Evolving Logical Specification in Information Systems

Stefan Conrad[1], Jaime Ramos[2], Gunter Saake[1], Cristina Sernadas[2]

1 Otto-von-Guericke–Universität Magdeburg
Institut für Technische Informationssysteme
Postfach 4120, D-39016 Magdeburg
Germany

{conrad|saake}@iti.cs.uni-magdeburg.de

2 Instituto Superior Técnico
Departamento de Matemática
Av. Rovisco Pais, P-1096 Lisboa Codex
Portugal

{jabr|css}@math.ist.utl.pt

April 1997

Abstract

Traditional logic-based specification approaches fix the structure and the dynamics of an object system at specification time. Information systems are applications with a very long life-time. Therefore, object and specification *evolution* to react on changing requirements is a relevant aspect of describing information systems as object societies.

We present a logical specification framework for evolving objects based on the concepts of object developed for the languages Troll and Gnome and the underlying temporal logic OSL. The syntactic notion of object descriptions is extended to explicitly manipulate temporal axioms during behaviour evolution. An extension of OSL called *dy*OSL establishes a logical framework where basic temporal formulae are evaluated by a second logical layer. *dy*OSL explicitly manipulates state-dependent sets of temporal formulae to model evolution of object axioms.

Syntax and semantics of *dy*OSL are given together with a translation of the specification language constructs into *dy*OSL. The corresponding proof theory is discussed together with guidelines for formally certifying object properties. Additionally, we show the use of the formalism for some specific information system scenarios.

Acknowledgments

We are grateful to Amilcar Sernadas for discussing with us this difficult problem. Work presented here was partially supported by PRAXIS XXI Project 2/2.1/MAT/262/94 SitCalc, ESPRIT III Working Group 8319 ModelAge, ESPRIT IV Working Group 22704 ASPIRE, and ESPRIT IV Working Group 23531 FIREworks.

Contents

1 Introduction **1**

2 Motivation and Language **5**

3 Syntax and Semantics of the Logic **13**

3.1 Signatures 13

3.2 Terms and Formulae 14

3.3 Pre-interpretation structures 16

3.4 Satisfaction 17

3.5 Specifications and Theories 20

4 Translation of Language into Logic **21**

5 Using the Logical Framework **25**

5.1 A Hilbert calculus 25

5.2 An invariant calculus 27

6 Concluding Remarks **31**

Chapter 1

Introduction

In the last years, logic-based approaches for specifying information systems were developed and investigated. Especially, in combination with object-oriented principles for the design of information systems ([Sernadas et al., 1991b]) a lot of work was done to find adequate logic-based approaches to describe the dynamic behaviour as well as the interaction of objects in information systems.

Mainly for describing the dynamic behaviour of objects in information systems, temporal logic ([Manna and Pnueli, 1992]) is a good and widely accepted choice. This is due to the fact that information systems (as a generalization of database systems) are state-based systems for which the state-based approach of temporal logic is obviously appropriate (cf. [Khosla and Maibaum, 1989]).

Therefore, it is no wonder that the (temporal) specification of information systems became a popular research area, especially because it is based on a clean and well-understood theory (cf. e.g. [Fiadeiro and Maibaum, 1991, Ehrich et al., 1993, Conrad, 1996, Sernadas et al., 1996]). Furthermore, several specification languages like ALBERT ([Dubois et al., 1993]), LCM ([Wieringa, 1991, Feenstra and Wieringa, 1993]), OBLOG ([Sernadas et al., 1987, Sernadas et al., 1991a]), GNOME ([Sernadas and Ramos, 1994]), and TROLL ([Jungclaus et al., 1996]) have been developed for supporting the specification of objects and their dynamic behaviour based on temporal (or dynamic) logics.

Of course there are other formal approaches to specifying dynamic behaviour of objects. For instance, conditional term rewriting ([Meseguer, 1993]) is another way of describing object behaviour. The specification language Maude ([Meseguer, 1992, Meseguer and Qian, 1993]) is based on such a conditional term rewriting semantics. In the database area a lot of interesting work concerning the description of database dynamics (e.g. in [Lipeck and Thalheim, 1993]) and temporal databases (cf. [Tansel et al., 1993]) can be found.

In contrast to other approach (like the evolving algebra approach by [Gurevich, 1991, Gurevich, 1993]) in our context the notion of evolution refers to the evolution of specifications during the runtime of a system.

However, all these approaches to specifying information systems assume that the dynamic behaviour of the system and its parts is totally known at specification time. In addition, once specified the behaviour is fixed for the entire life (or run) time of the information system. This restriction seems not to be adequate for a large number of typical information system applications. Because of the long life- (or run-) time of information systems changes of the dynamic behaviour are often required within the lifetime of a system.

Typical examples for such changes of the dynamic behaviour can already be found in usual application areas for information systems like libraries and banks. For a library it may happen that the rules for borrowing and returning books change in order to avoid too much administrative work in writing reminders. Due to changes of laws (or bank rules) it could become necessary to change the computation of yearly interests or to restrict the evolution of an account in some way. These changes are modifications of axioms describing the dynamic behaviour of the system. In consequence, we have to find a framework for describing evolving specifications where we can specify the evolution of first-order dynamic behaviour specifications.

In this paper we continue work started in [Saake et al., 1995b] where a first, very restricted form of evolving temporal specification was proposed. In [Conrad and Saake, 1995] and [Conrad and Saake, 1996] a first ad hoc formalization of a temporal logic allowing evolving behaviour was presented and discussed. Different ways of handling the evolution of specifications within a specification language were investigated in [Türker et al., 1996]. After having consolidating this preparatory work we are now able to present a proposal for an adequate specification language together with a formal definition of an extended temporal logic for specifying objects and their behaviour which may be dynamically modified.

For the logical part we build on the *Object Specification Logic* OSL proposed in [Sernadas et al., 1995]. For our purposes we need to distinguish two logical levels: the first level allows the usual temporal specification of the objects' behaviour by means of OSL, and the second level allows us to deal with behaviour evolution. By this separation into two levels we are able to define a clean extension of OSL. For the purposes of this paper, we focus the presentation on the treatment of behaviour evolution and ignore object-oriented concepts like inheritance. Incorporating those concepts into our logic can be done in the same way as it has been done for the original OSL in [Sernadas et al., 1995].

The remainder of this paper is organized as follows: In Chapter 2 we introduce and discuss an example specification for documents in an information system managing offers and contracts in a sales department. By means of this example we motivate the need for evolving behaviour specification and we additionally introduce basic parts of a corresponding object specification language. In Chapter 3 we define syntax and semantics of the logic, called *dynamic* OSL (*dy*OSL). Afterwards, we show how to translate the interesting parts of specifications, namely the evolving behaviour part, into *dy*OSL(Chapter 4). The usage of our logical framework in proving properties of objects from their specification is briefly dealt with in Chapter 5. Finally, we conclude by summarizing our approach presented in this paper and by pointing out open questions for future work.

Chapter 2

Motivation and Language

In this chapter, we discuss the limitations of current object-oriented specification approaches using temporal logic for describing the dynamic behaviour of objects. In order to overcome these limitations we propose an extension allowing evolution of the dynamic behaviour of objects. As starting point we use TROLL which is one of the object specification languages developed so far. By means of examples we motivate the extensions of the logic that are needed.

For presenting the current status of object-oriented specification we introduce a simple example.

Example 2.1 In an information system for the sales department of an enterprise all offers made to customers and all contracts with customers are to be managed. In Figure 2.1 a specification of an object class for documents (as a unifying notion for offers and contracts) is given using a TROLL-like specification language.

Here, documents have four attributes. The document number 'DocNo' is unique for each document. In the **identification** part the attribute 'DocNo' is specified to be used for identifying document objects. Therefore, there must not be duplicate document numbers in our information system. The document type 'DocType' states whether a document is currently an offer made to a customer, a preliminary contract which has to be confirmed by the customer, or already an official (legal) contract. The attribute 'Valid' provides a date until an offer or a preliminary contract is valid. The document text is stored as a value of the attribute 'Content'.

There are five kinds of events (actions) which may occur for document objects. The occurrence of a 'create' event denotes the birth of a document object, i.e. a new document is created. During the existence of the object 'revise', 'prepare_contract', and 'fix_contract' events may occur. A document is deleted when a 'resolve' event occurs for the corresponding object. The effect of each of these events is described in the **rigid axioms** part. Therefore, the occurrence of a 'create' event results in the specified values for the attributes. Moreover, the event 'addDocToOffers' is called in another object which has the name 'DocManager' (assuming that this object is specified elsewhere).

```
object class Documents
   identification  DocId: (DocNo)
  template
   attributes
      DocNo:       int,
      DocType:    {offer, pre_contract, contract},
      Valid:        date,
      Content:     text;
   actions
      birth  create(DocNo:int,Content:text),
      revise(NewContent:text),
      prepare_contract,
      fix_contract,
      death  resolve;
   rigid axioms
      create(D,C)
        changing  DocType = offer,
                  DocNo    = D,
                  Content  = C,
                  Valid      = now + 30
        calling   DocManager.addDocToOffers(self);
      revise(C)
        enabled    DocType = offer and Valid ≥ now,

        changing  Valid    = now + 30,
                  Content = C;
      prepare_contract
        enabled    DocType = offer and Valid ≥ now,
        changing  DocType = pre_contract,
                  Valid      = now + 10,
                  Content  = C;
      fix_contract
        enabled    DocType ≠ contract and Valid ≥ now,
        changing  DocType = contract,
        calling  DocManager.mvOfferToContracts(self);
      resolve
        calling
          [DocType≠contract] DocManager.mvOfferToArchive(self),
          [DocType=contract] DocManager.mvContractToArchive(self);
end object class
```

Figure 2.1: A specification of documents.

In addition to their effect on attribute values and the calling of other events we impose enabling conditions for 'revise', 'prepare_contract', and 'fix_contract' events. Thereby, such events are only allowed to occur if the corresponding enabling condition is satisfied. The calling of other events can also be combined with additional pre-conditions. For instance, the occurrence of a 'resolve' event may cause the occurrence of two different events in the object called 'DocManager'. Which of these two events is called depends on the value of the attribute 'DocType'.

The possible evolution of a document is specified such that, when it is created, it is an offer being valid for 30 days. Offers may be revised during their validity. After each revision the offer is again valid for 30 days. In case the customer is satisfied by the offer, the document can be changed to a contract. There are two ways for an offer to become a contract. The direct way is fixing the offer and turning it immediately into a contract. For that, the 'fix_contract' event can be used.

Sometimes, it might be useful to have an additional phase for preparing the official contract. By means of the 'prepare_contract' event the offer can be turned into a pre-contract. A pre-contract is only valid for 10 days. Within that period the contract must be fixed finally. Otherwise, the pre-contract becomes invalid. □

Considering the specification it becomes obvious that problems might occur in the future during the runtime of the system. The complete behaviour is fixed at specification time, i.e. the specification can only respect the possible dynamic behaviour of document objects in the information system as far as it can be foreseen at that moment. Of course, if we already expect that certain changes of requirements will occur later, we are able to specify that in the usual object specification framework.

However, if we look at the evolution of information systems in practice, we have to face the fact that there often are changes of requirements which could not be foreseen at specification time. Such changes may be due to the enacting of new laws or due to the adaptation of business rule to a unforeseen evolution of the market.

In usual specification approaches such changes are not allowed. In case the specification of an information system has to be modified the implemented system which is already running for some time does not fulfill the modified specification anymore. In consequence, the implementation must be modified according to the changes in the specification. However, considering the complete lifetime of an information system there is an obvious problem. For each instant of time the current implementation fits to the current specification.

Unfortunately, neither the original nor the modified specification describes the complete evolution of the objects in the information system. This is due to the fact that the modification of the dynamic behaviour was not described in the original specification and that the modified specification does usually not include the evolution of the system before the specification was modified.

In order to overcome this dilemma which is due to the restrictions of usual specification approaches we are developing a novel specification framework with more flexibility. The main idea of our approach is that we distinguish between rigid axioms and non-rigid axioms. Rigid axioms are axioms which cannot be changed at all. In contrast non-rigid axioms may be added, removed, or changed during runtime. For that additional actions must be introduced into the specification. We call this special kind of actions “mutators” or “mutation actions” (“mutation events” for concrete instances) because their occurrence causes a *mutation* of the object specification. In order to be able to introduce new axioms by occurrences of such mutation actions, a special type for axioms (**spec**) is provided. This type must not be used within rigid axioms. Only mutation events may have axioms as parameter.

For managing the set of currently valid non-rigid axioms we have to introduce a special kind of attribute called ‘axiom attribute’. In any case one axiom attribute is needed in which the set of currently valid non-rigid axioms is stored. In the following examples we call this axiom attribute ‘Axioms’. In fact, it might be useful for certain applications to have several such axioms attributes, e.g. for being able to switch between different sets of behaviour axioms (cf. [Türker et al., 1996]). In that case one of these axiom attributes must be marked as the one containing the currently valid axioms. For simplifying the presentation we only use one axiom attribute in the examples of this paper.

Example 2.2 Continuing our example, we now add a simple mechanism for allowing evolving behaviour. Due to the fact that the rigid part of the behaviour specification has already been given in Figure 2.1, we only present the extension of the specification in Figure 2.2.

The extension given here is a rather unrestricted one. It allows to add arbitrary behaviour axioms (and remove those axioms later on). For doing so, special actions, called **mutators**, are introduced. By means of an occurrence of a ‘add_new_rule’ mutation action we are able to add an arbitrary formula as a behaviour axiom for document objects. For instance, this formula may describe additional effects of an action or impose an additional enabling condition for some action. □

From a logical point of view, adding a new axiom can be considered as restricting the further behaviour of the object. Thereby, it is not possible to change that part of the behaviour which is specified in the **rigid axioms** section. It is only possible to specify additional effects of actions and to introduce restrictions e.g. with regard to enabling of actions. In this way it is also possible to disallow certain actions to occur at all.

Example 2.3 In the following we present examples for non-rigid axioms which could be added during runtime of our information system for managing documents. The syntax for representing those axioms is close to that we use for object specifications.

1. Due to a change of law it might become necessary to store the date when a contract was fixed. In order to bring this new requirement into our specification an occurrence of a mutation event ‘add_new_rule’ with the following parameter can be used:

```
object class Documents
      identification  DocId: (DocNo)
    template
      attributes
        ⋮
      actions
        ⋮
      rigid axioms
        ⋮
      axiom attributes
        Axioms    initialized { }
      mutators
        add_new_rule(Rule:spec)
        remove_rule(Rule:spec)
      dynamic specification
        add_new_rule(Rule)
          changing  Axioms = Axioms + { Rule }
        remove_rule(Rule)
          changing  Axioms = Axioms − { Rule }
  end object class
```

Figure 2.2: Extending the specification for documents.

```
fix_contract
    changing  Valid = now
```

Thereby, 'fix_contract' events will have an additional effect. They will set the attribute 'Valid' to the current date. Thereby, the date of fixing the contract can later be accessed.

2. Another kind of additional effects of an event can also be specified in the some way as it would be done for rigid axioms. For instance, let us assume that there is a special supervisor object called 'DocManager' which is responsible for keeping track of the evolution of documents. This supervisor object may cause the occurrence of a mutation event 'add_new_rule' with the following parameter:

   ```
   revise(C)
       calling  DocManager.inform_me(DocNo, C)
   ```

   From that moment each occurrence of a 'revise' event causes an occurrence of an 'inform_me' event in the supervisor object. This is new effect of 'revise' events—in addition to the effects described as rigid axioms.

3. In the same way we can restrict the enabling of events. For instance, due to a change of laws it might become necessary to keep document objects alive forever in case they

have reached to state of an official contract. In order to adapt the system to this new requirement we could add to following axiom which allows the occurrence of 'resolve' events only in case the document is not already a contract:

$$\begin{array}{l}\text{resolve}\\ \qquad \textbf{enabled}\ \ \text{DocType} \neq \text{contract}\end{array}$$

4. By imposing a pre-condition which cannot be fulfilled we can even disallow events to occur at all. For instance, we can forbid 'prepare_contract' events by adding the following axiom:

$$\begin{array}{l}\text{prepare\_contract}\\ \qquad \textbf{enabled}\ \ \textbf{false}\end{array}$$

Please note that we now have two enabling conditions which have to be fulfilled in case a 'prepare_contract' is to occur. The enabling condition '$\text{DocType} = \text{offer}$' which is specified as a rigid axiom is still valid. In addition, the new enabling condition '**false**' must be respected.

5. Of course, we may also add other kinds of axioms. For example, the following axiom is an integrity constraint restricting the allowed states of an object:

$$\begin{array}{l}\text{Valid} > \textbf{now} \ \ \rightarrow\\ \qquad (\text{DocType} = \text{offer}\ \textbf{or}\ \text{DocType} = \text{pre\_contract})\end{array}$$

This axiom requires documents to be an offer or a pre-contract in case the validity date is set to some date in the future. □

Due to the fact that everything which is specified in the **rigid axioms** section cannot be changed during runtime of the information system, we have to be very careful in specifying our system. In case we expect (or it could be possible) that the effect an event has on a certain attribute might change we should not specify the effect of this event in the **rigid axioms** section. Instead we can specify the default effect as part of the initial value of the axiom attribute 'Axioms'. Then, it is possible to change it later.

Of course, we should avoid to specify objects in a way that everything can be changed later. In that case, it will be rather difficult to prove interesting properties for those objects. Therefore, specifying objects requires a thorough design concerning the question which properties are rigid and which properties may be affected by evolution.

For being able to specify the evolution of dynamic behaviour as adequate as possible we can also impose enabling conditions to mutation events. Thereby, changes of the set of currently valid non-rigid behaviour axioms (the contents of the axiom attribute Axioms) can be better controlled. In certain cases it might even be useful to have integrity constraints restricting the contents of the axiom attribute to fulfill certain properties. The following example presents the restriction of a mutation event by imposing an enabling condition.

Example 2.4 The enabling of mutation actions can be restricted in the same way as it is possible for other events:

$\vdots$

dynamic specification

add_new_rule(Rule)

enabled DocType $\neq$ contract

changing Axioms $=$ Axioms $+ \{$ Rule $\}$

$\vdots$

Here, we require that the document type of a document must be different from 'contract' in order to allow occurrences of 'add_new_rule' mutation events. □

Considering the evolution of information systems in practice clearly shows that there is a real need for allowing behaviour evolution. Although there are several possibilities for extending existing specification language in an *ad hoc* manner, the question how evolving behaviour can adequately be reflected on the semantical level has still to be answered. In the next section we define syntax and semantics of a logic for that purpose.

Chapter 3

Syntax and Semantics of the Logic

In this chapter, we will present syntax and semantics of the logic *dy*OSL. *dy*OSL extends the logic OSL presented in [Sernadas et al., 1995] with a second temporal level, the *meta level*, allowing evolving axiom attributes.

3.1 Signatures

We assume as given once and for all the underlying data signature $\Sigma_{DT} = \langle S_{DT}, OP \rangle$, with $bool \in S_{DT}$ and $OP_{\epsilon,bool} = \{false, true\}$. We also assume as given a set of sorts S_{OB} (object identifiers).

Definition 3.1 *dy*OSL is based on the following sets of *sorts*: $S = S_{DT} \oplus S_{OB}$, $\underline{S} = S \oplus \{\tau\}$ and $\overline{S} = \underline{S} \oplus \{\mu\} \oplus \{\sigma\}$. □

The sort τ is called *action sort*, the sort μ is called *mutation sort* and the sort σ is called *specification sort*.

Definition 3.2 An *object signature* $\overline{\Sigma}$ is a tuple $\langle ACT, ATT, MUT, MAT \rangle$ such that:

- ACT is an S^*-indexed family of finite sets such that $\underline{\perp} \in ACT_\epsilon$;
- ATT is an $\underline{S}^* \otimes \underline{S}$-indexed family of finite sets such that $\underline{occ} \in ATT_{\epsilon,\tau}$;
- MUT is an $(\underline{S} + \{\sigma\})^*$-indexed family of finite sets such that $\overline{\perp} \in MUT_\epsilon$;
- MAT is an $\overline{S}^*$-indexed family of finite sets such that:
  - $\overline{occ} \in MAT_{\epsilon,\mu}$;
  - $Ax \in MAT_{\epsilon,\sigma}$. □

Given an object signature $\overline{\Sigma} = \langle ACT, ATT, MUT, MAT \rangle$, each element $z \in ACT_w$ is said to be an *action symbol* with parameter sorts w, each element $a \in ATT_{w,s}$ is said to be an *attribute symbol* with parameter sorts w and result sort s, each element $m \in MUT_w$ is said to be a *mutation action symbol* with parameter sorts w, and each element $q \in MAT_{w,s}$ is said to be a *mutation attribute symbol* with parameter sorts w and result sort s. An attribute symbol in $ATT_{\epsilon,s}$ with $s \in S$ is said to be a *slot symbol.* A mutation attribute symbol in $MAT_{\epsilon,\sigma}$ is said to be a *specification attribute symbol.* Specification attributes store sets of temporal axioms. The special specification attribute Ax contains the currently active set of axioms, whereas other specification attributes may be used to store sets of axioms for example during exceptional situations where the usual behaviour is overwritten for a period of time.

In the sequel we assume a given signature $\overline{\Sigma} = \langle ACT, ATT, MUT, MAT \rangle$.

3.2 Terms and Formulae

We assume as given once and for all an $\underline{S}$-indexed family $\underline{X}$ of infinite, pairwise disjoint sets of *base variables.*

Definition 3.3 The $\underline{S}$-indexed family $\underline{\mathcal{T}}$ of sets of *base terms* is inductively defined as follows:

- $y \in \underline{\mathcal{T}}_s$, provided that $y \in \underline{X}_s$;
- $f(t_1, \ldots, t_n) \in \underline{\mathcal{T}}_s$, provided that $f \in OP_{s_1 \ldots s_n, s}$ and $t_i \in \underline{\mathcal{T}}_{s_i}$, for every $i = 1, \ldots, n$;
- $z(t_1, \ldots, t_n) \in \underline{\mathcal{T}}_\tau$, provided that $z \in ACT_{s_1 \ldots s_n}$ and $t_i \in \underline{\mathcal{T}}_{s_i}$, for every $i = 1, \ldots, n$;
- $a(t_1, \ldots, t_n) \in \underline{\mathcal{T}}_s$, provided that $a \in ATT_{s_1 \ldots s_n, s}$ and $t_i \in \underline{\mathcal{T}}_{s_i}$, for every $i = 1, \ldots, n$.

$\square$

Each element of $\underline{\mathcal{T}}_s$ is called a base term of sort s.

Definition 3.4 The set $\underline{\mathcal{L}}$ of *base formulae* is inductively defined as follows:

- $t \in \underline{\mathcal{L}}$, provided that $t \in \underline{\mathcal{T}}_{bool}$;
- $(t = t') \in \underline{\mathcal{L}}$, provided that $t, t' \in \underline{\mathcal{T}}_s$;
- $\star \in \underline{\mathcal{L}}$;
- $(\neg\varphi) \in \underline{\mathcal{L}}$, provided that $\varphi \in \underline{\mathcal{L}}$;
- $(\varphi \Rightarrow \varphi'), (\varphi \mathsf{U} \varphi') \in \underline{\mathcal{L}}$, provided that $\varphi, \varphi' \in \underline{\mathcal{L}}$;

- $(\forall y\ \varphi) \in \underline{\mathcal{L}}$, provided that $\varphi \in \underline{\mathcal{L}}$ and $y \in \underline{X}_s$. □

As usual we are free to introduce abbreviations. In the sequel we use: $(\varphi \vee \varphi')$ for $((\neg\varphi) \Rightarrow \varphi')$, $(\varphi \wedge \varphi')$ for $(\neg((\neg\varphi) \vee (\neg\varphi')))$, $(\varphi \Leftrightarrow \varphi')$ for $((\varphi \Rightarrow \varphi') \wedge (\varphi' \Rightarrow \varphi))$, $(\exists y\ \varphi)$ for $(\neg(\forall y\ (\neg\varphi))$, $(\mathsf{X}\,\varphi)$ for $((\varphi \wedge (\neg\varphi))\ \mathsf{U}\ \varphi)$, $(\mathsf{F}\,\varphi)$ for $((\varphi \vee (\neg\varphi))\ \mathsf{U}\ \varphi)$, and $(\mathsf{G}\,\varphi)$ for $(\neg(\mathsf{F}(\neg\varphi)))$.

Additionally, we will use the *weak until operator*. It is defined as follows: $(\varphi\, \mathsf{U}^?\, \psi)$ is an abbreviation of $((\mathsf{G}\,\varphi) \vee (\varphi\, \mathsf{U}\, \psi))$. In the case of $\mathsf{U}^? *$ we mean $\varphi\, \mathsf{U}^?\, *$ where φ is any formula meaning that φ must hold until the next mutation if this one occurs.

The first level of our logic (the base level) defined so far is a usual linear temporal logic. The following definitions extend this logic to a second level (the meta level) manipulating *sets of axioms*.

As for the base level, we also assume as given once and for all an $\overline{S}$-indexed family $\overline{X}$ of infinite, pairwise disjoint sets of *meta variables*, such that $\overline{X}_s = \underline{X}_s$ for each $s \in \underline{S}$.

A formula without temporal operators and without the attribute symbol $\underline{occ}$ is called a *state formula*. We denote the set of all state formulae (at the base level) by $\underline{st\mathcal{L}}$.

Definition 3.5 The $\overline{S}$-indexed family $\overline{\mathcal{T}}$ of sets of *meta terms* is inductively defined as follows:

- $y \in \overline{\mathcal{T}}_s$, provided that $y \in \overline{X}_s$;
- $t \in \overline{\mathcal{T}}_s$, provided that $t \in \underline{\mathcal{T}}_s$;
- $m(t_1, \ldots, t_n) \in \overline{\mathcal{T}}_\mu$, provided that $m \in MUT_{s_1 \ldots s_n}$ and $t_i \in \overline{\mathcal{T}}_{s_i}$, for every $i = 1, \ldots, n$;
- $b(t_1, \ldots, t_n) \in \overline{\mathcal{T}}_s$, provided that $b \in MAT_{s_1 \ldots s_n, s}$ and $t_i \in \overline{\mathcal{T}}_{s_i}$, for every $i = 1, \ldots, n$;
- $[\varphi] \in \overline{\mathcal{T}}_\sigma$, provided that $\varphi \in \underline{\mathcal{L}}$. □

Each element of $\overline{\mathcal{T}}_s$ is called a meta term of sort s. A set $\{\varphi_0, \ldots, \varphi_n\}$ of base formulae can be denoted by the term $[\varphi_0 \wedge \ldots \wedge \varphi_n] \in \overline{\mathcal{T}}_\sigma$.

Definition 3.6 The set $\overline{\mathcal{L}}$ of *meta formulae* is inductively defined as follows:

- $t \in \overline{\mathcal{L}}$, provided that $t \in \overline{\mathcal{T}}_{bool}$;
- $(t = t') \in \overline{\mathcal{L}}$, provided that $t, t' \in \overline{\mathcal{T}}_s$;
- $(\neg\psi) \in \overline{\mathcal{L}}$, provided that $\psi \in \overline{\mathcal{L}}$;
- $(\psi \Rightarrow \psi'), (\psi\, \mathsf{U}\, \psi') \in \overline{\mathcal{L}}$, provided that $\psi, \psi' \in \overline{\mathcal{L}}$;

- $(\forall y\ \psi) \in \overline{\mathcal{L}}$, provided that $\psi \in \overline{\mathcal{L}}$ and $y \in \overline{X}_s$. □

Like for the base level, we are also free to introduce the first-order and temporal *abbreviations* for meta formulae.

Analogously to $\underline{st\mathcal{L}}$, a formula without temporal operators and without the meta attribute symbol $\overline{occ}$ is called a *state meta formula*. We denote the set of all state meta formulae by $\overline{st\mathcal{L}}$.

3.3 Pre-interpretation structures

We assume given an algebra dt over $\Sigma_{DT} = \langle S_{DT}, OP\rangle$ such that $dt_{bool} = \{false_{dt}, true_{dt}\}$. We also assume given an S_{OB}-indexed family *pop* (*pop* for *population*) of carrier sets for S_{OB}, and we extend the notation so that pop_s stands for dt_s when $s \in S_{DT}$, and pop_σ stands for $\underline{\mathcal{L}}$.

Definition 3.7 The set *act* of *actions* over Σ for *pop* is as follows:

$$\bigcup_{\langle s_1 \dots s_n\rangle \in S^*} \{z(u_1, \dots, u_n) : z \in ACT_{s_1 \dots s_n}, u_i \in pop_{s_i},\ \text{for } i = 1, \dots, n\}$$

□

In the sequel we write pop_τ for *act*.

Definition 3.8 The set att_s of *attributes of sort s* over Σ for *pop* is as follows:

$$\bigcup_{\langle s_1 \dots s_n\rangle \in \underline{S}^*} \{a(u_1, \dots, u_n) : a \in ATT_{s_1 \dots s_n, s}, u_i \in pop_{s_i},\ \text{for } i = 1, \dots, n\}$$

□

Definition 3.9 The set of *attributes* over Σ for *pop* is as follows:

$$\bigcup_{s \in \underline{S}} att_s$$

□

Definition 3.10 The set *mut* of *mutations* over Σ for *pop* is as follows:

$$\bigcup_{\langle s_1 \dots s_n\rangle \in (\underline{S} \oplus \{\sigma\})^*} \{m(u_1, \dots, u_n) : m \in MUT_{s_1 \dots s_n}, u_i \in pop_{s_i},\ \text{for } i = 1, \dots, n\}$$

□

In the sequel we write pop_μ for mut.

Definition 3.11 The set mat_s of *mutation attributes of sort s* over Σ for pop is as follows:

$$\bigcup_{\langle s_1 \ldots s_n \rangle \in \overline{S}^*} \{b(u_1, \ldots, u_n) : b \in MAT_{s_1 \ldots s_n, s}, u_i \in pop_{s_i}, \text{ for } i = 1, \ldots, n\}$$

□

Definition 3.12 The set of *mutation attributes* over Σ for pop is as follows:

$$\bigcup_{s \in \overline{S}} mat_s$$

□

We use the $\overline{S}$-indexed family obs to denote $att \oplus mat$, defined as follows:

- $obs_s = att_s \oplus mat_s$, for each $s \in \underline{S}$;
- $obs_s = mat_s$, for each $s \in (\overline{S} \setminus \underline{S})$.

For each element $o \in obs_s$, called *observation symbol* of sort s, we say that pop_s is *domain* D_o of o.

Definition 3.13 A *pre-interpretation structure* over Σ is triple $I = \langle dt, \{pop\}_{s \in S_{OB}}, \lambda \rangle$, with $\lambda = \{\lambda_k(o)\}_{k \in I\!N, o \in obs}$, such that:

- $\lambda_k(o) \in pop_s$ whenever $o \in obs_s$;
- $\lambda_k(\underline{occ}) \neq \underline{\perp} \Rightarrow \lambda_k(\overline{occ}) = \overline{\perp}$;
- if $\lambda_k(\underline{occ}) = \underline{\perp}$ then $\lambda_{k+1}(o) = \lambda_k(o)$ for each $o \in att$ such that $o \neq \underline{occ}$;
- if $\lambda_k(\overline{occ}) = \overline{\perp}$ then $\lambda_{k+1}(o) = \lambda_k(o)$ for each $o \in mat$ such that $o \neq \overline{occ}$. □

3.4 Satisfaction

After having defined the syntax of the logic and the underlying semantic structures, we can now start to define the satisfaction relation between formulae and structures.

Definition 3.14 An *assignment* ρ for pop is an $\overline{S}$-indexed family of maps $\rho_s : \overline{X}_s \to pop_s$.
□

Definition 3.15 Let $y \in \overline{X}_s$ and ρ, ρ' be assignments. We say that ρ and ρ' are *y-equivalent* iff for every $s' \in \overline{S}$ and $y' \in \overline{X}_{s'}$, $\rho_{s'}(y') = \rho'_{s'}(y')$ whenever $y' \neq y$. □

Definition 3.16 The *interpretation* of base terms in I for ρ at point $k \in I\!N$ is inductively defined as follows:

- $\underline{[\![y]\!]}_{I,\rho,k,s} = \rho_s(y)$;
- $\underline{[\![f(t_1,\ldots,t_n)]\!]}_{I,\rho,k,s} = f_{dt_{s_1\ldots s_n,s}}(\underline{[\![t_1]\!]}_{I,\rho,k,s_1},\ldots,\underline{[\![t_n]\!]}_{I,\rho,k,s_n})$;
- $\underline{[\![z(t_1,\ldots,t_n)]\!]}_{I,\rho,k,\tau} = z(\underline{[\![t_1]\!]}_{I,\rho,k,s_1},\ldots,\underline{[\![t_n]\!]}_{I,\rho,k,s_n})$;
- $\underline{[\![a(t_1,\ldots,t_n)]\!]}_{I,\rho,k,s} = \lambda_k(a)(\underline{[\![t_1]\!]}_{I,\rho,k,s_1},\ldots,\underline{[\![t_n]\!]}_{I,\rho,k,s_n})$. □

Definition 3.17 The *interpretation* of meta terms in I for ρ at point $k \in I\!N$ is inductively defined as follows:

- $\overline{[\![y]\!]}_{I,\rho,k,s} = \rho_s(y)$;
- $\overline{[\![t]\!]}_{I,\rho,k,s} = \underline{[\![t]\!]}_{I,\rho,k,s}$;
- $\overline{[\![m(t_1,\ldots,t_n)]\!]}_{I,\rho,k,\mu} = m(\overline{[\![t_1]\!]}_{I,\rho,k,s_1},\ldots,\overline{[\![t_n]\!]}_{I,\rho,k,s_n})$;
- $\overline{[\![b(t_1,\ldots,t_n)]\!]}_{I,\rho,k,s} = \lambda_k(b)(\overline{[\![t_1]\!]}_{I,\rho,k,_1},\ldots,\overline{[\![t_n]\!]}_{I,\rho,k,s_n})$;
- $\overline{[\![[\varphi]]\!]}_{I,\rho,k,\sigma} = \varphi$. □

Definition 3.18 The *satisfaction* of base formulae in I for ρ at point $k \in I\!N$ is inductively defined as follows:

- $I,\rho,k \,\underline{\Vdash}\, t$ iff $\underline{[\![t]\!]}_{I,\rho,k,bool} = true_{dt}$;
- $I,\rho,k \,\underline{\Vdash}\, (t = t')$ iff $\underline{[\![t]\!]}_{I,\rho,k,s} = \underline{[\![t']\!]}_{I,\rho,k,s}$;
- $I,\rho,k \,\underline{\Vdash}\, \star$ iff $\overline{[\![\overline{occ}]\!]}_{I,\rho,k,\mu} \neq \overline{\bot}$;
- $I,\rho,k \,\underline{\Vdash}\, (\neg\varphi)$ iff not $I,\rho,k \,\underline{\Vdash}\, \varphi$
- $I,\rho,k \,\underline{\Vdash}\, (\varphi \Rightarrow \varphi')$ iff not $I,\rho,k \,\underline{\Vdash}\, \varphi$ or $I,\rho,k \,\underline{\Vdash}\, \varphi'$;
- $I,\rho,k \,\underline{\Vdash}\, (\varphi \,\mathsf{U}\, \varphi')$ iff there is $k' \in I\!N$ such that $k < k'$, $I,\rho,k' \,\underline{\Vdash}\, \varphi'$, and for each $k'' \in I\!N$ if $k < k'' < k'$ then $I,\rho,k'' \,\underline{\Vdash}\, \varphi$;

- $I, \rho, k \underline{\Vdash} (\forall\ y\ \varphi)$ iff $I, \rho', k \underline{\Vdash} \varphi$ for every ρ' y-equivalent to ρ. □

Definition 3.19 Given a pre-interpretation structure I and a base formula $\varphi \in \underline{\mathcal{L}}$ we say that I *satisfies* φ, written $I \underline{\Vdash} \varphi$, iff $I, \rho, 0 \underline{\Vdash} \varphi$ for every assignment ρ.

It should be remarked, that we use an *anchored version of temporal logic* for *dy*OSL. This due to the fact that we do not want to have an implicit necessitation to all formulae dynamically added to the specification by mutations. Exactly that would be the case in a fluent version of temporal logic. Because it should be possible to again remove an axiom dynamically added before, we must avoid the implicit necessitation which states that something holds for ever and cannot be removed.

Definition 3.20 The *satisfaction* of meta formulae in I for ρ at point $k \in I\!N$ is inductively defined as follows:

- $I, \rho, k \overline{\Vdash}\, t$ iff $\overline{[\![t]\!]}_{I,\rho,k,bool} = true_{dt}$;
- $I, \rho, k \overline{\Vdash} (t = t')$ iff $\overline{[\![t]\!]}_{I,\rho,k,s} = \overline{[\![t']\!]}_{I,\rho,k,s}$;
- $I, \rho, k \overline{\Vdash} (\neg\psi)$ iff not $I, \rho, k \overline{\Vdash} \psi$
- $I, \rho, k \overline{\Vdash} (\psi \Rightarrow \psi')$ iff not $I, \rho, k \overline{\Vdash} \psi$ or $I, \rho, k \overline{\Vdash} \psi'$;
- $I, \rho, k \overline{\Vdash} (\psi \mathsf{U} \psi')$ iff there is $k' \in I\!N$ such that $k < k'$, $I, \rho, k' \overline{\Vdash} \psi'$, and for each $k'' \in I\!N$ if $k < k'' < k'$ then $I, \rho, k'' \overline{\Vdash} \psi$;
- $I, \rho, k \overline{\Vdash} (\forall\ y\ \psi)$ iff $I, \rho', k \overline{\Vdash} \psi$ for every ρ' y-equivalent to ρ. □

Definition 3.21 Given a pre-interpretation structure $I = \langle dt, \{pop\}_{s \in S_{OB}}, \lambda \rangle$ we say that I is an *interpretation structure* iff, for every assignment ρ and $k \in I\!N$,$I, \rho, k \underline{\Vdash} \lambda_k(Ax)$ whenever $\lambda_{k-1}(\overline{occ}) \neq \overline{\bot}$. □

Definition 3.22 Given an interpretation structure I and a meta formula $\psi \in \overline{\mathcal{L}}$, we say that I *satisfies* ψ, written $I \overline{\Vdash} \psi$, iff $I, \rho, 0 \overline{\Vdash} \psi$ for every assignment ρ. □

Definition 3.23 Let $\Psi \subseteq \overline{\mathcal{L}}$ and $\psi \in \overline{\mathcal{L}}$. We say that Ψ *entails* ψ, written $\Psi \overline{\vDash} \psi$ iff, for every interpretation structure I, if $I \overline{\vDash} \varphi$, for every $\varphi \in \Psi$, then $I \overline{\vDash} \psi$. □

Definition 3.24 Let $\Psi \subseteq \overline{\mathcal{L}}$. The *closure by entailment* of Ψ is the set $\Psi^{\overline{\vDash}} = \{\psi : \Psi \overline{\vDash} \psi\}$. □

3.5 Specifications and Theories

Finally, we can define the notions of *specification* and *theory* for the logic *dy*OSL.

Definition 3.25 An *object specification* is a pair $\langle\Sigma, \Gamma\rangle$ such that Σ is an object signature and Γ is again a pair $\langle\underline{\Gamma}, \overline{\Gamma}\rangle$ such that $\underline{\Gamma} \subseteq \underline{\mathcal{L}}$ and $\overline{\Gamma} \subseteq \overline{\mathcal{L}}$.

By splitting Γ into $\underline{\Gamma}$ and $\overline{\Gamma}$ we are able to distingusih between rigid axioms and non-rigid axioms, respectively.

Definition 3.26 An *object theory* is an object specification $\langle\Sigma, \Gamma\rangle$ such that $\Gamma^{\overline{\models}} = \Gamma$. $\square$

From now on we concentrate on a given object specification $spec = \langle\Sigma, \Gamma\rangle$.

Chapter 4

Translation of Language into Logic

In this chapter, we present the basic principles for translating object specifications with evolving behaviour into corresponding sets of *dy*OSL formulae. Due to the fact that the translation of the standard parts of object specifications, i.e. the specification of the rigid behaviour, into OSL is already described in detail in [Jungclaus, 1993], we here concentrate on the parts for specifying evolving behaviour.

We are going to show the translation of the language into the logic by example. For that we consider a modifiable timer.

Example 4.1 The specification of a modifiable timer is given in Fig. 4.1. The timer can be started by setting time and duration. The timer will then decrease the time until zero, then the alarm will be raised. The alarm will stop when the duration has been decreased to zero. For the action 'press_button' there is no effect specified as rigid axiom.

By means of the mutation actions 'm1' and 'm2' it is possible to change the behaviour of the timer. Each of these two mutation actions assign a certain effect to the 'press_button' action. 'm1' introduces an axiom that if 'press_button' occurs when the alarm is raised then the timer has to be restarted such that an alarm will be raised after 100 seconds (and this alarm will then last for 10 seconds). 'm2' adds an axiom describing that, if 'press_button' occurs when the timer has been started but the alarm has not yet be raised, then the raising of the alarm is delayed by 60 additional seconds. □

We now illustrate how to translate this specification into the logic. We will only sketch the main transformation steps. The first step is the extraction of the signature.

Example 4.2 *(Timer signature)* For the previous example, we get the following signature $\Sigma^{Timer} = \langle ACT, ATT, MUT, MAT\rangle$:

- $ACT_\epsilon = \{\perp, tic, press\_button\}$, $ACT_{nat\ nat} = \{start\}$;
- $ATT_{\epsilon,nat} = \{time, duration\}$, $ATT_{\epsilon,bool} = \{alarm\}$;

```
object Timer
 template
  attributes
    time:       nat   initialized  0,
    duration:   nat   initialized  0,
    alarm:      bool  initialized  false;
  actions
    start(time:nat,duration:nat),
    tic,
    press_button;
  rigid axioms
   start(t,d)
    changing  alarm     = false,
              time      = t,
              duration  = d;
   tic
    enabled   time≠ 0 or duration≠ 0,
    changing  if time ≥ 1  then time     = time-1,
              if time = 1  then alarm    = true,
              if time = 0  then duration = duration-1,
              if time = 0 and duration = 1 then alarm = false;
  axiom attributes
    Axioms  initialized {};
  mutators
    m1,
    m2,
    reset;
  dynamic specification
   m1
    changing  Axioms = Axioms +
                       { press_button
                            calling start(100,10) if alarm=true };
   m2
    changing  Axioms = Axioms +
                       { press_button
                            changing  if alarm=false
                                      then time=time+60 };
   reset
    changing  Axioms = {};
end object
```

Figure 4.1: Specification of a Timer.

- $MUT_\epsilon = \{\overline{\perp}, m1, m2, reset\}$;
- $MAT_{\epsilon,\mu} = \{\overline{occ}\}$, $MAT_{\epsilon,\sigma} = \{Ax\}$.

The *dy*OSL signature can be directly derived from the specification signature of a TROLL specification. The special signature elements $\underline{\perp}$, $\overline{\perp}$, $\overline{occ}$ and Ax have to be added for each specification. □

The next step is to extract the specification (note: we have to change the definition of specification, so that it includes the part corresponding to the rigid axioms)

Example 4.3 *(Timer specification)* For the previous example, we get the following specification $\langle\Sigma, \langle\underline{\Gamma}, \overline{\Gamma}\rangle\rangle$:

- $\underline{\Gamma}$ includes the following axioms:

  1. $time = 0 \wedge duration = 0 \wedge alarm$;
  2. $\mathsf{G}(\underline{occ} = tic \Rightarrow (time \neq 0 \vee duration \neq 0))$;
  3. $\mathsf{G}(\underline{occ} = start(t,d) \wedge \varphi^{time,duration,alarm}_{t,d,false} \Rightarrow (\mathsf{X}\,\varphi))$, for $\varphi \in \underline{st\mathcal{L}}$;
  4. $\mathsf{G}((\underline{occ} = tic \wedge time > 1 \wedge \varphi^{time}_{time-1}) \Rightarrow (\mathsf{X}\,\varphi))$, for $\varphi \in \underline{st\mathcal{L}}$;
  5. $\mathsf{G}((\underline{occ} = tic \wedge time = 1 \wedge \varphi^{time,alarm}_{time-1,true}) \Rightarrow (\mathsf{X}\,\varphi))$, for $\varphi \in \underline{st\mathcal{L}}$;
  6. $\mathsf{G}((\underline{occ} = tic \wedge time = 0 \wedge duration \neq 1 \wedge \varphi^{duration}_{duration-1}) \Rightarrow (\mathsf{X}\,\varphi))$, for $\varphi \in \underline{st\mathcal{L}}$;
  7. $\mathsf{G}((\underline{occ} = tic \wedge time = 0 \wedge duration = 1 \wedge \varphi^{duration,alarm}_{duration-1,false}) \Rightarrow (\mathsf{X}\,\varphi))$, for $\varphi \in \underline{st\mathcal{L}}$;

  Rule 1. results from the initialization of state attributes. Rule 2. is the translation of the enabling condition for the action tic. The remaining rules 3. to 7. model the effect of the actions on state formulae stating that valid state formulae are valid *after* the state change if we substitute in these formulae attribute names with the new attribute values (see [Sernadas et al., 1996] for more details).

- $\overline{\Gamma}$ includes the following axioms:

  1. $Ax = \{\}$;
  2. $\mathsf{G}(\overline{occ} = m1 \wedge \varphi^{Ax}_{Ax+\{\psi_1\}} \Rightarrow (\mathsf{X}\,\varphi))$ for $\varphi \in \overline{st\mathcal{L}}$ and ψ_1 is $(\star \Rightarrow ((\underline{occ} = press\_button \Rightarrow alarm = true)\,\mathsf{U}^?\,\star)) \wedge (\star \Rightarrow ((\underline{occ} = press\_button \Rightarrow \mathsf{X}(\underline{occ} = start(100, 10)))\,\mathsf{U}^?\,\star))$;
  3. $\mathsf{G}(\overline{occ} = m2 \wedge \varphi^{Ax}_{Ax+\{\psi_2\}} \Rightarrow (\mathsf{X}\,\varphi))$ for $\varphi \in \overline{st\mathcal{L}}$ and ψ_2 is $(\star \Rightarrow ((\underline{occ} = press\_button \Rightarrow alarm = false)\,\mathsf{U}^?\,\star)) \wedge (\star \Rightarrow ((\underline{occ} = press\_button \wedge \psi^{time}_{time+60} \Rightarrow (\mathsf{X}\,\psi))\,\mathsf{U}^?\,\star))$, where $\psi \in \underline{st\mathcal{L}}$;
  4. $\mathsf{G}((\overline{occ} = reset \wedge \varphi^{Ax}_{\{\}}) \Rightarrow (\mathsf{X}\,\varphi))$ for $\varphi \in \overline{st\mathcal{L}}$.

Again, rule 1. models the initialization of the meta attribute Ax. The remaining rules 2. to 4. model the valuation of the mutation actions in anlogy to the previous rules for the base level. □

Chapter 5

Using the Logical Framework

In this chapter we illustrate the use of the logic to prove some properties about *dy*OSL specifications (cf. [Sernadas et al., 1996] where a more detailed description of proofs can be found for the basic OSL).

5.1 A Hilbert calculus

We start by introducing a Hilbert calculus for the logic. We have to provide two consequence operators, one to reason at the base level and another to reason at the meta-level. These must be defined within the scope of a specification.

Definition 5.1 The *closure by consequence* of a specification $\langle\Sigma, \underline{\Gamma}, \overline{\Gamma}\rangle$ is the tuple $\langle\Sigma, \underline{\Gamma}^{\vdash}, \overline{\Gamma}^{\vdash}\rangle$ such that $\underline{\Gamma}^{\vdash}$ and $\overline{\Gamma}^{\vdash}$ are inductively defined as follows:

1. $\psi \in \underline{\Gamma}^{\vdash}$ provided that $\psi \in \underline{\Gamma}$;
2. $\psi \in \underline{\Gamma}^{\vdash}$ provided that $\psi \in Ax$;
3. $\psi \in \underline{\Gamma}^{\vdash}$ provided that ψ is an axiom of an anchored linear temporal, first order with equality calculus;
4. $\psi' \in \underline{\Gamma}^{\vdash}$ provided that ψ is a rigid data formula such that $\rho, dt \Vdash \psi$ for every assignment ρ;
5. $\psi' \in \underline{\Gamma}^{\vdash}$ provided that $\psi, (\psi \Rightarrow \psi') \in \underline{\Gamma}^{\vdash}$;
6. $\psi' \in \overline{\Gamma}^{\vdash}$ provided that $\psi, (\psi \Rightarrow \psi') \in \overline{\Gamma}^{\vdash}$;
7. $\varphi \in \underline{\Gamma}^{\vdash}$ provided that $\varphi \in \overline{\Gamma}^{\vdash}$, for each $\varphi \in \underline{\mathcal{L}}$;

8. $((z(x_1, \ldots, x_n) = z(x'_1, \ldots, x'_n)) \Rightarrow ((x_1 = x'_1) \wedge \ldots \wedge (x_n = x'_n))) \in \underline{\Gamma}^{\vdash}$
   provided that $z \in ACT_{s_1 \ldots s_n}$;

9. $((m(x_1, \ldots, x_n) = m(x'_1, \ldots, x'_n)) \Rightarrow ((x_1 = x'_1) \wedge \ldots \wedge (x_n = x'_n))) \in \overline{\Gamma}^{\vdash}$
   provided that $m \in MUT_{s_1 \ldots s_n}$;

10. $(\neg(z(x_1, \ldots, x_n) = z'(x'_1, \ldots, x'_{n'}))) \in \overline{\Gamma}^{\vdash}$
    provided that $z \in ACT_{s_1 \ldots s_n}$, $z' \in ACT_{s'_1 \ldots s'_{n'}}$ and $z \neq z'$;

11. $(\neg(m(x_1, \ldots, x_n) = m'(x'_1, \ldots, x'_{n'}))) \in \overline{\Gamma}^{\vdash}$
    provided that $m \in MUT_{s_1 \ldots s_n}$, $m' \in MUT_{s'_1 \ldots s'_{n'}}$ and $m \neq m'$;

12. $(\forall y(\bigvee_{s_1 \ldots s_n \in (\underline{S} \oplus \{\mu\})}(\bigvee_{z \in ACT_{s_1 \ldots s_n}}(\exists y_1 \ldots y_n \, (y = z(y_1, \ldots, y_n)) \ldots)))) \in \underline{\Gamma}^{\vdash}$;

13. $(\forall y(\bigvee_{s_1 \ldots s_n \in (\underline{S} \oplus \{\mu\})}(\bigvee_{m \in MUT_{s_1 \ldots s_n}}(\exists y_1 \ldots y_n \, (y = m(y_1, \ldots, y_n)) \ldots)))) \in \overline{\Gamma}^{\vdash}$;

14. $((\overline{occ} = m(t_1, \ldots, t_n) \wedge m(t_1, \ldots, t_n) \neq \overline{\bot}) \Leftrightarrow \star) \in \overline{\Gamma}^{\vdash}$;

15. $(\underline{occ} \neq \underline{\bot} \Rightarrow \overline{occ} = \overline{\bot}) \in \overline{\Gamma}^{\vdash}$;

16. $((\underline{occ} = \underline{\bot}) \Rightarrow ((a(x_1, \ldots, x_n) = y) \Rightarrow (\mathsf{X}(a(x_1, \ldots, x_n) = y)))) \in \underline{\Gamma}^{\vdash}$
    provided that $a \in ATT_{s_1 \ldots s_n, s}$ and $a \neq \underline{occ}$;

17. $((\overline{occ} = \overline{\bot}) \Rightarrow ((b(x_1, \ldots, x_n) = y) \Rightarrow (\mathsf{X}(b(x_1, \ldots, x_n) = y)))) \in \overline{\Gamma}^{\vdash}$
    provided that $b \in MAT_{s_1 \ldots s_n, s}$ and $b \neq \overline{occ}$. □

We briefly explain some of these rules (note that some of these are very similar to the ones presented in [Sernadas et al., 1996]). Rule 1 states that each axiom of the specification is a theorem: AX. Rule 3 captures the first-order linear temporal logic reasoning: LPO. Rules 5 and 6 are *modus ponens*: MP. Rule 7 states that the theorems of the base level are also theorems of the meta-level. Rules 8-13 reflect that actions (resp. mutation actions) are not interpreted and are generated: ACT (resp. MUT). Rule 14 reflects a property of $\star$ and mutations: $STAR$. Rules 15-17 reflect properties for the life-cycles: RUN.

The presented logical system is sound that is if $\varphi \in \Gamma^{\vdash}$ then $\Gamma \vDash \varphi$ where the latter means: an interpretation structure I satisfies φ whenever I satisfies all the formulae in Γ.

Definition 5.2 Let $spec = \langle \Sigma, \underline{\Gamma}, \overline{\Gamma} \rangle$ be a specification and $\Psi = \langle \underline{\Psi}, \overline{\Psi} \rangle$ such that $\underline{\Psi} \subseteq \underline{\mathcal{L}}$ and $\overline{\Psi} \subseteq \overline{\mathcal{L}}$. The *consequence of* Ψ on *spec* is the tuple $spec^{\Psi\vdash} = \langle \Sigma, \underline{\Gamma}^{\Psi\vdash}, \overline{\Gamma}^{\Psi\vdash} \rangle$ where $\underline{\Gamma}^{\Psi\vdash}$ and $\overline{\Gamma}^{\Psi\vdash}$ are inductively defined as follows:

- $\varphi \in \underline{\Gamma}^{\Psi\vdash}$ provided that $\varphi \in \underline{\Gamma}^{\vdash}$;

- $\varphi \in \underline{\Gamma}^{\Psi\vdash}$ provided that $\varphi \in \underline{\Psi}$;
- $\psi \in \underline{\Gamma}^{\Psi\vdash}$ provided that $(\varphi \Rightarrow \psi), \varphi \in \underline{\Gamma}^{\Psi\vdash}$;
- $\varphi \in \overline{\Gamma}^{\Psi\vdash}$ provided that $\varphi \in \overline{\Gamma}^{\vdash}$;
- $\varphi \in \overline{\Gamma}^{\Psi\vdash}$ provided that $\varphi \in \overline{\Psi}$;
- $\psi \in \overline{\Gamma}^{\Psi\vdash}$ provided that $(\varphi \Rightarrow \psi), \varphi \in \overline{\Gamma}^{\Psi\vdash}$. □

We denote $\varphi \in \underline{\Gamma}^{\Psi\vdash}$ (resp. $\varphi \in \overline{\Gamma}^{\Psi\vdash}$) by $\Psi\underline{\vdash}\varphi$ (resp. $\Psi\overline{\vdash}\varphi$).

5.2 An invariant calculus

In this section we present a calculus for proving invariants (cf. [Manna and Pnueli, 1992]).

Our aim is to give a proof-theoretic approach to certifying invariants for an evolving specification. Evolving sets of formulae are usually interpreted at runtime which is very flexible but does not allow for proving interesting global invariants, i.e. formulae that are to be satisfied all over.

We restrict ourselves to prove base invariants, i.e. invariants not containing meta formulae as subformulae. To prove invariants, we have to check the invariant against each *possible set of base axioms arising during specification evolution.* We call such formulae sets *completions.* A completion is a set of base-formulae, that completely specify the effects of actions, and which includes the set of rigid axioms. A completion is *reachable* if it can be constructed in the following way:

- For a given value of the meta attribute Ax, a completion is the union of following sets:
  - the rigid axioms $rigidaxioms$,
  - the values of Ax, and
  - a frame rule for each action act which has no explicit valuation rule in $rigidaxioms$ or Ax:

$$(\star \Rightarrow (((\underline{occ} = act \wedge \varphi) \Rightarrow (\mathsf{X}\,\varphi))\ \mathsf{U}^{?}\ \star))$$

- The value of Ax is the initial value or the result of an enabled sequence of mutation events.

The number of reachable completions is finite, if we use constant terms of type *spec* and set-operations on the Ax meta attribute only. For a proof, we have to consider all reachable completions.

Some invariants can even be proven based on the *rigidaxioms* only. In general, this is only possible for invariants containing no state-dependent attribute or in the situation where all actions are fully specified — for more interesting assertions, we have to use the presented method relying on completions.

Example 5.3 In the case of our example, from the specification of the *Timer*, we can "*derive*" the following completions:

- $C_1 = rigidaxioms +$
  $\{\ (\star \Rightarrow (((\underline{occ} = press\_button \wedge \varphi) \Rightarrow (\mathsf{X}\,\varphi))\ \mathsf{U}^?\ \star))\}$;
- $C_2 = rigidaxioms +$
  $\{\ (\star \Rightarrow (((\underline{occ} = press\_button \wedge alarm = true) \Rightarrow \mathsf{X}(\underline{occ} = start(100, 10)))\ \mathsf{U}^?\ \star)),$
  $(\star \Rightarrow (((\underline{occ} = press\_button \wedge \varphi \Rightarrow \mathsf{X}\,\varphi))\ \mathsf{U}^?\ \star))\ \}$;
- $C_3 = rigidaxioms +$
  $\{\ (\star \Rightarrow ((\underline{occ} = press\_button \wedge alarm = false \wedge \varphi^{time}_{time+60}) \Rightarrow (\mathsf{X}\,\varphi)))\ \mathsf{U}^?\ \star))\ \}$;
- $C_4 = rigidaxioms +$
  $\{\ (\star \Rightarrow (((\underline{occ} = press\_button \wedge alarm = true) \Rightarrow \mathsf{X}(\underline{occ} = start(100, 10)))\ \mathsf{U}^?\ \star)),$
  $(\star \Rightarrow ((\underline{occ} = press\_button \wedge alarm = false \wedge \varphi^{time}_{time+60}) \Rightarrow (\mathsf{X}\,\varphi)))\ \mathsf{U}^?\ \star))\ \}$;

The formula added in completion C_1 models the frame rule for the action $press\_button$: As long as nothing additional is stated in the meta part of the specification, this action has no effect on the object state. Completion C_2 includes this frame rule as well. For completion C_3 this frame rule is overruled by a rule changing all properties by modifying the time attribute. Completion C_4 includes all axioms added my the mutation actions 'm1' and 'm2'. In addition it contains the same frame rule as C_3 due to the fact that that frame rule overrules the one of C_2. □

In what follows, the results only hold when φ is a state formula (i.e., a formula without temporal operators and without the symbol $\underline{occ}$).

Definition 5.4 *(Global invariant rule)* Let $\langle \Sigma, \underline{\Gamma}, \overline{\Gamma} \rangle$ be a specification, and let $C_1, \ldots, C_n$ be its completions. Then:

- $(\mathsf{G}\,\varphi) \in \underline{\Gamma}^{\vdash}$ provided that:
  - $\underline{\vdash}\varphi$;

– $C_i \underline{\vdash}((\varphi \wedge \star) \Rightarrow (\varphi \, \mathsf{U}^? \star))$ for $i = 1, \ldots, n$. □

Beside the global invariant rule we also need the following local invariant rule.

Definition 5.5 *(Local invariant rule)* Let $\langle \Sigma, \underline{\Gamma}, \overline{\Gamma} \rangle$ be a specification, and let C be one of its completions. Then:

- $C \underline{\vdash}((\varphi \wedge \star) \Rightarrow (\varphi \, \mathsf{U}^? \star))$ provided that:
  – $C \underline{\vdash}((\varphi \wedge \star) \Rightarrow (((\underline{occ} = a \wedge \varphi) \Rightarrow (\mathsf{X}\, \varphi)) \, \mathsf{U}^? \star))$ for every $a \in ACT_w$; □

Note that in order to prove the soundness of this rule, we may use the following result: $(\star \wedge \varphi \Rightarrow (\mathsf{X}\, \varphi))$

Example 5.6 Assume that we want to prove the following property:

$$\underline{\vdash}(\mathsf{G}((time \neq 0) \Rightarrow (alarm = false)))$$

From example 5.3 we now that we have four possible completions, so we have to use rule 5.5 for each one of these completions. We only consider one of these. The others are similar. Let us consider C_3. Let α be $((time \neq 0) \Rightarrow (alarm = false))$.

(A) $\underline{\vdash}((time \neq 0) \Rightarrow (alarm = false))$

1.	$time = 0 \wedge duration = 0 \wedge alarm = true \in \underline{\Gamma}$	*AX*
2.	$(time \neq 0) \Rightarrow (alarm = false) \in \underline{\Gamma}^{\vdash}$	$LPO : 1$
3.	$\underline{\vdash}(time \neq 0) \Rightarrow (alarm = false)$	*by definition of derivation*

(B) $C_3 \underline{\vdash}((\alpha \wedge \star) \Rightarrow (((\underline{occ} = start(t,d) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \, \mathsf{U}^? \star))$

1.	$\mathsf{G}((\underline{occ} = start(t,d) \wedge (t \neq 0 \Rightarrow false = false)) \Rightarrow (\mathsf{X}\, \alpha)) \in \underline{\Gamma}$	*AX*
2.	$\mathsf{G}((\underline{occ} = start(t,d)) \Rightarrow (\mathsf{X}\, \alpha)) \in \underline{\Gamma}^{\vdash}$	$LPO : 1$
3.	$\mathsf{G}((\underline{occ} = start(t,d) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \in \underline{\Gamma}^{\vdash}$	$LPO : 2$
4.	$((\alpha \wedge \star) \Rightarrow (((\underline{occ} = start(t,d) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \, \mathsf{U}^? \star)) \in \underline{\Gamma}^{\vdash}$	$LPO : 4$
5.	$C_3 \underline{\vdash}((\alpha \wedge \star) \Rightarrow (((\underline{occ} = start(t,d) \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \, \mathsf{U}^? \star))$	*def of derivation*

(C) $C_3 \underline{\vdash}((\alpha \wedge \star) \Rightarrow (((\underline{occ} = press\_button \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \, \mathsf{U}^? \star))$

1.	$(\star \Rightarrow ((\underline{occ} = press\_button \wedge$ $((time + 60 \Rightarrow alarm = false) \Rightarrow (\mathsf{X}\, \alpha))) \, \mathsf{U}^? \star)) \in C_3$	*def of der.*
2.	$(\star \Rightarrow ((\underline{occ} = press\_button \Rightarrow alarm = false) \, \mathsf{U}^? \star)) \in C_3$	*def of der.*
3.	$((\alpha \wedge \star) \Rightarrow (((\underline{occ} = press\_button \wedge \alpha) \Rightarrow (\mathsf{X}\, \alpha)) \, \mathsf{U}^? \star)) \in C_3$	$LPO : 1, 2$

The rest of the proof is similar. The reader is referred to [Sernadas et al., 1996] for more detailed proofs of this sort. □

Chapter 6

Concluding Remarks

In this paper, we presented an extension of object specification for capturing evolving behaviour. For motivating the necessary of dealing with evolving behaviour we used an extended object specification language. Furthermore, we defined the logic *dy*OSL. Thereby, we provided a semantics based on an extended linear temporal logic. We discussed the translation of the specification language into that logic and, finally, we demonstrated the usage of the logical framework for proving object properties from their (dynamic) specifications.

The extensions we are considering form a stable basis for future work in which we aim at integrating additional aspects like normative specification (based on deontic logic). In [Saake et al., 1995a] we sketch a vision of a future specification language incorporating several specification concepts going beyond current object-oriented specification formalisms. The work presented here is a major step into that direction. This is due to the fact that we leave the level of first-order temporal logic by allowing the manipulation of first-order specification axioms within the specification. Thereby, we introduce some kind of reasoning capabilities into the objects.

We do not strive for a real higher-order approach. Nevertheless, we think that some higher-order aspects (as they have recently been investigated in several areas, for instance for algebraic specification in [Möller, 1994, Schobbens, 1994]) could help. Another approach using a 2-level logic is presented in [Gabbay and Hunter, 1993]. There, the second level in the logic is introduced for handling inconsistencies on the first level.

The presented proof-theoretic approach seems to be very restricted because we have to be able to compute all completions. However, we can see this situation from the opposite: If we can prove for each specification in isolation, that it satifies together with the rigid axioms the invariant, all of them can be savely added (by union) to Ax during object lifetime. This holds because of the construction of completions where the the number of 'frame rules' may only decrease if we add additional formulae to Ax.

Of course, there still is a large number of open questions due to the fact that the work presented here has to be considered as a starting point for further research. Among the issues for future work developing guidelines for specifying objects having an evolutionary behaviour is very important. We have to support specifiers in making their decisions which axioms should be rigid and which should be part of the dynamic specification part. Concerning the specification language syntactic restrictions must be provided for specifying behaviour evolution in a more controlled way. Furthermore, we have not dealt with schema evolution in this paper. It may happen that changing the dynamic behaviour of objects requires to introduce new attributes or to modify the types of existing attributes. For that, appropriate extensions of *dy*OSL are needed.

Bibliography

[Conrad, 1996] Conrad, S. (1996). A Basic Calculus for Verifying Properties of Interacting Objects. *Data & Knowledge Engineering*, 18(2):119–146.

[Conrad and Saake, 1995] Conrad, S. and Saake, G. (1995). Evolving Temporal Behaviour in Information Systems. In *HOA'95 — Higher-Order Algebra, Logic, and Term Rewriting (2nd Int. Workshop)*, pages PP7:1–16. Participant's Proceedings, Paderborn.

[Conrad and Saake, 1996] Conrad, S. and Saake, G. (1996). Specifying Evolving Temporal Behaviour. In Fiadeiro, J. and Schobbens, P.-Y., editors, *Proc. of the 2nd Workshop of the ModelAge Project (ModelAge'96), Sesimbra, Portugal*, pages 51–65. Departamento de Informatica, Universitade de Lisboa.

[Dubois et al., 1993] Dubois, E., Du Bois, P., and Petit, M. (1993). O-O Requirements Analysis: An Agent Perspective. In Nierstrasz, O., editor, *ECOOP'93 — Object-Oriented Programming, Proc. 7th European Conf., Kaiserslautern, Germany, July 1993*, pages 458–481. LNCS 707, Springer-Verlag, Berlin.

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

[Feenstra and Wieringa, 1993] Feenstra, R. B. and Wieringa, R. J. (1993). LCM 3.0: A Language for Describing Conceptual Models. Technical Report, Faculty of Mathematics and Computer Science, Vrije Universiteit Amsterdam.

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

[Gabbay and Hunter, 1993] Gabbay, D. and Hunter, A. (1993). Making Inconsistency Respectable: Part 2 – Meta-level handling of inconsistency. In Clarke, M., Kruse, R., and Moral, S., editors, *Proc. ECSQARU'93*, LNCS 747, pages 129–136. Springer-Verlag, Berlin.

[Gurevich, 1991] Gurevich, Y. (1991). Evolving Algebras: A Tutorial Introduction. *EATCS Bulletin*, 43:264–284.

[Gurevich, 1993] Gurevich, Y. (1993). Logic in Computer Science. In Rozenberg, G. and Salomaa, A., editors, *Current Trends in Theoretical Computer Science*, number 40 in Series in Computer Science, pages 223–394. World Scientific Press.

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

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

[Khosla and Maibaum, 1989] Khosla, S. and Maibaum, T. (1989). The Prescription and Description of State Based Systems. In Banieqbal, B., Barringer, H., and Pnueli, A., editors, *Temporal Logic in Specification*, pages 243–294. LNCS 398, Springer-Verlag.

[Lipeck and Thalheim, 1993] Lipeck, U. and Thalheim, B., editors (1993). *Modelling Database Dynamics: Selected Papers from the 4th Int. Workshop on Foundations of Models and Languages for Data and Objects.* Workshops in Computing. Springer-Verlag, London.

[Manna and Pnueli, 1992] Manna, Z. and Pnueli, A. (1992). *The Temporal Logic of Reactive and Concurrent Systems. Vol. 1: Specification.* Springer-Verlag, New York.

[Meseguer, 1992] Meseguer, J. (1992). Conditional Rewriting as a Unified Model of Concurrency. *Theoretical Computer Science*, 96(1):73–156.

[Meseguer, 1993] Meseguer, J. (1993). A Logical Theory of Concurrent Objects and Its Realization in the Maude Language. In Agha, G., Wegener, P., and Yonezawa, A., editors, *Research Directions in Object-Oriented Programming*, pages 314–390. MIT Press.

[Meseguer and Qian, 1993] Meseguer, J. and Qian, X. (1993). Logic-Based Modeling of Dynamic Object Systems. In Buneman, P. and Jajodia, S., editors, *Proc. of the 1993 ACM SIGMOD Int. Conf. on Management of Data, Washington, D.C., SIGMOD RECORD 22(2)*, pages 89–98. ACM Press.

[Möller, 1994] Möller, B. (1994). Ordered and Continuous Models of Higher-Order Specifications. In Heering, J., Meinke, K., Möller, B., and Nipkow, T., editors, *Proc. Int Workshop on Higher-Order Algebra, Logic, and Term Rewriting (HOA'93)*, pages 223–255. Springer LNCS 816.

[Saake et al., 1995a] Saake, G., Conrad, S., and Türker, C. (1995a). From Object Specification towards Agent Design. In Papazoglou, M., editor, *OOER'95: Object-Oriented and Entity-Relationship Modeling, Proc. of the 14th Int. Conf., Gold Coast, Australia*, pages 329–340. LNCS 1021, Springer-Verlag, Berlin.

[Saake et al., 1995b] Saake, G., Sernadas, A., and Sernadas, C. (1995b). Evolving Object Specifications. In Wieringa, R. and Feenstra, R., editors, *Information Systems — Correctness and Reusability. Selected Papers from the IS-CORE Workshop*, pages 84–99. World Scientific Publishing, Singapore.

[Schobbens, 1994] Schobbens, P.-Y. (1994). Extensions of Initial Models and their Second-order Proof Systems. In Heering, J., Meinke, K., Möller, B., and Nipkow, T., editors, *Proc. Int Workshop on Higher-Order Algebra, Logic, and Term Rewriting (HOA'93)*, pages 326–344. Springer LNCS 816.

[Sernadas and Ramos, 1994] Sernadas, A. and Ramos, J. (1994). The GNOME Language: Syntax, Semantics and Calculus. Research report, Instituto Superior Tecnico.

[Sernadas et al., 1995] Sernadas, A., Sernadas, C., and Costa, J. (1995). Object Specification Logic. *Journal of Logic and Computation*, 5(5):603–630.

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

[Sernadas et al., 1991a] Sernadas, A., Sernadas, C., Gouveia, P., Resende, P., and Gouveia, J. (1991a). OBLOG — Object-Oriented Logic: An Informal Introduction. Technical Report, INESC, Lisbon.

[Sernadas et al., 1996] Sernadas, A., Sernadas, C., and Ramos, J. (1996). A Temporal Logic Approach to Object Certification. *Data & Knowledge Engineering*, 19:267–294.

[Sernadas et al., 1991b] Sernadas, C., Resende, P., Gouveia, P., and Sernadas, A. (1991b). In the Large Object-Oriented Design of Information Systems. In Van Assche, F., Moulin, B., and Rolland, C., editors, *The Object-Oriented Approach in Information Systems*, pages 209–232. North Holland, Amsterdam.

[Tansel et al., 1993] Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., and Snodgrass, R., editors (1993). *Temporal Databases: Theory, Design, and Implementation.* Benjamin/Cummings.

[Türker et al., 1996] Türker, C., Conrad, S., and Saake, G. (1996). Dynamically Changing Behavior: An Agent-Oriented View to Modeling Intelligent Information Systems. In Raś, Z. W. and Michalewicz, M., editors, *Foundations of Intelligent Systems, Proc. of the 9th Int. Symposium on Methodologies for Intelligent Systems, ISMIS'96, Zakopane, Poland*, pages 572–581. LNAI 1079, Springer-Verlag, Berlin.

[Wieringa, 1991] Wieringa, R. J. (1991). A Formalization of Objects Using Equational Dynamic Logic. In Delobel, C., Kifer, M., and Masunaga, Y., editors, *Deductive and*

Object Oriented Databases, Proc. of the 2nd Int. Conf., DOOD'91, Munich, Germany, December 1991, pages 431–452. LNCS 566, Springer-Verlag, Berlin.