Multiple inheritance in the form of reduction

Ferhat Khendek, Gregor. v. Bochmann and Reinhard Gotzhein
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, Canada

Abstract
The use of multiple inheritance for deriving a new class from existing ones is a well established technique since the introduction of the object oriented languages. In this paper, we introduce multiple inheritance construction for processes given in the form of acceptance trees. The conformance relationship between the newly derived process and its parents is clearly established. Moreover, we discuss the applications of this construction.

I. Introduction

Interest in the object orientation concepts has grown considerably over the last few years. Among these concepts, inheritance is an important one. It has different meanings [Amer 87, Wegn 88]. It is seen as an incremental technique for classes construction and modification. By inheritance, we can derive a new object class from existing ones. It is seen as a syntactical construction to reuse code for implementation efficiency. The second meaning is that inheritance is viewed as a conformance relation between classes discovered afterward by analysis methods [Amer 89, Cusa 89a, Cusa 89b]. Conformance of a class C to a class C' means that an object of class C' can be replaced by an object of a class C in a any system without invalidating essential system properties.

In this paper, we introduce multiple inheritance for processes modelled by Acceptance Trees (AT's) [Henn 85]. The conformance relationship between the newly derived process and its parents is clearly established. In other words, we define a multiple inheritance operator that will allow us to derive from a set of processes P_1, P_2, ...P_n, a process P with the constraint that P conforms to P_1, P_2, ...and P_n.

For process theory, different kinds of equivalences are defined in the literature [DeNi 87]. These semantics depend on the model and the properties supposed important to consider. They range from the coarsest one, trace semantics [Hoar 78], to the finest one, bisimulation semantics [Miln 80]. Nevertheless, equivalence relations are too strong as conformance relations, since they are symmetrical. The ordering relations are more suitable for that purpose. In this paper we are interested in the reduction relation [Brin 86]. This relation includes two aspects, the trace and deadlock properties. Informally, process P_1 reduces process P_2, if and only if the traces of P_1 are included in the traces of P_2, and P_1 deadlocks less often than P_2.

The reduction relation will play the role of conformance relation. As mentioned above, the multiple inheritance operator will allow us to derive from a set of processes P_1, P_2, ...P_n, a process P with the constraint that P reduces P_1, P_2, ...and P_n. Therefore, process P may be substituted for P_1, P_2, ... or P_n with a confidence that (trace and deadlock) properties of the overall system are not altered for the worse. Among other applications, this construction allows us to give the formal meaning of reduction to multiple inheritance.

The remainder of this paper is structured as follows. Section II introduces the AT's model for nondeterministic processes as defined in [Henn 85] and the reduction relation [Brin 86] with some variations in the notations. The multiple inheritance construction for AT's is introduced in Section III as well as some of its properties. In Section IV, we conclude by a comparison with related works and discuss the applications of the multiple inheritance operator.

II. Processes and ordering relations

II. 1 Acceptance Trees model for nondeterministic processes

The AT's model for nondeterministic processes was introduced by Hennessy [Henn 85]. It consists of certain kind of rooted trees where both the nodes and the branches are labeled. The information about the possible nondeterministic choices are held by the nodes. In this paper, we consider only closed nodes [Henn 85]. We don't take into account divergent processes. In other words, the processes we consider are completely defined. For the remainder of this paper, we use the terms AT, process and specification as synonyms and we adopt the following notational conventions:

- We assume an universal non-empty set of actions L,
- Actions are denoted by a, b, c, ...,
- A trace t is a sequence of actions,
- ε: represents the empty trace,
- $t_1.t_2$: denotes the concatenation of traces t_1 and t_2,
- ø: represents the empty set.

Graphically, an AT is seen a rooted tree, where the branches are labeled by elements of L and the nodes by elements of the powerset of L (*P*(L)) called acceptance sets [Henn 85]. Alternatively, we represent an AT or a process as a set of pairs <t, A>, where t is a trace and A is an acceptance set. An acceptance set represents the potential sets of future actions, after t. Because of nondeterminism, the set A may have more than one element. The process can reach different internal states after performing t.

An AT satisfies the following consistency constraints [Henn 85]:

- C_0: ¢ <t, A>, A ≠ ø,
- C_1: ¢ <t, A>, A_1 □ A, and a □ A_1, there is one and only one <t.a, A'>,
- C_2: ¢ <t.a, A'>, ¡ <t, A>, such that A_1 □ A and a □ A_1,
- C_3: A is closed under union, that is, ¢ <t, A>, if A_1, A_2 □ A then A_1 " A_2 □ A,
- C_4: A is convex-closed, that is, ¢ <t, A>, if A_1, A_2□A and A_1⁄A_3⁄A_2, then A_3□A.

As an example, the set {<ε, {{a}}>, <a, {{b}, {c}, {b, c}}>, <a.b, {ø}>, <a.c, {ø}>} is an AT. It models a process, which may accept only b or c after executing a, depending on the reached state. It is represented graphically by the labeled tree of Figure 1.

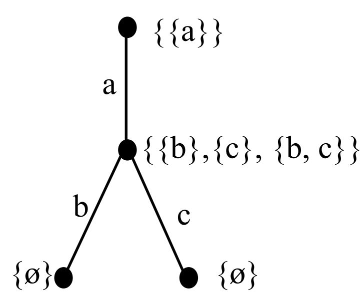

Figure 1. Example of an AT

II. 2. Ordering relations

Ordering and equivalence relations represent an important issue for process theory. From the practical point of view, these relations are used to define which process may be substituted by which other process without invalidating certain properties of the overall system. When processes are considered as abstract specifications, these relations are also used to define the set of valid implementations and the valid refinement steps.

Hennessy has defined an ordering relation for the AT's [Henn 85]. This relation captures the notion of "more deterministic than" between processes. Intuitively, P_1 is "more deterministic than" P_2, if P_1 and P_2 have the same set of traces, and P_1 deadlocks less often than P_2.

Definition 2.1 (P_1 is "more deterministic than" P_2, written $P_2 \leq P_1$) [Henn 85]
$P_2 \leq P_1$, iff
- ¢ $\langle t, A_2 \rangle$ □ P_2, ¡ $\langle t, A_1 \rangle$ □ P_1, and
- ¢ $\langle t, A_1 \rangle$ □ P_1, ¡ $\langle t, A_2 \rangle$ □ P_2, such that A_1 / A_2.

The first condition makes this relation a bit stronger than needed. Dropping it leads us to the reduction relation defined in [Brin 86]. Indeed, P_1 reduces P_2, if only if, the traces of P_1 are included in the traces of P_2 and P_1 deadlocks less often than P_2. This is what the second condition in Definition 2.1 is about. The reduction in terms of AT's is defined, formally, as follows:

Definition 2.2 (Reduction, written red)
Given two processes P_1 and P_2,
P_1 red P_2 iff ¢ $\langle t, A_1 \rangle$ □ P_1, ¡ $\langle t, A_2 \rangle$ □ P_2 such that A_1 / A_2 .

Putting P_1 in a context expecting P_2, will not change it for the worse. Despite P_1 has fewer traces than P_2, at most it may block only where P_2 may also block. In [Brin 86], the reduction relation is defined in terms of Labeled Transition Systems [Kelle 76]. It is easy to prove that both definitions are equivalent, given a mapping between these two models. It is also obvious that the reduction relation is a partial order over AT's.

III. Multiple inheritance construction

As mentioned before, we use the terms AT, process and specification as synonyms. In this section, we will introduce the multiple inheritance construction. It will allow us to derive, whenever it is possible, a process from a given set of processes. The derived processes, whenever it exists, reduces the processes from which it is derived. Indeed, there exist some situations where the multiple inheritance operator can not be applied. This is the case when the given processes are not compatible. Intuitively, two or more processes are said to be compatible, if and only if after any common trace, they may at least reach one common internal state.

Definition 3.1 (Compatibility between n processes)
AT_1, AT_2,..., AT_n are compatible, iff
¢ t, A_1, A_2, ..., A_n, we have (<t, A_1> □ AT_1 and <t, A_2> □ AT_2, ...and
<t, A_n> □ AT_n implies that A_1 (A_2 (.... ($A_n \neq ø$).

Figure 2 shows processes, which are two by two compatible, but incompatible all together.

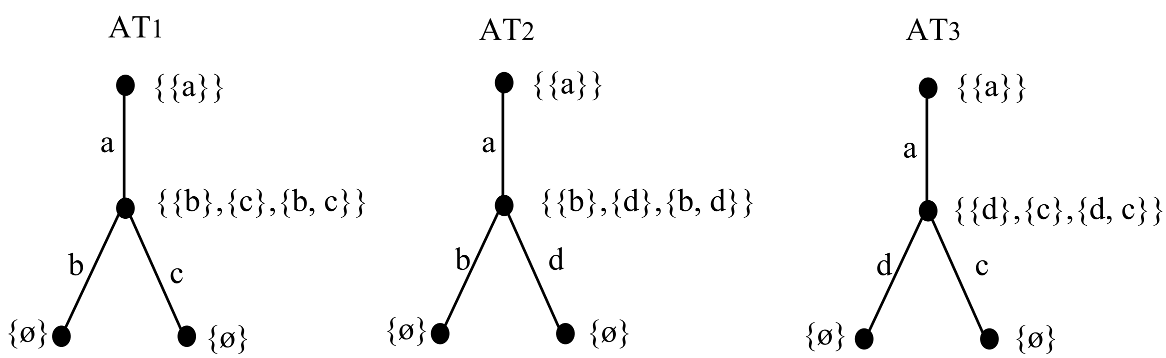

Figure 2. Examples of compatible and incompatible processes

Definition 3.2 (Multiple inheritance, written INH(AT_1, AT_2, ..., AT_n))
Given n compatible processes, AT_1, AT_2, ... and AT_n, we define formally
INH(AT_1, AT_2, ..., AT_n) = {<t, A> | A=(Ai | <t, Ai> □ AT_i, for i = 1, ..., n }.

Proposition 3.1
If AT_1, AT_2, ... and AT_n are compatible AT's, then INH(AT_1, AT_2, ..., AT_n) is an AT.

Proof

To prove that INH(AT_1, AT_2 ..., AT_n) is an AT, we have to prove that it satisfies the consistency constraints C_0, C_1, C_2, C_3, C_4.

- C_0: By definition of compatibility between AT's, if AT_1, AT_2 ..., AT_n are compatible, the intersection construction for a given trace t will not yield A = ø.

- C_1: Given a pair <t, A> □ INH(AT_1, AT_2 ..., AT_n), A_1 □ A, and a □ A_1, we have to prove that there exists one and only one pair <t.a, A'> □ INH(AT_1, AT_2 ..., AT_n).

<t, A> □ INH(AT_1, AT_2 ..., AT_n) implies that ¡ <t, A_i> □ AT_i such that A = (A_i , for i = 1, ..., n. This implies that A_1 □ A_i , for i = 1, ..., n. The AT_i for i = 1, ..., n are AT's, it follows that there exists one and only <t.a, A_i'> □ AT_i , for i = 1, ..., n.
AT_1, AT_2 ..., AT_n are compatible, it follows that there exists one and only one <t.a, A'> □ INH(AT_1, AT_2 ..., AT_n) such that A' = (A_i' ≠ ø, for i = 1, ..., n.

- C_2: Given a pair <t.a, A'> □ INH(AT_1, AT_2 ..., AT_n), we have to prove that ¡ <t, A> □ INH(AT_1, AT_2 ..., AT_n), such that ¡ A_1 □ A and a □ A_1.

<t.a, A'> □ INH(AT_1, AT_2 ..., AT_n) implies that ¡ <t.a, A_i'> □ AT_i such that A' = (A_i', for i = 1, ..., n. The AT_i , for i = 1, ..., n are AT's, it follows that ¡ <t, A_i> □ AT_i, such that ¡ A_{i1} □ A_i and a □ A_{i1} , for i = 1, ..., n. AT_1, AT_2 ..., AT_n are compatible, it implies that ¡ <t, A>□ INH(AT_1, AT_2 ..., AT_n) such that A = (A_i ≠ ø, for i = 1,..., n.
A ≠ ø , ¡ A_1 □ A, which also belongs to all A_i , for i = 1, ..., n. We have A_{i1} □ A_i (and a □ A_{i1} , for i = 1, ..., n) and the AT_i satisfy the constraint C_3, it follows that A_1 " A_{i1} □ A_i, for i = 1, ..., n. We have A_1 ⁄ A_1 " {a}⁄ A_1 " A_{i1}, because a □ A_{i1} , for i = 1, ..., n. By constraint C_4, it follows that A_1 " {a} □ A_i , for i = 1, ..., n, then A_1 " {a} □ A.

- C_3: Given a pair <t, A> □ INH(AT_1, AT_2 ..., AT_n), we have to show that if A_1, A_2 □ A then A_1 " A_2 □ A_P.

If <t, A> □ INH(AT_1, AT_2 ..., AT_n), then ¡ <t, A_i> □ AT_i, such that A = (A_i ≠ ø , for i = 1, ..., n. If there exist A_1, A_2 □ A, it means that there exist A_1, A_2 □ A_i, for i = 1, ..., n.

The AT_i, for i = 1, ..., n, are AT's, they satisfy constraint C_3, which means that A_1 " A_2 □ A_i, for i = 1, ..., n. It follows A_1 " A_2 □ A, since A = (A_i , for i = 1, ..., n.

- C_4: A is convex-closed, that is, ¢ <t, A> □ INH(AT_1, AT_2 ..., AT_n) , if A_1, A_2 □ A and $A_1 / A_3 / A_2$, then A_3 □ A.

It is similar to the proof of constraint C_3. ←

Figure 3 shows an example. The AT drawn in this figure is derived from AT_1 and AT_2 of Figure 2 by application of the multiple inheritance operator INH. It is straightforward to verify that INH(AT_1, AT_2) reduces AT_1 and AT_2.

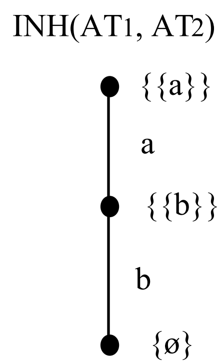

Figure 3. Example of multiple inheritance

In the following, we list some properties of the multiple inheritance construction.

Proposition 3.2
- AT_1 = INH(AT_1) = INH(AT_1, AT_1),
- If AT_1, AT_2 are compatible, then INH(AT_1, AT_2) =INH(AT_2, AT_1),
- If AT_1, AT_2, AT_3 are compatible, then

INH(AT_1, INH(AT_2, AT_3)) = INH(INH(AT_1, AT_2), AT_3) = INH(AT_1, AT_2, AT_3).

Proofs
They follow from Definition 3.2. ←

Proposition 3.3

If AT_1, AT_2, ..., AT_n are compatible, then

INH(AT_1, AT_2, ..., AT_n) red AT_1,

INH(AT_1, AT_2, ..., AT_n) red AT_2,

...

INH(AT_1, AT_2, ..., AT_n) red AT_n.

Proof

It follows from Definition 3.2 ←

Proposition 3.4

If AT_1, AT_2, ..., AT_n are compatible, then INH(AT_1, AT_2, ..., AT_n) is the "largest" common reduction of AT_1, AT_2, ... and AT_n.

Proof

Given a set of compatible processes AT_1, AT_2, ... and AT_n, we have to prove that for any process AT_x, if AT_x red AT_1, AT_x red AT_2, ... and AT_x red AT_n, then AT_x red INH(AT_1, AT_2, ..., AT_n).

Let AT_x be a process such that AT_x red AT_1, AT_x red AT_2, ... and AT_x red AT_n,

AT_x red AT_1, it follows from Definition 2.2 that ¢ <t, A_x> □ AT_x, ¡ <t, A_1> □ AT_1 such that A_x ⁄ A_1,

AT_x red AT_2, it follows from Definition 2.2 that ¢ <t, A_x> □ AT_x, ¡ <t, A_2> □ AT_2 such that A_x ⁄ A_2,

... and

AT_x red AT_n, it follows from Definition 2.2 that ¢ <t, A_x> □ AT_x, ¡ <t, A_n> □ AT_n such that A_x ⁄ A_n,

It follows that ¢ <t, A_x> □ AT_x, ¡ <t, A_1> □ AT_1 , ¡ <t, A_2> □ AT_2,..., ¡ <t, A_n> □ AT_n, such that A_x ⁄ A_1, A_x ⁄ A_2, ... and A_x ⁄ A_n,

From Definition 3.2 of multiple inheritance construction, it follows that ¢ <t, A_x> □ AT_x, ¡ <t, A> □ INH(AT_1, AT_2, ..., AT_n),with A = (A_i, for i = 1, ..., n,

Since A_x ⁄ A_i, for i = 1, ..., n, it follows that A_x ⁄ A,

Consequently AT_x red INH(AT_1, AT_2, ..., AT_n).

IV. Related works, applications and discussion

In some previous work [Boch 91], the issue of multiple inheritance with a conformance constraint was pointed out by Bochmann. For given class specifications C1, C2, ..., Cn, the conformance relation is defined such that INH(C1, C2, ..., Cn) conforms to C1, C2, ... and Cn. However, this conformance relation concerns only trace properties, whereas deadlock properties are not preserved. In other words, INH(C1, C2, ..., Cn) can not perform a trace of actions, which can not be performed by C1, C2, ... and Cn. However, it can deadlock in some situations where no one of C1, C2, ... and Cn will deadlock.

The multiple inheritance operator INH, introduced in this paper, is given in terms of AT's. Therefore, it can be applied for any process algebraic language where the processes can be interpreted by AT's.

Since the introduction of LOTOS [ISO 8807], the constraint oriented specification style has been promoted [Brin 89]. Different constraints on a component are described separately. The parallel operator is used as logical AND to connect all these constraints. The properties concerned by this conjunction are limited to the trace properties. If we consider as example of constraint process P1 of Figure 4, and compose it with itself by the parallel operator for a logical conjunction of constraint P1 with itself, we obtain process P2 of Figure 4. Their corresponding representations in terms of AT's are given by AT_1 and AT_2 of Figure 4, respectively. We have P1 (AT_1) red P2 (AT_2), but not the opposite. A conjunction of a constraint with itself doesn't lead to this constraint. This is due to the presence of nondeterminism in P1 and the operational aspect of the parallel operator. However, this is not the case for the multiple inheritance construction INH introduced in this paper, which plays a role of a conceptual operator for processes or constraints construction. We have AT_1 = INH(AT_1, AT_1).

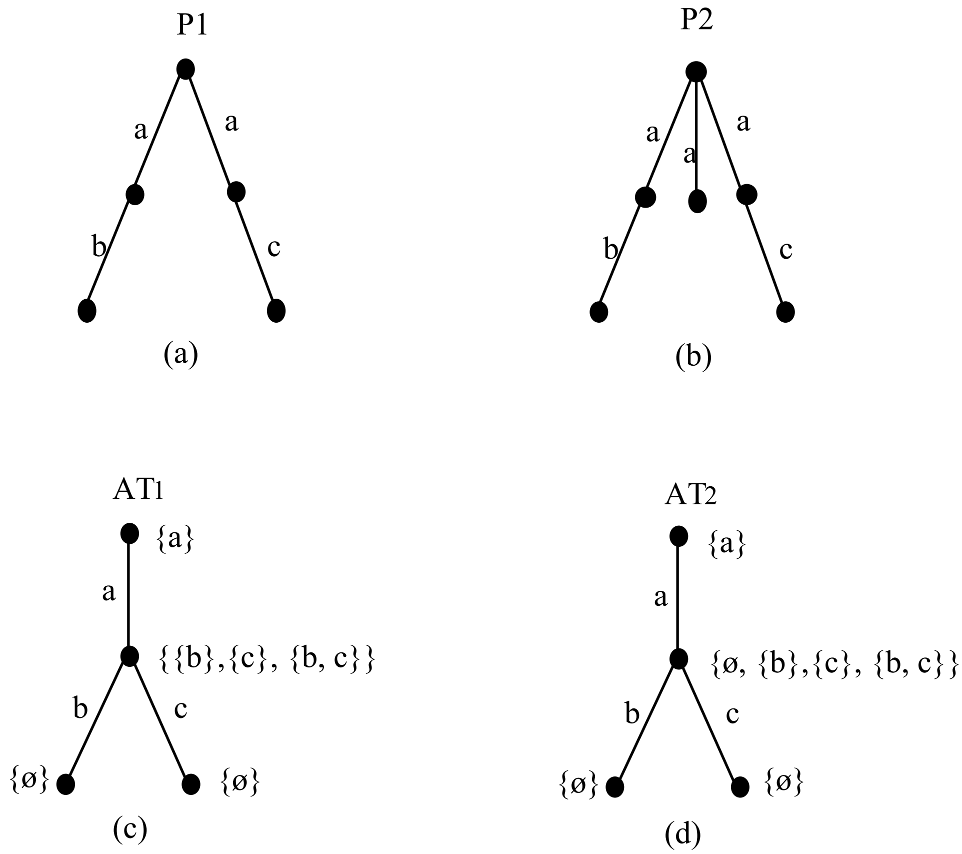

Figure 4. Constraint oriented specification style in LOTOS

Some similar work has been done in [Ichi 90]. They introduced a new construction H, which allows to build, incrementally, a new process P from two given processes P1 and P2 (P = P1 H P2), such that P1 and P2 are both extended by P. The extension is another ordering relation between processes defined in [Brin 86]. It concerns, specially, partial processes, which can be extended by adding new traces, without altering the deadlock properties for the worse. In other words, the new process has more traces and deadlocks less often than the previous one. The process P1 H P2 exists for any pair of processes P1 and P2. The construction H plays the role of our INH, and the extension relation plays the role of conformance. The subset of LOTOS adopted in this work does not include nondeterministic processes. The same relation is also chosen in [Cusa 89a] to introduce the concept of inheritance in an object oriented interpretation of LOTOS.

In some specification techniques [Brin 86, Quem 91], options are modeled as nondeterministic choice. For given specifications P1, P2, ... and Pn, compatibility means they have options in common. These common options are described by INH(P1, P2, ..., Pn). This is the "largest" common set of options, as stated by Proposition 3.4. This means no further design decisions are made. In the same manner, the multiple inheritance construction could be used to define the common subset service for interworking, given two service specifications [Boch 90]. In this case, some renaming of service primitives will be necessary.

When the reduction is taken as an implementation relation, the construction introduced in this paper allows us to define, whenever it is possible, a common implementation INH(P1, P2, ..., Pn) for a given set of specifications P1, P2, ... and Pn. This yields the advantage of handling only one implementation for a set of specifications and avoiding the interoperability problem between different implementations.

Acknowledgements
The authors would like to thank A. Schaff, P. de Saqui-Sannes and R. Fournier for their helpful comments on drafts of this paper. This research is funded by the Canadian Institute for Telecommunications Research (CITR).

References

[Amer 87] P. H. M. America, Inheritance and Subtyping in a parallel object-oriented language, ECOOP'87, pp. 235-242.

[Amer 89] P. H. M. America, A behavioural approach to subtyping in object-oriented programming languages, Philips Journal Research , Vol. 44, No. 2/3, pp. 365-383.

[Boch 90] G. v. Bochmann and P. Mondain-Monval, Design Principles for Communication Gateways, IEEE Journal on Selected Areas in Communications, Vol. 8, No. 1, January 1990.

[Boch 91] G. v. Bochmann, On the specialization of object behaviors, in J.Palsberg & M.I.Schwartzbach (ed.) Types, Inheritance and Assignments, a collection of position papers from the ECOOP'91 workshop W5, Geneva, Switzerland, July 1991.

[Brin 86] E. Brinksma, G. Scollo and S. Steenbergen, LOTOS specifications, their implemetations and their tests, Protocol Specification, testing and verification, VI, Montréal, Canada, 1986, Sarikaya and Bochmann (eds.).

[Brin 89] E. Brinksma, Constraint oriented specification in a constructive formal description technique, in Stepwise refinement fo distibuted systems, Models, Formalisms, Correctness, REX Workshop, Mook, The Netherlands, Bakker, Roever and Rozenberg (eds.).

[Cusa 89a] E. Cusack, S. Rudkin and C. Smith, An object oriented interpretation of LOTOS, FORTE'89,Vancouver, Canada, Vuong et and al.(ed.), pp. 265-284.

[Cusa 89b] E. Cusack, Refinement, Conformance and Inheritance, Workshop on the Theory and Practice of Refinement, Open University, 1989.

[DeNi 87] R. De Nicola, Extensional equivalences for transition systems, Acta Inform. 24 (1987), pp. 211- 237.

[Henn 85] M. Hennessy, Acceptances Trees, J. of ACM, Vol.32, No. 4, Oct. 1985, pp. 896 - 928.

[Hoar 78] C. A. R. Hoare, Communicating Sequential Processes, Comm. of the ACM 21 (1978), pp. 666 - 677.

[Ichi 90] H. Ichikawa, K. Yamanaka and J. Kato, Incremental Specification in LOTOS, IFIP Protocol Specification, Testing and Verification X (1990), Ottawa, Canada, Logrippo, Probert and Ural (ed.).

[ISO 8807] ISO - Information Processing Systems - Open Systems Interconnection, LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, DIS 8807, 1987.

[Kell 76] R. Keller, Formal verification of parallel programs, Comm. of the ACM 19 July 1976, pp. 371-384.

[Miln 80] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, Berlin, 1980.

[Quem 91] J. Quemada, A. Azcorra And S. Pavon, Development with LOTOS, Tutorial, FORTE '91, Sidney, Australia.

[Wegn 88] P. Wegner and S. B. Zdonik, Inheritance as an incremental modification mechanism or what like is and isn't like, ECOOP'88, pp. 55-77.