Acta Informatica 36, 591–616 (2000)

Specifying communication in distributed information systems

Hans-Dieter Ehrich[1], **Carlos Caleiro**[2]

[1] Abteilung Informationssysteme, Technische Universität, Postfach 3329, 38023 Braunschweig, Germany

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

Received: 28 September 1998 / 18 November 1999

Abstract. We present two logics that allow specifying distributed information systems, emphasizing communication among sites. The low-level logic D_0 offers features that are easy to implement but awkward to use for specification, while the high-level logic D_1 offers convenient specification features that are not easy to implement. We show that D_1 specifications may be automatically translated to D_0 in a sound and complete way. In order to prove soundness and completeness, we define our translation as a simple map of institutions. Our result may be useful for making implementation platforms like CORBA easier accessible by providing high-level planning and specification methods for communication.

1 Introduction

High-level planning, modeling and specification methods for distributed information systems are not yet well understood, especially when it comes to communication. As is typical also for other areas, we have low-level concepts that are easy to implement but not adequate for specification, and we have high-level concepts that are closer to the ways application people think and argue, so they are more convenient to use for modeling and specification, but are not easy to implement efficiently.

This paper shows how to bridge the gap in one particular respect: specifying and implementing synchronous communication in object systems.

This work was partially supported be the PRAXIS XXI Program and FCT, by PRAXIS XXI Projects 2/2.1/MAT/262/94 SitCalc, PCEX/P/MAT/46/96 ACL and 2/2.1/TIT/1658/95 LogComp, and ESPRIT IV Working Groups 22704 ASPIRE and 23531 FIREworks.

In [ECSD98], we presented the basic idea of two distributed logics, both capable of expressing communication among objects, one low-level and one high-level. We showed by example how useful the latter is for specification, and how formulae of the high-level logic may be automatically translated into formulae of the low-level logic. The latter corresponds with concepts incorporated in TROLL3, a specification language research prototype developed in the first author's group.

In this paper, we give simplified versions of the two logics that keep the essential features, though. Then we formally describe the translation process and prove that it is sound and complete.

In a sense, this paper gives the theoretical underpinning for ideas presented informally in [ECSD98]. We have taken care, however, to make the paper self-contained.

Among the logic-based formal methods used for specifying information systems, the work reported here is based on experiences with developing the OBLOG family of languages and their semantic foundations. OBLOG is being developed into a commercial product [Esp93]. In the academic realm, there are several related developments: TROLL [JSHS96, HJ95, EH96, Har97], GNOME [SR94], LCM [FW93] and ALBERT [DDPW94].

There are other approaches to formal object specification with a sound theoretical basis. The ones most closely related to ours are FOOPS [GM87, RS92, GS95] and MAUDE [Mes93]. FOOPS is based on OBJ3 [GW88] which is in turn based on equational logic. MAUDE is based on rewriting logic that is a uniform model of concurrency [Mes92].

In the OBLOG family, TROLL3 [DH97] is the first to address problems of concurrency and communication, and to integrate benefits from informal methods like OMT [JWH$^+$94]. So there is a graphical notation for TROLL3 called OMTROLL. Theoretical foundations of concurrency as applied to this approach have been explored in [ES95, Ehr99, Kus97], among others. [EH96] gives a brief overview of TROLL3 and OMTROLL and their logic foundations. TROLL does not only have a formal syntax and semantics, but also a method and a workbench [HDK$^+$97, GK97]. The approach is being developed along with a practical application project [KKH$^+$96, HDK$^+$97, GK$^+$98].

For specifying and reasoning about concurrent and distributed systems, two major approaches have been advocated. On one hand, there was a systematic study of the relations and equivalences between behaviours of systems that started in [Mil80] and was subsequently pursued by many people working in process algebra [Hoa85]. On the other hand, the use of modal logics [Gol92] for characterizing properties of systems has evolved from the early works of Floyd [Flo67] and Hoare [Hoa69] on reasoning about

programs. Among them we mention in particular dynamic logics [FL79, Har79, Pel87], temporal logics [Pnu77] and, more recently, logics of knowledge [FHMV95].

The distributed logics defined in this paper are based on temporal logic and an extension towards concurrency called n-agent logic. Temporal logic has been successfully applied to a number of reactive systems specification problems [MP92], it is the simplest logic that can not only deal with safety properties but also with liveness properties [Lam77]. n-agent logic was introduced and developed in [LT87, LMRT91, LRT92, Thi94, Ram96]. An agent corresponds to an object that may be thought of as a site in a distributed system.

There are several distinct approaches to reasoning about concurrency within the framework of temporal logic. The first and simplest accepts the naive view of a concurrent system modelled by nondeterministic interleaving together with adequate assumptions of fairness [GPSS80, Fra86] on the execution of its components. The use of branching temporal logics like *UB* [BAMP81], *CTL* [CE81] or *CTL** [EH83] and even of linear temporal logic for such purposes has been deeply studied [Pnu77, MP92, Wol95, Pen95]. In particular, most of the work on temporal logics for information systems and object orientation we rely on has been developed in this setting (e.g. [Ser80, SFSE89, FM92, SSC95, SSR96, DRCS97, SSC98]).

However, certain "subtleties" of concurrent systems are lost under such simplifications (see, for instance, [Pra86] for a discussion on the subject). The need for a precise notion of causality as reflected by the time structures adopted for the logics led to the study of partial order temporal logics [PW84, KP87, Pen95].

The basic difference is that the assumption of an omnipresent observer of the entire system being considered is dropped and replaced by a local causal perspective. It is the case of the logics for partially ordered computations introduced by Pinter and Wolper [PW84], of interleaving set temporal logics [KP87] and of temporal logics for reasoning about distributed transition systems [LPRT95], systems with product state spaces [Thi95a], occurrence nets [Rei92] and event structures [Pen88].

n-agent temporal logics can be found within the latter. They adopt event structures [Win87] enriched with information about its sequential agents [LT87] as models of concurrent systems.

This approach already complies with the fact that the view each individual agent may have of the whole distributed system at a particular instant in time is partial, due to the relative autonomy and spatial separation between agents, and has to be supported by communication [LRT92]. n-agent logics can explicitly distinguish sequential agents (localities) in the system, refer

to the local viewpoint of each agent, and express communication between agents (the major feature of distribution) [LT87, LRT92, Ram96].

Several versions of n-agent logics can be found, still reflecting different perspectives on how non-local information can be accessed by each agent [Ram96]. The logics D_0 and D_1 we propose assume that an assertion about another agent is only possible at a communication point. But for instance, the logics in [LMRT91] assume that, at each instant, the actual information about another agent is the one corresponding to the last communication with it.

Other n-agent logics [Ram96] do even consider the existence of a "present knowledge" modality allowing reference to non-local properties of agents, cf. [KR94].

In the following section, we outline by example the ideas of D_0 and D_1, their translation, and their use in an application case study. In section 3, we present syntax and interpretation of the two logics. The interpretation part refers to event structures but is kept as elementary as possible. In section 4, we give a detailed and precise account of how to reduce D_1 specifications to D_0 specifications. In section 5, we prove soundness and completeness of this reduction. To this end, we get into the institutional framework as defined by [GB92] and define our translation as a simple map of institutions as defined by [Mes89]. Our soundness and completeness results follow from well known results about institutions and their maps. Finally, concluding remarks in section 6 sum up the message of the paper and outline possibilities for further work.

2 Example

Suppose we want to specify that *"I will next call Jean and tell her to call you afterwards"*. The statement talks about a distributed system with three objects: I (i), Jean (j), and you (u). It says that u hears from i that i will next call j and tell her to call u sometime later. The following picture illustrates the communication structure. $\bigtriangledown$ denotes "now", the point in time when the above is uttered.

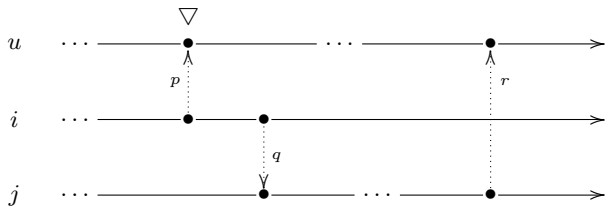


Fig. 1. Communication structure of example

$p \equiv$ " u hears from i that he will next call j and tell her to call u sometime later";
$q \equiv$ " i calls j and tells her to call u afterwards";
$r \equiv$ " j calls u".

Formally, the assertions may be expressed as follows. Let $x.\varphi$ mean that the formula φ holds true for object x, and let $x.@y$ mean that x is currently communicating with y.

$u.p$: in u, p holds true;
$i.(p \Leftrightarrow (@u \wedge (\mathsf{X}\, q)))$: in i, p means that there is communication with u, and q will hold next;
$j.(q \Leftrightarrow (@i \wedge (\mathsf{F}\, r)))$: in j, q means that there is communication with i, and r will hold sometime in the future;
$u.(r \Leftrightarrow @j)$: in u, r means that there is communication with j.

The fact that p, q and r denote communications may be expressed by
$u.(p \Rightarrow i.p)$ and $i.(p \Rightarrow u.p)$,
$j.(q \Rightarrow i.q)$ and $i.(q \Rightarrow j.q)$,
$u.(r \Rightarrow j.r)$ and $j.(r \Rightarrow u.r)$,

where p is shared by u and i, q is shared by i and j, and r is shared by u and j. In technical terms, "p is shared by u and i" means that there are two distinct propositional symbols, $u.p$ and $i.p$, and the same for q and r.

The formulae are all localized, they state facts from the viewpoints of objects. The communication expressions used so far are basic and simple synchronisation tags and state predicate callings:

- $@y$ is a synchronization tag, expressing that there is communication with y, and
- $(p \Rightarrow y.p)$ is state predicate calling, expressing that if p holds, then there is communication with y, whereupon p holds for y as well.

These are the communication primitives of our communication logic D_0. These primitives are used in the kernel version 3 of the TROLL specification language. It is straightforward to translate them to C^{++} [HDK+97, GK+98].

This way of communication specification, however, is rather low-level. It is awkward because it requires to introduce auxiliary communication predicates, p, q and r in our example, that have no direct real-world meaning, and it requires to introduce many detailed assertions of a purely technical nature.

A higher-level way of expressing communication is conceivable if we allow *arbitrary statements about other objects within objects*. Following this idea, the above example may be expressed as

$$u.i.(\mathsf{X}\, j.(\mathsf{F}\, @u)).$$

In D_1, we may write $i.j.\mathit{true}$ which has the same meaning as $i.@j$. So the above formulae is equivalent to $u.i.(\mathsf{X}\, j.(\mathsf{F}\, u.\mathit{true}))$. Locally for u, an assertion about i is being made, namely that $(\mathsf{X}\, j.(\mathsf{F}\, @u))$ holds for i. The intended meaning is that u *learns* the truth of this formula from i, namely that next $j.(\mathsf{F}\, @u)$ will hold. The latter means that i learns from j: in her local view, $(\mathsf{F}\, @u)$ holds. This means that, sometime in her future, $@u$ will hold: she will communicate with u.

This is the idea of communication logic D_1: any assertion about any other object may be made, meaning that there is communication with that other object, and the assertion holds in this other object while communication takes place.

Note that the D_1 formula captures all the information that the above set of D_0 formulae expresses. So there should be a way to recover the D_0 specification from the D_1 specification. And indeed there is, as we show in the subsequent sections.

First, however, we demonstrate the usefulness of D_1 by a real-world (though largely simplified) example adapted from [ECSD98]. The example is about business process modeling and is extracted from a national German project in which the first author is involved. A brief description of this project is useful for getting a feeling how D_1 can be used. The project is a cooperation with group 3.4 within PTB (Physikalisch Technische Bundesanstalt) in Braunschweig, the German federal institute of weights and measures. Group 3.4 is concerned with testing and certifying explosion proof electrical equipment such as motors, switches, etc., so that it is allowed to be operated in hazardous areas. The assessment procedure consists of testing the formal and informal documents, checking the design papers (mostly technical drawings), and the tests which are carried out. The group is divided into three subgroups dealing with experimental tests in the laboratories, basic administration work, and design approval, respectively.

Clients contact group 3.4 and ask for the certification of a specific electrical device. Administration employees then talk to them and to employees from the laboratory and from design approval. In some cases, design approval employees directly contact the client to ask for missing formal documents, technical drawings, etc.

In particular, the following actions take place during the business process of certifying an electrical device. A client may request the certification of an electrical device (*cert-req*). This implies that the administration officer orders sometime later appropriate tests at the laboratories (*test-ord*) and design approvals (*appr-ord*). The corresponding subgroups answer these queries by returning the results of the tests (*test-res(b)*) and design approvals (*appr-res(b)*), respectively, where $b \in \{\mathit{true}, \mathit{false}\}$. Depending on the results, the administration officer decides whether a certificate is issued (*cert-dec*).

Often it is the case that some formal documents are missing. In such a case the design approval officer clarifies the situation by asking the client for the missing papers (*clar-req*). The approval officer informs the administration in doing so. No tests will be carried out as long as the design approval papers are not complete. The approval officer informs the administration officer about the completeness of the application papers (*comp-res(b)*).

The business process is formalized using D_1 in the following way. The system consists of four objects, a CLIENT (*c*), an ADMINISTRATION officer (*a*), a LABORATORY officer (*l*), and a DESIGN APPROVAL officer (*d*). We have local sets of state predicates for these objects, P_c, P_a, P_l and P_d, respectively, given as follows. Let $b \in B$ where $B = \{\mathit{true}, \mathit{false}\}$.

CLIENT	$P_c ::=$ *cert-req*
ADMINISTRATION	$P_a ::=$ *test-ord* | *appr-ord* | *cert-dec(b)* | *dec=b* | *answers=0* | *answers=1* | *answers=2*
LABORATORY	$P_l ::=$ *test-res(b)*
DESIGN APPROVAL	$P_d ::=$ *clar-req* | *comp-res(b)* | *appr-res(b)*

The administration officer has an attribute *dec* which determines the certification decision made by him. Moreover, he has an attribute *answers* to store how many results he already received. I.e., this attribute determines whether both laboratory and design approval officers delivered their results to him (*answers*$= 2$), only one of them (*answers*$= 1$), or none (*answers*$= 0$).

The administration behaviour specification is given as follows.

ADMINISTRATION *behaviour*

a1 $a.(\mathit{test\text{-}ord} \Rightarrow (\mathsf{F}\ l.\mathit{test\text{-}res}(b)))$, for any $b \in B$;
a2 $a.((\neg\ \mathit{test\text{-}ord})\ \mathcal{U}\ d.\mathit{comp\text{-}res}(\mathit{true}))$;
a3 $a.((\neg\ \mathit{cert\text{-}dec}(b))\ \mathcal{U}(d.\mathit{appr\text{-}res}(b^{'})))$, for any $b, b^{'} \in B$;
a4 $a.((\neg\ \mathit{cert\text{-}dec}(b))\ \mathcal{U}(l.\mathit{test\text{-}res}(b^{'})))$, for any $b, b^{'} \in B$;
a5 $a.((c.\mathit{cert\text{-}req}) \Rightarrow (dec = \mathit{true}\ \wedge answers = 0))$;
a6 $a.((l.\mathit{test\text{-}res}(b)) \Rightarrow ((dec = dec \wedge b) \wedge (answers = answers + 1)))$, for any $b \in B$;
a7 $a.((d.\mathit{appr\text{-}res}(b)) \Rightarrow ((dec = dec \wedge b) \wedge (answers = answers + 1)))$, for any $b \in B$;
a8 $a.((answers = 2) \Rightarrow (\mathsf{F}\ \mathit{cert\text{-}dec}(dec)))$;
a9 $a.((\neg\ \mathit{cert\text{-}dec}(b))\ \mathcal{U}(answers = 2))$, for any $b \in B$;
a10 $a.((answers = n \wedge dec = b^{''}) \Rightarrow ((answers = n \wedge dec = b^{''})\ \mathcal{U}\ (l.\mathit{test\text{-}res}(b^{'}) \vee d.\mathit{appr\text{-}res}(b))))$, for any $b, b^{'}, b^{''} \in B$;

The first two axioms assure that the administration officer will get a result from the laboratory after he ordered tests, but those tests cannot be carried out unless the design approval officer reported completeness of the application papers. A decision about the application cannot be made until

both results, from design approval and laboratory, have been given (a3 and a4). The fifth axiom assures that the attributes have the right initialization values. a6 and a7 formalize which effects the results from laboratory or design approval, respectively, have on attributes *answers* and *dec*. They are abbreviations of syntactically correct D_1 formulae. E.g., formula a6 is an abbreviation of the following D_1 formula:

$$a.((dec = b^{'} \wedge answers = n) \Rightarrow \\ (\mathsf{X}\,(l.\textit{test-res}(b) \Rightarrow (dec = b^{'} \wedge b) \wedge answers = n + 1))).$$

Axioms a8 and a9 state that the administration officer will eventually make the decision after he received all results, but not earlier. The last axiom expresses that only the results from the laboratory or the design approval can change the decision. Thus, only two actions can change the value of that attribute. We assume overall frame axioms like, e.g., attributes can only be altered if an action takes place.

Given this specification and those of client, laboratory and design approval behaviours omitted here, we may show, among others, that a client will not be contacted by a design approval officer for a clarification after he has received the certification decision from the administration officer,

$$c.(\neg(a.\textit{cert-dec}(b) \wedge \mathsf{F}\; d.\textit{clar-req})).$$

The reader is invited to try the example with D_0 communication primitives.

3 Distributed logics

We take the view that a distributed system is a collection of communicating sequential objects, each with its own local logic. For object identification, we assume a fixed large enough set of object identifiers *Id*. As for the local logics, we assume that they are the same up to individual state predicates, typically expressing which attribute has which value, which actions occur or which actions are enabled. So each local logic is presented by a set of state predicates which we assume to be given.

Definition 1. A *system signature* is a pair $P = (I, \{P_i\}_{i \in I})$ such that

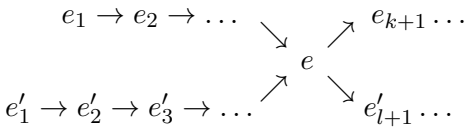
- $I \subseteq \mathit{Id}$ is a nonempty set (of object identifiers);
- P_i is a set (of state predicates), for each $i \in I$.

In what follows, we assume a fixed signature $P = (I, \{P_i\}_{i \in I})$.

A model of a distributed system over P consists of a set of distributed life cycles. Intuitively, a distributed life cycle is a global history of the system that consists of local histories, one for each object, fitting together in a coherent way.

Since objects are sequential, a history of an object is a finite or infinite sequence of events $e_1 \to e_2 \to \ldots$ An event is an abstraction of an occurrence of an action or of several simultaneous actions. An event will usually change the internal state. A relevant view of the state is captured by a function labelling each event with a set of local state predicates, namely those that hold in the state.

A global history can thus be seen as a collection of object histories, each one being a finite or infinite labelled sequence of events. Synchronous communication is modelled by events that are shared by several objects. So we have a set of threads glued together at communication points.



Naturally, event sharing has to satisfy a consistency condition: it must not introduce loops. That means that the entire distributed life cycle has to be a partial ordering (called *causality* in the theory of event structures [Win87]). It would be possible to use Kripke structures as interpretation structures, but we prefer to use the terminology of event structures because it is better known in the concurrency area.

Formally, we assume a finite or countably infinite global set Ev of system events, equipped with a "successor" relation $\to \subseteq Ev \times Ev$ whose reflexive and transitive closure $\to^*$ is a partial ordering. For later use, we introduce the notation $\to^+$ for the transitive closure of $\to$.

Each object $i \in I$ in the system may be engaged in a subset $Ev_i \subseteq Ev$ of system events, with local sequencing defined by the projection $\to_i^*$ where $\to_i = \to \cap\, Ev_i \times Ev_i$. We assume that an object is sequential, i.e., $\to_i^*$ is a branching structure (a set of trees). This represents the permissible behaviour of i with all its alternatives.

A life cycle of an object $i \in I$ is a subset $Lv_i \subseteq Ev_i$, equipped with the projection of $\to_i^*$ where the latter is a total ordering, together with a labelling representing state information. Thus, a life cycle is a linear sequence of labelled events. It represents a single run of object i.

Definition 2. Let $i \in I$ be an object. A *life cycle* for i over P is a triple $L_i = (Lv_i, \to_i^*, \lambda_i)$, where Lv_i and $\to_i^*$ are as introduced above, and $\lambda_i : Lv_i \to 2^{P_i}$ is a labelling function.

Definition 3. A *distributed life cycle* over P is an I-indexed family $L = \{L_i\}_{i \in I}$ of object life cycles over P.

The events in a distributed life cycle are given by $Lv = \bigcup_{i \in I} Lv_i$. Sequencing is given by $(\bigcup_{i \in I} \to_i)^*$. Note that the latter is not a total ordering

in general. Unrelated events represent concurrency, and events shared by the life cycles of several objects represent synchronous communication.

Note that labelling is individual for objects. That means that an event shared by several objects has several labels, one for each sharing object.

In the sequel, we assume a fixed distributed life cycle L over P.

We first introduce D_0, the lower-level distributed logic. Remember that the only communication formulae expressible in D_0 are state predicate callings and synchronisation tags.

Definition 4. The set of D_0 *formulas over* P is

$$\mathsf{D}_0(P) \quad ::= \quad \ldots \mid \mathsf{D}_0^i(P) \mid \ldots \quad (i \in I)$$

where, for each $i \in I$,

$$\begin{array}{lll} \mathsf{D}_0^i(P) & ::= & i.\mathsf{H}_0^i(P) \mid i.\mathsf{CC}_0^i(P) \\ \mathsf{H}_0^i(P) & ::= & @I^i \mid P_i \mid \mathit{false} \mid (\mathsf{H}_0^i(P) \Rightarrow \mathsf{H}_0^i(P)) \mid (\mathsf{H}_0^i(P)\ \mathcal{U}\ \mathsf{H}_0^i(P)) \\ \mathsf{CC}_0^i(P) & ::= & \ldots \mid (P_i \Rightarrow j.P_j) \mid \ldots \quad (j \in I^i). \end{array}$$

I^i is defined to be $I \setminus \{i\}$. D_0^i is the logic at locality i with callings to other localities. H_0^i is the "home" logic of object i. CC_0^i are the communication formulae expressible in object i. Note that communication formulae are stand-alone, they may not be embedded as subformulae of other formulae.

For examples of D_0, we refer to section 2.

The predicates in P are *flexible*, i.e., we intend to give them time-dependent meanings. The other symbols are *rigid*, i.e., we intend to give them time-independent meanings.

false is the usual logical constant, $\Rightarrow$ denotes logical implication, and $\mathcal{U}$ is the *until* temporal operator. $\varphi\ \mathcal{U}\ \psi$ means that φ will always be true from the next moment on until ψ becomes true for the next time; φ need not be true any more as soon as ψ holds; ψ must eventually become true.

As usual, we introduce derived connectives, e.g., $\neg\,\varphi$ for $\varphi \Rightarrow \mathit{false}$, true for $\neg\,\mathit{false}$, $\varphi \vee \psi$ for $\neg\,\varphi \Rightarrow \psi$, etc. We also make use of derived temporal operators like $\mathsf{X}\,\varphi$ for $\mathit{false}\ \mathcal{U}\ \varphi$ expressing *next*, $\mathsf{F}\,\varphi$ for $\varphi \vee \mathit{true}\ \mathcal{U}\ \varphi$ expressing *sometime*, $\mathsf{G}\,\varphi$ for $\neg(\mathsf{F}(\neg\,\varphi))$ expressing *always*, and $\varphi\ \mathcal{U}^\circ\ \psi$ for $\varphi \wedge \varphi\ \mathcal{U}\ \psi$ expressing *from now on ... until ...*

Our second distributed logic is D_1. It generalises D_0 in the sense that there are no special communication formulae: each D_1 formula *of another object* may serve as a communication formula—that may appear anywhere in context.

Definition 5. The set of D_1 *formulae over* P is

$$\mathsf{D}_1(P) \quad ::= \quad \ldots \mid \mathsf{D}_1^i(P) \mid \ldots \quad (i \in I)$$

where, for each $i \in I$,

$$\begin{array}{lll} \mathsf{D}_1^i(P) & ::= & i.\mathsf{H}_1^i(P) \\ \mathsf{H}_1^i(P) & ::= & @I^i \mid P_i \mid \mathit{false} \mid (\mathsf{H}_1^i(P) \Rightarrow \mathsf{H}_1^i(P)) \mid (\mathsf{H}_1^i(P)\ \mathcal{U}\ \mathsf{H}_1^i(P)) \mid \mathsf{CC}_1^i(P) \\ \mathsf{CC}_1^i(P) & ::= & \ldots \mid \mathsf{D}_1^j(P) \mid \ldots \quad (j \in I^i). \end{array}$$

It is obvious that $\mathsf{D}_0(P) \subseteq \mathsf{D}_1(P)$. Therefore, we define the satisfaction relation for a life cycle L on $\mathsf{D}_1(P)$ and then just restrict it to $\mathsf{D}_0(P)$.

Definition 6. The satisfaction relation $\models_{P,1}$ for L on D_1(P) is defined by

$$L \models_{P,1} i.\varphi \qquad \text{iff } L, e \models^i_{P,1} \varphi \text{ holds for every } e \in Lv_i.$$

For each $i \in I$ and $e \in Lv_i$, the relation $\models^i_{P,1}$ for L and e on H_1^i(P) is inductively defined as follows

$$\begin{array}{ll} L, e \models^i_{P,1} @j & \text{iff } e \in Lv_j; \\ L, e \models^i_{P,1} p & \text{iff } p \in \lambda_i(e); \\ L, e \models^i_{P,1} \mathit{false} & \text{does not hold} \\ L, e \models^i_{P,1} (\varphi \Rightarrow \psi) & \text{iff } L, e \models^i_{P,1} \varphi \text{ implies } L, e \models^i_{P,1} \psi; \\ L, e \models^i_{P,1} (\varphi\ \mathcal{U}\ \psi) & \text{iff there is an event } e' \in Lv_i \text{ with } e \to_i^+ e' \\ & \text{where } L, e' \models^i_{P,1} \psi, \text{ and } L, e'' \models^i_{P,1} \varphi \text{ for every} \\ & \text{event } e'' \in Lv_i \text{ such that } e \to_i^+ e'' \to_i^+ e'; \\ L, e \models^i_{P,1} j.\varphi & \text{iff } e \in Lv_j \text{ and } L, e \models^j_{P,1} \varphi. \end{array}$$

Definition 7. The satisfaction relation $\models_{P,0}$ for L on D_0(P) is defined by

$$L \models_{P,0} i.\varphi \qquad \text{iff } L, e \models^i_{P,0} \varphi \text{ holds for every } e \in Lv_i.$$

For each $i \in I$ and $e \in Lv_i$, the relation $\models^i_{P,0}$ for L and e on H_0^i(P) is defined by

$$L, e \models^i_{P,0} \varphi \qquad \text{iff } L, e \models^i_{P,1} \varphi.$$

Note that in D_1, a formula like $i.@j$ is equivalent to $i.j.\mathit{true}$. Still, this last formula is not in D_0 and therefore we chose to keep the @ synchronisation tag as primitive also in D_0. Moreover, it makes the inclusion of $\mathsf{D}_0(P)$ into $\mathsf{D}_1(P)$ more explicit.

4 Reduction

In this section, we give a precise account on how specifications in $\mathsf{D}_1(P)$ can be reduced to specifications in $\mathsf{D}_0(P \uplus C)$ where $P = (I, \{P_i\}_{i \in I})$ is a fixed system signature, and C is a new I-indexed family of sets of (communication) predicates. Naturally, $P \uplus C$ denotes the family $\{P_i \uplus C_i\}_{i \in I}$. $\uplus$ denotes disjoint union.

As illustrated above, the idea is that all communication subformulae in a given formula are replaced by explicit new communication predicates whose meaning is defined to be equivalent to that of the subformulae. In more detail, each implicit communication construct of D_1 (i.e., a non-local statement about a different object) is replaced by three constructs.

1. a 'communication predicate' symbol denoting the point of synchronization;
2. a 'definition' axiom stating what will happen at the callee's side after having passed the synchronization point;
3. two 'sharing' axioms stating that the caller will pass the point of synchronization if the callee does and vice versa.

Let us recall our example *"I will next call Jean and tell her to call you afterwards"*. In D_1, this may be expressed by the formula

$$u.i.(\mathsf{X}\, j.(\mathsf{F}\, @u)).$$

The idea is to recurrently scan this formula and replace its innermost communication subformulae by new communication predicates. The process terminates and yields a D_0 formula. Thus, in a first step, we should go as deep as $j.(\mathsf{F}\, @u)$ and introduce a new communication predicate $c(i, j.(\mathsf{F}\, @u))$ to be shared by i and j. By replacing the subformula by the new symbol, we obtain

$$u.i.(\mathsf{X}\, c(i, j.(\mathsf{F}\, @u))).$$

Iterating the process, we have to replace $i.(\mathsf{X}\, c(i, j.(\mathsf{F}\, @u)))$. It occurs in the scope of u, so the new communication predicate is denoted by $c(u, i.(\mathsf{X}\, c(i, j.(\mathsf{F}\, @u))))$ and shared by u and i. After replacing, we obtain

$$u.c(u, i.(\mathsf{X}\, c(i, j.(\mathsf{F}\, @u)))).$$

This formula is already in D_0.

Note that if we match p with $c(u, i.(\mathsf{X}c(i, j.(\mathsf{F}@u))))$ and q with $c(i, j.(\mathsf{F}@u))$, we obtain the intended result, cf. section 2. However, we do not have a communication predicate corresponding to r: there is no need to replace $@u$ because it is already in D_0.

The sharing axioms for the new communication predicates are obvious:

$j.(c(i, j.(\mathsf{F}\, @u)) \Leftrightarrow (@i \wedge (\mathsf{F}\, @u)))$;
$i.(c(u, i.(\mathsf{X}\, c(i, j.(\mathsf{F}\, @u)))) \Leftrightarrow (@u \wedge (\mathsf{X}\, c(i, j.(\mathsf{F}\, @u)))))$.

Formally, the family C of new communication predicates can be given an inductive definition, according to the motivation above. Remember that $I^i = I \setminus \{i\}$.

Definition 8. For each $i \in I$, let

– $C_i^0 = \emptyset$;
– $C_i^{n+1} = \{c(i, j.\varphi) : j \in I^i,\ j.\varphi \in \mathsf{D}_0(P \uplus C^n)\} \cup$
$\{c(k, i.\psi) : k \in I^i,\ i.\psi \in \mathsf{D}_0(P \uplus C^n)\}$.

The family $C = \{C_i\}_{i \in I}$ is defined by $C_i = \bigcup_{n \in \mathbb{N}} C_i^n$ for each $i \in I$.

As suggested in section 2, all these symbols must be properly axiomatized. The following axioms serve the purpose. Note that they are formulated in the logic $\mathsf{D}_0(P \uplus C)$.

Definition 9. The set of axioms corresponding to the family C of new communication predicates is $\Gamma = \bigcup_{i \in I} \Gamma_i \subseteq \mathsf{D}_0(P \uplus C)$ where, for each $i \in I$, we have

$$\begin{aligned}\Gamma_i = \ & \{i.(c(i, j.\varphi) \Rightarrow j.c(i, j.\varphi)) : c(i, j.\varphi) \in C_i\} \ \cup \\ & \{i.(c(k, i.\psi) \Rightarrow k.c(k, i.\psi)) : c(k, i.\psi) \in C_i\} \ \cup \\ & \{i.(c(k, i.\psi) \Leftrightarrow (@k \wedge \psi)) : c(k, i.\psi) \in C_i\}.\end{aligned}$$

The first two sets give the sharing axioms for each new communication predicate, while the third set gives the definition axioms for each new symbol.

In order to translate each formula from $\mathsf{D}_1(P)$ to $\mathsf{D}_0(P \uplus C)$, it suffices to capture the intuition outlined in section 2: we recurrently scan the formula and replace its innermost communication subformulae by their corresponding communication predicates. Clearly, this iteration stops after a finite number of loops—which is the maximum nesting depth of communication subformulae in the given formula. If the maximum nesting depth is m, then only new communication predicates up to C^m are necessary. Thus, if the formula is already in D_0, it is obvious that $m = 0$ and so no new symbols are needed.

Definition 10. For each $n \in \mathbb{N}$, let $\eta : \mathsf{D}_1(P \uplus C^n) \to \mathsf{D}_1(P \uplus C^{n+1})$ be defined by

– $\eta(i.\psi) = i.\psi$ if $i.\psi \in \mathsf{D}_0(P \uplus C^n)$;
– $\eta(i.\psi) = i.\psi'$ otherwise,

where ψ' is obtained from ψ by replacing each innermost communication subformula $j.\varphi \in \mathsf{D}_0(P \uplus C^n)$ occurring in the scope of k by $c(k, j.\varphi)$.

Definition 11. The translation function $\alpha : \mathsf{D}_1(P) \to \mathsf{D}_0(P \uplus C)$ is defined by

– $\alpha(i.\psi) = i.\psi$ if $i.\psi \in \mathsf{D}_0(P \uplus C)$;
– $\alpha(i.\psi) = \alpha(\eta(i.\psi))$ otherwise.

Proposition 12. *The translation $\alpha : \mathsf{D}_1(P) \to \mathsf{D}_0(P \uplus C)$ as given above is well defined.*

Proof. Each application of η reduces the maximum depth of nesting communication subformulae. The result then follows from the fact that formulae without communication subformulae must be in D_0. □

With respect to interpretation structures, it is obvious that a life cycle over P may be obtained from a life cycle over $P \uplus C$ by simply omitting the C symbols from the labelling. That is, for each $i \in I$ and each $e \in Lv_i$, we just consider $\lambda_i(e) \cap P_i$. This is enough to guarantee that our translation is complete.

We prove this in the next chapter, and also that the translation is sound. This is the case because, given any life cycle over P, there is a unique way of extending it to a life cycle over $P \uplus C$ that satisfies Γ.

5 Soundness and completeness

In order to prove soundness and completeness of the reduction translation from D_1 to D_0, we get into the institutional framework as defined by [GB92] and define our translation as a simple map of institutions as defined by [Mes89]. Our soundness and completeness results follow from well known results about institutions and their maps.

We use the notion of an (elementary) institution. For the purpose of this paper, morphisms between models are not relevant. Just recall that *Set* denotes the category of sets and functions, and *Cls* denotes the category of classes and functions. For an introduction into these and other category theoretic issues, we refer to [Mac71]. For what follows, we recall the definition of an (elementary) institution.

Definition 13. An *(elementary) institution* is a tuple $(\mathit{Sig}, \mathit{Sen}, \mathit{Mod}, \models)$ where

- *Sig* is a category (of signatures),
- $\mathit{Sen} : \mathit{Sig} \to \mathit{Set}$ is a (syntax) functor,
- $\mathit{Mod} : \mathit{Sig}^{op} \to \mathit{Cls}$ is a (semantics) functor,
- $\models$ is a family of (satisfaction) relations $\{\models_\Sigma \subseteq \mathit{Mod}(\Sigma) \times \mathit{Sen}(\Sigma)\}_{\Sigma \in |\mathit{Sig}|}$,

such that the following (satisfaction) condition holds for each signature morphism $\sigma : \Sigma \to \Sigma'$, $\varphi \in \mathit{Sen}(\Sigma)$ and each model $m' \in \mathit{Mod}(\Sigma')$,

$$m' \models_{\Sigma'} \mathit{Sen}(\sigma)(\varphi) \textit{ iff } \mathit{Mod}(\sigma)(m') \models_\Sigma \varphi.$$

Both our logics D_0 and D_1 can be seen as institutions. Both share the same category *Sig* whose objects are system signatures with morphisms $\sigma : (I, \{P_i\}_{i \in I}) \to (J, \{Q_j\}_{j \in J})$ such that $I \subseteq J$ and $\sigma = \{\sigma_i : P_i \to Q_i\}_{i \in I}$ is a family of functions.

As for the syntax functor Sen_1 of D_1, it assigns to each signature P the set $Sen_1(P) = \mathsf{D}_1(P)$ of its formulae. For each signature morphism $\sigma : P \to Q$, $Sen_1(\sigma) : \mathsf{D}_1(P) \to \mathsf{D}_1(Q)$ is the obvious translation replacing each symbol $p \in P_i$ in a formula by its image $\sigma_i(p) \in Q_i$. The syntax functor Sen_0 of D_0 is a restriction of Sen_1.

The two logics also share the same model functor *Mod*. To each signature P, *Mod* assigns the class of all distributed life cycles over P. As for signature morphisms $\sigma : P \to Q$, $Mod(\sigma)$ reduces each distributed life cycle L' over Q to a distributed life cycle L over P by defining $L_i = (Lv'_i, \to'^{*}_i, \sigma_i^{-1} \circ \lambda'_i)$. This means that the event structure is unchanged, and the labels are restricted to those in P.

The satisfaction relations for D_0 and D_1 are given by $\models_{P,0}$ and $\models_{P,1}$, respectively, cf. definitions 6 and 7.

Proposition 14. *Both* $\mathsf{D}_0 = (Sig, Sen_0, Mod, \models_0)$ *and* $\mathsf{D}_1 = (Sig, Sen_1, Mod, \models_1)$ *constitute (elementary) institutions.*

Proof. In both cases, the satisfaction condition is straightforward and may be proved by induction on the structure of formulae. Since translation of formulae along a signature morphism σ just amounts to replacing each $p \in P_i$ by $\sigma_i(p)$, it is enough to point out the fact that $p \in \sigma_i^{-1}(\lambda'_i(e))$ iff $\sigma_i(p) \in \lambda'_i(e)$. □

Given any institution $(Sig, Sen, Mod, \models)$, it is straightforward to define a category *Pres* of theory presentations whose objects are pairs of the form (Σ, Δ) where $\Delta \subseteq Sen(\Sigma)$, and a morphism $\sigma : (\Sigma, \Delta) \to (\Sigma', \Delta')$ consists of a signature morphism $\sigma : \Sigma \to \Sigma'$ such that $Sen(\sigma)(\Delta) \subseteq \Delta'$. It is obvious to define a forgetful functor $S : Pres \to Sig$ which only keeps the signature from a theory presentation. And it is also obvious to promote the functor $Mod : Sig^{op} \to Cls$ to a functor $Mod : Pres^{op} \to Cls$ by defining $Mod((\Sigma, \Delta)) = \{m \in Mod(\Sigma) : m \models_\Sigma \delta,\ \textit{for every}\ \delta \in \Delta\}$. The satisfaction condition of the institution guarantees that this functor is well defined.

With these auxiliary definitions, we can make precise the notion of a simple map of institutions.

Definition 15. A *simple map* from institution $(Sig, Sen, Mod, \models)$ to institution $(Sig', Sen', Mod', \models')$ is a triple (Φ, α, β) where

- $\Phi : Sig \to Pres'$ is a functor;
- $\alpha : Sen \Rightarrow Sen' \circ S' \circ \Phi$ is a natural transformation;
- $\beta : Mod' \circ \Phi^{op} \Rightarrow Mod$ is a natural transformation,

such that, for each $\Sigma \in |Sig|$, $\varphi \in Sen(\Sigma)$ and $m' \in Mod'(\Phi(\Sigma))$, the following (structurality) condition holds

$$m' \models'_{S'(\Phi(\Sigma))} \alpha_\Sigma(\varphi) \text{ iff } \beta_\Sigma(m') \models_\Sigma \varphi.$$

We now proceed to state the reduction presented above in terms of a simple map from the institution D_1 to the institution D_0.

Clearly, Φ shall associate to each system signature P the presentation $(P \uplus C, \Gamma)$ as defined in the previous section. It is obvious to extend this definition to a functor.

α corresponds to the iterative reduction function as defined in the previous section, also called α there.

As explained in the previous section, one can easily obtain a life cycle over P from a given life cycle over $P \uplus C$ by forgetting the symbols in C. We denote this transformation by β.

For both α and β, it is straightforward to check that they are natural transformations.

Lemma 16. *Let $c(i, j.\varphi) \in C_i$ and L a life cycle over $P \uplus C$ such that $L \models_{P \uplus C,0} \gamma$, for every $\gamma \in \Gamma$. Then we have, for each $e \in Lv_i$,*

$$L, e \models^i_{P \uplus C,0} c(i, j.\varphi) \text{ iff } L, e \models^i_{P \uplus C,0} j.\varphi.$$

Proof. Let $L, e \models^i_{P \uplus C,0} c(i, j.\varphi)$. Since $i.(c(i, j.\varphi) \Rightarrow j.c(i, j.\varphi)) \in \Gamma$, necessarily also $L, e \models^i_{P \uplus C,0} j.c(i, j.\varphi)$. Therefore, $e \in Lv_j$ and $L, e \models^j_{P \uplus C,0} c(i, j.\varphi)$. But $j.(c(i, j.\varphi) \Leftrightarrow (@i \wedge \varphi)) \in \Gamma$ thus $L, e \models^j_{P \uplus C,0} \varphi$ and therefore $L, e \models^i_{P \uplus C,0} j.\varphi$.

On the other hand, let $L, e \models^i_{P \uplus C,0} j.\varphi$. Necessarily, $e \in Lv_j$ and $L, e \models^j_{P \uplus C,0} \varphi$. Reusing $j.(c(i, j.\varphi) \Leftrightarrow (@i \wedge \varphi)) \in \Gamma$, from the fact that $e \in Lv_i$ it follows that $L, e \models^j_{P \uplus C,0} c(i, j.\varphi)$. But $j.(c(i, j.\varphi) \Rightarrow i.c(i, j.\varphi)) \in \Gamma$, so $L, e \models^j_{P \uplus C,0} i.c(i, j.\varphi)$ and thus $L, e \models^i_{P \uplus C,0} c(i, j.\varphi)$. □

Proposition 17. *The triple $(\Phi, \alpha, \beta) : \mathsf{D}_1 \to \mathsf{D}_0$ constitutes a simple map of (elementary) institutions.*

Proof. The structurality condition is an immediate consequence of the lemma above. In fact, let $i.\varphi \in \mathsf{D}_1(P)$ and L a life cycle over $P \uplus C$ such that $L \models_{P \uplus C,0} \gamma$, for every $\gamma \in \Gamma$. To prove that $L \models_{P \uplus C,0} \alpha(i.\varphi)$ iff $L \models_{P,1} i.\varphi$, it is enough to proceed by induction on the maximum nesting depth of communication subformulae in $i.\varphi$. In each step of α, the innermost communication subformulae are replaced by symbols with equivalent meaning as proved in the lemma above. Thus the overall meaning of $i.\varphi$ is maintained. □

As promised, we now prove that D_0 and D_1 are equally expressible by using well known results about institutions and simple maps.

Given any institution $(\mathit{Sig}, \mathit{Sen}, \mathit{Mod}, \models)$, we can introduce semantic entailment in the usual way. For each signature $\Sigma \in |\mathit{Sig}|$, we define $\models_\Sigma \subseteq \mathcal{P}(\mathit{Sen}(\Sigma)) \times \mathit{Sen}(\Sigma)$ by $\Delta \models_\Sigma \varphi$ iff for each $m \in \mathit{Mod}(\Sigma)$, $m \models_\Sigma \varphi$ whenever $m \models_\Sigma \delta$ for every $\delta \in \Delta$. Of course, although we use the same symbol $\models_\Sigma$ both for satisfaction and entailment, it is easy to distinguish between the two concepts in any given context.

The following proposition is drawn from [Mes89], cf. Lemma 28 there.

Proposition 18. *Let $(\Phi, \alpha, \beta) : (\mathit{Sig}, \mathit{Sen}, \mathit{Mod}, \models) \to (\mathit{Sig}', \mathit{Sen}', \mathit{Mod}', \models')$ be a simple map of institutions. Then, for each signature $\Sigma \in |\mathit{Sig}|$, $\Delta \subseteq \mathit{Sen}(\Sigma)$ and letting $\Phi(\Sigma) = (\Sigma', \Gamma')$, the following condition holds*

$$\mathit{Mod}((\Sigma, \Delta)) \supseteq \beta_\Sigma(\mathit{Mod}'((\Sigma', \alpha_\Sigma(\Delta) \cup \Gamma'))).$$

Consequently, for each $\varphi \in \mathit{Sen}(\Sigma)$, we have that

$$\Delta \models_\Sigma \varphi \text{ implies } \alpha_\Sigma(\Delta) \cup \Gamma' \models'_{\Sigma'} \alpha_\Sigma(\varphi).$$

This result immediately applies to our reduction map from D_1 to D_0.

Corollary 19. *Let $\Delta \subseteq \mathsf{D}_1(P)$. Then,*

$$\mathit{Mod}((P, \Delta)) \supseteq \beta(\mathit{Mod}((P \uplus C, \alpha(\Delta) \cup \Gamma))).$$

Consequently, for each $i.\varphi \in \mathsf{D}_1(P)$,

$$\Delta \models_{P,1} i.\varphi \text{ implies } \alpha(\Delta) \cup \Gamma \models_{P \uplus C, 0} \alpha(i.\varphi).$$

We may see this property as stating the completeness of our reduction map: everything that can be obtained from the original D_1 specification can also be obtained from its translation into D_0. Of course, this is a desirable property but it is only meaningful if we can also guarantee soundness. For that purpose we need another general result about simple maps of institutions, cf. [CM97].

Proposition 20. *Let $(\Phi, \alpha, \beta) : (\mathit{Sig}, \mathit{Sen}, \mathit{Mod}, \models) \to (\mathit{Sig}', \mathit{Sen}', \mathit{Mod}', \models')$ be a simple map of institutions such that each $\beta_\Sigma : \mathit{Mod}'(\Phi(\Sigma)) \to \mathit{Mod}(\Sigma)$ is surjective. Then, for each signature $\Sigma \in |\mathit{Sig}|$, $\Delta \subseteq \mathit{Sen}(\Sigma)$ and letting $\Phi(\Sigma) = (\Sigma', \Gamma')$, the following condition holds*

$$\mathit{Mod}((\Sigma, \Delta)) = \beta_\Sigma(\mathit{Mod}'((\Sigma', \alpha_\Sigma(\Delta) \cup \Gamma'))).$$

Consequently, for each $\varphi \in \mathit{Sen}(\Sigma)$, we have that

$$\Delta \models_\Sigma \varphi \text{ iff } \alpha_\Sigma(\Delta) \cup \Gamma' \models'_{\Sigma'} \alpha_\Sigma(\varphi).$$

In order to obtain our soundness result we therefore have to show that our model translation fulfils the surjectivity property stated above.

Proposition 21. *If L is a distributed life cycle over P then it is possible to extend L to $P \uplus C$ in such a way that $L \models_{P \uplus C,0} \Gamma$.*

Proof. Suppose that $L = \{L_i\}_{i \in I}$ and that each $L_i = (Lv_i, \rightarrow_i^*, \lambda_i)$. The proof proceeds by inductively extending L to L^n by extending its labelling function to $\lambda_i^n : Lv_i \rightarrow 2^{P \uplus C^n}$. Since each $C_i^0 = \emptyset$, we take $\lambda_i^0 = \lambda_i$. In each step, for each new symbol $c(i, j.\varphi) \in C^{n+1} \setminus C^n$, proceed as follows

- first, taking into account the definitional axiom $j.(c(i, j.\varphi) \Leftrightarrow (@i \wedge \varphi))$, build $\lambda_j^{n+1}(e)$ by extending $\lambda_j^n(e)$ with $c(i, j.\varphi)$ iff $L^n, e \models^j_{P \uplus C^n,0} (@i \wedge \varphi)$;
- then, taking into account the sharing axioms $j.(c(i, j.\varphi) \Rightarrow i.c(i, j.\varphi))$ and $i.(c(i, j.\varphi) \Rightarrow j.c(i, j.\varphi))$, build $\lambda_i^{n+1}(e)$ by extending $\lambda_i^n(e)$ with $c(i, j.\varphi)$ iff $e \in Lv_j$ and $c(i, j.\varphi) \in \lambda_j^{n+1}(e)$.

Clearly, if we take the limit of this definition, we obtain a life cycle L over $P \uplus C$ that satisfies Γ. □

In fact, it turns out that our model translation is an isomorphism. The uniqueness of the extension presented above can be easily inferred from the proof.

We can now conclude that our reduction map is sound. So we have proved the main result of this paper: any D_1 specification Δ is equivalent to the D_0 specification $\alpha(\Delta) \cup \Gamma$, in the sense that they have exactly the same models once you forget about the new symbols in the labelling.

Theorem 22. *Let $\Delta \subseteq \mathsf{D}_1(P)$. Then we have*

$$\mathit{Mod}((P, \Delta)) = \beta(\mathit{Mod}((P \uplus C, \alpha(\Delta) \cup \Gamma))).$$

As for entailment, an immediate consequence is the following.

Corollary 23. *For each $i.\varphi \in \mathsf{D}_1(P)$, we have*

$$\Delta \models_{P,1} i.\varphi \ \textit{ iff } \ \alpha(\Delta) \cup \Gamma \models_{P \uplus C,0} \alpha(i.\varphi).$$

We can say even more: the same holds if we do not translate $i.\varphi$ on the right-hand side, and consider entailment in $\mathsf{D}_1(P \uplus C)$.

Corollary 24. *Let $\Delta \subseteq \mathsf{D}_1(P)$ and $i.\varphi \in \mathsf{D}_1(P)$. Then we have*

$$\Delta \models_{P,1} i.\varphi \ \textit{ iff } \ \alpha(\Delta) \cup \Gamma \models_{P \uplus C,1} i.\varphi.$$

Proof. We are, of course, using the inclusions $\mathsf{D}_1(P) \subseteq \mathsf{D}_1(P \uplus C)$ and $\mathsf{D}_0(P \uplus C) \subseteq \mathsf{D}_1(P \uplus C)$. The result immediately follows from the fact that $\mathit{Mod}((P, \Delta)) = \beta(\mathit{Mod}((P \uplus C, \alpha(\Delta) \cup \Gamma)))$ and by noting that β just forgets about C symbols, and so does not lose any information which may be relevant for evaluating $i.\varphi$ that is written using only symbols in P. □

Of course, despite the fact that C is in general a rather large set, in practice, given a finite specification Δ in D_1 only a finite part of C is necessary. In that case, the reduction procedure can be made effective and our equal-expressibility result still applies.

6 Concluding remarks

The main result presented in this paper is to show how—and prove that—D_1 specifications may be automatically translated to D_0 specifications in a sound and complete way. D_1 is more suitable for specification, and D_0 may be more easily implemented, e.g., translated to C^{++}. This suggests a practical method to use D_1 for high-level planning and specification, especially of communication aspects in distributed information systems. The results may help to make implementation platforms like CORBA easier accessible.

We experimented with D_1 specifications in the framework of a TROLL application project (cf. section 2 or [ECSD98], respectively, for an extended example). The experiences are encouraging, but further work is needed before D_1 specification can be part of a well-understood methodology.

A thorougher study of D_0 and D_1 is still needed, many theoretical issues with practical impact remain to be studied. Among these, we have to analyze the expressive power of our logics, search for complete axiomatizations and, most of all, look for decidability results. We may easily proceed to establish an encoding of D_0 or D_1 into plain propositional linear temporal logic, again following the institutional or general logics approach. Syntactically, this amounts to turning the localities into propositional symbols together with suitably changing the localized temporal operators. Semantically, a linear model corresponds in an obvious way to a sequentialization of a distributed life cycle. Using this translation, we immediately get a decidability result for the validity problem in D_0 and D_1 by capitalizing on the well known decidability results for propositional linear temporal logic. A careful analysis of the complexity of the translation also gives us an upper bound on the complexity of the decision of validities in D_0 and D_1. Moreover, the above mentioned correspondence between linear and distributed models embodies the fact that the distributed logics are "trace consistent". That is, satisfiability is invariant with respect to different possible sequentializations of the same distributed model [Pel93, Thi95b]. Such a result, then, paves

the way to an efficient use (in our setting) of well-known partial-order verification techniques such as model-checking [Val90, GW94, BGS98]. This approach also hints to the possibility of basing a decision procedure for our logics on distributed tableaux methods, in a spirit similar to the one found in [Mas98].

Moreover, a deeper analysis of these distributed logics in terms of general mechanisms of logical combination is also in progress [CSS99, SSC99, SSC97a, SSC97b]. We expect that this effort may help on several of the open tasks previously mentioned, since general transfer theorems for combined logics often exist [BdR97, FG92, FG96, Gab96b, Gab96a]. Thus, we expect that an axiomatization, decision procedure, etc., for these logics can be obtained by suitably combining the axiomatizations, decision procedures, etc., for the local logics. In particular, we expect to obtain a complete axiomatization for a small extension of D_1 by using the idea of trios from [BdR97] and appropriately axiomatizing the type of combined structure we are dealing with. Please note that a distributed life cycle in our sense (let us say, for two objects) consists of two linear models put together in a very particular way. All we need is to be able to express the definition of distributed life cycle in our logical distributed language. This idea is consistent with a remark by R. Ramanujam in [Ram96] about existing axiomatizations for certain versions of n-agent logic: "What is particularly interesting about the axiom systems is that we can present them in two layers: at the local level, we have a system for each agent, and at the global level, a system to put them together".

At this point, a word of justification is due for getting into the institutional framework. Clearly, we could have stated and proved everything about D_0 and D_1 directly. Even if we were completely unaware of institutions. Or knowing of their existence and importance, we could as well have omitted them from the presentation. However, we are convinced that the practical importance of general results such as those made available by nearly two decades of research in the field of institutions is not yet fully appreciated. And this paper is a mere example of their usefulness. In fact, not only our logics are much more concisely presented as institutions (although harder to grasp, technically speaking) but also our reduction mechanism, already outlined in [ECSD98], is stated in a much more elegant way. And what is more, we manage to prove the equal-expressibility of D_0 and D_1 in a detailed way, by just applying suitable well-known results about simple maps of institutions.

Another interesting problem, related to the expressive power of our logics, is their relationship with other versions of n-agent logics. It is obvious that there is no way in our logics to talk about the last communication with another object, as in [LMRT91], or of asserting that for sure some property must hold for another object even if we are not currently communicating with it, as in [Ram96]. However, it would be quite easy to extend our logics

with a since $\mathcal{S}$ operator, with a meaning dual to that of until $\mathcal{U}$, but referring to the past instead of the future. In that case, the last communication with another agent could be expressed as an abbreviation by

$$i.(\mathrm{LAST}_j\ \varphi) \equiv i.(j.\varphi \vee ((\neg\, @j) \wedge ((\neg\, @j)\ \mathcal{S}\ j.\varphi))).$$

That is, "i knows that φ held for j the last time they talked" is given by "either they are communicating now and φ holds for j, or that is not the case now but it happened sometime in the past and i and j have not talked since then".

Even more interestingly, present knowledge about another agent can be expressed by

$$i.(\mathrm{NOW}_j\ \varphi) \equiv i.(j.\varphi \vee ((\neg\, @j) \wedge ((\neg\, @j)\ \mathcal{S}\ j.(\varphi\ \mathcal{U}\ @i)))).$$

That is, "i is sure that φ must presently hold for j" is given by "either they are communicating now and φ holds for j, or that is not the case now but sometime ago j ensured that she would have φ until she talked to i again, and they have not talked ever since".

What is more, all the work on reduction presented in this paper seems to carry on smoothly to this extension with past operators. Moreover, results like decidability or completeness also seem easy to extend, once established for future-only versions.

Acknowledgements. Special thanks are due to Narciso Marti-Oliet who gave us the hint to try maps of logics. We also thank the coauthors of [ECSD98], Amilcar Sernadas and Grit Denker, who cooperated to develop the ideas that are worked out in this paper. We appreciate being able to build on institutions, general logic and especially maps of logic, thanks to the seminal work of Joseph Goguen, Rod Burstall and Jose Meseguer. The referees gave valuable hints and suggestions for improving the presentation of the paper. Last but not least, we acknowledge the support and inspiring atmosphere in our groups at TU Braunschweig and IST Lisbon.

References

[BAMP81] M. Ben-Ari, Z. Manna, A. Pnueli: The temporal logic of branching time. 8th ACM Symposium on Principles of Programming Languages, Williamsburg, VA, p 164–176, 1981. Also, Acta Informatica, 20(3): 207–226 (1983)

[BdR97] P. Blackburn, M. de Rijke: Zooming in, zooming out. Journal of Logic, Language and Information, 6: 5–31 (1997)

[BGS98] M. Benerecetti, F. Giunchiglia, L. Serafini: Model checking multiagent systems. Journal of Logic and Computation, 8(3): 373–400 (1998)

[BKP84] H. Barringer, R. Kuiper, A. Pnueli: Now you may compose temporal logic specifications. In Proc. 16th ACM Symp. Theory of Computing, p 51–63, ACM Press, 1984

[CE81] E. Clarke, E. Emerson: Design and synthesis of synchronization skeletons using branching time temporal logic. In Workshop on Logics of Programs, p 52–71, LNCS 131, 1981

[CM97] M. Cerioli, J. Meseguer: May I borrow your logic? (Transporting Logical Structures along Maps) Theoretical Computer Science, 173(2): 311–347 (1997)

[CSS99] C. Caleiro, C. Sernadas, A. Sernadas: Parameterisation of logics. Recent Developments in Algebraic Development Techniques, Selected Papers, J.L. Fiadeiro (ed.) LNCS 1589, p 48-62, Berlin: Springer 1999

[DDPW94] E. Dubois, Ph. Du Bois, M. Petit, S. Wu: Albert: A formal agent-oriented requirements language for distributed composite systems. In: P. Hartel, S. Saake, E. Dubois (eds.) Proc. Workshop on Formal Methods for Information System Dynamics (CAiSE'94), p 25–39, 1994

[DH97] G. Denker, P. Hartel: TROLL – An Object Oriented Formal Method for Distributed Information System Design: Syntax and Pragmatics. Informatik-Berichte 97-03, TU Braunschweig 1997

[DRCS97] G. Denker, J. Ramos, C. Caleiro, A. Sernadas: A linear temporal logic approach to objects with transactions. In AMAST97, volume LNCS vol. 1349. Berlin: Springer 1997

[ECSD98] H.-D. Ehrich, C. Caleiro, A Sernadas, G. Denker: Logics for Specifying Concurrent Information Systems. In: J. Chomicki, G. Saake, editors, Logics for Databases and Information Systems, p 167–198. Dordrecht: Kluwer Academic Publishers, 1998

[EH83] E. Emerson, J. Halpern: "sometimes" and "not never" revisited: on branching versus linear time temporal logic. 10th ACM Symposium on Principles of Programming Languages, Austin, p 127–140, 1983. Also, Journal of the ACM, 33(1): 151–178 (1986)

[EH96] H.-D. Ehrich, P. Hartel: Temporal specification of information systems. In: A. Pnueli, H. Lin (eds.) Proc. Int. Workshop in Honor of Chih-Sung Tang, World Scientific, Singapore, p 43–70, 1996

[Ehr99] H.-D. Ehrich: Object Specification. In: E. Astesiano, H.-J. Kreowski, B. Krieg-Brückner (eds.) Algebraic Foundations of Systems Specification, Chapter 12, p 435–465. IFIP State-of-the-Art Reports, Berlin: Springer 1999

[Eme90] E. Emerson: Temporal and modal logic. In: J. van Leeuwen (ed.) Handbook of Theoretical Computer Science, p 997–1072. Amsterdam: Elsevier, 1990

[ES95] H.-D. Ehrich, A. Sernadas: Local Specification of Distributed Families of Sequential Objects. In: E. Astesiano, G. Reggio, A. Tarlecki (eds.) Recent Trends in Data Types Specification, Proc. 10th ADT Workshop/5th COMPASS Workshop, Selected papers, LNCS 906, p 219–235. Berlin: Springer 1995

[Esp93] Espírito Santo Data Informática (ESDI), Lisbon. OBLOG Users Manual, 1993. Supplied with OBLOG-CASE v1.0 product kit.

[FG92] M. Finger, D. Gabbay: Adding a temporal dimension to a logic system. Journal of Logic, Language and Information, 1: 203–233 (1992)

[FG96] M. Finger, D. Gabbay: Combining temporal logic systems. Notre Dame Journal of Formal Logic, 37(2): 204–232 (1996)

[FHMV95] R. Fagin, J. Halpern, Y. Moses, M. Vardi: Reasoning about knowledge. MIT Press, 1995

[FL79] M. Fischer, R. Ladner: Propositional dynamic logic of regular programs. Journal of Computation and System Sciences, 18: 194–211 (1979)

[Flo67] R. Floyd: Assigning meanings to programs. In: J. Schwartz (ed.) Mathematical Aspects of Computer Science - Proceedings of the Symposium on Applied Mathematics, volume 19, p 19–32. American Mathematical Society, 1967

[FM92] J. Fiadeiro, T. Maibaum: Temporal theories as modularization units for concurrent systems specification. Formal Aspects of Computing, 4: 239–272 (1992)

[Fra86] N. Francez: Fairness. Berlin Heidelberg New York: Springer 1986

[FW93] R.B. Feenstra, R. Wieringa: Lcm 3.0: A language for describing conceptual models – syntax definition. Technical report, Vrije Universiteit Amsterdam, 1993

[Gab96a] D. Gabbay: Fibred semantics and the weaving of logics: part 1. Journal of Symbolic Logic, 61(4): 1057–1120 (1996)

[Gab96b] D. Gabbay: An overview of fibred semantics and the combination of logics. In: F. Baader, K. Schulz (eds.) Frontiers of Combining Systems, p 1–55. Dordrecht: Kluwer, 1996

[Gab98] D. Gabbay: Fibring Logics. Oxford University Press, 1998

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

[GK97] A. Grau, M. Kowsari: A validation system for object-oriented specification of information systems. In: R. Manthey, V. Wolfenhagen (eds.) Advances in Databases and Information Systems, Proc. 1st East-European Symposium ADBIS'97, St. Petersburg, 2-5 Sep 97., Electronic Workshops in Computing, Springer, 1997

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

[Gol92] R. Goldblatt: Logics of Time and Computation. CSLI, 1992

[GPSS80] D. Gabbay, A. Pnueli, S. Shelah, J. Stavi: On the temporal analysis of fairness. In: Proceedings of the 7th ACM Symposium on Principles of Programming Languages, 1980

[GS95] J.A. Goguen, A. Socorro: Module composition and system design for the object paradigm. Journal of Object oriented Programming, 7(14), 1995

[GW88] J.A. Goguen, T. Winkler: Introducing OBJ3. Technical report, SRI International, 1988

[GW94] J. Godefroid, P. Wolper: A partial approach to model checking. Information and Computation, (110): 305–326 (1994)

[Har79] D. Harel: First-Order Dynamic Logic. LNCS 68 Berlin: Springer 1979

[Har97] P. Hartel: Konzeptionelle Modellierung von Informationssystemen als verteilte Objektsysteme. Reihe DISDBIS. infix-Verlag, Sankt Augustin, 1997

[HDK+97] P. Hartel, G. Denker, M. Kowsari, M. Krone, and H.-D. Ehrich: Information systems modelling with TROLL formal methods at work. Information Systems, 22(2-3): 79–99 (1997)

[HJ95] P. Hartel, R. Jungclaus: Modeling Business Processes over Objects. Int. Journal of Cooperative Information Systems, 4(2): 165–188 (1995)

[Hoa69] C. A. R. Hoare: An axiomatic basis for computer programming. Communications of the ACM, 12(10): 576–580 (1969)

[Hoa85] C. A. R. Hoare: Communicating sequential processes. Englewood Cliffs, NJ: Prentice-Hall, 1985

[JSHS96] R. Jungclaus, G. Saake, T. Hartmann, C. Sernadas: TROLL – A Language for Object-Oriented Specification of Information Systems. ACM Transactions on Information Systems, 14(2): 175–211 (1996)

[JWH+94] R. Jungclaus, R. J. Wieringa, P. Hartel, G. Saake, T. Hartmann: Combining TROLL with the Object Modeling Technique. In: *Innovationen bei Rechen- und Kommunikationssystemen.* GI-Fachgespräch FG 1: Integration von semiformalen und formalen Methoden für die Spezifikation von Software. Informatik aktuell, pp 35-42, Berlin: Springer, 1994

[GK+98] A. Grau, J. Küster Filipe, M. Kowsari, S. Eckstein, R. Pinger, H.-D. Ehrich: The TROLL Approach to Conceptual Modelling: Syntax, Semantics and Tools. In Proc. of the 17th Int. Conference on Conceptual Modeling (ER'98), Singapore, 1998

[KKH+96] M. Krone, M. Kowsari, P. Hartel, G. Denker, and H.-D. Ehrich: Developing an Information System Using TROLL: an Application Field Study. In: P. Constantopoulos, J. Mylopoulos, Y. Vassiliou (eds.) Proc. 8th Int. Conf. on Advanced Information Systems Engineering (CAiSE'96), LNCS 1080, p 136–159, Berlin: Springer, 1996

[KP87] S. Katz, D. Peled: Interleaving set temporal logic. In 6th ACM Symposium on Principles of Distributed Computing, p 178–190, 1987

[KR94] P. Krasucki, R. Ramanujam: Knowledge and the ordering of events in distributed systems. In: Proc. Theoretical Aspects of Reasoning About Knowledge, p 267–283, Morgan Kaufmann, 1994

[Kus97] J. Küster Filipe: Putting synchronous and asynchronous object modules together. Tech. Report 97-05, TU Braunschweig, 1997

[Lam77] L. Lamport: Proving the correctness of multiprocess programs. IEEE Transactions on Software Engineering, 3(2): 125–143 (1977)

[LMRT91] K. Lodaya, M. Mukund, R. Ramanujam, 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

[LPRT95] K. Lodaya, R. Parikh, R. Ramanujam, P. Thiagarajan: A logical study of distributed transition systems. Information and Computation, 119(1): 91–118 (1995)

[LRT92] K. Lodaya, R. Ramanujam, P.S. Thiagarajan: Temporal logics for communicating sequential agents: I. International Journal of Foundations of Computer Science, 3: 117–159 (1992)

[LT87] K. Lodaya, P.S. Thiagarajan: A modal logic for a subclass of event structures. In Th. Ottmann (ed.) Proc. 14th Int. Colloq. on Automata, Languages and Programming, LNCS 267, p 290–303, Berlin: Springer, 1987

[Mac71] S. MacLane: Categories for the working mathematician. Berlin: Springer, 1971

[Mas98] F. Massacci: Tableau methods for formal verification of multi-agent distributed systems. Journal of Logic and Computation, 8(3): 373–400 (1998)

[Mes89] J. Meseguer: General Logic. In: H.-D. Ebbinghaus et al (ed.) Logic Colloquium '87, p 275–329. Amsterdam: North-Holland, 1989

[Mes92] J. Meseguer: Conditional rewriting as a unified model of concurrency. Theoretical Computer Science, 96(1): 73–156 (1992)

[Mes93] J. Meseguer: A Logical Theory of Concurrent Objects and its Realization in the Maude Language. In: G. Agha, P. Wegner, A. Yonezawa (eds.) Research Directions in Concurrent Object-Oriented Programming, p 314–390. The MIT Press, 1993

[Mil80] R. Milner: A calculus of communicating systems. LNCS 92, Berlin: Springer Verlag, 1980

[MP92] Z. Manna, A. Pnueli: The Temporal Logic of Reactive and Concurrent Systems. New York: Springer, 1992

[Pel87] D. Peleg: Concurrent dynamic logic. Journal of the ACM, 34(2): 450–479 (1987)

[Pel93] D. Peleg: All from one and one from all: on model checking using representatives. LNCS 697, Springer, p 409–423, 1993

[Pen88] W. Penczek: A temporal logic for event structures. Fundamenta Informaticae, 11(3): 297–326 (1988)

[Pen95] W. Penczek: Branching time and partial order in temporal logics. In: L. Bolc, A. Szalas (eds.) Time and Logic: A Computational Approach, p 179–228, UCL Press, 1995

[Pnu77] A. Pnueli: The temporal logic of programs. In Proc. 19th Annual Symposium on Foundations of Computer Science, p 46–57, IEEE Computer Society. New York, 1977

[Pra86] V.R. Pratt. Modeling Concurrency with Partial Orders. International Journal of Parallel Programming, 15(1): 33–71 (1986)

[PW84] S. Pinter, P. Wolper: A temporal logic for reasoning about partially ordered computations. In: 3rd ACM Symposium on Principles of Distributed Computing, p 28–37, 1984

[Ram96] R. Ramanujam: Locally linear time temporal logic. In: Proc. 11th Annual IEEE Symposium on Logics in Computer Science, p 118–127, IEEE Computer Society. New York, 1996

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

[Rei92] W. Reisig: Elements of a temporal logic coping with concurrency. Technical report, Institut für Informatik, Technische Universität München, 1992

[RS92] L. Rapanotti, A. Socorro: Introducing FOOPS. Technical report, PRG-TR-28-92, Programming Research Group, Oxford University Computing Laboratory, 1992

[Ser80] A. Sernadas: Temporal aspects of logical procedure definition. Information systems, 5: 167–187 (1980)

[SFSE89] A. Sernadas, J. Fiadeiro, C. Sernadas, H.-D. Ehrich: The basic building blocks of information systems. In: E. Falkenberg, P. Lindgreen (eds.) Information Systems Concepts: An In-Depth Analysis, p 225–246, Amsterdam: North-Holland 1989

[SHJE94] G. Saake, T. Hartmann, R. Jungclaus, H.-D. Ehrich: Object-oriented design of information systems: TROLL language features. In: J. Paredaens, L. Tenenbaum (eds.) Advances in Database Systems, Implementations and Applications. Wien: Springer, CISM Courses and Lectures no. 347, 1994

[SR94] A. Sernadas, J. Ramos: The GNOME language: Syntax, semantics and calculus. Technical report, Tech. Report, Instituto Superior Técnico, Lisboa, 1994

[SSC95] A. Sernadas, C. Sernadas, J.F. Costa: Object specification logic. Journal of Logic and Computation, 5: 603–630 (1995)

[SSC97a] A. Sernadas, C. Sernadas, C. Caleiro: Synchronization of logics. Studia Logica, 59(2): 217–247 (1997)

[SSC97b] A. Sernadas, C. Sernadas, C. Caleiro: Synchronization of logics with mixed rules: Completeness preservation. In: AMAST97, LNCS 1349. Berlin Heidelberg New York: Springer, 1997

[SSC98] A. Sernadas, C. Sernadas, C. Caleiro: Denotational semantics of object specification. Acta Informatica 35, 729–773 (1998)

[SSC99] A. Sernadas, C. Sernadas, C. Caleiro: Fibring of logics as a categorial construction. Journal of Logic and Computation 9(2), p 149–179 (1999)

[SSE87] A. Sernadas, C. Sernadas, H.-D. Ehrich: Object-oriented specification of databases: An algebraic approach. In: P. Hammerslay (ed.) Proc. 13th Int. Conf. on Very Large Databases, VLDB'87, p 107–116, Brighton, 1987. Palo Alto: Morgan-Kaufmann, 1987

[SSR96] A. Sernadas, C. Sernadas, J. Ramos: A temporal logic approach to object certification. Data and Knowledge Engineering, 19: 267–294 (1996)

[Thi94] P.S. Thiagarajan: A Trace Based Extension of Linear Time Temporal Logic. In: Proc. 9th annual IEEE Symposium on Logic in Computer Science, p 438–447, IEEE Computer Society Press, 1994

[Thi95a] P. Thiagarajan: Ptl over product state spaces. Technical report, School of Mathematics, SPIC Science Foundation, Madras, India, 1995

[Thi95b] P. Thiagarajan: A trace consistent subset of PTL. LNCS 962, Springer, 1995

[Val90] A. Valmari: A stubborn attack on state explosion. LNCS 531, Springer, p 156–165, 1990

[Win87] G. Winskel: Event Structures. In: W. Brauer, W. Reisig, G. Rozenberg (eds.) Petri Nets: Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986, Part II, Proc. Advanced Course, Bad Honnef, September 1986, p 325–392, LNCS 255, Springer, 1987

[Wol95] P. Wolper: On the relation of programs and computations to models of temporal logic. In: L. Bolc, A. Szalas (eds.) Time and Logic: A Computational Approach, p 131–178. UCL Press, 1995