specify the design of such equipments in detail. During the testing procedure it is important to compare the standard values with the tests results.

Pressure Test

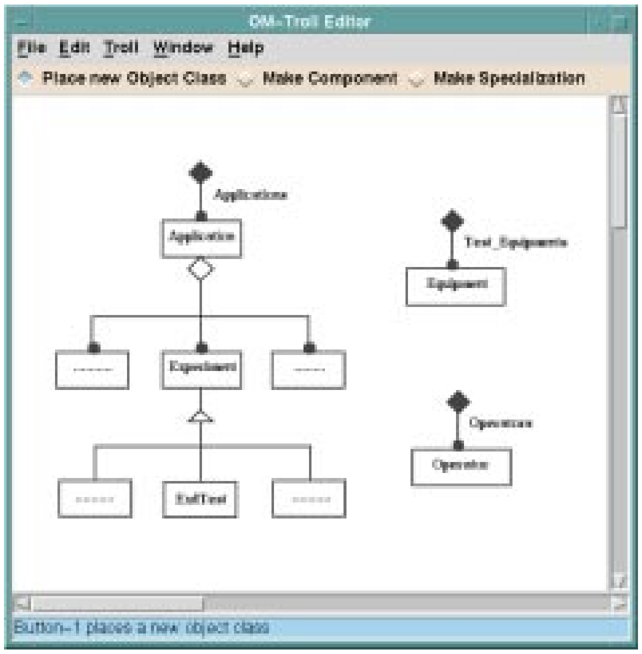
All parts of the equipment which potentially can ignite in an explosive atmosphere are tested. The test consist of provoking an internal explosion with an explosive mixture, whereby the pressure during the explosion is measured by different hardwares like digital and analog sensors.

Modelling the Pressure Test

The community diagram is depicted in figure 3. The system is composed of three kinds of objects: *Applications, Operators* and *Test_Equipments* which are described by the object classes *Application, Operator* and *Test_Equipment* respectively.

We will describe *Application* in detail. An application has several components one of which is *Experiment*. Components are represented in OMTROLL by a diamond. *Experiment* is a multiple component which means that several experiments belong to one application. This is denotated by a filled dot. As already mentioned there are different kinds of experimental tests. One of these tests is the pressure test. It is modelled as a specialisation of *Experiment*, namely *ExdTest*. Specialisation is depicted by a triangle. The information of the setup for the experiment as well as the result of the experiment will be stored in this object.

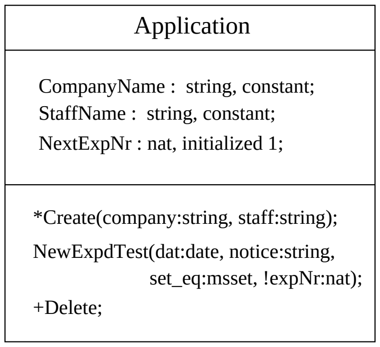
We will not go into detail about the interaction between the system objects. Rather we will later describe some more details about *Application*. Nevertheless, we roughly give an overview of the system process to illustrate the intended meaning. The operator interacts with the experiment by reading its setup and triggers the test equipment to perform the experiment. Afterwards the test equipment delivers the result of the experiment to the operator who will store it into the record of the pressure test.



Button-1 places a new object class

Figure 3 - Community Diagram of the Pressure Test

One of the features of the OMTROLL editor is to provide the different diagrams for specifying object systems. In particular, the editor does also support the representation of object class declaration diagrams. In figure 4 the object declaration diagram of *Application* is depicted. It denotes the signature of the object class, i.e., attributes and actions. An application has three attributes for storing general information such as the company name, staff name and the next experiment number. There are three actions for an application. A birth action creates an application and needs a company name and a staff name as input parameters. After being created we can assign new experiments to the application (*NewExpdTest*). This actions has the following parameters: a date, a comment and the setup for the experiment. As an output parameter the action gives back the experiment number.



Application

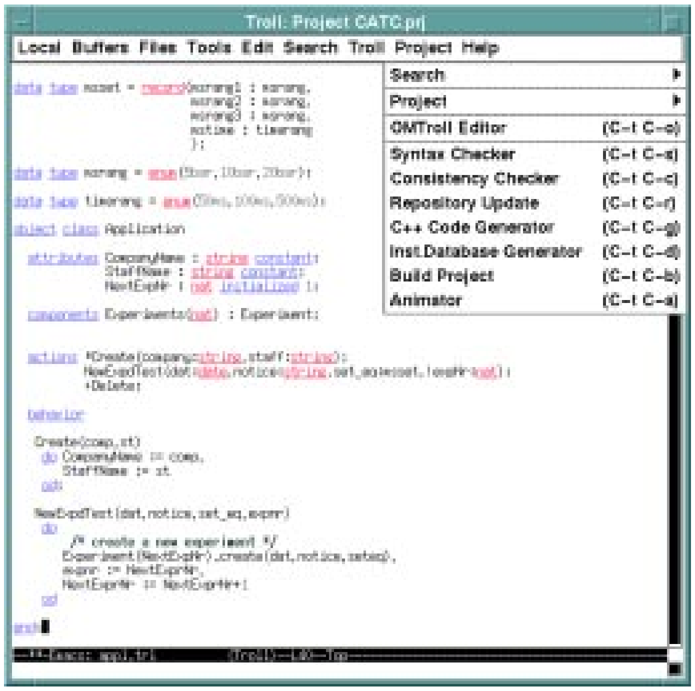
CompanyName : string, constant;
StaffName : string, constant;
NextExpNr : nat, initialized 1;

*Create(company:string, staff:string);
NewExpdTest(dat:date, notice:string,
set_eq:msset, !expNr:nat);
+Delete;

Figure 4 - Object Class Declaration Diagram of *Application*

Besides the graphical notation our tool also supports textual specification (TROLL editor). The designer can switch between both options of the editor and the translation of graphical to textual models and vice versa is done automatically. In figure 5 a screen shot of the TROLL specification of *Application* is given.

We give a brief overview of the functionalities of the TROLL editor. A specification can be split to several files that are managed by the TROLL editor. Moreover, documentation files for the specification are automatically generated. Syntax and consistency of the specification can be checked and other tools of the system can be called.


Troll: Project CATC.prj
Local Buffers Files Tools Edit Search Troll Project Help
Search
Project
OMTroll Editor (C-t C-o)
Syntax Checker (C-t C-s)
Consistency Checker (C-t C-c)
Repository Update (C-t C-r)
C++ Code Generator (C-t C-g)
Inst.Database Generator (C-t C-d)
Build Project (C-t C-b)
Animator (C-t C-a)

```
data type mzset = record(mzrang1 : mzrang,
                         mzrang2 : mzrang,
                         mzrang3 : mzrang,
                         notime : timerang
                         );

data type mzrang = enum(5bar,10bar,20bar);

data type timerang = enum(50ms,100ms,500ms);

object class Application

  attributes CompanyName : string constant;
             StaffName : string constant;
             NextExpNr : nat initialized 1;

  components Experiments(nat) : Experiment;


  actions *Create(company:string,staff:string);
          NewExpdTest(dat:date,notice:string,set_eq:mzset,!expNr:nat);
          +Delete;

  behavior

   Create(comp,st)
    do CompanyName := comp,
       StaffName := st
    od;

   NewExpdTest(dat,notice,set_eq,expnr)
    do
       /* create a new experiment */
       Experiment(NextExpNr).create(dat,notice,seteq),
       expnr := NextExpNr,
       NextExpNr := NextExpNr+1
    od

end
--**-Emacs: appl.trl          (Troll)--L40--Top-----
```

Figure 5 - Troll Specification of Application

A possible animation session is depicted in figures 6, 7 and 8. In figure 6 we can see the objects of our system. Here we can create new objects (instances) or choose one of them for visualising its interface.

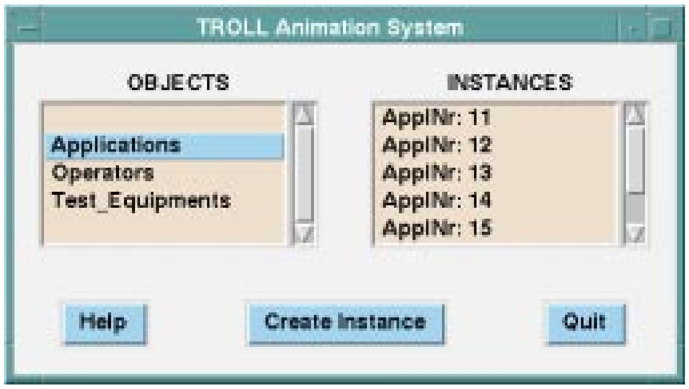
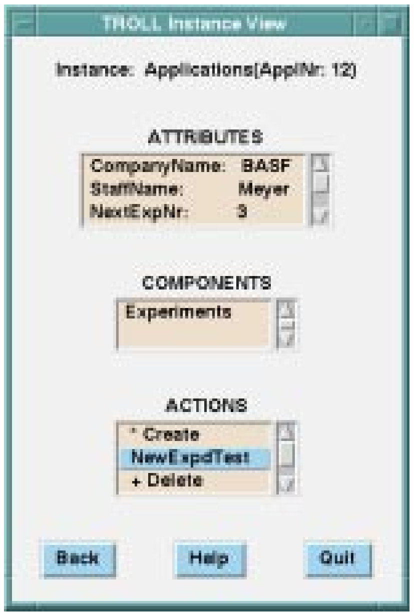


Figure 6 - Visualisation of the system objects

If we choose the object *Application(12)* in the above window we can observe its interface, i.e. the actual values of its attributes, its components and actions (see figure 7). Here we can navigate across its components, specialisations and object-valued attributes.



Instance: Applications(ApplNr: 12)

ATTRIBUTES

CompanyName:	BASF
StaffName:	Meyer
NextExpNr:	3

COMPONENTS

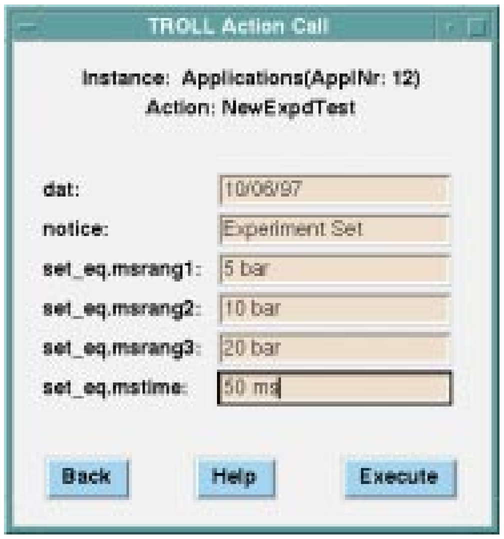
Experiments

ACTIONS

* Create
NewExpdTest
+ Delete

Figure 7 - Interface of an Application

By choosing the action *NewExpdTest* of an Application we get a dialog in which we can give the values of the input parameters (see figure 8). Then the action can be executed. In case there are output parameters their values are given back. We can now observe the new object system states provoked by the action occurrence. If the execution could not be executed due to some specification constraints, the system reports the reasons to the user.



Instance: Applications(ApplNr: 12)
Action: NewExpdTest

dat:	10/06/97
notice:	Experiment Set
set_eq.msrang1:	5 bar
set_eq.msrang2:	10 bar
set_eq.msrang3:	20 bar
set_eq.mstime:	50 ms

Figure 8 - Dialog for input parameters

6 Conclusion and Outlook

The animation of conceptual models eases the user participation in the validation process. Through the model animation users and specifiers can observe the dynamic properties of the model and check them against the intended requirements. In this paper we presented the basic features and architecture of an animation environment for the modelling language TROLL. We used an example from our project CATC. We have already finished modelling and implementing phases for one of the Laboratories of Group 3.5 in PTB. Our experience showed us that tool support is crucial for success of big projects. We do not yet have the support we wished to have. The project led to a vast list of requirements for adequate tool support. A major problem in software development is the tracing of requirements through the different development documents, the consistent introduction of new or the changing of existing requirements. This problem does appear in our approach too. Changes in the graphical specification have to be propagated into the textual and from there furthermore into the implementation. It is rather cumbersome to change all documents by hand and try to ensure consistency between them. The impatient engineer will anyhow fix the implementation to the changing requirements and won't care about the specification. The only way to overcome this problem is via CASE tools. We introduced the industrial context of our application domain, picked one small part out of it and illustrated the specification of this part.

Acknowledgments

We would like to thank Juliana Küster Filipe and Grit Denker for their last minute help. Moreover, we are grateful to Juliana Küster Filipe for proof reading an earlier version of this paper.

References

[1] J. Bowen and M. Hinchey. Ten Commandments of Formal Methods. *IEEE Computer*, 28(4):56–63, April 1995.

[2] D. Coleman, P. Arnold, S. Bodoff, S. Dollin, H. Gilchrist, F. Hayes, and P. Jeremes. *Object-oriented Development - The Fusion Method*. Prentice-Hall, 1994.

[3] H. Dalianis. A Method for Validating a Conceptual Model by Natural Language Discourse Generation. In P. Loucopoulos, editor, *Proc. 4th Int. Conf. on Advanced Information Systems Engineering (CAiSE'92)*, pages 425–444. Springer, Berlin, LNCS 593, 1992.

[4] G. Denker. Transactions in Object-Oriented Specifications. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers*, pages 203–218. Springer, Berlin, LNCS 906, 1995.

[5] G. Denker. Reification – Changing Viewpoint but Preserving Truth. In M. Haveraan, O. Owe, and O.-J. Dahl, editors, *Recent Trends in Data Types Specification, Proc. 11th Workshop on Specification of Abstract Data Types joint with the 8th General COMPASS Meeting. Oslo, Norway, September 1995. Selected papers.*, pages 182–199. Springer, 1996. LNCS 1130.

[6] G. Denker and P. Hartel. TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics. Informatik-Bericht 97–03, Technische Universität Braunschweig, 1997.

[7] G. Denker and J. Küster Filipe. Towards a Model for Asynchronously Communicating Objects. In H.-M. Haav and B. Thalheim, editors, *Proc. 2nd Int. Baltic Workshop on Databases and Information Systems, Tallinn, June 12-14, 1996*, pages 182–193. Institute of Cybernetics, 1996.

[8] E. Du Bois, P. Du Bois, and F. Dubru. Animating Formal Requirements Specifications of Cooperative Information Systems. In M. Brodie, M. Jarke, and M. Papazoglou, editors, *Proc. 2nd Int. Conf. Cooperative Information Systems (CoopIS'94)*, pages 101–112, 1994.

[9] H.-D. Ehrich. Object Specification. Informatik-Bericht 96–07, TU Braunschweig, 1996.

[10] H.-D. Ehrich and P. Hartel. Temporal Specification of Information Systems. In A. Pnueli and H. Lin, editors, *Logic and Software Engineering, Proc. Int. Workshop in Honor of C.S. Tang,Beijing, 14-15 August 1995*, pages 43–71. World Scientific, 1996.

[11] H.-D. Ehrich and A. Sernadas. Local Specification of Distributed Families of Sequential Objects. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Types Specification, Proc. 10th Workshop on Specification of Abstract Data Types joint with the 5th COMPASS Workshop, S.Margherita, Italy, May/June 1994, Selected papers*, pages 219–235. Springer, Berlin, LNCS 906, 1995.

[12] N. Fuchs. Specifications are (preferably) executable. *Software Engineering Journal*, 7(5):323–334, 1992.

[13] M. Gogolla, S. Conrad, G. Denker, R. Herzig, and N. Vlachantonis. A Development Environment for an Object Specification Language. *IEEE Transactions on Knowledge and Data Engineering*, 7(3):505–508, June 1995.

[14] J. Gosling. *Unix Emacs*. Carnegie Mellon Computer Science, Dept. Pittsburgh, PA, 1982.

[15] J. Gulla. A General Explanation Component for Conceptual Modeling in Case Environments. *ACM Transactions on Information Systems*, 14(3):297–329, 1996.

[16] D. Harel. On visual formalisms. *Commun. ACM*, 31(5):514–530, 1988.

[17] D. Harel. Biting the Silver Bullet - Towards a Brighter Future of Systems Development. *IEEE Computer*, 25(1):8–20, January 1992.

[18] P. Hartel. *Konzeptionelle Modellierung von Informationssystemen als verteilte Objektsysteme*. Reihe DISDBIS. infix-Verlag, Sankt Augustin, 1997.

[19] P. Hartel, G. Denker, M. Kowsari, M. Krone, and H.-D. Ehrich. Information systems modelling with TROLL formal methods at work. *Information Systems*, 22(2-3):79–99, 1997.

[20] T. Hartmann, R. Jungclaus, and G. Saake. Animation Support for a Conceptual Modelling Language. In V. Mařík, J. Lažanský, and R. Wagner, editors, *Proc. 4th Int. Conf. on Database and Expert Systems Applications (DEXA), Prague*, pages 56–67. LNCS 720, Springer, Berlin, 1993.

[21] I. Hayes and C. Jones. Specifications are not (necessarily) executable. *Software Engineering Journal*, 4(6):330–338, 1989.

[22] M. Hinchey and J. Bowen, editors. *Applications of Formal Methods*. Prentice-Hall International, Series in Computer Sciences, 1995.

[23] R. Jungclaus, R. Wieringa, P. Hartel, G. Saake, and T. Hartmann. Combining TROLL with the Object Modeling Technique. In B. Wolfinger, editor, *Innovationen bei Rechen- und Kommunikationssystemen. GI-Fachgespräch FG 1: Integration von semi-formalen und formalen Methoden für die Spezifikation von Software*, pages 35–42. Springer, Informatik aktuell, 1994.

[24] M. Kowsari. Formal Object Oriented Specification Language TROLL in Information System Design. In H.-M. Haav and B. Thalheim, editors, *Doctoral Consortium of 2nd Int. Baltic Workshop on Databases and Information Systems, Tallinn, June 12-14, 1996*, 1996.

[25] M. Krone, M. Kowsari, P. Hartel, G. Denker, and H.-D. Ehrich. Developing an Information System Using TROLL: an Application Field Study. In P. Constantopoulos, J. Mylopoulos, and Y. Vassiliou, editors, *Proc. 8th Int. Conf. on Advanced Information Systems Engineering (CAiSE'96)*, pages 136–159. Springer, Berlin, LNCS 1080, 1996.

[26] V. Lalioti and P. Loucopoulos. Visualisation for Validation. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proc. 5th Int. Conf. on Advanced Information Systems Engineering (CAiSE'93)*, pages 143–163. Springer, Berlin, LNCS 685, 1993.

[27] P. G. Larsen and N. Plat. Standards for Non-Executable Specification Languages. *Computer Journal*, 35(6):567–573, 1992.

[28] O. Lindland and J. Krogstie. Validating Conceptual Models by Transformational Prototyping. In C. Rolland, F. Bodart, and C. Cauvet, editors, *Proc. 5th Int. Conf. on Advanced Information Systems Engineering (CAiSE'93)*, pages 165–183. Springer, Berlin, LNCS 685, 1993.

[29] P. Mukherjee. Computer-Aided Validation of Formal Specifications. *Software Engineering Journal*, pages 133–140, July 1995.

[30] A. Olivé and M. Sancho. Validating Conceptual Specifications through Model Execution. *Information Systems*, 21(2):167–186, 1996.

[31] C. Rolland and C. Proix. A Natural Language Approach for Requirements Engineering. In P. Loucopoulos, editor, *Proc. 4th Int. Conf. on Advanced Information Systems Engineering (CAiSE'92)*, pages 257–277. Springer, Berlin, LNCS 593, 1992.

[32] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice-Hall, Englewood Cliffs, NJ, 1991.

[33] M. Schönhoff and M. Kowsari. Specifiying the Remote Controlling of Valves in an Explosion Test Environment. In J. Fitzgerald, C. Jones and P. Lucas, editors, *Formal Methods Europe, FME'97, 4th Intern. Symposium, Technical University Graz, Austria, 15-19 September, 1997*, pages 201-220. Springer, Berlin, LNCS 1313, 1997.

[34] R. Wieringa, R. Jungclaus, P. Hartel, T. Hartmann, and G. Saake. OMTROLL – Object Modeling in TROLL. In U. Lipeck and G. Koschorreck, editors, *Proc. Intern. Workshop on Information Systems – Correctness and Reusability IS-CORE '93, Technical Report, University of Hannover No. 01/93*, pages 267–283, 1993.