Achieving Qualities During the Development of LOTOS Specifications

Thomas Lambolais, Jeanine Souquières

Achieving Qualities During the Development of LOTOS Specifications

Thomas Lambolais *Jeanine Souquières*

LORIA, UMR 7503, B.P. 239,
F-54 506 Vandœuvre-lès-Nancy, CEDEX, France.

December, 1997

Abstract

The LOTOS language is useful for the specification of services and communication protocols. Main recognised qualities of such specifications are simplicity, readability, provability, modularity (extensionability, rearrangeability) or composability. In this paper, we identify and formalise some of the qualities which arise "during" the development of LOTOS specifications. Some other qualities cannot be formally defined and must be obtained by specific methods. Formalised qualities can be achieved through development operators which consist of mechanisms that operate on intermediate states of a development.

Keywords

LOTOS, process algebra, specification, development process.

1 Introduction

Our goal is to assist the development of trustworthy LOTOS specifications. LOTOS (Language Of Temporal Ordering Specification) [9] is an algebraic language intended for specifying communication protocols in distributed systems. It is composed of two parts: (i) an abstract data types algebra, namely ACT ONE [5]; and, (ii) a process algebra inspired from Milner's CCS [14] and Hoare's CSP [7]. The process algebra is usually intended for the behavioral specification, i.e. the specification of terms whose main objective is *continuation*, whereas the data algebra is well suited for the data and functions, i.e. terms whose main objective is *termination*. In this paper, we are mainly concerned with the process algebra part.

Developed specifications should have a great number of 'good properties'. General and usually desired qualities of a LOTOS specification are [19]: *modularity*, *rearrangeability*, *generality*, *simplicity*, and *readability*. We refer to these properties as *final* and global qualities wanted on the whole specification.

In this paper, we deal with *intermediate* qualities desired for instance for single processes. Intermediate qualities are those qualities desired on intermediate states of

the specification. In the case of LOTOS, such intermediate qualities can be *effectiveness* [18] or *determinacy* [6, 14]. In this paper, we shall define these two notions and identify some others, such as *rigidity* and *flexibility*.

The second question we are concerned with refers to the *method* required to develop a formal text with required intermediate qualities. Our solution consists of:

- Developing a specification step-by-step, by decomposing the development process into states and transitions between these states.
- Managing the development through a workplan which consists of a tree of tasks. A task is a goal to achieve in the specification.
- Making the development evolve from one state to another by means of development operators.

These three points are the core of the PROPLANE model [13, 12]. PROPLANE is an interactive development model presented in section 2. Development operators capture the rationale and the methodological aspects of a construction. Intermediate desired properties of LOTOS processes to be defined can be added to proposed tasks.

This paper is structured as follows. In section 2, we briefly present the PROPLANE model which can support several construction methods via development operators. In section 3, we define and formalise some properties and qualities useful during the development process. In section 4, we show the main guidelines of an incremental and iterative development method. Section 5 concludes this presentation.

2 Brief Presentation of The PROPLANE Model

In this section, we present the main principles and the definitions of the basic terms of the PROPLANE model. The components are: *a development, development operators, a workplan, tasks, reductions, a product, links between workplan and product,* and the METAPROG language.

A *development* aims at building a text describing problem requirements. This text is called the *product*. In the PROPLANE framework [16, 17, 12], a development is performed step-by-step by only one developer. PROPLANE aims at assisting this developer to go from informal requirements to formal specifications. The main idea is to incrementally develop a *workplan* that describes the various steps of a development. This workplan is a tree of *tasks*. A task denotes a goal to achieve during the development: it can either be formal, like a piece of product to develop, or informal, like a piece of informal requirements to clarify. A step is achieved by the application of an operator, called a *development operator*. In each step, a task is *reduced* by decomposing it into subtasks corresponding to new subproblems. A development step consists of two viewpoints: the workplan and the specification, which are related by links. These links describe part of the specification under control by the workplan. They are expressed in a **meta-language**, i.e. a language which aims at describing the specification, called METAPROG.

Figure 1 shows an example of a development state. It contains a piece of the specification of the user part of the signalling protocol in ISDN systems. The left-hand side represents the workplan state, and the right-hand side shows the corresponding product described in the METAPROG language.

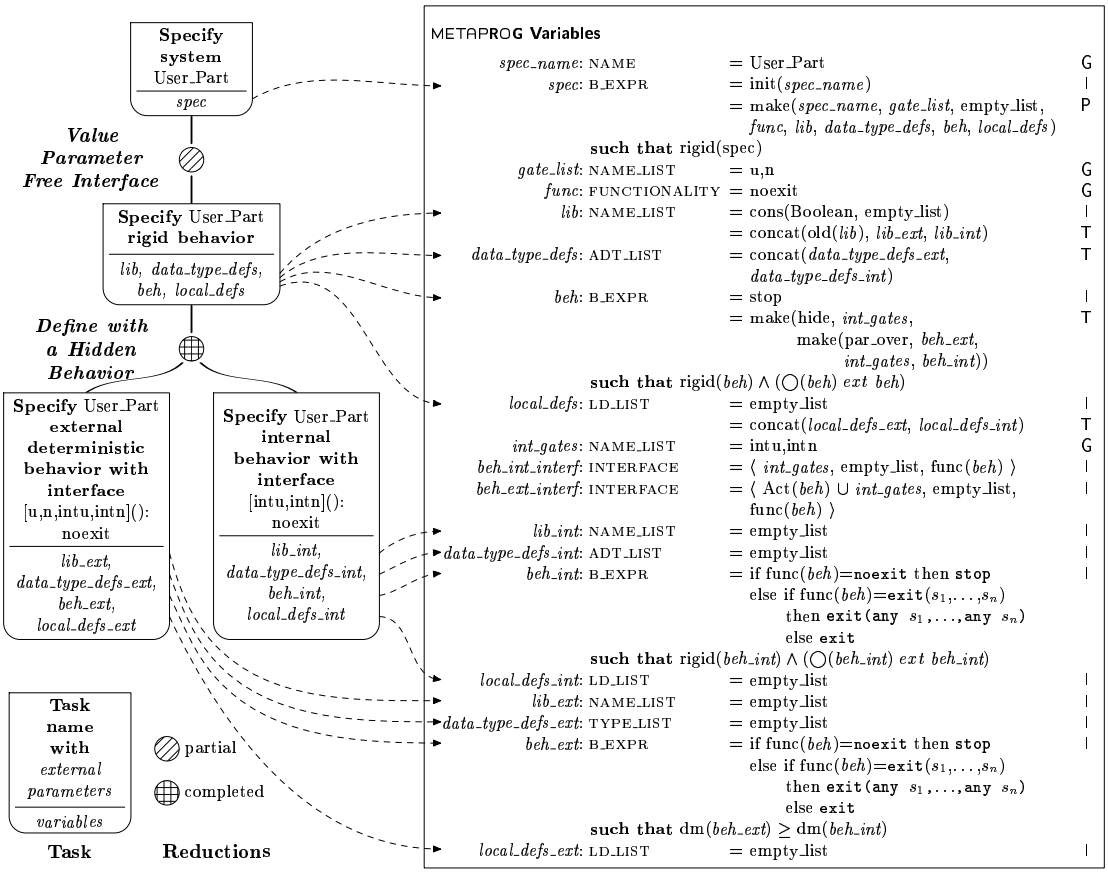

Figure 1: Workplan and associated meta-program

Workplan. Workplans are modelled by *and/or*-trees with two kinds of nodes: *task* nodes and *reduction* nodes. A *reduction* represents a way of accomplishing a task by introducing new subtasks. In the example, two development operators have been applied.

The states of a reduction (partial or completed) indicate whether a development operator has entirely or partially achieved its goal.

Product. The product, which consists of both formal and informal text, is set up simultaneously with the workplan. The right-hand side of Figure 1 represents a product state in LOTOS, described by the language METAPROG. The principle of METAPROG is that all product entries are identified by variables. These variables are typed according to the abstract syntax of the specification language. For example, in Figure 1, *beh* denotes the global behavior expression of the specification. It has been introduced by the task "**Specify** User_Part **rigid behavior**", and a second definition is given after the

application of operator "*Define with a Hidden Behavior*". In the right end column, variables are given a status, which can either be G (Given), when a value is interactively given by the user, I (Initial), when the value is defined by an intermediate METAPROG expression, or T (Terminate) when the definition cannot change any more.

METAPROG: Workplan–Product Links. The workplan is related to the product: each task defines one or several variables. These links are represented in Figure 1 by arrows between the variables appearing in both the workplan and the product state. These variables are described in the METAPROG language. METAPROG is made up of a generic description language and a logic. The logic is the temporal linear logic of Manna and Pnueli [15]. This logic, associated with the METAPROG functions and predicates, enables us to give some *pre/post-conditions* for these variables. In the following section, we shall define some meta-notions that can be incorporated within these pre/post-conditions.

In the right part of Figure 1, the post-conditions are the four lines beginning with '**such that**'. The 'rigid' predicate, the 'dm' function (determinacy measure) and the extension relation (ext) will be defined in the next section. For instance, the property '$\bigcirc(beh)\ ext\ beh$' denotes the fact that the next value of the behavior must extend the current one (the modal operator $\bigcirc$ is overloaded and applies on propositions and other kind of values): the chosen approach is an incremental one.

This makes the tasks of the workplan correspond to *properties* and *qualities* to achieve, as described by METAPROG postconditions. For example, the first operator applied, named "*Value Parameter Free Interface*", introduces only one subtask whose goal is to define a particular behavior: it must be defined as a *rigid* behavior.

The preconditions do not appear in Figure 1. They appear in the *development operator* definitions. They are used to define whether an operator is applicable or not.

Development Operators. Development operators work in parallel on both the workplan and the product, to reduce a task and to construct, or modify, the product text. Parameters of the operators are interactively acquired from the specifier and from the current development state.

Development Step. At each step of the development, the specifier chooses a task among the unreduced or partially reduced ones and an operator among those that could be applied to the task. If no operator is suitable, it is possible to define one dynamically.

An operator can be applied to a task if (i) the type of the variable to be defined and the type of the specification text to be added match, (ii) if the workplan parameters are available, and (iii) if the precondition is satisfied. The operator application prompts the specifier to supply values corresponding to the interactive parameters and checks whether the supplied values satisfy the operator precondition.

A step description keeps track of (i) all operators that could have been applied at this step, (ii) the interactive parameters given by the specifier, (iii) an informal rationale recording some justification for the choice, and (iv) possible constraints that must hold for the introduced tasks.

Prototype Tool. A prototype tool consisting of a plan editor connected to a product manager [16] is being implemented. The tool, equipped with a graphical interface, is

designed to be parameterised with specification languages and libraries of development operators. These libraries can be extended at any time. When a library of development operators is large enough, the main activity of the specifier no longer consists of writing down the specification, but in applying predefined operators to evolve the specification. This approach relieves the developers of many humdrum activities and allows them to focus on the reasoning underlying the elaboration of the specification.

Within PROPLANE, the reasoning is modelled as a logical sequence of assertions. The employed logic is the temporal logic of Manna and Pnueli, whereas assertions are specified by tasks and post conditions. In the following section, we define some of the properties that can be used to express such kinds of assertions.

3 Definitions of Managed Qualities During the Development

In this section, we formally define several meta-notions useful during a development process. These meta-notions are notions over the target language LOTOS: they either measure specific properties of LOTOS processes such as effectiveness or determinacy; or they identify several kinds of processes such as *components*.

3.1 Prerequisite

Let us first recall some definitions which can be found in the literature concerning determinacy and effectiveness. These definitions are explained in greater detail in [12]. We first need to recall some common definitions on process algebras: labeled transition systems, actions, processes, sequence of actions, acceptance sets [6, 3].

Let $\mathcal{G}$ be a countable set of observable actions. Let $Act = \mathcal{G} \cup \{\mathbf{i}\}$, $Act^+ = Act \cup \{\delta\}$ and $\mathcal{G}^+ = \mathcal{G} \cup \{\delta\}$, where $\mathbf{i}$ is the internal or silent action and δ is the successful termination action. Let $\mathscr{E}$ be a set of processes. Every LOTOS process in $\mathscr{E}$ corresponds to a, possibly infinite, labeled transition system.

Definition 1 (Labeled transition system) *A* labeled transition system *is a triple* $\langle \mathscr{E}, Act^+, \longrightarrow \rangle$, *where*

i) $\mathscr{E}$ *is an arbitrary set of processes*

ii) Act^+ *is an arbitrary set of actions*

iii) $\longrightarrow$ *is a relation in* $\mathscr{E} \times Act^+ \times \mathscr{E}$, *such that* $(P, a, P') \in \longrightarrow$ *is written* $P \xrightarrow{a} P'$ *and means that the process* P *may evolve to the process* P' *by performing the action* a.

For a sequence of actions, $s = a_1 \cdots a_n, n \geq 0, a_i \in Act^+$, let us recall Milner's notations:

$$
\begin{array}{rcl}
\hat{s} & \text{is} & \text{the sequence obtained from } s \text{ by removing all occurrences of } \mathbf{i}, \\
\varepsilon & = & \hat{\mathbf{i}}, \\
\xrightarrow{s} & = & \xrightarrow{a_1} \cdots \xrightarrow{a_n}, \\
\stackrel{s}{\Longrightarrow} & = & \xrightarrow{\mathbf{i}^*} \xrightarrow{a_1} \xrightarrow{\mathbf{i}^*} \cdots \xrightarrow{\mathbf{i}^*} \xrightarrow{a_n} \xrightarrow{\mathbf{i}^*} .
\end{array}
$$

LOTOS processes definition is not recalled here, we refer to [9, 12]. For any LOTOS process P, we define the following sets.

Definition 2 (Sets of actions, sets of action sequences and set of sets of actions)

$init(P) = \{a \in Act^+ \mid P \xrightarrow{a}\}$	*initial actions of P;*
$out(P) = \{a \in Act^+ \mid P \stackrel{a}{\Longrightarrow}\}$	*initial actions, observationaly accessible to P;*
$\overline{out}(P) = \{a \in \mathcal{G}^+ \mid P \stackrel{a}{\Longrightarrow}\}$	*initial observable actions, observationaly accessible to P;*
$\widetilde{out}(P) = \{a \in Act^+ \mid P \stackrel{\hat{a}}{\Longrightarrow}\}$	*actions from $out(P) \cup \{\mathbf{i}\}$;*
$Act(P) = \{a \in \mathcal{G} \mid \exists s \in \mathcal{G}^*, P \stackrel{s}{\Longrightarrow} \stackrel{a}{\Longrightarrow}\}$	*actions of P ;*
$Tr(P) = \{s \in \mathcal{G}^* \mid P \stackrel{s}{\Longrightarrow}\}$	*action sequences of P;*
$\mathscr{A}(P,s) = \{init(P') \mid P \stackrel{s}{\Longrightarrow} P' \wedge P' \not\xrightarrow{\mathbf{i}}\}$	*acceptance set of P after s.*

Note that $\mathscr{A}(P,s)$ is a set of sets. Acceptance sets are presented for instance in [6]. They can be used to define testing preorders.

Definition 3 (Observational determinacy [14]) *A process P is* deterministic *if for all $a \in Act$, $P \stackrel{a}{\Longrightarrow} P'$ and $P \stackrel{a}{\Longrightarrow} P''$ imply $P' \approx P''$, with P' deterministic.*

'$\approx$' denotes the observational equivalence (see [14]). This definition does not enable us to measure how deterministic a process is. For instance, we cannot say if a process P is more deterministic than a process Q. For such purposes, we refer to the cardinality of acceptance sets.

Definition 4 (Determinacy measure) *Let P be a process and s an action sequence. The function* dm *is defined by*

$$\begin{array}{rccl} \mathrm{dm}: & \mathscr{E} \times \mathcal{G}^* & \to & \mathbb{N} \\ & \langle P, s\rangle & \mapsto & \#\mathscr{A}(P,s) \end{array}$$

We can prove that P is deterministic according to Definition 3 if and only if $\mathrm{dm}(P,\varepsilon) = 1$. We now recall the definition of effective processes.

Effective transition systems. Several definitions of effectiveness can be found in the literature: De Simone [4] states that every system whose states have a countable set of derivatives is effective; Baeten, Bergstra and Klop [1] define effective systems as those that have a recursively enumerable set of states, and for which each state has a finite number of derivatives; Frits Vaandrager's definition [18], which is the most restrictive and the one used in this paper, says that:

A transition system is effective if, for all states, it is possible to effectively determine the set of derivative states. In each state, after an accepted action, the abstract machine knows what it can do.

3.2 Rigid Processes

Let us define the notions of *component*, *width* and *weakest width* of a process, and finally *flexible* and *rigid* processes. The problem is that it is not obvious to deal directly with the notion of effective processes as defined above. In particular, there is no link between the LOTOS syntax and effectiveness. Similarly, irregular processes are defined as processes whose labeled transition systems are infinite: it is not obvious for the specifier to know the construction mechanisms which lead to irregular processes. Usually, a LOTOS specifier deals with LOTOS processes and not with their corresponding transition systems. The rigidity notion is a practical means to avoid non-effective and irregular processes.

Definition 5 (Component) *Let $L,\ G \subseteq \mathcal{G}$ be two observable sets of actions, let $g \in L$ and $P,\ P_1,\ P_2,\ Q$ be some processes. P_1 and P_2 are* components *of P if P is defined from P_1 and P_2, or from P_1 and Q, or from P_1 only, by one of the following ways:*

$$\begin{array}{rl} P \equiv & \textbf{hide}\ L\ \textbf{in}\ P_1\ \texttt{|[}G\texttt{]|}\ P_2 \\ \mid & \textbf{par}\ g\ \textbf{in}\ L\ \texttt{|[}G\texttt{]|}\ P_1 \\ \mid & P_1 \gg Q \\ \mid & P_1\ \texttt{[>}\ Q \end{array}$$

Every component of P_1 or P_2 is a component of P.

Definition 6 (Width) *Let P be a process, $G \subseteq \mathcal{G}$ be a set of observable gates, $s_1, \ldots, s_n$ be sort names. The* width *of P is defined by the inductive function $wd\colon \mathscr{E} \to \mathbb{N}^{*+}$:*

$$\begin{array}{rcl} \textit{By default,}\quad wd(P) & = & 1 \\ wd(\textbf{hide}\ G\ \textbf{in}\ P) & = & wd(P) \\ wd(P\ \texttt{|[}G\texttt{]|}\ Q) & = & wd(P) + wd(Q) \\ wd(P\ \texttt{[]}\ Q) & = & max(wd(P), wd(Q)) \\ wd(a\texttt{;}P) & = & wd(P) \\ wd(stop) & = & wd(\textbf{exit}) \ = \ 1 \\ wd(P \gg Q) & = & max(wd(P)+1, wd(Q)) \\ wd(P\ \texttt{[>}\ Q) & = & max(wd(P)+1, wd(Q)) \\ wd(\textbf{par}\ G\ \textbf{in}\ P) & = & \sum_{g\in G} wd(P[g]) \\ wd(\textbf{choice}\ G\ \textbf{in}\ P) & = & max_{g\in G}(wd(P[g])) \\ wd(\textbf{let}\ v_1\colon s_1, \ldots, v_n\colon s_n\ \textbf{in}\ P) & = & wd(P(v_1, \ldots, v_n)) \\ wd(\textbf{choice}\ v_1\colon s_1, \ldots, v_n\colon s_n\ \textbf{in}\ P) & = & max_{(v_1,\ldots,v_n)\in(\mathcal{Q}(s_1),\ldots,\mathcal{Q}(s_n))}(wd(P(v_1, \ldots, v_n))) \end{array}$$

A component of width 1 is said to be sequential.

Unfortunately, thus defined width is not preserved by strong equivalence '$\sim$' [14]. Indeed, by the expansion theorem [14], every parallel structure is equivalent to a sequential one. The meta-notions of components and width are then purely syntactical notions. One way to define a stable function of width is the following.

Let us consider the laws of the LOTOS process algebra [9, 12], without the expansion theorem, like oriented rewriting rules. One can show, treating commutativity and associativity rules separately, that the rule system is confluent and terminates. Let us call *structural equivalence*, written $\cong$, the equivalence class determined by the rewrite system. If two structurally equivalent processes are in normal form, then they have the same width. The weakest width $wwd(P)\colon \mathscr{E} \to \mathbb{N}$ is defined as follows.

Definition 7 (Weakest width) *Let P be a process. Let P' be a distinguished normal form of P with respect to $\cong$. We call $wwd(P)$ the weakest number of parallel components of P, defined by*

$$wwd(P) := wd(P').$$

By definition, the weakest width is preserved by structural equivalence. Structural reorganisations are necessary during the construction. Weakest width is stable under restructuring operations.

Parallel recursive structures often lead to infinite minimal width. Effective processes having an infinite weakest width will be called *flexible*. The notion of flexibility is somehow the opposite of rigidity. In order to establish a link with the LOTOS syntax, we define the rigidity upon the width notion.

Definition 8 (Rigid and flexible processes) *A process P is* rigid *if $wwd(P) \leq \infty$. P is* flexible *if P is not rigid and P is effective.*

Hence, we have the following propositions. These two propositions are useful to support the development process of rigid processes.

Proposition 1 *Let P be a process.*

- *P is rigid if and only if all its derivatives are rigid.*
- *P is rigid if it has a finite number of rigid components.*

We refer to [12] for the details of the proof.

From Definition 6, it follows that it is possible to limit or to put guards on recursive calls in order to obtain rigid processes. The following proposition makes a link between syntactical considerations and rigid processes.

Proposition 2 *If no derivative of P is defined by a recursive expression inside a parallel composition ($|\,|$, $|||$, $|[S]|$, **par** S **in** $\_$); or on the left of a sequential composition ($>\!\!>$), or a disabling ($[>$), then P is rigid.*

The proof of this proposition uses Definitions 6, 7 and 8, and proceeds by structural induction on P.

4 Development Approach

4.1 Incremental Point of View: the Extension Relation

A conservative extension of a specification consists of providing new properties without constraining any of the existing ones. In the case of LOTOS, extension relations have been defined by Brinksma, Scollo and Steenbergen [2]. These relations are introduced to compare implementations with their specifications. They can be used to define a notion of dynamic conformance such as testing equivalence. In a development process, they can also be used to define a semantic view of an incremental approach. Brinksma's relation has been studied in such a context by Ichikawa et al [8]. The same relation has been applied to incremental specification and composition in [10].

The formal definition of the extension relation proposed in [2] is based on trace analysis. We call it the testing extension relation and denote it ext_t, where $P_2 \ ext_t \ P_1$ means that P_2 extends P_1 according to [2]. Unfortunately, the ext_t relation is not preserved under derivations. That is, if P extends Q, we cannot be sure that for all α-derivatives P' of P such that $\alpha \in init(Q)$, there exists an α-derivative Q' of Q such that P' extends Q'. Moreover, the ext_t relation is defined from a global point of view, and, like trace equivalence, is not easy to implement efficiently.

This leads us to define a stronger extension relation, ext, as follows.

Definition 9 (Observational extension) *Let P_1 and P_2 be two processes. P_2 observationaly extends P_1, written $P_2 \ ext \ P_1$, if and only if*

(i) *For all $a \in Act^+$, whenever $P_1 \xrightarrow{a} P_1'$, then for some P_2', $P_2 \stackrel{\hat{a}}{\Longrightarrow} P_2'$, and $P_2' \ ext \ P_1'$;*

(ii) *For all $\alpha \in \widetilde{out}(P_1)$, whenever $P_2 \stackrel{\hat{\alpha}}{\Longrightarrow} P_2'$, then for some P_1', $P_1 \stackrel{\hat{\alpha}}{\Longrightarrow} P_1'$, and $P_2' \ ext \ P_1'$.*

We can show (see [12] for full details), that ext is stronger than ext_t, i.e. $P \ ext \ Q$ implies $P \ ext_t \ Q$. Moreover, unlike observational simulation, ext_t is a preorder with respect to observational equivalence. Although ext does not have the same selection power as ext_t (especially because it is stronger), it is easier to implement. The reason is that it is defined like a bisimulation. It implies the set of states to observe decreases at each step.

Many development operators defined in the PROPLANE model are *syntactically incremental* in the sense that they add new constructs to LOTOS processes. The ext relation enables us to define *incremental development operators* under PROPLANE in an *operational* and a *semantic* sense. These operators are claimed to be operational since they can effectively be translated into the METAPROG language, whereas the translation would not be effective if we had used the ext_t relation. They are also claimed to be semantic since they deal with the adjunction of new properties to existing behaviors. Incremental semantic development operators are studied in [12].

4.2 Example of an Incremental Development Method

Unlike effectiveness or rigidity, a quality like 'simplicity' is subjective and cannot be formalised. The simplicity criterion can be handled via a development method. Here are the main guidelines and steps of an example of a purely incremental method to develop protocol specifications in LOTOS:

1. Simplify the requirements: either (a) isolate a part of requirements; or (b), make an abstraction of some requirements. At the same time, keep track of further extensions to incorporate.

   The result of this step is an informal text which constitutes the new requirements, and a non-exhaustive list of informal extensions;

2. Specify roughly the simplified requirements, without having in mind specific qualities;

   The result of this step is a potentially incomprehensible or semantically incorrect piece of LOTOS specification. It must be regarded as a first prototype intended to be reviewed and improved;

3. Study criteria like rigidity/flexibility and effectiveness on the formal text developed in the previous step. This study should result in a *simplification* and *correction* of the first specification;

4. Consider one of the extensions identified at step 1. Proceed again as with step 2, but keep in mind the need to follow the ext relation;

5. Go back to step 3 and check that the ext relation is satisfied;

6. Repeat steps 4 and 5.

Hence, this method is iterative. It is based on the following statement: in general, the first ideas of a developer are not the simplest ones; a simple idea may be obtained by refining a previous one.

The second point of the method is to mix informal and formal approaches. The first step of the method is an informal one. The underlying motivations are the same as in Knuth's literate programming approach [11]: take benefit of natural language and make the specifier translate his/her ideas as if (s)he would communicate to other people rather than to a machine.

This example of method can be used like a frame in the PROPLANE model. However, it cannot be directly described by development operators: each one of the above steps needs several operator applications. This implies various kinds of operators. The first ones (Step 1) produce informal comments in the LOTOS specification. In the PROPLANE environment, the results are structured LaTeX formatted outputs. The second ones (Step 2) are syntactical operators. The third ones manage the properties defined in Section 3. They consist of analysis operators rather than construction ones. During step 4, most operators are semantically incremental ones, based on the ext relation. Step 5 is dedicated to check pieces of the specification that have not been developed using incremental operators. At present, this verification is not supported by operators.

5 Conclusion

The PROPLANE model is a framework which allows the user to proceed step by step by means of development operators. We identified and formally defined some *intermediate* qualities, like rigidity, flexibility, determinacy and effectiveness. Since we gave operational definitions of such properties, it is possible to manage them by development operators *during* the development process.

Other qualities cannot be directly obtained by development operators. We formally defined an operational notion of extension over LOTOS processes. In the case of the 'simplicity criterion', we illustrated how an interactive and incremental method based on our extension relation could proceed. At each step of this method, several development operators are implied.

Further work consists of studying validation and verification operators, since our method includes checking tasks during the development.

References

[1] J.C.M. Baeten, J.A. Bergstra, and J.W. Klop. On the Consistency of Koomen's faire Abstraction Rule. *Theoretical Computer Science*, 51(1/2):129–176, 1987.

[2] Ed Brinksma, Guiseppe Scollo, and Chris Steenbergen. LOTOS Specifications, their Implementations and their Tests. In Gregor von Bochmann and Behcet Sarikaya, editors, *Proc. Protocol Specification, Testing and Verification VI*, pages 349–360, Amsterdam, The Netherlands, June 1986. North-Holland.

[3] Rance Cleaveland and Matthew Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, 5:1–20, 1993.

[4] Robert de Simone. *Calculabilité et expressivité dans l'algèbre de processus parallèles MEIJE*. PhD thesis, université de Paris VII, 1984.

[5] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1 (Equations and Initial Semantics)*, volume 6 of *ETACS Monographs on Theorical Computer Science*. Springer Verlag, 1985.

[6] Matthew Hennessy. *Algebraic Theory of Processes*. Series in the Foundations of Computing. MIT Press, 1988.

[7] Charles Antoine R. Hoare. *Communicating Sequential Processes*. Prentice Hall international series in Computer Science, London, 1985.

[8] H. Ichikawa, K. Yamanaka, and J. Kato. Incremental specifications in LOTOS. In Luigi M. S. Logrippo, Robert L. Probert, and Hassan Ural, editors, *Proc. Protocol Specification, Testing and Verification X*, pages 183–196, Amsterdam, The Netherlands, 1990. North-Holland.

[9] ISO, IS 8807. *Information Processing Systems, Open Systems Interconnection, LOTOS—A Formal Description Technique Based On the Temporal Ordering of Observational Behaviour*, July 1988.

[10] Ferhat Khendek and Gregor von Bochmann. Incremental Construction Approach for Distributed System Specifi cations. Technical Report 857, Département d'informatique et de recherche opérationnelle, université de Montréal, Faculté des arts et des sciences, C.P. 6128, succursale A, Montréal, P.Q. H3C 3J7, January 1993.

[11] Donald E. Knuth. Literate programming. *The Computer Journal*, 27(2):97–111, May 1984.

[12] Thomas Lambolais. *Modélisation du développement de spécifications LOTOS*. PhD thesis, Institut national polytechnique de Lorraine, October 1997.

[13] Nicole Lévy and Jeanine Souquières. Modelling Specifi cation Construction by Successive Approximations. In M. Johnson, editor, *6th International AMAST conference*, pages 351–364, Sydney, Australia, December 1997. Lecture Notes in Computer Science 1349, Springer Verlag.

[14] Robin Milner. *Communication and Concurrency*. HOARE, C.A.R., Prentice Hall International, 1989. Series in Computer Science.

[15] Amir Pnueli and Zohar Manna. *The Temporal Logic of Reactive and Concurrent Systems • Specifi cation •*, volume 1. Springer-Verlag, 1992.

[16] Jeanine Souquières and Nicole Lévy. PROPLANE: A Specifi cation Development Environment. In *5th International Conference on Algebraic Methodology and Software Methodology Amast'96*, Munich (G), July 1996. Lecture Notes in Computer Science 1101, Springer-Verlag.

[17] Jeanine Souquières and Robert Darimont. La description du développement de spécifi cations. *Technique et Science Informatiques*, 14(9), novembre 1995.

[18] Frits W. Vaandrager. Expressiveness Results for Process Algebra. In J.W. de Bakker, W.P. de Roever, and G. Rozenberg, editors, *Semantics: Foundations and Applications*. Proceedings of the REX Workshop, Springer-Verlag LNCS, 1993.

[19] Christopher A. Vissers, Guiseppe Scollo, Marten van Sinderen, and Ed Brinksma. Specifi cation Styles in Distributed Systems Design and Verifi cation. Technical report, University of Twente, Fac. Informatics, 7500 AE Enschede, NL, 1990.

Corresponding author:

Thomas Lambolais
LORIA, Campus Scientifi que, B.P. 239
F-54506 Vandœuvre-lès-Nancy, CEDEX, FRANCE
Tel. 00 33 3 83 59 20 19; Fax 00 33 3 83 41 30 79
e-mail: `lambolai@loria.fr`