Local Specification of Distributed Families of Sequential Objects*

Hans-Dieter Ehrich[1] and Amílcar Sernadas[2]

[1] Abteilung Datenbanken, Technische Universität
Postfach 3329, D-38023 Braunschweig, Germany
[2] Departamento de Matemática, Instituto Superior Técnico
Av. Rovisco Pais, P-1096 Lisboa Codex, Portugal

Abstract. Fully concurrent models of distributed object systems are specified using linear temporal logic that does not per se cope with concurrency. This is achieved by employing the principle of local sequentiality: we specify from local viewpoints assuming that there is no intra-object concurrency but full inter-object concurrency. Local formulae are labelled by identity terms. For interaction, objects may refer to actions of other objects, e.g., calling them to happen synchronously. A locality predicate allows for making local statements about other objects. The interpretation structures are global webs of local life cycles, glued together at shared communication events. These interpretation structures are embedded in an interpretation frame that is a labelled locally sequential event structure. Two initiality results are presented: the category of labelled locally sequential event structures has initial elements, and so has the full subcategory of those satisfying given temporal axioms. As in abstract data type theory, these initial elements are obvious candidates for assigning standard semantics to signatures and specifications.

1 Introduction

In abstract data type theory, higher-order model classes like isomorphism classes of many-sorted algebras are specified with (conditional) equational logic that does not per se allow for specifying such classes. The trick is to employ some general higher-order principle to specifiable classes. A popular principle of this kind is initiality, i.e., restriction to initial models.

This paper suggests an analogous trick for distributed systems specification. The higher-order principle is *local sequentiality*. Employing local sequentiality, we can specify fully concurrent models of distributed computation using a logic that does not per se cope with concurrency.

* This work was partly supported by the EU under ESPRIT-III BRA WG 6071 IS-CORE (Information Systems - COrrectness and REusability) and BRA WG 6112 COMPASS (COMPrehensive Algebraic approach to System Specification and development) and by ESDI under research contract OBLOG (OBject LOGic). To appear in *Recent Trends in Data Type Specification*, E. Astesiano, G. Reggio and A. Tarlecki, eds. Springer LNCS 1995

Local sequentiality means that we distinguish between local objects and global families of objects, making the general assumption that there is no intra-object concurrency but full inter-object concurrency.

Objects are locally specified in a sequential process logic, linear temporal logic in our case. Local formulae are labelled by identity terms. Interaction is specified by locally referring to actions of other objects, e.g., calling them to happen synchronously. For making statements about other objects, a locality predicate is introduced.

The interpretation structures for linear temporal logic are life cycles, i.e., linear causality chains of events. Accordingly, our interpretation structures for local specification are global webs of local life cycles, glued together at shared communication events. These interpretation structures are embedded in an interpretation frame that is a labelled locally sequential event structure. Interpretation frames are models of processes.

This extends the simple models we used in our previous work where interpretations were (sets of) life cycles [SEC90, EGS91] or menues [ES91]. The life cycle model has been used as a semantic basis for object specification languages OBLOG [SSE87, SSG+91], TROLL [JSHS91, SJE92, Ju93, HSJHK94], TROLL *light* [CGH92, GCH93], and GNOME [SR94].

Our work has been greatly influenced by related work on specification languages and theoretical foundations. In a sense, we integrate work on algebraic specification of data types [EGL89, EM85, EM90] and databases [Eh86, EDG88], process specification [Ho85, Mi89], the specification of reactive systems [Se80, MP89, Sa91a], conceptual modeling [Ch76, Bo85, EGH+92, SF91, SJH93] and knowledge representation [ST89, MB89].

Approaches to logic and algebraic foundations of object-orientation and concurrency have given essential input to the work reported here. The results in [FSMS91, FM91, FM92, SSC92] have been influential. FOOPS [GM87, GW90] has provided insights in the algebraic nature of objects. Algebraic approaches to concurrency are given in [AR92, MM93, Br93].

The local specification logic and interpretation structures put forward in this paper are influenced by the n-agent logic in [LMRT91], but we deviate from that logic in essential respects: we have more elementary temporal operators, and our interpretations are quite different.

In a companion paper [ESSS95], the foundations outlined here are applied to the semantic description of an abstract object specification language.

2 Object signatures and interpretations

2.1 Data signatures

We assume that the reader is familiar with data signatures and their algebraic interpretations, but we briefly introduce our notation and terminology.

A *data signature* is a pair $\Sigma_D = (S_D, \boldsymbol{\Omega}_D)$ where S_D is a set of *data sorts*, and $\boldsymbol{\Omega}_D = \{\Omega_{x,s}\}_{x \in S_D^*, s \in S_D}$ is an $S_D^* \times S_D$-indexed family of sets of *data operation* *symbols*. Given an S_D-indexed set $\boldsymbol{X} = \{X_s\}_{s \in S_D}$ of variable symbols, the *Σ_D-terms over $\boldsymbol{X}$* are denoted by $\boldsymbol{T}_{\Sigma_D}(\boldsymbol{X})$.

If $x = s_1 \ldots s_n$, we write $\omega : s_1 \times \ldots \times s_n \to s$ or $\omega : x \to s\,[\in \boldsymbol{\Omega}]$ for $\omega \in \Omega_{s_1 \ldots s_n, s}$.

An *interpretation* of Σ_D is a Σ_D-algebra $\boldsymbol{U}$ with carrier sets s_U for each $s \in S_D$, and operations $\omega_U : x_U \to s_U$ for each operator $\omega : x \to s \in \boldsymbol{\Omega}$. If $x = s_1 \ldots s_n \in S_D^*$, then $x_U = s_{1U} \times \ldots \times s_{nU}$ is the cartesian product. The interpretation of a term $t \in \boldsymbol{T}_{\Sigma_D}(\boldsymbol{X})$ in $\boldsymbol{U}$ with a given variable assignment θ is denoted by t_U^θ.

The class of all Σ_D-algebras is structured by *Σ_D-algebra morphisms*. A Σ_D-algebra morphism $\boldsymbol{h} : \boldsymbol{U} \to \boldsymbol{V}$ is a family of maps $\boldsymbol{h} = \{h_s : s_U \to s_V\}_{s \in S_D}$ such that, for each operator $\omega : x \to s \in \boldsymbol{\Omega}$ and each element $\boldsymbol{a} \in x_U$, we have $h_s(\omega_U(\boldsymbol{a})) = \omega_V(\boldsymbol{h}_x(\boldsymbol{a}))$.

The nice properties of the category Σ_D**-alg** of Σ_D-algebras and Σ_D-algebra morphisms are well known and have been utilized for elegant semantic constructions for abstract data type specifications [EGL89, EM85, EM90].

2.2 Class and instance signatures

Classes in the sense of what follows are *object* classes, not classes in the sense of set theory.

Definition 1. Let Σ_D be a data signature. A *class signature over Σ_D* is a triple $\Sigma_C = (S_O, I, A)$ where S_O is a set of *object sorts*, $I = \{I_{x,b}\}_{x \in S_{DO}^*, b \in S_O}$ is an $S_{DO}^* \times S_O$-indexed set family of *instance operators*, and $A = \{A_{x,b}\}_{x \in S_{DO}^*, b \in S_O}$ is an $S_{DO}^* \times S_O$-indexed set family of *action operators*. Here, $S_{DO} = S_D \cup S_O$.

Like for data operators, we use the notation $i : x \to b$ for $i \in I_{x,b}$ and $a : x \to b$ for $a \in A_{x,b}$.

Example 1. We give a signature for a class of flip-flops using an intuitive ad-hoc notation that easily translates to our formalism. The specification says that flip-flops can be created, set, reset and destroyed. Moreover, we have an infinite set of flip-flop identities: `F1`, `F2`, `R(`n`)` for all natural numbers n, and a "next" flip-flop `N(`f`)` for every flip-flop f. Each flip-flop identity is associated with a flip-flop instance. This does not necessarily mean that the system has infinitely many flip-flops, an instance may have several "alias" names. This aspect is not persued in the present paper.

```
class flip-flop;
   object sort FF;
   data sort nat;
   actions create, set, reset, destroy;
   instances F1, F2, R : nat, N : FF;
end flip-flop.
```

The notation is intended to mean the following class signature. The underlying data signature is assumed to contain the sort `nat` of natural numbers.

$S_O = \{\mathtt{FF}\}$

$I = \{\mathtt{F1} : \to \mathtt{FF},\ \mathtt{F2} : \to \mathtt{FF},\ \mathtt{R} : \mathtt{nat} \to \mathtt{FF},\ \mathtt{N} : \mathtt{FF} \to \mathtt{FF}\}$

$A = \{\mathtt{create} : \to \mathtt{FF},\ \mathtt{destroy} : \to \mathtt{FF},\ \mathtt{set} : \to \mathtt{FF},\ \mathtt{reset} : \to \mathtt{FF}\}$

There are infinitely many flip-flop identities, e.g., `F1`, `F2`, `R(0)`, `R(1)`, ... , `N(F1)`, `N(F2)`, `N(R(0))`, `N(N(R(0)))`, etc.

flip-flops can be created, set, reset, and destroyed. The "value" of a flip-flop is represented by the fact that `set` or `reset`, respectively, is *enabled* (cf. section 3). □

The interpretation of a class signature Σ_C is indirectly given by (1) extending the underlying data signature Σ_D to cover the identities and actions specified in the class signature, and (2) deriving an instance signature Σ_I for the identities and individual action alphabets of all object instances.

The data signature extension is defined as follows. Each object sort b goes with the two data sorts of object identities b^i and object actions b^a, respectively. Thus, the object sorts S_O give rise to two sets of data sorts called S_O^i and S_O^a. Let $S^i = S_D \cup S_O^i$ and $S^a = S_D \cup S_O^a$. For $x = s_1 \ldots s_n \in S_{DO}^*$, we define $x^i = s_1^i \ldots s_n^i \in S^{i*}$ where, for $j \in \{1, \ldots, n\}$, $s_j^i = s_j$ if $s_j \in S_D$. The notation x^a is defined correspondingly.

Definition 2. Given a data signature $\Sigma_D = (S_D, \boldsymbol{\Omega}_D)$ and a class signature $\Sigma_C = (S_O, I, A)$ over Σ_D, the *extended data signature* $\Sigma = (S, \boldsymbol{\Omega})$ is given by

$$S = S_D \cup S_O^a \cup S_O^i$$

where $S_O^a = \{b^a \mid b \in S_O\} \cup \{ac\}$ and $S_O^i = \{b^i \mid b \in S_O\} \cup \{id\}$, and

$$\boldsymbol{\Omega} = \boldsymbol{\Omega}_D \cup \boldsymbol{\Omega}_O^a \cup \boldsymbol{\Omega}_O^i \cup \boldsymbol{\Omega}_O^{ai}$$

where, for every object sort $b \in S_O$ and every $x \in S_{DO}^*$, we have $\Omega_{O;b^i x^a, b^a}^a = A_{x,b}$, $\Omega_{O;x^i,b^i}^i = I_{x,b}$, $\Omega_{O;b^a p^i, bool}^{ai} = \{ai_{bp}\}$. The other sets in the families are empty. Moreover, for every object sort $b \in S_O$, we assume $b^i \leq id$ and $b^a \leq ac$.

Before we explain the transformation, we illustrate it by the `flip-flop` example.

Example 2. The `flip-flop` class signature in example 1 transforms to the following data signature extension.

sorts $\mathtt{FF}^i \leq \mathtt{id}$, $\mathtt{FF}^a \leq \mathtt{ac}$

ops $\mathtt{F1},\ \mathtt{F2} : \to \mathtt{FF}^i$

$\mathtt{R} : \mathtt{nat} \to \mathtt{FF}^i$

$\mathtt{N} : \mathtt{FF}^i \to \mathtt{FF}^i$

$\mathtt{create},\ \mathtt{set},\ \mathtt{reset},\ \mathtt{destroy} : \mathtt{FF}^i \to \mathtt{FF}^a$

$\mathtt{ai}_{\mathtt{FF},\mathtt{FF}} : \mathtt{FF}^a \times \mathtt{FF}^i \to \mathtt{bool}$ □

The sort id is to be interpreted by all identities in the system. Because of the standard unique naming assumption in object-oriented systems, this should be the disjoint union of the interpretations of the sorts b^i for $b \in S_O$. Using order sorting, we have $b^i \leq id$ for every object sort $b \in S_O$.

The sort ac is to be interpreted by all actions in the system, i.e., by the union of the interpretations of the sorts b^a for $b \in S_O$. We do not require disjointness here, we will allow for overlap expressing that actions are shared among objects. But still, using order sorting, we have $b^a \leq id$ for every object sort $b \in S_O$.

The data identity operators $i : x^i \to b^i \in \boldsymbol{\Omega}_O^i$ are derived from the object identity operators $i : x \to b \in I$. They may be parameterized over data, including identities but not actions. So we can cope with objects identified by other objects.

The data action operators $a : b^i \times x^a \to b^a \in \boldsymbol{\Omega}_O^a$ are derived from the object action operators $a : x \to b \in A$. They may have actions ("methods") as parameters. Implicitly, this allows for identity parameters as well since each action carries the identity of its object with it. Data action operators are associated with object instances, so we have identities of sort b as additional parameter. If $a(i, t_1, \ldots, t_n) \in T_\Sigma(\boldsymbol{X})_{b^a}$, we call i the *identity of action* $a(i, t_1, \ldots, t_n)$.

For any object sorts $b, p \in S_O$, the id-of-action operators $ai_{bp} : b^a \times p^i \to \mathit{bool}$ are to be interpreted by relations associating with each individual action the identities of its objects. Of course, the equation $ai_{bb}(a(u, y), u) = \mathit{true}$ should hold for every $a : b^i \times x^a \to b^a \in \boldsymbol{\Omega}_O^a$. Moreover, if the occurrence of action a' of sort $i' : b'$ implies the occurrence of action a'' of sort $i'' : b''$ (e.g., via synchronous action calling), then a' should also belong to the actions of i'', i.e., $ai_{b'b''}(a', i'') = \mathit{true}$. The idea is that $ai_{bp}(a, i) = \mathit{true}$ should hold whenever action a affects object i.

Definition 3. An *instance signature* is a pair $\Sigma_I = (Id, \boldsymbol{Ac})$ where Id is a set of *identities*, and $\boldsymbol{Ac} = \{Ac_i\}_{i \in Id}$ is an Id-indexed family of *action alphabets*.

An instance signature is a distributed action alphabet, providing a local action alphabet for each object. Communication is established by actions shared among two or more objects.

Definition 4. The *partners* of an action α are $P(\alpha) = \{i \in Id \mid \alpha \in Ac_i\}$.

Let $\Sigma_C = (S_O, I, A)$ be a class signature over the data signature $\Sigma_D = (S_D, \boldsymbol{\Omega}_D)$. Let $\boldsymbol{U}$ be an interpretation of the extended data signature Σ.

Definition 5. The *instance signature* determined by Σ and $\boldsymbol{U}$ is $\Sigma_I = (Id, \boldsymbol{Ac})$ where $Id = id_U$ is the global set of identities, and $\boldsymbol{Ac} = \{Ac_i\}_{i \in Id}$ is the Id-indexed family of action alphabets $Ac_i = \{\alpha \in ac_U \mid ai_{qb}(\alpha, i) = \mathit{true}$ for some sort $q \in S_O$ and the sort b of $i\}$.

2.3 An event-based interpretation of instance signatures

The interpretations of instance signatures are concurrent processes synchronized via shared events, i.e., action occurrences.

Suitable interpretation structures can be based on any distributed process model, for instance Petri nets or distributed transition systems. But the interpretation structures should fit to the specification logic. Choosing the latter suggests the appropriate process model.

As an example, for simplicity, and because of its proven practicality [MP89, Se80, Sa91a, Sa91b], we use linear temporal logic for specifying the local properties of individual objects. The suitable interpretation structures for linear temporal logic are linear traces of events, also called life cycles [SEC90, EGS91].

There is an obvious distributed model based on individual life cycles and event sharing, namely global webs of life cycles glued together at shared communication events. This is the model we adopt here. A frame for this model is an event structure that is locally sequential.

Event structures were introduced by Winskel [Wi80]. A recent survey of models for concurrency including event structures is [WN93]. We briefly give the definition and introduce our notation. Our main deviation from standard notation is that we use $\rightarrow^*$ for causality instead of the usual $\leq$ because we use the latter for order sorting, and we want to use event structures where causality is the reflexive and transitive closure of a base relation $\rightarrow$ of "step" causality. The following definition is taken from [WN93].

Definition 6. A (discrete prime) *event structure* is a triple $E = (Ev, \rightarrow^*, \#)$ where Ev is a set of events and $\rightarrow^*, \# \subseteq Ev \times Ev$ are binary relations called *causality* and *conflict*, respectively. Causality $\rightarrow^*$ is a partial ordering, and conflict $\#$ is symmetric and irreflexive. For each event $e \in E$, its *local configuration* $\downarrow e = \{e' \mid e' \rightarrow^* e\}$ is finite. Conflict propagates over causality, i.e., $e \# e' \rightarrow^* e'' \Rightarrow e \# e''$ for all $e, e', e'' \in Ev$. Two events $e, e' \in Ev$ are *concurrent*, $e\, co\, e'$ iff $\neg(e \rightarrow^* e' \vee e' \rightarrow^* e \vee e \# e')$.

In the sequel, the order-theoretic notions refer to causality.

Definition 7. Let E be an event structure. A *life cycle* in E is a maximal totally ordered sub-event structure of E.

Definition 8. A *sequential* event structure E is an event structure in which (1) there is a unique minimal element $\varepsilon \in Ev$, (2) every local configuration $\downarrow e$ is totally ordered, and (3) causally independent events are always in conflict, i.e., for all events $e, f \in Ev$, we have $e \# f$ iff neither $e \rightarrow^* f$ nor $f \rightarrow^* e$ holds. The events $Ev_+ = Ev - \{\varepsilon\}$ are called the *proper* events.

Intuitively, the causally minimal event represents an imaginary "initial event" where no action occurred so far. This represents the "pre-natal" state of an object where nothing happened yet (not even a "birth" action).

With respect to causality, a sequential event structure E is a rooted tree. It represents a set of life cycles grouped together by equal prefixes. A single life cycle is also a sequential event structure.

Sequential event structures are our model of objects. There is no intra-object concurrency. Thus, conflict is a derived concept. We omit it from notation. Assuming that causality $\rightarrow^*$ is the reflexive and transitive closure of a base relation $\rightarrow$ of "step" causality, we arrive at the notation

$$E = (Ev, \rightarrow)$$

for sequential event structures. We will extend this notation to any event structure where conflict is determined from causality by a general assumption. Actually, this holds for all event structures we consider here.

The interpretation structures we envisage for a given instance signature Σ_I are labelled locally sequential Σ_I-event structures. This is our model for fully concurrent families of sequential objects. For modelling communication, the local event sets Ev_i may overlap: an event $e \in Ev_i \cap Ev_j \cap \ldots$ is shared by objects $i, j, \ldots \in Id$.

We make these notions precise, but first we need some notation. Given a set I and an I-indexed family $\boldsymbol{M} = \{M_i\}_{i \in I}$ of sets, we denote their union by $\bigcup \boldsymbol{M} = \bigcup_{i \in I} M_i$. If $\boldsymbol{E} = \{E_i\}_{i \in I}$ is a family of event structures, we denote their union by $\bigcup \boldsymbol{E} = (\bigcup_{i \in I} Ev_i, \bigcup_{i \in I} \rightarrow_i)$.

The notation is justified by the assumption that global conflicts are only those inherited by local conflicts: for all events $e, f \in E$, we have $e \# f$ iff, for some object i and some events $e', f' \in Ev_i$, we have a local conflict $e' \#_i f'$ while $e' \rightarrow^* e$ and $f' \rightarrow^* f$. That is, we globally assume $e \, co \, f$ as a default.

Definition 9. Let $\Sigma_I = (Id, \boldsymbol{Ac})$ be an instance signature. A *Σ_I-event structure* E is the union $\bigcup \boldsymbol{E}$ of an Id-indexed family $\boldsymbol{E} = \{E_i\}_{i \in I}$ of sequential event structures where $E_i = (Ev_i, \rightarrow_i)$. The set family of proper events is denoted by $\boldsymbol{Ev}_+ = \{Ev_{i+}\}_{i \in Id}$, and $Ev_+ = \bigcup \boldsymbol{Ev}_+$.

That is, Σ_I-event structures are *locally sequential.* This model is similar to the n-agent model described in [LMRT91].

Definition 10. Given a Σ_I-event structure E, a *distributed life cycle* $L = (Lc, \rightarrow)$ *in* E is an event structure $L \subseteq E$ that is the union of a family $\boldsymbol{L} = \{L_i\}_{i \in Id} \subseteq \boldsymbol{E}$ of life cycles in E, i.e., $L_i \subseteq E_i$ for every $i \in Id$.

A distributed life cycle is a system of life cycles for the individual objects, glued together at shared "communication" events.

Our interpretation structures are *labelled* distributed life cycles within labelled Σ_I-event structures, so we have to introduce labelling.

Definition 11. Let $E = (Ev, \rightarrow)$ be a Σ_I-event structure. A *labelling for* E is a map $\bar{\alpha} : Ev_+ \rightarrow \bigcup \boldsymbol{Ac}$ that is the union $\bar{\alpha} = \bigcup \boldsymbol{\alpha}$ of a family of maps $\boldsymbol{\alpha} = \{\alpha_i : Ev_{i+} \rightarrow Ac_i\}_{i \in Id}$ satisfying the following condition: for all events $e', e'' \in Ev$, we have $\bar{\alpha}(e') \neq \bar{\alpha}(e'')$ whenever there is an event $e \in Ev$ such that $e \rightarrow e'$ and $e \rightarrow e''$.

Intuitively, each proper event is the occurrence of an action, and the label is supposed to be this action. The imaginary start events ε_i do not have labels. The labels of the immediate successors of an event e are the actions *enabled* at e (cf. definition 21). These must be distinct for different successor events.

Definition 12. An *interpretation frame* for a given instance signature Σ_I is a labelled Σ_I-event structure $\bar{E} = (E, \bar{\alpha})$. An *interpretation structure* within $\bar{E}$ is a labelled distributed life cycle $\bar{L} = (L, \bar{\alpha}\mid_L)$ where $L \subseteq E$.

Now we define one particular interpretation frame determined by a given instance signature Σ_I. The idea is obvious: the events are defined to be all possible occurrences of actions.

However, not every combination of local configurations is a meaningful context for an action to occur. For instance, if I invite you to meet, then both of us must "remember" that invitation, otherwise the meeting cannot take place. More precisely, when we meet, the two of us must not be in local configurations where you are sure I invited you, and I am sure I never did.

The relevant concept is that of a consistent configuration for an action in which it can possibly occur. In such a configuration, any two objects must agree on their past communication. Generalizing the notation for local configurations $\downarrow e$ to sets, we define $\downarrow C = \{e' \mid \exists e \in C : e' \to^* e\}$ for any subset $C \subseteq Ev$ of events.

Definition 13. Given a Σ_I-event structure $E = (Ev, \to)$, a *configuration in* E is a set of events $Cf \subseteq Ev$ with the properties (1) $Cf = \downarrow Cf$ and (2) Cf is conflict-free, i.e., $(Cf \times Cf) \cap \# = \emptyset$. If $I \subseteq Id$ is a set of object identities, then a *configuration for* I in E is a configuration that is the range $\gamma(I)$ of a map $\gamma : I \to Ev$ such that $\gamma(i) \in Ev_i$ for every $i \in I$.

A configuration Cf for I in E contains one event $e_i = \gamma(i) \in Ev_i$ for every $i \in I$. These events e_i need not be distinct. Two different objects $i, j \in I$ may share an event in Cf, i.e., we may have $e_i \in Ev_j$ as well. In this case, e_i and e_j must be causally related because otherwise they would be in conflict.

Definition 14. Given an instance signature $\Sigma_I = (Id, \boldsymbol{Ac})$, the *interpretation frame* $\bar{E}(\Sigma_I) = (E(\Sigma_I), \bar{\alpha})$ is inductively defined as follows. $E(\Sigma_I) = \bigcup \boldsymbol{E}(\Sigma_I)$ where $\boldsymbol{E}(\Sigma_I) = \{E_i\}_{i \in Id}$ where $E_i = (Ev_i, \to_i)$, for each object $i \in Id$.

(1) $\varepsilon_i \in Ev_i$ for every object $i \in Id$.
(2) if α is an action and Cf is a configuration for α's partners $P(\alpha)$, then $Cf\,\alpha \in Ev_i$ for every partner $i \in P(\alpha)$; as for labelling and causality: we have $\bar{\alpha}(Cf\,\alpha) = \alpha$, and $e \to_i Cf\,\alpha$ for every $e \in Cf \cap Ev_i$ and every $i \in P(\alpha)$.

The basis of this inductive definition is given by birth events that happen in configurations consisting of one or more "initial events" ε_i. The event $Cf\,\alpha$ represents α occurring in the configuration Cf. This event is shared by the partners of α.

We prove that the construction is sound.

Theorem 15. *For each instance signature Σ_I, $\bar{E}(\Sigma_I)$ is an interpretation frame for Σ_I.*

Proof: We have to show that every local event structure $E_i = (Ev_i, \rightarrow_i)$, $i \in Id$, is sequential, and that the labels are ok, i.e., immediate successors have different labels.

We prove the first by induction over the structure of $\bar{E}(\Sigma_I)$.

Let $i \in Id$ be an identity, and let $e \in E_i$. If $e = \varepsilon_i$, then $\downarrow_i e = \{\varepsilon_i\}$ is trivially totally ordered, where $\downarrow_i e = \downarrow e \cap E_i$. Otherwise, we have $e \in Cf\,\alpha$ for some action $\alpha \in Ac_i$ and some configuration Cf for $P(\alpha)$. Assume that every life cycle prefix so far, i.e., every local configuration $\downarrow_j f$ for $f \in Cf$ and $j \in P(\alpha)$, is totally ordered. Then we have to show that $\downarrow_j Cf\,\alpha$ is totally ordered as well for every $j \in P(\alpha)$.

Assume that, for some partner $j \in P(\alpha)$, $g, g' \in \downarrow_j Cf$. Since $\downarrow_j Cf$ is a linear trace, g and g' are causally related in E_j, say $g \rightarrow_j^* g'$. Since both are causal for $Cf\,\alpha$, $\downarrow_j Cf\,\alpha$ is totally ordered as well.

As for labelling, we have to show that labels of events with a common immediate predecessor are different. But this is obvious from construction since the events are *defined* by the actions applied to the configurations representing the current states of the action's partners. □

Labelled Σ_I-event structures are related by *morphisms*. Adapting general event structure morphisms from [WN93] to our structures, we arrive at the following definition. Let $\Sigma_I = (Id, \boldsymbol{Ac})$ be an instance signature.

Definition 16. Let $E_1 = (Ev_1, \rightarrow_1)$ and $E_2 = (Ev_2, \rightarrow_2)$ be two Σ_I-event structures. A *Σ_I-event structure morphism* $h : E_1 \rightarrow E_2$ is the union $\bigcup \boldsymbol{h}$ of an Id-indexed family of partial surjective maps $\boldsymbol{h} = \{h_i : Ev_{1i} \cdots\!\twoheadrightarrow Ev_{2i}\}_{i \in Id}$ such that $h : Ev_1 \cdots\!\twoheadrightarrow Ev_2$ is a partial surjective map, and for all $e_1, e_2 \in Ev_1$, if $h(e_1)$ and $h(e_2)$ are both defined, then $h(e_1) \rightarrow_2 h(e_2)$ iff $e_1 \rightarrow_1 e_2$.

These morphisms are easily extended to *labelled* Σ_I-event structures: if $h(e)$ is defined, then $\bar{\alpha}_2(h(e)) = \bar{\alpha}_1(e)$.

Labelled Σ_I-event structures and their morphisms form a category Σ_I**-evt**. We cannot go into a detailed analysis of this category, we just mention that it has initial elements[3]. In fact, we can prove that the labelled Σ_I-event structure constructed in definition 14 is an initial element.

Theorem 17. *$\bar{E}(\Sigma_I)$ is initial in Σ_I**-evt**.*

Proof: The initial morphism $h : \bar{E}(\Sigma_I) \rightarrow \bar{F}$ to some labelled Σ_I-event structure $\bar{F}$ is defined by sending the minimal element ε_i to the corresponding minimal element in F_i, for every $i \in Id$. The other events are mapped as follows. If $h(Cf)$ is defined for (every event in) a configuration Cf, then we define, for every action α, $h(Cf\,\alpha) = h(Cf)\alpha$ if the latter exists, otherwise $h(Cf\,\alpha)$ is undefined. By $h(Cf)\alpha$ we mean the immediate successor of all events in $h(Cf)$ labelled α. It is not hard to see that this defines a morphism and that it is unique. □

[3] For obvious reasons, we avoid the usual term "object" for an element of a category.

3 Object specification

Our logic for object class specification is a linear temporal logic with locality. We illustrate the idea by means of the `flip-flop` example.

It is beyond the scope of this paper to develop an abstract specification language for class signatures like the one in the `flip-flop` example 1 (cf. [ESSS95]). We give the "target" logic directly into which such a language is to be translated.

That is, we give a specification logic for the extended data signature $\Sigma = (S, \boldsymbol{\Omega})$.

The S-indexed family $\boldsymbol{T}_\Sigma(\boldsymbol{X})$ of sets of *terms* is defined as usual, employing an S-indexed family $\boldsymbol{X}$ of sets of *variables*.

Definition 18. The set $L_\Sigma(\boldsymbol{X})$ of *formulae* is inductively defined as follows:

- $(t_1 =_{ss} t_2) \in L_\Sigma(\boldsymbol{X})$ provided that $t_1, t_2 \in T_\Sigma(\boldsymbol{X})_s$;
- $\tau(i) \in L_\Sigma(\boldsymbol{X})$ provided that $i \in T_\Sigma(\boldsymbol{X})_{id}$;
- $\alpha(i)$, $\triangleright \alpha(i) \in L_\Sigma(\boldsymbol{X})$ provided that $\alpha(i) \in T_\Sigma(\boldsymbol{X})_{ac}$ where $i \in T_\Sigma(\boldsymbol{X})_{id}$ is the identity of α;
- $(\neg \varphi) \in L_\Sigma(\boldsymbol{X})$ provided that $\varphi \in L_\Sigma(\boldsymbol{X})$;
- $(\varphi \Rightarrow \varphi') \in L_\Sigma(\boldsymbol{X})$ provided that $\varphi, \varphi' \in L_\Sigma(\boldsymbol{X})$;
- $(\exists x\ \varphi) \in L_\Sigma(\boldsymbol{X})$ provided that $x \in X_s$ for some $s \in S$ and $\varphi \in L_\Sigma(\boldsymbol{X})$;
- $(\mathsf{X}_i\, \varphi), (\mathsf{F}_i\, \varphi), (\mathsf{Y}_i\, \varphi), (\mathsf{P}_i\, \varphi) \in L_\Sigma(\boldsymbol{X})$ provided that $i \in T_\Sigma(\boldsymbol{X})_{id}$ and $\varphi \in L_\Sigma(\boldsymbol{X})$.

Definition 19. The set $L_\Sigma^\tau(\boldsymbol{X})$ of *local formulae* is the set $\{i : \varphi \mid i \in T_\Sigma(\boldsymbol{X})_{id}, \varphi \in L_\Sigma(\boldsymbol{X})\}$.

The locality predicate $\tau(i)$ says that the formula is local to i, i.e., the "owner" of the local formula is communicating with i. The predicate $\alpha(i)$ means that action α has just occurred in object i. The predicate $\triangleright \alpha(i)$ means that action α is enabled in object i, i.e., it may happen next. Of course, actions must be enabled when they occur (i.e., just before they have just occurred). The symbols for the local temporal operators have the following meaning: X_i means next in i, F_i means sometime in the future of i, P_i means sometime in the past of i, and Y_i means previous (*yesterday*) in i. X_i and Y_i are meant to be the strong versions, i.e., the next and previous states, respectively, have to exist.

We apply the usual rules for omitting brackets, and we introduce further connectives through abbreviations, e.g., $(\varphi \vee \varphi')$ for $((\neg \varphi) \Rightarrow \varphi')$. The same applies to temporal operators, e.g., $(\mathsf{X}_i^?\, \varphi)$ for $(\neg(\mathsf{X}_i(\neg \varphi)))$, $(\mathsf{G}_i\, \varphi)$ for $(\neg(\mathsf{F}_i(\neg \varphi)))$, $(\mathsf{Y}_i^?\, \varphi)$ for $(\neg(\mathsf{Y}_i(\neg \varphi)))$, and $(\mathsf{H}_i\, \varphi)$ for $(\neg(\mathsf{P}_i(\neg \varphi)))$.

Definition 20. An *object specification* is a pair $Ospec = (\Sigma, \Phi)$ where Σ is an extended data signature, and $\Phi \subseteq L_\Sigma^\tau(\boldsymbol{X})$ is a set of local formulae as axioms.

Example 3. We specify a class of flip-flops, based on the signature given in example 2.

$$
\begin{array}{ll}
\textbf{sorts} & \texttt{nat},\ \texttt{FF}^i \le \texttt{id},\ \texttt{FF}^a \le \texttt{ac} \\[1ex]
\textbf{ops} & \texttt{F1},\ \texttt{F2} : \to \texttt{FF}^i \\
& \texttt{R} : \texttt{nat} \to \texttt{FF}^i \\
& \texttt{N} : \texttt{FF}^i \to \texttt{FF}^i \\
& \texttt{create},\ \texttt{set},\ \texttt{reset},\ \texttt{destroy} : \texttt{FF}^i \to \texttt{FF}^a \\
& \texttt{ai}_{\texttt{FF},\texttt{FF}} : \texttt{FF}^a \times \texttt{FF}^i \to \texttt{bool}
\end{array}
$$

$$
\begin{array}{l}
\textbf{axioms}\ \forall f, g{:}\texttt{FF},\ n{:}\texttt{nat} \\
\quad f\!: \ \texttt{create}(g) \Rightarrow \tau(g) \\
\quad f\!: \ \texttt{set}(g) \Rightarrow \tau(g) \\
\quad f\!: \ \texttt{reset}(g) \Rightarrow \tau(g) \\
\quad f\!: \ \texttt{destroy}(g) \Rightarrow \tau(g) \\[1ex]
\quad f\!: \ \triangleright \texttt{create}(f) \Leftrightarrow \neg \triangleright \texttt{set}(f) \wedge \neg \triangleright \texttt{reset}(f) \\
\quad f\!: \ \texttt{create}(f) \Rightarrow \mathsf{Y}_f \triangleright \texttt{create}(f) \\
\quad f\!: \ \texttt{create}(f) \Rightarrow \triangleright \texttt{set}(f) \\
\quad f\!: \ \texttt{create}(f) \Rightarrow \mathsf{G}_f(\neg \mathsf{P}_f\, \texttt{destroy}(f) \Rightarrow \neg(\triangleright \texttt{set}(f) \Leftrightarrow \triangleright \texttt{reset}(f))) \\
\quad f\!: \ \texttt{set}(f) \Rightarrow \mathsf{Y}_f \triangleright \texttt{set}(f) \\
\quad f\!: \ \texttt{set}(f) \Rightarrow \triangleright \texttt{reset}(f) \\
\quad f\!: \ \texttt{reset}(f) \Rightarrow \mathsf{Y}_f \triangleright \texttt{reset}(f) \\
\quad f\!: \ \texttt{reset}(f) \Rightarrow \triangleright \texttt{set}(f) \\
\quad f\!: \ \texttt{destroy}(f) \Rightarrow \mathsf{Y}_f \triangleright \texttt{destroy}(f) \\
\quad f\!: \ \texttt{destroy}(f) \Rightarrow \neg \triangleright \texttt{set}(f) \wedge \neg \triangleright \texttt{reset}(f) \\[1ex]
\quad f\!: \ \texttt{set}(f) \Rightarrow \mathsf{X}_f\, \texttt{set}(\texttt{N}(f)) \\[1ex]
\quad \texttt{F1}\!: \ \texttt{set}(\texttt{F1}) \Rightarrow \texttt{set}(\texttt{F2}) \\
\quad \texttt{F2}\!: \ \texttt{destroy}(\texttt{F2}) \Rightarrow \mathsf{P}_{\texttt{F1}}\, \texttt{set}(\texttt{F1}) \\[1ex]
\quad \texttt{R}(n)\!: \ \texttt{set}(\texttt{R}(n)) \Rightarrow \mathsf{F}_{\texttt{R}(n)}\, \texttt{set}(\texttt{R}(n+1))
\end{array}
$$

The first four axioms say that only shared actions may occur locally. Actually, these axioms need not be given explicitly because they are satisfied anyway in every interpretation (cf. definition 21 below).

Axiom five gives a necessary and sufficient condition for a flip-flop's creation to be enabled: it must neither be in a set nor in a reset state. Axiom six is the instance of a general rule: no action occurs unless it is enabled. Axioms seven and eight describe the state after creation: the flip-flop is set, and from now on it is always either set or reset as long as it is not destroyed. Axioms nine to twelve give the obvious preconditions and effects of set and reset. Axiom thirteen is another instance of the general rule mentioned. The fourteenth axiom puts a flip-flop after destruction in the same state as before creation: resurrection is possible!

The fifteenth axiom puts a clockwork of flip-flops into operation: flip-flops in the chain `N` are set step by step, one after the other, once an initial one is set.

The sixteenth axiom talks about two particular flip-flops, it says that whenever `F1` has been set, then `F2` has been set at the same time. This is an instance of synchronous *action calling*. As a consequence, we have $\texttt{set}(\texttt{F1}) \Rightarrow \tau(\texttt{F2})$ and thus $ai_{\mathtt{FF},\mathtt{FF}}(\mathtt{set}(\mathtt{F1}), \mathtt{F2}) = \mathtt{true}$, i.e., $\mathtt{set}(\mathtt{F1})$ is also an action of F2. The seventeenth axiom says that F2 can only be destroyed if it has at least once been set by F1 via action calling.

The eighteenth and last axiom says that whenever $\mathtt{R}(n)$ has been set, then $\mathtt{R}(n+1)$ will eventually be set. R is a "lazy" version of clockwork N. Consider the following alternative:

$$\mathtt{R}(n) : \mathtt{set}(\mathtt{R}(n)) \Rightarrow \mathsf{F}_{\mathtt{R}(n+1)}\, \mathtt{set}(\mathtt{R}(n+1))$$

This does not say the same: in the former case, a communication between $\mathtt{R}(n)$ and $\mathtt{R}(n+1)$ is required at the moment when $\mathtt{R}(n+1)$ is set. In the latter case, the communication must take place *right now* and reassure $\mathtt{R}(n)$ that its successor $\mathtt{R}(n+1)$ will eventually be set.

Here is a simple fact that is entailed by the axioms:

$$\mathtt{F1}: \ \mathtt{set}(\mathtt{F1}) \Rightarrow \tau(\mathtt{F2}) \wedge \mathsf{P}_{\mathtt{F2}}\, \mathtt{create}(\mathtt{F2}).$$

□

The logic $L_{\Sigma}(\boldsymbol{X})$ over an extended data signature is interpreted over an instance signature $\Sigma_I = (Id, \boldsymbol{Ac})$ based on a data signature Σ_D and a data universe $\boldsymbol{U}$, and an interpretation structure $\bar{L} = (L, \bar{\alpha}\mid_L)$ within an interpretation frame $\bar{E} = (E, \bar{\alpha})$, as described in section 2. Please remember that $L = \bigcup\{L_i\}_{i \in Id}$ where $L_i = (Lc_i, \rightarrow_i)$, and $E = \bigcup\{E_i\}_{i \in Id}$ where $E_i = (Ev_i, \rightarrow_i)$.

In particular, the local formulas are interpreted in a local configuration $\downarrow e$ of L and a variable assignment θ. Of course, data terms are to be interpreted globally in $\boldsymbol{U}$.

Definition 21. The *satisfaction* relation $\models$ is inductively defined by the following rules:

- $\bar{L}, \downarrow e, \theta \models i : (t_1 =_{ss} t_2)$ iff $t_1{}_U^\theta = t_2{}_U^\theta$;
- $\bar{L}, \downarrow e, \theta \models i : \tau(j)$ iff $e \in Lc_{i_U^\theta} \cap Lc_{j_U^\theta}$;
- $\bar{L}, \downarrow e, \theta \models i : \alpha(j)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{\alpha}(e) = \alpha(j)$;
- $\bar{L}, \downarrow e, \theta \models i : \triangleright\alpha(j)$ iff $e \in Lc_{i_U^\theta}$ and, for some $e' \in Ev_{i_U^\theta}$, $e \rightarrow_{i_U^\theta} e'$ and $\bar{\alpha}(e') = \alpha(j)$;
- $\bar{L}, \downarrow e, \theta \models i : (\neg\, \varphi)$ iff $e \in Lc_{i_U^\theta}$ and not $\bar{L}, \downarrow e, \theta \models i : \varphi$;
- $\bar{L}, \downarrow e, \theta \models i : (\varphi \Rightarrow \varphi')$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e, \theta \models i : \varphi'$ or not $\bar{L}, \downarrow e, \theta \models i : \varphi$;
- $\bar{L}, \downarrow e, \theta \models i : (\exists x\ \varphi)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e, \theta' \models i : \varphi$ for some x-equivalent assignment θ';
- $\bar{L}, \downarrow e, \theta \models i : (\mathsf{X}_j\, \varphi)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e', \theta \models j : \varphi$ for some event $e' \in Lc_{j_U^\theta}$ such that $e \rightarrow_{i_U^\theta} e'$ or $e \rightarrow_{j_U^\theta} e'$;
- $\bar{L}, \downarrow e, \theta \models i : (\mathsf{F}_j\, \varphi)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e', \theta \models j : \varphi$ for some event $e' \in Lc_{j_U^\theta}$ such that $e \rightarrow^* e'$;
- $\bar{L}, \downarrow e, \theta \models i : (\mathsf{Y}_j\, \varphi)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e', \theta \models j : \varphi$ for some event $e' \in Lc_{j_U^\theta}$ such that $e' \rightarrow_{i_U^\theta} e$ or $e' \rightarrow_{j_U^\theta} e$;

– $\bar{L}, \downarrow e, \theta \models i : (\mathsf{P}_j\, \varphi)$ iff $e \in Lc_{i_U^\theta}$ and $\bar{L}, \downarrow e', \theta \models j : \varphi$ for some event $e' \in Lc_{j_U^\theta}$ such that $e' \rightarrow^* e$.

This requires some explanation.

The first rule is straightforward, equations between data terms are interpreted globally. The second rule says that j is local in i iff i's current event is shared by j. The third rule says that, in any local configuration of a proper event, there is precisely one action that just occurred (possibly shared with some other object j), given by the label.

The forth rule is a little unusual in that it is not interpreted in an isolated life cycle but in a life cycle *in context*. Intuitively, $\alpha(j)$ is enabled if it may happen in some next step in the *frame*, not necessarily in the life cycle. A more classic way to capture this would use life cycles with one-step look-ahead ("barbed wires") as interpretation structures.

Rules five to seven are adapted from predicate calculus.

The eighth rule requires some thought: the events e and e' may belong to different life cycles! For i to know that φ holds for j tomorrow, i and j may communicate either today or tomorrow.

Also in the nineth rule, e and e' may belong to different objects. $i : \mathsf{F}_j\, \varphi$ holds in a local configuration $\downarrow e$ for i iff $j : \varphi$ holds in some future configuration $\downarrow e'$ for j where e' causally depends on e. This causal dependency may involve a chain of objects $i = i_1, \ldots, i_n = j$ where successive objects communicate via some shared event.

The last two rules are the past-directed analoga of future-directed rules eight and nine.

Definition 22. A labelled distributed life cycle $\bar{L}$ *satisfies* a local formula $i : \varphi$, written $\bar{L} \models i : \varphi$, iff $\bar{L}, \downarrow e, \theta \models i : \varphi$ for every local configuration $\downarrow e$ and every variable assignment θ. A local formula $i : \varphi$ is *valid* in the interpretation frame $\bar{E}$, written $\bar{E} \models i : \varphi$, iff $\bar{L} \models i : \varphi$ for every distributed life cycle $\bar{L}$ in $\bar{E}$.

Interpretation structures represent single runs of processes. Since we are interested in entire processes, we may ask which interpretation frames represent specified processes, i.e., have the property that all axioms are valid. In particular, it is interesting to look at these interpretation frames within the category Σ_I-**evt** of all interpretation frames, with morphisms as defined in definition 16. Since there is an initial element in Σ_I-**evt** (theorem 17), the obvious question to ask is whether the same holds for the subcategory of interpretation frames satisfying given axioms Φ. The answer is positive.

Let *Ospec* $= (\Sigma, \Phi)$ be an object specification (cf. definition 20), and let Σ_I be the instance signature determined by Σ and a given data universe $\boldsymbol{U}$ (cf. definition 5). Let ***Ospec*-evt** be the full subcategory of all elements in Σ_I-**evt** satisfying Φ.

Theorem 23. ***Ospec-evt*** *has initial elements.*

Proof idea: an initial element is given by the largest labelled sub-event structure of $\bar{E}(\Sigma_I)$, the initial element of Σ_I-**evt** (cf. theorem 17), containing all distributed life cycles satisfying all axioms in Φ. □

As in abstract data type theory, the initial elements of Σ_I-**evt** and ***Ospec*-evt** are obvious candidates for assigning standard semantics to signatures and specifications, respectively.

4 Concluding Remarks

The local specification logic and distributed semantics presented in this paper deserve further study. An obvious next step is to provide a proof system. [LMRT91] give a sound and complete proof system for their propositional n-agent logic, but that does not carry over to our case. It is not clear whether a complete proof system for our logic exists. But still, there are software tools for analysing and animating temporal logic specifications [Sa91b] that can be adapted to our approach.

Our theory will be extended towards *structured* specifications. That means that we have to introduce and study appropriate signature and specification morphisms and corresponding forgetful functors on interpretation frames and structures. This would provide the basis for studying composition as well as parameterization. For composition, there are two aspects: composing sequential objects from components [ESSS95], and composing concurrent families from subfamilies.

There is work in progress to incorporate reification issues in our framework [De94, De95]. It is well known that the temporal operators of our logic are not suitable for action refinement, the logic in [CE94] (which is similar to Hennessy-Milner logic [HM85]) may be better suited. This is an indication that we should work in a family of related logics, as put forward in [MM94].

While reification requires a more "operational" logic, specification expressiveness and comfort suggest to move in the opposite direction. An obvious extension of our logic is to introduce branching-time operators. Branching-time formulae cannot be interpreted in single life cycles. That means that our interpretation frames will play the role of interpretation structures.

Acknowledgments

The authors are grateful to their colleagues in the IS-CORE and COMPASS projects for inspiration about the nature of objects and how to reason about them. Special thanks are due to Cristina Sernadas and Gunter Saake for attention, hints, suggestions, objections, and understanding. The work of José Fiadeiro and Tom Maibaum has been influential. Also many thanks to Grit Denker, and to the Hildesheim process logic group of Ulla Goltz for surprising insights in the nature of process logics, concurrency, and locality. Finally, we are grateful to the anonymous referees for friendly comments and constructive criticism.

References

[AR92] E. Astesiano and G. Reggio. Algebraic Specification of Concurrency. Recent Trends in Data Type Specification, LNCS 655, Springer-Verlag, Berlin 1992

[Bo85] A. Borgida. Features of Languages for the Development of Information Systems at the Conceptual Level. *IEEE Software* 2 (1985), 63-73

[Br93] M. Broy. Functional Specification of Time-Sensitive Communicating Systems. *ACM Transactions on Software Engineering and Methodology* 2 (1993), 1-46

[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

[Ch76] P. P Chen. The Entity-Relationship Model—Toward a Unified View of Data. *ACM Transactions on Database Systems*, Vol. 1, No. 1, 1976, 9–36

[De94] G. Denker. Object Reification (Extended Abstract). Working Papers of the International Workshop on Information Systems – Correctness and Reusability, IS-CORE'94. R. Wieringa and R. Feenstra, eds. Technical Report IR-357, VU Amsterdam 1994

[De95] G. Denker. Transactions in Object-Oriented Specifications. This volume

[EDG88] H.-D. Ehrich, K. Drosten, and M. Gogolla. Towards an Algebraic Semantics for Database Specification. In: R. Meersmann and A. Sernadas (eds.). *Proc. 2nd IFIP WG 2.6 Working Conf. on Database Semantics "Data and Knowledge" (DS-2)*, Albufeira (Portugal), 1988. North-Holland, Amsterdam, 119-135

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data & Knowledge Engineering, North-Holland*, Vol. 9, No. 2, 1992, 157-204

[EGL89] H.-D. Ehrich, M. Gogolla, and U. Lipeck. Algebraische Spezifikation Abstrakter Datentypen. Teubner–Verlag, Stuttgart 1989

[EGS91] H.-D. Ehrich,J. Goguen, and A. Sernadas. A Categorial Theory of Objects as Observed Processes. Proc. REX/FOOL School/Workshop, J. W. deBakker et. al. (eds.), LNCS 489, Springer-Verlag, Berlin 1991, 203-228

[Eh86] H.-D. Ehrich. Key Extensions of Abstract Data Types, Final Algebras, and Database Semantics. In: D. Pitt et al. (eds.): *Proc. Workshop on Category Theory and Computer Programming*. Springer, Berlin, LNCS series, 1986, 412-433

[EM85] H. Ehrig and B. Mahr. Fundamentals of Algebraic Specification 1. Springer-Verlag, Berlin 1985

[EM90] H. Ehrig and B. Mahr. Fundamentals of Algebraic Specification 2. Springer-Verlag, Berlin 1985

[ES91] H.-D. Ehrich and A. Sernadas. Fundamental Object Concepts and Constructions. Information Systems – Correctness and Reusability, Proc. ISCORE Workshop'91 (G. Saake and A. Sernadas, eds.), Informatik-Berichte 91-03, Techn. Univ. Braunschweig 1991, 1-24

[ESSS95] H.-D. Ehrich, G. Saake, A. Sernadas, and C. Sernadas. Distributed Temporal Logic for Concurrent Object Families (Extended Abstract). Proc. ISCORE Workshop '94, R. Wieringa, ed. World Scientific Publishers. To appear 1995

[FM91] J. Fiadeiro and T. Maibaum. Towards Object Calculi. Information Systems – Correctness and Reusability, Proc. ISCORE Workshop'91 (G. Saake and A. Sernadas, eds.), Informatik-Berichte 91-03, Techn. Univ. Braunschweig 1991, 129-178

[FM92] J. Fiadeiro and T. Maibaum. Temporal Theories as Modularisation Units for Concurrent System Specification. *Formal Aspects of Computing* 4 (1992), 239-272

[FSMS91] J. Fiadeiro, C. Sernadas, T. Maibaum, and G. Saake. Proof-Theoretic Semantics of Object-Oriented Specification Constructs. In: R. Meersman, W. Kent, and S. Khosla (eds.). *Object-Oriented Databases: Analysis, Design and Construction (Proc. 4th IFIP WG 2.6 Working Conference DS-4, Windermere (UK))*, Amsterdam, 1991. North-Holland, 243-284

[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

[GM87] J. A. Goguen and J. Meseguer. Unifying functional, object-oriented and relational programming with logical semantics. *Research Direction in Object-Oriented Programming*, B.Shriver,P.Wegner (eds.), MIT Press 1987, 417-477

[GW90] J. A. Goguen and D. Wolfram. On Types and FOOPS. In: R. Meersman, W. Kent, and S. Khosla (eds.). *Object-Oriented Databases: Analysis, Design and Construction (Proc. 4th IFIP WG 2.6 Working Conference DS-4, Windermere (UK))*, Amsterdam, 1991. North-Holland

[HM85] M. Hennessy and R. Milner. Algebraic Laws for Nondeterminism and Concurrency. Journal of the ACM 32 (1985), 137-161

[Ho85] C. A. R. Hoare. Communicating Sequential Processes. Prentice-Hall, Englewood Cliffs, NJ, 1985

[HSJHK94] T. Hartmann, G. Saake, R. Jungclaus, P. Hartel, and J. Kusch. Revised Version of the Modeling Language `TROLL`. Informatik-Bericht 94-03, TU Braunschweig 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

[Ju93] R. Jungclaus. Modeling of Dynamic Object Systems, a Logic-based Approach. Advanced Studies in Computer Science. Vieweg Verlag, Braunschweig/Wiesbaden, 1993

[LMRT91] K. Lodaya, M. Mukund, R. Ramanujam, and P.S. Thiagarajan. Models and Logics for True Concurrency. in P.S. Thiagarajan (ed.): Some Models and Logics for Concurrency. Advanced School on the Algebraic, Logical and Categorical Foundations of Concurrency. Gargnano del Garda, 1991

[MB89] J. Mylopoulos and M. Brodie, (eds.). Readings in Artificial Intelligence & Databases. Morgan Kaufmann Publ. San Mateo, 1989

[Mi89] R. Milner. Communication and Concurrency. Prentice-Hall, Englewood Cliffs, 1989

[MM93] N. Martí-Oliet and J. Meseguer. Rewriting Logic as a Logical and Semantic Framework. Report SRI-CSL-93-05, SRI International, Menlo Park 1993

[MM94] N. Martí-Oliet and J. Meseguer. General Logics and Logical Frameworks. In: D. M. Gabbay (ed.). *What is a Logical System?*. Oxford University Press 1994. To appear

[MP89] Z. Manna and A. Pnueli. The Anchored Version of the Temporal Framework. In: J. deBakker, W. deRoever, and G. Rozenberg (eds.). *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency.* LNCS 354, Springer-Verlag, Berlin, 1989, 201-284

[Sa91a] G. Saake. Conceptual Modeling of Database Applications. In: Karagiannis, D. (ed.): *Proc. 1st IS/KI Workshop, Ulm (Germany), 1990.* Springer, Berlin, LNCS 474, 1991, 213-232

[Sa91b] G. Saake. Descriptive Specification of Database Object Behaviour. *Data & Knowledge Engineering* 6 (1991), 47-74

[Se80] A. Sernadas. Temporal Aspects of Logical Procedure Definition. *Information Systems*, Vol. 5, 1980, 167–187

[SEC90] A. Sernadas, H.-D. Ehrich, and J.-F. Costa. From processes to objects. *The INESC Journal of Research and Development 1:1*, pages 7-27, 1990

[SF91] C. Sernadas and J. Fiadeiro. Towards Object-Oriented Conceptual Modelling. *Data & Knowledge Engineering* 6 (1991), 479-508

[SJE92] G. Saake, R. Jungclaus, and H.-D. Ehrich. Object-Oriented Specification and Stepwise Refinement. In J. de Meer, V. Heymer, and R. Roth, editors, *Proc. Open Distributed Processing, Berlin (D), 8.-11. Okt. 1991 (IFIP Transactions C: Communication Systems, Vol. 1)*, pages 99-121. North-Holland, 1992

[SJH93] G. Saake, R. Jungclaus, and T. Hartmann. Application Modelling in Heterogenous Environments Using an Object Specification Language. *International Journal of Intelligent and Cooperative Information Systems* 2 (1993), 425-449

[SR94] A. Sernadas and J. Ramos. The GNOME Language: Syntax, Semantics and Calculus. Tech. Report, Instituto Superior Técnico, Lisboa 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)

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

[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

[ST89] J. W. Schmidt and C. Thanos (eds.). Foundations of Knowledge Base Management. Springer-Verlag, Berlin, 1989

[Wi80] G. Winskel: Events in Computation. PhD thesis, University of Edinburgh

[WN93] G. Winskel and M. Nielsen. Models for Concurrency. Report DAIMI PB – 463, Computer Science Department, Aarhus University 1993

This article was processed using the LaTeX macro package with LLNCS style