Developing an Information System using TROLL - an application field study *

M. Krone**, M. Kowsari, P. Hartel, G. Denker, H.-D. Ehrich

Technische Universität Braunschweig, Informatik, Abt. Datenbanken
Postfach 3329, D–38023 Braunschweig, Germany
e–mail: {M.Kowsari|P.Hartel|G.Denker|HD.Ehrich}@tu-bs.de

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 effects in the future.
keywords: 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 [Bro87]. There is a small but growing community of people who propose and promote formal methods in Software Engineering [WL93, BH94]. 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 [GSW93].

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 [Har95] [BS93]. In today's organizations we do not find many people who know formal methods [BH94]. This means we have to invest in their education. If this investment is

* Work reported here was partially supported by CEC under ESPRIT-II Basic Research Working Group No. 6112 COMPASS and by CEC under ESPRIT BRA WG 6071 IS-CORE, PTB, OBLOG SOFTWARE S.A. Lisbon.

** Now at Siemens AG, Transportation Systems, Systems Engineering, P.O.Box 3327, D-38023 Braunschweig, email:Maren.Krone@bwg4.erl1.siemens.net

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) [Esp93]. The formalism allows us to build intelligent tools which allows us to speed up development drastically. Graphical representations help to overcome the fear of embarking on formalisms. Methodological guidelines [BS93] 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 computer aided testing and certifying (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. At the end of the year it got clear that the chance of success with the chosen approach was rather low [HS94]. At that time we decided to use a formal approach [KH95]. 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 almost one year we have already collected several experiences, positive as well as negative ones. Furthermore, we will give some hints, why our first approach without formal methods did not succeeded.

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 [HJ95]. 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 understanding is the prerequisite for deciding which part of the organization shall be computerized and how this system is embedded into the organization.

An abstract model of the organization can help us to achieve the required understanding. This model has to cover all aspects which are relevant with respect to the organizational activities. These aspects define what we call the Universe of Discourse (UoD).

Based on the UoD model we decide what will actually be supported by the Information System. The model defines the functional requirements of the later system. It abstracts from non-functional requirements, like technologies that shall be used for implementation.

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 [Spi89], VDM [Jon89] do not provide the right level of abstraction for modelling. Further on they emphasize on structural aspects and do not allow for an intuitive 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 [Hoa85] 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 [RBP+91] or "Object-Oriented Software Engineering" [Jac92] 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. Some formal specification languages have already object oriented extensions e.g., VDM++ [DK92], MooZ [MC90]. 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 [JSHS96]. The TROLL approach incorporates many ideas which have been developed over the past 8 years. Much work in the theoretical foundations [SSE87, ESS88, EJDS94, DE95, ES95] and on methodological [SJ92, SJH93, HJ95] 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 [KHHS95] a specification tool for TROLL is available. The TROLL method [JWH+94] 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 first 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 Sect. 4 a small part of the conceptual model, some methodological guidelines and the relationship between the mathematical formalism and the conceptual model. Our first experiences are summed up in Sect. 5. We end the paper in Sect. 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[3] (PTB).

The PTB [JB87] 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

[3] federal institute of weights and measures

of SI units [4], cooperation in national and international technical committees, physical safety engineering serving the protection against explosions etc.

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 [EN 78a, EN 78b]. 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 and modelling 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.

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

1. support *experimental test* like PRESSTEST JOINTTEST and others. JOINTTEST will serve as the case study of this paper, which will be introduced in detail in Sect. 4.
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

[4] international system of units

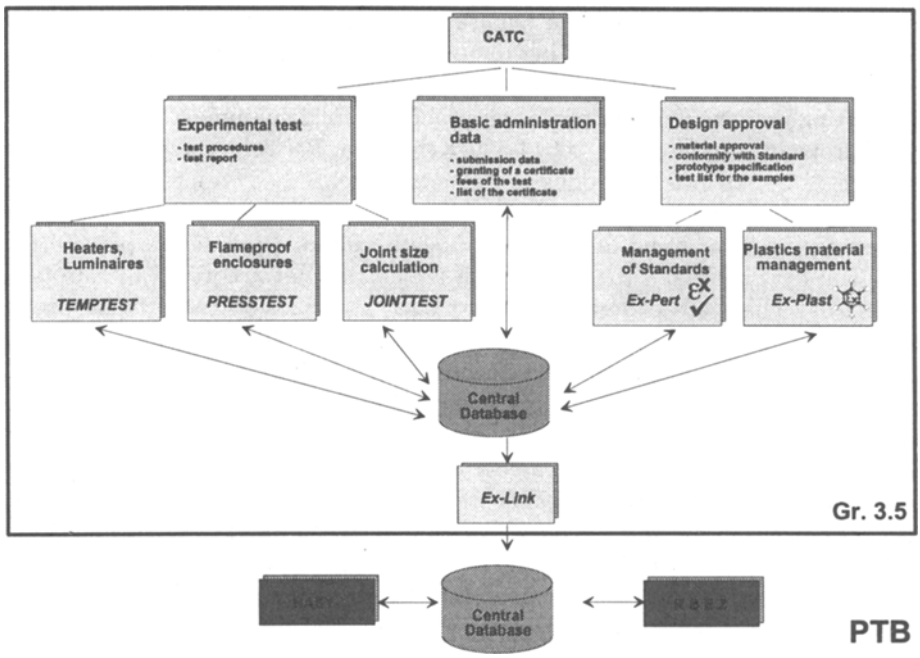

Fig. 1. CATC – Overview

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 flameproof 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[5], HASY[6]) 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.

[5] archive and documentation application
[6] settlement and calculation application

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 model 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.

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*, *object type*, and *object class* specifications.
- *Parameterized data types* allow for the construction of new date types based on a fixed universe of predefined data types.
- An object type specification consists of a set of *attributes*, *actions* and *constraints*. Attributes describe the state of an object of that type and the actions determine the possible object evolution. Constraints allow for the definition of static and transitional invariants over the object state.
- Object types may be constructed over other object types (*aggregation*). Such types 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 type may be the *specialization* of another object type. The specialized type 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 supertype are always satisfied by an object of the subtype.
- Object classes are declared over object types. They describe the potential sets of objects in the system. Interactions between the objects of different classes describe the global synchronization relations.

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

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 [ES95]. In the following we intuitively explain these semantic ingredients. Moreover, in Sect. 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 [Her95, GH91]. 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 [Her95, GH91].

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 [ES95] 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 *sometimes in the future*) and for the past (*yesterday* and *sometimes 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 $\triangleright\alpha \in L_\Sigma(X)$ (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), $\mathtt{F}_i\varphi \in L_\Sigma(X)$ (sometimes in the future), $\mathtt{Y}_i\varphi \in L_\Sigma(X)$ (yesterday), and $\mathtt{P}_i\varphi \in L_\Sigma(X)$ (sometimes in the past);

We will give some examples of formulas in Sect. 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 rooted 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 for 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 Sect. 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 Sect. 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 describe in detail the relevant aspects. 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 comes along with the graphical notation OMTROLL. After a brief introduction to the development methodology of TROLL we partially present the design of the JOINTTEST. The specification of JOINTTEST which corresponds to the real world is much more complex than the restricted version we present in this paper. We restrict ourselves because of space limitations. Instead of explaining the whole complexity we rather give an intuitive specification of JOINTTEST explaining most of the concepts of TROLL and illustrating their use in requirements analysis and design specification.

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

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 [HO71].

In Fig. 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 Fig. 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.

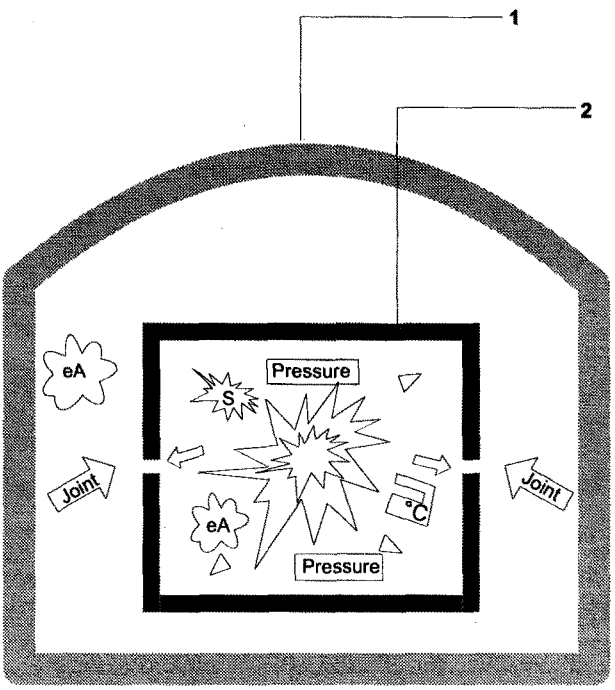

Fig. 2. Flame proof joint test, 1: autoclave , 2: enclosure, eA : explosion atmosphere, s: spark, $°C$: surrounding temperature

Modelling the Universe of Discourse with TROLL

In Sect. 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. The process of finding them is a rather creative one and we believe that there do not exist prefixed rules for it. However, if an entity has been identified to be relevant, we can follow some methodological guidelines to build a model of it.

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 requirements analysis 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 client does not yet know, what he wants, the formalism helps him and the

Joint	Data acc. to EN 50 018 - 1977/ VDE 0171			all to construction drawing						test sample		
	Minimum length of joint	Distance l (for bores within flamepath)	Maximum width of joint i_K (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		Length of joint L	Distance l	Width of testing gap i_E
								min.	max. $= i_C$			
K1	12.5		0.15	140	-	$\frac{7.5+00.5}{7.45-0.05}$		0.05	0.15	15.42	$\frac{7.57}{7.333}$	0.225
K2	12.5		0.15	141	-	$\frac{7.5+00.5}{7.45-0.05}$		0.051	0.148	15.43	$\frac{7.57}{7.333}$	0.227

Fig. 3. Table of flame path joints

developer in understanding the general setting of the problem domain.

The diagrams also model different aspects like communication, object composition and hierarchies etc. of the system. The textual representation in TROLL syntax is the result of the design specification stage. There is a smooth boundary between these two life cycles and therefore we prefere the database terminology of conceptual design (fixing the functional requirements, UoD) and logical design (textual representation of the model in TROLL syntax). Together they form an evolutional software engineering process consisting of iterations of analysis and design stages.

The following enumeration gives a short overview about the different diagrams textual notation respectively and their usage:

1. The *Community Diagram* (see Fig. 4) defines the static structure of the system. It consists of all object types, their composition and inheritance hierarchies, specialization and aggregation of object types 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 [JWH+94].
2. The next step is to define an *Object Declaration Diagram* (see Fig. 5) for each object type 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* (see Fig. 6) one can define explicitly the lifecycle 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 [Har88].
4. Then the *Object Communication Diagram* (see Fig. 7) depicts the communication between object types. 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 [CAB+94].
5. Finally the *Data Type Diagram* represents user defined data types over standard data types.
6. The result of the design is always a textual description in the TROLL syntax. It can now be written down and comprises the details represented in the figures. This is only a frame of the system specification which has to be refined by defining additional constraints, updates, ...

Please keep in mind, that there are usually several iterations of the following described process.

We start our specification of the UoD of JOINTTEST with the Object Community Diagram of JointNode which is depicted in Fig. 4. In this part of CATC we deal with the six object types: `JointNode`, `JointTable`, `Joint`, `ExpJointPart`, `ConstJointPart` and `JointPart`.

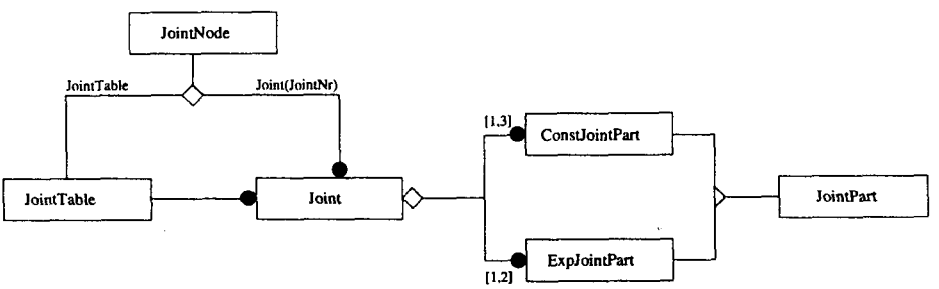

Fig. 4. Object Community Diagram of `JointNode`[7]

`JointNode` is the object type that depicts the special part of CATC concerning joint tests. In this universe we have joints and joint tables. We simplified the specification because of space limitation to one joint table and several joints. 1-n relationships between objects are shown as lines with filled circles at the object type which might occur more than ones. The diamond stands for aggregation of object types and the triangle is the diagrammatic notion for specialization. Thus `JointNode` is an aggregation of one `JoinTable` and one or more `Joints`. Joints

[7] Do not confuse the word table with relational database table.

can be constructed of several parts, a constructive part (`ConstJointPart`) and an experimental part (`ExpJointPart`). The constructive part is concerned with comparing data according to the standard with data according to the construction drawing (see Sect. 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 Fig. 3). There may be one to three constructive parts and one to two experimental parts. The object type `JointPart` depicts a specialization, which consists of those attributes and actions, the constructive and experimental parts have in common.

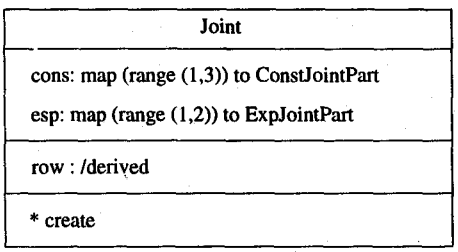

Fig. 5. Object Declaration Diagram of `Joint`

Now we can refine each component of the JointNode and represent it by Object Declaration Diagrams. The diagram for the object joint is represented in Fig. 5.

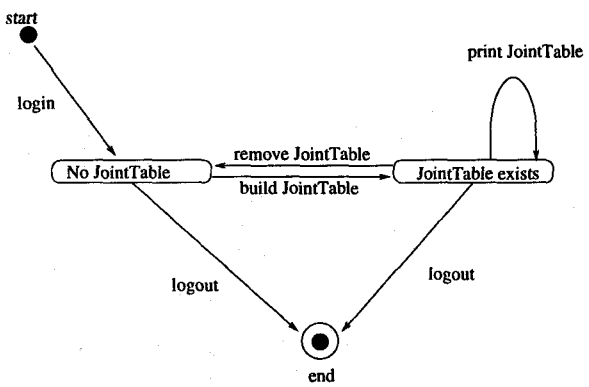

Fig. 6. Object Behavior Diagram of the staff

The behavior of the staff is illustrated in Fig. 6. A staff object is born by login and by it she is in the "NoJointTable" state. This is the beginning of the lifecycle of the object. A staff object may logout immediately after login and by this she

will leave the system. Therefore, logout is the *death* action of a staff object and terminates a life cycle. 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 she 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.

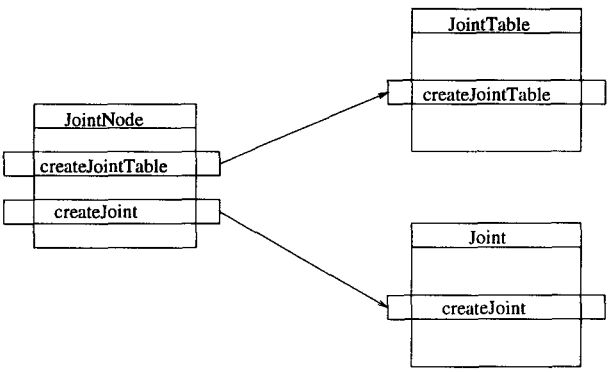

Fig. 7. Object Communication Diagram between `JointNode`, `JointTable` and `Joint`

The communication between object type JointNode and object types JointTable and Joint is depicted in Fig. 7. The action createJoint of object type JointNode corresponds directly to a createJoint action of the object type Joint. The action createJointTable of object type JointNode corresponds to a createJointTable action of object type JointTable. Here the simplification we made is easy to see: in the real world specification of JOINTTEST usually the relations between actions are more complex.

Now we can start with the last step of our specification procedure by collecting all developed details of the figures and constructing TROLL frames which can be enhanced by further details.

```
object type JointNode
uses JointTable, Joint, ...
components
    Joint(nr:int) : Joint
    JointTable : JointTable
attributes JNr: int initial 1
actions
   new  birth
   createJoint  calls Joint(JNr).create
                updates JNr:=JNr+1
   createJointTable calls JointTable.create
end
```

We start with object type JointNode. There are two components: A joint node has several joints which are identified by a number (Joint(nr:int)) 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 born 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, 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 types, i.e., JointTable and Joint, respectively.

```
object type JointTable
uses Joint
attributes Joints : listOf Joint
actions
   create birth;
   insertJoint(j:Joint) = updates Joints:= append(Joints,j)
end
```

The object type 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. JointTable has a list of joints as attributes. Besides the *birth* 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 types ConstJointPart and ExpJointPart, as well as the generalization of both JointPart. Object type 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 Fig. 3 where these attributes appear.

```
object type JointPart
uses real  ...
attributes
    gap, width, a, b ... : real
actions
     ...
end
```

The object types ConstJointPart and ExpJointPart are specializations of JointPart which have further attributes. The inheritance relation is monotone. That means, that we carry over all axioms of the supertype into the subtype. The behavior of the subtype is in full compliance with that of the supertype. For our case study, we specialize JointPart to ConstJointPart which has a derived attribute:

```
object type ConstJointPart
inherits JointPart  monotone
attributes
   l : real derived (a+b);
actions
      ...
end
```

Now we come to object type Joint:

```
object type Joint
uses ConstJointPart, ExpJointPart, ...
components
    cons : map (range (1,3)) to ConstJointPart
    exp  : map (range (1,2)) to ExpJointPart
attributes
    row : listOf(record(a: real,..., l: real))
          derived
           concat(toList(select jp.a, jp.b, jp.width, jp.gap, jp.l
                         from jp in range(cons)),
                  toList(select jp.a, jp.b, jp.width, jp.gap, 0.0
                         from jp in range(exp)))
actions
   create birth
end
```

An object of type 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 Fig. 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 for maps. Range gives the set of all elements of the co-domain of a map. 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 value of ExpJointPart. The result of this concatenation is one row.

Up to now we only specified object types. Thus, we still have no instances of the object types. Those will be generated by specifying object classes. For

space limitations we simplify our sets of instances such that we have one object of type `staff` and one object of type `JointNode`. The TROLL specification of `staff` corresponds to the behavior diagram in Fig. 6.

```
object type staff
actions
    login birth
    newJoint
    buildJointTable
end

object class Manager:staff
    interactions
        Manager.newJoint calls JN.createJoint
        Manager.buildJointTable calls JN.createJointTable
end
object class JN:JointNode    end
```

We showed a part of the specification of the UoD of `JointNode`. We abstracted from a lot of details because of space limitations. Though we illustrated the use of TROLL for specifying Information Systems of industrial size, especially, we explained the adequacy of its concepts. Besides the expressive power one of the main advantages of our approach is the well-defined semantics.

Semantics

We will define the semantics of our case study in terms of the notions given in Sect. 3. According to Sect. 3 we will reflect the statical part of the system through an extended data signature. Each object type 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 Sect. 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 *object class* definitions. Thus, `Manager`$\in$ `Manager`^{i} and `JN`$\in$ `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` $\in$ `JointNode`^{a}.

Passing some technical details which can be found in [ES95] we arrive at an *instance signature* $\Sigma_I = (Id, Ac)$ with

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

So far we only reflected the statical part of the specification. TROLL features like *calls*, *updates* determine the behavior of objects of the corresponding type.

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 type `JointNode`, $O \in$ `JointNode`^{i}, `x`$\in$ `int` the following formula

$$O : \odot \texttt{new} \Rightarrow F_O \neg \triangleright \texttt{new}.$$

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* actions. The following formula reflects this:

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

The *updates* part 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`.

$$O : Y_O \triangleright \texttt{JNr.r(x)} \wedge \odot \texttt{createJoint} \Rightarrow \triangleright \texttt{JNr.r(x+1)}$$

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

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

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 Fig. 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.

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.

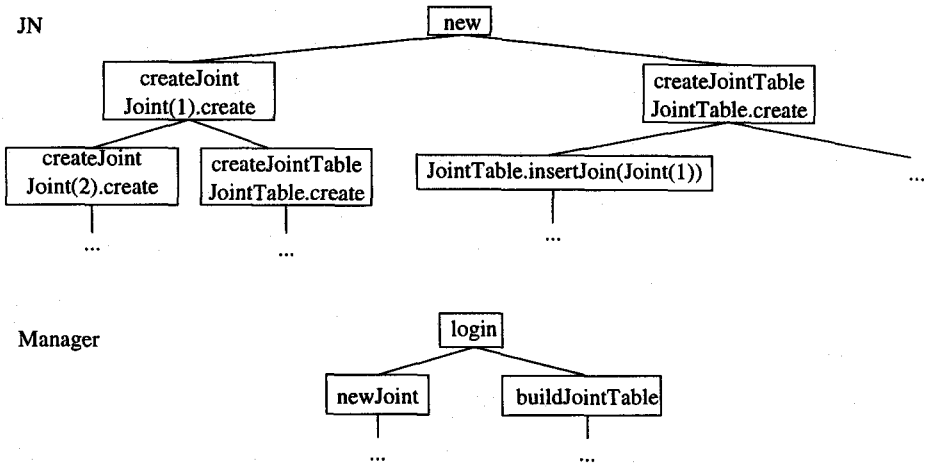

Fig. 8. Sequential event structures of nodes

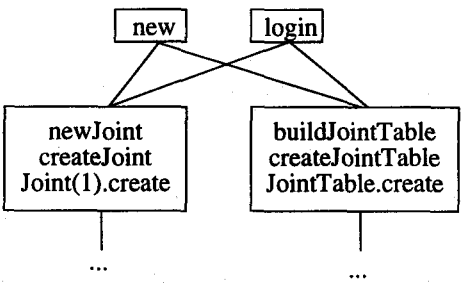

Fig. 9. Concurrent event structure

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 Fig. 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 Fig. 9 represents the life cycle, where after the concurrent creation of both objects a joint table is built. To summarize, in Fig. 9 the creation events are concurrent, whereas the other two events are in conflict and therefore, denote different possible runs.

5 Experiences

The project we described in this paper is in its initial state. We are still in the phase of modelling the Universe of Discourse of CATC. However, we made already several positive experiences.

The first attempts of managing the project with a popular object oriented analysis and design method [HS94] 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 modeling of the system. Since they typically stay for 6 months in the project we had a high personal 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.

The current team developing the system is now composed of 11 students 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 fortunal 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 to 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 an 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 the needed in gen-

eral 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.

Roughly spoken the specification is twofold: First we dealt with more general aspects by developing the 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.

Most probably the project will move from a pure national one to an international one. This turned out just recently. If this happens, we hope to have a high degree of reuse. 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.

The specification phase which we have already left behind, has much clarified our minds and helped us to understand what should be implemented. We are still working on the implementation at the moment. About 40000 of C++ code whith 100 object classes resulting from 5000 lines of TROLL have been written so far. That makes a factor of 1:8.

For the next steps we expect little problems with the how and we are almost sure that "we get what we want", due to the fact, that we solved the question of "what to do" in the specification phase. Of course we will have to deal with minor programming errors in the currently coding phase. But these are not the typical problems that result in a failure of the project.

As soon as the first user interface windows are compiled, we can test and verify the functionality of the system and thus compare wether our assumptions meet reality.

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.

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 [Dan95] 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 lead to a vast list of requirements for adequate tool support. 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 [Esp93]. 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 [DE95]. We hope that future developments will provide us with a basis for a practical reification method that allows for error free implementations of specifications.

References

[BH94] 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*, pages 105–117. LNCS 873, Springer, Berlin, 1994.

[Bro87] F.P. Brooks. No Silver Bullet – Essence and Accidents of Software Engineering. *IEEE Computer*, 20(4):10–19, Apr. 1987.

[BS93] J. Bowen and V. Stavridou. The Industrial Take-up of Formal Methods in Safety-Critical and Other Areas: A Perspective. In *[WL93]*, pages 183–195, 1993.

[CAB+94] D. Coleman, P. Arnold, S. Bodoff, S. Dollin, H. Gilchrist, F. Hayes, and P. Jeremes. *Object-oriented Development - The Fusion Method.* Prentice-Hall, 1994.

[Dan95] C. Danker. Transformation of TROLLobject specifications into schema of relational databases. Diploma Thesis at Techn. Univ. Braunschweig, 1995.

[DE95] 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*, pages 103–118. World Scientific, 1995.

[DK92] 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.

[EJDS94] 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*, pages 201–218. Springer Verlag, Wien, CISM Courses and Lectures no. 347, 1994.

[EN 78a] *VDE: EN 50014 Elektrische Betriebsmittel für explosionsgefährdete Bereiche - Allgemeine Bestimmung.* VDE-Verlag, 1978.

[EN 78b] *VDE: EN 50018 Elektrische Betriebsmittel für explosionsgefährdete Bereiche - druckfeste Kapselung 'd'.* VDE-Verlag, 1978.

[ES95] 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.

[Esp93] Espirito Santo Data Informatica, Lisbon. *OBLOG CASE V1.0 - The User's Guide*, 1993.

[ESS88] H.-D. Ehrich, A. Sernadas, and C. Sernadas. Abstract Object Types for Databases. In K. R. Dittrich, editor, *Advances in Object-Oriented Database Systems*, pages 144–149, Bad Münster am Stein, 1988. LNCS 334, Springer, Berlin, 1988.

[GH91] 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.

[GSW93] T. Günther, K.-D Schewe, and I. Wetzel. On the Derivation of Executable Database Programs from Formal Specifications. In *[WL93]*, pages 351–366, 1993.

[Har88] D. Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.

[Har95] T. Hartmann. *Entwurf einer Sprache für die verhaltensorientierte konzeptionelle Modellierung von Informationssystemen.* Reihe DISBD. infix-Verlag, Sankt Augustin, 1995. *To appear.*

[Her95] R. Herzig. *Zur Spezifikation von Objektgesellschaften mit TROLL light.* Fortschritt-Berichte Reihe 10, Nr. 336. VDI-Verlag, Düsseldorf, 1995.

[HJ95] P. Hartel and R. Jungclaus. Modeling Business Processes over Objects. *Int. Journal of Intelligent and Cooperative Information Systems*, 1995. *To appear.*

[HO71] W. Wettisch H. Olenik, H. Rentzsch. *Handbuch für den Explosionsschutz.* W.Girardet, Zürich, 1971.

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

[HS94] T. Hohnsbein and H. Schafiee. *Reengineering des Programms DRUCKMESS in der PTB.* Doppelstudienarbeit an der Universität Braunschweig, Abt.Datenbanken, 1994.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering.* Addison-Wesley, Reading, MA, 1992.

[JB87] H. Rechenberg J. Bortfeld, W. Hanser. *100 Jahre Physikalisch-Technische Reichsanstalt/Bundesanstalt 1887-1987.* VCH Verlagsgesllschaft, München, 1987.

[Jon89] C.B. Jones. *Systematic Software Development using VDM.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. TROLL – A Language for Object-Oriented Specification of Information Systems. *ACM Transactions on Information Systems*, 1996. *To appear.*

[JWH+94] 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*, pages 35–42. Springer, Informatik aktuell, 1994.

[KH95] 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*, pages 88–93. Universität Hildesheim Institut für Informatik, Juni 1995.

[KHHS95] 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)*, pages 38–47. Springer Verlag, Berlin, LNCS 978, 1995.

[MC90] S.L. Meira and A.L.C. Cavalcanti. Modular Object-Oriented Z Specifications. In *Z User workshop, Oxford.* Springer-Verlag, 1990.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design.* Prentice-Hall, Englewood Cliffs, NJ, 1991.

[SJ92] 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*, pages 228–245. Springer, London, 1992.

[SJH93] 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.

[Spi89] J.M. Spivey. *The Z notation - a reference manual.* Prentice-Hall, Englewood Cliffs, NJ, 1989.

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

[WL93] J.C.P Woodcock and P.G. Larsen, editors. *FME'93: Industrial-Strength Formal Methods.* LNCS 670, Springer, Berlin, 1993.