From LOTOS to LNT

Hubert Garavel[1,2,3](✉), Frédéric Lang[1,2,3], and Wendelin Serwe[1,2,3]

[1] INRIA, Grenoble, France
{hubert.garavel,frederic.lang,wendelin.serwe}@inria.fr
[2] Univ. Grenoble Alpes, LIG, 38000 Grenoble, France
[3] CNRS, LIG, 38000 Grenoble, France
http://convecs.inria.fr

Abstract. We revisit the early publications of Ed Brinksma devoted, on the one hand, to the definition of the formal description technique LOTOS (ISO International Standard 8807:1989) for specifying communication protocols and distributed systems, and, on the other hand, to two proposals (Extended LOTOS and Modular LOTOS) for making LOTOS a simpler and more expressive language. We examine how this scientific agenda has been dealt with during the last decades. We review the successive enhancements of LOTOS that led to the definition of three languages: E-LOTOS (ISO International Standard 15437:2001), then LOTOS NT, and finally LNT. We present the software implementations (compilers and translators) developed for these new languages and report about their use in various application domains.

Keywords: Abstract data type · Algebraic specification · Concurrency theory · E-LOTOS · Formal description technique · Formal method · Formal specification · LOTOS · LNT · Process algebra · Process calculus · Specification language

1 Introduction

The present article was written in honor of Ed Brinksma and included in a collective *Festschrift* book offered to him at the occasion of his 60th birthday.

The first part of Ed Brinksma's research career has been devoted to the design of formal methods for the specification of communication protocols and distributed systems, the LOTOS language being the common theme and vital lead for the scientific contributions. This first part approximately extends over twelve years, between 1984 (as dated by the conference article [14]) and 1995 (as dated by the book chapter [9]). It was directly succeeded, with some chronological overlap, by a second part centered on conformance testing for protocols (with a first paper [16] published in 1991) and a third part centered on real time and performance evaluation (with early papers, e.g., [15] published in 1995).

In the present article, we focus on this first part, in which we distinguish two different threads of work: (i) the definition of the LOTOS language, which culminated with its adoption by ISO (International Standard 8807:1989) and (ii)

J.-P. Katoen et al. (Eds.): Brinksma Festschrift, LNCS 10500, pp. 3–26, 2017.
DOI: 10.1007/978-3-319-68270-9_1

the elaboration of two proposals for enhancing LOTOS, by introducing valuable features not present in the standard, either because they were not ready on time when it was adopted or because they did not reach international consensus.

The present article is organized as follows. Section 2 recalls the contributions of Ed Brinksma to the definition of LOTOS and gives a brief account of the impact of this language in academia and industry. The two next sections review two early languages proposed by Ed Brinksma for enhancing LOTOS, namely Extended LOTOS (Sect. 3) and Modular LOTOS (Sect. 4). The three next sections present three more recent languages that, between 1993 and now, have been proposed to supersede LOTOS, namely E-LOTOS (Sect. 5), LOTOS NT (Sect. 6), and LNT (Sect. 7), with some discussion about the actual impact of these languages. Finally, Sect. 8 gives a few concluding remarks.

2 LOTOS

Among all publications of Ed Brinksma related to the definition of LOTOS, we highlight three key contributions, each of a different nature and scope:

– Obviously, the ISO Draft International Standard defining LOTOS [58] occupies a place of choice. Even if earlier drafts of LOTOS had circulated before (e.g., Ed Brinksma's first tutorial on LOTOS [10] given in 1985) and even if experiments with LOTOS had already been done at some universities (e.g., the model-checking verification of protocols in 1986 [33,34]), this Draft International Standard published in 1987 was the first complete, coherent definition of LOTOS made available to the international community. Two years after, this document reached its final status by being approved as the ISO International Standard 8807:1989 [60].
  The definition of LOTOS was a collective achievement done within an ISO committee (project 97.21.20.2) under the leadership of Ed Brinksma, who was the editor in charge of producing the standard. Tommaso Bolognesi, Günter Karjoth, Luigi Logrippo, Jan de Meer, Elie Najm, Juan Quemada, Pippo Scollo, Alaister Tocher, Jan Tretmans, and Chris Vissers participated, among others, in this committee.
  The resulting LOTOS language was an audacious combination of the most recent innovations in formal methods at that time. To describe and manipulate data structures, the LOTOS committee selected abstract data types—more precisely, a dialect of the algebraic language ACT ONE [25,26,80]. To describe the behaviour of concurrent processes, the committee retained the key ideas of process algebra, blending into a single language the best features of several calculi, namely CCS [82], TCSP [17], and Circal [81]; LOTOS also brought original ideas, such as its "disable" operator, which models nondeterministic disruption (e.g., crashes and failures), and its "enable" operator, which allows value-passing sequential continuation after the termination of a group of parallel processes.
  The definition of LOTOS provided in the ISO standard was fully formal, much in line with the longstanding Dutch tradition of computer-language

definitions. The syntax was given as a BNF grammar; the static semantics was specified as a set of mathematical constraints and functions defined by induction over syntactic constructs; the semantics of data types was expressed as a many-sorted term algebra obtained by quotienting the algebra generated by a derivation system; finally the behavioural semantics of processes was defined operationally using a set of structured operational semantics rules. This formal definition was followed by annexes providing informal explanations and complementary information.

- Jointly written with Tommaso Bolognesi, Ed Brinskma's tutorial on LOTOS [6] is also a highly cited publication. Written in a lively style and illustrated with a wealth of examples, this tutorial targets the end users of LOTOS. It is orthogonal and complementary to the (somewhat dry) ISO standard definition, primarily oriented towards language implementers and semanticists.
- Another insightful contribution is Ed Brinskma's 1989 paper on constraint-oriented specification [12]. It is well-known that the decomposition of a computer system into concurrent/parallel tasks may take two forms: it is either *physical* if the decomposition closely reflects the actual distribution of tasks over processors, or *logical* otherwise, if the decomposition is rather intended to provide the system with a modular structure that does not necessarily correspond to its actual topology. Ed Brinksma develops the latter approach in the framework of the LOTOS multiway rendezvous, which enables two or more processes to synchronize, negotiate, and exchange data values during one atomic event. The paper formulates the fundamental intuition of *parallel composition as conjunction*, meaning that the multiway rendezvous achieves the logical conjunction of all the individual constraints expressed by a set of processes running concurrently. This idea enables a certain degree of "declarative" programming (namely, constraint solving) to be introduced in the framework of a fundamentally "operational" (i.e., constructive, imperative) language such as LOTOS. The usefulness of the approach is demonstrated on realistic examples of communication protocols [12], but it is also relevant to other application domains, e.g., hardware circuits ([41] shows how the complex arbitration protocol of the SCSI-2 bus can be concisely modelled using an eight-party LOTOS rendezvous) or robotics ([47] illustrates how a software controller for an entire manufacturing plant can be obtained as the parallel composition of many simple controllers, one for each device or degree of freedom of a device in the plant).

Retrospectively, the international effort invested in LOTOS was successful in several respects:

- Although LOTOS is a committee-designed language based on two very different concepts (algebraic data types and process calculi), it achieves a suitable compromise and a fair integration between its various elements. All its language constructs (perhaps with the exception of the **choice** and **par** operators on gate lists) derive from concrete needs and are useful in practice.
- LOTOS is clearly more abstract and higher level than the two other standardized languages it was competing with (namely, Estelle [59] and SDL [18]), and

proved that a specification language could be formal and executable at the same time.
- The design of LOTOS made it clear that process calculi were not only mathematical notations for studying concurrency theory, but that they could be turned into computer languages used to model real-life systems. LOTOS was indeed the first process calculus in which large specifications of complex systems (e.g., protocols and services of OSI and ISDN networks) were produced. Later, it was shown that the high abstraction level of LOTOS makes it also suitable to other application domains, e.g., multiprocessor architectures and asynchronous circuits.
- The LOTOS community put strong emphasis on software tools, often in the framework of European projects such as SEDOS, LOTOSphere, SPECS, EUCALYPTUS-1 and -2, etc. Today, most of these tools are no longer available, but the CADP toolbox[1] [45] is still actively maintained. Also, many ideas present in early LOTOS tools would certainly benefit from modern developments in symbolic execution and verification technology.

On the negative side, one can point out two main shortcomings of LOTOS:

- Despite its status of international standard, LOTOS did not manage to unite the academic community working on process calculi. Not only the preexisting algebras/calculi ACP, CCS, and CSP remained, but new languages appeared, e.g., μCRL. This resulted in fragmented efforts and a lack of critical mass that became apparent in the mid-90s.
- LOTOS also failed to gain wide industrial acceptance, mostly due to its so-called "steep learning curve". Because it is an abstract, expressive, and flexible language based on concepts absent from mainstream languages, LOTOS is best used by high-level experts rather than average software programmers: this is unfortunately a fatal flaw as far as dissemination is concerned.

3 Extended LOTOS

As soon as the definition of LOTOS was frozen as an ISO standard, it appeared that the language was not fully satisfactory and that some of its features could be redesigned in a better way. Ed Brinksma's role as the editor of the LOTOS standard did not prevent him from suggesting enhancements to LOTOS.

His first contribution in this respect is his PhD thesis [11], defended in 1988, which proposes a language named "Extended LOTOS" that significantly differs from LOTOS. Concerning data specifications, Extended LOTOS keeps the abstract data types of LOTOS, but adds better support for modules. Concerning behavioural specifications (namely, concurrent processes), Extended LOTOS brings deeper changes:

- It introduces a notion of *action product* inspired from SCCS [83], whereas LOTOS only has simple actions.

[1] http://cadp.inria.fr.

- Extended LOTOS attempts at unifying in one single operator both forms of sequential composition (action prefix and "enable") that exist in LOTOS.
- Extended LOTOS breaks with the algebraic style of LOTOS and other process calculi by replacing unary and binary operators with n-ary constructs having a fully bracketed syntax, e.g., "**sel** B_1 [] B_2 [] ... [] B_n **endsel**" for nondeterministic choice or "**par** B_1 || B_2 || ... || B_n **endpar**" for parallel composition.
- Extended LOTOS proposes other desirable features, among which a **par** operator ranging over a finite domain of values.

Although Extended LOTOS has never been actually implemented, these ideas had the merit to point out the main shortcomings of LOTOS and made it clear that the language, despite its status of international standard, still deserved major enhancements.

4 Modular LOTOS

Published three years later, a deliverable (edited by Ed Brinksma) of the LOTOSphere project [13] adopts a point of view orthogonal to that of Extended LOTOS: leaving aside all ideas for improving the behaviour part of LOTOS, this deliverable focuses on enhancements to the data part of LOTOS, in which usability problems have been identified as most crucial, and proposes a new language called "Modular LOTOS", two synthetic presentations of which can also be found in [9,90]. Modular LOTOS suggests the following enhancements:

- Distinction between *constructors* and *functions*, whereas LOTOS made no difference between these two forms of operations;
- Introduction of *partial functions*, whereas LOTOS only allowed totally defined operations;
- Support for *built-in types* (e.g., natural numbers, integer numbers, strings) and *generic data structures* (e.g., lists, sets, arrays, etc.);
- Introduction of *modules* gathering data and/or behaviour definitions, namely, types, constructors, functions, and processes;
- Introduction of module interfaces (called *descriptions*) that can be used to hide certain definitions contained in modules;
- Introduction of *renaming* to avoid name clashes between different modules;
- Support for *generic modules* parameterized by descriptions.

To our knowledge, Modular LOTOS has never been implemented, although key ideas (namely, distinction between constructors and functions, partial functions, and splitting of large LOTOS specifications into multiple files) were already supported in the CÆSAR.ADT compiler for LOTOS [35]. At this point, Ed Brinskma shifted his research interests to other topics, but the LOTOS reform movement he had initiated expanded rapidly.

5 E-LOTOS

Between 1993 and 2001, an ISO committee gathered under the lead of Juan Quemada to revise the LOTOS standard. Arnaud Février, Hubert Garavel, Alan Jeffrey, Guy Leduc, Luc Léonard, Luigi Logrippo, José Mañas, Elie Najm, Mihaela Sighireanu, and Jacques Sincennes participated in this committee as regular contributors, with the help of more than twenty occasional contributors [89].

At the beginning, the proposed changes were modest, trying to repair rather than replace LOTOS; as time passed, it appeared that more radical enhancements were desirable. This work eventually resulted in a new language named E-LOTOS (for "Enhanced LOTOS") approved as ISO/IEC International Standard 15437:2001 [61]. Tutorials on E-LOTOS can be found in [101], [56,57,70]. Compared to LOTOS, E-LOTOS brings deep changes that aim at greater expressiveness and/or better user-friendliness:

– Concerning the data types, E-LOTOS goes far beyond the ideas suggested for Modular LOTOS. Rather than enhancing ACT-ONE, E-LOTOS removes it, replacing abstract data types with a functional language—an approach also explored in [5], which proposes a concurrent language combining a process calculus (CCS) and a functional language (ML). E-LOTOS goes even further by giving its functional language an imperative flavour: in particular, E-LOTOS variables can be assigned and E-LOTOS functions can have output (i.e., call by result) parameters to return multiple results, which, in conventional functional languages, is usually done by returning tuple values.
– E-LOTOS data types can be records (with named or unnamed fields) or (possibly recursive) types defined by a list of constructors. E-LOTOS also provides predefined types (Booleans, naturals, integers, rationals, floating-point reals, characters, and strings) and abbreviations for declaring enumerated types, records, sets, and lists.
– Contrary to LOTOS, in which the data and behaviour parts are two entirely different sub-languages, E-LOTOS tries to unify functions and processes; functions can be seen as particular cases of processes that only do local calculations before terminating, do not perform any observable or invisible action, and do not let time elapse. Consequently, functions and processes share a number of common constructs, among which: variable assignments, **if-then-else** conditionals, **case** with pattern matching, **while** loops, **for** loops, etc.
– In both its data and behaviour parts, E-LOTOS introduces a unique sequential composition operator, which unifies the action-prefix and "enable" operators present in the behaviour part of LOTOS.
– E-LOTOS provides support for exception handling. In the data part of E-LOTOS, exceptions bring a convenient solution to the need for partial functions. In the behaviour part, exceptions allow some involved communication protocols to be described compositionally—see [49] for an advocacy paper on exceptions in process calculi.
– Gates (i.e., communication ports) are explicitly typed in E-LOTOS, whereas they are untyped in LOTOS—see [36] for an introduction to gate typing,

which leads to more readable specifications, detects communication mismatches at compile time rather than at run time, and provides a simple solution to the "structured events" issue in the constraint-oriented style.
- To express *quantitative time* aspects, the behaviour part of E-LOTOS allows to specify constraints on the duration of actions and/or the instant(s) at which they may occur. Such features are required to describe isochronous protocols and real-time systems precisely, and many timed extensions of LOTOS have been proposed, e.g., ET-LOTOS [71,72] and RT-LOTOS [22].
- The behaviour part of E-LOTOS introduces a n-ary parallel operator [51] that generalizes the three binary parallel composition operators of LOTOS. This new operator is easier to use, more readable, and enables m-among-n synchronization (in particular, the 2-among-n synchronization of CCS).
- The behaviour part of E-LOTOS also introduces new operators, such as *rename* (which allows to change the name of observable actions and exceptions, to merge or split gates, and to add or remove offers from actions) and *suspend/resume* (which generalizes the "disable" operator of LOTOS by allowing resumable interrupts to be modelled).
- Finally, E-LOTOS provides *modules* that may contain types, functions, and/or processes. Modules can be imported and exported; they have *interfaces* for information hiding and can be *generic*.

Due to its new features resulting from multiple, sometimes conflicting influences, and despite the unification between functions and processes, E-LOTOS is a complex language, with an impressive number of semantic rules. The E-LOTOS standard has 120 pages (+80 pages of annexes), while the LOTOS standard has only 70 pages (+70 pages of annexes). It is therefore unclear whether E-LOTOS brings a satisfactory answer to the "steep learning curve" issue with LOTOS.

This probably explains why E-LOTOS only had a marginal impact in practice. Very few case studies have been done using E-LOTOS; one can mention [96,99] (which compares LOTOS and E-LOTOS on a common example), [21,24,92,93]. To our knowledge, E-LOTOS has never been implemented in software tools (except perhaps [24] or [74]) nor taught in university classes.

In some sense, the shift from LOTOS to E-LOTOS is reminiscent of the shift from Algol 60 to Algol 68: a simple, elegant, yet limited language was replaced by a larger, more expressive, formally defined language, which, because of its growth in complexity, failed to build a sufficient momentum of interest among its potential users.

6 LOTOS NT

6.1 Design of LOTOS NT

In 1997, when it became manifest that E-LOTOS was getting too large and too complex, INRIA Grenoble started investigating a fallback solution. This led to the design of LOTOS NT (where "NT" stands for "New Technology"), a simplified dialect of E-LOTOS that could be feasibly implemented and provide an actual replacement solution for LOTOS.

It was decided to not introduce in LOTOS NT some questionable features that significantly contribute to the complexity of E-LOTOS, among which: type synonyms, anonymous tuples (i.e., the possibility, borrowed from ML, that any list of values put between parentheses creates a new value having a valid, yet undeclared tuple type), extensible records, type equality relation based on structure equivalence (rather than name equivalence), subtyping relation based on record subtyping, etc.

For the same reasons, two features present in E-LOTOS but absent from LOTOS, the suspend-resume operator and the support for quantitative time, were not introduced in LOTOS NT, as it was felt that the potential applications of such features were already covered by competing formalisms such as timed automata [2] and were not worth the effort/impact ratio.

The formal definition of LOTOS NT (syntax, static semantics, and dynamic semantics) was given in [94]. Rationale for the design of LOTOS NT (and of E-LOTOS as well, since LOTOS NT influenced the latest evolutions of E-LOTOS) can be found in [50].

6.2 Implementation of LOTOS NT

To implement this language, a compiler named TRAIAN[2] [95] has been developed at INRIA Grenoble since 1997. It is built using the SYNTAX [7] and FNC-2 [63] compiler-generation tools designed at INRIA Rocquencourt. Unfortunately, FNC-2 ceased to be maintained in 1999, which prevented TRAIAN from being completed; as a consequence, TRAIAN only handles the data part of LOTOS NT (i.e., types and functions) but not the behaviour part (i.e., processes and channels).

As it is, TRAIAN performs lexical analysis, syntactic analysis, abstract syntax tree construction, static semantics analysis of LOTOS NT data specifications, and translates these into C programs, which can in turn be compiled and executed. TRAIAN has been regularly maintained and enhanced: ten releases have been issued since 1998, the latest version of TRAIAN (dated 2016) containing 55,500 lines of FNC-2 and C code.

6.3 Applications of LOTOS NT

Although TRAIAN only supports a fragment of LOTOS NT, it has useful applications in compiler construction. Our approach [44] consists in using the SYNTAX compiler generator for the lexical and syntactic analyses, together with LOTOS NT for semantical aspects, in particular the definition, construction, and traversals of abstract trees. Some involved parts of the compiler can be written directly in C if necessary, but most of the compiler is usually written in LOTOS NT, which is then translated into C code by TRAIAN.

[2] http://vasy.inria.fr/traian.

The combined use of SYNTAX, LOTOS NT, and TRAIAN proves to be satisfactory, as regards both the rapidity of development and the quality of resulting compilers. So far, twelve compilers have been developed at INRIA Grenoble using this approach: AAL [75], ATLANTIF [97], CHP2LOTOS [46], CTRL2BLK [76], EVALUATOR 4.0 [79], EXP.OPEN 2.0 [67], FSP2LOTOS [68], GRL2LNT [62], LNT2LOTOS [19], NTIF [43], PIC2LNT [77], and SVL [23,42,66].

7 LNT

7.1 Design of LNT

Because of the limitations of TRAIAN, LOTOS NT does not provide a replacement solution for LOTOS. The need for a better language based on process calculi remains [38,39], even if all prior attempts have failed to provide a usable solution.

In 2005, a new opportunity was found to progress this agenda: the Bull company was interested in using the CADP toolbox to formally verify multiprocessor architectures, but was reluctant to use LOTOS as a modelling language, mostly due to the verbosity of the LOTOS data part. To ease the writing of large specifications by Bull, still using the existing CADP tools, INRIA Grenoble undertook the development of a translator to convert LOTOS NT data types and functions into LOTOS ones. This made it possible to produce specifications combining a data part written in LOTOS NT (more concise and less error-prone than LOTOS) and a behaviour part written in LOTOS. The translator converted such composite specifications into plain LOTOS ones, which then could be analyzed by the CADP tools.

A first version of this translator was delivered to Bull in July 2005. Since then, the translator has been constantly improved and extended to handle new LOTOS NT features. In 2007, support for the behaviour part of LOTOS NT was added; this progressively removed the need for composite specifications, as it became possible to write entire specifications in LOTOS NT, with no LOTOS code at all.

Due to the rapid evolution of this translator, its input language gradually diverged from the original LOTOS NT implemented in TRAIAN, which remained quite stable in comparison. To avoid ambiguities, it was decided in 2014 to give this input language a new name ("LNT"), while reserving the name "LOTOS NT" for the language accepted by TRAIAN—such a distinction was not made in papers published before Spring 2014, in which LOTOS NT and LNT were used as synonyms.

The current definition of LNT is given in [19]. In a nutshell, LNT combines, in a single language with an Ada-like syntax designed to favour readability, selected features borrowed from imperative languages, functional languages, and value-passing process calculi:

– An LNT specification is a set of *modules*, each of which may import other modules and define *types*, *functions*, *channels*, and/or *processes*.

- A *type* is either predefined (namely, **bool**, **nat**, **int**, **real**, **char**, and **string**), defined by specifying the *free constructors* that generate its domain of values, or defined using the *type combinators* **array**, **list**, **range**, **set**, **sorted list**, **sorted set**, and **where** (the latter enabling predicate subtyping).
- A *function* is either predefined (namely, logical, arithmetical, and relational operations on predefined types), automatically generated for some user-defined type (such as free constructors, but also equality, order relations, field accessors and selectors, etc., which are generated if the user requests them), or have a handwritten definition provided by the user.
- A *channel* is a gate type that, following the ideas of [36], specifies the types of values to be sent or received during interactions on a given gate. There exist two special channels: **none**, which expresses that no value can be sent or received (this is useful for pure synchronization and exceptions), and **any**, which permits all values to be sent or received (this allows gates to be "untyped", as in LOTOS, thus ensuring backward compatibility).
- A *process* is a program fragment that, as in LOTOS and other process calculi, executes and communicates with its environment by sending and/or receiving values on a set of gates.

Globally, LNT has four different kinds of routines, of increasing complexity:

- A *constructor* has only **in** parameters (call by value), no explicit definition, and does not raise exceptions.
- A *pure function* has only **in** parameters, an implicit or explicit definition, and may raise exceptions if needed (this provides for partially-defined functions).
- A *procedural function* (or *procedure*, for short) may have **in**, **out** (call by result), or **in-out** (call by value-result) parameters; unlike constructors and pure functions, it does not necessarily return a result; it usually has an explicit definition and may raise exceptions.
- A *process* may also have **in**, **out**, or **in-out** parameters; it has an explicit definition, may raise exceptions [49], and interacts with its environment by means of gates. The key difference between processes and other routines is that the execution of processes can be nondeterministic and let time elapse (the execution semantics is that of process calculi and labelled transition systems) whereas the three former kinds of routines execute deterministically and atomically (the execution semantics is that of functional languages).

LNT possesses three main concepts for denoting computation:

- An *expression* corresponds to the usual notion of expression in imperative programming languages. It is an algebraic term built using constants, variables, and calls to constructors and pure functions. The evaluation of each expression is deterministic (it always returns the same result or raises the same exception), atomic (it is expected to terminate and take a negligible amount of time), and free from side effects (it does not modify variables).
- An *instruction* corresponds to the usual notion of statement in imperative programming languages. Instructions serve to explicitly define the bodies of

LNT functions. Basic instructions include: **null** (which does nothing), assignment to a variable or an array element, **return** of a function result, **raise** of an exception, procedure call, **assert**, etc. Instructions can be combined using structured-programming constructs, such as sequential composition, **if-then-else** conditionals, **case** with pattern matching, **for** and **while** loops, loops with **break** clauses, and declarations of variables with a limited scope. Because instructions manipulate and modify a store, the semantics of LNT relies on static analysis to prohibit all situations where uninitialized variables could be used; this way, instructions have an imperative-programming syntax and a functional-programming semantics. Like the evaluation of expressions, the execution of instructions is deterministic and atomic.

– A *behaviour* is the LNT equivalent of a LOTOS "behaviour expression". Behaviours serve to define the bodies of LNT processes. Behaviours can be seen as a superset of instructions since they contain all instructions (except **return**) but also include additional constructs specific to process calculi: **stop** (deadlock), communication on a gate (possibly with sending and/or receiving values), assignment of a non-deterministic value to a variable, process call, forever loop without **break** clause, non-deterministic choice (which is n-ary, rather than binary), parallel composition (which is n-ary and *graphical* [51], i.e., explicitly describes the communications/synchronizations between concurrent behaviours), gate hiding, and disruption (i.e., the "disable" operators of LOTOS). Unlike instructions, the execution of behaviours is nondeterministic, non-atomic, and may never terminate.

In the design of LNT, one main attainment is the integration in a single language of two very different models of computations: imperative/functional languages and process calculi. This was not the case with LOTOS, nor with its competitors Estelle and SDL, both of which clumsily amalgamate state machines with another formalism for data computation. Such a unification, which had been tried without success in E-LOTOS, is now effective in LNT. A key issue was the status of sequential composition [40], which led to question and discard some well-established habits of process calculi, especially the action-prefix operator of CCS and LOTOS, and the use of dynamic semantics in place of static semantics.

7.2 Implementation of LNT

Contrary to the four aforementioned languages (Extended-LOTOS, Modular LOTOS, E-LOTOS, and LOTOS NT), the definition and implementation of which were planned as two successive steps (the latter being never undertaken or never completed), LNT was designed in a radically different way, using an "agile" approach. Every new language feature was first implemented and assessed on a growing base of non-regression tests before being adopted for LNT.

Initially designed as a standalone tool, the translator from LNT to LOTOS became an integral part of the CADP toolbox in 2010. Actually, this translator is not one single tool, but comprises three complementary tools:

- LPP[3] ("LNT PreProcessor") is a small translator (2000 lines of C and Lex code) that expands the user-friendly LNT notations for literal constants (numbers, characters, strings, etc.) into algebraic terms making use of predefined LOTOS sorts and operations defined in custom libraries.
- LNT2LOTOS[4] is a rather complex translator developed using the aforementioned SYNTAX/TRAIAN technology (3800 lines of SYNTAX code, 35,500 lines of LOTOS NT, and 2900 lines of C code). LNT2LOTOS translates an LNT specification into LOTOS code, possibly augmented with some little C code fragments.
- LNT.OPEN[5] is a small utility (400 lines of shell script) that provides a top-level entry point for processing LNT specifications with the CADP tools and, more specifically, with the CÆSAR.ADT [35] and CÆSAR [48] compilers for LOTOS, and the OPEN/CÆSAR framework [37] for simulation, verification, and testing. Taking as input an LNT specification and an OPEN/CÆSAR application program, LNT.OPEN first translates (the various modules composing) the LNT specification into LOTOS by calling LPP and LNT2LOTOS, then compiles the generated LOTOS specification by calling CÆSAR.ADT and CÆSAR, and finally invokes the OPEN/CÆSAR application program to explore and analyze the corresponding state space on the fly.

Without exposing in full detail the algorithms implemented in LNT2LOTOS, these are some key principles underlying the translation:

- The main guideline is to keep the translation as simple as possible, so as to swiftly upgrade the translator each time the definition of LNT evolves. Consequently, duplication of work between the translator and the LOTOS compilers is avoided, meaning that the translator does not implement certain static semantics checks at the LNT level if the same checks are later performed at the LOTOS level. In particular, the translator makes no attempt to infer and check the types of LNT value expressions, deferring these tasks to the LOTOS compilers operating on the generated code.
- The channels used for typing LNT gates raise a specific problem because, on the one hand, the LNT2LOTOS translator is not intended to perform type checking and, on the other hand, LNT gates, which are typed, are translated into LOTOS gates, which are untyped, so that type-checking errors at the LNT level cannot be detected at the LOTOS level. To address this problem, LNT2LOTOS generates, for each LNT channel C, one or several overloaded LOTOS constant functions f_C, which take as parameters the expected typed values specified for C and always return true. For each LNT action involving some gate G whose channel is C, a LOTOS Boolean guard is generated, which invokes function f_C with the input or output offers of the action, thus expressing in LOTOS the type-checking constraints arising from the definition of C. If the action is not well-typed at the LNT level, the corresponding

[3] http://cadp.inria.fr/man/lpp.html.
[4] http://cadp.inria.fr/man/lnt2lotos.html.
[5] http://cadp.inria.fr/man/lnt.open.html.

guard will provoke at the LOTOS level a type-checking error at compile time; otherwise, the guard will evaluate to true at run time.

- The LNT2LOTOS translator performs, on LNT functions and processes, static analyses not done at the LOTOS level; for instance, it rejects (or warns about) unused variables, variables used without being assigned before, variables assigned but never used, variables shared between concurrent processes, etc. Such checks are either required by the LNT semantics (see [40] for a discussion) or suitable to ensure that LNT specifications remain as simple as possible, so as to increase readability and efficiency of verification.
- The predefined types of LNT (**bool**, **nat**, etc.) are implemented using base libraries written in LOTOS and C code. The user-defined types (built using free constructors or type combinators) are translated into LOTOS abstract data types (possibly with some additional C code meant for efficiency), together with their associated functions (equality, order relations, field accessors and selectors, etc.).
- Although, in LNT, user-defined functions and processes have the same functional/imperative style and share many constructs (e.g., assignments, **assert**, **raise**, **if-then-else**, pattern-matching **case**, **for** and **while** loops, loops with **break**, etc.), the two algorithms that translate, respectively, these functions and processes into LOTOS are very different, due to the fundamental asymmetry, in the target language, between the data part (based on abstract data types) and the behaviour part (based on process calculi).
- Our algorithm for translating LNT functions generalizes the one proposed in [88], which translates into Horn clauses a small subset of C functions with only integer and list types. Our algorithm translates LNT functions into LOTOS (non-constructor) operations, which are defined using algebraic equations considered as conditional term-rewrite rules. The translation takes advantage of the rewrite strategy implemented in the CÆSAR.ADT compiler, which assumes a decreasing priority between equations. Notice that each LNT function having **out** parameters (call by result) or **in-out** parameters (call by value-result) translates to several LOTOS functions. The translation of certain LNT constructs (**assert**, **case**, and loops) also generates auxiliary LOTOS functions.
- Our algorithm for translating LNT processes takes its roots in our prior works on the translation to LOTOS of three modelling languages for hardware and software systems: the CHP2LOTOS translator [46] for CHP, the FLAC translator [4] for Fiacre, and the FSP2LOTOS translator [68] for FSP. The algorithm is involved, but four main points are worth being highlighted.

  (i) Certain behavioural LNT constructs directly map to equivalent LOTOS ones. For instance, each LNT gate translates to a corresponding LOTOS gate and each LNT process translates to a corresponding LOTOS process. The algorithm benefits from the fact that both LOTOS and LNT have an *action-based* (rather than *state-based*) semantics and share a common semantic model (namely, labelled transition systems). Thanks to action-based semantics, the translation can freely introduce auxiliary LOTOS processes and variables, still preserving the semantic model (which would not be possible with

state-based semantics); in particular, execution traces are identical at the LOTOS and LNT levels, which avoids the usual need for a reverse translation of diagnostics from the target to the source level.
(ii) Certain behavioural LNT constructs are too powerful to be expressed using only the behaviour part of LOTOS; for such constructs, the data part of LOTOS must also be used, by generating auxiliary sorts, operations, and algebraic equations. For instance, the **case** construct present in LNT processes is translated using both the behaviour part of LOTOS (nondeterministic choice and Boolean guards are used to express the selection between the various **case** branches) and the data part of LOTOS (equations, considered as rewrite rules, are used to express pattern matching, which is not supported by the behaviour part of LOTOS).
(iii) An involved part of the algorithm translates the LNT parallel composition operator, which is n-ary, into an algebraic combination of LOTOS parallel composition operators, which are binary. Such a translation does not always succeed, meaning that certain network topologies specified in LNT cannot be expressed in LOTOS [51]; however, we never faced this problem in real-life case studies. Also, it was not possible to introduce in LNT the concept of n-among-m synchronization proposed in [51], because it is not supported in LOTOS; such a limitation is more annoying in practice, e.g., for the specification of Web services, which quite often require 2-among-m synchronization.
(iv) Another involved part of the algorithm translates the LNT sequential composition operator (which is unique, symmetric, atomic, and lets all values of variables assigned on its left-hand side flow implicitly to its right-hand side [40]) into one of the two LOTOS sequential composition operators, either the action-prefix operator (which is asymmetric, atomic, and lets variable values flow implicitly from its left- to its right-hand side) or the "enable" operator (which is symmetric, non-atomic as it generates a τ-transition, and forbids variable values to flow from its left- to its right-hand side except if these variables are explicitly listed in an **accept** clause). Following the principles set for the CHP2LOTOS translator [46], we chose to generate action prefix as much as possible and "enable" only when unavoidable, which produces better LOTOS code at the expense of a more involved translation. To fight state-space explosion and preserve strong equivalence between the LNT and LOTOS specifications, we slightly deviated from LOTOS semantics by adding a special pragma "`(*! atomic *)`" that instructs the LOTOS compiler not to generate a τ-transition when implementing the "enable" operator. There are many other algorithmic subtleties, such as the creation of auxiliary "continuation" processes for those LNT behaviours following loops and conditionals (i.e., **case**, **if**, and **select**), the translation of parallel composition occurring on the left-hand side of sequential composition where each parallel branch computes the values of different variables, the translation of **out** and **in-out** parameters of LNT processes into **exit** results returned by LOTOS processes, the need to respect the strict typing rules set by LOTOS "functionality" constraints, the optimization of tail-recursive process instantiations, etc.

In addition to the above tools, which ultimately translate an LNT specification into a sequential C program, there also exists a compiler named DLC ("Distributed LNT Compiler") [31,32] that translates an LNT specification into a set of C programs executing concurrently and communicating through TCP sockets; to produce such a distributed implementation, the DLC compiler exploits the concurrent architecture defined by the parallel composition operators present in the LNT specification.

7.3 Applications of LNT

The usability of the LNT language gradually increased with the progress of its translator to LOTOS. As of mid-2009, this translator was sufficiently complete and robust to allow a total shift from LOTOS to LNT at INRIA Grenoble, where no LOTOS code has been manually written since then, LNT being now the preferred high-level language for modelling concurrent systems and analyzing them using the CADP tools.

At Grenoble INP and Université Grenoble-Alpes, LNT has also replaced LOTOS to teach master students the fundamentals of concurrency theory. We observed that LNT enables students to better focus on high-level concepts, rather than getting lost in low-level details of LOTOS syntax and static semantics.

The LNT language and its tools have been used for many case studies, at INRIA Grenoble and in other academic or industrial labs as well (we only mention those not affiliated with the authors' institutions):

- *Avionics*: verification of an equipment failure management protocol[6] and of a ground-plane communication protocol[7] [52,98] provided by Airbus;
- *Cloud computing*: verification of self-configuration protocols[8] [27] (Orange Labs), of the Synergy reconfiguration protocol for component-based systems[9] [8], and of dynamic management protocol for cloud applications[10] [1];
- *Distributed algorithms*: verification and performance evaluation of mutual exclusion protocols[11] [78], verification of multiway synchronization protocols[12] [28,30,32], specification and rapid prototyping of Stanford's RAFT distributed consensus algorithm[13] [29,32], and performance evaluation of concurrent data structures[14] [102] (RWTH Aachen, Germany and Chinese Academy of Sciences, Beijing, China);
- *Hardware design*: formal analysis and co-simulation of a dynamic task dispatcher[15] [69] (STMicroelectronics), formal analysis of ARM's ACE cache

[6] http://cadp.inria.fr/case-studies/09-k-failure-management.html.
[7] http://cadp.inria.fr/case-studies/09-h-tftp.html.
[8] http://cadp.inria.fr/case-studies/11-i-selfconfig.html.
[9] http://cadp.inria.fr/case-studies/11-h-synergy.html.
[10] http://cadp.inria.fr/case-studies/13-g-dynamic-management.html.
[11] http://cadp.inria.fr/case-studies/10-f-mutex.html.
[12] http://cadp.inria.fr/case-studies/13-d-multiway.html.
[13] http://cadp.inria.fr/case-studies/15-g-raft.html.
[14] http://cadp.inria.fr/case-studies/16-b-concurrent.html.
[15] http://cadp.inria.fr/case-studies/11-g-dtd.html.

coherency protocol[16] [64,65] (STMicroelectronics), verification and rapid prototyping of an asynchronous model of the Data Encryption Standard[17] [91], verification of a fault-tolerant routing algorithm for a network-on-chip[18] [103] (University of Utah, USA);
- *Human-computer interaction*: specification and validation of graphical user interfaces for a prototype control room of a nuclear power plant[19] [85] and of plastic user interfaces exploiting domain ontologies[20] [20] (Toulouse, France);
- *Industrial systems*: model-based testing of the CANopen field bus and EnergyBus architecture[21] [53] (Saarland University, Germany), formal specification and rapid prototyping of a software controller for a metal processing plant[22] [47].

Another indication of the practical usefulness of LNT is given by its use as a target language in a growing number of translators, which implement various languages by translating them to LNT. Indeed, LNT suitably replaces LOTOS for automatically-generated code as well as for handwritten code, since the translation to LNT is much easier than the translation to LOTOS, and because it is now preferable to let the LNT2LOTOS translator take in charge all algorithmic subtleties required to produce valid and efficient LOTOS code. We are aware of the following tools (again, we do not mention the authors' institutions):

- The BPMN2Py/Py2LNT translators[23] [55,86] for analyzing choreographies of Web services specified in WS-CDL (Université de Nantes, France);
- The CMT translator [73] for the BPEL/WSDL specification languages for Web services (Tsinghua University, Beijing, China and MIT, Cambridge, MA, USA), and another, more complete algorithm for translating BPEL/WDSL/Xpath/XML Schema to LNT [98];
- The DFTCalc tool[24] [3,54] for Dynamic Fault Trees (University of Twente, The Netherlands);
- The EB^{3}2LNT translator[25] [100] for the EB^3 specification language for information systems (Université Paris Est, France);
- The GRL2LNT translator[26] [62] for the GRL specification language for GALS (*Globally Asynchronous Locally Synchronous*) systems;
- The OCARINA tool[27] [84] for the AADL architecture description language (ISAE, Toulouse, France and University of Sfax, Tunisia);

[16] http://cadp.inria.fr/case-studies/13-e-ace.html.
[17] http://cadp.inria.fr/case-studies/15-f-des.html.
[18] http://cadp.inria.fr/case-studies/13-f-utahnoc.html.
[19] http://cadp.inria.fr/case-studies/14-d-hmi.html.
[20] http://cadp.inria.fr/case-studies/15-d-plastic-user-interfaces.html.
[21] http://cadp.inria.fr/case-studies/14-c-energybus.html.
[22] http://cadp.inria.fr/case-studies/17-a-production-cell.html.
[23] http://cadp.inria.fr/software/12-e-choreography.html.
[24] http://cadp.inria.fr/software/12-i-dftcalc.html.
[25] http://cadp.inria.fr/software/13-a-eb3.html.
[26] http://cadp.inria.fr/software/14-c-grl.html.
[27] http://cadp.inria.fr/software/15-b-ocarina.html.

- The PIC2LNT translator[28] [77] for the applied π-calculus (an extension of the π-calculus with typed data values);
- the VBPMN translator[29] [87] for the BPMN language for describing business processes (Université Paris Ouest, France).

8 Conclusion

Computer systems handling asynchronous concurrency are inherently complex and cannot be reliably designed without adequate specification languages supported by sound analysis tools. Ed Brinksma contributed to this agenda in two significant ways: (i) by leading the definition and standardization of the LOTOS language, which exposed the key ideas of process calculi to a large audience and sparkled considerable interest in academia and industry; (ii) by sending a clear signal that LOTOS, despite its qualities, was not the end of the road and that further enhancements were possible and desirable.

The present paper provided a retrospective account of the evolution of the LOTOS-based family of specification languages, starting from LOTOS itself, reviewing the successive proposals for enhancing LOTOS (Extended LOTOS, Modular LOTOS, E-LOTOS, and LOTOS NT), and ending with LNT, the most recent descendent, which preserves the most valuable ideas of process calculi but entirely modifies the shape of the language to make it compatible with mainstream programming languages. The feedback acquired by using LNT for the design of complex industrial systems suggests that LNT provides a viable and effective replacement for LOTOS. Quoting STMicroelectronics engineers: "Although modeling the [Dynamic Task Dispatcher] in a classical formal specification language, such as LOTOS, is theoretically possible, using LNT made the development of a formal model practically feasible" [69].

Concerning future work, we can highlight two main research directions:

- The LNT language is not yet frozen and can still be further enhanced. For instance, the unification of exceptions across functions and processes is almost complete. We now consider equipping processes with optional **return** behaviours, so that functions become a strict subset of processes. We also plan to introduce, beyond assertions that already exist in LNT, pre-conditions, post-conditions, and loop invariants that would allow the application of mainstream theorem provers and static analyzers to LNT specifications.
- The current implementation of LNT by translation to LOTOS is justified by the reuse of existing LOTOS tools. It is intellectually challenging, but sometimes overly complex: for instance, LNT functions, written in a functional/imperative style, are first translated to LOTOS algebraic equations, and then compiled back to imperative C code. A native implementation of LNT would certainly be simpler and more efficient; it would also overcome

[28] http://cadp.inria.fr/software/13-d-pic2lnt.html.
[29] http://cadp.inria.fr/software/16-a-vbpmn.html.

certain LOTOS limitations that currently prevent useful constructs, such as the **trap** operator for exception catching [49] and the n-among-m synchronization pattern in parallel composition [51], from being added to LNT.

References

1. Abid, R., Salaün, G., Bongiovanni, F., De Palma, N.: Verification of a dynamic management protocol for cloud applications. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 178–192. Springer, Cham (2013). doi:10.1007/978-3-319-02444-8_14
2. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. **126**(2), 183–235 (1994)
3. Arnold, F., Belinfante, A., Van der Berg, F., Guck, D., Stoelinga, M.: DFTCalc: a tool for efficient fault tree analysis. In: Bitsch, F., Guiochet, J., Kaâniche, M. (eds.) SAFECOMP 2013. LNCS, vol. 8153, pp. 293–301. Springer, Heidelberg (2013). doi:10.1007/978-3-642-40793-2_27
4. Berthomieu, B., Bodeveix, J.P., Farail, P., Filali, M., Garavel, H., Gaufillet, P., Lang, F., Vernadat, F.: FIACRE: an intermediate language for model verification in the TOPCASED environment. In: Laprie, J.C. (ed.) Proceedings of the 4th European Congress on Embedded Real-Time Software (ERTS 2008), Toulouse, France, January 2008
5. Berthomieu, B., Le Sergent, T.: Programming with behaviors in an ML framework — the syntax and semantics of LCS. In: Sannella, D. (ed.) ESOP 1994. LNCS, vol. 788, pp. 89–104. Springer, Heidelberg (1994). doi:10.1007/3-540-57880-3_6
6. Bolognesi, T., Brinksma, E.: Introduction to the ISO specification language LOTOS. Comput. Netw. ISDN Syst. **14**(1), 25–59 (1988)
7. Boullier, P., Jourdan, M.: A new error repair and recovery scheme for lexical and syntactic analysis. Sci. Comput. Program. **9**(3), 271–286 (1987)
8. Boyer, F., Gruber, O., Salaün, G.: Specifying and verifying the SYNERGY reconfiguration protocol with LOTOS NT and CADP. In: Butler, M., Schulte, W. (eds.) FM 2011. LNCS, vol. 6664, pp. 103–117. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21437-0_10
9. Brinksma, E., Leih, G.: Enhancements of LOTOS. In: Bolognesi, T., Lagemaat, J., Vissers, C. (eds.) LOTOSphere: Software Development with LOTOS, pp. 453–466. Kluwer Academic Publishers, Dordrecht (1995)
10. Brinksma, E.: A tutorial on LOTOS. In: Diaz, M. (ed.) Proceedings of the 5th IFIP International Workshop on Protocol Specification, Testing and Verification (PSTV 1885), Moissac, France, pp. 171–194. North-Holland, Amsterdam, June 1985
11. Brinksma, E.: On the design of Extended LOTOS - a specification language for open distributed systems. Ph.D. thesis, University of Twente, November 1988
12. Brinksma, E.: Constraint-oriented specification in a constructive formal description technique. In: de Bakker, J.W., de Roever, W.-P., Rozenberg, G. (eds.) REX 1989. LNCS, vol. 430, pp. 130–152. Springer, Heidelberg (1990). doi:10.1007/3-540-52559-9_63
13. Brinksma, E.: Task 1.4 Deliverable on Language Enhancements, LOTOSphere (ESPRIT Projet 2304) Document ref. Lo/WP1/T1.4/N0016/V3, 146 p., April 1992

14. Brinksma, E., Karjoth, G.: A specification of the OSI transport service in LOTOS. In: Yemini, Y., Strom, R.E., Yemini, S. (eds.) Proceedings of the 4th IFIP International Workshop on Protocol Specification, Testing and Verification, Skytop Lodge, PA, USA, pp. 227–251. North-Holland, Amsterdam, June 1984
15. Brinksma, E., Katoen, J.P., Langerak, R., Latella, D.: A stochastic causality-based process algebra. Comput. J. **38**(7), 552–565 (1995)
16. Brinksma, E., Tretmans, J., Verhaard, L.: A framework for test selection. In: Jonsson, B., Parrow, J., Pehrson, B. (eds.) Proceedings of the IFIP WG6.1 9th International Symposium on Protocol Specification, Testing and Verification, Stockholm, Sweden. pp. 233–248. North-Holland, Amsterdam, June 1991
17. Brookes, S.D., Hoare, C.A.R., Roscoe, A.W.: A theory of communicating sequential processes. J. ACM **31**(3), 560–599 (1984)
18. CCITT: Specification and Description Language. Recommendation Z.100, International Consultative Committee for Telephony and Telegraphy, Geneva, March 1988
19. Champelovier, D., Clerc, X., Garavel, H., Guerte, Y., McKinty, C., Powazny, V., Lang, F., Serwe, W., Smeding, G.: Reference Manual of the LNT to LOTOS Translator (Version 6.7), INRIA, Grenoble, France, July 2017
20. Chebieb, A., Ameur, Y.A.: Formal verification of plastic user interfaces exploiting domain ontologies. In: Zhiqiu, H., Jun, S. (eds.) Proceedings of the International Symposium on Theoretical Aspects of Software Engineering (TASE 2015), Nanjing, China, pp. 79–86. IEEE Computer Society, Washington, D.C. (2015)
21. Clark, R.G., Moreira, A.: Use of E-LOTOS in adding formality to UML. J. Univers. Comput. Sci. **6**(11), 1071–1087 (2000)
22. Courtiat, J., Santos, C.A.S., Lohr, C., Outtaj, B.: Experience with RT-LOTOS, a temporal extension of the LOTOS formal description technique. Comput. Commun. **23**(12), 1104–1123 (2000)
23. Crouzen, P., Lang, F.: Smart reduction. In: Giannakopoulou, D., Orejas, F. (eds.) FASE 2011. LNCS, vol. 6603, pp. 111–126. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19811-3_9
24. de Souza, W.L., et al.: Design of distributed multimedia applications (DAMD). In: Hutter, D., Stephan, W., Traverso, P., Ullmann, M. (eds.) FM-Trends 1998. LNCS, vol. 1641, pp. 77–91. Springer, Heidelberg (1999). doi:10.1007/3-540-48257-1_4
25. Ehrig, H., Fey, W., Hansen, H.: An algebraic specification language with two levels of semantics. Bericht No. 83-03, Fachbereich 20-Informatik, Technische Universität Berlin (1983)
26. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. EATCS Monographs on Theoretical Computer Science, vol. 6. Springer, Heidelberg (1985). doi:10.1007/978-3-642-69962-7
27. Etchevers, X., Salaün, G., Boyer, F., Coupaye, T., Palma, N.D.: Reliable self-deployment of distributed cloud applications. Softw. Pract. Exp. **47**(1), 3–20 (2017)
28. Evrard, H.: Génération automatique d'implémentation distribuée à partir de modèles formels de processus concurrents asynchrones. Thèse de Doctorat, Université de Grenoble, July 2015
29. Evrard, H.: DLC: compiling a concurrent system formal specification to a distributed implementation. In: Chechik, M., Raskin, J.-F. (eds.) TACAS 2016. LNCS, vol. 9636, pp. 553–559. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49674-9_34

30. Evrard, H., Lang, F.: Formal verification of distributed branching multiway synchronization protocols. In: Beyer, D., Boreale, M. (eds.) FMOODS/FORTE - 2013. LNCS, vol. 7892, pp. 146–160. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38592-6_11
31. Evrard, H., Lang, F.: Automatic distributed code generation from formal models of asynchronous concurrent processes. In: Aldinucci, M., Daneshtalab, M., Leppänen, V., Lilius, J. (eds.) Proceedings of the 23rd Euromicro International Conference on Parallel, Distributed and Network-based Processing - Special Session on Formal Approaches to Parallel and Distributed Systems (PDP/4PAD 2015), Turku, Finland, pp. 459–466. IEEE Computer Society Press, Washington, D.C., March 2015
32. Evrard, H., Lang, F.: Automatic distributed code generation from formal models of asynchronous processes interacting by multiway rendezvous. J. Log. Algebr. Methods Program. **88**, 121–153 (2017)
33. Garavel, H.: Utilisation du système CESAR pour la vérification de protocoles spécifiés en LOTOS. Rapport SPECTRE C2, Laboratoire de Génie Informatique - Institut IMAG, Grenoble, December 1986
34. Garavel, H.: Vérification de programmes LOTOS à l'aide du système QUASAR. Master's thesis, Institut National Polytechnique de Grenoble, September 1986
35. Garavel, H.: Compilation of LOTOS abstract data types. In: Vuong, S.T. (ed.) Proceedings of the 2nd International Conference on Formal Description Techniques FORTE 1989, Vancouver BC, Canada, pp. 147–162. North-Holland, Amsterdam, December 1989
36. Garavel, H.: On the introduction of gate typing in E-LOTOS. In: Dembinski, P., Sredniawa, M. (eds.) Proceedings of the 15th IFIP International Workshop on Protocol Specification, Testing and Verification (PSTV 1995), Warsaw, Poland, pp. 283–298. Chapman & Hall, New York, June 1995
37. Garavel, H.: OPEN/CÆSAR: an open software architecture for verification, simulation, and testing. In: Steffen, B. (ed.) TACAS 1998. LNCS, vol. 1384, pp. 68–84. Springer, Heidelberg (1998). doi:10.1007/BFb0054165
38. Garavel, H.: Défense et illustration des algèbres de processus. In: Mammeri, Z. (ed.) Actes de l'Ecole d'été Temps Réel ETR 2003, Toulouse, France. Institut de Recherche en Informatique de Toulouse, September 2003
39. Garavel, H.: Reflections on the future of concurrency theory in general and process calculi in particular. In: Palamidessi, C., Valencia, F.D. (eds.) Proceedings of the LIX Colloquium on Emerging Trends in Concurrency Theory, Ecole Polytechnique de Paris, France, 13–15 November 2006. Electronic Notes in Theoretical Computer Science, vol. 209, pp. 149–164. Elsevier Science Publishers, Amsterdam, April 2008. Also available as INRIA Research Report RR-6368
40. Garavel, H.: Revisiting sequential composition in process calculi. J. Log. Algebr. Methods Program. **84**(6), 742–762 (2015)
41. Garavel, H., Hermanns, H.: On combining functional verification and performance evaluation using CADP. In: Eriksson, L.-H., Lindsay, P.A. (eds.) FME 2002. LNCS, vol. 2391, pp. 410–429. Springer, Heidelberg (2002). doi:10.1007/3-540-45614-7_23
42. Garavel, H., Lang, F.: SVL: a scripting language for compositional verification. In: Kim, M., Chin, B., Kang, S., Lee, D. (eds.) FORTE 2001. IIFIP, vol. 69, pp. 377–392. Kluwer Academic Publishers, Dordrecht (2002). doi:10.1007/0-306-47003-9_24

43. Garavel, H., Lang, F.: NTIF: a general symbolic model for communicating sequential processes with data. In: Peled, D.A., Vardi, M.Y. (eds.) FORTE 2002. LNCS, vol. 2529, pp. 276–291. Springer, Heidelberg (2002). doi:10.1007/3-540-36135-9_18
44. Garavel, H., Lang, F., Mateescu, R.: Compiler construction using LOTOS NT. In: Horspool, R.N. (ed.) CC 2002. LNCS, vol. 2304, pp. 9–13. Springer, Heidelberg (2002). doi:10.1007/3-540-45937-5_3
45. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Int. J. Softw. Tools Technol. Transf. (STTT) **15**(2), 89–107 (2013). Springer
46. Garavel, H., Salaün, G., Serwe, W.: On the semantics of communicating hardware processes and their translation into LOTOS for the verification of asynchronous circuits with CADP. Sci. Comput. Program. **74**(3), 100–127 (2009)
47. Garavel, H., Serwe, W.: The unheralded value of the multiway rendezvous: illustration with the production cell benchmark. In: Hermanns, H., Höfner, P. (eds.) Proceedings of the 2nd Workshop on Models for Formal Analysis of Real Systems (MARS 2017), Uppsala, Sweden, vol. 244, pp. 230–270. Electronic Proceedings in Theoretical Computer Science, April 2017
48. Garavel, H., Sifakis, J.: Compilation and verification of LOTOS specifications. In: Logrippo, L., Probert, R.L., Ural, H. (eds.) Proceedings of the 10th IFIP International Symposium on Protocol Specification, Testing and Verification (PSTV 1990), Ottawa, Canada, pp. 379–394. North-Holland, Amsterdam, June 1990
49. Garavel, H., Sighireanu, M.: On the introduction of exceptions in LOTOS. In: Gotzhein, R., Bredereke, J. (eds.) Proceedings of the IFIP Joint International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification (FORTE/PSTV 1996), Kaiserslautern, Germany, pp. 469–484. Chapman & Hall, New York, October 1996
50. Garavel, H., Sighireanu, M.: Towards a second generation of formal description techniques - rationale for the design of E-LOTOS. In: Groote, J.F., Luttik, B., Wamel, J. (eds.) Proceedings of the 3rd International Workshop on Formal Methods for Industrial Critical Systems (FMICS 1998), Amsterdam, The Netherlands, pp. 187–230. CWI, Amsterdam, May 1998. Invited lecture
51. Garavel, H., Sighireanu, M.: A graphical parallel composition operator for process algebras. In: Wu, J., Chanson, S.T., Gao, Q. (eds.) Formal Methods for Protocol Engineering and Distributed Systems. IAICT, vol. 28, pp. 185–202. Kluwer Academic Publishers, Dordrecht (1999)
52. Garavel, H., Thivolle, D.: Verification of GALS systems by combining synchronous languages and process calculi. In: Păsăreanu, C.S. (ed.) SPIN 2009. LNCS, vol. 5578, pp. 241–260. Springer, Heidelberg (2009). doi:10.1007/978-3-642-02652-2_20
53. Graf-Brill, A., Hermanns, H., Garavel, H.: A model-based certification framework for the EnergyBus standard. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 84–99. Springer, Heidelberg (2014). doi:10.1007/978-3-662-43613-4_6
54. Guck, D., Spel, J., Stoelinga, M.: DFTCalc: reliability centered maintenance via fault tree analysis (tool paper). In: Butler, M., Conchon, S., Zaïdi, F. (eds.) ICFEM 2015. LNCS, vol. 9407, pp. 304–311. Springer, Cham (2015). doi:10.1007/978-3-319-25423-4_19
55. Güdemann, M., Salaün, G., Ouederni, M.: Counterexample guided synthesis of monitors for realizability enforcement. In: Chakraborty, S., Mukund, M. (eds.)

ATVA 2012. LNCS, vol. 7561, pp. 238–253. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33386-6_20

56. Huecas, G., Llana-Díaz, L., Quemada, J., Robles, T., Verdejo, A.: Process calculi: E-LOTOS. In: Bowman, H., Derrick, J. (eds.) Formal Methods for Distributed Processing: A Survey of Object-Oriented Approaches, pp. 77–104. Cambridge University Press, Cambridge (2001)
57. Huecas, G., Llana-Díaz, L., Robles, T., Verdejo, A.: E-LOTOS: an overview. In: Marsan, M.A., Quemada, J., Robles, T., Silva, M. (eds.) Proceedings of the Workshop on Formal Methods and Telecommunications (WFMT'99), Zaragoza, Spain, pp. 94–102. Prensas Universitarias de Zaragoza, September 1999
58. ISO/IEC: LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Draft International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection, Geneva, July 1987
59. ISO/IEC: ESTELLE - A Formal Description Technique Based on an Extended State Transition Model. International Standard 9074, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection, Geneva, September 1988
60. 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, Geneva, September 1989
61. ISO/IEC: Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, International Organization for Standardization - Information Technology, Geneva, September 2001
62. Jebali, F., Lang, F., Mateescu, R.: Formal modelling and verification of GALS systems using GRL and CADP. Formal Asp. Comput. **28**(5), 767–804 (2016)
63. Jourdan, M., Parigot, D.: Application development with the FNC-2 attribute grammar system. In: Hammer, D. (ed.) CC 1990. LNCS, vol. 477, pp. 11–25. Springer, Heidelberg (1991). doi:10.1007/3-540-53669-8_71
64. Kriouile, A., Serwe, W.: Formal analysis of the ACE specification for cache coherent systems-on-chip. In: Pecheur, C., Dierkes, M. (eds.) FMICS 2013. LNCS, vol. 8187, pp. 108–122. Springer, Heidelberg (2013). doi:10.1007/978-3-642-41010-9_8
65. Kriouile, A., Serwe, W.: Using a formal model to improve verification of a cache-coherent system-on-chip. In: Baier, C., Tinelli, C. (eds.) TACAS 2015. LNCS, vol. 9035, pp. 708–722. Springer, Heidelberg (2015). doi:10.1007/978-3-662-46681-0_62
66. Lang, F.: Compositional verification using SVL scripts. In: Katoen, J.-P., Stevens, P. (eds.) TACAS 2002. LNCS, vol. 2280, pp. 465–469. Springer, Heidelberg (2002). doi:10.1007/3-540-46002-0_33
67. Lang, F.: Exp.Open 2.0: a flexible tool integrating partial order, compositional, and on-the-fly verification methods. In: Romijn, J., Smith, G., van de Pol, J. (eds.) IFM 2005. LNCS, vol. 3771, pp. 70–88. Springer, Heidelberg (2005). doi:10.1007/11589976_6. Full version available as INRIA Research Report RR-5673
68. Lang, F., Salaün, G., Hérilier, R., Kramer, J., Magee, J.: Translating FSP into LOTOS and networks of automata. Formal Asp. Comput. **22**(6), 681–711 (2010)
69. Lantreibecq, E., Serwe, W.: Formal analysis of a hardware dynamic task dispatcher with CADP. Sci. Comput. Program. **80**(Part A), 130–149 (2014)
70. Leduc, G., Jeffrey, A., Sighireanu, M.: Introduction à E-LOTOS. In: Cavalli, A. (ed.) Ingénierie des protocoles et qualité de service. Collection IC2, chap. 6, pp. 213–253. Hermès, Paris (2001)

71. Léonard, L., Leduc, G.: An introduction to ET-LOTOS for the description of time-sensitive systems. Comput. Netw. ISDN Syst. **29**(3), 271–292 (1997)
72. Léonard, L., Leduc, G.: A formal definition of time in LOTOS. Formal Asp. Comput. **10**(3), 248–266 (1998)
73. Li, X., Madnick, S., Zhu, H., Fan, Y.: Improving data quality for web services composition. In: Proceedings of the 7th International Workshop on Quality in Databases (QDB 2009), Lyon, France, August 2009
74. Massetto, F.I., de Souza, W.L., Zorzo, S.D.: Simulator for E-LOTOS specifications. In: Proceedings of the 35th Annual Simulation Symposium (SS 2002), San Diego, California, USA, pp. 389–394. IEEE Computer Society, Washington, D.C., April 2002
75. Mateescu, R.: A generic framework for model checking software architectures. In: Augusto, J.C., Ultes-Nitsche, U. (eds.) Proceedings of the 2nd International Workshop on Verification and Validation of Enterprise Information Systems (VVEIS 2004), Porto, Portugal. INSTICC Press, April 2004. Keynote presentation
76. Mateescu, R., Monteiro, P.T., Dumas, E., de Jong, H.: Computation tree regular logic for genetic regulatory networks. In: Cha, S.S., Choi, J.-Y., Kim, M., Lee, I., Viswanathan, M. (eds.) ATVA 2008. LNCS, vol. 5311, pp. 48–63. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88387-6_6
77. Mateescu, R., Salaün, G.: PIC2LNT: model transformation for model checking an applied pi-calculus. In: Piterman, N., Smolka, S.A. (eds.) TACAS 2013. LNCS, vol. 7795, pp. 192–198. Springer, Heidelberg (2013). doi:10.1007/978-3-642-36742-7_14
78. Mateescu, R., Serwe, W.: Model checking and performance evaluation with CADP illustrated on shared-memory mutual exclusion protocols. Sci. Comput. Program. **78**(7), 843–861 (2013)
79. Mateescu, R., Thivolle, D.: A model checking language for concurrent value-passing systems. In: Cuellar, J., Maibaum, T., Sere, K. (eds.) FM 2008. LNCS, vol. 5014, pp. 148–164. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68237-0_12
80. de Meer, J., Roth, R., Vuong, S.: Introduction to algebraic specifications based on the language ACT ONE. Comput. Netw. ISDN Syst. **23**(5), 363–392 (1992)
81. Milne, G.J.: CIRCAL and the representation of communication, concurrency, and time. ACM Trans. Progr. Lang. Syst. **7**(2), 270–298 (1985)
82. Milner, R. (ed.): A Calculus of Communicating Systems. LNCS, vol. 92. Springer, Heidelberg (1980). doi:10.1007/3-540-10235-3
83. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. **25**, 267–310 (1983)
84. Mkaouar, H., Zalila, B., Hugues, J., Jmaiel, M.: From AADL model to LNT specification. In: de la Puente, J.A., Vardanega, T. (eds.) Ada-Europe 2015. LNCS, vol. 9111, pp. 146–161. Springer, Cham (2015). doi:10.1007/978-3-319-19584-1_10
85. Oliveira, R., Dupuy-Chessa, S., Calvary, G., Dadolle, D.: Using formal models to cross check an implementation. In: Luyten, K., Palanque, P. (eds.) Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS 2016), Brussels, Belgium, pp. 126–137. ACM, New York, June 2016
86. Poizat, P., Salaün, G.: Checking the realizability of BPMN 2.0 choreographies. In: Proceedings of the 27th Symposium On Applied Computing (SAC 2012), Riva del Garda, Italy. ACM Press, New York, March 2012
87. Poizat, P., Salaün, G., Krishna, A.: Checking business process evolution. In: Kouchnarenko, O., Khosravi, R. (eds.) FACS 2016. LNCS, vol. 10231, pp. 36–53. Springer, Cham (2017). doi:10.1007/978-3-319-57666-4_4

88. Ponsini, O., Fédèle, C., Kounalis, E.: Rewriting of imperative programs into logical equations. Sci. Comput. Program. **56**(3), 363–401 (2005)
89. Quemada, J.: E-LOTOS Has Born, February 1997. Email announcement available from ftp://ftp.inrialpes.fr/pub/vasy/publications/elotos/announce-97.txt
90. Roth, R., de Meer, J., Storp, S.: Data specifications in Modular LOTOS. In: Bolognesi, T., Lagemaat, J., Vissers, C. (eds.) LOTOSphere: Software Development with LOTOS, pp. 467–479. Kluwer Academic Publishers, Dordrecht (1995)
91. Serwe, W.: Formal specification and verification of fully asynchronous implementations of the Data Encryption Standard. In: van Glabbeek, R., Groote, J.F., Höfner, P. (eds.) Proceedings of the International Workshop on Models for Formal Analysis of Real Systems (MARS 2015), Suva, Fiji. Electronic Proceedings in Theoretical Computer Science, vol. 196. Open Publishing Association (2015)
92. Shankland, C., Verdejo, A.: Time, E-LOTOS, and the FireWire. In: Marsan, M.A., Quemada, J., Robles, T., Silva, M. (eds.) Proceedings of the Workshop on Formal Methods and Telecommunications (WFMT 1999), Zaragoza, Spain, pp. 103–119. Prensas Universitarias de Zaragoza, September 1999
93. Shankland, C., Verdejo, A.: A case study in abstraction using E-LOTOS and the FireWire. Comput. Netw. **37**(3/4), 481–502 (2001)
94. Sighireanu, M.: Contribution à la définition et à l'implémentation du langage "Extended LOTOS". Thèse de Doctorat, Université Joseph Fourier (Grenoble), January 1999
95. Sighireanu, M., Catry, A., Champelovier, D., Garavel, H., Lang, F., Schaeffer, G., Serwe, W., Stoecker, J.: LOTOS NT User's Manual (Version 2.8), INRIA/CONVECS, Grenoble, France, 109 p. ftp://ftp.inrialpes.fr/pub/vasy/traian/manual.pdf
96. Sighireanu, M., Turner, K.: Requirement capture, formal description and verification of an invoicing system. Research Report RR-3575, INRIA, Grenoble, December 1998
97. Stöcker, J., Lang, F., Garavel, H.: Parallel processes with real-time and data: the ATLANTIF intermediate format. In: Leuschel, M., Wehrheim, H. (eds.) IFM 2009. LNCS, vol. 5423, pp. 88–102. Springer, Heidelberg (2009). doi:10.1007/978-3-642-00255-7_7
98. Thivolle, D.: Langages modernes pour la vérification des systèmes asynchrones. Thèse de Doctorat, Université Joseph Fourier, Grenoble, France and Universitatea Politehnica din Bucuresti, Bucharest, Romania, April 2011
99. Turner, K.J., Sighireanu, M.: (E)-LOTOS: (enhanced) language of temporal ordering specification. In: Frappier, M., Habrias, H. (eds.) Software Specification Methods: An Overview Using a Case Study, pp. 166–190. Springer, London (2001). doi:10.1007/978-1-4471-0701-9_10
100. Vekris, D., Lang, F., Dima, C., Mateescu, R.: Verification of EB3 specifications using CADP. Formal Asp. Comput. **28**(1), 145–178 (2016)
101. Verdejo, A.: E-LOTOS: Tutorial and Semantics. Master's thesis, Departamento de Sistemas Informáticos y Programación, Universidad Complutense de Madrid, Spain, June 1999
102. Wu, H., Yang, X., Katoen, J.-P.: Performance evaluation of concurrent data structures. In: Fränzle, M., Kapur, D., Zhan, N. (eds.) SETTA 2016. LNCS, vol. 9984, pp. 38–49. Springer, Cham (2016). doi:10.1007/978-3-319-47677-3_3
103. Zhang, Z., Serwe, W., Wu, J., Zheng, T.Y.H., Myers, C.: An improved fault-tolerant routing algorithm for a network-on-chip derived with formal analysis. Sci. Comput. Program. **118**, 24–39 (2016)