TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG

INFORMATIK-BERICHTE

94–05

A Basic Calculus for Verifying Properties of Synchronously Interacting Objects

Stefan Conrad

Abt. Datenbanken
Techn. Universität Braunschweig
Postfach 3329
D-38023 Braunschweig, Germany

Braunschweig

Mai 1994

A Basic Calculus for Verifying Properties of Synchronously Interacting Objects

Stefan Conrad

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

Abstract

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

Keywords: object specification, dynamic objects, verification, basic calculus, synchronous interaction

[0]Work reported here has been partially supported by the CEC under Grant No. 6112 (COMPASS) and BMFT under Grant No. 01 IS 203 D (KorSo).

Contents

1. Introduction

In the last years, formal specification techniques have received more and more attention (cf. [San88, BHL90, Wir90, AR93]). They are used for supporting the program and system development process. Assuming a correct specification of a system, each step within the software development process can or should be verified or validated to yield a correct result relative to the formal specification.

Especially, the development of complex software systems requires more and more formal support in order to get a grasp of the increasing requirements on software. Here, main criteria are reliability and correctness. Typically, such software systems can be characterized as reactive systems or information systems.

Among the different approaches object-orientation seems to be most adequate for capturing static and dynamic aspects of such systems. Object-oriented modelling of the Universe of Discourse (UoD) allows an easy-to-understand treatment of entities which are to be represented in an information system. Object-oriented modelling or specification is now widely accepted and understood and is based on semantic foundations which have been worked out in [Mes90, SE91, Wie91, EGS92, EDS93, e.g.].

A formal specification is an indispensable basis for proving properties of the system to be developed, e.g., the correctness of the final implementation can only be stated with regard to a formal specification. However, only considering the correctness of implementation steps cannot guarantee that the final implementation meets our needs. Therefore, it is necessary to convince oneself of the adequacy of specifications as early as possible. There are two main ways of checking adequacy. On the one hand we can validate the specification by generating prototype systems from the specification or by using an animation system. On the other hand we can formulate crucial properties of the information system (or of the objects constituting the system) and prove these properties on the basis of the given specification.

Here, we concentrate on the latter point. We present a basic calculus for formulating and proving object properties. It is a many-sorted, first-order sequent-based calculus which brings together the advantages of two other approaches ([FM91a] and [SSC92]). For the intended application area, the verification of object specifications for information systems, a propositional logic is not sufficient. We have restricted the admissible formulas to Gentzen-formulas which allows to work with a simple and comprehensible proof system.

We put the main emphasis of this work on the development of a calculus which can be used for different object-oriented specification languages. The calculus is intended to provide a common basis for verification purposes. Therefore, we have focused our attention to very elementary concepts. Nevertheless, we have taken care that further concepts can easily be integrated into the basic calculus and, thus, it becomes possible to tailor the calculus to specific needs.

The remainder of this paper is organized as follows. First, we sketch our notion of object and object community. Next, we introduce the syntax of the basic calculus. Then, an appropriate model theory based on linear (Kripke-) structures is outlined which captures hierarchical composition of object communities. Afterwards, we present a basic proof system for the calculus, before we discuss how compositionality of proofs corresponds to composition of object communities and their models. Finally, we conclude by pointing out some obvious and meaningful extensions and by briefly comparing our calculus with related work.

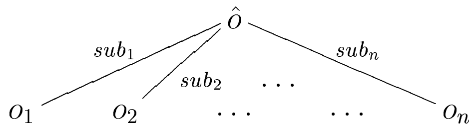
2. Objects as Units of Conceptual Modelling

In the sequel we briefly sketch our view of objects and object communities. For simplicity we focus on a small number of essential concepts.

First of all we have to state that we are considering dynamic objects, i.e., objects have an internal state which can be changed by the occurrence of events. Objects have attributes, and the values of these attributes determine object states. The attributes may be data-valued as well as object-valued. Furthermore multi-valued attributes are possible. Object-valued attributes have references to objects as values.

The dynamics of objects is based on local events. The occurrence of an event in some object causes a state transition to that object, i.e., changes of attribute values. The occurrence of events may be restricted by enabling conditions depending on the object state. Furthermore, we demand that at each moment there may be at most one event occurring in an object. Of course, in an object community there may be several events occurring simultaneously in different objects.

By sub-object relationships objects are organized into object hierarchies. Each object may have several sub-objects, but only one super-object. In consequence, the objects of an object community always form a tree induced by sub-object relationships. The following diagram depicts this fact in an abstract way. An object $\hat{o}$ has objects $o_1, \ldots, o_n$ as sub-objects. For each sub-object o_i there is a corresponding sub-object name sub_i given in $\hat{o}$:



In such an object hierarchy a super-object can be regarded as a manager object for its sub-objects. In consequence, we do not need to consider the additional concept of a "class", because a super-object acts as a class as well. Moreover, we get class attributes, meta classes (i.e., classes of classes), and further concepts required occasionally for free.

Between objects in an object community interaction is possible. The basic mechanism we are considering is event calling, i.e., the occurrence of an event in an object may require the occurrence of another event in another object. These events must occur simultaneously. If the called event is not enabled to occur then the calling event is also not allowed to occur. This mechanism realizes a kind of synchronous interaction.

Example 2.1

In Fig. 1 an object community state is shown. This object community is part of a li-

Figure 1: An object community state.

brary information system. For simplicity we only consider objects representing authors and one special object (called authorclass object) managing the authors. The authorclass object has an attribute `NumberOfAuthors` which always contains the current number of authors present in the library. Of course, there may be further attributes. We have certain events for creating and deleting the authorclass object (`createClass` and `destroyClass`). Moreover, there are events for adding authors to the library and for removing authors from the library (`addAuthor` and `removeAuthor`). The authors to be managed are represented by sub-objects of this authorclass object. In order to distinguish between the author objects, there is a unique (local) name for each of them. In the example the authors are identified by names like `Authors(12)` or `Authors(42)`.

Each author object has attributes: the attributes `Name` and `Status` for the name of the author and for his/her marital status. The attribute `SoldBooks` allows to store the number of books sold in the year which is given as parameter. Of course, author objects can be created and deleted (events `create` and `destroy`). In addition it is possible to change the name of an author (by the event `changeName`) and to store a number of sold books for some year (`storeSoldBooks`).

In the specification of these objects we can additionally restrict the occurrences of events by giving conditions telling in which states of an object certain events may occur and by describing life cycles an object must go through. A typical interaction can be specified, for example, such that the occurrence of an `addAuthor` event in the authorclass object causes the simultaneous occurrence of the `create` event of a sub-object.

We use this example as running example throughout the following sections. $\bowtie$

For specifying objects we developed the specification language TROLL *light* [CGH92, GCH93, HCG94] in which we realized the concepts sketched above. This language is a dialect of TROLL [JSHS91] which is tailored for a more general framework for specifying object communities (cf. [EGS92, EDS93, e.g.]). Another related specification language based on the same fundamental ideas is OBLOG [CSS89, SSG$^+$91].

3. Syntax

In this section we define the syntactical structure of the basic calculus. First, we introduce data signatures and community signatures. Then, we deal with the construction of terms, propositions, and formulas.

Definition 3.1 (Data Signature)

A data signature $\Sigma_D = (\mathrm{S_D}, \Omega_D, \Pi_D)$ consists of

- a set $\mathrm{S_D}$ of data sorts,
- a family of sets of operation symbols $\Omega_D = \langle \Omega_{D,wd} \rangle_{wd \,\in\, (\mathrm{S_D}^* \times \mathrm{S_D})}$, and
- a family of sets of predicate symbols $\Pi_D = \langle \Pi_{D,w} \rangle_{w \,\in\, \mathrm{S_D}^*}$. ◁

Example 3.1

In our running example (see Ex. 2.1) we use the following fragment of a data signature $\Sigma_D = (\mathrm{S_D}, \Omega_D, \Pi_D)$:

$$
\begin{array}{lcl}
\mathrm{S_D} & = & \{\mathtt{string}, \mathtt{nat}, \ldots\} \\
\vdots & & \\
\Omega_{D,\mathtt{string}} & = & \{\mathtt{bottom}_{\mathtt{string}}, \ldots\} \\
\Omega_{D,\mathtt{nat}} & = & \{\mathtt{bottom}_{\mathtt{nat}}, \ldots\} \\
\vdots & & \\
\Omega_{D,\mathtt{string\ string\ string}} & = & \{\mathtt{concat}, \ldots\} \\
\Omega_{D,\mathtt{nat\ nat\ nat}} & = & \{+, -, *, \ldots\} \\
\vdots & & \\
 & & \\
\vdots & & \\
\Pi_{D,\mathtt{string\ string}} & = & \{\mathtt{substr}, \ldots\} \\
\Pi_{D,\mathtt{nat\ nat}} & = & \{>, <, >=, <=, \ldots\} \\
\vdots & &
\end{array}
$$

It should be noted that we assume a constant $\mathtt{bottom}_s$ for each data sort $s \in \mathrm{S_D}$. This constant represents the undefined value. ⋈

Definition 3.2 (Community Signature)

A community signature $\Sigma_C = (\Sigma_D, \mathcal{SB}_C, \textsc{ObG}_C, \textsc{Att}_C, \textsc{Evt}_C)$ consists of

- a data signature $\Sigma_D = (\mathrm{S_D}, \Omega_D, \Pi_D)$,
- a tuple $\mathcal{SB}_C = (\mathrm{S_O}, \mathrm{S_E}, \mathtt{evt})$ with a set $\mathrm{S_O}$ of object sorts, a set $\mathrm{S_E}$ of (local) event sorts, and a (global) event sort $\mathtt{evt}$,
  (in the sequel we use the following abbreviation: $\mathrm{S} := \mathrm{S_D} \cup \mathrm{S_O}$)
- a family of sets of object name generators $\textsc{ObG}_C = \langle \textsc{ObG}_{C,w,o'} \rangle_{w,o' \in (\mathrm{S}^* \to \mathrm{S_O})}$,
- a family of sets of attribute symbols $\textsc{Att}_C = \langle \textsc{Att}_{C,ows} \rangle_{ows \in (\mathrm{S_O} \times \mathrm{S}^* \times \mathrm{S})}$, and
- a family of sets of event symbols $\textsc{Evt}_C = \langle \textsc{Evt}_{C,w,o\_\mathtt{evt}} \rangle_{w,o\_\mathtt{evt} \in (\mathrm{S}^* \to \mathrm{S_E})}$.

The sets $\mathrm{S_O}$, $\mathrm{S_E}$, and $\mathrm{S_D}$ are pairwise disjoint. Furthermore, there is a bijective mapping between $\mathrm{S_O}$ and $\mathrm{S_E}$, such that there is a unique sort $o\_\mathtt{evt} \in \mathrm{S_E}$ for each $o \in \mathrm{S_O}$.

Finally, we define a useful abbreviation: $\hat{\mathrm{S}} := \mathrm{S} \cup \{\mathtt{evt}\} \cup \mathrm{S_E}$. $\triangleleft$

Here, we have defined the notion of community signature such that it is independent of any concrete object specification language. In [Con94a] we present a special version of this calculus tailored for the specification language TROLL *light*.

By means of object name generators we can construct terms for identifying objects. We will later see that a certain condition is necessary for unique identification.

The differentiation between local event sorts ($o\_\mathtt{evt} \in \mathrm{S_E}$) and a global event sort $\mathtt{evt}$ is necessary and reasonable because it permits more flexibility in constructing terms and formulating axioms as we will see below.

Example 3.2

In explanation of this definition we consider the community signature for the running example. In Ex. 2.1 we have considered objects of kind "authorclass" having objects of kind "author" as sub-objects. The corresponding community signature is $\Sigma_{\mathtt{Authorclass}}$ (which we sometimes abbreviate as $\Sigma_{\mathtt{AC}}$). We assume a data signature Σ_D to be given (see Example 3.1).

$\Sigma_{\mathtt{Authorclass}}$ must include two object sorts, i.e., `author` and `authorclass`, and for each of these object sorts a corresponding local event sort (`author_evt` and `authorclass_evt`). Of course, there is also the global event sort `evt`. The attribute symbols and event symbols belonging to these two kinds of objects are listed in Fig. 2. $\bowtie$

From now on we omit the index C in $\textsc{ObG}_C$, $\textsc{Att}_C$, and $\textsc{Evt}_C$ and the index D in Ω_D and Π_D provided that no confusion can arise.

After having introduced data signatures and community signatures we can define terms, propositions, and formulas over community signatures.

$$
\begin{array}{ll}
\mathcal{SB}_{\mathtt{AC}} & = (\ \{\mathtt{authorclass},\ \mathtt{author}\},\\
& \quad\ \{\mathtt{authorclass\_evt},\ \mathtt{author\_evt}\},\\
& \quad\ \mathtt{evt}\ )\\
\textsc{ObG}_{\mathtt{AC},\,\mathtt{authorclass\ nat},\ \mathtt{author}} & = \{\ \mathit{Authors}\ \}\\
\textsc{Att}_{\mathtt{AC},\,\mathtt{authorclass\ nat}} & = \{\ \mathit{NumberOfAuthors}\ \}\\
\textsc{Att}_{\mathtt{AC},\,\mathtt{author\ string}} & = \{\ \mathit{Name},\ \mathit{Status}\ \}\\
\textsc{Att}_{\mathtt{AC},\,\mathtt{author\ nat\ nat}} & = \{\ \mathit{SoldBooks}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{authorclass\_evt}} & = \{\ \mathit{createClass},\ \mathit{destroyClass}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{nat},\ \mathtt{authorclass\_evt}} & = \{\ \mathit{removeAuthor}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{nat\ string\ string},\ \mathtt{authorclass\_evt}} & = \{\ \mathit{addAuthor}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{string\ string},\ \mathtt{author\_evt}} & = \{\ \mathit{create}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{string},\ \mathtt{author\_evt}} & = \{\ \mathit{changeName}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{nat\ nat},\ \mathtt{author\_evt}} & = \{\ \mathit{storeSoldBooks}\ \}\\
\textsc{Evt}_{\mathtt{AC},\,\mathtt{author\_evt}} & = \{\ \mathit{destroy}\ \}
\end{array}
$$

Figure 2: Components of the community signature $\Sigma_{\mathtt{Authorclass}}$.

Definition 3.3 (Term)

Given a community signature $\Sigma_C = (\Sigma_D, \mathcal{SB}, \textsc{ObG}, \textsc{Att}, \textsc{Evt})$ and an $\hat{\mathrm{S}}$-indexed family of sets of variables, $X = \langle X_s \rangle_{s \in \hat{\mathrm{S}}}$. The minimal $\hat{\mathrm{S}}$-indexed family $\textsc{Term}(X) = \langle \textsc{Term}_s(X) \rangle_{s \in \hat{\mathrm{S}}}$ of sets of terms over Σ_C and X is inductively defined by:

- if $x \in X_s$ $(s \in \hat{\mathrm{S}})$, then $x \in \textsc{Term}_s(X)$,
- if $\omega \in \Omega_{d_1 \ldots d_n d}$ and $t_i \in \textsc{Term}_{d_i}(X)$, then $\omega(t_1, \ldots, t_n) \in \textsc{Term}_d(X)$,
- if $\mathrm{og} \in \textsc{ObG}_{\mathrm{s_1 \ldots s_n, o}}$ and $t_i \in \textsc{Term}_{s_i}(X)$, then $\mathrm{og}(\mathrm{t_1}, \ldots, \mathrm{t_n}) \in \textsc{Term}_{\mathrm{o}}(\mathrm{X})$,
- if $\mathrm{e} \in \textsc{Evt}_{\mathrm{s_1 \ldots s_n, o\_\mathtt{evt}}}$ and $t_i \in \textsc{Term}_{s_i}(X)$, then $\mathrm{e}(\mathrm{t_1}, \ldots, \mathrm{t_n}) \in \textsc{Term}_{\mathrm{o\_\mathtt{evt}}}(\mathrm{X})$, and
- if $t_o \in \textsc{Term}_o(X)$ and $t_{o\_\mathtt{evt}} \in \textsc{Term}_{o\_\mathtt{evt}}(X)$, then $t_o.t_{o\_\mathtt{evt}} \in \textsc{Term}_{\mathtt{evt}}(X)$. $\triangleleft$

For building global event terms we have introduced the constructor "." (called the "event constructor"). Thereby, we can put together a term of an object sort $o \in \mathrm{S_O}$ and a term of the corresponding local event sort $o\_\mathtt{evt} \in \mathrm{S_E}$ to form a term of the global event sort `evt`.

Example 3.3

Taking the data signature Σ_D given in Example 3.1 and the community signature $\Sigma_{\mathtt{Authorclass}}$ given in Example 3.2 we can construct the following terms. For this we assume a family X of sets of variables with $X_{\texttt{string}} = \{S, S'\}$, $X_{\texttt{nat}} = \{N\}$, $X_{\texttt{authorclass}} = \{AC\}$, $X_{\texttt{author}} = \{A\}$, and $X_{\texttt{author\_evt}} = \{EV\}$:

$$
\begin{array}{ll}
AC & \in \textsc{Term}_{\texttt{authorclass}}(X) \\
A, \quad Authors(AC, 42) & \in \textsc{Term}_{\texttt{author}}(X) \\
createClass, \quad removeAuthor(N) & \in \textsc{Term}_{\texttt{authorclass\_evt}}(X) \\
create(S, \text{"}\texttt{single}\text{"}), \quad changeName(\text{"}\texttt{E.Kishon}\text{"}) & \in \textsc{Term}_{\texttt{author\_evt}}(X) \\
AC.removeAuthor(17), & \\
Authors(AC, 42).create(S, S'), \quad A.EV & \in \textsc{Term}_{\texttt{evt}}(X)
\end{array}
$$

These terms already show why we have explicitly distinguished between local and global event sorts. Without local event sorts we could not build terms like $A.EV$ and $AC.removeAuthor(17)$. But these terms are necessary to talk about the effects of concrete local events (e.g., $removeAuthor(17)$) to arbitrary objects of the corresponding object sort. Furthermore, a term like $myauthor.EV$ (where $myauthor$ identifies a concrete object) could be useful for expressing that an arbitrary event can happen to the specific object $myauthor$. $\bowtie$

Definition 3.4 (Proposition)

Given a community signature Σ_C and a $\hat{\text{S}}$-indexed family of sets of variables $X = \langle X_s \rangle_{s \in \hat{\text{S}}}$. The set of propositions $\textsc{Prop}(\text{X})$ over Σ_C and X is the minimal set satisfying:

- if $e \in \textsc{Term}_{\texttt{evt}}(X)$, then $enable(e) \in \textsc{Prop}(\text{X})$ and $occur(e) \in \textsc{Prop}(\text{X})$,
- if $\pi \in \Pi_{d_1 \ldots d_n}$ and $t_i \in \textsc{Term}_{d_i}(X)$, then $\pi(t_1, \ldots, t_n) \in \textsc{Prop}(\text{X})$,
- if $\text{a} \in \textsc{Att}_{\text{os}_1 \ldots \text{s}_n \text{s}}$, $t_o \in \textsc{Term}_o(X)$, $t_i \in \textsc{Term}_{s_i}(X)$, and $t \in \textsc{Term}_s(X)$, then $\text{a}(\text{t}_\text{o}, \text{t}_1, \ldots, \text{t}_\text{n}, \text{t}) \in \textsc{Prop}(\text{X})$,
- if $t_1, t_2 \in \textsc{Term}_s(X)$ $(s \in \hat{\text{S}})$, then $(t_1 = t_2) \in \textsc{Prop}(\text{X})$,
- if $P \in \textsc{Prop}(\text{X})$, then $\neg P \in \textsc{Prop}(\text{X})$, and
- if $e \in \textsc{Term}_{\texttt{evt}}(X)$ and $P \in \textsc{Prop}(\text{X})$, then $[e]P \in \textsc{Prop}(\text{X})$. $\lhd$

For events there are state-dependent predicates (*enable* and *occur*) for expressing whether an event may occur (resp. is enabled) and whether an event is really occurring. Further predicate symbols for state-dependent predicates are attribute symbols. This is due to the fact that the value of an attribute may change in the course of time. Propositions of the form $[e]P$ are called positional propositions.

Example 3.4

Using the terms given in Example 3.3 we can build the following propositions as propositions over the community signature $\Sigma_{\texttt{Authorclass}}$ (cf. Example 3.2):

$$
\begin{array}{ll}
enable(AC.removeAuthor(17)), & occur(Authors(AC, 42).create(S, S')), \\
Name(Authors(AC, 42), \text{"}\texttt{E.Kishon}\text{"}), & [A.create(S, S')] \neg\, enable(A.create(S, S')), \\
[AC.removeAuthor(17)] NumberOfAuthors(AC, 6). &
\end{array}
$$

$\bowtie$

Definition 3.5 (Formula)

Given propositions $P_1, \ldots, P_n, Q_1, \ldots, Q_m \in \textsc{Prop}(\mathrm{X})$ over a community signature Σ_C. Then

$$\{P_1, \ldots, P_n\} \rightarrow \{Q_1, \ldots, Q_m\}$$

is a formula over Σ_C. The set of all formulas which can be constructed over a community signature Σ_C and an $\hat{\mathrm{S}}$-indexed family of sets of variables $X = \langle X_s \rangle_{s \in \hat{\mathrm{S}}}$ is denoted by $\textsc{Form}(X)$.

In general we omit braces in formulas and write instead $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$. $\triangleleft$

Example 3.5

Continuing Examples 3.1–3.4 we present some formulas over the community signature $\Sigma_{\texttt{Authorclass}}$:

$$\rightarrow [A.changeName(S)]Name(A, S)$$
$$NumberOfAuthors(AC, 7) \rightarrow [AC.removeAuthor(17)]NumberOfAuthors(AC, 6)$$

$\bowtie$

Thus, we have provided the syntactical basis of the calculus. The next section presents a model-theoretic semantics.

4. Model-Theoretic Semantics

The model-theoretic semantics of the basic calculus is founded on a variation of Kripke-structures. Before we define these interpretation structures we introduce state-independent interpretation of sorts and operations. Then we proceed with term evaluation. Afterwards we can give the definition of interpretation structures. After that we continue by dealing with satisfaction of propositions and truth of formulas. Finally, we introduce the notions “semantic entailment” and “semantic closure”.

Definition 4.1 (State-independent Interpretation)

Interpretations of sorts $s \in \hat{\mathrm{S}}$, of operation symbols $\omega \in \Omega_{d_1 \ldots d_n, d}$, of predicate symbols $p \in \Pi_{d_1 \ldots d_n}$, of object name generators $\mathrm{og} \in \mathrm{O{\scriptsize BG}}_{\mathrm{s_1 \ldots s_n, o}}$, and of event symbols $\mathrm{e} \in \mathrm{E{\scriptsize VT}}_{\mathrm{s_1 \ldots s_n}, \mathrm{o\_evt}}$ are denoted by $I(s)$, $I(\omega)$, $I(p)$, $I(\mathrm{og})$, and $I(\mathrm{e})$, respectively:

- $I(s)$ denotes a carrier set for $s \in \hat{\mathrm{S}}$,
- $I(\omega) : I(d_1) \times \ldots \times I(d_n) \to I(d)$ for $\omega \in \Omega_{d_1 \ldots d_n, d}$,
- $I(p) \subseteq I(d_1) \times \ldots \times I(d_n)$ for $p \in \Pi_{d_1 \ldots d_n}$,
- $I(\mathrm{og}) : I(s_1) \times \ldots \times I(s_n) \to I(o)$ for $\mathrm{og} \in \mathrm{O{\scriptsize BG}}_{\mathrm{s_1 \ldots s_n, o}}$, and
- $I(\mathrm{e}) : I(s_1) \times \ldots \times I(s_n) \to I(o\_\mathtt{evt})$ for $\mathrm{e} \in \mathrm{E{\scriptsize VT}}_{\mathrm{s_1 \ldots s_n}, \mathrm{o\_evt}}$.

For the data signature we assume an algebra $A(\Sigma_D) = (A(\mathrm{S_D}), A(\Omega_D), A(\Pi_D)) \in \mathrm{ALG}_{\Sigma_D}$ as given. Thereby, $I(s)$ for each $s \in \mathrm{S_D}$, $I(\omega)$ for each $\omega \in \Omega_D$, and $I(p)$ for each $p \in \Pi_D$ is fixed.

Object sorts $o \in \mathrm{S_O}$ and local event sorts $o\_\mathtt{evt} \in \mathrm{S_E}$ are interpreted by the term algebras $\mathrm{T{\scriptsize ERM}}_o$ and $\mathrm{T{\scriptsize ERM}}_{o\_\mathtt{evt}}$, resp. — modulo equivalence of terms induced by $A(\Sigma_D)$.

The carrier set $I(\mathtt{evt})$ can be constructed by the disjoint union of all Cartesian products which are obtained from the carrier sets of object sorts and their corresponding local event sorts:

$$I(\mathtt{evt}) = \biguplus_{o \in \mathrm{S_O}} (I(o) \times I(o\_\mathtt{evt}))$$

The event constructor $\_.\_ : \biguplus_{o \in \mathrm{S_O}} (o \times o\_\mathtt{evt}) \to \mathtt{evt}$ is injective and surjective. $\lhd$

In the following we use the notation $\underline{s_i}$ to denote an element of $I(s_i)$.

Definition 4.2 (Assignment)

An assignment $\mathcal{A}$ for a given family X of sets of variables assigns a value $\mathcal{A}(x) \in I(s)$ to each variable $x \in X_s$. $\lhd$

Such an assignment is necessary for interpreting terms containing variables.

Definition 4.3 (Term Evaluation)

The value $[\![t]\!]^{\mathcal{A}}$ of a term t by employing an assignment $\mathcal{A}$ is inductively defined as follows:

- for $x \in X_s$ we have $[\![x]\!]^{\mathcal{A}} = \mathcal{A}(x)$,
- for $\omega \in \Omega_{d_1 \ldots d_n, d}$ and $t_i \in \text{T}\small{\text{ERM}}_{d_i}(X)$ we have $[\![\omega(t_1, \ldots, t_n)]\!]^{\mathcal{A}} = I(\omega)([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}})$,
- for $\mathrm{og} \in \mathrm{O\small{B}G}_{\mathrm{s_1 \ldots s_n, o}}$ and $t_i \in \text{T}\small{\text{ERM}}_{s_i}(X)$ we have $[\![\mathrm{og}(\mathrm{t_1}, \ldots, \mathrm{t_n})]\!]^{\mathcal{A}} = \mathrm{I}(\mathrm{og})([\![\mathrm{t_1}]\!]^{\mathcal{A}}, \ldots, [\![\mathrm{t_n}]\!]^{\mathcal{A}})$,
- for $\mathrm{e} \in \mathrm{E\small{VT}}_{\mathrm{s_1 \ldots s_n, o\_\texttt{evt}}}$ and $t_i \in \text{T}\small{\text{ERM}}_{s_i}(X)$ we have $[\![\mathrm{e}(\mathrm{t_1}, \ldots, \mathrm{t_n})]\!]^{\mathcal{A}} = \mathrm{I}(\mathrm{e})([\![\mathrm{t_1}]\!]^{\mathcal{A}}, \ldots, [\![\mathrm{t_n}]\!]^{\mathcal{A}})$, and
- for $t_o \in \text{T}\small{\text{ERM}}_o(X)$ and $t_{o\_\texttt{evt}} \in \text{T}\small{\text{ERM}}_{o\_\texttt{evt}}(X)$ we have $[\![t_o.t_{o\_\texttt{evt}}]\!]^{\mathcal{A}} = I(.)([\![t_o]\!]^{\mathcal{A}}, [\![t_{o\_\texttt{evt}}]\!]^{\mathcal{A}})$. $\lhd$

The term evaluation is defined in such a way that all terms are interpreted independent of time and, hence, independent of the state of the object community. This is in contrast to some other approaches like [FS90, FM91a, FM91b].

Definition 4.4 (State Proposition)

The set of state propositions SProp is defined by:

$$
\begin{aligned}
\mathrm{SP\small{ROP}} :=\ & \\
& \{\mathrm{a}(\underline{\mathrm{o}}, \underline{\mathrm{s_1}}, \ldots, \underline{\mathrm{s_n}}, \underline{\mathrm{s}}) \mid \mathrm{a} \in \mathrm{A\small{TT}}_{\mathrm{os_1 \ldots s_n s}}, \underline{\mathrm{o}} \in \mathrm{I}(\mathrm{o}), \underline{\mathrm{s_1}} \in \mathrm{I}(\mathrm{s_1}), \ldots, \underline{\mathrm{s_n}} \in \mathrm{I}(\mathrm{s_n}), \underline{\mathrm{s}} \in \mathrm{I}(\mathrm{s})\} \\
\cup\ & \{enable(\underline{o.e}) \mid \underline{o.e} \in I(\texttt{evt})\} \\
\cup\ & \{occur(\underline{o.e}) \mid \underline{o.e} \in I(\texttt{evt})\}
\end{aligned}
$$

$\lhd$

Here, a remark concerning the difference between Prop and SProp is necessary. The set Prop provides all propositions which may be constructed *syntactically* (i.e., these propositions are uninterpreted and may contain variables). In contrast the set of state propositions SProp includes all *interpreted* propositions which are predicate expressions built with state-dependent predicate symbols and interpreted value. In the following definition we use the set SProp as domain for assigning each state of an object community a subset of SProp as interpretation.

Definition 4.5 (Interpretation Structure)

Given a community signature Σ_C and for Σ_C a state-independent interpretation, an interpretation structure $\mathcal{IS} = (\mathcal{W}, \mathcal{N}, \rho)$ for Σ_C consists of

- an infinite, totally ordered set $\mathcal{W} = \{w_0, w_1, \ldots\}$ of (community) states,
- a successor function $\mathcal{N} : \mathcal{W} \to \mathcal{W}$, which fixes the total order on $\mathcal{W}$ (i.e., $\mathcal{N}(w_i) = w_{i+1}$ for all $i \in I\!N_0$),
- a function $\rho : \mathcal{W} \to 2^{\text{SProp}}$ assigning a set of state propositions to each community state.

The function ρ has to fulfill the following condition:

$$occur(\underline{o}.\underline{e}) \in \rho(w) \;\wedge\; occur(\underline{o}.\underline{e'}) \in \rho(w) \quad \Longrightarrow \quad \underline{e} = \underline{e'}$$

As notational simplification we define a function $\mathcal{O}cc : \mathcal{W} \to 2^{I(\texttt{evt})}$ with $\mathcal{O}cc(w) := \{\ \underline{o.e} \mid occur(\underline{o.e}) \in \rho(w)\ \}$ for each $w \in \mathcal{W}$. $\lhd$

By this definition we restrict our considerations to linear interpretation structures because we require a total order on $\mathcal{W}$. This is a reasonable restriction because we strive for a basic calculus being as simple as possible and, therefore, we do not include operators requiring a branching time interpretation. Although we require $\mathcal{W}$ to be an infinite set, finite life cycles are not excluded for objects. This is due to the fact that there may be states in which no event occurs. We also allow "trivial" interpretation structures without any event occurrence. The condition imposed on ρ reflects our assumption about objects that there must not happen more than one event at a moment (resp., in a state).

For illustrating interpretation structures we use diagrams in which the states are connected by edges representing the successor relationship (as known, e.g., from state transition diagrams). We label each edge starting at state w_i by the set of event occurrences $\mathcal{O}cc(w_i)$:

$$\mathcal{IS}: \qquad w_0 \xrightarrow{\mathcal{O}cc(w_0)} w_1 \xrightarrow{\mathcal{O}cc(w_1)} w_2 \xrightarrow{\mathcal{O}cc(w_2)} w_3 \xrightarrow{\mathcal{O}cc(w_3)} w_4 \xrightarrow{\mathcal{O}cc(w_4)} w_5 \xrightarrow{\mathcal{O}cc(w_5)} \cdots$$

Example 4.1

As an example for an interpretation structure we consider an intrepretation structure $\mathcal{IS}_{\texttt{Authorclass}} = (\mathcal{W}, \mathcal{N}, \rho)$ for the community signature $\Sigma_{\texttt{Authorclass}}$ given in Example 3.2. $\mathcal{N}$ is the successor function on $\mathcal{W}$ as required in Def. 4.5. In Fig. 3 we partially describe $\rho : \mathcal{W} \to 2^{\text{SProp}}$ assuming an arbitrary but fixed element $\underline{o} \in I(\texttt{authorclass})$. Furthermore, we use the following abbreviations:

$$\begin{aligned} \underline{a12} &:= I(Authors)(\underline{o}, I(12)) \\ \underline{a13} &:= I(Authors)(\underline{o}, I(13)) \end{aligned}$$

$\bowtie$

$$
\begin{array}{lll}
\rho(w_0) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(\bot)), \\
 & & \quad Name(\underline{a12}, I(\bot)),\ Status(\underline{a12}, I(\bot)), \\
 & & \quad occur(\underline{o}.I(createClass)) \qquad \} \\
\rho(w_1) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(0))), \\
 & & \quad Name(\underline{a12}, I(\bot)),\ Status(\underline{a12}, I(\bot)), \\
 & & \quad occur(\underline{o}.I(addAuthor)(I(12), I(\texttt{"F.Kafka"}), I(\texttt{"married"}))), \\
 & & \quad occur(\underline{a12}.I(create)(I(\texttt{"F.Kafka"}), I(\texttt{"married"}))) \qquad \} \\
\rho(w_2) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(1))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"F.Kafka"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\bot)),\ Status(\underline{a13}, I(\bot)), \\
 & & \quad occur(\underline{o}.I(addAuthor)(I(13), I(\texttt{"J.Joyce"}), I(\texttt{"single"}))), \\
 & & \quad occur(\underline{a13}.I(create)(I(\texttt{"J.Joyce"}), I(\texttt{"single"}))) \qquad \} \\
\rho(w_3) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"F.Kafka"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\texttt{"J.Joyce"})),\ Status(\underline{a13}, I(\texttt{"single"})), \\
 & & \quad SoldBooks(\underline{12}, I(1993), I(\bot)), \\
 & & \quad occur(\underline{a12}.I(storeSoldBooks)(I(1993), I(42))) \qquad \} \\
\rho(w_4) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"F.Kafka"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\texttt{"J.Joyce"})),\ Status(\underline{a13}, I(\texttt{"single"})), \\
 & & \quad SoldBooks(\underline{a12}, I(1993), I(42)), \\
 & & \quad occur(\underline{a13}.I(storeSoldBooks)(I(1993), I(555))), \\
 & & \quad occur(\underline{a12}.I(changeName)(I(\texttt{"E.Kishon"}))) \qquad \} \\
\rho(w_5) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(2))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"E.Kishon"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\texttt{"J.Joyce"})),\ Status(\underline{a13}, I(\texttt{"single"})), \\
 & & \quad occur(\underline{a13}.I(storeSoldBooks)(I(1993), I(555))), \\
 & & \quad occur(\underline{a12}.I(changeName)(I(\texttt{"E.Kishon"}))) \qquad \} \\
\vdots & & \vdots \\
\rho(w_{1342}) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(123))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"E.Kishon"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\texttt{"J.Joyce"})),\ Status(\underline{a13}, I(\texttt{"single"})), \\
 & & \quad occur(\underline{0}.I(removeAuthor)(I(13))), \\
 & & \quad occur(\underline{a13}.I(destroy)) \qquad \} \\
\rho(w_{1343}) & \supseteq & \{\ NumberOfAuthors(\underline{o}, I(122))), \\
 & & \quad Name(\underline{a12}, I(\texttt{"E.Kishon"})),\ Status(\underline{a12}, I(\texttt{"married"})), \\
 & & \quad Name(\underline{a13}, I(\bot)),\ Status(\underline{a13}, I(\bot)), \\
 & & \quad occur(\underline{a13}.I(storeSoldBooks)(I(1994), I(666))) \qquad \} \\
\vdots & & \vdots
\end{array}
$$

Figure 3: Fragment of a concrete interpretation structure $\mathcal{IS}_{\texttt{Authorclass}}$.

After this introduction of interpretation structures we can define satisfaction of propositions.

Definition 4.6 (Satisfaction of Propositions)

The satisfaction of propositions $P \in \textsc{Prop}(\mathrm{X})$ over a community signature Σ_C by an interpretation structure $\mathcal{IS}$ for Σ_C and an assignment $\mathcal{A}$ in a state $w \in \mathcal{W}$ (written $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models P$) is inductively defined as follows:

- for $p \in \{occur, enable\} \cup \bigcup_{ows \in \mathrm{S_O}\times \mathrm{S}^* \times \mathrm{S}} \textsc{Att}_{o\, s_1 \ldots s_n s}$:
  $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models p(t_1, \ldots, t_n)$ iff $p([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}}) \in \rho(w)$,
- for $p \in \Pi_{d_1 \ldots d_n}$: $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models p(t_1, \ldots, t_n)$ iff $([\![t_1]\!]^{\mathcal{A}}, \ldots, [\![t_n]\!]^{\mathcal{A}}) \subseteq I(p)$,
- $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models \neg P$ iff not $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models P$,
- $\langle \mathcal{IS}, \mathcal{A}, w\rangle \models [o.e]P$ iff
  - $[\![o]\!]^{\mathcal{A}}.[\![e]\!]^{\mathcal{A}} \in \mathcal{O}cc(w) \wedge \langle \mathcal{IS}, \mathcal{A}, \mathcal{N}(w)\rangle \models P$, or
  - $[\![o]\!]^{\mathcal{A}}.\underline{e'} \in \mathcal{O}cc(w) \wedge \underline{e'} \neq [\![e]\!]^{\mathcal{A}}$, or
  - $(\neg\exists \underline{o'}.\underline{e'} \in \mathcal{O}cc(w) : \underline{o'} = [\![o]\!]^{\mathcal{A}}) \wedge \langle \mathcal{IS}, \mathcal{A}, \mathcal{N}(w)\rangle \models [o.e]P$. $\triangleleft$

The last item of the definition deals with the satisfaction of propositions with positional operator. Given an assignment $\mathcal{A}$ a proposition $[o.e]P$ is satisfied in a state $w \in \mathcal{W}$ iff we can conclude from the fact, that the event $[\![e]\!]^{\mathcal{A}}$ occurs next in the object denoted by $[\![o]\!]^{\mathcal{A}}$, that P holds in the subsequent state (after the occurrence of $[\![e]\!]^{\mathcal{A}}$). For clarification we consider the three possible cases in detail:

- The first case is that the event $[\![e]\!]^{\mathcal{A}}$ occurs directly in state w in the object denoted by $[\![o]\!]^{\mathcal{A}}$:

  $$\mathcal{IS}: \qquad \cdots \xrightarrow{\ldots} w \xrightarrow{\ldots, [\![o]\!]^{\mathcal{A}}.[\![e]\!]^{\mathcal{A}}, \ldots} w' \xrightarrow{\ldots}$$

  Then, for $[o.e]P$ to be satisfied in w, P must be satisfied in w'.

- The second case is given by the situation that in state w an event $\underline{e'}$ different from $[\![e]\!]^{\mathcal{A}}$ occurs in object $[\![o]\!]^{\mathcal{A}}$:

  $$\mathcal{IS}: \qquad \cdots \xrightarrow{\ldots} w \xrightarrow{\ldots, [\![o]\!]^{\mathcal{A}}.\underline{e'}, \ldots} w' \xrightarrow{\ldots} \qquad \underline{e'} \neq [\![e]\!]^{\mathcal{A}}$$

  In that case we say that $[o.e]P$ holds in w without any further condition.[1]

[1] The reason for this is that we consider $[o.e]P$ as an implicite implication: if $o.e$ occurs then P holds afterwards.

- The remaining case is that in state w no event is occurring in object $[\![o]\!]^{\mathcal{A}}$ at all:

$$\mathcal{IS}: \qquad \cdots \xrightarrow{\cdots} w \xrightarrow{\mathcal{O}cc(w)} w' \xrightarrow{\cdots} \qquad \neg\exists \underline{e} : [\![o]\!]^{\mathcal{A}}.\underline{e} \in \mathcal{O}cc(w)$$

  Here, we have an idle-transition for object $[\![o]\!]^{\mathcal{A}}$ from w to w'. This may be due to the occurrence of events in other objects which do not effect the object $[\![o]\!]^{\mathcal{A}}$. From the local point of view of the object $[\![o]\!]^{\mathcal{A}}$ the state w' cannot be distinguished from w. Therefore, we say that $[o.e]P$ is satisfied in state w if and only if $[o.e]P$ is also satisfied in w'.

Because there may happen at most one event in an object at a moment, there is no further case and these three cases are disjoint.

Example 4.2

For the interpretation structure $\mathcal{IS}_{\texttt{Authorclass}}$ given in Example 4.1 we regard some propositions and check whether they are satisfied in certain community states under the assumption of some assignment. As assignments we use $\mathcal{A}_1$ and $\mathcal{A}_2$ where $\mathcal{A}_1(AC) = \underline{o}$, $\mathcal{A}_1(A) = \underline{a12}$ $(= I(\texttt{Authors}(\underline{o}, I(12)))$), $\mathcal{A}_1(S) = I(\text{"}\texttt{F.Kafka}\text{"})$ and $\mathcal{A}_2(AC) = \underline{o}$, $\mathcal{A}_2(A) = \underline{a12}$ $(= I(\texttt{Authors}(\underline{o}, I(12)))$), $\mathcal{A}_2(S) = I(\text{"}\texttt{E.Kishon}\text{"})$:

$$\begin{array}{lcl}
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_2 \rangle & \models & Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_2 \rangle & \models & \neg Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle & \models & \neg SoldBooks(A, 17) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_1 \rangle & \models & occur(A.create(\text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_4 \rangle & \models & [A.changeName(S)]Name(A, S) \\
\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1 \rangle & \models & \\
\multicolumn{3}{l}{\quad \neg[AC.addAuthor(12, \text{"}\texttt{F.Kafka}\text{"}, \text{"}\texttt{married}\text{"})]NumberOfAuthors(AC, 0)}
\end{array}$$

$\bowtie$

Before we proceed to define truth of formulas we have to point out that the propositions $[o.e]\neg P$ and $\neg [o.e]P$ are in general not equal with regard to their satisfaction. This is due to the implicit implication mentioned in the last footnote and can easily be seen by comparing $A \to \neg B$ and $\neg(A \to B)$ which are also not equivalent.

Definition 4.7 (Truth of Formulas)

A formula $\phi \equiv P_1, \ldots, P_n \to Q_1, \ldots, Q_m$ is called true in an interpretation structure $\mathcal{IS}$ for Σ_C (written $\mathcal{IS} \models \phi$) if and only if the following condition holds:

$$\begin{array}{rl}
\multicolumn{2}{l}{\text{for each } w \in \mathcal{W} \text{ and each assignment } \mathcal{A}:} \\
 & \langle \mathcal{IS}, \mathcal{A}, w \rangle \models P_1 \text{ and } \ldots \text{ and } \langle \mathcal{IS}, \mathcal{A}, w \rangle \models P_n \\
\text{implies} & \langle \mathcal{IS}, \mathcal{A}, w \rangle \models Q_1 \text{ or } \ldots \text{ or } \langle \mathcal{IS}, \mathcal{A}, w \rangle \models Q_m
\end{array}$$

$\triangleleft$

Example 4.3

For checking whether a formula is true we have to inspect all states of an interpretation structure. Because we have presented the interpretation structure $\mathcal{IS}_{\texttt{Authorclass}}$ incompletely in Example 4.1 we can only show for some formulas that are not true in $\mathcal{IS}_{\texttt{Authorclass}}$. To verify this it is sufficient to find a community state and an assignment for which the considered formula does not hold. As assignments we reuse the assignments $\mathcal{A}_1$ and $\mathcal{A}_2$ of Example 4.2:

- $Name(A, \perp) \;\rightarrow\; Status(A, "\texttt{single}")$

  This formula cannot be true in $\Sigma_{\texttt{Authorclass}}$ because $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_0\rangle \models Name(A, \perp)$ holds, but $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_1, w_0\rangle \models Status(A, "\texttt{single}")$ does not hold.

- $NumberOfAuthors(AC, 0)$
  $\quad \rightarrow\; [AC.addAuthor(12, "\texttt{F.Kafka}", "\texttt{married}")]NumberOfAuthors(AC, 2)$

  Here, $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1\rangle \models NumberOfAuthors(AC, 0)$ holds. Moreover, the event $AC.addAuthor(12, "\texttt{F.Kafka}", "\texttt{married}")$ occurs in state w_1.
  However, $NumberOfAuthors(AC, 2)$ is not satisfied in the following state w_2, i.e., we have the situation that $\langle \mathcal{IS}_{\texttt{Authorclass}}, \mathcal{A}_2, w_1\rangle \models [AC.addAuthor(12, "\texttt{F.Kafka}", "\texttt{married}")]NumberOfAuthors(AC, 2)$ does not hold.

Of course, there are also formulas which are true in $\Sigma_{\texttt{Authorclass}}$, provided that the fragment of this interpretation structure is completed adequately. Such formulas are for instance,

$$\rightarrow\; [A.create(S, S')]Name(A,\ S)$$

$$occur(AC.addAuthor(Nr, S, S')) \;\rightarrow\; occur(Authors(AC, Nr).create(S, S'))$$

The first one describes that after the occurrence of a `create` event for an author object the attribute `Name` of this author object has exactly the value given as parameter to the `create` event. The second one deals with interaction between an authorclass object and one of its sub-objects. An occurrence of an `addAuthor` event in an authorclass object always causes an occurrence of a `create` event in the corresponding sub-object. The parameter of the `create` event are given by the parameter of the `addAuthor` event. $\bowtie$

Definition 4.8 (Semantic Entailment)

Given a community signature Σ_C and a set Φ of formulas over Σ_C. That a formula ϕ is semantically entailed by Φ (written $\Phi \models_{\Sigma_C} \phi$) is defined by:

$$\Phi \models_{\Sigma_C} \phi \qquad :\Longleftrightarrow \qquad \left(\begin{array}{ll} \multicolumn{2}{l}{\text{for each interpretation structure } \mathcal{IS} \text{ for } \Sigma_C\text{:}} \\ & \mathcal{IS} \models \phi' \text{ for each } \phi' \in \Phi \\ \text{implies} & \mathcal{IS} \models \phi \end{array}\right)$$

$\triangleleft$

Definition 4.9 (Semantic Closure)

Given a community signature Σ_C and a set Φ of formulas over Σ_C. The semantic closure $\Phi^{\models_{\Sigma_C}}$ of Φ is the set of all formulas over Σ_C which are semantically entailed by Φ:

$$\phi \in \Phi^{\models_{\Sigma_C}} \qquad :\Longleftrightarrow \qquad \Phi \models_{\Sigma_C} \phi$$

$\lhd$

Finally, we can define the notion "model".

Definition 4.10 (Model)

An interpretation structure $\mathcal{IS}$ over a community signature Σ_C is a model for a set Φ of formulas over Σ_C, if and only if all formulas in Φ are true in $\mathcal{IS}$. $\lhd$

Because the logical description of an object community is given as a set of formulas we call a model for this set of formulas also a model of the considered object community.

Assuming a set of formulas being a logical description of our running example, we could complete the fragment of the interpretation structure $\Sigma_{\texttt{Authorclass}}$ given in Example 4.1 in order to get a model for this object community.

In proving object properties by means of a theorem proving system, however, concrete models are not to the fore. Instead the syntactical derivation of true formulas is more important. Therefore, the next section deals with an axiomatic semantics for our calculus.

5. Axiomatic Semantics

In the sequel we define the syntactical derivation of formulas. First we introduce the notion of substitution. Then we present basic inference rules for our calculus. Afterwards we define syntactical derivation (or syntactical entailment) as application of inference rules and introduce the notion of "generated theory". We conclude by discussing soundness and completeness issues of the basic calculus.

Definition 5.1 (Substitution)

A substitution s is a function which substitutes a finite number of variables by terms. These terms must not contain any variable which is replaced by this substitution. A concrete substitution is written as $\{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$. The application of a substitution s on a proposition P is denoted by $s(P)$. $\triangleleft$

Substitutions are used in context with inference rules. Applying a substitution to propositions in a formula results in a new formula which is an instantiation of the former one.

Definition 5.2 (Inference Rules)

The basic calculus includes the following inference rules:

$$\frac{}{P \rightarrow P}\ \textbf{(axiom)} \qquad \frac{R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, \neg R}\ \textbf{(negR)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R'}\ \textbf{(ext)} \qquad \frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R}{\neg R, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}\ \textbf{(negL)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, R \qquad R, P'_1, \ldots, P'_k \rightarrow Q'_1, \ldots, Q'_l}{P'_1, \ldots, P'_k, P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m, Q'_1, \ldots, Q'_l}\ \textbf{(cut)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{s(P_1), \ldots, s(P_n) \rightarrow s(Q_1), \ldots, s(Q_m)}\ \textbf{(sub)} \qquad \frac{occur(o_1.e_1) \rightarrow occur(o_2.e_2)}{[o_2.e_2]P \rightarrow [o_1.e_1]P}\ \textbf{(occ)}$$

$$\frac{P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m}{[o.e]P_1, \ldots, [o.e]P_n \rightarrow [o.e]Q_1, \ldots, [o.e]Q_m}\ \textbf{(nex)}$$

$\triangleleft$

Disregarding the last three rules we have a typical proof system for a propositional calculus with Gentzen-clauses as formulas (cf. for instance [Gen35, Gal93]). The rule **(sub)** is necessary for renaming and replacing variables in order to get more concrete instances of a formula. Occurrence dependencies between events can be transferred into positional operators by rule **(occ)**. The last rule makes it possible to introduce positional operators. Such a rule is also used in [Fia88, FM90, FM91b, u.a.] for similar calculi.

Essentially, the presented inference rules are dealing with the propositional part of our calculus. Here, we have refrained from adding further rules, e.g., for dealing with equality. This is due to the fact that our main goal is the development of a basic calculus which can later be extended.

Before we proceed to the definition of syntactical entailment we have to mention that properties of objects which are inherent in our object model must be given as axioms (or axiom schemes). Here, we only want to give a few examples (with $\mathrm{a} \in \mathrm{ATT}_{\mathrm{o}\,\mathrm{s}_1 \ldots \mathrm{s}_n, \mathrm{s}}$):

$$
\begin{aligned}
&occur(O.E) \rightarrow enable(O.E) \\
&occur(O.E_1), occur(O.E_2) \rightarrow O.E_1 = O.E_2 \\
&\mathrm{a}(\mathrm{O}, \mathrm{P}_1, \ldots, \mathrm{P}_\mathrm{n}, \mathrm{V}_1),\ \mathrm{a}(\mathrm{O}, \mathrm{P}_1, \ldots, \mathrm{P}_\mathrm{n}, \mathrm{V}_2) \rightarrow \mathrm{V}_1 = \mathrm{V}_2
\end{aligned}
$$

The first axiom describes that an occurring event must be enabled. The second one states that there cannot be two (or more) different events occurring in the same object at the same moment. The last one is an axiom scheme and requires for each attribute that the attribute value is unique for each instant of time.

In the next definition we use the terms “formula schema” and “instantiation of a formula schema”. We call $\psi = P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ a formula schema if $P_1, \ldots, P_n, Q_1, \ldots, Q_m$ are variables standing for arbitrary propositions (where n and m may be unfixed). An instantiation of such a formula schema is a formula of the basic calculus such that each occurrence of a variable in the formula schema is replaced by the same proposition.

Definition 5.3 (Syntactical Entailment)

A formula ϕ can be (syntactically) entailed from a set Φ of formulas (written $\Phi \vdash_{\Sigma_C} \phi$) iff

- $\phi \in \Phi$ (the trivial case), or
- $\Phi \vdash_{\Sigma_C} \phi_1, \ \ldots, \ \Phi \vdash_{\Sigma_C} \phi_n$, and there is an inference rule

  $$\frac{\psi_1 \quad \ldots \quad \psi_n}{\psi} \quad ,$$

  where $\psi_1, \ldots, \psi_n, \psi$ are formula schemas and

  - $\phi_1, \ldots, \phi_n$ are instantiations of the formula schemas $\psi_1, \ldots, \psi_n$ and
  - ϕ is an instantiation of ψ,

where all instantiations replace the same variable by the same proposition. $\triangleleft$

Please note that we distinguish between "semantic entailment" (Def. 4.8) and "syntactical entailment". In order to avoid confusion we use the adjectives "semantic" and "syntactical" whenever necessary.

Definition 5.4 (Generated Theory)

The theory generated by a set Φ of formulas over a community signature Σ_C is denoted by $\Phi^{\vdash_{\Sigma_C}}$ and is obtained by:

$$\phi \in \Phi^{\vdash_{\Sigma_C}} \qquad :\Longleftrightarrow \qquad \Phi \vdash_{\Sigma_C} \phi$$

$\triangleleft$

Thus, given a logical description Φ_C of an object community with signature Σ_C, the generated theory $\Phi^{\vdash_{\Sigma_C}}$ is the set of all formulas over Σ_C which can be syntactically entailed (by means of the inference rules given in Def. 5.2).

The following theorem states that the calculus is sound, i.e., that we cannot syntactically entail formulas which are not semantically entailed.

Theorem 5.1 (Soundness)

The basic calculus is sound, i.e., $\Phi^{\vdash_{\Sigma_C}} \subseteq \Phi^{\models_{\Sigma_C}}$, resp.:

$$\Phi \vdash_{\Sigma_C} \phi \quad \Longrightarrow \quad \Phi \models_{\Sigma_C} \phi \qquad \bigcirc$$

Sketch of proof:

The main point in proving this theorem is the soundness of the rules **(nex)** and **(occ)**. For the other rules we can easily adopt soundness proofs of other sequent calculi. This is due to the fact that these rules do not involve positional operators and, therefore, we can reduce soundness of these rules to the soundness of the corresponding rules for first order predicate calculi.

For the soundness of the rule **(nex)** we may argue that the truth of $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$ is sufficient for the truth of $[E]P_1, \ldots, [E]P_n \rightarrow [E]Q_1, \ldots, [E]Q_m$. This results from the fact that the introduction of the positional operator can be seen as a restriction on the set of states which have to satisfy $P_1, \ldots, P_n \rightarrow Q_1, \ldots, Q_m$.

In [Con94a] we give a more detailed and formal proof for the soundness of these inference rules accept for the **(occ)** rule. The soundness of **(occ)** can easily be seen by considering the following cases for two events $o_1.e_1$ and $o_2.e_2$ for which we assume that $occur(o_1.e_1) \rightarrow occur(o_2.e_2)$ is true:

Neither $o_1.e_1$ nor $o_2.e_2$ occurs (in a certain state):

Then $[o_1.e_1]P$ and $[o_2.e_2]P$ are satisfied in that state for each P. Thus, $[o_2.e_2]P \rightarrow [o_1.e_1]P$ holds.

o_2, e_2 occurs, but $o_1.e_1$ does not:
Here, $[o_1.e_1]P$ is satisfied and thereby $[o_2.e_2]P \rightarrow [o_1.e_1]P$ holds, too.

$o_1.e_1$ occurs:
It follows that $o_2.e_2$ also occurs (due to the premise). Thus, if P holds after the occurrence of $o_2.e_2$ then, of course, it also holds after the occurrence of $o_1.e_1$.

In fact, this consideration is a little bit simplified. The formal proof has to deal with more details.

After having stated the soundness of the proof system with regard to the underlying model theory, the question of completeness arises. We do not present a theorem for completeness due to two main reasons:

- On the one hand we have not aimed at completeness in constructing the proof system. Instead we have only intended to provide a simple and clear collection of sound inference rules as a basis for later extensions. Therefore, the given proof system only copes with the logical kernel common to all intended extensions and modifications of our calculus.
- On the other hand, regarding other calculi for which the question of completeness has already been answered yields a meaningful assessment for our calculus. It is well-known that there are complete calculi for first-order predicate logic. By means of skolemization first-order predicate logic can obviously be embedded into our calculus. Thereby, our calculus is at least as expressive as first-order predicate calculus. If our calculus is more expressive, then it cannot be complete at all (assuming a strong notion of completeness). For a large class of temporal logics Abadi has already shown that these logics cannot be strongly complete due to the fact that they are at least as expressive as Peano arithmetics which is known to be incomplete [Aba89].

In consequence, we have a justified assumption that our calculus cannot be completed w.r.t. the strong notion of completeness.

We conclude this section by giving an example of a simple proof.

Example 5.1

Continuing the running example the property we want to prove is as follows: after the occurrence of an `addAuthor` event in the authorclass object the name of the new author object is exactly the name given as parameter to the `addAuthor` event:

$$\rightarrow [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\ S)$$

The proof uses two formulas (φ_1, φ_2) which were already given in Ex. 4.3:

$$\dfrac{\dfrac{\varphi_1}{\varphi_3}\,\textbf{(sub)} \qquad \dfrac{\varphi_2}{\varphi_4}\,\textbf{(occ)}}{\varphi_5}\,\textbf{(cut)}$$

where

$$\varphi_1: \;\; \rightarrow \;\; [A.create(S, S')]Name(A,\, S)$$

$$\varphi_2: \;\; occur(AC.addAuthor(Nr, S, S')) \;\rightarrow\; occur(Authors(AC, Nr).create(S, S'))$$

$$\varphi_3: \;\; \rightarrow \;\; [Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\, S)$$

$$\begin{aligned}\varphi_4: \;\; & [Authors(AC, Nr).create(S, S')]Name(Authors(AC, Nr),\, S) \\ & \qquad \rightarrow \;\; [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\, S)\end{aligned}$$

$$\varphi_5: \;\; \rightarrow \;\; [AC.addAuthor(Nr, S, S')]Name(Authors(AC, Nr),\, S)$$

Starting with φ_1 the substitution of A by $Authors(AC, Nr)$ yields φ_3. Formula φ_4 is obtained by applying the rule **(occ)** to φ_2. Finally, we can combine φ_3 and φ_4 using the rule **(cut)**. The result is φ_5, i.e., the formula to be proved. $\bowtie$

6. On Models and Proofs for Composed Specifications

In the sequel we discuss the relationship between models of a composed specification and models of its components. Furthermore, we consider the corresponding theories and, thus, proofs. Beforehand, we have an abstract look at object hierarchies.

First, we regard a universal scheme for object hierarchies. Therefore, we need a super-object $\hat{o}$ having sub-objects $o_1, \ldots, o_n$ as depicted in Fig. 4.

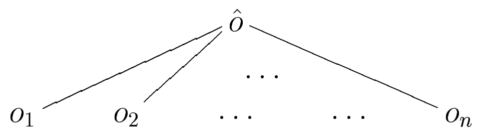


Figure 4: Object hierarchy (scheme).

Each of these objects is specified by a TROLL *light* template: let $\hat{o}$ be described by a specification $\hat{S}$, o_1 by S_1, o_2 by S_2, ..., and o_n by S_n. In general, there is no need for these specifications to be pairwise distinct. In fact, a super-object often has a large number of similar sub-objects described by only one specification. For instance, in Example 2.1 we have presented such an object community, where the super-object described by the object specification `Authorclass` has sub-objects described by the object specification `Author`.

In correspondence to the object hierarchy, we can state relationships between the specifications involved (see Fig. 5 where the directed edges mean that, e.g., specification S_1 describes a component of $\hat{S}$).

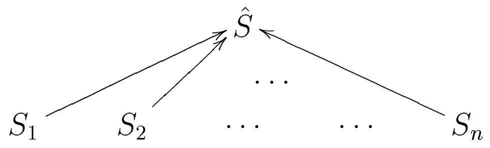


Figure 5: Relationships among the corresponding specifications.

Whereas the object of an object community always form a tree (a directed acyclic graph), the relationship graph for the corresponding specifications is directed, but in general not acyclic.

Although we often talk of $\hat{o}$ as a single object, we sometimes mean the whole object hierarchy with $\hat{o}$ as root when using $\hat{o}$. In consequence, $o_1, \ldots, o_n$ do not need to be single objects, they may stand for object hierarchies as well.

Next, we consider models. Each model describes a possible life cycle of an object community. Restricting the above given object hierarchy to the super-object $\hat{o}$ and two sub-objects o_1, o_2, the following diagram depicts a possible life cycle for $\hat{o}$ (i.e., it illustrates an interpretation structure $\mathcal{IS} = (\mathcal{W}, \mathcal{N}, \rho)$ for $\Sigma_{\hat{S}}$):

$$\mathcal{IS}: \; w_0 \xrightarrow{\hat{o}.e_1} w_1 \xrightarrow{\hat{o}.e_2,\, o_1.e_1,\, o_2.e_1} w_2 \xrightarrow{\hat{o}.e_3,\, o_1.e_2} w_3 \xrightarrow{o_1.e_3, o_2.e_2} w_4 \xrightarrow{\hat{o}.e_4,\, o_2.e_3} w_5 \xrightarrow{o_1.e_4} \cdots$$

State transitions are caused by sets of simultaneously occurring events. This life cycle for $\hat{o}$ includes life cycles for the sub-objects o_1 and o_2. The latter life cycles can be extracted by only taking the events for o_1 and for o_2, respectively, into account. Thereby, we obtain two life cycles (and, thus, two corresponding interpretation structures $\mathcal{IS}^{o_1} = (\mathcal{W}^{o_1}, \mathcal{N}^{o_1}, \rho^{o_1})$ and $\mathcal{IS}^{o_2} = (\mathcal{W}^{o_2}, \mathcal{N}^{o_2}, \rho^{o_2})$):

$$\mathcal{IS}^{o_1}: \; w_0^{o_1} \longrightarrow w_1^{o_1} \xrightarrow{o_1.e_1} w_2^{o_1} \xrightarrow{o_1.e_2} w_3^{o_1} \xrightarrow{o_1.e_3} w_4^{o_1} \longrightarrow w_5^{o_1} \xrightarrow{o_1.e_4} \cdots$$

$$\mathcal{IS}^{o_2}: \; w_0^{o_2} \longrightarrow w_1^{o_2} \xrightarrow{o_2.e_1} w_2^{o_2} \longrightarrow w_3^{o_2} \xrightarrow{o_2.e_2} w_4^{o_2} \xrightarrow{o_2.e_3} w_5^{o_2} \longrightarrow \cdots$$

With regard to the single objects, state transitions are caused by at most one event. However, there may be state transitions without an event occurrence, so-called *idle-transitions*. An idle-transition represents the possibility for the environment of the object to change without concerning the considered object itself. As we have already pointed out in Sect. 4, life cycles with idle-transitions can be considered as logically equivalent to life cycles without idle-transitions (from the local point of view of the considered object). We have indexed the states in the life cycle diagrams for o_1 and o_2 by the corresponding object names in order to clarify that these states do not equal those in the life cycle of $\hat{o}$. Especially, the functions ρ^{o_1} and ρ^{o_2} differ from ρ due to the fact that the signatures Σ_{S_1} and Σ_{S_2} are only signatures of components, i.e., they are only parts of $\Sigma_{\hat{S}}$. Hence, $\mathrm{SP{\scriptstyle ROP}}^{o_1}$ and $\mathrm{SP{\scriptstyle ROP}}^{o_2}$ are only subsets of SPROP.

Thereby, we can easily construct models for sub-objects from a given model of their super-object. In principle, we only have to restrict ρ to $\mathrm{SP{\scriptstyle ROP}}^{o_1}$ and $\mathrm{SP{\scriptstyle ROP}}^{o_2}$, respectively. Thus, a rather simple projection is sufficient for extracting models for sub-objects (Fig. 6).

In contrast, the other direction usually does not work. We cannot compose arbitrary models for components in order to obtain a model for the composed specification. On the one hand, arbitrary models of sub-objects cannot always be put together because of properties specified for the super-object, i.e., if the specification of super-objects requires an interaction between the sub-objects which is not fulfilled by the combination of the chosen models for the sub-objects. On the other hand, the super-object may have additional attributes and events for which there is no information in models of sub-objects.

$$
\begin{tikzcd}
 & & \mathcal{M}_{\hat{S}} \arrow[dll, "\pi_1"'] \arrow[dl, "\pi_2"] \arrow[d, phantom, "\cdots"] \arrow[drr, "\pi_n"] & & \\
\mathcal{M}_{S_1} & \mathcal{M}_{S_2} & \cdots & \cdots & \mathcal{M}_{S_n}
\end{tikzcd}
$$

Figure 6: Relationships among the corresponding models.

After having seen that models may be decomposed according to the object hierarchy we go over to the corresponding theories. The following considerations are valid for generated theories (Def. 5.4) as well as for semantic closures (Def. 4.9). Therefore, we just talk of theories, i.e., of sets of formulas closed under a given deduction principle. Such a set of formulas is a logical description of an object community, e.g., we have a theory $\mathcal{T}h_S$ of formulas over Σ_S.

In contrast to the projection of models in Fig. 6, the theories $\mathcal{T}h_{S_1}, \ldots, \mathcal{T}h_{S_n}$ for the sub-objects cannot be obtained by restricting $\mathcal{T}h_{\hat{S}}$ to the signatures of S_1, …, S_n, respectively. This property is however not necessary for proving object properties. Instead the opposite direction from a theory $\mathcal{T}h_{S_i}$ of a sub-object to the theory $\mathcal{T}h_{\hat{S}}$ of the super-objects has to be considered. Here, we can easily see that $\mathcal{T}h_{S_i}$ is always a (closed) sub-theory of $\mathcal{T}h_{\hat{S}}$, i.e., $\mathcal{T}h_{S_i} \subseteq \mathcal{T}h_{\hat{S}}$ for each i. In order to prove this, we can show that each formula $\phi \in \mathcal{T}h_{S_i}$ must also be included in $\mathcal{T}h_{\hat{S}}$. This is due to the fact that ϕ is fulfilled by each model $\mathcal{M}_{S_i}$ and, thereby, ϕ holds in each model $\mathcal{M}_{\hat{S}}$ as well. Otherwise, if ϕ does not hold in some model $\mathcal{M}_{\hat{S}}$, it cannot hold in the model $\mathcal{M}_{S_i}$ yielded by the model projection.

The relationships between the theories corresponding to the community signatures from Fig. 5 are depicted in Fig. 7.

$$
\begin{tikzcd}
 & & \mathcal{T}h_{\hat{S}} \arrow[d, phantom, "\cdots"] & & \\
\mathcal{T}h_{S_1} \arrow[urr, hook, "\subseteq"] & \mathcal{T}h_{S_2} \arrow[ur, hook, "\subseteq"] & \cdots & \cdots & \mathcal{T}h_{S_n} \arrow[ull, hook, "\supseteq"']
\end{tikzcd}
$$

Figure 7: Inclusion hierarchy of the corresponding theories.

The property that theories for sub-objects are included in the theory of the super-object is important for proving object properties. Having proved a property of a component locally with regard to its (sub-) object community described by S_i, this property may also be used in the context of super-object, i.e., the (monotonic) extension of community signature preserves the validity of proofs. Hence, we are able to carry out proofs as local as possible.

The properties of models and theories as we have sketched them above are strong indications of having an institution (cf. [GB92]). Indeed, we can prove that our calculus fits into an institution [Con94a].

7. Related Work and Conclusion

We have presented a basic calculus for verifying synchronously interacting objects. This calculus was originally designed for the object specification language TROLL *light* [CGH92, GCH93] which is a dialect of TROLL [JSHS91, Jun93] restricted to a small number of essential concepts. However, the calculus may be used for other object specification languages as well, provided they have a similar view of objects and, especially, they assume synchronous interaction.

For TROLL *light* we have developed a transformation yielding a set of formulas as a logical description from a given TROLL *light* specification [Con94b, Con94a]. Furthermore, in [Sie94] a first prototype of this calculus was realized on top of the generic theorem prover Isabelle [Pau90, Pau92]. By means of this prototype we have already been able to carry out several proofs. At the time being, however, the degree of automation is quite limited.

Our calculus essentially originated in the following approaches: the object calculi presented in [FM91a, FM91b] and the object specification logic OSL [SSC92]. We have combined these two approaches. From the object calculi we have mainly borrowed the syntactical structure of formulas. But we have avoided to allow terms which have to be interpreted in dependence of time. This would complicate the proof system, especially the substitution of terms would have to be handled more carefully. Instead we have adopted the view from the OSL that only predicates should be time-varying.

We have extended the local reasoning about object properties as it is possible in the object calculi to a global reasoning by allowing global event term in positional operators. This was mainly affected by the OSL in which referring to different objects is allowed. We have pursued further this idea and, thus, we have obtained a calculus which allows a more uniform treatment of objects with regard to global reasoning about an object community. Especially, we do not need to distinguish different proof systems for local and global reasoning.

Here, we have presented a basic calculus for which several extensions might be useful. We restricted the treatment of temporal properties to the use of positional operators. Introducing the temporal operators F and G is necessary for being able to talk about further temporal properties.

Another interesting direction is the integration of deontic logic concepts (cf. [MW93, WM93]). Deontic logic seems to be a promising way of dealing with temporarily non-normative behavior in information systems, e.g. the violation of integrity constraints during a transaction. A first approach preparing the integration of deontic concepts into our calculus is given in [CE94].

With regard to the intended application area, the object-oriented modeling of information systems, possible realizations of further object-oriented concepts must be considered. Especially, inheritance should be included. Although a simple form of inheritance can already be expressed within the basic calculus we do not think that this will be sufficient.

Apart from all technical extensions of the basic calculus another main aspect of future work must be the investigation of reasoning about objects and object communities in practice. Up to now, there seems to exist a general lack of knowledge about practical needs as well as a non-understanding of how to adequately reflect the interaction of communicating objects in proofs and proof systems.

Acknowledgements:

Thanks to G. Denker, H.-D. Ehrich, M. Gogolla, R. Herzig, and N. Vlachantonis for many fruitful discussions on our common work in the KorSo project. Furthermore, P. Hartel, Th. Hartmann, J. Kusch, P. Löhr-Richter, K. Neumann, and G. Saake have asked and discussed with me on certain aspects of this work. Last but not least several partners of the German project KorSo and of the working groups COMPASS and IS-CORE (Basic Research Actions funded by the CEC) have contributed.

References

[Aba89] M. Abadi. The Power of Temporal Proofs. *Theoretical Computer Science*, 65(1):35–83, 1989.

[AR93] E. Astesiano and G. Reggio. Algebraic Specification of Concurrency. In M. Bidoit and C. Choppy, editors, *Recent Trends in Data Type Specification — Proc. 8th Workshop on Specification of Abstract Data Types*, pages 1–39. Springer, Berlin, LNCS 655, 1993.

[BHL90] D. Bjorner, C.A.R. Hoare, and H. Langmaack, editors. *VDM'90: VDM and Z — Formal Methods in Software Development.* Springer, LNCS 428, 1990.

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

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

[Con94a] S. Conrad. *Ein Basiskalkül für die Verifikation von Eigenschaften synchron interagierender Objekte.* Fortschritt-Berichte Reihe 10, Nr. 295. VDI-Verlag, Düsseldorf, 1994. (to appear).

[Con94b] S. Conrad. On Certification of Specifications for TROLL *light* Objects. In H. Ehrig and F. Orejas, editors, *Proc. 9th Workshop on Abstract Data Types – 4th Compass Workshop (WADT/Compass'92)*, pages 158–172. Springer, LNCS 785, 1994.

[CSS89] J.-F. Costa, A. Sernadas, and C. Sernadas. OBL-89 Users Manual (Version 2.3). Internal Report, INESC, Lisbon, 1989.

[EDS93] H.-D. Ehrich, G. Denker, and A. Sernadas. Constructing Systems as Object Communities. In M.-C. Gaudel and J.-P. Jouannaud, editors, *Proc. TAPSOFT'93: Theory and Practice of Software Development*, pages 453–467. Springer, LNCS 668, 1993.

[EGS92] H.-D. Ehrich, M. Gogolla, and A. Sernadas. Objects and their Specification. In M. Bidoit and C. Choppy, editors, *Proc. 8th Workshop on Abstract Data Types (ADT'91)*, pages 40–65. Springer,LNCS 655, 1992.

[Fia88] J. Fiadeiro. *Cálculo de Objectos e Eventos*. PhD thesis, Instituto Superior Técnico, Technical University of Lisbon, 1988.

[FM90] J. Fiadeiro and T. Maibaum. Describing, Structuring and Implementing Objects. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Foundations of Object-Oriented Languages (Proc. REX/FOOL Workshop, Noordwijkerhood (NL))*, pages 274–310. Springer, Berlin, LNCS 489, 1990.

[FM91a] J. Fiadeiro and T. Maibaum. Temporal Reasoning over Deontic Specifications. *Journal of Logic and Computation*, 1(3):357–395, 1991.

[FM91b] J. Fiadeiro and T. Maibaum. Towards Object Calculi. In G. Saake and A. Sernadas, editors, *Information Systems — Correctness and Reusability, Workshop IS-CORE '91, ESPRIT BRA WG 3023, London*, pages 129–178. Informatik-Bericht 91–03, Technische Universität Braunschweig, 1991.

[FS90] J. Fiadeiro and A. Sernadas. Logics of Modal Terms for System Specification. *Journal of Logic and Computation*, 1(2):187–227, 1990.

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

[GB92] J.A. Goguen and R.M. Burstall. Institutions: Abstract Model Theory for Specification and Programming. *Journal of the ACM*, 39(1):95–146, 1992.

[GCH93] M. Gogolla, S. Conrad, and R. Herzig. Sketching Concepts and Computational Model of TROLL *light*. In A. Miola, editor, *Proc. 3rd Int. Conf. Design and Implementation of Symbolic Computation Systems (DISCO'93)*, pages 17–32. Springer, LNCS 722, 1993.

[Gen35] G. Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 1935.

[HCG94] R. Herzig, S. Conrad, and M. Gogolla. Compositional Description of Object Communities with TROLL light. In C. Chrisment, editor, *Proc. Basque Int. Workshop on Information Technology (BIWIT'94)*. Cepadues Society Press (France), 1994.

[JSHS91] R. Jungclaus, G. Saake, T. Hartmann, and C. Sernadas. Object-Oriented Specification of Information Systems: The TROLL Language. Informatik-Bericht 91-04, TU Braunschweig, 1991.

[Jun93] R. Jungclaus. *Modeling of Dynamic Object Systems—A Logic-Based Approach*. Advanced Studies in Computer Science. Vieweg, Braunschweig/Wiesbaden, 1993.

[Mes90] J. Meseguer. A Logical Theory of Concurrent Objects. *ACM SIGPLAN Notices*, 25(10):101–115, 1990. Proc. OOPSLA.

[MW93] J.-J.Ch. Meyer and R.J. Wieringa. Deontic Logic: A Concise Overview. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science — Normative System Specification*, pages 3–16. Wiley, Professional Computing, 1993.

[Pau90] L.C. Paulson. Isabelle: The Next 700 Theorem Provers. In P. Odifreddi, editor, *Logic and Computer Science*, pages 361–385. Academic Press, 1990.

[Pau92] L.C. Paulson. Introduction to Isabelle. Technical report, Computer Laboratory, University of Cambridge, 1992.

[San88] D. Sannella. A Survey of Formal Software Development Methods. Technical Report, University of Edinburgh, 1988.

[SE91] A. Sernadas and H.-D. Ehrich. What Is an Object, After All? In R. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design and Construction (Proc. 4th IFIP WG 2.6 Working Conference DS-4, Windermere (UK))*, pages 39–70. North-Holland, 1991.

[Sie94] Jens Siemer. Realisierung des TROLL *light*–Verifikationskalk"uls mit Hilfe des generischen Theorembeweisers ISABELLE. Master Thesis, TU Braunschweig, 1994.

[SSC92] A. Sernadas, C. Sernadas, and J.F. Costa. Object Specification Logic. Internal Report, INESC, University of Lisbon, 1992. To appear in Journal of Logic and Computation.

[SSG+91] A. Sernadas, C. Sernadas, P. Gouveia, P. Resende, and J. Gouveia. OBLOG — Object-Oriented Logic: An Informal Introduction. Technical report, INESC, Lisbon, 1991.

[Wie91] R. Wieringa. Equational Specification of Dynamic Objects. In R.A. Meersman, W. Kent, and S. Khosla, editors, *Object-Oriented Databases: Analysis, Design & Construction (DS-4), Proc. IFIP WG 2.6 Working Conference, Windermere (UK) 1990*, pages 415–438. North-Holland, 1991.

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

[WM93] R.J. Wieringa and J.-J.Ch. Meyer. Applications of Deontic Logic in Computer Science: A Concise Overview. In J.-J.Ch. Meyer and R.J. Wieringa, editors, *Deontic Logic in Computer Science — Normative System Specification*, pages 17–40. Wiley, Professional Computing, 1993.