— Extended Position Paper —

Formal Specification of Distributed Information Systems

Jan Vis Ed Brinksma Rolf A. de By
{vis, brinksma, deby}@cs.utwente.nl

Universiteit Twente

May 6, 1994

Abstract

The design of distributed information systems tends to be complex and therefore error-prone. However, in the field of monolithic, i.e. non-distributed, information systems much has already been achieved, and by now, the principles of their design seem to be fairly well-understood. The past decade has shown also remarkable progress in the development and application of formal methods for distributed systems, in particular in the area of protocol systems. For both application areas techniques and tools have been developed that have been accepted by considerable user communities.

The project we describe here aims to study the combination of two formalisms that have been (largely) developed at the University of Twente, viz. the process algebraic protocol specification language LOTOS and the object-oriented database specification language TM. Its objective is to combine the strengths of both formalisms and their associated tools for the specification, verification, testing, and design of distributed information systems.

Overview of this paper

Specification of distributed information systems appears to be too complex to deal with on a mere intuitive basis. The TM/LOTOS integration project aims at formal specification of such systems. The two main objectives of this paper are:

- to outline the TM/LOTOS project and compare it to other approaches, and
- to present several ideas and a few results in more detail.

We state the objectives and context of this project in section 1. Afterwards we present the main characteristics of LOTOS and TM.
Then, in section 2, we discuss the scientific relevance of this project, and compare it to a number of other approaches.
Finally, in section 3, we present a few results and some ideas for further research.

1 Research objectives and context

1.1 Two views on distributed information systems

The issue of distributed information systems can be approached from two different points of view. We can consider such systems as:

- **information systems**
  to deal with *data structures* and *database transactions.*
- **distributed systems**
  to deal with *concurrency* and *distributed processes.*

Each of these approaches in isolation would give an incomplete view of a distributed information system, since neither of them allows dealing with all of the relevant concepts.

Before discussing the integration of both points of view, we want to emphasise that the respective areas are indeed complex by themselves. To gain better control of the complexity and correctness of designs, the use of formal methods has been widely advocated, and a considerable number of such methods has been proposed and/or elaborated for various application areas of computer science. Some of these formal methods have proven their capabilities in practice too.

In particular, formal techniques and tools have been developed for specifying both monolithic information systems and distributed systems. Some of these formal techniques and tools have been accepted by considerable user communities, and a few have even been the subject of standardization.

A next, and opportune, step in this development is the application of formal techniques to the specification and analysis of distributed information systems. The idea to combine formal methods developed for the two above-mentioned categories into a unified framework is a logical approach that could profit from the considerable work that has already been done.

We are investigating this idea in the context of a pair of formalisms that both represent state-of-the-art approaches in their respective fields of application, viz. TM and LOTOS. (In the following sections, we will give an overview of these two specification languages.) The specific objectives of this research project are:

- to develop a formalism for distributed information system design, based on the process algebraic approach of LOTOS and the data modelling capabilities of TM,
- to develop a general architectural and analytical theory of distributed information systems based on the TM/LOTOS formalism, and
- to study the support of the design of distributed information systems by methods and tools based on the TM/LOTOS formalism.

In the following sections, we describe the main features of both LOTOS and TM. In section 3, while discussing the particulars of several research topics, we will go into details wherever necessary.

1.2 LOTOS — specifying distributed systems

LOTOS is an internationally standardized formal description technique (ISO 8807) for the specification of distributed, concurrent systems. In particular, it has been developed for the formal definition of OSI service and protocol specifications.

LOTOS is based on a process algebraic model that allows for an explicit treatment of parallelism and nondeterminism, in a context of *multi-way synchronization.* The model induces a powerful analytical theory that can be applied to the analysis and transformation of specifications. This has enabled the development of appropriate design methods and a sophisticated tool environment, which give LOTOS great practical strength in spite of its complete formal semantics. The pure process algebraic sublanguage, called *Basic LOTOS*, has been orthogonally extended with the abstract data type *(ADT)* formalism ACT ONE, for the definition of data structures that are used both for the definition of communication data and as parameters of process definitions (cf. [ISO 1989, EVD 1989, BoBr 1987, Br-a 1990]), thus providing a mechanism powerful enough to model even databases. In itself, therefore, LOTOS provides a complete formal system for the description of distributed information systems (cf. [WiSi 1990]).

In practice, however, information systems are hardly ever specified using ADT's, whereas relational, entity-relationship, or – more recently – object-oriented languages are most frequently used. Just the redefinition of such data models in terms of ADT definitions would already be an enormous task. We neither expect such an approach to result in a more thorough understanding of distributed information systems, nor in design supporting tools that are adequate for practical use in this area.

Moreover, the use of an ADT formalism for the specification of data structures in LOTOS has been a source of criticism in the context of more pragmatic applications of the language where the use of non-constructive specification features at a high abstraction level is less relevant. Also from this perspective, there exists interest in the combination of LOTOS with other formalisms for the representation of data types. Experience with such alternative combinations could be used as input material for a revision of the International Standard (ISO standards undergo revision procedures every 5–6 years).

1.2.1 Short description of LOTOS

Basically, the LOTOS language consists of *behaviour expressions*, which are used to describe the possible *event sequences*, structured as a tree of events, each node in this tree representing the choice between a number of allowed sequences, and the edges – directed from the root towards the branches – representing events. These behaviour expressions are used to specify processes. An event is supposed to be atomic, and participation of one or more processes in the event is indicated by *action names*, occurring in the respective behaviour expressions. Furthermore there is a *choice* operator, indicating that a process can choose between several behaviours, in interaction with its environment. An *inactive process* can be specified explicitly. In fact these three features are sufficient to specify any finite set of finite sequences of events. The concept of *process abstraction* has been added for (recursively) specifying infinite event sequences, as well as infinite *sets* of allowed event sequences. The scope of action names can be restricted by means of *hiding.*

So far, we mentioned only features, which most – if not all – process algebra languages have in common. The particular strength of LOTOS lies in its more sophisticated operators for parallelism, like *enabling*, *disabling*, and *restricted synchronization.*

For every finite set of action names, a restricted synchronization operator exists, which takes two behaviour expressions as its arguments, yielding a new behaviour expression, which is defined as the parallel execution of its two arguments, while both expressions must synchronize on exactly those events, whose action names occur in the synchronization set of the operator.

The binary disabling operator indicates that the behaviour of its left-hand operand, which is initially active, can be disrupted at any point, by any initial action of the right-hand behaviour expression, which then remains active.

The binary enabling operator, which expresses sequential composition of processes, indicates that the behaviour of its left-hand operand, which is initially active, can by means of a special *exit action* pass control – possibly with a value – to the right-hand behaviour expression.

Actions within a behaviour expression can be *guarded* by boolean expressions in the ADT language. Participation of the process in such an event can only take place, if the result of the guard is True. The guard can also be a so called 'simple equation', which just checks the equality of two values.

Another feature of LOTOS processes is that *relabeling* of all action names is explicit in each instantiation of a process, instead of introducing a separate operator for this purpose.

1.3 TM — specifying monolithic information systems

The TM language has been developed for describing conceptual schemas of object-oriented databases ([BaBZ 1991]). Its strength stems from its richness as a specification language and its formal, type-theoretic background. The TM language incorporates state-of-the-art features of object-oriented data models, such as complex objects and multiple inheritance, but it also extends known models to general set constructions, in the context of full static type-checkability. TM is equipped with its own design methodology and various tools have been developed for the TM language, amongst others a type-checker and a prototype generator.

Standardization has, until now, hardly had any influence on the formal description of information systems. Some form of de facto standardization has occurred with respect to the relational data model. This model, however, appears to be inconvenient for the modelling of complex data. Beyond the relational model no standard approaches exist, and it may take some time before consensus on a more sophisticated standard formalism will be reached.

One of the main challenges in this respect, that has been identified more recently, is the problem of making distinct information systems co-operate. This is usually referred to as the 'interoperability problem'. There exists a firm belief that this problem can be solved using an object-oriented data modelling paradigm, which subsumes the (e.g. relational) data models of existing information systems.

In object-oriented database systems, programming is merged with data-structuring, enabling the designer of a database to have all the advantages of a clean conceptual design, as well as the possibility of enforcing better software engineering. In order to integrate database systems and programming languages, it was necessary to unify the database (or information system) concept of *data model* with the programming-language concept of *type system*.

TM aims at providing such a unifying framework. It distinguishes itself from many other data models in having a completely defined formal semantics. The formal model has been based on Cardelli-Wegner type theory ([CaWe 1985]). By extending existing object-oriented data models with a logical formalism and general set constructs, TM offers a wide range of possibilities for describing database schemas in an object-oriented framework. In contrast to the ADT theory of LOTOS, which is based on the construction of initial algebraic models, the TM model has been given a purely declarative, set-theoretical semantics. We believe that the latter approach may lead to more intuitively appealing data models in the specification of information systems.

Another reason for using a language in which complex objects can be directly defined stems from the observation that databases typically consist of large collections of data showing many interrelationships. In databases, these interrelationships deal with various kinds of referential integrity and the organization of data in so-called generalization/specialization hierarchies. In object-oriented type theory, this hierarchical structure is described by means of a subtyping relation. In ADT theory, there also exists a need for expressing interrelationships between data by employing subtyping. Although there exist approaches to incorporate a notion of subtyping in ADT theory – most notable is the approach taken in the OBJ model ([FGJM 1985]) – the resulting systems lack readability due to heavy use of mathematical formalism (e.g. category theory [SFSE 1989]), which can, in our opinion, be avoided.

1.3.1 Short description of TM

TM knows, besides *basic types* like boolean and integer, several ways to compose types: *records*, *variants*, and *functions*. Furthermore *sets* and *lists* can be defined over any type, in particular predicative sets can be constructed.

In addition, TM knows *sorts*. A sort has an underlying type, determining the general structure of its instances, and it may also contain *constraints* on this structure, and *methods* that extract information from it.

In TM, *objects* of some sort can be defined, and structured in a *class*. Each object can be identified uniquely. The class can have additional constraints and methods.

These are the basic building blocks of TM databases, and a wide range of operators has been defined to access these structures. Furthermore, the database can have its own constraints and methods.

1.4 Tool integration

Various tools are (or have been) developed around TM:

- A TM type-checker has been implemented to offer support for correct typing of a TM database.
- A support tool has been implemented that automatically generates a prototype of a database specified in TM and it offers facilities with which the user can test the prototype and thus the database specification.

- Within the ESPRIT-3 project IMPRESS, a tool is developed that can verify whether the integrity constraints in a certain database state are actually satisfied. These constraints apply both to static (data-structural) and dynamic (transaction-dependent) aspects of specifications written in TM.

For LOTOS the powerful LOTOS Integrated Tool Environment (LITE) has been developed. It consists of a considerable number of LOTOS related tools for the specification, analysis, simulation, transformation, compilation, and testing of LOTOS processes. It is intended that a number of these tools will be redesigned for the TM/LOTOS-formalism. In addition to the experience that was gained from the development of LITE, it is expected that (parts of) existing tools and the architecture of LITE can be re-used. This seems feasible as many tools have been developed on the basis of powerful meta-tools that can also be applied to the new language definition (e.g. the syntax-driven editor), and because the interface between the process algebraic and the data type parts of LOTOS is well-defined and based on simple concepts. A crucial design tool such as the LOTOS simulator can be adapted relatively easy on the basis of a TM expression evaluator.

Some tool functionalities will require more effort, such as the development of a static semantics checker for the TM/LOTOS formalism (this will include the functionalities of the TM type checker).

2 Scientific Relevance

The viability of a process algebraic approach to the specification of information systems has already been illustrated by the work of Wieringa [Wier 1990]. There, the process algebra ACP [BeKl 1985] has been used to model concurrency and control flow aspects of conceptual models. An important difference between ACP and LOTOS is, that the former has been developed as a basic calculus for the theoretical study of concurrency. In ACP the formal expressivity and generality of the model have been important criteria.

LOTOS, on the other hand, has been developed specifically to support the structured design of distributed systems in terms of general architectural concepts, as explained in [VSSB 1991, Br-b 1990]. This has led to considerable differences in the treatment of, for example, parallel composition and the modelling of disruptive behaviour. As a result, LOTOS specifications of distributed systems can reflect the informal design structure quite closely, and tend to be more concise than their ACP counterparts. We expect that the same advantages of LOTOS apply in the case of distributed information systems.

We compare our approach to other attempts to model distributed information systems:

2.1 Petri net based approaches

Petri net-based descriptions of distributed information systems (cf. [Reis 1985]) have proved to be succesful in describing prototype versions. An important disadvantage of classical Petri nets is, however, that descriptions of realistically sized systems tend to become unmanageable in size because of the lack of suitable modularity and abstraction features. Also the definition of data structures poses problems in the context of Petri nets. To overcome such problems some of the Petri net formalisms are extended with expressive annotation languages (cf. [Jens 1992]). This improves their applicability, though it complicates the formal semantics.

Petri nets offer a conceptually clear solution to the problem of real concurrency, while LOTOS has an interleaved semantics. An alternative semantics, which takes real concurrency into account, has been defined for Basic LOTOS (cf. [Lang 1992]), but for full LOTOS, such a semantics is still under development. In many cases, however, there is no need to consider true concurrency.

A strong point that Petri nets have in common with our approach, is the availability of a considerable body of analytical theory and tools that can be used for simulation and verification. We feel however, that LOTOS offers a more natural approach in the context of databases than Petri nets, due to the high level of abstraction that can be reached in specifications. In particular we refer to the possibility of separately expressing constraints according to the so called *constraint-oriented specification style* of LOTOS (cf. [VSSB 1991, Br-b 1990]).

2.2 Category theoretical approaches

Category Theory provides for a powerful modelling tool in the sense that it is completely formal. Work in the area of describing information systems in terms of Category Theory has been performed by, for example, Sernadas and Ehrich (cf. [SFSE 1989]). Category Theory in itself is actually too formal to be used as a practical tool on a large scale for the average designer. Rather than attempting to build a usable design language on a category theoretical model, we have based ours (viz. TM) on a semantically simpler formal model, which is, though equally expressive, more geared to the database context.

3 Results and ideas for further research

We have been studying the application area of workflow management specification by means of a small, though – in our opinion – representative workflow example for loan handling by a bank. This case had been analysed to some extent (cf. [B&c. 1993]), but not formally specified. Our attempts to give a specification in TM/LOTOS strongly improved our insight in the area and resulted in several ideas concerning formal semantics for TM/LOTOS. We will discuss some of these ideas in the following sections. Until now, our efforts have mostly been directed towards understanding of the following ideas:

- Various ways to use data in LOTOS processes, and considering the consequences. We mention existing possibilities as well as ideas to extend the language. We address:
  - Communication between processes.
  - Guarding actions by TM constraints.
  - Gate substitution.
  - Process parametrization.
  - Event parametrization.
- Connection of a (monolithic) database to a process, in search for ways to link events in the process to database transactions.

3.1 Using data in process specification

3.1.1 Communication between processes

In process algebraic languages, communication between (sub-)processes is usually specified by means of value passing. Besides *value passing*, LOTOS knows value *matching* and value *generation*. Due to the use of multi-way synchronization, any number of processes, in which a particular action name occurs, can participate in a single event. Communication can be realized by adding a restriction on the set of possible communication results to each participating process. The resulting communication value of the event must satisfy the requirements of all participants, that is, the value must be in the intersection of all sets of values allowed by each process. If this intersection appears to be empty, no synchronization can take place.

In the case of one process demanding the value to be 37, and another process allowing it to be any integer value, they will synchronize, agreeing upon the value 37. Besides this principle of value passing, there are other possibilities:

For example, if two processes both demand the communicated value to be 37, of type integer, they can synchronize, but if their demands would be contradictory, no synchronization would take place. This form of synchronization is called value matching.

Another example: one process demands the communicated value, of type integer, to be larger than 3, and another process demands it to be less than 8. In this case both processes can synchronize on any of the integer values 4, 5, 6, and 7. Here the concept of value generation appears: each of these values is allowed as a result, since neither of the possible outcomes is to be preferred to any other. A more extreme example would be, two or more processes that don't restrict the value at all. In that case, an arbitrary value is generated. All participating processes however, have to agree upon the *type* of the resulting value.

Clearly, there is no general notion of 'direction of communication', although it can be modelled by this more expressive mechanism. We could still use the term 'sender' for a process that demands the communication result to be of a specific value.

Actually, any data model would serve for modelling communication in this way. The use of an object-oriented data language like TM however, tempts to introduce an even more powerful notion of communication. Instead of ensuring that all processes participating in an event put the same demand on the type of the result, we can allow different types to be offered by various processes, and take a common *subtype* of those types, that is, a lower bound according to the subtyping relation. Then the result will be of all demanded types, due to the TM axiom: if x is of type σ and σ is a subtype of τ, then x is also of type τ. Since we still want to be able to express the more straightforward forms of communication that we mentioned earlier, we must demand the resulting value to have the *greatest lower bound* of the offered types as its minimal type. Apparently, such a generalized synchronization context is only well-typed if the offered types have a greatest lower bound. Synchronization can take place, only if a value of that minimal type exists, which meets all demands of the participating processes.

This more general communication concept could for example be applied to the situation where one process creates a record with the name and address of a person, while another process creates a record with his name and salary. The communicated result would be a record with the name, address, and salary of this person.

3.1.2 Guarding actions by TM constraints

We explained, that guards can be used to place demands on a communicated value. Thus, synchronization will depend upon compatibility of data constraints. Originally, two kinds of guards occurred in LOTOS: simple equations and boolean expressions. In TM, however, simple equations are also of the boolean type, so there remains only one kind of guards.

Furthermore, TM knows the concept of *sorts*, which are in fact types extended with constraints and methods. This concept seems very suitable to express communication, since in this way, separately formulated guards can be given the semantics of constraints included in a sort. Introduction of this principle requires the powerful communication mechanism, based on greatest lower bounds of types, we proposed in the last part of section 3.1.1, since equal types with different constraints form different sorts.

3.1.3 Gate substitution

Relabeling is a standard notion in process algebraic languages. The relabeling mechanism in LOTOS is explicit, that is, all observable events in the behaviour expression of a process specification have to be mentioned in the process header, in any fixed order, and these so called *formal gate parameters* are explicitly substituted by *actual gate names* for each instance of the process. This concept offers at least some form of genericity in process definitions (besides the concept of process parametrization we mention in section 3.1.4). An example is a semaphore with formal gates 'occupy' and 'release', which can be instantiated in many different ways, for example with the actual gate names 'lock' and 'unlock'.

Integration of the process and the database concept – at least in our approach – appears to induce a strong need for genericity, in order to deal with complex database structures. One could imagine a higher degree of freedom in process specification, in order to achieve a higher level of abstraction, e.g. in situations where the number or the nature of gates is not fixed. An approach could be based on concepts of π-calculus. On the other hand, the TM concepts of set, list and variant type might be applied to gate parameters in process headers and perhaps even inside the restricted synchronization operators.

In general, this idea appears to result in too wild a semantics to our taste, though in a somewhat restricted form it might be useful. In particular, it seems sensible to allow genericity of processes only as far as all actually occurring variations can be determined statically. This is necessary, when we want to convert those specifications into already existing LOTOS concepts in a straightforward way. This idea might appear particularly useful in combination with the concept of event parametrization, discussed in section 3.1.5.

In the body of a LOTOS process, defined by some behaviour expression, only formal gates can be used instead of action names. In many cases, however, it appears that some of these gates are always instantiated by the same action name. This leads to overcomplete process headers. An idea to improve readability of – especially – top-down refined specifications might be to allow binding of formal gates, within a process body, not exclusively by that process itself, but also by another process that uses an instantiation of that process. Another way to explain this effect, is that such gates are not mentioned in the process header, but always implicitly instantiated by the action with the same name as the gate. However, we didn't yet study problems that might rise concerning static semantics checking.

3.1.4 Process parametrization

In order to allow general recursion and more genericity in process specifications, LOTOS knows the concept of process parametrization, that is, each instantiation of a parametrized process, is accompanied by the local substitution of its formal parameters. (This mechanism resembles calling ordinary recursive procedures in an imperative language.) This concept remains applicable, regardless of the data modelling formalism we use.

3.1.5 Event parametrization

A powerful concept in process algebra theory is parametrization of events. The idea is to use not merely action names to express participation in an event, but an action name extended with some parameters. In contrast to the concept of value matching (explained in section 3.1.1), where several demands on a value are related to a single event, in the context of event parametrization different values are used to indicate different events, which – as a consequence – cannot synchronize at all.

In LOTOS the concept of event parametrization is not present, although a restricted form can be simulated directly, with some effort. The main reason for the omission is, that dynamically changing synchronisation conditions could disturb, or at least complicate the validation of specifications. Nevertheless, it seems to be a natural construct in situations where many events need to be distinguished, although they play similar roles in the process structure.

3.2 Considering a database as a process

When observing an object-oriented database, we notice that it can process queries and method calls. A TM method is some kind of function, that belongs to – and operates upon – an object, a class, a module, or even to the entire database, and it can take a predetermined number of input parameters. In the case that its result is a modified version of the structure to which it belongs, it can be defined either as an update method or a retrieval method. If the structure is different, it must be defined as a retrieval method. Update methods can be used to modify the database physically, but TM also knows the concept of hypothetical updates, whose semantics is to create a temporary copy of the database, in order to apply further methods to the modified copy, but leaving the database itself intact. A query resembles a retrieval method, but can be formulated ad hoc, that is, independantly from the database.

One way to describe the behaviour of the database, is to consider each query or method call as a single event, during which the input parameters are processed, and the result is produced.

In the context of *distributed databases* however, this doesn't seem a satisfactory characterization, since a query posed to one (part of the) database might lead to the call of a method in another that is physically separated from it, and the result can be produced only after the method has committed. Therefore, it seems more appropriate to take the *call* of a method – or query – and the generation of its *result* as two different events, the former taking the input parameters, and the latter communicating the result. In this way, greater flexibility is obtained.

3.3 Planned research items

- The ideas presented in this paper must be investigated in more detail.
- Connection of several (monolithic) databases to a network process may appear to be essentially more complex than a single database, so we will attempt to generalize our concepts.
- An important research topic shall be the support of *correctness preserving transformations* (cf. [BrKa 1991, PiVi 1990, Scht 1989]) that correspond to design steps in the development of distributed information systems.
- The more traditional topics in the study of concurrent systems, such as the analysis of *deadlock*, *livelock*, and *reachability*, will also be addressed in this more specialized context.

Conclusions

- Attempts to apply TM/LOTOS to workflow management specification have lead to several ideas to extend the LOTOS language, although a careful weighing of pros and cons will be necessary.
- Incorporation of TM data modelling into the LOTOS language seems a fruitful approach to the formal specification of distributed information systems.
- Intuitively, a database in a distributed contex can be seen as a process by considering both the call of a method or query and the delivery of its result as events.
- Development of tools and technics for this TM/LOTOS formalism seems possible, and can, to a large extent, be based on existing material for LOTOS and TM.

References

[BaBZ 1991] Balsters, H., de By, R. A., Zicari, R., Typed sets as a basis for object-oriented database schemas, in: Proceedings Computer Science in the Netherlands (CSN-SION), November 7–8, 1991, Utrecht, The Netherlands.

[BeKl 1985] Bergstra, J.A., Klop, J.W., Algebra of communicating processes with abstraction, in: Theoretical Computer Science, 37(1):77–121, 1985.

[B&c. 1993] Breitbart, Y., Deacon, A., Schek, H.-J., Sheth, A., Weikum, G., Merging Application-centric and Data-centric Approaches to Support Transaction-oriented Multi-system Workflows, SIGMOD Records, Vol. 22, Nr. 3, 1993.

[BoBr 1987] Bolognesi,T., Brinksma, E., in: Introduction to the ISO Specification Language LOTOS Computer Networks and ISDN Systems, 14, 1987, pp. 25–59.

[Br-a 1990] Brinksma, E., Specification modules in LOTOS, in: Formal Description Techniques – II (ed. Vuong), pp. 101–115, North Holland Publishing Company, 1990.

[Br-b 1990] Brinksma, E., Constraint-Oriented Specification in a Constructive Formal Description Technique.

[BrKa 1991] Brinksma, E., Kars, P., From data structure to process structure, Technical Report University of Twente INF91-38, 1991.

[CaWe 1985] Cardelli, L., Wegner, P., On understanding types, data abstraction, and polymorphism, in: Computing Surveys, Vol. 17, No. 4, December 1985.

[EVD 1989] Eijk, P. H. J., Vissers, C. A., Diaz, M., The formal description technique LOTOS — results of the ESPRIT/SEDOS project, North Holland, Amsterdam, 1989.

[FGJM 1985] Futatsugi, K., Goguen, J.A., Jouannaud, J.-P., Meseguer, J., Principles of OBJ2, in: Proceedings 12^{th} ACM Symposium on Principles of Programming Languages, New Orleans, 1985.

[ISO 1989] ISO, LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, Int. Standard ISO 8807 (ISO, 1989).

[Jens 1992] Jensen K., Coloured Petri Nets — Basic Concepts, Analysis Methods and Practical Use, EATCS Monographs on Theoretical Computer Science, Springer-Verlag.

[Lang 1992] Langerak, R., Transformations and Semantics for LOTOS, Ph.D. Thesis, Universiteit Twente, Enschede, 1992.

[PiVi 1990] Pires, L.F., Vissers, C.A., Overview of the LOTOSPHERE design methodolgy, in: Proceedings of the ESPRIT Conference, pp. 371–387, Commission of the European Communities DGXIII, Kluwer Academic Publishers, 1990.

[Reis 1985] Reisig W., Petri Nets: an Introduction, Prentice Hall.

[Scht 1989] Schot, J., Systematic design of telecommunication systems using formal description techniques, in: Proceedings of the ESPRIT Conference, Commission of the European Communities DGXIII, North Holland Publishing Company, 1989.

[SFSE 1989] Sernadas, A., Fiadeiro, J., Sernadas, C., Ehrich, H.-D., The basic building blocks of information systems, in: Information System Concepts: An In-depth Analysis, Elsevier, Amsterdam, 1989, pp. 225–246.

[VSSB 1991] Vissers, C. A., Scollo, G., Sinderen, M. van, Brinksma, E., Specification Styles in Distributed Systems Design and Verification, in: Theoretical Computer Science 89 (1991), 179–206.

[WiSi 1990] Widya, I., Sinderen, M. van, On the design and formal specification of a transaction processing protocol, in: Formal Description Techniques – III (eds. Quemada, Manas, Thomas), North Holland Publishing Company, 1990.

[Wier 1990] Wieringa, R. J., Algebraic Foundations for Dynamic Conceptual Models, Ph.D. Thesis, Vrije Universiteit, Amsterdam, 1990.