in Formal Modeling: Actors, Open Systems and Biological Systems. LNCS 7000. pages 57–73. Springer. Stanford, USA. Aug. 2011 © Springer

Mathematical Models of Object-Based Distributed Systems

Carlos H. C. Duarte [⋆] [⋆⋆]

BNDES, Av. República do Chile 100, Rio de Janeiro, RJ, 20001-970, Brazil

Abstract. We propose an alternative characterisation of object-based distributed systems in terms of algebraic structures and topological spaces. Some examples are given in order to attest the adequacy of this approach to the subject. We also illustrate a method of transference of results from these mathematical theories that can further contribute to the advancement of distributed systems theory.

Keywords: Algebraic Structures; Topological Spaces; Distributed Systems; Software Development.

1 Introduction

Distributed systems are hard to design and understand because we lack intuition for them [24]. Although this was pointed out almost two decades ago, it appears that the theory and practice of distributed systems development has not evolved sufficiently since then so as to completely unravel the inherent complexity of the notion of distribution. On the contrary, diverse formalisms and technologies proliferate, usually adding more complexity to this problem.

A plausible and frequently adopted approach for obtaining a better intuition concerning distributed systems consists in developing an abstract and faithful representation of such systems, notably by adopting a class of mathematical structures that allows us to represent and analyse any property of interest. Here, such mathematical structures are called distributed system *formal models*.

It has been a longstanding tradition in Computer Science, and more generally in Logic, to adopt algebraic structures as the underlying mathematical entities against which the satisfiability of logically formulated properties is inspected [26]. The formal models usually adopted in distributed systems development are not different: (fair) transition systems [21], I/O automata [17], edge reversing graphs [5], event structures [19] and many others are defined as algebraic structures.

By no means we need to restrict ourselves to the algebraic character of mathematical structures in the study of distributed systems. In this paper, in particular, our aim is precisely to propose a novel characterisation of distributed systems in terms of a diverse mathematical theory: topological spaces. In fact, this approach is not entirely new:

[⋆] E-mail: carlos.duarte@computer.org, Web: http://chcduarte.webs.com

[⋆⋆] The definitions and results in this paper can be regarded as an attempt to obtain more general axiomatic theories than those developed by Carolyn Talcott and her colleages over the past two decades on the theory of asyncronous object-based distributed systems.

Alpern and Schneider provided a topological characterisation of safety and liveness properties of concurrent and distributed systems in [3], while Herlihy, Shavit, Saks and Zaharoglou gave in [14, 23] a topological characterisation of the class of decision problems that can be solved using asynchronous wait-free deterministic distributed shared memory computation. Here, our purpose is to widen this approach in order to cover not only distributed computing but also distributed systems development in general, including the specification and verification of their structural and behavioural properties.

The proposed characterisation of distributed systems yields a method of transference of results from the underlying mathematical theories. Frequently adopted by mathematicians, such method consists in a scheme to transpose verified results concerning a well established theory to another one which is still in development. In this paper, we also illustrate this method, which we believe can further contribute to the advancement of distributed systems theory.

We regard our algebraic and topological characterisation of distributed systems to be the main original contribution reported in this paper. The corresponding theoretical results are important towards establishing a framework for the analysis and simulation of distributed system behaviours, facilitating their comprehension. They also establish a general formal foundation for the definition of logical systems devoted to the compositional development of object-based distributed systems.

The remainder of the paper is organised as follows: Section 2 introduces the relevant distributed system notions; Section 3 presents the underlying algebraic structures that are used throughout the paper; while Section 4 develops a topological characterisation and analysis of distributed systems. We conclude the paper commenting on related and future work.

2 Distributed System Notions

Distributed systems can be identified in many different contexts. A computer network of a corporation can be regarded as a distributed system. Software applications providing support to electronic commerce, distance education and electronic government, through widely distributed computer networks such as the Internet, can also be considered as examples of this family of systems. Moreover, postal systems in which the dispatch, processing and delivery of letters is manually performed are inherently distributed.

It is very hard to propose an exact definition of distributed systems considering that our aim is to capture with such a definition the aforementioned examples. Consequently, we will consider sufficient throughout this text to say that a *distributed system* is a set of at least loosely coupled autonomous objects potentially situated at distinct localities.

Here, the term *object* corresponds to an abstraction, which can represent distinct types of entities, such as humans, intelligent agents, software components or processing units. The assumption that objects are at least *loosely coupled* captures, on the one hand, the intuition that completely disconnected sets of objects define in fact separated (sub-)systems, and, on the other hand, that fully coupled objects cannot present any autonomous behaviour.

The term *location* denotes a reference to the physical or virtual position of each object and is perhaps what fundamentally distinguishes our definition from others available in the literature. We find it convenient to presume that objects are situated (even if we frequently forget this), since, whenever this is identified as a requirement in a particular development, it is a sufficient condition to characterise the system as *inherently distributed*, in opposition to systems which are made distributed due to a decision adopting a distribution technology.

With the notions defined above, it is already possible to express many distributed system *conceptual models*, capturing distinct sets of choices concerning the modes of object creation, configuration, interaction and failure that regulate the structure and behaviour of a family of distributed systems. In the present paper, we recurrently rely on examples of dynamically configured asynchronous message passing and statically configured distributed shared memory systems.

We adopt the Actor Model [1] as a reference to message passing systems. Actors are independent units of interaction and computation. They interact solely via asynchronous point-to-point message passing. The delivery of messages is guaranteed and, as a result of consuming a message, an actor may change its local state, create finitely many objects and dispatch a finite number of messages to the actors which became known at creation time or through message consumption.

Concerning distributed shared memory systems, we adopt Unity [6] as a reference model. Objects in Unity are defined in terms of memories denoted by variables and also by multiple assignments. They interact among themselves and with their environment by reading and writing on shared memories, possibly synchronising on such occurrences. Unity adopts a weak fairness assumption requiring that continuously enabled assignments eventually occur. It is only due to such occurrences that variables defining object states can change.

Conceptual models such as Actors and Unity are formulated to enable the specification and verification of distributed systems. We use here the financial systems domain to contrast the respective modelling approaches. Two types of objects are postulated to exist: persons, active objects capable of receiving and disbursing amounts of money, and accounts, passive objects over which credit and debt operations can be performed. Typical relationships between these objects are those of deposit and withdrawal. We also allow persons and accounts to exchange money directly among themselves.

Persons and accounts are seen here as object types of the same kind. We consider that they are both money repositories and take advantage of this consideration to propose a unique specification for their kind. We present in Fig. 1 their specification according to the Actor Model and in Fig. 2 according to Unity. We should point out that the *get* and *put* actions in these specifications should be respectively read as a reception and a disbursement of money whenever repositories are seen as persons. The same rationale applies to accounts, credits and debts.

We use a a temporal logic language in these specifications [10], instead of the usual declarative ones [1, 6], to specify local object behaviours. The conditions in each specification are usually called *local constraints*, since they restrict the set of life cycles admissible for each object. For instance, (i) if we get some amount of money from a repository, this amount will be deduced from the respective balance in a moment strictly

Actor AMoneyRep
data types $\mathsf{Addr}, \mathsf{Float}$
attributes $bal : \mathsf{Float}$
messages $get?(\mathsf{Addr}, \mathsf{Float}), get!(\mathsf{Addr}, \mathsf{Float}), put?(\mathsf{Addr}, \mathsf{Float}), put!(\mathsf{Addr}, \mathsf{Float})$
axioms $m, n : \mathsf{Addr}; k, x, y : \mathsf{Float}$

$$(get?\,n\,x) \land bal = k \to (\neg((get?\,m\,y) \lor (put?\,m\,y)))\hat{\mathbf{U}}(\mathbf{send}\ n\ (get!\ \mathbf{self}\ x) \land bal = k - x) \quad (1.1)$$
$$(put?\,n\,x) \land bal = k \to (\neg((get?\,m\,y) \lor (put?\,m\,y)))\hat{\mathbf{U}}(\mathbf{send}\ n\ (put!\ \mathbf{self}\ x) \land bal = k + x) \quad (1.2)$$
$$get?\,n\,x \to bal \geq x \quad (1.3)$$

Fig. 1. Actor-based specification of money repositories.

Process UMoneyRep
data types Float
memories $bal : \mathsf{Float}$
actions $get(\mathsf{Float}), put(\mathsf{Float})$
axioms $k, x, y : \mathsf{Float}$

$$get(x) \land bal = k \to (\neg (get(y) \lor put(y)))\hat{\mathbf{U}}(bal = k - x) \quad (2.1)$$
$$put(x) \land bal = k \to (\neg (get(y) \lor put(y)))\hat{\mathbf{U}}(bal = k + x) \quad (2.2)$$
$$get(x) \to bal \geq x \quad (2.3)$$

Fig. 2. Unity-based specification of money repositories.

in the future[1] and no other event is allowed to happen until then (1.1 and 2.1); (ii) this is only allowed to happen if the current balance is greater or equal to the required amount (1.3 and 2.3).

The differences in modelling style are apparent in our specifications: whereas actors interact by receiving, consuming and sending back asynchronous messages from/to their clients, Unity processes are synchronous, in the sense that events and variable changes are perceived to happen simultaneously by the participant objects. We should stress that, even using a particular specification formalism, the respective objects comply with the semantics of the corresponding conceptual models. For instance, a withdrawal represented in the standard Unity notation by a guarded assingment of a value x to an account balance shared variable is represented in specification UMoneyRep by the action identifier $get(x)$ and two axioms are used to express its pre and post conditions: axiom (2.3) defines the assingment guard while axiom (2.1) defines its results.

In order to address the configuration and correlation of local object behaviours, coordination specifications are adopted. Instead of proposing textual specifications for the aforementioned types of distributed objects in particular operation contexts, we per-

[1] It is important to mention that the standard semantics of these models also prevent two local events of the same object to happen concurrently.

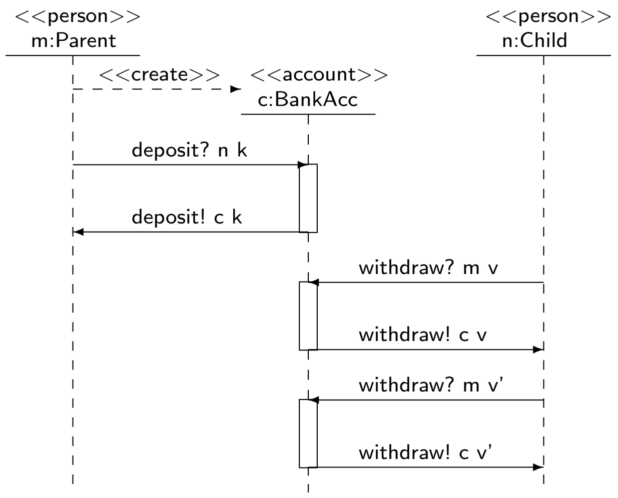

Fig. 3. Message-based scenario for a financial system behaviour.

form this task diagrammatically: Fig. 3 details a possible message-based scenario for the behaviour of a financial system, whereas Fig. 4 uses a shared-memory state-based notation. These specifications are formulated using UML [20].

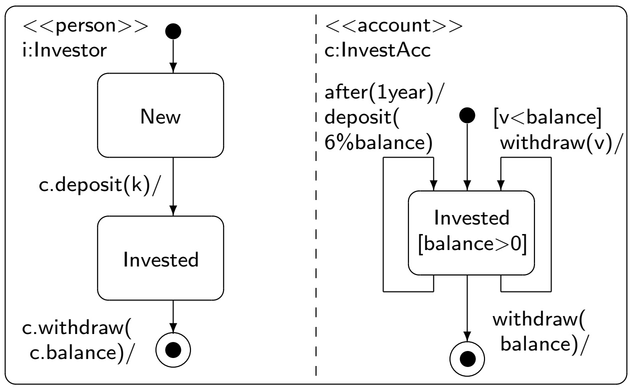

Fig. 4. Shared-memory state-based specification for financial system behaviours.

3 Algebra

3.1 Time Frames

A central issue in distributed systems formal modelling consists in the identification of the structure and role of time, since different families of distributed systems appear to require distinct time structures in order to express the relatively autonomous behaviour of their constituent objects. In view of this diversity, we define here time structures in a very general manner and study afterwards how the usually adopted time frames can be derived. The role of time in distributed systems modelling are discussed in Section 3.2.

Definition 1 (Time Frame). A *time frame* is a(n algebraic) structure $\langle\langle\mathbf{T},\leq\rangle,L\rangle$:

- $\mathbf{T}$ is a non-empty set defining a *time domain*;
- $\leq\ \subseteq\ \mathbf{T}\times\mathbf{T}$ is a(n order) relation, with $\langle\mathbf{T},\leq\rangle$ satisfying the following axioms:
  (R1) Reflexivity: For every $x\in\mathbf{T}$, $x\leq x$;
  (R2) Anty-symmetry: For every $\{x,y\}\subseteq\mathbf{T}$, $x\leq y\wedge y\leq x\rightarrow x=y$;
  (R3) Transitivity: For every $\{x,y,z\}\subseteq\mathbf{T}\ x\leq y\wedge y\leq z\rightarrow x\leq z$; (transitivity)
- L is a set of *linear time flows*: every $\lambda\in L$ is a function with $\mathsf{dom}\ \lambda\in\mathscr{P}_{+}(\mathbf{T})$ and $\mathsf{cod}\ \lambda\in\mathscr{I}_{+}(\mathbf{L})$[2], for some fixed $\langle\mathbf{L},\preceq\rangle$[3] satisfying (R1-3) and
  (LIN) Linearity: For every $\{x,y,z\}\subseteq\mathbf{L}$, $x\preceq y\rightarrow z\preceq x\vee y\preceq z$;
  and each λ satisfying the following axioms:
  (OO) Injectivity: For every $\{x,y\}\subseteq\mathsf{dom}\ \lambda\ \lambda(x)=\lambda(y)\rightarrow x=y$;
  (OT) Surjectivity: For every $y\in\mathsf{cod}\ \lambda$, $\exists x\cdot x\in\mathsf{dom}\ \lambda\wedge\lambda(x)=y$;
  (MO) Monotonicity: For every $\{x,y\}\subseteq\mathsf{dom}\ \lambda$, $x\leq y\rightarrow\lambda(x)\preceq\lambda(y)$.

The discrete time frames usually adopted in concurrent and distributed systems development can be classified according to their linear or branching nature. Branching time has been considered a convenient framework not only for the implementation of model checking tools but also for the specification of one kind of *enabledness property* which expresses what may be happening at the current time instant [10]. On the other hand, linear time is the mainstream assumption, due to its simplicity, but nevertheless allows one to express enabledness as the satisfaction of sufficient pre-conditions for a specific imminent occurrence [15].

Definition 1 easily captures linear or branching time frames. In particular, the assumption of linearity is captured whenever L is postulated to be a singleton. The fact that $\mathbf{L}$ and $\mathbf{T}$ are left unspecified in such kind of postulate makes it possible to express discrete or dense time extra assumptions. We do not, however, require that $\langle\mathbf{T},\leq\rangle$ obeys only such restrictions taking into account inherently distributed physical systems for which the idealised time frame is imaginary.

In fact, the adoption of linearisation functions in L can be viewed precisely as a mechanism to map higher-dimensional time structures into the forth dimension of space-time, which corresponds to our standard linear intuition concerning time flows. Incidentally, we capture the existence of *initial time instants* in all time flows by postulating that each $\lambda\in L$ is bounded below.

[2] We denote by $\mathscr{P}(X)$ the set of all subsets of X and by $\mathscr{I}(X)$ the set of all (open and closed) intervals defined in terms of X. Moreover, whenever we use the symbol $+$ as a subscript, the elements of these sets are non-empty.

[3] By abuse of notation, we put $\mathsf{dom}\ L\overset{\mathrm{def}}{=}\mathbf{T}$ and $\mathsf{cod}\ L\overset{\mathrm{def}}{=}\mathbf{L}$.

3.2 Distributed System Formal Models

The role of time in distributed systems development consists precisely in allowing the association of autonomous objects to corresponding behaviours and placements. The following definition captures this intuition:

Definition 2 (Object Structure). An *object structure* is a 5-tuple $\langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P\rangle$ defined in terms of the disjoint non-empty sets **T**, **Ev** and **Loc**, where:

– $\mathscr{T} = \langle\langle \mathbf{T}, \leq\rangle, L\rangle$ is a time frame;
– $\mathscr{V}$ is an *event structure*, defined in terms of an *event domain* **Ev**;
– $\mathscr{S}$ is a *location structure*, defined in terms of a *location domain* **Loc**;
– $B : \mathbf{Ev} \to \mathscr{P}\left(\mathscr{I}_{+}(\mathsf{cod}\, L)\right)$ defines a *behaviour*[4];
– $P : \mathbf{Loc} \to \mathscr{P}\left(\mathscr{I}_{+}(\mathsf{cod}\, L)\right)$ defines a *placement*;

provided that the following axiom is satisfied:

(MAX) Maximality: For every $e \in \mathbf{Ev}$ and $s \in \mathbf{Loc}$, $((\cap B(e)) = \{\}) \wedge ((\cap P(s)) = \{\})$.

Structures as defined above are said to be partial order based in [22], since they rely on the definition of underlying partial-order relations as their time frames, $\mathscr{T}$. The time flows derived from these frames, L, are used in the assignment of a time dependent interpretation to events and locations, through B and P respectively. We leave the event and location structures $\mathscr{V}$ and $\mathscr{S}$ partially unspecified, since they are quite dependent upon the application domain at hand: events may carry values and locations may have hierarchical structure, for instance to represent realistic patterns of interaction and nested geographic regions respectively.

Concerning our financial information system specifications, the events that populate **Ev** are represented by *get* and *put* (along with their variations formed by the use of the suffixes *?* and *!*). Therein, we could have postulated that objects are explicitly located. In that case, the values populating **Loc** would define a domain on their own, specified according to an enumeration such as the following one:

$$\mathsf{Loc} \overset{\text{def}}{=} \text{HOME} \mid \text{ATM} \mid \text{AGENCY}$$

In this context, it makes sense to talk about a new requirement concerning transaction costs depending on the client location. For instance, transactions performed at an agency would be ten times more expensive than the others. In order to treat this requirement in our specifications, not only would we have to define Loc as above, but we would also have to add a new parameter to each event and modify all the axioms to deduce the cost from the account balance whenever performing any transaction. Note that systems with location dependent requirements are regarded as inherently distributed.

Our example can be further extended so that we can illustrate the definition above. Suppose now that the aforementioned client is in fact a bank employer with a daily regular behaviour pattern: every day, he (i) leaves home early in the morning and comes back at night; (ii) stays in his agency during the whole day, and (iii) withdraws some

[4] Note that, since $\mathsf{cod}\, L$ is totally ordered, an interval S of $\mathsf{cod}\, L$ is a subset of $\mathsf{cod}\, L$ such that, whenever $\{x, y\} \subseteq S$, $\forall z \in \mathsf{cod}\, L \cdot x \leq z \leq y \to z \in S$.

money before working and deposits the remaining amount after work, both using an ATM. Considering the previous definitions, the following could be a possible object structure for this behaviour:

$$\begin{array}{ll} P(\mathsf{HOME}) = \{[0',8'],[20',24']\} & \\ P(\mathsf{AGENCY}) = \{[9',19']\} & B(\mathit{withdraw}) = \{[8'30"]\} \\ P(\mathsf{ATM}) = \{[8'30"],[19'30"]\} & B(\mathit{deposit}) = \{[19'30"]\} \end{array}$$

This example shows that, even if we forget locations, our previous definition is far more general than those found in the literature. For example, events may be durative. This is accomplished by requiring that each event be assigned by B to the set of time intervals during which the event occurs in a time flow. The usually adopted instantaneous events [10] can be obtained by postulating that each occurrence is represented by a singleton, something captured by an axiom such as (INS) below. The local structures proposed in [22] are obtained by additionally requiring each object structure to obey sequentiality (SEQ) in a linear time structure. If we further require discreteness, as in [7], this can be captured by the assumption of yet another axiom, (DISC). Concerning localities, (XOR) may be postulated to ensure that two of them are never occupied at the same instant by a single object:

(INS) Instantaneity: For every $e \in \mathbf{Ev}$ and $x \in \mathscr{P}\left(\mathscr{I}_{+}(\mathsf{cod}\, L)\right)$,
$x \in B(e) \to (\forall y, z \cdot y \in x \wedge z \in x \to y = z)$;

(SEQ) Sequentiality: For every distinct $\{e_1, e_2\} \subseteq \mathbf{Ev}$,
$(\cup B(e_1)) \bigcap (\cup B(e_2)) = \{\}$;

(DISC) Discreteness: For every $\{x, y\} \subseteq \mathbf{T}$,
$x \leq y \to \exists z \cdot (x \leq z \wedge \nexists u \cdot x \leq u \wedge u \leq z) \wedge \exists w \cdot (w \leq y \wedge \nexists u \cdot u \leq y \wedge w \leq u)$;

(XOR) Orthogonality: For every distinct $\{s_1, s_2\} \subseteq \mathbf{Loc}$,
$(\cup P(s_1)) \bigcap (\cup P(s_2)) = \{\}$;

Given two object structures, we could have naively proceeded to attempt to define diverse modes of interaction. For instance, we could have stated that two objects synchronise at an event occurrence if the event belongs to their structures and its interpretations according to B in each of these structures have a time instant in common. However, this definition would not be sufficiently general to encompass the postulation of durative events. Worse, the involved structures could have time frames with distinct order-theoretic characters. In fact, such diversity reflects the nonexistence of universally valid time frames, a consequence of the acceptance of the theory of relativity governing inherently distributed systems.

Considering this rationale, the definition of distributed system (formal) models must be relational, in view of the necessity to establish correspondences between the time frames of each pair of objects, as well as the boundary of each (sub-)system in relation to its environment:

Definition 3 (Distributed System Model). A *distributed system model* is a 4-tuple $\langle \mathscr{O}, E, I, C \rangle$, where:

- $\mathscr{O}$ is an *object universe*, a countable set of object structures;
- $E : \mathscr{O} \to \mathbf{Bool}$ is an *environment* identification function;

- $I : \mathscr{O} \to \mathbf{Bool}$ is an *internal object* identification function;
- $C : \mathscr{O} \to \mathscr{O}$ is a partial map which induces a family of *time correlation* functions: for each $o = \langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P \rangle$, $o \in \mathsf{dom}\, C$, such that, if there is $o' = \langle \mathscr{T}', \mathscr{V}', \mathscr{S}', B', P' \rangle$ with $C(o) = o'$, then there are also $\kappa_o : \mathbf{T} \to \mathbf{T}'$ and $\kappa_{o'} : \mathbf{T}' \to \mathbf{T}$ both obeying (MO).

such that the following axiom is satisfied:

(SEP) Separation: For every $o \in \mathscr{O}$, $E(o) \to \neg I(o)$.

The structural interface of a distributed system model – $\mathscr{O}$, E and I – is rather conventional and attempts to generalise the structures proposed in [2, 25]: $\mathscr{O}$ captures an universe of distributed objects, E identifies the objects which belong to the *environment* (those *external* to the modelled system(s)), whereas I spots the *internals*, objects which cannot directly interact with the environment. Such interfaces will play a central role in the definition of model composition below.

It is important to point out that, since we do not make any assumption concerning (the existence of) real physical locations of distributed objects (if we had, we could also have proposed a metric establishing the distance between distributed objects), we cannot presume a fixed relationship between their respective time frames. In particular, since objects may have a strictly logical character, they are not obliged to follow relativistic correlations between time and space. Of course, in most cases object locations are given in terms of spatial coordinates and, in these cases, the respective structures will have to respect further spatio-temporal constraints.

The relational character of C is a consequence of the assumptions above. This map is important, however, not only to correlate object structure time frames but also to avoid violations in temporal causality chains (time travel), something prevented by the order preserving requisite (monotonicity) posed on this relationship.

3.3 Interaction and Composition

We are now able to formalise the distinction between synchronous and asynchronous modes of interaction, in the sense of [13]. Provided an object structure $\mathscr{O}$, we denote by $\mathbf{Ev}_{\mathscr{O}}$ the set of events of the underlying event structure $\mathscr{V}_{\mathscr{O}}$. Given $E \subseteq \mathbf{Ev}_{\mathscr{O}}$, we say that an *occurrence* D of the events in E is *totally synchronous* if D belongs to the interpretation of each event in E, that is $\forall e \cdot e \in E \to D \in B(e)$. This mode of interaction is unrealistic and used only in idealised models of distributed systems. Therefore, *partial synchrony* is usually adopted instead, which is defined by the existence of a common time point t in the given occurrences $D \in B(e)$ of each event e, that is $\exists t \cdot \forall e \cdot e \in E \wedge D \in B(e) \to t \in D$. Clearly, these definitions concern only to intra-object interaction, but they can be extended to inter-object interaction by considering that the involved objects all belong to the same distributed system model and are time correlated. Consequently, we say that objects interact in *(partially) synchronous* mode if (partial) event synchronisation is adopted as the sole mode of interaction.

The definition of synchrony above provides a sufficient characterisation of this notion but not a necessary one. A sufficient and necessary characterisation is obtained by asserting that objects rely on an upper bound on interaction delays, clock drifts or relative object speeds [13]. The usual way of guaranteeing this interaction pattern is to require that objects communicate using a blocking scheme. Objects rely on this kind of synchronisation information by referring to facilities provided by their (operational) environment, whose existence is presumed or defined in linguistic terms during the system development. In Section 3.4, we address the specification of such facilities.

The messages exchanged between persons and accounts illustrate point-to-point asynchronous interaction. In particular, the semantics of these events can be defined in terms of time intervals that capture the periods of time in which messages remain in transit. On the other hand, *deposit* and *withdraw* actions performed simultaneously by investors and their accounts are instances of synchronous interaction.

The semantics of interaction outlined above captures the view that only time correlated objects of the same model are able to interact. Now we wish to generalise this view to take into account a design method in which distinct distributed systems are modelled by isolated structures and are composed afterwards. The composition of distributed system models is defined below:

Definition 4 (Composability). Two distributed system models $\mathscr{M}_i = \langle \mathscr{O}_i, E_i, I_i, C_i \rangle$, $i \in \{0,1\}$, are *composable* if and only if, given that $\mathscr{O} = \mathscr{O}_0 \cup \mathscr{O}_1$, the following axiom is satisfied:

(DISJ) Disjointedness: For every $o \in \mathscr{O}$, $\neg (I_0 \uparrow_{\mathscr{O}}(o) \wedge I_1 \uparrow_{\mathscr{O}}(o))$ [5];

Definition 5 (Model Composition). A *distributed system model* $\mathscr{M} = \langle \mathscr{O}, E, I, C \rangle$ is the *composition* of composable distributed system models $\mathscr{M}_i = \langle \mathscr{O}_i, E_i, I_i, C_i \rangle$, $i \in \{0,1\}$, if the following axioms are satisfied:

(GEN) Generalisation: $\mathscr{O} = \mathscr{O}_0 \cup \mathscr{O}_1$;
(EXT) Scoping: For every $o \in \mathscr{O}$, $E(o) \to E_0 \uparrow_{\mathscr{O}}(o) \vee E_1 \uparrow_{\mathscr{O}}(o)$;
(IN) Internalisation: For every $o \in \mathscr{O}$, $I_0 \uparrow_{\mathscr{O}}(o) \vee I_1 \uparrow_{\mathscr{O}}(o) \to I(o)$;
(COR) Correlation: For every $i \in \{0,1\}$ and $o \in \mathscr{O}$, $C_i(o) = o' \to C(o) = o'$.

Much in the way that identifying an object in different models is an important part of the composition activity, particularly in order to ensure the satisfiability of the axioms above, the formulation of time correlation functions is also part of this activity. Note, however, that no new such function is required to exist in a composed model, since its components are not obliged to maintain between themselves time correlated objects.

A model composition example is obtained by considering as sub-systems a parent with an external bank account object, a child with its external account and the account itself, a sub-system composed by this single object. By composing these sub-systems pairwise, presuming that all their objects are time correlated and that the account becomes an internal object in the composition, we obtain a configuration having as a possible behaviour that described by Fig. 3. Note that the composition of these sub-systems is not unique, since we can also define a composed system with all the objects as internals, a so-called *closed system* [1].

[5] $f \uparrow_G / f \downarrow_H$ denote the generalisation / particularisation of the boolean valued function f to the set G / H, with $\mathsf{dom}\, f \subseteq G$, $H \subseteq \mathsf{dom}\, f$ and $f \uparrow_G(x) \overset{\text{def}}{=} \begin{cases} f(x), & \text{for } x \in \mathsf{dom}\, f \\ \text{FALSE}, & \text{otherwise.} \end{cases}$

3.4 Distributed System Representations

We study in this section distributed system representations, such as the specifications presented in Section 2. We are not, however, concerned here with the syntax of the adopted formal languages, since there are many available alternatives. A possibility is the use of the languages proposed in [22], which are layered: there is a local temporal language for defining objects and another language for expressing global system properties. Another possibility is the adoption of a single language with different usage contexts [9]: the local view of each object is captured by object specifications, whereas system configurations and global properties are represented in the context of coordination specifications.

The semantics of such representations can be given in terms of object structures and (composed) distributed system models. We have already sketched how message and action symbols can be interpreted using such structures. The semantics of states and memory values can be given in the same predicative way.

Some semantically rich symbols are often found in such representations:

self: The immutable unique identification of an object;
loc: The current location (set) of an object;
acq: The current set of *acquaintances* of an object – the (identities of the) objects that became known at creation time or through object interaction;

Distributed system classifications are formulated in terms of the potential knowledge of these notions by the respective objects. If **self** is available, the system is said to be *identified*, or else it is said to be *anonymous* [4]. Whenever **loc** is available, the system is said to be *location aware*. If the interpretation of this symbol is not constant, the system objects are said to be *mobile* [8]. Moreover, if **T** is accessible by some objects, then we are dealing with *temporised* systems. By referring directly or indirectly to **T**, object computation and communication can be defined to happen more or less synchronously. We postpone the presentation of the classification corresponding to **acq** to Section 4.2.

It is important to distinguish if the above notions belong to the underlying models or are definable in terms of other notions. For this purpose, it becomes necessary to introduce the semantic notions of interpretation, satisfaction, truth and consequence [12]. Given a collection of specification symbols Δ (a signature) whose generated language is denoted by $lang(\Delta)$, we postulate the existence of an interpretation of Δ, a map $[\cdot]$ that, provided an object structure o, assigns each symbol in Δ to an event or location of o. Note that Δ may be partitioned in action/message and state symbols and that $lang(\Delta)$ may have other logical symbols such as the above, which are not part of signatures. The satisfaction $o \models^t_\lambda p$ of a sentence $p \in lang(\Delta)$ in a moment $t \in \mathsf{dom}\ \lambda$ of a time flow $\lambda \in L$, a component of $\mathscr{T}$ of a structure $o = \langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P \rangle$, is defined by recursion on the structure of $lang(\Delta)$. The base cases of this definition in terms of Δ are:

– $o \models^t_\lambda s$ iff $\exists d \cdot d \in B([s]^o) \wedge t \in d$, if $[s]^o \in \mathbf{Ev}_o$;
– $o \models^t_\lambda s$ iff $\exists d \cdot d \in P([s]^o) \wedge t \in d$, if $[s]^o \in \mathbf{Loc}_o$;

A sentence $p \in lang(\Delta)$ is locally true in o and λ ($o \models_\lambda p$) whenever $o \models^t_\lambda p$ for every $t \in \mathsf{dom}\ \lambda$. Whenever this is the case for every λ, p is simply true in o ($o \models p$). A sentence is valid if it is true in any such structure for Δ ($\models p$). A consequence relation

$\Psi \models p$ between a set of sentences Ψ and a sentence p can be defined by stating that $o \models p$ whenever $o \models q$, for every $q \in \Psi$ and any admissible object structure o for Δ.

4 Topology

Since this section relies on topology, we present below the relevant definitions.

The aggregate of elements of a family of subsets F_X of a given set X is a topological space T_X whenever it satisfies the following axioms:

(T1) $\{\} \in F_X$ and $X \in F_X$;
(T2) $X_1 \cap \ldots \cap X_n \in F_X$ for any $n \in \mathbb{N}$ and every $X_i \in F_X$, $1 \le i \le n$;
(T3) $\bigcup_{X_i \in P} X_i \in F_X$ for every $P \subseteq F_X$;

Elements of X are called points, elements of F_X are entitled opens and F_X is named a topology on X.

Given a topological space $T_X = (X, F_X)$, the complement of A under X is denoted by A' (that is, $A' \overset{\text{def}}{=} X - A$). Closed sets are complements of opens in F_X.

4.1 Behavioural Properties

Here we recast the topological characterisation of safety and liveness properties of concurrent and distributed systems of [3] in terms of our own formal framework.

We first provide necessary and sufficient logical characterisations for safety and liveness. Given a signature Δ and an object structure $o = \langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P \rangle$ for Δ, $p \in lang(\Delta)$ is said to be a safety (liveness) property if and only if:

(SAFE) Safety: For every $\lambda \in L_{\mathscr{T}}$,
$(\neg(o \models_\lambda p) \Rightarrow (\exists t \in \mathsf{dom}\ \lambda \cdot \forall \lambda' \in L_{\mathscr{T}} \cdot \lambda \sqsubseteq^o_t \lambda' \to \neg(o \models_{\lambda'} p)))$ [6]
(LIVE) Liveness: For every $\lambda \in L_{\mathscr{T}}$, $(\exists \lambda' \in L_{\mathscr{T}}; t \in \mathsf{dom}\ \lambda \cdot \lambda \sqsubseteq^o_t \lambda' \wedge o \models^t_{\lambda'} p)$;

The first axiom states that, if we are not dealing with a safety property, it is possible to identify an instant in which a "bad thing" falsifies the property. On the other hand, the second axiom states that a liveness property guarantees the occurrence of a "good thing" at some instant. Examples of these properties are respectively that a withdrawal cannot happen if the account does not hold the required funds and that investors eventually demand repayment of the invested amounts.

Taking advantage of the necessary and sufficient characters of the above definitions and of the fact that object behaviours and placements are completely determined by their underlying time flows when related by $\sqsubseteq$, from now on we deal with distributed system properties by relying on the corresponding sets of time flows. That is, we use $P \overset{\text{def}}{=} \{\lambda \in L_{\mathscr{T}} | o \models_\lambda p\}$ instead of p.

[6] We denote by $\lambda \sqsubseteq^o_t \lambda'$ the dominance of a function λ by another one λ' of the same type in an object structure o up to t. It is defined by:

$$\lambda \sqsubseteq^o_x \lambda' \overset{\text{def}}{=} \forall y \cdot y \le x \to \begin{pmatrix} \forall e \cdot \lambda^{-1}(y) \in B(e) \to \lambda'^{-1}(y) \in B(e) \wedge \\ \forall s \cdot \lambda^{-1}(y) \in P(s) \to \lambda'^{-1}(y) \in P(s) \end{pmatrix}$$

Now we adopt these sets to provide a topological characterisation for safety and liveness properties. First note that $\sqsubseteq_x^o$ obeys (R1) and (R3) for each $x \in \mathbf{T}$, that is, $\sqsubseteq_x^o$ is a pre-order. Given $X \subseteq L_{\mathscr{T}}$, we define the following operators:

- $\mathit{Int}(X) \overset{\text{def}}{=} \{\lambda' \in L_{\mathscr{T}} | \exists \lambda \in X; x \in \mathsf{dom}\ \lambda \cdot \lambda \sqsubseteq_x^o \lambda'\}$;
- $\mathit{Cl}(X) \overset{\text{def}}{=} \{\lambda \in L_{\mathscr{T}} | \exists \lambda' \in X; x \in \mathsf{dom}\ \lambda' \cdot \lambda \sqsubseteq_x^o \lambda'\}$;

$\mathit{Int}(X)$ is a set of time flows with common prefixes in X. As in [3], these sets are considered to be opens. They correspond exactly to liveness properties here. $\mathit{Cl}(X)$ is a closed set and corresponds to a safety property. It is not difficult to see that opens obey (T1), (T2) and (T3). Therefore, their family $\mathit{Int}(X)$, $X \subseteq L_{\mathscr{T}}$, defines a topology.

Notice that the indexed sequences of program states used in [3], determined by our time flows λ and λ' in (LIVE), are respectivelly required to be finite and infinite. Consequently, liveness properties are characterised as dense sets therein. We do not consider this to be a reasonable requirement in a general temporal setting (so long as we maintain that "something good" eventually happens) and, as a result of this abstraction, obtain a characterisation in which the following holds:

Lemma 1. Liveness properties are closed under arbitrary intersections.

This is a direct consequence of the set-theoretic definition of *Int*. Consequently, by duality, safety properties are closed under arbitrary unions.

The main result concerning safety and liveness can be formulated as follows:

Theorem 1 (Behavioural Characterisation). Every property is an intersection of a safety and a liveness property.

Proof: Given a signature Δ and an object structure $o = \langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P \rangle$ for Δ, it suffices to show, for $P \subseteq L_{\mathscr{T}}$ representing $p \in \mathit{lang}(\Delta)$, that $P \subseteq \mathit{Int}(X) \cap \mathit{Cl}(Y)$ for some $X \cup Y \subseteq L_{\mathscr{T}}$. But, for each $\beta \in P$, $\beta \in \mathit{Int}(\{\beta\}) \cap \mathit{Cl}(\{\beta\})$. Therefore:

$$\begin{aligned} P &\subseteq \textstyle\bigcup_{\beta \in P} (\mathit{Int}(\{\beta\}) \cap \mathit{Cl}(\{\beta\})) \quad \subseteq (\bigcup_{\beta \in P} \mathit{Int}(\{\beta\})) \cap (\bigcup_{\beta \in P} \mathit{Cl}(\{\beta\})) \\ &\subseteq \mathit{Int}(\textstyle\bigcup_{\beta \in P} \{\beta\}) \cap \mathit{Cl}(\bigcup_{\beta \in P} \{\beta\}) \subseteq \mathit{Int}(P) \cap \mathit{Cl}(P) \ \blacksquare \end{aligned}$$

4.2 Structural Properties

In this section, we show that structural properties of distributed systems can also be characterised in topological terms.

Given an object structure $o = \langle \mathscr{T}, \mathscr{V}, \mathscr{S}, B, P \rangle$ and a countable value domain Addr, we postulate the existence of $\{\mathit{self}(\mathsf{Addr}), \mathit{acq}(\mathsf{Addr})\} \subseteq \mathbf{Ev}_o$ and that these events respectively capture the semantics of **self** and **acq** if available in the adopted representation language[7]. The *configuration* of the respective object for $\lambda \in L_{\mathscr{T}}$ and $t \in \mathsf{dom}\ \lambda$ is defined by the following function:

$$\mathit{Conf}_\lambda(t) \overset{\text{def}}{=} \{i : \mathsf{Addr} | t \in \cup B(\mathit{acq}(i))\}$$

[7] Moreover, that $\forall x : \mathsf{Addr} \cdot \mathit{self}(x) \rightarrow \mathit{acq}(x)$.

This notion can be generalised to each object $o \in \mathscr{O}$ of a distributed system model $\mathscr{M} = \langle \mathscr{O}, E, I, C \rangle$ as follows:

$$Conf^{o}_{\lambda}(t) \stackrel{\text{def}}{=} \{i : \mathsf{Addr} | t \in \cup B_o(acq(i)) \wedge C(o) = o' \wedge \kappa_o(t) \in \cup B_{o'}(self(i))\}$$

Without loss of generality, we deal with objects themselves in place of their identifications, since these are unique and immutable. Concerning each distributed system $X \subseteq \mathscr{O}$ that $\mathscr{M}$ represents, we say that it has a *static configuration* if and only if, for each $o \in X$, $Conf^{o}_{\lambda}(t)$ is constant over $t \in \mathsf{dom}\ \lambda$, for every $\lambda \in L_{\mathscr{T}}$. Otherwise, it is said to have a *dynamic configuration*.

Going back to Section 3.3, it is easy to see that the example system composed by three objects has a static configuration whenever the unique allowed interaction pattern is that described by Fig. 3. On the other hand, if child or parent objects are allowed to interact with the external environment, the system will have a dinamic configuration.

It is the open and closed character of some distributed system configurations that is subject to a topological definition. Towards this, we need two further generalisations of distributed system configuration notions, respectively without any reference to the passage of time and covering sets of objects instead of single objects. The corresponding definitions are presented below:

$$Conf(o) \stackrel{\text{def}}{=} \bigcup_{\substack{t \,\in\, \mathsf{dom}\,\lambda \\ \lambda \,\in\, L_{\mathscr{T}}}} Conf^{o}_{\lambda}(t); \qquad Conf(X) \stackrel{\text{def}}{=} \bigcup_{o \in X} Conf(o)$$

Given $X \subseteq \mathscr{O}$, $Conf(X)$ is considered to be an open. Again, opens obey (T1), (T2) and (T3) and the family $Conf(X)$, $X \subseteq \mathscr{O}$, defines a topology. Moreover,

Lemma 2. Configurations are closed under arbitrary intersections.

The following is a structural counterpart to our behavioural characterisation:

Theorem 2 (Structural Characterisation). Every object universe is an union of open and closed disjoint sets of objects.
Proof: Given a model $\mathscr{M} = \langle \mathscr{O}, E, I, C \rangle$, let the set of internals be $J \stackrel{\text{def}}{=} \{o \in \mathscr{O} | I(o)\}$, the externals set be $F \stackrel{\text{def}}{=} \{o \in \mathscr{O} | E(o)\}$ and $R \stackrel{\text{def}}{=} \mathscr{O} - J - F$. Hence:

1. The open set containing all the objects directly reachable from the environment is $Conf(F)$. Therefore, $Conf(F)'$ is the closed set corresponding to J;
2. The open set of objects corresponding to R is $Conf(J) \cap Conf(F)$;
3. The closed set corresponding to F is $(Conf(J) \cup (Conf(J) \cap Conf(F)))'$.

These three sets are disjoint and their union corresponds to $\mathscr{O}$. ■

It is interesting to mention that the set R above corresponds precisely to the *receptionist* objects of [1] and, more generally, to the objects reachable from the environment. These are the central objects in each distributed system model.

It is not surprising to reach the conclusion above, since it captures the standard intuition in distributed systems development that internals are substitutable and the environment uncertain. The theorem shows in particular that, if we ignore the empty sets that determine trivial distributed system models and adopt the terminology mentioned in Section 3.3, each single closed system corresponds precisely to a closed object set in our topology: it determines the set of internals of the model, whereas the sets of externals and receptionists are empty in this case.

4.3 Transference of Results

Now we show how results on topological spaces can be transfered directly to distributed system theory. We focus our attention in the problem of determining the class of distributed system models in which external objects not able to reach the modelled systems have been eliminated therein.

Due to the nature of this problem, we are obliged to introduce other topological notions. We say that $f : A \to B$ is continuous if the inverse images of closed sets under f are closed. In a topological space $T_X = (X, F_X)$, a sub-set $Y \subseteq X$ is said to be connected if there is no way to define Y as a union of two disjoint nonempty open sets. The connected component of $p \in X$ is a sub-set $C_p \subseteq X$ such that $p \in C_p$, C_p is connected and, if $p \in C$ for some connected $C \subseteq X$, $C \subseteq C_p$. The connected components of T_X are the respective sets that partition X. The following result is used in the sequel:

(CC) The connected components of a topological space are closed sets;

Let us spell out what we mean by an operation of restriction (modulo equivalence). This and other operations on models can be represented by means of injective set inclusion functions on objects whose inverse images map safety properties and internals into similar entities and also preserve separation and correlation. Note that the inverse image of these functions map closed sets (safety properties and internal objects) into closed sets, characterising them to be continuous.

Theorem 3 (Model Restriction). Each distributed system model can be restricted to an equivalent model without disconnected externals.
Proof: Given $\mathscr{M} = \langle \mathscr{O}, I, E, C \rangle$, let $A = \{A_i | A_i \subseteq \mathscr{O}\}$ be the family of connected components of the topological space $(\mathscr{O}, \{Conf(X) | X \subseteq \mathscr{O}\})$. Due to Theorem 2, we know that $\mathscr{O} = J \cup R \cup F$ for some closed J and F and some open R sets as defined therein. By (CC), we know that each A_i is a closed set. If $|A| = 0$, $\mathscr{M}$ is the trivial model which is clearly equivalent to itself. Alternatively, if $|A| \geq 1$, due to the dual of Lemma 2, $D = \bigcup_{A_i \subseteq F} A_i$ is the closed set of disconnected externals.

Put $\mathscr{O}' \overset{\text{def}}{=} \mathscr{O} - D$ and $\mathscr{M}' \overset{\text{def}}{=} \langle \mathscr{O}', I \downarrow_{\mathscr{O}'}, E \downarrow_{\mathscr{O}'}, C \downarrow_{\mathscr{O}'} \rangle$. Clearly, the function $f : \mathscr{O}' \to \mathscr{O}$ defined by the identity on $\mathscr{O}'$ satisfies all the requirements to be considered a restriction. Consequently, $\mathscr{M}'$ is the representative of the class of models equivalent to $\mathscr{M}$. ■

The result above illustrates how to take advantage of topology results, namelly (CC), to develop distributed systems theorems. If it is proven to be decidable for some class of models, it can be used to develop static analyses of distributed system representations that may suggest simplifications to software engineers. With a bit of ingenuity, it can also be extended to formalise distributed garbage collection.

5 Concluding Remarks

In the present paper, we proposed a novel characterisation of object-based distributed systems in terms of algebraic structures and topological spaces. Although there seems to exist a growing consensus concerning the importance of topological methods in distributed computing, we are not aware of other research efforts that address their whole development process in this way.

The proposed algebraic structures are sufficiently general to express most distributed system notions, such as diverse modes of object creation, configuration, interaction and timing usually found in the literature. In particular, they are a generalisation of the algebraic notions first developed in [2, 25]. The topological analysis of these structures allowed us to recast here behavioural results of [3], to develop similar results concerning distributed system configurations and also to exemplify how topological results can be transfered to distributed systems theory. We consider these to be the main original contribution of our research.

The reported research yields a general foundation for the definition of logical systems devoted to the compositional development of object-based distributed systems. It is also important towards establishing a semantically rich framework for the analysis and simulation of distributed system behaviours, facilitating their comprehension.

We expect to refine in the future the connections of the reported research with our previous work on object-based mobility [8], specification [9] and implementation [11]. It is in perspective an extension of this work towards applying the theory of dynamical systems to distributed systems development. Another interesting direction for future work is to formalise the method of transference of results from the aforementioned mathematical theories to distributed systems theory using Institutions [12].

Acknowledgement: The author gladly acknowledges that the remaks from an anonymous referee contributed to improve the readability of this paper.

References

1. G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems*. MIT Press, 1986.
2. G. Agha, I. Mason, S. Smith, and C. Talcott. A foundation for actor computation. *Journal of Functional Programming*, 7(1):1–72, 1997.
3. B. Alpern and F. B. Schneider. Defining liveness. *Information Processing Letters*, 21(4):181–185, October 1985.
4. H. Attiya, M. Snir, and M. Warmuth. Computing on an anonymous ring. *Journal of the ACM*, 35(4):845–876, Oct. 1988.
5. V. Barbosa and E. Gafni. Concurrency in heavily loaded neighborhood-constrained systems. *ACM Transactions on Programming Languages and Systems*, 11:592–584, 1989.
6. K. M. Chandy and J. Misra. *Parallel Program Design, A Foundation*. Addison-Wesley, 1988.
7. G. Denker and H. D. Ehrich. Specifying distributed information systems: Fundamentals of an object-oriented approach using distributed temporal logic. In H. Bowman and J. Derrick, editors, *Prof. 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems Conference (FMOODS'97)*, volume 2, pages 89–104. Chapman and Hall, 1997.
8. C. H. C. Duarte. A proof-theoretic approach to the design of object-based mobility. In H. Bowman and J. Derrick, editors, *Proc. 2nd IFIP Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'97)*, pages 37–53. Chapman and Hall, July 1997.
9. C. H. C. Duarte and T. Maibaum. A rely-guarantee discipline for open distributed systems design. *Information Processing Letters*, 74(1–2):55–63, April 2000.
10. C. H. C. Duarte and T. Maibaum. A branching-time logical system for open distributed systems development. *Electronic Notes on Theoretical Computer Science*, 67, 2002.

11. C. H. C. Duarte and C. Talcott. Clara: An actor language for high performance distributed computing. In *Proc. 12th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'2000)*, pages 20–37, October 2000.
12. J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39(1):95–146, January 1992.
13. V. Hadzilacos and S. Toueg. Fault-tolerant broadcasts and related problems. In *Distributed Systems*, pages 97–145. Addison-Wesley, 1993. Chapter 5 of [18].
14. M. P. Herlihy and N. Shavit. The topological structure of asynchronous computation. *Journal of the ACM*, 46:856–923, 1999.
15. L. Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994.
16. S. Lefschetz. *Algebraic Topology*. American Mathematics Society, 1942.
17. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
18. S. Mullender, editor. *Distributed Systems*. Addison-Wesley, 2nd edition, 1993.
19. M. Nielsen, G. Plotkin, and G. Winskel. Petri-nets, event-structures and domains - part i. *Theoretical Computer Science*, 13:85–108, 1981.
20. O. M. G. OMG. *Unified Modelling Language Specification*. Object Management Group — OMG, June 1999. Version 1.3.
21. G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Computer Science Department, University of Aahus, 1981.
22. R. Ramanujam. Locally linear time temporal logic. In *Proc. 11th IEEE Symposium on Logic in Computer Science*, pages 118–127. IEEE Computer Society Press, 1996.
23. M. Saks and F. Zaharoglou. Wait-free k-set agreement is impossible: The topology of public knowledge. *Siam Journal on Computing*, 29:1449–1483, 2000.
24. F. B. Schneider. What good are models and what models are good. In *Distributed Systems*, pages 17–26. Addison-Wesley, 1993. Chapter 12 of [18].
25. C. Talcott. Composable semantic models for actor theories. *Higher-Order and Symbolic Computation*, 11(3):281–343, 1998.
26. A. Tarski. *Logics, Semantics and Metamathematics*. Oxford Publishing Company, 1956.