Université de Liège

Faculté des Sciences Appliquées

Systèmes et Automatique

Institut d'Electricité Montefiore, B28
Université de Liège au Sart Tilman
B-4000 Liège 1 (Belgique)

Esprit Project 5341

OSI95

The OSI95 Transport Service with Multimedia support

***

A Method for Applying LOTOS at an Early Design Stage and Its Application to the ISO Transport Protocol

G. Leduc

[Led 94] G. Leduc, **A Method for Applying LOTOS at an Early Design Stage and Its Application to the ISO Transport Protocol**, *The OSI95 Transport Service with Nultimedia Support*, A. Danthine, University of liège, Belgium (ed.), Research Reports ESPRIT - Project 5341 - OSI95 - Volume 1, Springer-Verlag, 1994, pp. 151-180.

Date: July 1994
RS94-02

A Method for Applying LOTOS at an Early Design Stage and its Application to the ISO Transport Protocol

Guy Leduc
Research Associate of the National Fund for Scientific Research (Belgium)
Université de Liège, Institut d'Electricité Montefiore, B 28, B-4000 Liège 1, Belgium
Email: leduc@montefiore.ulg.ac.be

We propose a method applicable to the design of large and abstract LOTOS specifications. More precisely, this method explains how to generate a constraint-oriented specification which is an adequate abstract and modular starting point of a complex design process leading to implementation. It is illustrated on a substantial part of the ISO Transport Protocol class 4 which is considered as a complex and stable case study. Having proved the feasibility of the method on this protocol, it was then used to specify (parts of) the new transport protocol TPX of OSI95.

Keywords: LOTOS, Transport Protocol, Formal Specification, Constraint-oriented style.

1 Introduction

The purpose of this method is to give the necessary guidelines for applying LOTOS [ISO 8807] at an early design stage of distributed systems. At this stage, the specification is required to be abstract, structured and easily modified. *Abstract* means that the specification is free from implementation details: it just describes what the behaviour of the system is. *Structured* means that the specification is decomposed into well-defined sub-parts dealing with specific aspects of the system. This is helpful for understanding the global behaviour. *Easily modified* means that a local change will not require major changes in the whole specification. This characteristic is important at an early design stage because many functions are changed in or added to or removed from the system.

The method presented here does not cover the transformation process of such an initial specification into a more implementation-oriented specification. These transformations are often called refinement transformations, and have to preserve the semantics of the specification. The transformations that are discussed in the previous paragraph (i.e. addition, removal and change of a function) are of a different nature, and usually affect the design: the semantics of the specification will generally change completely.

To clarify these ideas, we will first present in section 2 our view of the full design process composed of specification, validation and implementation activities, with a special emphasis on LOTOS.

Section 3 presents several LOTOS specification styles [VSv 91] and their purpose. Among them we find the constraint-oriented style, which is the main style used throughout the rest of the report.

Section 4 presents the method and section 5 explains the application of this method on a large case study, viz. the ISO transport protocol entity (class 4) [ISO 8073]. The LOTOS specification is presented in [Led 92c]. This specification has been based on the LOTOS specification of IS 8073 [ISO 10024]. The syntax and static semantics of this specification have been checked by the ESPRIT II / LOTOSPHERE Toolset [vEi 92] and some parts have been simulated by SMILE [EeW 93]. The specification has not been validated further up to now.

2 Design Process

This section is not a survey of all the known results on design with formal description techniques. Instead it gives a brief overview of the design activities and shows how they are related.

The design of a (concurrent) system is a complex activity that starts from very abstract or general requirements on the structure and/or the behaviour of the system, and ends up with an actual implementation of them. General requirements on the structure are for instance the need to fit in an architectural model such as the OSI Reference Model [ISO 7498] for open systems. Requirements on the behaviour may be purely functional or include some performance aspects.

The design process is composed of several activities that may be divided into two categories, viz. synthesis and analysis. Synthesis is any type of creative activity dealing mainly with the conception, the formalization and the implementation of a system. The purpose of analysis is to validate (to some extent) the design at different stages. Figure 2.1 presents a simplified view of the synthesis activities in the design process. Stages are represented by shaded circles, and synthesis activities by white rectangles.

Conception remains an informal activity that usually ends up with an *informal* or semi-formal *specification* of some desired features of the system. The first interesting synthesis activity where FDTs play a central role is the *formalization* of these ideas, and the achievement of a - preferably abstract - *formal specification*. This constitutes the first main phase of the design process. Here the structuring features of the FDT are important, as well as its abstraction facilities. These two criteria, viz. abstraction and structure, are the bases of the classification of several specification styles in LOTOS [VSv 91] which will be presented in section 3. Ideally, these activities of conception and formalization should be carried out hand in hand to conceive better protocols and services. Appropriate methods for using FDTs are essential at this level. The main objective of this report is to present such a method and apply it on a well-defined case study, viz. ISO 8073 (class 4).

The second main phase of the design process is the generation of a valid implementation of the specification. The greater the degree of abstraction of both the semantic model of the FDT and the specification itself, the greater the gap between specification and implementation is. Very often, the implementation cannot be derived directly from the formal specification. It may be so, for instance, because the FDT has a non-

constructive character (e.g. logic) or because the specification is highly non-deterministic, or does not have an appropriate structure or has no structure at all. The solution to this problem is a more progressive approach, going stepwise towards an implementation. Figure 2.1 only shows the last step, referred to as the *formal implementation specification*. The successive transformations leading from an intermediate stage to the next one are synthesis activities. They may be automatic or not. A transformation however is supposed to preserve somehow the properties of the system. Examples of transformations may be a change in the structure of the specification to get closer to an implementation structure, the resolution of some non determinism or the translation into another FDT. After several transformations, the specification may be such that an automatic generation of some programming code is possible, at least for parts of the specification. This stepwise methodology is found in Software Engineering Standards where it is referred to as the Waterfall model. More details can be found in [QAP 92].

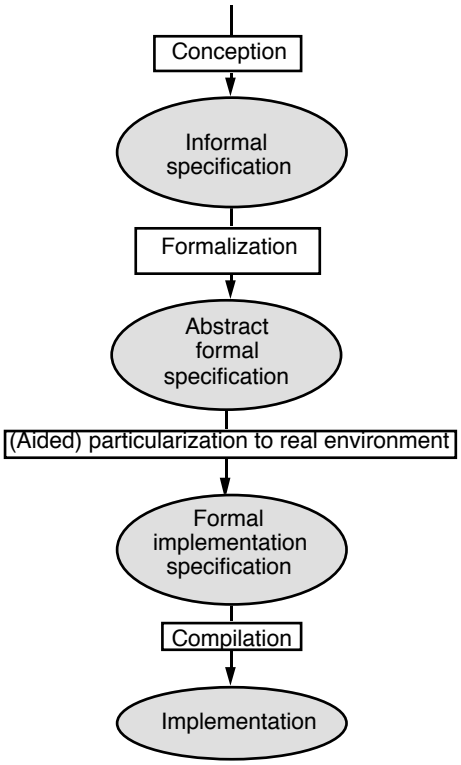

Fig. 2.1
Synthesis activities in the design process

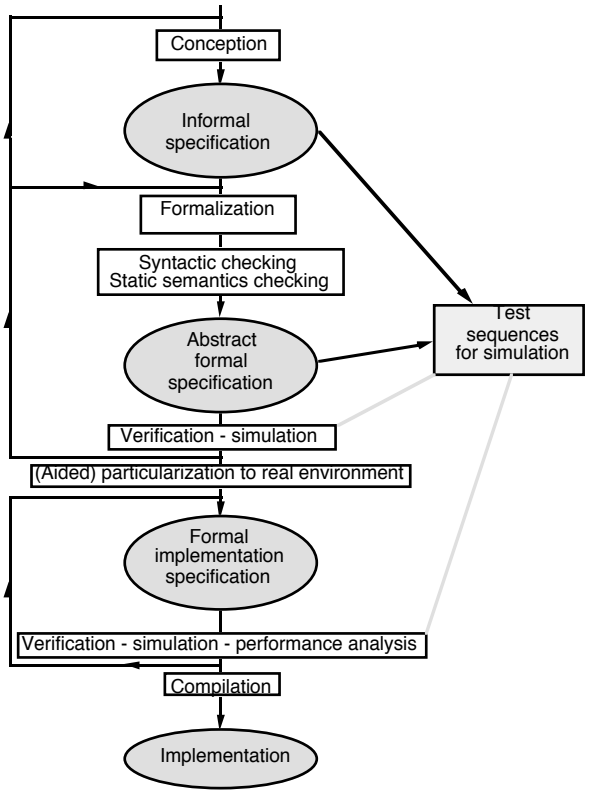

Fig. 2.2
Analysis and synthesis activities in the design process

Figure 2.2 introduces analysis activities. They are also represented by white rectangles. Moreover some useful related objects such as the *simulation sequences* (defined later) are represented by shaded rectangles.

Some analysis activities are generally required at the level of the abstract formal specification. Firstly, the specification must have a well-defined meaning, which requires correct syntax and static semantics. It turns out in practice that many conceptual errors are detected at this early stage simply because they often lead to syntactic

discrepancies for instance. However, some additional analyses are required to give more confidence and/or insight into the formal specification. Not all these analyses may be formal since the ultimate goal is to be convinced that the specification reflects the initial informal ideas. Some FDTs that have a constructive character may help in this informal validation, since the specification may often be animated or simulated. The designer may thus experiment with the specification and look at how it reacts in specific situations. FDTs with no constructive character, such as logic, usually express the specification with distinct requirements or properties. As far as each informal requirement has been translated into a formal one, this may need less (or no) machinery to be convinced that the specification is an appropriate model of the intended system.

Of course, many FDTs have been designed to support some formal reasoning about the specification, referred to as *verification*. This activity largely depends upon the FDT that is used to describe the protocol. However, the basic idea is the same: in order to verify whether a formal description is "correct", it is mandatory to define formally the meaning of this term. Besides the classical absence of pathological situations such as deadlocks, things are not correct per se, but with respect to something else usually more simple. Therefore, a formal proof of correctness starts with TWO formal descriptions: the first one is the specification that has to be verified (e.g. a protocol), whereas the second one is a simpler reference specification (e.g. a service) that the first one must satisfy. This satisfaction relation between two specifications may have different forms depending on the FDT used. When both specifications are expressed in the same FDT based on a labelled transition system (LTS) model (e.g. CCS or LOTOS), the verification consists mainly of checking some equivalence relation between the two specifications. When the second specification is expressed in an appropriate logic, model-checking techniques are used. For property-oriented specifications, some proofs of consistency may also be carried out at this stage. Other verification techniques are based on the exhaustive generation of all the reachable global states of a system description to check for the absence of pathological situations. This technique has been used with descriptions based on finite-state transition systems.

Many proof techniques cannot be applied directly to the formal specification itself, but require a preliminary translation into a tractable internal form (e.g. a labelled transition system, or a finite state automaton) that allows verification. However, for realistic specifications, this translation faces the well-known state explosion problem. Note, however, that it is now possible to analyse systems of up to 5.10^8 states [Hol 92].

During the second phase of the design process, i.e. the transformations towards the implementation-oriented specification, there may exist rules that guarantee that some properties are preserved if the transformations have a specific form. In this case, no validation[1] of the transformation is necessary. Otherwise, some verification or at least some simulation should be carried out. The simulation may be based on sequences of events (simulation sequences) generated from the initial specification [MAQ 92]. The nature of the proof is closely related to the computation model used. This proof usually consists of verifying that the behaviours described by the two design stages

[1] The term validation is used is a broader sense than verification. Validation covers any activity that contributes to give more confidence in the design, whereas verification necessitates the proof or guarantee of a certain property.

are equivalent, or that the behaviours of the less abstract specification constitute a subset of (or satisfy more properties than) the behaviours of the more abstract one. When a specification can be (semantically) mapped onto a Labelled Transition System, one may apply all the proof techniques that exist for these LTSs. These techniques are mainly based on the existence of adequate equivalences and other relations based on a concept of observation and/or implementation. There exist several reasonable ways to observe systems [dNi 87, vGl 90, Led 91]: observation equivalence [Par 85, Mil 89], testing equivalence or some related preorders [dNi 84, BHR 84, BSS 87, Hen 88] have been defined for that purpose. A general framework for dealing with this transformation or implementation process has been defined in [Led 91, Led 92b]. Examples of transformation rules are given in [Lan 90, Mas 92].

Ideally, a performance analysis ought to be carried out at several intermediate stages of the design to quantitatively support some design decisions and their associated transformation steps. However, the difficulty is to bridge the gap between the formal specification and an adequate performance model.

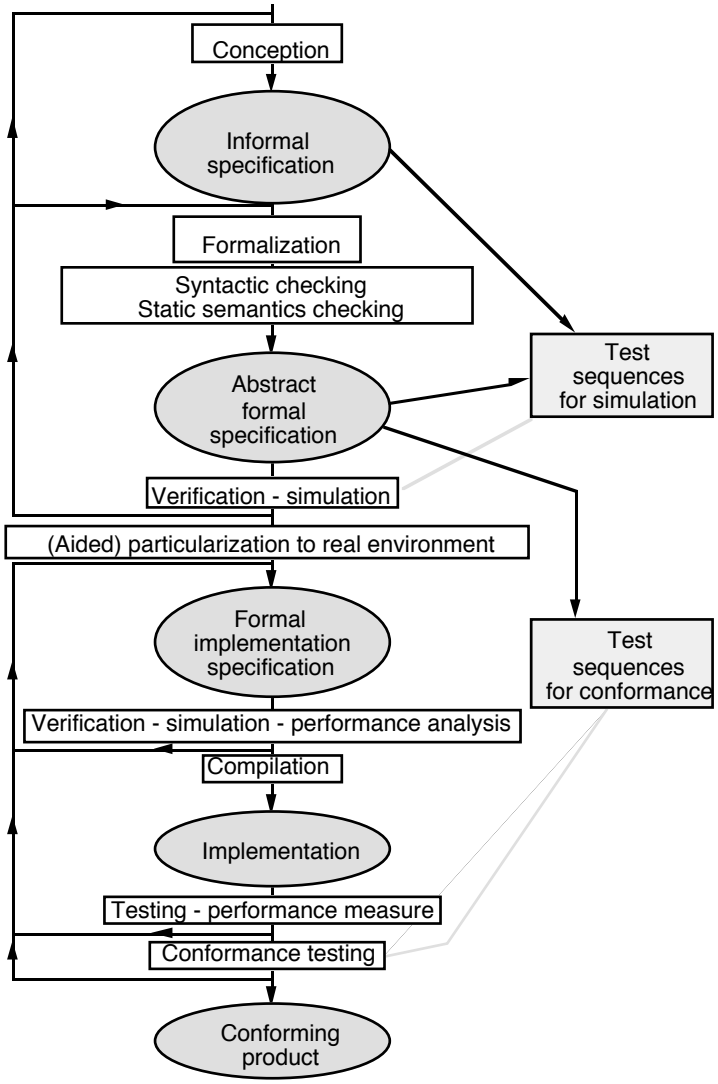

Fig. 2.3 The design process including testing

Figure 2.3 finally illustrates the complete design process with a third phase denoted testing phase. It consists of experimenting with the system implementation integrated in its real environment. This is particularly useful when the initial specification is a standard. In this case some conformance to the standard is usually required to ensure interoperability between implementations. We will only discuss dynamic conformance that is expressed by test suites. These suites describe how the implementation under test will be stimulated and how it should normally react. Ideally, these test suites should be derived from the formal specification of the standard. If the tests succeed, the implementation is claimed to conform to the standard. Note that the simulation sequences introduced in figure 2.2 may already be conformance test sequences. In LOTOS, a testing theory has been proposed in [Bri 88a], extended and brought into play in [WBL 91, Led 92a]. Conformance testing is now widely accepted but the theory is incomplete here too, e.g. the generation of test sequences and the estimation of their coverage are difficult topics.

Most, if not all, FDTs do not totally support this design process. Some are more appropriate to expressing very abstract specifications rather than implementation-oriented ones, or the converse. Some have underlying models that provide a rigorous mathematical basis for the analysis or transformation of a specification. Others are limited to informal validations.

FDTs such as Estelle and LOTOS have proved their abilities to specify very complex protocols and services of the OSI RM [ISO 10023, ISO 10024]. However, they fail to provide concise and readable specifications, especially in the Estelle case. The second main phase of the design process is more critical for all the techniques. The applicability of LOTOS, for instance, has been investigated in the Pangloss/ESPRIT project [Bog 89, Mar 91] and in the Lotosphere/ESPRIT project [vKv 90, QAP 92].

For realistic specifications of respectable size, tools play an essential role in the design process to automate some activities or assist the designer in others. Prototype tools for the standardised languages Estelle (e.g. [CoS 92, Bud 92, ADG 93]), LOTOS (e.g. [GHL 88, MaM 89, QPF 89, Fer 89, MaV 90, Gar 90, GaS 90, vEi 92, EeW 93]) and SDL (e.g. [SSR 89, Atl 90, Hol 92]) are now available.

3 Specification Styles

In practice, it is necessary to produce specifications that are comprehensible and to specify their functionality efficiently. However, depending on the purpose of the specification and its level in the design process described in section 2, different specification styles may be used. In [VSv 91] four specification styles have been identified and described. The two basic criteria used for classifying these styles are structure and abstraction.

Along the *structure* axis, two description types are defined, viz. structured and unstructured:

- The *structured* description type is defined as a composition of basic subsystems influencing each other.
- The *unstructured* description type is defined as a whole without any decomposition in terms of subsystems influencing each other.

Note, in the above definitions, the importance of the expression "influencing each other". This means that the term "structure" has to be considered as a *spatial* structure instead of a *temporal* structure. A temporally structured behaviour description is for instance a *sequential* composition of several behaviours (or subsystems) B1, B2 and B3. In LOTOS, we write B1 >> B2 >> B3. In a temporal structure, the subsystems *cannot* influence each other (except by fixing an initial state). This temporal structure is important but not considered in this report. Instead we focus on the spatial structure. A spatially structured behaviour description is for instance the parallel composition of several behaviours (or subsystems) B1, B2 and B3. In LOTOS, we write for instance[2] B1 || B2 || B3. In a spatial structure, the subsystems can influence each other (except in the special case of interleaving: |||).

A (spatial) structure should not be confused with an implementation architecture even if a parallel operator is used. Structure is used as a way to present complex behaviours appropriately to allow a global understanding. In this respect, any complex system will require a structured description to be apprehended by a person. For instance, when in logic we write a specification of an object as the conjunction of several properties (e.g. property1 and property2 and property3), we use a structuring operator that is the logical "and" operator. However this does not introduce any architectural constraints in the design. In LOTOS, the parallel composition operator may be used as a kind of "and" operator without inducing an implementation architecture.

Let us summarise the presentation of the "structure" axis as follows. Structure is understood as spatial structure, and does not mean "implementation architecture". This last aspect is precisely one of the criteria leading to the other classification axis: the abstraction axis.

Along the *abstraction* axis, two description types may also be defined, viz. extensional and intentional types:

- The *extensional* description type is defined solely in terms of its external observable behaviour, viz. in terms of the interactions between the system and its environment.
- The *intensional* description type is defined with respect to internal (unobservable) state variables, or in terms of parts that interact internally (in an unobservable way).

Whereas the structure had a priori nothing to do with implementation details, the abstraction axis explicitly refers to them. A description will be considered extensional (i.e. abstract) if it does not contain internal details. By contrast, a description will be intensional (i.e. concrete) if it goes beyond the description of only the external behaviour as seen by an external user.

Despite the careful definition of the term "structure", the reader might still be convinced that structure and abstraction are not independent criteria because a structure inevitably influences an implementer and therefore goes against abstraction. We hope to further clarify the distinction between structure and abstraction after a detailed classification of the styles. Indeed, it will turn out that a specification style may be extensional and structured, or intensional and unstructured.

In LOTOS, these description types are easily related to the use of specific operators:

- In the extensional type, no "hide" operator is allowed. This means no internal interaction point are allowed.

---

[2] There are of course other parallel composition operators in LOTOS.

- In the unstructured type, no “parallel composition” operator is allowed.

Using these two orthogonal criteria, one naturally derives four specification styles:

- Intensional and unstructured (no parallel composition): the *state-oriented* style in which the system is seen as a single resource whose internal state space is explicitly defined. This style, therefore, presents only observable interactions - not spatially structured - and a representation of the global state space that these interactions manipulate.
- Intensional and structured: the *resource-oriented* style in which both observable and internal interactions are presented. The temporal ordering relationship of the observable interactions is defined by the parallel composition of separate resources that hides the internal interactions. Each resource is defined by a temporal ordering of external and internal interactions or of merely internal interactions.
- Extensional and unstructured (no hide, no parallel composition): the *monolithic* style in which only observable interactions are presented. Their temporal ordering relationship is defined as a collection of alternative sequences of interactions.
- Extensional and structured (no hide): the *constraint-oriented* style in which only observable interactions are presented, but their temporal ordering relationship is composed as the conjunction of separate constraints (like the conjunction operator in logic). Each constraint is defined on the subset of the interaction set that is relevant to it. An in-depth presentation of this style may be found in [Bri 89].

As defined, these styles are mutually exclusive. If the description is unstructured, there is a single process, which is state-oriented or monolithic. Note that in practice these two styles are often combined together, which leads to a model of an extended automaton. If the system description is structured (constraint-oriented or resource-oriented), it is possible to use a different style to specify each of the basic components of the system specification. For instance, each component of a resource-oriented specification may be specified in any of the four styles. This recursive use of the styles stops when all the basic processes are unstructured.

The usefulness of the styles will be further explained in the next section.

4 Method Guidelines

4.1 Use of Specification Styles in the Design Process

In the design process presented in section 2, the initial (abstract) LOTOS specification is successively transformed to reach a final (implementation-oriented) LOTOS specification that can be efficiently compiled. The four specification styles are very useful to support these transformations. Their characteristics make them suitable at specific stages of the design.

The starting point of the design is an abstract and structured specification. Abstraction is needed because this specification must be as tolerant as possible regarding possible implementations. This is particularly crucial for the specification of

a standard. Structure is needed because the object to be described is very often so complex that it cannot be globally apprehended by a person. These two elements lead us to an extensional and structured style, i.e. the constraint-oriented specification style, as the basic style for the initial LOTOS specification. The whole method explained in this report deals only with this initial abstract and structured LOTOS specification. There are three reasons for this.

First, this initial specification is the *reference* specification for all possible implementations. It is the only one that can be input to standardization bodies like ISO.

Second, the design of this abstract specification is far from being a trivial task. It needs the support of a method, which is today mostly learned by experience only.

Third, a methodology for covering the whole design trajectory (i.e. from the initial specification down to a implementation specification) needs much more resources, and was outside the scope of OSI95. Such an implementation methodology can be found in the Lotosphere / ESPRIT project [QAP 92].

The objective of our work may now be made more precise. It consists of finding for LOTOS a method for tackling the inherent difficulty of specifying large systems in an abstract way suitable for ISO. This goes beyond the selection of only a specification style, which would be the constraint-oriented style in this case. The method will explain the procedure to be followed by a LOTOS designer who wants to use the constraint-oriented style.

The main problem, which renders the specification process quite complex, originates from the fact that LOTOS does not fully support the constraint-oriented style. A solution to this problem would consist of enhancing LOTOS to allow it to support perfectly the constraint-oriented style. Some specific problems and known means to extend LOTOS are briefly described at the end of section 4.2. Even if this basic research would probably be the right way to go in the long term, it is beyond the scope of OSI95. Therefore, we will keep LOTOS and the constraint-oriented style “as is”, and focus our objective on a method that is intended to help LOTOS designers to circumvent the difficulties faced when using the constraint-oriented style on large case studies.

Going down towards implementation, abstraction becomes less important whereas implementation details become highly desirable: adding architectural details is the main purpose of the implementation process, which is the second phase of the implementation process presented in section 2.

In this activity, the first constraint-oriented specification will become progressively more concrete in various ways. For instance, in a software system, choices have to be made about modules, interfaces, data structures, algorithms, … Since the gap between a “high-level” specification and a “low-level” implementation is usually large, some “intermediate”, less and less abstract, specifications are produced. The first constraint-oriented specification is thus likely to be replaced by a resource-oriented specification whose basic components will be in turn specified in the constraint-oriented style or any unstructured style. The process may then continue downward in the structure, and transform the constraint-oriented subsystems into adequate resource-oriented ones. The process stops when all the basic modules are unstructured.

4.2 Guidelines for Using the Constraint-Oriented Style

The constraint-oriented style is intrinsically related to the parallel composition operator that allows the synchronisation of a certain (possibly infinite) number of processes at interaction points. These interaction points are necessarily external because no hide operator in allowed. A system is thus described as a collection of sub-processes interconnected together as illustrated in figure 4.1.

The first design decision is thus this decomposition in terms of subsystems. This problem is far from being simple and we will try to give in this section a general method to solve this difficult problem. Our method contains several phases denoted:

a) Identification of the main functions
b) Identification of the data structures updated or simply referred to by the functions
c) Grouping of functions according to data structures
d) Mapping of (groupings of) functions onto LOTOS processes (i.e. constraints)
e) Refinement step: For each LOTOS process defined at the end of step d), the whole process may be applied again, which leads to a recursive method.

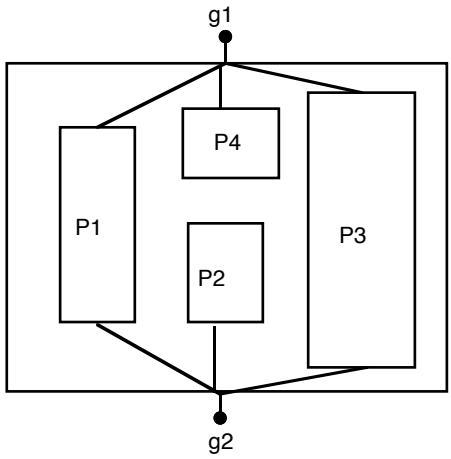

Fig. 4.1 An example of a constraint-oriented specification structure

a) Identification of The Main Functions

The first step consists of isolating the main functions of the specification. There is no need to define precisely all the elementary functions. Instead, it will be better to stay first at a high level of design of the specification. The precise substructure of these functions will be taken into account later, during step e (refinement step).

b) Identification of the Data Structures Updated or Simply Referred to by the Functions

The purpose of this step is to prepare the mapping of the functions onto LOTOS processes. This mapping has to respect some well-defined principles to circumvent some known deficiencies of the constraint-oriented style, as further explained at the end of section 4.2.

The main task consists of enumerating *all* the data structures used in the specification, and associating them with the functions. During this association process, we consider only the data structures that the function *updates* (i.e. read-write access). The data structures which the function simply *refers to* (i.e. read-only access) will be considered in the next step.

c) Grouping of Functions According to Data Structures

When a single function updates a data structure, it is easy to model the function as the manager of this data structure. By contrast, when several functions may update the same data structure, we have to be careful. Two possibilities exist:

a) We merge the functions into one process. This has the drawback of losing the separation of concern between processes.
b) We extract from each function the part dealing with the update and we isolate the management of the data structure in a new and distinct process. This process is such that it includes all the basic functionalities extracted from each function. This approach is not always feasible when the update of the data structure is deeply intertwined within the respective functions.

d) Mapping of (Groupings of) Functions onto LOTOS Processes (i.e. Constraints)

Now we consider the data structures that are *referred to* by the function (i.e. read-only access).

From the previous classification, we have groups of functions that will be associated with LOTOS processes. We still need to know whether some of these processes must have read access to the data structures localised in other processes. Again, to answer this question we have to take every process one by one and analyse its behaviour. If such read access is needed, we will have to provide a way to allow this access in the LOTOS specification. There are several possibilities:

a) We merge each process that needs read access with the process handling the data structure. Again, this has the drawback of losing the separation of concern between processes.
b) We add an attribute to all interactions occurring at a suitable interaction point common to both processes. This attribute will be the value of the data structure at the time of the interaction, and will be imposed by the process that knows it.

e) Refinement Step

After the first three steps, we have a constraint-oriented LOTOS specification whose structure reflects the separation of concerns between the main functions. For the functions that, at this stage, remain too complex to be specified in a single unstructured LOTOS process, the whole method may be applied again in a similar way. This leads naturally to a recursive method.

4.2.1 Known Deficiencies of LOTOS to Support the Constraint-Oriented Style

Ideally, every function should be mapped onto a distinct process (or constraint). This allows a clear separation of concerns between functions that can be explained independently from each other. However this ideal separation cannot always be carried out as far as we would like. For instance in LOTOS, even if the functions may be specified independently, they have sometimes to refer to some knowledge handled by another process. In other words, processes sometimes need to exchange knowledge about their context (or state). The problem then originates from the fact that there is no concept of shared variable in LOTOS. Therefore a process has to communicate with the others to get some information. The only basic hypothesis is that processes that refer to such a variable should have at least one common interaction point with the owner of the variable. If it is not the case, this interaction point has to be (artificially) added to the process to allow it to get information.

The following example will illustrate this. Suppose process B is modelling the constraints on the allowed numbering of successive TPDUs. B is responsible for the updates of a context variable such as the number of the last TPDU sent on a connection. The other processes have no way to access this variable. Thus if another process C needs this value to perform its function correctly (e.g. a retransmission of TPDUs up to the last number), there are two solutions: either C has observed all the TPDUs sent at a given interaction point and has registered permanently the highest number, or C has to exchange information with the first process B. The first solution implies that the updating function of the last TPDU number is performed in two processes; this is cumbersome and counter-intuitive. The exchange of information, which constitutes the second solution, is only possible at external interaction points since there are no internal ones in a constraint-oriented specification. Therefore, the environment of the system has to participate in these interactions; this is clearly undesirable.

Of course, since our method is based on the constraint-oriented style, it will suffer from these limitations. We will try however to minimize their impact. The ideal solution would be extend LOTOS according to the section 4.2.2 for instance.

4.2.2 Known LOTOS Extension to Solve this Problem

This problem could be overcome for the larger part if it was possible to hide partially observable actions to the external environment, i.e. to hide some attributes of the observable actions that only concern the communication among the subsystems but not the environment. In this case, by adding suitable attributes to the observable actions (e.g. the value of some variables that are of interest for several functions), we allow an additional exchange of knowledge between the subsystems. If a suitable construct may hide these attributes to the environment, then the goal is achieved.

A formal approach that would solve this problem has been proposed in [Bri 88b]. The underlying semantic model of LOTOS has however to be changed to incorporate the notion of a composite event. A composite event is an event composed of more elementary actions that have to occur all at the same instant (i.e. a composite event remains atomic) to say that the composite event occurs (at that instant). The idea then comes as a hiding operator that may hide some actions of a composite event. The way to map a LOTOS event into a composite event in the semantic model is quite straight-

forward: it suffices to split an event into as many actions as there are attributes. For instance, the LOTOS event *g?x:int?n:nat* would be mapped onto the composite event $<g_1?x:int, g_2?n:nat>$ where g_1 and g_2 are the elementary interaction points in the semantic model. The internal event *i* would be simply mapped onto the empty event <>. Then, it is possible to hide only one of the g_i's; in this case this consists in removing the g_i component from all the composite actions. Removing all attributes leads of course to classical hiding, and to the empty event, which is the model of *i*.

Another problem that this approach solves is the possibility to specify the successful termination of a process when a specific event occurs. In LOTOS, the *exit* construct, which is used to model termination, has some known shortcomings. For instance, consider the process that is specified as *(P [> Q) >> R*, if *P* terminates successfully, then *Q* is disabled. However, suppose *P* is defined as *P := a; b; exit*, then we could think that what we have specified is that *P* has terminated successfully when *b* has occurred. This is wrong, after *b* has occurred, the global process is described by *(exit [> Q) >> R* (which is equivalent to *i; R [] Q*), and *Q* may still occur. This is only the internal event modelling the relay between *exit* and *R* that disables *Q*. In other words, it is impossible to model that *Q* is disabled as soon as event *b* has occurred; which is unfortunate. The composite event solves this problem easily. It suffices to replace the *b* action (which is the composite event <*b*>) by a composite event $<b, \delta>$ where δ is the well-known termination action in the semantic model of LOTOS. Any composite event containing δ as a component becomes a termination event.

5 Application of the Method to ISO 8073 (Class 4)

In this section we illustrate the method presented in section 4 on a large case study, viz. a transport entity as specified in ISO 8073 class 4, referred to as TP4 in the sequel. According to ISO 8073, if class 4 is supported by a transport entity, then it is required that class 2 be supported too. In CCITT X224 describing the same transport protocol, this static conformance requirement is different: this is class 0 which is required. Nevertheless, in our specification we consider that our transport entity only supports class 4.

5.1 The Model of a TP4 Entity

According to the Reference Model [ISO 7498], a transport protocol entity interacts with transport service users by means of transport service primitives exchanged at some local TSAPs, and with the network service provider by means of network service primitives at some local NSAPs.

In LOTOS, this architecture will be modelled as a system with two external interaction points, viz. t and n, modelling respectively the (set of all local) TSAPs and the (set of all local) NSAPs. TP4 [ISO 8073] is connection-oriented at both the transport and network service interfaces. Therefore each SAP is in turn composed of CEPs. Primitives exchanged at service interfaces will then have the following structure:

t ?ta:TAddress ?tcei:TCEI ?tsp:TSP

n ?na:NAddress ?ncei:NCEI ?nsp:NSP
where ta and tcei (resp. na and ncei) identify the TSAP and TCEP (resp. NSAP and NCEP) where the transport (resp. network) service primitive occurs, and tsp (resp. nsp) is a primitive proper.

5.2 Application of the Method

Step a: Functions Performed by a TP4 Entity

In class 4, all basic functions are supported except the error release - which is used only in Classes 0 and 2 to release a transport connection on the receipt of an N-DISCONNECT or N-RESET indication - and all TPDUs are supported except the RJ (Reject) TPDU.

We list hereafter all the functions supported by a TP4 entity. The function names are those provided in the standard, and the numbers between parentheses refer to the clauses of the standard where the functions are described.

- Assignment to network connection (§ 6.1)
  - Initial assignment of a transport connection to a (set of) network connection(s)
  - Creation of a network connection
  - New assignment in relation to splitting
- TPDU transfer (§ 6.2)
  - TPDU -> NSP
  - NSP -> TPDU
- Segmenting and reassembling (§ 6.3)
  - Segmenting (TSDU -> set of TPDUs)
  - Reassembling (set of TPDUs -> TSDU)
- Concatenation and separation (§ 6.4)
  - Concatenation (set of TPDUs -> NSDU)
  - Separation (NSDU -> set of TPDUs)
- Connection establishment (§ 6.5, § 12.2.2.2.b)
  - Initiator function
  - Responder function
  - Negotiation (classes, TPDU size, format, checksum, QoS parameters, expedited data)

  (other functions strongly related to the connection establishment are dealt with separately, e.g. assignment to network connections, transport connections reference allocation, mapping of transport addresses onto network addresses)
- Connection refusal (§ 6.6)
- Normal release (§ 6.7)
  - Explicit variant normal release only (§ 6.7.5)
  - Release of a network connection
- Association of TPDUs with transport connections (§ 6.9)
  - Identification of TPDUs (§ 6.9.4.1)
  - Association of individual TPDUs (§ 6.9.4.2)
- Data TPDU numbering (§ 6.10)

- Expedited data transfer (Network normal data variant only) (§ 6.11, § 12.2.3.4)
- Reassignment after failure (this function is provided using procedures other than those used in § 6.12, refer to § 12.2.2.2.a)
- Retention until acknowledgement of TPDUs (§ 6.13)
- Resynchronization (this function is provided using procedures other than those used in § 6.14, refer to table 23, line NRSTind)
- Multiplexing and demultiplexing (§ 6.15)
  - Multiplexing
  - Demultiplexing
- Explicit flow control (§ 6.16, § 12.2.3.6, § 12.2.3.9)
  - Explicit flow control proper (sending part) (§ 12.2.3.9)
  - Explicit flow control proper (receiving part) (§ 12.2.3.9)
  - Use of flow control confirmation parameter (§ 12.2.3.9)
- Checksum (§ 6.17)
  - Compute checksum
  - Verify checksum
- Frozen references (§ 6.18, § 12.2.1.1.6)
- Retransmission on time-out (§ 6.19, § 12.2.1.2.i, § 12.2.1.1.4, § 12.2.1.1.1, § 12.2.3.1.2)
- Resequencing (§ 6.20, § 12.2.3.5)
- Inactivity control (§ 6.21, § 12.2.3.3, § 12.2.3.1.1)
- Treatment of protocol errors (§ 6.22)
- Splitting and recombining (§ 6.23)
  - Splitting
  - Recombining
- Sequencing of received AK TPDUs (§ 12.2.3.7)
- Procedures for transmission of AK TPDUs (§ 12.2.3.8, § 12.2.1.1.3, § 12.2.3.1.2)
  - Transmission of AK TPDUs (§ 12.2.3.8.1, § 12.2.1.1.3, § 12.2.3.1.2)
  - Sequence control for transmission of AK TPDUs (§ 12.2.3.8.2)
  - (Optional) retransmission of AK TPDUs after CDT set to zero (§ 12.2.3.8.3, § 12.2.3.1.2)
  - (Optional) retransmission procedures following reduction of the upper window edge (§ 12.2.3.8.4, § 12.2.3.1.2)
- Encoding of TPDUs (§ 13)
  - Encoding of CR TPDU (§ 13.3)
  - Encoding of CC TPDU (§ 13.4)
  - Encoding of DR TPDU (§ 13.5)
  - Encoding of DC TPDU (§ 13.6)
  - Encoding of DT TPDU (§ 13.7)
  - Encoding of ED TPDU (§ 13.8)
  - Encoding of AK TPDU (§ 13.9)
  - Encoding of EA TPDU (§ 13.10)
  - Encoding of ER TPDU (§ 13.12)
- Mapping of transport addresses onto network addresses (§ 5.3.1.2 e)
- Transport connection identification (§ 5.3.1.3.e + local mapping on TCEP)

- Network connection identification (local mapping on NCEP: implicit in the standard)
- Options supported by the transport entity (implicit in the standard)
- Provision of transport connections (implicit in the standard)
- Provision of network connections (implicit in the standard)
- Transport connection reference allocation and release (implicit in the standard)
- Backpressure at the transport service interface (missing in the standard)
  Backpressure means that the transport entity may decide not to accept a T-DATA.request for some local reasons. This function is necessary for reasons of conformance with the transport service definition where this phenomenon may appear. In the transport entity, application of this backpressure is of course a local matter (e.g. an implementation may use backpressure when it would overflow its local memory resources, due to flow control or retention at the protocol level).

These functions may be classified as follows:
1. Functions whose scope is a transport connection, i.e. functions which are associated with a transport connection and are completely independent from any other existing transport connection or network connection. There will be as many instances of these functions as there are transport connections handled by the entity.

- Segmenting and reassembling
- Connection establishment
- Connection refusal
- (Explicit variant) normal release
- Data TPDU numbering
- Expedited data transfer
- Retention until acknowledgement of TPDUs
- Explicit flow control
- Checksum
- Retransmission on time-out
- Resequencing
- Treatment of protocol errors
- Sequencing of received AK TPDUs
- Procedures for transmission of AK TPDUs
- Encoding of TPDUs
- Transport connection identification
- Backpressure at the transport service interface

2. Functions whose scope is a network connection, i.e. functions which are associated with a network connection and are completely independent from any other existing network connection or transport connection. There will be as many instances of these functions as there are network connections handled by the entity.

- TPDU transfer
- Concatenation and separation
- Normal release (Part: Release of a network connection)

- Association of TPDUs with transport connections (Part: Identification of TPDUs)
- Resynchronization
- Network connection identification

3. Functions whose scope is the overall transport entity, i.e. functions which cannot be classified in the first two categories either because they are associated with a relation between transport connections and network connections, or because they refer to sets of transport or network connections. There will be one instance of these functions in the transport entity.

- Assignment to network connection
- Association of TPDUs with transport connections (Part: Association of individual TPDUs)
- Reassignment after failure
- Multiplexing and demultiplexing
- Frozen references
- Inactivity control
- Splitting and recombining
- Mapping of transport addresses onto network addresses
- Options supported by the transport entity
- Provision of transport connections
- Provision of network connections
- Transport connection reference allocation and release

In addition, the relation between the transport connections and the network connections is a *many-to-many* relation, due to multiplexing and splitting.

The more general structure we can therefore think of is the one presented in figure 5.1. The process declaration is as follows:

```
process TPEntity [t,n] (tas:Taddresses, nas:Naddresses,
                        tpeo: TPEOptions) :noexit :=
      (TCs [t,n] (…)  |[n]|  NCs [n] (…))
      |[t,n]|
      Entity_functions [t,n] (…)
endproc
```

where tas (resp. nas) is the set of transport (resp. network) addresses supported by this transport entity, and tpeo is the set of options supported by the entity (e.g. supported classes, roles, optional procedures, maximal TPDU size supported for each supported class).

In this structure the data flow between t and n is handled by process TCs, and split between the instances of TC. Process Entity_functions is not used to transfer information between t and n; its purpose is to add constraints which can only be expressed at the entity level.

This structure is however unable to take account of the concatenation function. Indeed, the many-to-many relation between transport and network connections becomes even more complex due to segmentation and concatenation functions, and especially their intertwining with multiplexing and splitting functions. The essence of

the problem is the presence of *concatenation and separation* functions in the transport entity specification, which clearly enforces the identification of a third important data structure (i.e. the TPDU) in addition to the unavoidable TSDU and NSDU. In order to understand this point, it is important to know that a TSDU may be segmented into several TPDUs (possibly sent on *different* network connections), and several TPDUs (possibly from *different* transport connections) may be concatenated into a single NSDU. Therefore, a process TC alone cannot directly deal with a NSDU which is potentially associated with several TCs. TC cannot go further than dealing with a TPDU. Instead, NC is the only place where NSDUs may be dealt with.

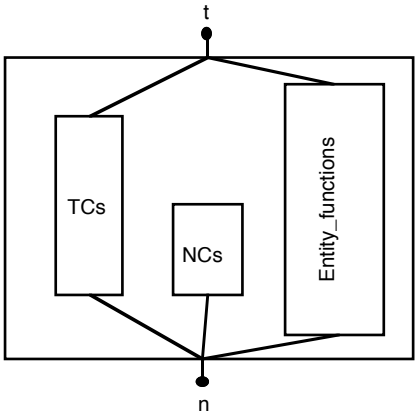

Fig. 5.1 Naïve structure of a TPEntity

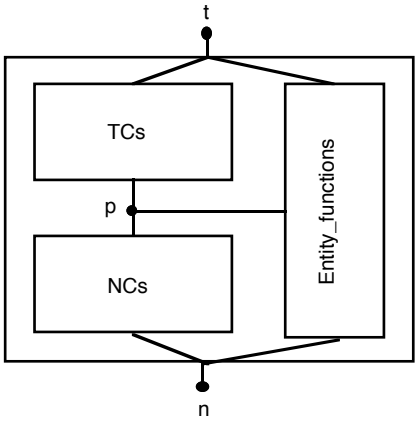

Fig. 5.2 TPEntity

Therefore, if we want to preserve the separation of concerns between functions related to transport and network connections, the only solution is to introduce an internal interaction point between TCs and NCs. This interaction point will be used to transfer the TPDUs between TCs and NCs, so that NCs are able to concatenate TPDUs or separate NSDUs when necessary. This internal interaction point will be denoted p.

We have also to add the interaction point p to process Entity_functions in order to remain as general as possible, and to allow all three processes to synchronize on p.

The more general structure we can therefore think of is the one presented in figure 5.2. The process declaration is as follows:

```
process TPEntity [t,n] (tas:Taddresses, nas:Naddresses,
                        tpeo: TPEOptions) :noexit :=
  hide p in
       (TCs [t,p] (…)    |[p]|   NCs [p,n] (…))
       ||
       Entity_functions [t,p,n] (…)
endproc
```

It is likely that some parameters of TPEntity will not be referred to by some subprocesses, e.g. tas by NCs. To decide on that, it is necessary to check which function refers to which parameter.

Step b: Identification of Data Structures

Instead of doing so, we propose to directly generalize this process as explained in point b of the method (section 4.2). We first enumerate *all* the data structures (not only the TPEntity parameters) used by a transport entity, and associate them with the functions. During this association, a distinction will be made between the data structures simply *referred to* by the function (i.e. read-only access) and those *updated* by the function (i.e. read-write access).

The reason why we propose this generalization is that we already foresee that these data structures will be needed when going downwards in the refinement of the transport entity structure (i.e. when TCs, NCs and Entity_functions will have to be described).

From our understanding of the informal text, we have been able to extract the following data structures of a TP4 entity. For each of them, we have also listed the TP4 functions that are responsible for updating them (Table 5.1).

Many of these data structures are explicit in the standard and are therefore easy to find. We realized however that some other data structures (e.g. Local Transport Connection References) are needed to write a complete specification in LOTOS. We did not follow some specific guidelines to find these data structures. Past experience of protocol specification is probably the central criterion here.

Among these data structures we find some "timers" (e.g. inactivity timer). We use this term in the following sense: the time *variable* whose value keeps track of the passing of time. This is clearly a data structure. By contrast, the function of updating this value is done by a timer *process*(e.g. inactivity control).

Step c: Grouping of Functions or Isolation of Data Structure Handling

When several functions may update the same data structure, some problems arise. We presented the two ways of tackling them in section 4.2, together with their advantages and shortcomings:

i) We merge the functions together into one process.
ii) We extract from each function the part dealing with the update and we isolate the management of the data structure in a new and distinct process. This solution is likely to require that the remaining function cores will still need to know the value of the isolated data structure. This read-access problem will be dealt with in part d of the method where it best fits. It will imply the presence of extra attributes to the interaction points common to the processes concerned. Each new attribute will convey the value of a data structure.

Table 5.1 List of data structures and corresponding functions that update them

Data structure	Updated by (function)
At the entity level	
Local TSAPs (free and in use)	Provision of transport connections
Local NSAPs (free and in use)	Provision of network connections
Local Transport Connection references (Free, in use, frozen)	Transp. connect. ref. alloc. and release, frozen ref.

Supported options	(Static)
Mapping table: Remote TSAPs -> Remote NSAPs	(Static)
Many-to-many relation (TCs <-> NCs)	Assign. netw. connect., reassign. after failure, multiplexing, splitting
Inactivity timer	Inactivity control
Reference timer	Frozen ref.
At the transport connection level	
My Transport Connection Id	Transport connection identification
Negotiated TPDU size	Segmentation [3]
Negotiated format	Data TPDU numbering, Expedited data transfer, Transmission of AK TPDUs [4]
Negotiated use of checksum	Checksum [4]
Queue of Incoming Transport Service Primitives to be (segmented and) sent as TPDUs	Segmentation
Queue of retained TPDUs after sending	Retention until acknowledgement of TPDUs
Queue of Incoming TPDUs to be (reassembled and) delivered as Transp. Service Primitives	Reassembling, resequencing
Role (initiator, responder)	Connection establishment
Credit of last AK received in sequence	Explicit flow control (sending part)
Credit of last AK sent	Explicit flow control (receiving part)
Highest number of DT sent	Data TPDU numbering
Highest number of DT received in sequence	Resequencing
Highest number of ED sent	Expedited data transfer
Highest number of ED received	Expedited data transfer
Highest number of AK received in sequence	Sequencing of received AK TPDUs
Highest number of AK sent	Transmission of AK TPDUs
Highest number of EA received	Expedited data transfer
Highest number of EA sent	Expedited data transfer
Retransmission timer	Retransmission on time-out
Window timer	Transmission of AK TPDUs
Maximum number of transmissions	Retransmission on time-out
At the network connection level	
My Network Connection Id	Network connection identification
Queue of TPDUs to be (concatenated and) transmitted as Network Service Primitives	Concatenation, TPDU transfer (TPDU -> NSP)
Queue of Network Service Primitives to be (separated and) delivered as TPDUs	Separation, TPDU transfer (NSP -> TPDU)

A third solution is a variant of (i). Instead of merging the functions, it would consist of adding a new internal interaction points in each group to allow some synchronisation between the functions that update the same data structure. This is what has been done at the first design stage (Figure 5.2). However, we want to use this solution as a last resort only to stay as close as possible to a pure constraint-oriented style.

Some basis for choosing between solutions (i) and (ii) above may be the nature of the data structure which is common to the respective functions. Let us suppose that the data structure is specified as a data type and not as a LOTOS process. When sev-

[3] It may seem counter-intuitive to consider that the data structure is updated by these processes. This is indeed a particular case: being updated only once, at connection establishment, it is more natural to associate the data structure directly with the processes which will refer to it later on.

eral functions refer to a complex data structure (i.e. data type) like a queue, we recommend solution (i) because solution (ii) would lead to the addition of a new attribute for conveying the state of a complex data type (viz. the queue) repeatedly. In our opinion, this way of doing would make the specification less understandable. For example, the functions "reassembling" and "resequencing", which share the same queue of incoming TPDUs, are better to be merged.

Another reason for merging functions according to solution (i) is that, after having looked carefully at these functions, they do not appear any more as separate concerns. For example, the two parts of the function "TPDU transfer" can simply be merged respectively with the functions "concatenation" and "separation".

Let us review the listed data structures and their localization within functions according the general idea explained above.

According to the previous discussion we recommend that the functions grouped together hereafter be merged into a single process:

- Reassembling + resequencing
- Assignment to a NC + Reassignment after failure + Multiplexing + Splitting
- Frozen references + Transport Connection references allocation and release
- Concatenation + TPDU transfer (emission)
- Separation + TPDU transfer (reception)

The "negotiated format" data structure does not necessitate a merge between "Data TPDU numbering", "Expedited data transfer" and "Transmission of AK TPDUs" because these values are only "updated" once (viz. during the connection establishment).

Step d: Mapping of (Groupings of) Functions onto LOTOS Processes

Now that this classification has been achieved, we have groups of functions which will be associated with LOTOS processes. We still need to know whether some of these processes must have a read-access to the data structures localized in other processes. Again, to answer this question, we have to take every process one by one, and analyse its behaviour. If such read-access is needed, we will have to provide a means to allow this access in the LOTOS specification. There are several possibilities:

a) We merge each process which needs read-access with the process handling the data structure. Again, this has the drawback of losing the separation of concern between processes.

b) We add an attribute to all interactions occurring at a suitable interaction point common to both processes. This attribute will be the value of the data structure at the time of the interaction, and will be imposed of course by the process which knows it. There are two sub-cases.

b1) This common interaction point is external, i.e. t or n. In this case, the attribute is visible at the system interface and, since the external environment has to participate in these interactions, the specification of the environment itself has to include these attributes in each of its offers. This is unacceptable because the environment has to be designed independently: changing the structure of interactions at n (resp. at t) would imply a redesign of the network (resp. transport) service specification.

b2) This common interaction point is internal, i.e. p. In this case, this seems the perfect solution since the external environment ignores everything.

Therefore, if p is a common interaction point of both processes, solution b2) will be selected. Otherwise, we have to apply solution a) because b1) is not applicable without a partial hiding operator.

Let us review the listed data structures and the processes which need a read-access to them (Table 5.2).

Table 5.2 List of data structures and functions that refer to them

Data structures	Referred to by (function)
Local TSAPs (free and in use)	-
Local NSAPs (free and in use)	-
Local Transport Connection references (Free, in use, frozen)	-
Supported options	-
Mapping table: Remote TSAPs -> Remote NSAPs	-
Many-to-many relation (TCs <-> NCs)	Association of individual TPDUs
Inactivity timer	-
Reference timer	-
My Transport Connection Id	-
Negotiated TPDU size	-
Negotiated format	-
Negotiated use of checksum	-
Queue of Incoming Transport Service Primitives to be (segmented and) sent as TPDUs	-
Queue of retained TPDUs after sending	Retransmission on time-out
Queue of Incoming TPDUs to be (reassembled and) delivered as Transport Service Primitives	-
Role (initiator, responder)	-
Credit of last AK received in sequence	Retention until acknowledgement
Credit of last AK sent	Resequencing, Transmission of AK TPDUs
Highest number of DT sent	-
Highest number of DT received in sequence	Transmission of AK TPDUs
Highest number of ED sent	-
Highest number of ED received	-
Highest number of AK received in sequence	Explicit flow control (sending part, use of conf. param.)
Highest number of AK sent	Explicit flow control (receiving part), Resequencing
Highest number of EA received	-
Highest number of EA sent	-
Retransmission timer	-
Window timer	-
Maximum number of transmissions	-
My Network Connection Id	-
Queue of TPDUs to be (concatenated and) emitted as Network Service Primitives	-
Queue of Network Service Primitives to be (separated and) delivered as TPDUs	-
Role (owner, non-owner)	-

By applying some criteria explained in phase (c) of the method, we recommend that the functions grouped together hereafter be merged into a single process:

- Association of individual TPDUs merged with
  (Assignment to a NC + Reassignment after failure + Multiplexing + Splitting)
- Retransmission on time-out merged with Retention until acknowledgement

For other functions, we recommend not to merge and therefore to add the following attributes to all interactions at p:

- Highest number of AK sent (i.e. the lower bound of the receiving window)
- Credit of last AK sent
- Highest number of AK received in sequence (i.e. the lower bound of the transmit window)
- Highest number of AK received in sequence
- Highest number of DT received in sequence

The "Highest number of AK sent" together with the "Credit of last AK sent" define the Receiving window. The "Highest number of AK received in sequence" together with the "Highest number of AK received in sequence" define the Transmit window.

Figures 5.3, 5.4 and 5.5 illustrate a TC process, a NC process and the Entity_functions process according to the above study.

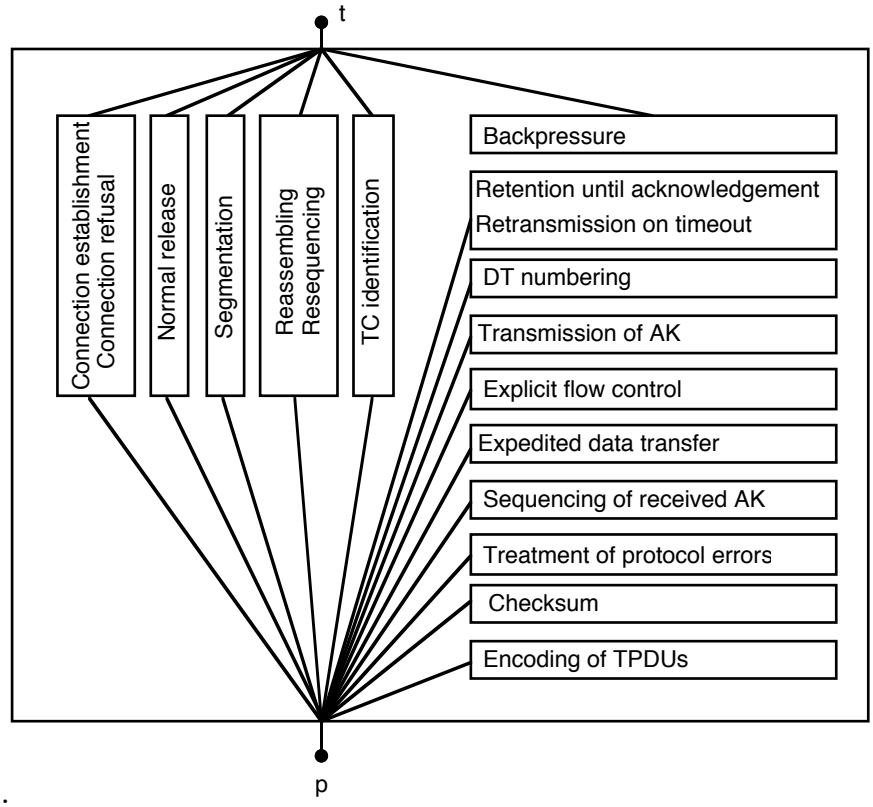

Fig. 5.3 A TC process

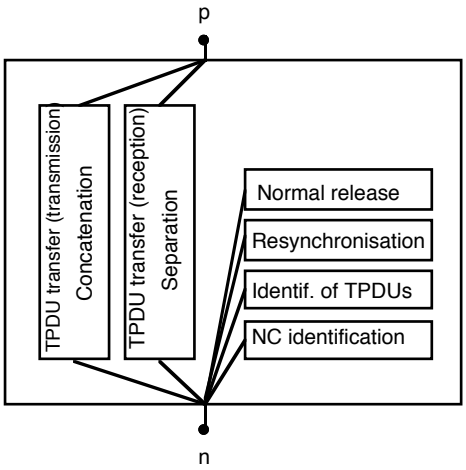

Fig. 5.4 An NC process

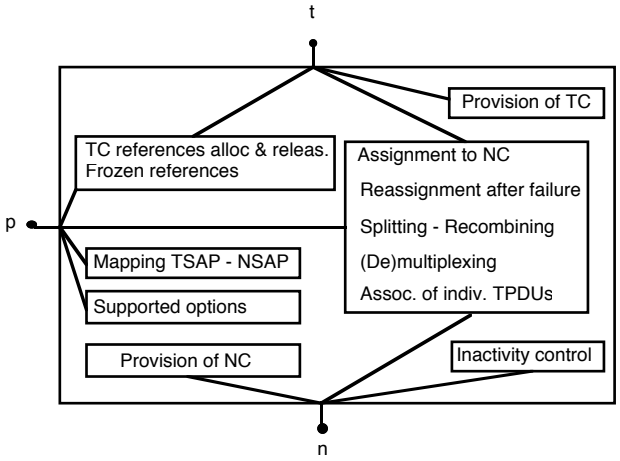

Fig. 5.5 The Entity_functions process

There remains an important step in the method. We have introduced an internal interaction point p where all the TPDUs are passing. The TPDU is the main attribute of actions occurring at p, but we have seen that additional attributes are necessary to allow an exchange of knowledge between processes attached at p.

We will see now that it is useful to go further and add another set of attributes to the p actions in order to avoid the duplication of functions in several processes.

Many functions (processes) refer explicitly to the internal structure of the TPDU (i.e. the values of the various fields of the TPDU) in order to behave correctly. Usually, a process does not need to know the whole TPDU but only a useful abstraction of it, e.g. if it is correct, in sequence, received in the window or a duplicate. However, two problems may occur regarding the computing of such abstractions:

- The processes are unable to compute these abstractions because they have only a partial knowledge of the context of the transport entity. They know only the data structures managed by them, and the values of the attributes coming with the TPDU. This information may well be insufficient to perform the appropriate function. This may be solved if suitable abstractions are computed by the processes which have the information to do so, and then add new attributes to pass their value to all other processes.
- When the same abstractions of a TPDU are needed by several processes which have enough information to compute them, we have to avoid that they be computed in all of them. This may be avoided if only one performs this computation and passes the value in a new attribute.

It is rather easy to list the useful abstractions of a TPDU as done hereafter. Note that some of them are only applicable to a subset of the TPDUs (e.g. DT (Data), AK (Acknowledgement of DT), ED (Expedited Data), EA (Acknowledgement of ED)):

- its direction of flow (either up or down)
- if it is a duplicate or not
- if it is correct or in error (this boolean value may be generalized and replaced by a set containing all the errors associated with a passing TPDU, the set being empty if the TPDU is correct)

- if it is (received) in sequence
- if its number is in the receiving window
- if it is kept or not on reception

These attributes have to be computed by only one process. Let us review (Table 5.3) which process may compute which attribute, and then select the appropriate one. If some attributes cannot be computed by any process, we have to identify which information is missing in which process, and then add it as a new attribute.

Table 5.3 List of attributes at "p" and functions that compute them

Attribute	May be computed by (function)
Direction	Segmentation (for sending non-duplicated CR (Connect Request), DT, ED, DR (Disconnect Request))
	Retransmission on time-out (for sending duplicated CR, CC (Acknowledgement of CR), DT, ED, DR)
	Connection establishment + refusal (for sending CR, CC, DR)
	Normal release (for sending DR, DC (Acknowledgement of DR))
	Transmission of AK (for sending AK)
	Expedited data transfer (for sending EA)
	TPDU transfer (reception) + separation (for receiving any TPDU)
Duplication at emission	Segmentation (for non-duplicated CR, DT, ED, DR)
	Connection establ. + refusal (for non-duplicated CR, CC, DR)
	Normal release (for non duplicated DR, DC)
	Retransm. on time-out (for duplicated CR, CC, DT, ED, DR)
	Transmission of AK (for AK) (but no duplication is possible)
	Expedited data transfer (for EA)
Duplication at reception	Connection establ. + refusal (for CR, CC, DR)
	Connection release (for DR)
	Resequencing (for DT)
	Sequencing of received AK TPDUs (for AK)
	Expedited data transfer (for ED, EA)
Errors at reception	Encoding of TPDUs (for encoding errors)
	Checksum (for bad checksums)
	Treatment of protocol errors
Reception in sequence	Resequencing (for DT)
	Expedited data transfer (for ED)
	Sequencing of AK TPDUs (for AK)
	Expedited data transfer (for EA)
In the receive window	Explicit flow control (for DT)
Kept or not	Resequencing (for DT)

Taking account of these new attributes at gate "p", there may be some redundancy with previously selected attributes. This redundancy has to be carefully analysed and removed. In this example, the attribute "Credit sent in the last AK" may be removed for the following reason: it was referred to by "Resequencing" and "Transmission of AK TPDUs" to check whether a received DT TPDU was in the receive window. Now that the special attribute "In the Receive window" exists, which directly informs any process about this fact, the credit becomes useless.

Step e: Refinement of Processes

In this section the term "refinement" has the following meaning which may slightly differ from its usual sense. Consider the function "Expedited Data Transfer" which has been specified as a single LOTOS process up to now. When looking closer at this function in the informal text, one comes to the conclusion that this function can be split into several sub-functions such as "Numbering of ED", "Transmission of EA", "Reception of EA" and "DT blocking". This decomposition can be done for other functions in a similar manner, e.g. the process "Explicit Flow Control" will be split for instance into "Explicit Flow Control on transmission", "Explicit Flow Control on reception", and "use of flow control confirmation parameter".

Thus decomposition into sub-functions is what we call refinement in this section. During this refinement process, the constraint-oriented style remains the sole style used; that is, a function is specified as a collection of processes (sub-functions) combined with a parallel operator, without introducing additional (internal) interaction points. Therefore, the sub-processes are synchronized via the existing interaction points, and each of them introduces only the constraints associated with its sub-function. The constraints resulting from this composition are simply the union of the constraints of each sub-process.

This refinement may create some difficulties which may lead to the addition of some attributes to the existing interaction points. The reason is that the whole method explained up to now has to be reapplied to each process. That means that if sub-processes have to access some data structures whose scope is restricted to another sub-process, they have to exchange knowledge via suitable attributes in their interactions. Since we have the requirement that no additional interaction point is allowed, the attributes should be added to existing interaction points.

Let us consider the refinement of "Expedited data transfer" and apply our method. In order to do that, reconsider all the data structures whose updates are the responsibility of this process (Table 5.4).

Table 5.4 List of attributes that are computed by the function "expedited data transfer"

Attribute	May be updated by (function)
Negotiated format	Expedited data transfer
Highest number of ED sent	Expedited data transfer
Highest number of ED received	Expedited data transfer
Highest number of EA received	Expedited data transfer
Highest number of EA sent	Expedited data transfer

Suppose that this process is refined into the four processes "Numbering of ED", "Transmission of EA", "Reception of EA" and "Suspension of DT flow". We have to assign each data structure to one of these sub-processes: the one which updates it (Table 5.5).

Then we analyse whether the other sub-processes need read access to values of data structures not updated by themselves (Table 5.6).

Table 5.5 Refinement of the function "expedited data transfer" and new updates of attributes

Attribute	May be updated by (function)
Negotiated format	ED numbering, EA transmission
Highest number of ED sent	ED numbering
Highest number of ED received	EA transmission
Highest number of EA received	EA reception
Highest number of EA sent	EA transmission

Table 5.6 Refinement of the function "expedited data transfer" and new needs for references

Attribute	May be referred to by (function)
Highest number of ED sent	Suspension of DT flow
Highest number of ED received	-
Highest number of EA received	Suspension of DT flow
Highest number of EA sent	-

We see that "Suspension of DT flow" needs to know the values of "Highest number of ED sent" and "Highest number of EA received". Therefore, these values have to be added to all interactions at gate p (which is the sole gate of this process) in order to allow the needed exchange of knowledge between sub-processes.

The shortcoming of this approach is that all the event offers of other functions have to be changed to include these new attributes as well. This problem was already encountered during the first decomposition and is due to the lack of a partial hiding operator in LOTOS.

As a characterization of our design we propose the final set of attributes at gate "p" and their meaning.

`p ? tr: TPid`	The transport connection reference given by the protocol
`? ni: NId`	The network connection identification
`? d: Dir`	The direction in which the ETPDU parameter passes (either *Send* or *Receive*)
`? c: Copy`	The notification of duplication (either *New* or *Dupl*)
`? err: TPerr`	The set of protocol errors detected on the passing ETPDU parameter
`? IsInRWindow: Bool`	The notification of presence in the receive window
`? rw: TPDUWindow`	The receive window, i.e. the pair (highest number of AK sent, highest number of AK sent + credit sent in the last AK)
`? sw: TPDUWindow`	The sending window, i.e. the pair (highest number of AK received in sequence, highest number of AK received in sequence + credit received in the last AK in sequence)
`? kon: KeptOrNot`	The notification of storage or deletion of the passing ETPDU parameter
`? IsInSeq: Bool`	The notification of being in sequence with previous ETPDU parameters
`? MaxDTRecInSeq: TPDUNumber`	The highest number of DT TPDU received in sequence
`? etpdu: ETPDU`	The passing TPDU in its encoded form

The outline structure of the resulting specification is depicted on figures 5.2, 5.3, 5.4 and 5.5.

6 Conclusion

In this report, a method for applying the constraint-oriented style to large LOTOS specifications has been presented and illustrated on a case study of relatively high complexity, viz. the ISO transport protocol class 4. Our approach allows us to circumvent some limitations of the constraint-oriented style in LOTOS but not all of them. For instance, the LOTOS specifications of all the main modules of TP4 are constraint-oriented, except the topmost module which is resource-oriented. Nevertheless, the specification has a cleaner structure than the official LOTOS specification of IS 8073 [ISO 10024, KLR 93] which uses much more the resource-oriented style, and thereby introduces several additional internal interaction points and specific associated synchronisations in processes. To go further, LOTOS, despite its powerful parallel composition operator, would need some improvements to support the constraint-oriented style better. The operators which ought to be more flexible are mainly the parallel composition and hiding operators. Nevertheless, we think that when LOTOS is applied according to the proposed method, it offers enough flexibility to specify objects as complex as a transport protocol entity or service. This has been further examined in the OSI95 project on the TPX protocol [BLL 92].

A substantial part of ISO 8073 class 4 has been specified according the method presented in the previous section. The commented LOTOS specification size is 5100 lines (2700 lines of abstract data types and 2400 lines of processes). However, our specification is far from being complete. In particular, process NC has been left unspecified, as well as a large part of the process Entity_functions. The main work has focused on process TC, which includes the main transport functions that are almost independent from the nature of the underlying network service. This choice is motivated by the fact that this method is intended to be applied in a second step to specify TPX which is a transport protocol relying on a connectionless-mode network service provider. In such a case, processes NCs and Entity_functions are much simpler.

References

[ADG 93] B. Algayres, L. Doldi, H. Garavel, Y. Lejeune, C. Rodriguez, **VESAR: a Pragmatic Approach to Formal Specification and Verification,** in: *Computer Networks & ISDN Systems 25 (7)* (1993).

[Atl 90] M. Atlevi, **SDT - A Real Time CASE Tool for the CCITT Specification Language SDL,** in: S. T. Vuong, ed., *Formal Description Techniques II* (North-Holland, Amst. 1990) 37-41.

[BHR 84] S.D. Brookes, C.A.R.Hoare, A.W.Roscoe, **A Theory of Communicating Sequential Processes,** *Journ. ACM*, Vol. 31, No. 3, July 1984, 560-599.

[BLL 92] Y. Baguette, L. Léonard, G. Leduc, A. Danthine, O. Bonaventure, **OSI95 Enhanced Transport Facilities and Functions,** OSI95/Deliverable ULg-A/P, 11-12-1992, pp. 277.

[Bog 89] K. Bogaards, **LOTOS Supported System Development,** in: K.J. Turner, ed., *Formal Description Techniques* (North-Holland, Amsterdam, 1989) 279-294.

[Bri 88a] E. Brinksma, **A Theory for the Derivation of Tests,** in: S. Aggarwal, K. Sabnani, eds., *Protocol Specification, Testing and Verification, VIII* (North-Holland, 1988) 63-74.

[Bri 88b] E. Brinksma, **On the Design of Extended LOTOS, a Specification Language for Open Distributed Systems,** Doctoral Dissertation, Twente University of Technology, Department of Informatics, Enschede, The Netherlands, 1988.

[Bri 89] E. Brinksma, **Constraint-oriented Specification in a Constructive Formal Description Technique,** in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg, eds., *Stepwise Refinement of Distributed Systems - Models, Formalisms, Correctness*, LNCS 430 (Springer-Verlag Berlin Heidelberg New York, 1989), 130-152.

[BSS 87] E. Brinksma, G. Scollo, C. Steenbergen, **Process Specifications, their Implementations and their Tests,** in: G.v. Bochmann, B. Sarikaya, eds., *Protocol Specification, Testing and Verification, VI* (North-Holland, Amsterdam, 1987) 349-360.

[Bud 92] S. Budkowski, **Estelle Development Toolset (EDT),** in: *Computer Networks & ISDN Systems 25* (1) (1992) 63-82.

[CoS 92] J.-P. Courtiat, P. de Saqui-Sannes, **ESTIM : an Integrated Environment for the Simulation and Verification of OSI Protocols Specified in Estelle,** in: *Computer Networks & ISDN Systems 25* (1) (1992) 83-98.

[dNi 87] R. De Nicola, **Extensional Equivalences for Transition Systems,** *Acta Informatica 24*, (Springer - Verlag, Berlin Heidelberg, 1987) 211-237.

[EeW 93] H. Eertink, D. Woltz, **Symbolic Execution of LOTOS Specifications,** in: M. Diaz, R. Groz, eds., *Formal Description Techniques V* (North-Holland, Amst., 1993) 295-310.

[Fer 89] J.-C. Fernandez, **Aldébaran : A Tool for Verification of Communicating Processes**, Technical Report SPECTRE, Laboratoire de Génie Informatique - Institut IMAG, 1989.

[Gar 90] H. Garavel, **Compilation of LOTOS Abstract Data Types**, in: S. T. Vuong, ed., *Formal Description Techniques II* (North-Holland, Amsterdam, 1990) 147-162.

[GaS 90] H. Garavel, J. Sifakis, **Compilation and Verification of LOTOS Specifications**, in: L. Logrippo, R.L. Probert and H. Ural, eds., *Protocol Specification, Testing and Verification X*, (North-Holland, Amsterdam, 1990) 379-394.

[GHL 88] R. Guillemot, M. Haj-Hussein, L. Logrippo, **Executing Large LOTOS Specifications,** in: S. Aggarwal, K. Sabnani, eds., *Protocol Specification, Testing and Verification, VIII* (North-Holland, Amsterdam, 1988) 399-410.

[GuL 89] R. Guillemot, L. Logrippo, **Derivation of Useful Execution Trees from LOTOS Specifications by Using an Interpreter**, in: K.J. Turner, ed., *Formal Description Techniques*, Elsevier Science Publishers B.V. (North-Holland, Amsterd., 1989) 311-324.

[Hol 92] Gerard J. Holzmann, **Practical Methods for the Formal Validation of SDL Specifications,** *Computer Communications,* vol. 15, n°2, March 92.

[ISO 7498] ISO-TC97/SC16/WG1, **Information Processing Systems - Open Systems Interconnection - Basic Reference Model**, IS 7498, 1984.

[ISO 8073] ISO-TC97/SC6/WG4, **Information Processing Systems - Open Systems Interconnection - Connection-mode Transport Protocol Specification,** IS 8073, 1986.

[ISO 8807] ISO/IEC-JTC1/SC21/WG1/FDT/C, **Information Processing Systems - Open Systems Interconnection - LOTOS, a Formal Description Technique Based on the Temporal Ordering of Observational Behaviour**, IS 8807, February 1989.

[ISO 10023] ISO/IEC-JTC1/SC6/WG4, **Information Technology - Telecommunications and Information Exchange between Systems - Formal Description of ISO 8072 in LOTOS,** TR 10023, July 1992.

[ISO 10024] ISO/IEC-JTC1/SC6/WG4, **Information Technology - Telecommunications and Information Exchange between Systems - Formal Description of ISO 8073 (Classes 0,1,2,3) in LOTOS,** TR 10024, 1992.

[KLR 93] H. Kremer, J. v.d. Lagemaat, A. Rennoch, G. Scollo, **A Critical Synthesis of a Standardization Experience,** in: M. Diaz, R. Groz, eds., *Formal Description Techniques V* (North-Holland, Amsterdam, 1993) 231-246.

[Lan 90] R. Langerak, **Decomposition of Functionality: a Correctness Preserving LOTOS Transformation,** in: L. Logrippo, R. Probert, H. Ural, eds., *Protocol Specification, Testing and Verification X*, (North-Holland, Amsterdam, 1990) 229-242.

[Led 91] G. Leduc, **On the role of Implementation Relations in the Design of Distributed Systems,** in: *Coll. des Publications de la Faculté des Sciences Appliquées de l'Université de Liège, n° 130* (Liège, 1991), Thèse d'agrégation de l'enseignement supérieur, 283 p.

[Led 92a] G. Leduc **Conformance Relation, Associated Equivalence, and New Canonical Tester in LOTOS,** in: B. Jonsson, J. Parrow, B. Pehrson, eds, *Protocol Specification, Testing and Verification XI* (North Holland, 1992); updated in: Rept. No. S.A.R.T. 91/05/13, Université de Liège, Dept. Systèmes et Automatique, B28, B-4000 Liège 1, Belgium, August 1991.

[Led 92b] G. Leduc, **A Framework Based on Implementation Relations for Implementing LOTOS Specifications,** in: Computer Networks & ISDN Systems 25 (1) (1992) 23-41.

[Led 92c] G. Leduc, **A Methodology for the Design of Large LOTOS Specifications and its Application to ISO 8073,** OSI95/Deliverable ULg-3/P/V3, 09-1992, 89p. (SART 92/19/05)

[MaM 89] J. Mañas, T. de Miguel, **From LOTOS to C**, in: K.J. Turner, ed., *Formal Description Techniques* (North-Holland, Amsterdam, 1989) 79-84.

[MAQ 92] T. Miguel, A. Azcorra, J. Quemada, J. Mañas, **A Pragmatic Approach to Verification, Validation and Compilation**, in: *Proceedings of the Third Lotosphere Workshop & Seminar*, Pisa, Sept. 1992,

[Mar 91] F. Marso, **LOTOS as an Aid in the Design of Computer Communication Systems by Stepwise Refinement**, Doctoral dissertation, University of Liège, Dept. Systèmes et Automatique, B28, Liège, Belgium, Sept. 1991.

[Mas 92] T. Massart, **A calculus to Define Correct Transformations of LOTOS Specifications,** in: G. Rose, K. Parker, eds, *Formal Description Techniques IV* (North Holland, Amsterdam, 1992) 281-296.

[MaV 90] E. Madelaine and D. Vergamini, **Auto : A Verification Tool for Distributed System Using Reduction of Finite State Automata Networks**, in: S. T. Vuong, ed., *Formal Description Techniques II* (North-Holland, Amsterdam, 1990) 61-66.

[Mil 89] R. Milner, **Communication and Concurrency,** (Prentice-Hall Int., London,1989).

[Par 85] D. Park, **Concurrency and Automata on Infinite Sequences**, *Theorical Computer Science, 5th G1-Conference*, Springer Verlag, 1985.

[QAP 92] J. Quemada, A. Azcorra, S. Pavón, **The LOTOSPHERE Design Methodology**, in: *Proceedings of the Third Lotosphere Workshop & Seminar,* Pisa, Sept. 1992,

[QPF 89] J. Quemada, S. Pavón, A. Fernández, **Transforming LOTOS Specification with LOLA - The Parametrised Expansion**, in: K.J. Turner, ed., *Formal Description Techniques* (North-Holland, Amsterdam, 1989) 45-54.

[SSR 89] R. Saracco, J.R.W. Smith, R.Reed, **Telecommunications Systems Engineering using SDL,** (North-Holland, Amsterdam, 1989).

[vEi 92] P. van Eijk, **The LOTOSPHERE Integrated Tool Environment Lite**, in: G. Rose, K. Parker, eds, *Formal Description Techniques IV* (North Holland,Amsterd., 1992) 471-474.

[vGl 90] R.J. van Glabeek, **The Linear Time - Branching Time Spectrum,** in: J.C.M. Baeten, J.W. Klop, eds., CONCUR '90, *Theories of Concurrency: Unification and Extension*, LNCS 458 (Springer - Verlag, Berlin Heidelberg New York, 1990) 278-297.

[vKv 90] P. van Eijk, H. Kremer, M. van Sinderen, **On the Use of Specification Styles for Automated Protocol Implementation from LOTOS to C,** in: L. Logrippo, R.L. Probert and H. Ural, eds., *Protocol Specification, Testing and Verification X*, (North-Holland, Amsterdam, 1990) 157-168.

[VSv 91] C.A. Vissers, G. Scollo, M. van Sinderen, E. Brinksma, **Specification Styles in Distributed Systems Design and Verification,** in: *Theoretical Computer Science*, 89:179-206, 1991.

[WBL 91] C.D. Wezeman, S. Batley, J.A. Lynch, **Formal Methods to Assist Conformance Testing - A case study,** in: J. Quemada, J. Mañas, E. Vazquez, eds., *Formal Description Techniques III* (North-Holland, Amsterdam, 1991) 147-174.