Pergamon

Information Systems Vol. 22, No. 2&3, pp. 79–99, 1997
Copyright© 1997 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0306-4379/97 $15.00 + 0.00

INFORMATION SYSTEMS MODELLING WITH TROLL FORMAL METHODS AT WORK

PETER HARTEL, GRIT DENKER, MOJGAN KOWSARI, MAREN KRONE and HANS-DIETER EHRICH

Technische Universität Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D–38023 Braunschweig, Germany

(Received 15 October 1996; in final revised form 1 April 1997)

Abstract — In this paper we present a national project located in the area of computer aided testing and certifying (CATC) of physical devices. The objective of this project is to develop an information system that supports the various activities of different user groups in a German federal institute of weights and measures. We decided to use formal methods right from the beginning of the project. Our approach is based on the formal object oriented specification language TROLL. Starting point of the development is an abstract model of the organization which will serve later on as a formal basis for implementation. We present parts of this specification and its relations with the underlying formal semantics. The experiences we made so far are rather positive and we expect further positive effects in the future. Copyright ©1997 Elsevier Science Ltd

Key words: Object Oriented Specification, Case Study, Information System, Information Modelling, Requirements Engineering, Formal Method

1. INTRODUCTION

The development of a large information system is by far no trivial task. One main problem with it is to ensure "that we get what we want". In the past 25 years many suggestions have been made on how to tackle complex software engineering projects. However, there is no silver bullet yet [3]. There is a small but growing community of people who propose and promote formal methods in Software Engineering [45, 2]. Most times these people come from academia. The acceptance of formal methods in industry is still low. This is mainly due to the fact that formal methods are thought to be complex, hard to handle and not suitable for real world applications [18].

In order to make formal methods attractive for industry they have to fulfill several requirements. They have to be easy to learn and to teach [22, 1]. In today's organizations we do not find many people who know formal methods [2]. This means we have to invest in their education. If this investment is too high or people feel that they are not able to master the formalism then there will be a low chance of success. Formal methods have to be supported by tools (e.g. semantic editor, testing, prototyping) [15]. The formalism allows us to build intelligent tools which can speed up development drastically. Graphical representations help to overcome the fear of embarking on formalisms. Methodological guidelines [1] are another important issue for the acceptance of formal methods.

We present in this paper the use of formal methods for the development of an information system in an industrial environment. The project is located in the area of *C*omputer *A*ided *T*esting and *C*ertifying (CATC) which is conducted by the federal institute of weights and measures of Germany. About 100 employees settled in three labs will use the system. When the project started in the beginning of 1994 no formal methods were applied. Mid of 1995 it became clear that the chance of success with the chosen approach was rather low [25]. At that time we decided to use a formal approach [32]. This paper presents the problem domain of our project and gives a brief introduction into the mathematical formalisms underlying our approach. It exemplifies the use of the method by presenting a small part of the development. After more than one year we have collected several experiences, positive as well as a few negative ones. Furthermore, we will give some hints, why our first approach without formal methods did not succeed.

The objective of the project is to develop an information system that supports the activities of different user groups in the federal institute. Such activities are often called business processes [21]. The complexity of the organization and the system that is supposed to support this organization is rather high. Besides, the system has to integrate already existing applications and re-specified ones. In order to be able to develop such a system we have to get a deep understanding of the organizational structures. This is the prerequisite for deciding which part of the organization shall be computerized and how this system is embedded into the organization.

In order to achieve this understanding we have to build an abstract model of the organization which has to cover all structural aspects and organizational activities. This model represents what we call the Universe of Discourse (UoD).

A formal adequate method should allow for the modelling of the intended system on a high abstract level. Existing and widely accepted formal languages like Z [44], VDM [28] do not provide the right level of abstraction for modelling. Semantic data models emphasize on structural aspects and do not allow for the modelling of complex behavioral aspects. On the other hand there exist numerous formal approaches towards process modelling. Most of them either neglect the static aspects like CSP [24] or do not come with the concepts needed for information systems modelling.

Object orientation is a typical answer towards this problem. The object oriented paradigm recognizes as primary concept the object. An object allows for an intuitive presentation of real world entities and may reflect their behavioral and static properties. Methods like OMT [40] or "Object-Oriented Software Engineering" [27] are quite popular. However, they miss the required formality. The project we are going to present started with such an informal method and did not achieve the desired results. This resulted in a loss of confidence in such informal approaches.

The solution of this dilemma can be the combination of formalisms and object oriented methods. Several formal specification languages have already object oriented extensions e.g., VDM++ [9], MooZ [36]. Even with this adaption of object orientation they still cope with a low level of abstraction.

We decided to apply the formal and object-oriented specification language TROLL [30, 20]. The TROLL approach incorporates many ideas which have been developed over the past 8 years. Much work in the theoretical foundations [43, 12, 10, 7, 11] and on methodological [41, 42, 21] issues has been done.

The TROLL approach supports the declarative specification of conceptual models. It integrates concepts for the modelling of dynamic, structural and process aspects. With the TBench [34] a specification tool for TROLL is available. The TROLL method [31] combines an intuitive diagrammatic notation with a textual one.

In this paper we introduce the problem domain, the information system to be developed and our experiences we made by using a formal approach. In the next section we give an introduction to the application field of the federal institute. Section 3 depicts an overview of the formalisms underlying our method. We introduce in section 4 a small part of the conceptual model, some methodological guidelines and the relationship between the mathematical formalism and the conceptual model. Our experiences are summed up in section 5. We end the paper in section 6 with future expectations and some conclusions.

2. DESCRIPTION OF THE PROBLEM DOMAIN

In this section we provide an introduction to the problem domain of our case study. We want to give some idea about important aspects of our specific application, the requirements of the intended system, and the complexity we have to deal with.

Our case study is located in the area of the Physikalisch-Technische Bundesanstalt[†] (PTB).

The PTB [26] is a federal institute for science and technology and the highest technical authority for metrology and physical safety engineering in Germany. Its tasks are research in physics and technology, realization and dissemination of SI units[‡], cooperation in national and international technical committees, physical safety engineering serving the protection against explosions etc.

[†]federal institute of weights and measures
[‡]international system of units

The group 3.5 'explosion protected electrical equipment' is concerned with the testing and certifying of explosion proof electrical equipment. The basis are the European standards EN 50014-50028 [13, 14]. Such equipment is allowed to be set into hazardous areas because it has been approved and certified due to European harmonized standards. The assessment procedure consists of testing the formal and informal documents, checking the design papers (i.e., technical drawings) and the tests which are carried out according to European standards. There are experimental tests such as explosion tests, flame propagation tests and thermal-electrical investigations. Currently, all steps which are necessary for this are carried out manually by the staff in charge and are worked out individually. About 100 employees settled in three labs of the group 3.5 are now concerned with testing and certifying.

On average 1000 certificates a year are issued. It is important that all informations in connection with a certificate are available and reusable at any time. Because of the huge amount of data a standardized archive and catalogue of all existing certificates of explosion proof equipment is planned which will be integrated in a software package called CATC (Computer Aided Testing and Certifying). The design, modelling, and implementation of CATC is the long-term aim of the cooperation with the database group of TU Braunschweig started in 1994.

The technical constraints fixed by PTB for CATC are as follows: In order to support rapid communication between staff and operators on the one hand and between staff and the secretaries who are settled in different buildings on the other hand the group 3.5 is operating a local network. The employed client/server system (IBM LAN SERVER 4.0) supports database application programs. The database management system (DB2/2) is based on the relational model.

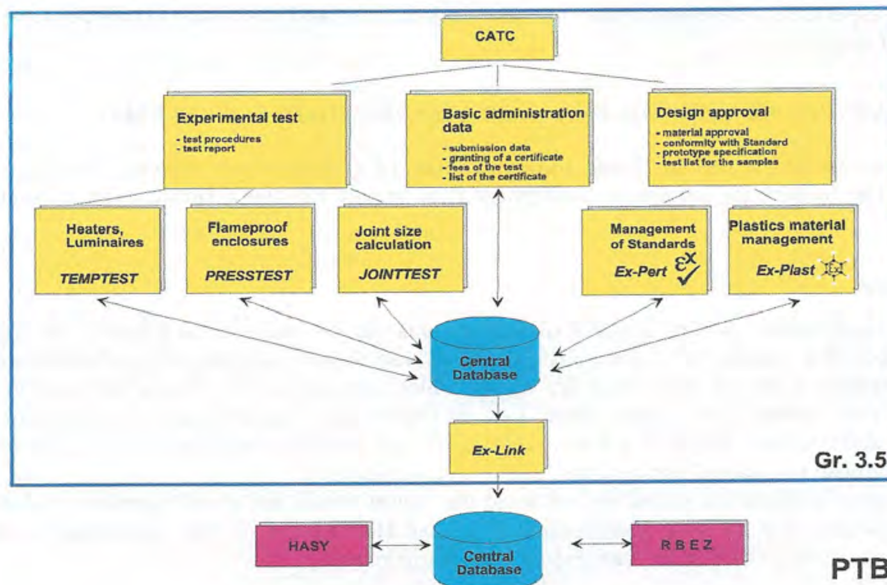

Fig. 1: CATC – Overview

CATC has to support several different problem domains. As such it has to:

1. support *experimental test*,
2. manage *basic administration data* and
3. allow for *design approval*.

Fig. 1 illustrates the hierarchical structure of the intended Information System.

The *administration* management includes the registration of formal information of the manufacturer, the settlement of accounts and legal matters. This information is essential for the following tests in the certification process and has to be permanently available.

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. It provides the relevant clauses of the standards such that the certification becomes more efficient. Required data can be carried out faster and easier at every desk.

The subsystem for the *experimental tests* performed by operators in the test lab stores all relevant data. The focus is to ensure that for example with flame-proof enclosure the parts which potentially can ignite an explosive atmosphere are placed in an enclosure which can withstand the pressure developed during an internal explosion of an explosive mixture and which prevents the transmission of the explosion to the explosive atmosphere surrounding the enclosure. During the explosion every 0.2 second 30 kbyte data are produced. Thus, there is a conflict between hardware, which works in real time, and the multi-tasking operating system (OS/2).

CATC has via LAN access to the central database of the PTB, where common data are stored. There are further programs for administration (RBEZ†, HASY‡) which access that database (see Fig. 1, Ex-Link). CATC is not a standalone Information System but it has to be embedded in an existing environment. Besides, we have to deal with existing application programs which have to be re-specified (e.g., JOINTTEST) because they were erroneous. These re-specified parts have to be embedded in the new Information System structure. In addition, there is the link to the multiply accessed PTB wide database.

To summarize, we have a safety-critical application area that comprises technical aspects as well as database aspects in a heterogeneous complex environment and that has to consider existing and re-developed applications.

3. A FORMAL METHOD FOR CONCURRENT OBJECT SYSTEMS

In this section we introduce our basic understanding of systems and objects. We give an introduction to the underlying semantic framework that serves for the formalization of system specifications.

3.1. The Concepts

Our intuitive understanding of concurrent object systems can be described as follows: An object system is composed of a number of concurrent objects. These objects are the *nodes* of the system. Every object describes a set of *sequential life cycles* which are sequences of *local actions* of the object. Objects may interact with each other, i.e., an object may call a local action of another object. Such a *global action* forces a synchronization of the participating objects, i.e., all local actions which compose the global action must occur simultaneously.

An object system describes a global web of local life cycles which are glued together at shared communication points. TROLL is a specification language that allows for the modelling of such concurrent object systems. The basic features of the language are:

- A *system specification* is a set of *data type*, a set of *object class* specifications which are prototypical object descriptions and a number of *object declarations.*
- *Parameterized data types* allow for the construction of new date types based on a fixed universe of predefined data types.
- An object class specification consists of a set of *attributes*, *actions*, and *constraints.* Attributes describe the state of an object of that class and the actions determine the possible object evolution. Constraints allow for the definition of static and transitional invariants over the object state.

†archive and documentation application
‡settlement and calculation application

- Object classes may be constructed over other object classes (*aggregation*). Such classes describe complex objects, i.e., objects which are composed of component objects. The specifications of the component objects are embedded into the specification of the aggregation. This allows us to define constraints over the aggregation, i.e., the objects in composition. It also enables the definition of local interactions inside the complex object. In this way we may construct complex local actions of the local actions of the object in composition. Thus, a local action of an aggregated object may consist of different local actions of its components.

- An object class may be the *specialization* of another object class. The specialized class may have additional properties to the inherited ones. Inheritance may be monotone. In this case we talk about save inheritance, i.e., all axioms being valid for an object of the superclass are always satisfied by an object of the subclass.

- Concurent objects are declared over object classes. These declarations describe the potential objects in the system. Interactions between different objects describe the global synchronization relations.

The case study which will be introduced in section 4 illustrates some of the language features. There we will explain the concepts in more detail.

3.2. The Semantics

Semantics is 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, 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. An exhaustive description of the model theory is given in [11]. In the following we intuitively explain these semantic ingredients. Moreover, in section 4 the semantic notions are illustrated by example.

Static structures are needed to describe the state of objects. Such static structures are defined by data signatures and their algebraic interpretation. We assume a *data signature* $\Sigma_D = (S_D, \leq, \Omega_D)$ with a given number of data sorts S_D which are the predefined ones and the constructed sorts, a partial order on data sorts $\leq$, and data operations over these sorts Ω_D, whereby every constructed sort induces a number of operations. For instance, for the data sort `list` there are predefined operations `concat`, `append`, etc. The interpretation of such a signature is a Σ_D*-algebra*. In order to make statements over object states we adapt a logic calculus [23, 17]. This calculus is especially suited in the domain of information systems since it provides powerful means to express queries over objects in a declarative way. It goes beyond this paper to explain all the features of this calculus. The interested reader is referred to [23, 17].

In order to specify object systems we have to extend the data signature by sorts and operations which describe objects. For this purpose we introduce so-called *extended data signatures*. This signature extends the data sorts S_D by a special data sort S_O^i of *objects identities* and the data operations by S_O^a, the *object actions*. Thus, data terms are built over an extended data signature $\Sigma = (S, \leq, \Omega)$, $S = S_D \cup S_O^i \cup S_O^a$, $\Omega = \Omega_D \cup \Omega_O^i \cup \Omega_O^a$, which is the basis of data terms as well as identity and action terms, i.e., $i \in T_\Sigma(X)_{id}$ and $\alpha \in T_\Sigma(X)_{ac}$, respectively.

Skipping some technical details which can be found in [11] we arrive at a so-called *instance signature* $\Sigma_I = (Id, Ac)$, which consists of a set of identities Id representing all objects of the system, and a set of actions Ac_i for every object $i \in Id$. With the help of the case study which will be presented in the next section we illustrate these notions. Instance signatures will be the basis for constructing models in the framework of event structures. Up to this point we have covered all structural aspects of an object system description.

We introduce a temporal logic to deal with system dynamics. This logic is a first order predicate logic extended by two predicates on actions (*enabling* and *occurrence* of actions) and temporal operators for the future (*tomorrow* and *sometime in the future*) and for the past (*yesterday* and *sometime in the past*).

Let $\Sigma = (S, \leq, \Omega)$ be an extended data signature over an S-indexed family of sets of variables $X = \{X_s\}_{s \in S}$ and let $T_\Sigma(X)$ be the set of Σ data terms. The set of formulae L_Σ of the object logic is inductively defined as follows:

- if $t_1, t_2 \in T_\Sigma(X)_s$ then $t_1 =_{ss} t_2 \in L_\Sigma(X)$;
- if $\alpha \in T_\Sigma(X)_{ac}$ then $\rhd\alpha \in L_\Sigma(X)$ (enabled action) and $\odot\alpha \in L_\Sigma(X)$ (occurred action);
- if $\varphi, \psi \in L_\Sigma(X)$ and $x \in X_s$ then $\neg\varphi, \varphi \vee \psi, \exists x \ \ \varphi \in L_\Sigma(X)$;
- if $\varphi \in L_\Sigma(X)$ and $i \in T_\Sigma(X)_{id}$ then $\mathtt{X}_i\varphi \in L_\Sigma(X)$ (tomorrow), $\mathtt{F}_i\varphi \in L_\Sigma(X)$ (sometime in the future), $\mathtt{Y}_i\varphi \in L_\Sigma(X)$ (yesterday), and $\mathtt{P}_i\varphi \in L_\Sigma(X)$ (sometime in the past);

We will give some examples of formulas in section 4 by translating our case study.

Instance signatures together with temporal logic formulas which describe the behavior of objects are interpreted over labelled event structures. Each node of an object system has a labelled sequential event structure as a model, and the object system is modelled by a concurrent labelled event structure built of the sequential event structures by event sharing. Thus, nodes have sequential models whereas concurrency comes into play in the object system.

A sequential event structure is a triple $E = (Ev, \rightarrow^*, \#)$, where Ev is a set of events, $\rightarrow^*$ is a partial order (causality), and $\#$ is a symmetric reflexive order (conflict). Moreover it satisfies three conditions: (1) there exists a unique, minimal element $\epsilon \in Ev$, (2) all configurations $\downarrow e := \{e' \mid e' \rightarrow^* e\}$ are totally ordered, and (3) $e \# e' \Leftrightarrow \neg(e \rightarrow^* e' \vee e' \rightarrow^* e)$ for all $e, e' \in Ev$.

Thus, a sequential event structure is a tree where every branching point indicates conflict. Since conflict is a derived concept we denote sequential event structures by $E = (Ev, \rightarrow)$, where $\rightarrow^*$ is the reflexive transitive closure of the irreflexive step relation $\rightarrow$.

These sequential event structures are put together via event sharing to form concurrent event structures which are models of the system. In the system model concurrency arises and conflict remains to be local, i.e., $e \# f$ for $e, f \in Ev$ iff there is an object i and locally conflicting events $e', f' \in Ev_i : e' \# f'$ such that $e' \rightarrow^* e$ and $f' \rightarrow^* f$. Events are concurrent, $e \ co \ e'$, iff $\neg(e \rightarrow^* e' \vee e' \rightarrow^* e \vee e \# e')$.

Models of object systems are labelled event structures $\bar{E} = (E, \mu)$ which are built from locally sequential event structures E_i: $\bar{E} = \bigcup_{i \in Id}(E_i, \mu_i)$. The labelling function μ maps each event to a global action, i.e., set of local actions: $\mu : Ev \rightarrow \mathcal{P}_f^+(Ac)$, $\mu = \bigcup_{i \in Id} \mu_i$, $\mu_i : Ev_i \rightarrow Ac_i$.

In section 4 we will come back to this. First we specify our case study and afterwards we will depict part of the model.

4. THE CASE STUDY

In section 2 we described our problem domain. For illustrating the use of TROLL in the design of Information Systems, we focus on one part of the CATC system, namely JOINTTEST.

In the following we will give an informal description of the JOINTTEST component. First we briefly explain some *technical notions* which are necessary for the specification. Then we will introduce the specific requirements of JOINTTEST especially the *process of* JOINTTEST. We exclude details of complex obligatory calculations, because it would go beyond the scope of this paper. Afterwards, we present the *modelling of the Universe of Discourse with* TROLL. The textual object oriented specification language TROLL (*Textual Representation of an Object Logic Language*) comes along with the graphical notation OMTROLL (*Object Modeling with* TROLL). After a brief introduction to the development methodology of TROLL we partially present the design of the JOINTTEST. However, we restrict ourselves to a simplified version of JOINTTEST that is sufficient to illustrate and explain all concepts of TROLL and their use in the modelling of information systems.

The last part *semantics* will forge the link back to section 3 by presenting parts of the semantics of the case study.

4.1. Technical Notions

The *flame proof-joint*, *joint* for short, is the place where corresponding surfaces of two parts of an enclosure come together and prevent the transmission of an internal explosion to the explosive atmosphere surrounding the enclosure [19].

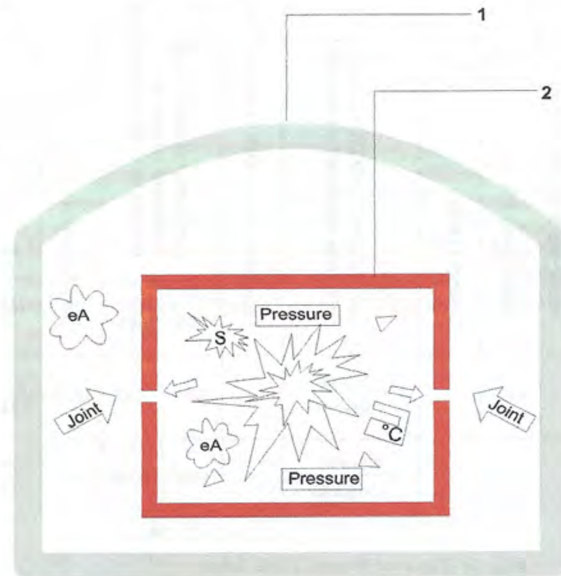

Fig. 2: Flame Proof Enclosure Test
Flame proof joint test, 1: autoclave , 2: enclosure, eA : explosion atmosphere, s: spark, $^{\circ}C$: surrounding temperature

In figure 2 we illustrate a test surrounding for flame proof joint tests. The main components to measure and estimate joints according to the standard given are the *width* and the *gap of a joint*. The width of a joint is the shortest distance from the inside to the outside of an enclosure. The gap of a joint is the distance between the corresponding surfaces when the electrical apparatus has been assembled. The prototype tests on flame proof is comprised of tests on the ability of the enclosure to withstand pressure and of tests on the non-transmission of an internal ignition. Therefore the enclosure is placed in a test chamber called autoclave and some explosive mixture is introduced into the enclosure.

The European Standard specifies the design of flame proof joints in detail. During the testing procedure it is important to compare the standards values of the widths and gaps of the joints with the applicants value resulting from the explosion tests.

Process of JointTest

There are two groups: staff and operators who can manipulate joints. The applicant, i.e., the one who wants some device to be certified by PTB, sends the table of flame path joints (see figure 3). There are three different kinds of values:

- Columns 2-4 give the data according to EN 50018.
- Columns 5-8 include data according to construction drawings.
- The last three columns of the table are values resulting from tests.

The staff compares the data according to EN50018 with construction drawing and decides, whether the values are satisfactory. The operators verify the values provided by the applicant and

report their results to the staff. The staff is responsible for the assessment of the values measured by the operator.

	Data acc. to EN 50 018 - 1977/ VDE 0171			all to construction drawing						test sample		
Joint	Minimum length of joint	Distance l (for bores within flamepath)	Maximum width of joint i_R (flat, combined, cyclindric, bearinggap and actuationsgap)	Width of joint L (c+d)	Distance l (a+b)	Size of diameter with ISO-symbol for internal and outside measurement	Dimensions (acc. to DIN 7160, DIN 7161)	Constructional width of gap = difference of diameter between bore and cyclindric bore min.	max. $= l_c$	Length of joint L	Distance l	Width of testing gap i_E
K1	12.5		0.15	140	-	7.5+00.5 / 7.45-0.05		0.05	0.15	15.42	7.57 / 7.333	0.225
K2	12.5		0.15	141	-	7.5+00.5 / 7.45-0.05		0.051	0.148	15.43	7.57 / 7.333	0.227

Fig. 3: Table of Flame Path Joints

4.2. Modelling the Universe of Discourse with TROLL

In section 1 we mentioned the problems arising with developing huge information systems in complex organizational structures. Our method to overcome these problems is to specify the Universe of Discourse of the problem domain rather than the application program itself. The object oriented paradigm is well suited for this. Anyhow, one major problem in UoD modelling is the identification of the relevant objects.

In order to elaborate a UoD model, the TROLL method integrates a number of diagrams which allow for a pictorial presentation of static and dynamic aspects of the model. These diagrams are easy to understand and therefore well suited for discussing the essential aspects of the system with the client. Thus, we use TROLL in the analysis phase to fix the functional requirements. Due to the formalism the usual misunderstandings between the developer and the client in what the system really shall do can be diminished. Even in the case when the clients do not yet know, what they want, the formalism helps them and the developers in understanding the general setting of the problem domain.

The diagrams represent different aspects of the system like communication, object composition and specialization hierarchies etc. However, they do not allow a complete system's specification. Starting from a digrammatic specification one can derive a textual spec. This textual document is then further enriched in order to achieve a complete specification of the functional system requirements. The result of the conceptual modeling is a set of diagrams and textual documents, which represent different views of the system. In order to maintain the consistency of all these documents a sophisticated tool support is needed.

However, the diagrams we introduced are not new. They are quite similar to those of other methods (e.g., OMT) and represent different system's aspects. We did not try to invent a new graphical representation technique but adapted existing and widely accepted techniques. Our main concern was to establish a graphical notation with a well-defined semantics. The diagrams used in OMTROLL have a theoretical underpinning such that there is no ambiguity in their meaning.

The interested reader is referred to [8] for an in depth discussion of the graphical notation. In [8] the correct translation of OMTROLL to TROLL is given. Thus, semantics of OMTROLL is defined via translation to TROLL which in turn has a precise semantics using event structure models. We are currently working on tool support for the automatic translation of graphical specifications to TROLL.

We will start by explaining the different diagrams. We focus on the diagrams which represent the object community and the properties of the objects. Besides, a specification contains a number of *Data Type Diagrams* for the modeling of complex data structures. These data structures represent the information stored and manipulated by the objects.

Community Diagram

The *Community Diagram* (see figure 4) is the primary diagram of the specification. It constitutes the global system architecture by showing the static system structure. The Community Diagram shows all object classes and their relationships, i.e., composition and inheritance hierarchies. The used notation is quite similar to OMT and was adapted to TROLL [31].

A part of the community diagram of the CATC system is depicted in figure 4. It consists of the object classes `JointNode`, `JointTable`, `Joint`, `ExpJointPart`, `ConstJointPart` and `JointPart`.

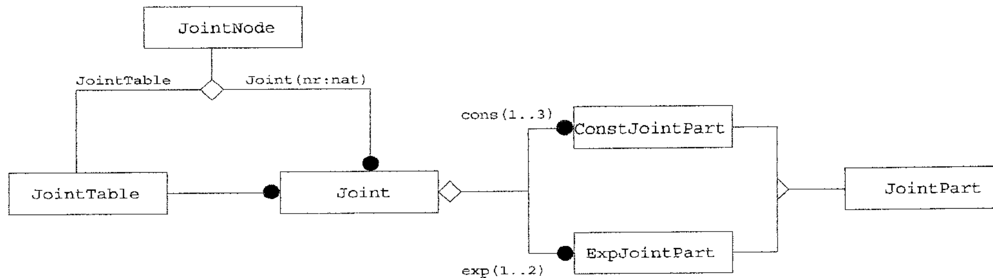

Fig. 4: Fragment of the Object Community Diagram of CATC

`JointNode` is the object class that models the special part of CATC concerning joint tests. In this universe we have joints and joint tables. We simplified the specification to one joint table and several joints. The diamond stands for aggregation of objects and the triangle is the diagrammatic notion for specialization. Thus a `JointNode` object is an aggregation of one `JointTable` and one or more `Joints`. A `JointTable` has a list of joints as attributes. This is modelled as an object-valued attribute. Joints can be constructed of several parts, constructive parts (`ConstJointPart`) and experimental parts (`ExpJointPart`). The constructive part is concerned with comparing data according to the standard with data according to the construction drawing (see section 4, Process of JOINTTEST). The experimental part deals with the results of test measurement. Up to five parts can belong to one `joint` which form one row in the table of flame path joints (see figure 3). There may be one to three constructive parts and one to two experimental parts. The object class `JointPart` depicts a specialization, which consists of those attributes and actions, the constructive and experimental parts have in common.

Object Declaration Diagram

The interfaces of the objects specified in the comunity diagram are specified in the *Object Declaration Diagram* (see figure 5). For each object class an attribute and an action alphabet has to be specified. The attributes represent the observable state of an object of that class. Actions model the operational interface of the objects. The actions are declared by a classwide unique identifier and a list of parameters. Actions that create new objects are called birth actions and are indicated by an *. The life of an object is terminated with the occurrence of a death action. Such actions are marked with an +.

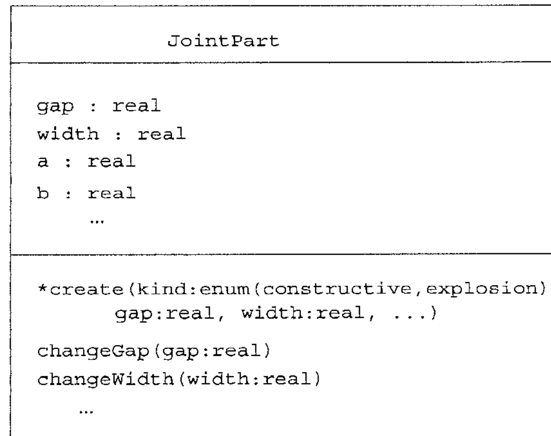

Fig. 5: Object Declaration Diagram of JointPart

Object Behavior Diagram

The longterm behavior of objects is specified in *Object Behavior Diagrams* (see figure 6). These diagrams are more operational and allow for an explicit description of the permitted object life cycles.

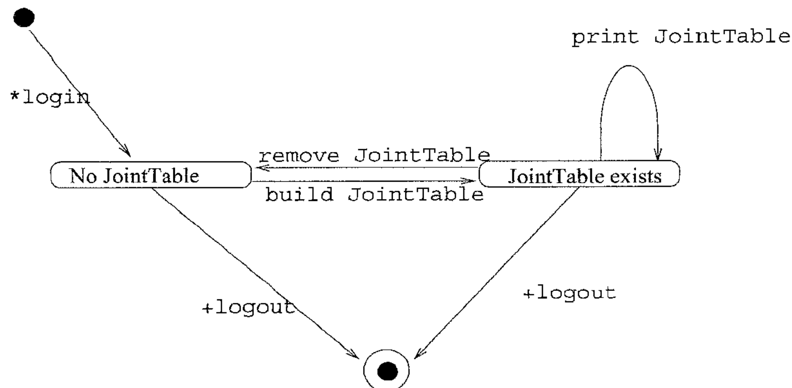

Fig. 6: Object Behavior Diagram of the Staff

The behavior of staff objects is illustrated in figure 6. Staff objects are born by login and by it they are in the "NoJointTable" state. This is the beginning of the lifecycle of a staff object. It may logout immediately after login and by this will leave the system. Logout is the *death* action of a staff object. After a login a staff object may build a joint table. By this action the life cycle state changes to the "JointTable exists" state. Now it can work with the joint table, e.g., print it or do other things not specified here. From this life cycle state a staff object may logout or remove the joint table. The latter action will change back to the life cycle state where no joint table exists.

Object Communication Diagram

Interactions between objects are depicted in *Object Communication Diagrams*. An interaction represents a causality relation between the actions in interaction. The chosen notation is quite similar to that one of Fusion [4].

The communication between `JointNode` objects and `JointTable` and `Joint` objects is depicted in figure 7. The action `createJoint` of a `JointNode` object causes directly a create action of a `Joint` object. The action `createJointTable` of a `JointNode` causes a create action of a `JointTable`.

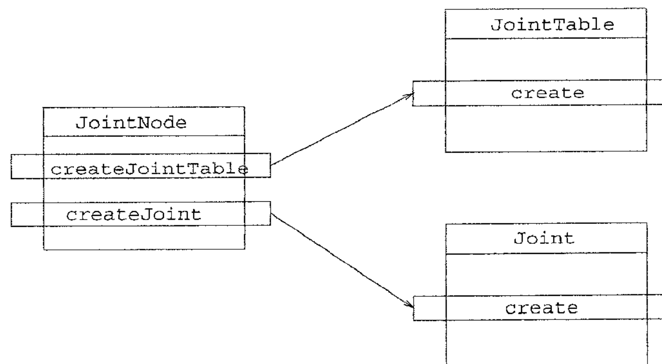

Fig. 7: Object Communication Diagram between JointNode, JointTable and Joint

Textual Specification

We can now derive from the diagrammatic specification a textual one. For each object class depicted in the community diagram we generate a textual specification fragment. The textual specification of an object class is devided into a signature part and a behavior part. The former consists of the declaration of attributes, actions and components and the latter is given by the definition of operations and local constraints.

```
object class JointNode
  attributes
    JNr: nat initialized 1;
  components
    Joint(nr:nat) : Joint;
    JointTable : JointTable;
  actions
     * new;
     createJoint;
     createJointTable;
  behavior
    createJoint
      do Joint(JNr).create; JNr:=JNr+1 od;
    createJointTable
      do JointTable.create od;
end;
```

We start with the object class JointNode. There are two components: A joint node has several joints which are identified by a number (Joint(nr:*nat*)) and a joint table (JointTable) as components. Moreover, an attribute JNr is specified to save the current joint number. The initial value of the attribute is 1. Thus, after a JointNode has been created by new, JNr has value 1. There are two actions specified, one for creating joints another one for creating a joint table. The former one takes the current joint number JNr, increments JNr, calls synchronously the birth action in Joint, and assigns the new joint to the name Joint(JNr). These birth actions are specified in the corresponding object classes, i.e., JointTable and Joint, respectively.

```
object class JointTable
  attributes
     Joints : list(Joint);
  actions
     * create;
     insertJoint(j:Joint);
  behavior
     insertJoint(j:Joint)
       do Joints:= append(Joints,j) od;
end;
```

The object class `JointTable` has a list of joints as attributes. These are object-valued attributes. In contrast to components, object-valued attributes do not belong to `JointTable`, instead they are readable from `JointTable`. In this sense object-valued attributes are links to other objects such that their attributes can be read and used for some computations or their actions can be called. Besides the creation action there is one action specified to append new joints to this list, i.e., to append further links. `insertJoint` is the action which takes a joint as parameter and appends this joint to the list `Joints`.

Before we specify `Joint` we introduce the object classes `ConstJointPart` and `ExpJointPart`, as well as the generalization of both `JointPart`. The object class `JointPart` comprises all attributes which are also part of the specializations. Every joint has a `gap`, a `width`, and further attributes named `a`, `b`, etc. See the table of flame path joints in figure 3 where these attributes appear. The creation of a `JointPart` is parameterized by the necessary values for the joint attributes and an indication, if the joint shall be an explosion joint or a constructive joint.

```
object class JointPart
   attributes
      gap : real;
      width : real;
      a : real;
      b : real;
        ...
   actions
      * create(kind:enum(constructive,explosion), gap:real,
                    width:real, a:real, b:real, ...)
      ...
end;
```

The object classes `ConstJointPart` and `ExpJointPart` are specializations of `JointPart` which have further attributes. In principle inheritance in TROLL is monotone. That means, that we carry over all axioms of the superclass into the subclass. The behavior of the subclass is in full compliance with that of the superclass. For our case study, we specialize `JointPart` to `ConstJointPart` which has an additional derived attribute:

```
object class ConstJointPart
  aspect of JointPart if create(kind, gap, width, ...) and kind = constructive;
  attributes
   l : real derived (a+b);
  actions
     ...
end;
```

Now we come to the object class `Joint`:

```
object class Joint
  components
    cons(1..3) : ConstJointPart;
    exp(1..2) : ExpJointPart;
  attributes
    row : list(record(a: real, ..., gap:real, l: real))
              derived concat(toList(select jp.a, ..., jp.gap, jp.l
                                     from jp in range(cons)),
                             toList(select jp.a, jp.b, ..., jp.gap, 0.0
                                     from jp in range(exp)));
  actions
    * create;
      ...
end;
```

An object of class `Joint` has up to five components. Three components are constructive joint parts and another two are experimental joint parts. The former ones are those which will be derived from the construction drawings, whereas the latter are fixed by explosion test done by the operators in the labs. There is an attribute called `row` for joints. This attribute corresponds to one row of the table of flame path joints in figure 3. The sort of this attribute is quite complex. This is due to the fact that in `row` the information of all components are collected. We specified a `select` statement to extract this information and exploited this way the logic calculus which provides concepts for querying object states. We explain this by starting from the innermost select clauses: The select clause returns a bag of records. Each record incorporates five real numbers representing `width`, `gap`, etc. of one joint. We query the constructive joints as well as the experimental joints. We get all joints by the implicitly defined operation `range`. `Range` gives the set of all elements of the co-domain of the declared component, i.e., all existing component objects. Here, `range(cons)` delivers all constructive joint parts. We select the values of the attributes and transform the bag to a list. To be able to concatenate experimental and constructive joint part list, we introduced 0.0 as `l` (length-) value of `ExpJointPart`. The result of this concatenation is one `row`.

Up to now we only specified object classes. Thus, we still have no object instances. These will be generated by declaring the system's objects. We simplify our sets of instances such that we have one object of class `staff` and one object of class `JointNode`. Anyhow, the object identifiers may be parameterized. Thus we can declare infinite sets of objects. The TROLL specification of `staff` corresponds to the behavior diagram in figure 6.

```
object class staff
  actions
    * login;
    newJoint;
    buildJointTable;
      ...
end;

objects JN:JointNode end;

objects Manager:staff
   behavior
     newJoint
       do JN.createJoint od;

     buildJointTable
       do JN.createJointTable od;
end;
```

We showed a simplified part of the specification of the UoD of JointNode and staff. The full specification encompases several pages of TROLL specification. Besides the expressive power one of the main advantages of our approach is the well-defined semantics.

4.3. Semantics

We will define the semantics of our case study in terms of the notions given in section 3. According to section 3 we will reflect the statical part of the system through an extended data signature. Each object class in the diagram establishes an object sort $b \in S_O$. For instance, we have as object sorts JointNode, JointTable, Joint, ConstJointPart, ... $\subseteq S_O$. In section 3 we pointed out that each object sort gives rise to two data sorts, i.e., object identities S_O^i and object actions S_O^a. The former are fixed by the instance declarations. Thus, Manager∈ $\texttt{Manager}^i$ and JN∈ $\texttt{JointNode}^i$. Each object will constitute a node in the system and each node will be interpreted by a sequential event structure. We will later come back to this. Object actions are given by the specification, i.e., we have createJoint, createJointTable ∈ $\texttt{JointNode}^a$.

Passing some technical details which can be found in [11] we arrive at an *instance signature* $\Sigma_I = (Id, Ac)$ with

$$\begin{aligned} Id &= \{\texttt{Manager}, \texttt{JN}\} \\ Ac_{\texttt{Manager}} &= \{\texttt{newJoint}, \texttt{buildJointTable}\} \\ Ac_{\texttt{JN}} &= \{\texttt{new}, \texttt{createJoint}, \texttt{createJointTable}, \\ &\quad \texttt{Joint(1).create}, \texttt{Joint(2).create}, \ldots, \\ &\quad \texttt{JointTable.create}, \texttt{JointTable.insertJoint(Joint(1))}, \ldots\}. \end{aligned}$$

So far we only reflected the statical part of the specification. TROLL features like interactions and attribute updates determine the behavior of objects of the corresponding class. To cover the behavior formally we use temporal logic. For illustration purposes we translate parts of JointNode into temporal formulas. A birth action can only take place at the beginning of an object life cycle. After it has occurred it cannot be executed for the rest of the object life. Thus, we receive for every object O of class JointNode, O∈ $\texttt{JointNode}^i$, x∈ int the following formula

$$\texttt{O} : \odot\texttt{new} \Rightarrow \mathsf{G}_{\texttt{O}} \neg \rhd \texttt{new}.$$

The formula asserts that whenever the birth action has occurred, in the future it will never be enabled.

All other actions are enabled after the execution of the birth action. The following formula reflects this:

$$\texttt{O} : \odot\texttt{new} \Rightarrow \rhd \texttt{createJoint} \wedge \rhd \texttt{createJointTable}.$$

The attribute changes of the specification intuitively expresses, that after the occurrence of a createJoint action attribute JNr is increased by one. Therefore, whenever it was possible to read a value n for the attribute in a previous state, and when action createJoint happened in the current state, it must be possible to read the value n+1 for the attribute. The reading of attributes is expressed via an action r.

$$\texttt{O} : \mathsf{Y}_{\texttt{O}} \rhd \texttt{JNr.r(x)} \wedge \odot\texttt{createJoint} \Rightarrow \rhd \texttt{JNr.r(x+1)}$$

The local interaction between components of JointNode is translated into the following formula:

$$\texttt{O} : \odot\texttt{createJoint} \wedge \texttt{JNr.r(x)} \Rightarrow \odot\ \texttt{Joint(x).create}.$$

That means, whenever a joint is created with a specific number synchronously the birth action has to take place in the corresponding component.

The global interaction between Manager and JointNode corresponds to:

$$\texttt{Manager} : \odot\texttt{newJoint} \Rightarrow \odot\ \texttt{JN.createJoint}.$$

We only illustrated the translation of some TROLL concepts to temporal logic formulas.

Now we are able to explain the interpretation structures. Models for single objects are sequential event structures. In figure 8 we depicted parts of the models of Manager and JN. Events are framed and labelled with the actions which occur. Lines between events denote causality.

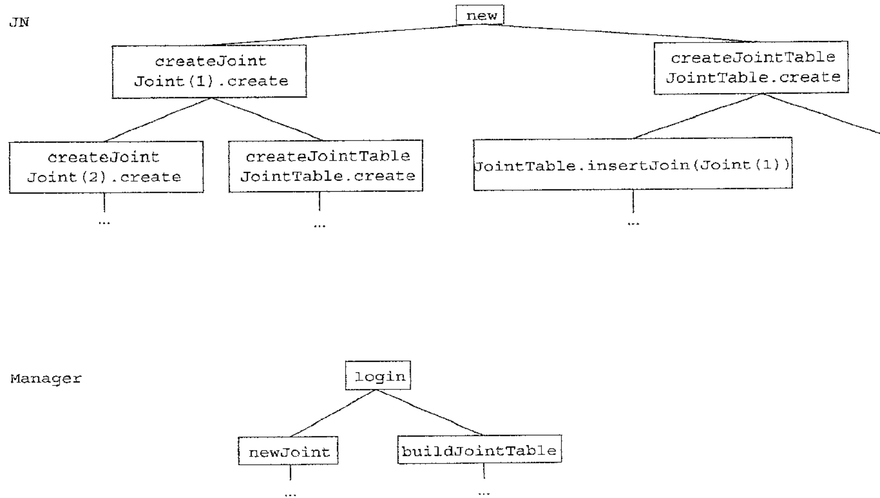

Fig. 8: Sequential Event Structures of JN and Manager

Each branch of a sequential event structure is a possible run of the corresponding object. For JN the following life cycles are depicted. After the creation of JN either a new joint is created or a joint table is created. In the former case two actions take place concurrently. This corresponds to the interaction rule specified for `createJoint` in `JointNode`. Analogously, the `createJointTable` takes place together with the birth action in the corresponding component. After the creation of the joint table, joints may inserted. Similarly, after the creation of a joint further actions may take place.

For `Manager` we specified the beginnings of two life cycles. After the object has been created, a `Manager` may build a new joint table or a new joint.

In sequential event structures events are either causally related or in conflict. There is no concurrency in sequential models. Concurrency comes into play when these sequential event structures are put together to form a concurrent event structure via shared events. For instance, in figure 9 we illustrate how this is done. After the concurrent creation of both objects JN and `Manager` either the `Manager` may create a new joint and by this calls the `createJoint` action of JN, which again calls `Joint(1).create`. Thus, three actions take place concurrently, one of the object `Manager` and another two of the compound object JN. Analogously, the right branch in figure 9 represents the life cycle, where after the concurrent creation of both objects a joint table is built. To summarize, in figure 9 the creation events are concurrent, whereas the other two events are in conflict and therefore, denote different possible runs.

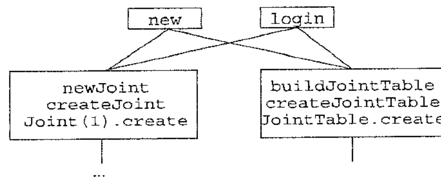

Fig. 9: Concurrent Event Structure

5. EXPERIENCES

The first phase of the project described in this paper has been terminated. We have finished the modelling of the Universe of Discourse of CATC and we have implemented the subsystem of CATC for one out of three laboratories. There have been several positive experiences with using object-oriented formal techniques in the design process.

The first attempts of managing the project with a popular object oriented analysis and design method [25] failed. The team that is developing the system is composed of students and full-time employees. Students did not just have to cope with implementation tasks, they were also involved in the modelling of the system. Since they typically stay for 6 months in the project we had a high personnel fluctuation. Students that left the team took a lot of knowledge that was supposed to be documented in their models with them. This was the information supposed to be giving the semantics of the models. Due to the informality there was no common understanding of many models and a lot of things had to be discussed over and over again whenever new people entered the project. This was one of the major reasons for us to restart the project following a formal approach. It turned out to be much less critical when members leave the project. The documentation they leave is less ambiguous.

Since the start of the project in 1994 the team has always been composed out of 8-12 students on average and three full-time employees. All team members have similar backgrounds (computer science, mathematics) and therefore use the same terminology. One employee is settled in the federal board and fortunately has a background of computer science and of the problem domain. The two other employees are settled in the university with special interests in formal methods and mathematics. But none of the students had knowledge about TROLL so we trained them in a special TROLL seminar taking place every two weeks at the very beginning. Both employees at the university are TROLL specialists and one of them is the designer of additional TROLL concepts. He spent a lot of time on answering specific questions, the students had.

An advantage often mentioned in relation with formal method is the possibility to verify correctness. The verification issue is not of central importance in our project.

The project is located in a federal institute but many developers are students. These students have to communicate with staff member respectively operators. The gap between students and federal board employees in their understanding of technical and administrational processes is evident. Therefore it was important to improve the communication skills. Here especially the support by diagrammatic notations had proven valuable and was confirmed by all project members. The diagrams are based on concepts well known in computer science e.g., entity/relationship diagrams (community diagram), finite automata (behavior diagram) or programming languages (textual representation of TROLL). Indeed, both the diagrams and the TROLL text were intuitive for students as well as for federal board employees. Furthermore the fact that they had to develop with TROLL and thus were compelled to formalize their ideas, brought out a lot of misunderstanding in early stages. Here we had to handle the usual problem, that the federal board employees had some ideas about what they needed in general but not in all details. We had long term discussions about the overall model and this lead in our opinion to a quite good understanding of the general setting of the PTB world.

The specification phase is an iterative process since the discovered misunderstandings came up gradually. Here more tool support is necessary. Re-specifying is an important part of the work and re-doing already developed parts over and over is disappointing enough. Tool support turned out to be one key factor in order to avoid frustration when having to change a model again.

The development process of the project started with an analysis phase, where small groups of students got an overview in specific sub-problem domains. During several meetings all team members contributed to develop a first rough global architecture. Afterwards we divided the complex task in smaller units to be worked on in groups of 1-2 students. It turned out that the following, highly iterative, non-sequential process was followed by all students to widen their expertise in their sub-application domain (s. figure 10).

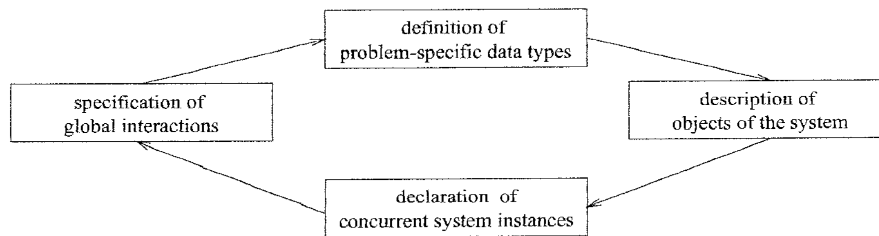

Fig. 10: Development Process

definition of problem-specific data types A lot of interviews with staff members or operators and investigations of reports and formulas used so far in PTB is necessary to understand in detail the data to be processed in the information system. This knowledge has to be converted into user-defined data types.

description of objects of the system Besides the data types the objects of the systems have to be identified. Starting with a set of objects, their structure and behavior has to be fixed subsequently. Attributes and components are specified, actions are fixed and their effect of attributes is defined. Constraints on the behavior of complex objects are modelled. This also includes the specification of interaction constraints in aggregated objects.

declaration of concurrent system instances One main design decision is made when it comes to the point where the degree of concurrency is established. In this step objects are either aggregated to complex objects or remain concurrent to other objects of the system.

specification of global interactions Due to the degree of concurrency, global interactions have to be specified. It is necessary to get a deep understanding of the business process to be able to appropriately model the work- and data-flow.

These are the main four steps of the design process. Each step has to be refined and re-done several times and in all of these steps graphical as well as textual TROLL can be used. It turned out that the specification phase is an iterative process since the discovered misunderstandings came up gradually. This implies re-specification of all parts: user-defined data types, specification of objects, changing the degree of concurrency together with the global interactions. The students had to communicate frequently to clarify and refine interface definitions. Besides bilateral discussions we had regular meetings with all team members to propagate important design decisions concerning interfaces. The global architecture has not been changed rather than refined. Thus, roughly spoken the specification is twofold: First we dealt with more general aspects using mainly diagrams and secondly we attacked more fine grain problems with the textual notation. This involves two different views of the world, a global and a detailed one. The advantage is that we achieve first a rather stable global view before we consider details. Changes to the finer grained specification documents did not affect the global view.

We had very positive experiences concerning the integration of the various system parts modelled and implemented by different students. Due to the fact, that the students had to constrain themselves to the interfaces they agreed on with the other team members, there were very few problems in integrating the different modules. Thus, the formality and clearly defined semantics of the TROLL specifications carried over to the partly automatically derived implementations. The fact that specification is the most important part of the work is also reflected by the amount of time used for modelling and implementation, respectively. On average 70% of the time was spent on specification purposes whereas only 30% of the time was spent on implementation including integration.

Another advantage of formal specifications is the independence of implementation aspects, though implementations can be partly derived from specifications. As illustrated in figure 11 the functional requirements which are part of the specification translate to the implementation. On one hand, the part of the specification which describes the information to be stored together with the queries formalized in TROLL can be directly transformed into database definition construct as usually done in the logical design step of database. In our project we are using a relational database. Especially for this target database technology we gave general transformation rules of TROLL specifications to relational database schemes. But not only structural aspects are modelled. Since our approach is object-oriented, there is a big amount of knowledge how the system can evolve and behave. This is reflected by translating it into the target language of the application programs, which in our case is C++. For instance, information about attributes, actions and their effects on attributes, inheritance relations, etc. are directly translated into the corresponding language constructs in C++. Again, there exists general transformation rules. But not only single TROLL concepts are translated. We could also take advantage of more complex information. For instance, calling chains between several objects have been implemented using transactions. Thus, the functional requirements fixed by the formal model carry over to the implementation and are enlarged by non-functional requirements which result from the implementation language chosen, performance aspects, etc.

However, it should be clear to the reader that the non-functinal requirements are not less important than the functional ones. Non-functional requirements have been documented aside the functional in a separat document. We did use for this purpose plain German language and did refer when needed to the formal specification document. The separation of concerns led to a significant complexity reduction. In the early design phase people had not to worry about non-functional requirements and constraints. They could concentrate on the system's funtionalities. E.g. a timing constraint on an application transaction has at first hand nothing to do with the business function it shall implement, i.e., its functional description can be separated from the timing constraint.

Before starting the implementation the non-functional requirements had to be fixed. They had major impact on the following implementation phase, e.g. indexing of the database, data replication, multi-process architecture with synchronous or asynchronous communication etc.

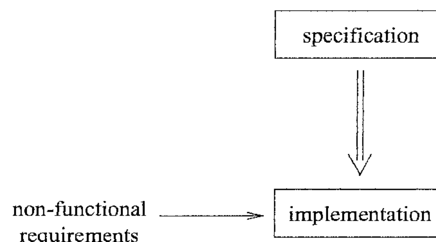

Fig. 11: From Specification to Implementation

The specification phase which we have already left behind, has much clarified our minds and helped us to understand what should be implemented. In [33] we predicted few problems with the implementation due to minor programming errors. Now after finishing the first implementation phase, we know that our forecast was right, although we did not perform any formal checking or verification of the implementation against the specification. However, we did a careful code review. Code and specification were exchanged between the different team members. Everybody had to review the code of another team member against the corresponding specification. Like this, many mistakes were eliminated at forehand.

We have finished the implementation for one laboratory and are currently working on implementing it for the other two laboratories. About 70000 lines of C++ code resulting from about 10000 lines of TROLL have been written so far. That makes a factor of 1:7. This proves that the efforts done in earlier project phases are not useless: Firstly, it is a lot easier to read the TROLL

specification to understand the problem domain rather than reading the C++ code. New students joined the project. They started their work with understanding the specification documents. But not only specification parts where helpful and could be used for the next two laboratories. Also the corresponding parts of the implementation have been reused. Secondly, a lot of things carry over from the specification to the implementation by partly automatically generating code from specifications.

The project will move from a pure national one to an international one. This will be the opportunity to get more experiences concerning reuse issues. Since most business facts and rules are formalized we expect that we can easily adapt our models to this new dimension of the problem domain. This potential future may prove the strategical advantage of our choice of a formal approach.

6. CONCLUSIONS AND OUTLOOK

In this paper we presented a field-study where we applied the formal specification language TROLL to the modelling of an Information System of a rather reasonable size. It was not our concern to give a concise language description. We do therefore dispense with a detailed comparision with other exisiting specification approaches. We are sure that other languages like TELOS [37], LCM [16], ALBERT [39], or OASIS [38] are also suited for the specification of such a system. It would be interesting to develop a common field-study of a reasonable size with the different approaches and try to evaluate their strength and weakness. A comparision of TROLL with several other approaches has been published by different authors [29, 6].

In the previous sections we introduced the industrial context of our application domain, picked one small part out of it and illustrated the specification of it. We said less about the implementation issues and the integration of existing and re-specified applications since the project is still in its initial state. We are sure that the next stages will bring up much more details and worthy information with respect to these subjects.

Furthermore we think about an automatic generation of application programs or frames from specifications. First steps in that directions have already been made, e.g., an approach to generate a relational database model from TROLL specifications [5] has been developed.

Tool support is crucial for the 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 propagted into the textual and from there furtheron 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. A first prototype of a TBENCHhas been implemented. It deals with the problem of coupeling the graphical and the textual TROLL specificaion. Details of this tool implementation can be found in [34, 35]. However as a prototype it does not yet have any multiuser capabilities, which are mandatory in huge projects. Most important are tools that allow for a fast change of specification documents while ensuring the consistency of the whole project. For presentation and discussion of the models we need documentation support. These documents have to show different views of the models. An example for a specification environment is the OBLOG workbench [15]. Besides providing comfortable support for the modelling of systems based on a mathematical formalism, it facilitates the generation of end applications which serve for model validation. Unfortunately, our system platform made it impossible for us to use this environment.

The reification from specification to implementation is one objective we want to reach in the near future. The first theoretical results have been developed [7]. We hope that future developments will provide us with a basis for a practical reification method that allows for error free implementations of specifications.

Acknowledgements — We thank all students which worked with us in the project. They gave us very valuable feedback concerning the adequacy of TROLL in information system design. Moreover, they made a supreme effort to lead the project to a success. We also thank the employees from PTB for their patience in discussing details of the application domain. Our activities in the CATC-project have been constructively criticized by our Braunschweig colleagues.

REFERENCES

[1] J. Bowen and V. Stavridou. The industrial take-up of formal methods in safety-critical and other areas: A perspective. In *[45]*, pp. 183–195 (1993).

[2] J.P Bowen and M.G. Hinchey. Seven more myths of formal methods: Dispelling industrial prejudices. In M. Naftalin, T. Denvir and M. Bertrani, editors, *FME'94: Industrial Benefit of Formal Methods*, pp. 105–117, LNCS 873, Springer, Berlin (1994).

[3] F.P. Brooks. No silver bullet – Essence and accidents of software engineering. *IEEE Computer*, **20**(4):10–19 (1987).

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

[5] C. Danker. Transformation of TROLL object specifications into schema of relational databases. Diploma Thesis at Techn. Univ. Braunschweig, *in German* (1995).

[6] A. Delgado-Ruiz, D. Pitt and C. Smythe. A review of object-oriented approaches in formal methods. *The Computer Journal*, **38**(10):777–784 (1995).

[7] G. Denker and H.-D. Ehrich. Action reification in object oriented specification. In R. J. Wieringa and R. B. Feenstra, editors, *Information Systems - Correctness and Reusability, Selected Papers from the IS-CORE Workshop*, pp. 103–118. World Scientific (1995).

[8] G. Denker and P. Hartel. TROLL – *An Object-Oriented Formal Method for Distributed Information Systems Design: Syntax and Pragmatics.* Informatik-Bericht, Technische Universität Braunschweig, *To appear* (1997).

[9] E.H.Dürr and J.v. Katwijk. VDM++, A formal specification language for object-oriented design. In *Proceedings of TOOLS7 (Technology of object-oriented languages and systems)*, Prentice-Hall (1992).

[10] H.-D. Ehrich, R. Jungclaus, G. Denker and A. Sernadas. Object-oriented design of information systems: Theoretical foundations. In J. Paredaens and L. Tenenbaum, editors, *Advances in Database Systems, Implementations and Applications*, pp. 201–218. Springer Verlag, Wien, CISM Courses and Lectures no. 347 (1994).

[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*, pp. 219–235, Springer, Berlin, LNCS 906 (1995).

[12] H.-D. Ehrich, A. Sernadas and C. Sernadas. Abstract object types for databases. In K. R. Dittrich, editor, *Advances in Object-Oriented Database Systems*, pp. 144–149, Bad Münster am Stein, LNCS 334, Springer, Berlin, 1988 (1988).

[13] *VDE: EN 50014 Elektrische Betriebsmittel für explosionsgefährdete Bereiche - Allgemeine Bestimmung.* VDE-Verlag (1978).

[14] *VDE: EN 50018 Elektrische Betriebsmittel für explosionsgefährdete Bereiche - druckfeste Kapselung 'd'.* VDE-Verlag (1978).

[15] Espirito Santo Data Informatica, Lisbon. *OBLOG CASE V1.0 – The User's Guide* (1993).

[16] R. Feenstra and R. Wieringa. LCM 3.0: A Language for Describing Conceptual Models – Syntax Definition. Report ir-344 , Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam (1993).

[17] M. Gogolla and U. Hohenstein. Towards a semantic view of an extended entity-relationship model. *ACM Transactions on Database Systems*, **16**(3):369–416 (1991).

[18] T. Günther, K.-D. Schewe and I. Wetzel. On the derivation of executable database programs from formal specifications. In *[45]*, pp. 351–366 (1993).

[19] W. Wettisch, H. Olenik and H. Rentzsch. *Handbuch für den Explosionsschutz.* W.Girardet, Zürich (1971).

[20] P. Hartel. *Conceptual Modeling of Information Systems as Distributed Object Systems.* PhD thesis, Technische Universität Braunschweig, in German (1996).

[21] P. Hartel and R. Jungclaus. Modeling business Processes over objects. *Int. Journal of Intelligent and Cooperative Information Systems, To appear* (1995).

[22] T. Hartmann. *Entwurf einer Sprache für die verhaltensorientierte konzeptionelle Modellierung von Informationssystemen*, volume 1 of *Reihe DISDBIS.* infix-Verlag, Sankt Augustin (1995).

[23] R. Herzig. *Zur Spezifikation von Objektgesellschaften mit* TROLL *light.* Fortschritt-Berichte Reihe 10, Nr. 336, VDI-Verlag, Düsseldorf (1995).

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

[25] T. Hohnsbein and H. Schafiee. *Reengineering des Programms DRUCKMESS in der PTB.* Project work at Techn. Univ. Braunschweig (1994).

[26] H. Rechenberg, J. Bortfeld and W. Hanser. *100 Jahre Physikalisch-Technische Reichsanstalt/Bundesanstalt 1887-1987.* VCH Verlagsgesllschaft, München (1987).

[27] I. Jacobson. *Object-Oriented Software Engineering.* Addison-Wesley, Reading, MA (1992).

[28] C.B. Jones. *Systematic Software Development using VDM.* Prentice-Hall, Englewood Cliffs, NJ (1989).

[29] R. Jungclaus. *Modeling of Dynamic Object Systems—A Logic-Based Approach.* Advanced Studies in Computer Science, Vieweg Verlag, Braunschweig/Wiesbaden (1993).

[30] R. Jungclaus, G. Saake, T. Hartmann and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems,* **14**(2):175–211 (1996).

[31] R. Jungclaus, R.J. 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,* pp. 35–42. Springer, Informatik aktuell (1994).

[32] M. Kowsari and P. Hartel. Ein Fallbeispiel zur Evaluation einer Objektorientierten Methodik. In C. Eckert, H.J. Klein and T. Polle, editors, *7. Workshop Grundlagen von Datenbanken,* pp. 88–93. Universität Hildesheim Institut für Informatik (1995).

[33] 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),* pp. 136–159. Springer, Berlin, LNCS 1080 (1996).

[34] J. Kusch, P. Hartel, T Hartmann and G. Saake. Gaining a uniform view of different integration aspects in a prototyping environment. In *Proc. 6th Int. Conf. on Database and Expert Systems Applications (DEXA'95),* pp. 38–47, Springer Verlag, Berlin, LNCS 978 (1995).

[35] J. Kusch, P. Hartel, T. Hartmann and G. Saake. Integration einer Prototyping-Umgebung durch Objektorientierte Spezifikation. Preprint 5, Univ. Magdeburg, Fakultät für Informatik (1995).

[36] S.L. Meira and A.L.C. Cavalcanti. Modular object-oriented Z specifications. In *Z User Workshop, Oxford,* Springer-Verlag (1990).

[37] J. Mylopoulos, A. Borgida, M. Jarke and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems,* **8**(4):325 – 362 (1992).

[38] O. Pastor, F. Hayes and S. Bear. OASIS: An object-oriented specification language. In P. Loucopoulos, editor, *Proceedings of the CAiSE'92 conference,* pp. 348–363, Berlin, Springer, LNCS 593 (1992).

[39] P. Du Bois. *The Albert-II Language – On the Specification and the Use of a Formal Specification Language for Requirements Analysis.* PhD thesis, Facultes Universitaires Notre-Dame de la Paix Namur, Belgien (1995).

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

[41] G. Saake and R. Jungclaus. Specification of Database Applications in the TROLL-Language. In D. Harper and M. Norrie, editors, *Proc. Int. Workshop Specification of Database Systems, Glasgow, July 1991,* pp. 228–245, Springer, London (1992).

[42] G. Saake, R. Jungclaus and T. Hartmann. Application modelling in heterogeneous environments using an object specification language. *Int. Journal of Intelligent and Cooperative Information Systems,* **2**(4):425–449 (1993).

[43] 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,* pp. 107–116. VLDB Endowment Press, Saratoga (CA) (1987).

[44] J.M. Spivey. *The Z Notation - A Reference Manual.* Prentice-Hall, Englewood Cliffs, NJ (1989).

[45] J.C.P. Woodcock and P.G. Larsen, editors. *FME'93: Industrial-Strength Formal Methods.* LNCS 670, Springer, Berlin (1993).