E-LOTOS: an Overview

Gabriel Huecas[1], Luis F. Llana-Díaz[2], Tomas Robles[1], and Alberto Verdejo[2]

[1] Dpto. Ingeniería Telemática
Universidad Politécnica de Madrid. Spain
gabriel@selva.dit.upm.es
trobles@dit.upm.es
[2] Dpto. Sistemas Informáticos y Programación
Universidad Complutense de Madrid. Spain
llana@sip.ucm.es
jalberto@eucmos.sim.ucm.es

Abstract E-LOTOS (Enhanced LOTOS) includes several new features according with the user's needs detected during the last years. Simplification of the language, inclusion of programming language features (especially imperative features), modularity, new operators, and inclusion of formal support for time specification and analysis, make the new language much more industrial applicable, extend their applicability fields, and offers a powerful tool to cover Systems development cycle from requirements capture to implementation production.

Keywords: E-LOTOS, LOTOS, Formal Description Techniques, Formal Methods.

1 Introduction

At the Internet age, things change faster than ever before. In this context, LOTOS [ISO88] (Language of Temporal Ordering Specification), a Formal Description Technique developed within ISO for the formal specification of open distributed systems, is currently under revision in ISO [Que98], in the Work Item "Enhancements to LOTOS," giving rise to a revised language called E-LOTOS (Enhanced LOTOS). Maintaining the strong formal basics of LOTOS(based on CCS [Mil89], CSP [Hoa85], and ACT ONE [EM85]), the new language intends to include most of the user's requirements related with expressive power and structuring capabilities, besides user-friendliness. User's requirements come from the industrial environment, that wants to use a language closer to the actual high level programming languages, reducing the learning curve of the new engineers, and the availability of tools that produce effective implementations of the specified products. Scientific community wants to increment the expressive power of the language to deal with classical problems like time analysis, new operators, and unification of some language structures. Among the enhancements introduced in E-LOTOS, the most important ones are:

- the notion of quantitative time: in E-LOTOS we can define the exact time at which events or behaviors may occur,

- the data part, both the definition of new data types and the construction of values of predefined types, that provides data types similar to those of the (functional) programming languages, maintaining the formal support,
- modularity, which allows the definition of types, functions, and processes in separate modules, by controlling their visibility by means of module interfaces, and the definition of generic modules, useful for code reuse,
- some imperative sentences (loops, if-then-else, case, etc), that make the language useful for covering the last steps of the software life cycle, when implementations are developed, and make easier to the habitual user of this programming paradigm the job of specifying systems. Besides these instructions, the use of *write-many* variables (variables that can be assigned several times) is another imperative feature introduced in E-LOTOS.

This paper will explain the main implications of the inclusion of those elements, and make some discussions about the alternatives discussed during the standardization process, remarking the implications in the use of the language in Software engineering process, systems specifications, and standards body works. In the following sections, we describe each of the above points.

2 Time

The capability of measuring time is very interesting when specifying protocols, see [QMdL94,LL94]. This feature was not included in the definition of LOTOS because the study of introducing time in process algebras was just starting.

Time can be introduced in the operational semantics of process algebras in two equivalent ways: with time transitions and action transitions [LL94] or with timed action transitions [QMdL94]. In the first one, there are two kind of transitions: timed transitions and action transitions. An action transition stands for a communication as in LOTOS, while a timed transition indicates passing of time: $B \xrightarrow{t} B'$ indicates that behavior B' is the behavior B after t units of time. The idea of a timed action transition is to join an action transition and a time transition in just one transition. So $B \xrightarrow{g\,t} B'$ indicates that behavior B communicates throw gate g at time t. It is also necessary to indicate a function of time passing: $Age(B,t)$ is the behavior B after t units of time. The first one is the approach chosen in E-LOTOS because it is compatible with LOTOS: it is only necessary to introduce the timed transitions.

The main characteristics of time in E-LOTOS are:

- The time domain can be either discrete or continuous. The standard does not define a type time but describes the features that this type has to fulfill in any implementation of E-LOTOS. These properties are:
  - the time domain is a commutative, cancellative monoid with addition + and unit 0. Thus, it satisfies the properties:
    - $d_1 + d_2 = d_2 + d_1$
    - if $d_1 + d = d_2 + d$ then $d_1 = d_2$

* $d_1 + (d_2 + d_3) = (d_1 + d_2) + d_3$
* $d + 0 = 0 + d = d$

where d_1, d_2, and d are variables over the time domain.

- the order given by $d_1 \leq d_2$ if and only if $\exists d \,.\, d_1 + d = d_2$ is a total order.

At first glance, it seems that is better to have a continuous time domain. But it has several problems. First, one can specify a Zeno behavior:

$$B_1 \xrightarrow{1/2} B_2 \xrightarrow{1/4} B_3 \cdots B_k \xrightarrow{1/2^k} B_{k+1} \cdots$$

This process will not leave time pass 1 unit of time, this is an undesirable behavior. Another problem is more philosophical, can two processes communicate at an exact instant?, for instance $\frac{1}{3}$. It seems more realistic to consider a discrete time domain: the unit of time can be as small as one wants, but it is fixed.

– Internal actions are urgent: a process cannot idle if it can execute an internal action. It guarantees the progress of a system; if there were no urgent actions the systems could idle forever. Moreover, it is enough to consider as urgent only internal actions. A complete system can be seen as a black box where all actions are hidden. In E-LOTOS actions carry out on hidden gates are urgent.
– Time is deterministic. If non determinism appears in a behavior is because it is *doing something*. It is not possible that a behavior B evolves to two different behaviors B_1 and B_2 only by time passing. The E-LOTOS static semantics, that checks which behaviours are semantically correct, ensures time determinism, by rejecting behaviours that could be time nondeterministic.

3 Data types

One of the most criticized part of LOTOS is the data type definition language, ACT ONE [EM85], based on algebraic semantics, where equations are used to define the data types semantics. This language is not too user-friendly and suffers from several limitations such as the semi-decidability of equational specifications (tools cannot implement a procedure that checks equality between two values in a general way), the lack of modularity, and the inability to define partial operations.

In E-LOTOS, ACT ONE has been substituted by a new language in which data types are declared in a similar way as they are in functional languages (ML, Haskell), by using more constructive type definitions, based on a syntactic and semantic distinction between constructors and functions, which ease the computation of data values, defined in an operational way. In contrast to LOTOS, where there is a separation between processes and functions, E-LOTOS considers functions as a kind of processes. A function in E-LOTOS is any process with the following characteristics: it is deterministic; it cannot communicate (i.e. it has no gates), and so its only capabilities are to return values and raise exceptions; and

has no real-time behaviour (i.e. a function is an immediately exiting process). Therefore, the expression (sub)language is very similar to that of behaviours, once the elements related to these characteristics are removed. This allows that the semantics used to evaluate expressions is the same operational semantics used to describe the execution of a behaviour.

E-LOTOS has a set of predefined, "built-in" data types (bool, nat, int, rational, float, char, and string) with associated operations. These data types are available within any specification. The language allows record types to be easily defined and dealt with: it is possible to declare a record by giving the list of its fields together with their types; for example, we can define the record type

(*name* => string, *address* => string, *age* => nat)

and we can access to each field with the "*dot*" notation, for example *rec*.*name*, provided variable *rec* has the above type.

There are also a set of type schemes that are translated to usual type and function declarations, and that are used to make easier the definition of typical types (as suggested by the "rich term syntax" of [Pec94]). For example, we can define *enumerated* data types, like

```
type color is
  enum Blue, Red, Green, Yellow, Pink
endtype
```

In E-LOTOS sets and lists of values of a given type, like sets of naturals, or lists of strings, may be defined, with (predefined) functions to manipulate their values.

Besides the predefined types, the user can define two kinds of types: type synonyms and new data types. A type synonym declaration simply declares a new identifier for an existing type, like type Complex to represent complex numbers,

```
type Complex is
  (real => float, imag => float)
endtype
```

The declaration of a new data type consists of the enumeration of all the constructors for that type, each one with the types of its arguments; for example, the type of messages, data messages or acknowledgment messages, may be defined as follows:

```
type pdu is
  send(packet,bit) | ack(bit)
endtype
```

These new data types may be recursive, for example, a queue of integers,

```
type intQueue is
    Empty
  | Add(intQueue,int)
endtype
```

and functions dealing with this type can be defined in a recursive way as we show in the following section.

Another feature regarding data types is what is called in E-LOTOS "extensible" record types. The record type

(*target* => destination, **etc**)

represents a type of records with *at least* one field called *target* of type destination. This, together with the built-in subtyping relation between record types (a record type is subtype of another record type, that must be extensible, if the former has at least all the fields the latter has) means that we have as values of the above type all the record values that have *at least* a field with that name and type.

4 Modularity

Due to the LOTOS limited form of modularity, whose modules only encapsulate types and operations but not processes, and do not support abstraction (every object declared in a module is exported outside)[1], E-LOTOS has a new modularization system, which allows to define a set of related objects (types, functions, and processes) inside a module and to control what objects the module exports (by means of interfaces), to include within a module the objects declared in other modules (by means of importation clauses), to hide the implementation of some objects (by means of opaque types, functions, and processes), and to build generic modules.

In order to facilitate this modularization, a separation between the concept of module *interface* and definition *module* is done. An interface declares the visible objects of a module and what the user need to know about them (the name of a data type or a function profile, for example). A module gives the definition (or implementation) of objects (visible or not).

E-LOTOS allows to build *generic modules*, that is, modules with parameters. The features of these parameters are specified by means of interfaces. For example, we can define the data type of queues whose elements are of any type by defining the requirements of these elements in an interface:

```
interface Data is
  type elem
endint
```

and by defining a generic module whose parameters has to fulfill the requirements in the interface:

```
generic GenQueue(D:Data) is
  type queue is
      Empty
    | Add(queue,elem)
```

[1] A critical evaluation of LOTOS data types from the user point of view can be found in [Mun91].

```
  endtype
  function addQueue(q:queue,e:elem):queue is
    Add(q,e)
  endfunc
  function front(q:queue):elem raises [EmptyQueue] is
    case q in
        Empty -> raise EmptyQueue
      | Add(Empty,?e) -> e
      | Add(Add(?q,?e),any:elem) -> front(Add(q,e))
    endcase
  endfunc
  ...
endgen
```

In order to use a generic module we have to instantiate it, by providing actual parameters, which must be modules that match the corresponding interface. A module matches an interface whether it implements at least the objects declared in the interface. We can instantiate the above generic queue to make a queue of natural numbers:

```
module NatQueue is
  GenQueue(D => NaturalNumbers renaming(types nat := elem))
endmod
```

5 Imperative Features

The need of imperative features was soon established in the first years of LOTOS. For example, the specification of loops by means of recursion was counterintuitive and syntactically heavy and uncomfortable.

In E-LOTOS, the basic and common imperative constructors were included, such as conditionals (**if** – **then** – **else**, **case**) and loops (**for**, **while**, infinite **loop**). Of course, behavior sequentialization was given an homogeneous treatment, by merging sequential composition ("`;`") and behavior enabling ("`>>`") in just one sequential operator ("`;`").

Write-once variables of LOTOS were a point of criticism: they fits pretty well in the operational semantics of LOTOS, but imposed a strong restriction on the use of variables, besides their counterintuitive syntax. In E-LOTOS, write-many variables were introduced, but on a safe use assured by static semantics means was a must. The key problem was compositionality: behaviors are supposed to interact explicitly, avoiding the always negative side-effects.

The solution required to choose a model to which specifiers were already exposed: the UNIX's "`fork(1)`," where a branching implies duplication of variable space. With that, a operational semantics was still affordable. The remaining problem is when branches join: $B_0\,;(B_1 \,||\, B_2)\,;B_3$. The static semantics assured that variables are modified by just B_1 or B_2, but no both at the same time.

Typed gates, functions, "**in**" and "**out**" parameters, abbreviated parameters lists were introduced as helpful shortcuts or useful constructors thanks to the industry feedback to LOTOS community.

"**out**" parameters substitute the old "**exit**" functionality, improving flexibility and readability. Functions and processes were unified at the semantic level: a function is a process which performs just a termination action upon termination.

One of the most powerfull characteristic of LOTOS was the modelling of synchronization events: a gate name followed by an expression list. However, there was no way to specify globally which was the communication model for a gate. In other words, each process communicating on certain gate should know the whole event structure. But there was no way to fix such structures. Two improvements have been included in E-LOTOS to help designers: the first one is that gates can be typed. However, E-LOTOS allows the old LOTOS style, in which gates were not typed. In fact, that means that *any* type would appear in the event structure. E-LOTOS implicitly types gates with the predefined type **any**, which obviously matches every type in a specification. The second one is that partial synchronization is now allowed by means of record subtyping. That is of relevance for using a constraint oriented approach which can be performed in E-LOTOS in a much more concise way as each constraint is aware only of the part of the event structure related to it.

6 New Operators

The expressive power of LOTOS was one of its stronger points. But some improvement were added in E-LOTOS.

From different fields, the need for exceptions was clearly established. They were introduced, together with the "**trap**" operator that describes the exception handlers. "**raise**," "**break**," and "**signal**" are the three different ways to throw an exception, which particular meanings for several constructions. Exceptions can be thrown outside a system (in JAVA, for example, all exceptions must be captured by a program). There are no default exception manager, and there are no "finally" clause (a "finally" clause is always executed, whether an exception was thrown or not and independently of the exception type).

A more general parallel operator was introduced. It is n-ary and supports the synchronization of n out of m processes ($n \leq m$). The new operator is more readable, as it explicitly identifies the synchronizing gates for each composed behavior. Another parallel operator, called parallel over values, was also introduced. It represents the interleaving of a series of instantiations of a common (template) behaviour, one for each value in a given list. This allows, for example, to put in parallel a series of nodes of a network, each one with a different identifier, taken from a list of identifiers.

The Suspend/Resume operator generalizes the LOTOS disabling. With it, a behaviour may be suspended (as old disabling) and resumed explicitly. That allows the modeling of interruptions, immediate treatment, etc.

A new, explicit renaming operator, applicable to gates and exceptions, were introduced. It allows not just name changing, but also structure modification, as merging or splitting gates and adding or removing fields from the structure of events.

The "**let**" constructor was superseded by a more common, intuitive "**var**" variable declaration sentence. Besides variable declaration and scope hiding, it allows initialization as well.

A minor difference in E-LOTOS is the removal of precedences. All binary operators (`||`, `|||`, `|[...]|`, `[..>`) have the same precedence, which is less that "**;**" precedence (common sense). To mix several of these binary operators, explicit constructors are provided (**dis** ... **enddis**, **fullsync** ... **endfullsync**, etc).

7 Conclusions

E-LOTOS new language offers key characteristics to succeed both in the industrial and the academically environments. From the industrial point of view the new language offers a simplified syntax, more friendly data types (similar to high level language data types), imperative-like structures (loops, decisions, exceptions, etc.), write-many variables, etc. From the academically point of view it offers formal support for time properties, new parallel operators, typed gates, and many other minor modifications that increment the systems description power of the language. Nevertheless, the language does not lose their stronger point: a good and solid formal support for verification, validation, and system analysis in general. During the standardization process, inputs have been received and integrated from research and industrial groups of Europe, America, and Asia. The new language reflects new trends in the formal language definition with influences from another formal languages and borrowing elements from common high level language paradigms.

References

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1.* Springer-Verlag, 1985.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[ISO88] ISO/IEC. *LOTOS—A formal description technique based on the temporal ordering of observational behaviour.* International Standard 8807, International Organization for standardization — Information Processing Systems — Open Systems Interconnection, Genève, September 1988.

[LL94] G. Leduc and L. Lèonard. An enhanced version of timed LOTOS and its application to a case study. In R. Tenney, P. Amer, and Ü. Uyar, editors, *FORTE '93.* North Holland, 1994.

[Mil89] Robin Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[Mun91] Harold B. Munster. LOTOS specification of the MAA standard, with an evaluation of LOTOS. NPL Report DITC 191/91, National Physical Laboratory, Teddington, Middlesex, UK, September 1991.

[Pec94] Charles Pecheur. A proposal for data types for E-LOTOS. Technical report, University of Liège, October 1994. Annex H of ISO/IEC JTC1/SC21/WG1 N1349 Working Draft on Enhancements to LOTOS.

[QMdL94] J. Quemada, C. Miguel, D. de Frutos, and L. Llana. A timed LOTOS extension. In T. Rus, editor, *Theories and Experiencies for Real-Time System*

Development, volume 2 of *AMAST Series in Computing*, pages 239–263. World Scientific, 1994.

[Que98] Juan Quemada, editor. Final committee draft on Enhancements to LOTOS. ISO/IEC JTC1/SC21/WG7 Project 1.21.20.2.3., May 1998.