RSI Comments on Working Draft on Enhancements to LOTOS ISO/IEC JTC1/SC21/WG7 WI 1.21.20.2.3

1997/06/15

Romania votes AGAINST promoting the Working Draft on Enhancements to LOTOS as a Final Draft, with the following 4 comments.
Romania will change its NO vote to YES with the satisfactory resolution of the Major Technical comments below.

ITEM: **RSI-1** **Anonymous records**

CLAUSE: General

QUALIFIER: Major technical

RATIONALE: Although anonymous records are used to formalize the semantics of the language, we believe that their introduction in E-LOTOS language is not a good design choice. The main reasons are the following:

1. **No added expressiveness.** As the number of anonymous records in a given description is necessarily finite, it is always possible to translate any description with anonymous records into an equivalent one in which all distinct record types are given unique type identifier. Moreover, the ODP Trader example given in Annex A of the document does not use this "capability" of the language.

2. **A questionable and limited convenience.** From the practical point of view, we see two advantages to anonymous records:

   - The possibility of using structured types without declaring them.
   - The possibility to declare functions that returns several results.

   However, in the same time, these two advantages could be considered as disadvantages: the first one goes against a fundamental software methodology as discussed at point 5 below and leads to lack of reusability; the second one is a limited version of having '**in**'/'**out**' parameters for functions and processes.

3. **Lack of convenience.** The ability to call functions declared with named parameters using positional arguments is lost by considering anonymous records as arguments for function. The positional call is possible *only* when the function was declared with a tuple argument (!).

4. **Complexity of the semantics.** Introducing anonymous records in E-LOTOS makes the static and the dynamic semantics more complex:

   - It introduce the notion of *type expressions* at the language level. In standard LOTOS, the type of each expression is simply a sort identifier. With anonymous records the type of each

expression is either a sort identifier, or a record of type expressions. Nested records are thus allowed.

Type expressions are used at the semantic level to express the semantics of imperative features. However, this cannot justify their presence at the language level. Moreover, the semantics given by means of type expressions presents some inconveniences as discussed in ITEM 2.

- It introduce the notion of *structure equivalence* for type expressions. In standard LOTOS, two expressions have the same type iff their sort identifier are identical (this is *name equivalence*). In the current draft, the situation is more complicated, a mixture of name and structure equivalence is used.

5. **Subversion of software methodologies.** The anonymous records go against the recommended methodologies for software design and programming. It is to be feared that lazy programmers and specifiers may write poorly documented descriptions, in which complex data are structured in (nested) records without naming the type of this data (as it is often the case in LISP).

   This is a common concern for most programming languages designers. The name equivalence was chosen for languages as Pascal, Ada, C, C++, etc.

   Also, anonymous records and structure equivalence go against object-oriented methodologies for design of complex systems, which recommend to give explicit names to every data structure manipulated by the software under design.

6. **Absence of compatibility with standard LOTOS.** As mentioned above, all types in LOTOS are sort identifiers and the equivalence of types is the name equivalence. As long as the upward compatibility requirement is stated, it has also to be stated to reject anonymous records and structural equivalence of types.

7. **Absence of interoperability with other ISO languages.** To be a successful and widely used language, E-LOTOS should interoperate smoothly with other main computer languages. Selecting anonymous records approach put restrictions on the interoperability of E-LOTOS with other languages:

   - To be able to implement an external function in Ada, C, C++, one must either declare the function with a tuple argument, or , in absence of subtyping, do static semantics conversions from the labelled function call to the positional call. The reverse problem is also true: to call from a C program an E-LOTOS function, one have to translate always the labelled call into a positional one.
   - The use of anonymous records in the profile of functions does not allow the mixture between the '**in**' and '**out**' parameters of the functions or processes. This lack will prevent to invoke, from an E-LOTOS description, routines defined in libraries of C, where the '**in**' and '**out**' parameters are merged. Moreover, it will be not possible to use E-LOTOS to specify the behaviour of systems which interfaces are specified in IDL (where mixture of parameters is also allowed), thus limiting the usefulness of E-LOTOS for description of ODP systems.

8. **Inefficient implementation.** As regards efficiency of code generated by compliers, it is clear giving a name to each typed used into the specification is easier than generating new name for each anonymous record used and testing structural equivalence of types.

PROPOSED CHANGE: The anonymous records must be eliminated from the language. The type expressions allowed to the language level have to be sort identifiers, '**none**', and '**any**' types.
The record types have to be named using type synonym declarations. In this way, the syntax of communication does not change.

More complex type expressions have to be used only in the semantics.

ITEM: **RSI-2 Built-in subtyping**

CLAUSE: General

QUALIFIER: Major technical

RATIONALE: Built-in subtyping in records is a dangerous way to introduce polymorphism in E-LOTOS. The main reasons which justify our reject of such design choice are the following:

1. **Absence of compatibility with standard LOTOS.** It implies the suppression of overloading facilities that currently exist in LOTOS. This go against the goal of compatibility with LOTOS defined in the scope of the New Work Item.

2. **A questionable compatibility problem.** The presence of subtyping is justified by upward compatibility with the LOTOS untyped gates. This is not a real problem how long it can be easily solved by translating LOTOS description with untyped gates in E-LOTOS descriptions with gates having as type the union type of all types of data communicated over each gate.

3. **Complexity of the type-checking algorithm.** The type-checking algorithm for subtyping is significantly more complex (exponential time) than one for overloading (linear time). This will increase the complexity of E-LOTOS type-checkers, with the large risk that subtyping based protocol descriptions not to be checked in a reasonable amount of time.

4. **Complexity of the type system.** In absence of subtyping, no prove that each expression has a principal type have to be given. We mention that it is not proved in the current document.

5. **No useful added expressiveness.** The kind of subtyping needed in ODP systems is merely an equivalence relation between behaviours combined with an inheritance relation between modules (objects). The built-in subtyping introduced in E-LOTOS does not respond to ODP needs, although its name may induce confusion in the mind of ODP users.

   Moreover, the built-in subtyping is less powerful than the combination of generic modules with user defined subtyping on abstract data types. A such proposal was presented in a early version of the module system. It has the advantage to solve subtyping *statically*, when modules are flattened.

6. **Dynamic type checking.** The subtyping is heavily based on dynamic type-checking. From the practical point of view, we see two advantages to dynamic type-checking:

   - There are interesting programs that are more easily expressed in a dynamically type checked language. However, there are many, many programs that can quite easily and conveniently be expressed using a statically typed language.
   - Statically type checked are by necessity more verbose than dynamically type checked ones. Some of verbosity is useful for documentation purposes, but the rest is only of interest to the compiler.

   However, in the same time we mention three disadvantages of the dynamic type-checking which advocate in favor of static type-checking:

   - Errors detected at compile time do not have to be discovered at run time.
   - With dynamically type checked code, it is possible to ship code with type errors to customers, whereas with statically type checked code, it is not.

- Dynamically typed code incurs extra overhead. A language based on dynamic type-checking is destinate mainly to be interpreted (so very slow), and no to be compiled. The compilers based on dynamic type-checking are very complex because they must store type informations and manage memory w.r.t. to type informations.

7. **Oddness of language constructs.** The extensible records allowed by the subtyping introduce some odd "facilities" when combined with process and function declarations. For example, whenever the value parameter of a process is an extensible record (i.e., a record ended by '**etc**'), this processes can be invoked by adding more parameters than those initially declared by its profile. It is difficult to claim the usefulness of a such capability, but it is clear that it goes against elementary software methodologies.

   Another example is possibility of declaring unomogenous lists (lists with elements of type '**any**') instead of overloaded constructors and rich term syntax which are simpler to implement and to type.

8. **Oddness in semantics.** Although the subtyping was introduced also for semantics purposes, it induces some nasty problems. For example, due to subtyping, the sequential composition ($B_1 ; B_2$) semantics does not allow associativity if the second behaviour (B_2) is a non-terminating one ('**exit** (**none**') type).

   It is worth noticing that even for experts is difficult to capture all the powerful of a such semantics.

PROPOSED CHANGE: The built-in subtyping must be replaced by genericity and user defined subtyping on abstract data types.

ITEM: **RSI-3 Overloading**

CLAUSE: General

QUALIFIER: Major technical

RATIONALE: Although it is mentioned in the tutorial part of the CD (page 13) that overloading is treated by the module system, the proposed E-LOTOS language cannot support overloading (which already exists in standard LOTOS). We advocate here the main arguments in favor of overloading:

1. Overloading is primary intended for notation convenience. It is not obvious that removing overloading from E-LOTOS is the right way to obtain "a more user-friendly notation for datatype description" (goal defined in the scope of the New Work Item).

2. In absence of overloading of operators, the predefined operators '+', '-', etc. must have different name for each type. The alternative solution of giving the type '$(\mathbf{any}, \mathbf{any}) \rightarrow \mathbf{any}$' to such operator in order to support overloading induce inconsistency in the typing of predefined expressions.

3. User-defined, overloaded functions do not prevent type-checking from being done at compile time. Overloading does not add much complexity to static and dynamic semantics, since overloading treatment is usually treated by the static semantics, without impact on dynamic semantics.

4. Well know efficient algorithm exists to perform type-checking and solve overloading.

5. Overloading fits well with the genericity of the actual module system: it allows to instantiate generic module several times without containing user to give new name for operators at each instantiation.

6. Overloading is existing practice in LOTOS. Forbidding overloading in E-LOTOS would raise difficult compatibility issues with the current standard. In such case, an algorithm should be provided to translate existing LOTOS descriptions into ones without overloading.

7. The rich term syntax proposed by Charles Pecheur relies on the existence of overloading.

8. The overloading is compatible with type theory because there exits theorem-provers based on type theory (PVS, ISABELLE) which supports overloading.

PROPOSED CHANGE: The overloading should be introduced instead of subtyping.

ITEM: **RSI-4 Editorial inconsistencies, lacks, or errors**

CLAUSE: General

QUALIFIER: Editorial

RATIONALE: We give below a list of inconsistencies, lacks, or errors of the present document:

1. Section 2.1 / page 12: The '/' function does not have the Div exception declared in its profile, although the Div exception can be raised at the run time.

2. Section 2.1 / page 14: Mention that 'nil' and 'cons' names cannot be reused because the form of subtyping supported does not includes overloading of functions.

3. Section 2.1 / page 15: At this point the '**any**' pattern is not presented.

4. Section 2.1 / page 16: There should be precise if 'string' should be a user-defined type or a predefined type. Moreover, there are no definition of the predefined types of the language.

5. Section 2.1 / page 16: There should be precise the functionality of the Counter process or to be given the 'by default' functionality of processes. After the 'by default' type of gates, add: "Also, the by default functionality of a process is exit (none)."

6. Section 2.1 / page 16: Here is claimed that '**any**' type is used as type of gate which can communicate data of any type instead of '(**etc**)' type.

7. Section 2.1 / page 17: There should be give a formal definition for subtyping of built-in types as integers, floats.

8. Section 2.1 / page 17: Substitute 'communicationor' by 'communication or'.

9. Section 2.1 / page 17: There should be precise the scope of the 'Match' exception raised by the patterns matching.

10. Section 2.1 / pages 18, 29: There should be clarified the impact of having expressions which are not normal forms into the patterns. The main problem is that the expression can raise an exception, which is different from 'Match' exception.

11. Section 2.1 / page 21: The non-deterministic assignment for expressions is not in the grammar. So, the paragraph "Since the is an expression ..." should be removed.

12. Section 2.1 / page 22: Into the imperative version of the partition, guards on pattern matching are used, without being presented.

13. Section 2.1 / page 23: You should precise either the 'Inner' name is generated as a new name for the loop or it is a reserved word.

14. Section 2.1 / page 27: Inconsistent notation; replace 'de $\Rightarrow$ dest' by 'de:dest'.

15. Section 2.1 / page 27: Typos error; replace 'direcions' by 'directions'.

16. Section 2.1 / page 28: Typos error; replace '$Gamma_i$' by 'Γ_i'.

17. Section 2.1 / page 28: Typos error; add a '; ' after `?x:=`**any** time.

18. Section 2.1 / page 30: Typos error; add a ': ' after mid gate.

19. Section 2.1 / page 31: In the paragraph 'The left is continuously ...' replace 'resume' by 'resumes'. To avoid confusion, add 'left' before all 'behaviour' words.

20. Section 2.1 / pages 15–16: The use of the same name for '**any**' type identifier (in subtyping) and '**any**' pattern disposes to confusions.

21. Section 3.2.1 / page 46: There are some reserved words which do not appear on the list given, and some other which are not used. For example:

infix	**var**	`( )`	`{ }`	`,`	`.`
`;`	`:`	`|`	`?`	`!`	`=>`
`=`	`<>`	`:=`	`[ ]`	`->`	`#`
`>`					

    The otherwise keyword is not used.

    It hould be specified that the reserved words cannot be used as identifiers (except maybe `=`).

22. Section 3.2.3 / page 47: There is no definition for special constant non-terminal class SCon. An additional annex should precise the initial basic of the language (its static semantics and its dynamic semantics).

23. Section 3.3 / pages 48–55: There are a lot of ambiguities in the concrete grammar, due to the lack of precedence between the operators. For example, the expression P`:=`E`;`E can be parsed in two ways `(`P`:=`E`);`E or P`:=(`E`;`E`)`.

    The association rules between binary operators are not specified.

24. Section 3.3 / pages 48–55: The `(`B`)` and `(`E`)` clauses do not appear in the grammar; they are useful to give explicitly the precedence between operators.

25. Section 3.3 / pages 50–51: The actual syntax (`(`P`:=`E`)`) introduce some problems due to the exhaustiveness of the pattern P. To avoid such problems, the classical form of the assignment V`:=`E seems to be appropiate.

26. Section 3.3 / page 50: The grammar given for 'rename' operator is not LALR1. There is no possibility to make difference between a gate renaming or a signal renaming looking to the first token. As suggested in the semantics chapter, '**gate**' and '**signal**' keywords may be added before a gate and a signal identifier, respectively.

27. Section 3.3 / pages 50–51: The grammar given for 'if-then-else' construct does not have an '**elsif**' clause. This will conduce at a cumbersome overlapping of 'if-then-else' constructs.

28. Section 4.2.1 and 4.10.23/ pages 58 and 92: The language syntax considered in this chapter does not correspond with those given in Section 3.3. Examples: declaration of formal parameters for functions and processes, hide construct.

29. Section 4.10.9 / page 79: Although in section 4.10.1 the dynamic semantics use labels of form μ (RN), the notation used here is μ (N).

30. Section 4.10.17 / page 85: In the first rule of static semantics, the RT and RT' have to be disjoint, w.r.t. "write-one" politics; add the guard [RT, RT' have disjoint fields].

    The rule of timed dynamic semantics have to be:

$$\frac{\forall N\,.\,((\mathcal{E} \vdash N \Rightarrow \textbf{any} \text{ and } \mathcal{E} \vdash (P \Rightarrow N) \Rightarrow (RN)) \text{ implies } \mathcal{E} \vdash B[RN] \stackrel{\epsilon(d+d')}{\longrightarrow} B')}{\mathcal{E} \vdash \textbf{choice } P \textbf{ after}(d) \texttt{ [] } B \stackrel{\epsilon(d')}{\longrightarrow} \textbf{choice } P \textbf{ after}(d+d') \texttt{ [] } B'}[0 < d']$$

31. Section 4.10.18 / pages 87–88: The second rule of the static semantics have to have '**exit**(RT) ' in place of '**guarded**(RT) ' because how long the behaviour B is not guarded, and the **exit** action is not trapped, the all behaviour can do an unguarded **exit**.

    In the conclusion of the first rule of the untimed dynamic semantics, the last B have to be B'.

32. Section 4.10.19 / page 89: The N identifier is also used for normal forms.

    The static semantics of vectors of gates $\vec{G_i}$ is

$$\mathcal{C} \vdash \vec{G_i} \Rightarrow \textbf{gate}(\vec{RT_i})$$

33. Section 4.10.20 / page 90: The second premise of the second rule of the static semantics composes (by mistake) RT with the context $\mathcal{C}$.

    In the untimed dynamic semantics the RN given by the pattern matching is not distinguished from the RN given in $\mu(RN)$.

34. Section 4.10.24 / pages 93–94: The $argv pattern and value are not presented.

35. Section 4.10.26 / page 96: The static semantics given for 'loop' type check the following example:

$$?x\texttt{:=}3\texttt{;}\textbf{loop forever } x\texttt{:}\textbf{bool in } x\texttt{:=}x < 4 \textbf{ endloop}$$

    However, at the run time a type error will be raised (for the second iteration). For this, add a fourth premises to the rule:

$$C; RT_1, RT_2 \vdash B_2 \Rightarrow \textbf{exit}(RT_1, RT_2)$$

    This version will reject cases of run-time type error, but will accept expressions as:

$$?x\texttt{:=}3\texttt{;}\textbf{loop forever } x\texttt{:}\textbf{bool in } x\texttt{:=}(x = x) \textbf{ endloop}$$

36. Section 4.10.32 / page 98: The default clause of the 'if-then-else' construct is **exit**. This will constraint the user to describe guarded behaviours of LOTOS

$$\texttt{[}E\texttt{]} \rightarrow B$$

    by explicitly specifying the else part:

$$\textbf{if } E \textbf{ then } B \textbf{ else stop endif}$$

37. Section 4.12 / pages 102–104: The syntactic category of the record expressions is not entirely a syntactic sugar (as mentioned): for disjoint union record expressions you give a static semantics translation.

38. Section 4.13.5 / page 105: In the the three rules given for the untimed dynamic semantics the RE has been replaced by E.

39. Section 4.15 / pages 106–107: This syntactic category overlaps the NP and the LV categories. Moreover, it does not appear in the list of non-terminals given at page 45.

40. Section 2.2 / page 36: The note on the Router example does not explain the presence of the pervasive modules as Natural.

41. Section 2.2.1.2 / page 38: There is a duplicate page 39 and the page 121 is not in the document.

42. Section 2.2.1.3 / page 40: The A1 module allows access only to S, x, and y.

43. Section 2.2.1.4 / pages 40–41: Interface and module expressions are imported by the clauses 'import'.

    Since Natural is a pervasive module do you have to import it in the NatMonoid example?

    Add a note saying that importation is made using a union with matching of common names and not a disjoint union. So the multiple importation is allowed.

44. Section 2.2.1.5 / page 42: A formal semantics for equation typing and their validity (verification semantics) have to be given.

45. Section 5.1.2.3 / page 110: It is not specified the 'matches' relation for the contexts of processes.

46. Section 5.2.2 / page 110: The type of the behavioural part of a specification is not specified.

47. Sections 5.3.8. and 5.4.4 / pages 113 and 114: It is not clear how mutually recursive type, function, and processes declarations can be supported.

48. Section 5.4.3 / page 114: The type of the value declaration could be only S; replace S by T.

49. Section 5.7.2 / page 117: The element to be renamed has to be already into the context $\mathcal{C}$; add the premise $\mathcal{C} \Rightarrow S \equiv T$ at the static semantics rule.

50. Section 5.8.1 / page 118: The context resulting from the static semantics evaluation of RME is $\mathcal{B}$.