On Combining Functional Verification and Performance Evaluation Using CADP

Hubert Garavel[1] and Holger Hermanns[2]

[1] INRIA Rhône-Alpes / VASY, 655, avenue de l'Europe
F-38330 Montbonnot Saint-Martin, France
[2] Formal Methods and Tools Group, University of Twente,
P.O. Box 217, NL-7500 AE Enschede, The Netherlands

Abstract. Considering functional correctness and performance evaluation in a common framework is desirable, both for scientific and economic reasons. In this paper, we describe how the CADP toolbox, originally designed for verifying the functional correctness of LOTOS specifications, can also be used for performance evaluation. We illustrate the proposed approach by the performance study of the SCSI-2 bus arbitration protocol.

1 Introduction

The design of models suited for performance and reliability analysis of systems is difficult because of their increase in size and complexity, in particular for systems with a high degree of irregularity. Traditional performance models like Markov chains and queueing networks are not easy to apply in these areas, mainly because they lack hierarchical composition and abstraction means. Therefore, if attempts are nowadays made to assess performance of complex designs, they are most often isolated from the system design cycle. This *insularity problem* of performance evaluation [10] is undesirable.

On the other hand, to describe and analyse the functional properties of designs, various specification formalisms exist, which enable systems to be modelled in a compositional, hierarchical manner. A prominent example of such specification formalisms is the class of *process algebras*, which provide abstraction mechanisms to treat system components as black boxes, making their internal implementation details invisible.

Among the many process algebras proposed in the literature, LOTOS [27,7,35] has received much attention, due to its technical merits and its status of ISO/IEC International Standard. CADP (*Caesar/Aldebaran Development Package*) [17] is a widespread tool set for the design and verification of complex systems. CADP supports the process algebra LOTOS for specification, and offers various tools for simulation and formal verification, including equivalence checkers (bisimulations) and model checkers (temporal logics and modal μ-calculus).

Facing these advanced means to construct correct models of complex systems, it appears most interesting to investigate how performance evaluation can be carried out on the basis of such models, and this is what the present paper is about. Functional correctness and performance evaluation being two facets of the same problem, which is the proper functioning of a system, it is desirable to address them together, both for scientific and economic reasons. This requires *(i)* a common theoretical framework, *(ii)* a common language for modelling both functional and performance aspects, *(iii)* a common methodology for combining both aspects, and *(iv)* software tools implementing the appropriate algorithms.

To arrive at this joint consideration of functionality and performance, we follow the approach advocated in [23]. We start from a functionally verified LOTOS specification, in which we introduce timing related information, which expresses that certain events are delayed by a random time (governed by an exponential distribution or, more generally, a phase-type distribution).

To support this methodology, we use the existing software components of CADP, as well as a novel tool named BCG_MIN, which we developed for minimising stochastic models. We illustrate the approach with an industrial case study: the bus arbitration protocol used in the SCSI-2 [2] standard.

We are not the first to advocate a joint consideration of functional verification and performance evaluation. This idea has driven the development of stochastic Petri nets [1], stochastic process algebras [24,29,26,4,20], as well as other approaches, e.g., [6]. Our proposal can be considered as a pragmatic outcome of research on stochastic algebras, other tools in this context being the PEPA-workbench [14], TWOTOWERS [3], and the TIPPTOOL [21]. Although on a superficial level all these tools implement an approach similar to ours, only TWOTOWERS provides support for both functional verification as well as performance evaluation. Moreover, we are not aware of any publication considering both functional correctness and performance properties for industrial scale applications, with the exception of [23], where a verified LOTOS specification of a telephone system is studied with respect to performance properties. One conclusion of [23] was a lack of tool support for doing industrial strength case studies, a problem that we address here explicitly.

This paper is organised as follows. Section 2 explains how the process algebra LOTOS can be used for modelling Markovian aspects, and describes extensions of CADP to support performance evaluation. The functional part of the SCSI-2 case study is introduced in Section 3, while Section 4 covers the performance-related modelling and analysis aspects for the SCSI protocol. Finally, Section 5 concludes the paper.

2 The Proposed Approach

Our approach to combining functional verification and performance evaluation is pragmatic in the sense that, instead of developing new models, new languages and new tools, it is, to a large extent, based on prior work for 'classical' (i.e., non-stochastic) process algebras, and especially the CADP tools. However, to address performance aspects, the CADP tools (originally designed for functional verification only) must be extended and combined with performance tools. To do so, several challenging issues must be addressed. In this section, we present the principles of our approach and their practical implementation.

2.1 Interactive Markov Chains

To define the operational semantics of process algebras, the usual model is that of *labelled transition systems* (LTS for short). An LTS is a directed graph whose vertices denote the global *states* of the system and whose edges correspond to the *transitions* permitted by the system. Each transition is labelled by an *action*, and there is one distinguished state considered as the *initial state*.

As regards functional verification, many verification techniques (such as those implemented in CADP) are based on the LTS model.

As regards performance evaluation, many stochastic models derived from state-transition diagrams have been proposed. Our approach is based on the *Interactive Markov Chains* model [19] (IMC for short), which is well-adapted to process algebras. An IMC is simply an LTS whose transitions can be either labelled with an action (as in an 'ordinary' LTS) or with special labels of the form "`rate` λ", where λ belongs to the set of positive reals. A transition "`rate` λ" going out of some state S is called a *delay transition* and expresses an internal delay in state S. More precisely, it indicates that the time t spent in S follows a so-called *negative exponential distribution function* $Prob\{t \leq x\} = 1 - e^{-\lambda x}$, to be read as: the probability that state S is exited at time x the latest equals $1 - e^{-\lambda x}$. The parameter λ of the distribution is called a *Markov delay*; it is also referred to as the *rate* of the distribution (the rate being the reciprocal value of the mean duration of an exponentially distributed delay). The IMC model is very general in several respects:

- It contains, as two particular cases, the LTS model (which is obtained when there is no delay transition) and the well-known *Continuous Time Markov Chain* model (which is obtained when there are only delay transitions). The latter model (CTMC for short) has been extensively studied in the literature and is equipped with various efficient evaluation strategies (see, e.g. [34]).
- The IMC model allows nondeterminism in states, i.e., two identical action transitions leaving the same state. Nondeterminism is an important feature if the IMC model is to be generated automatically from higher-level languages such as process algebra.
- Unlike some stochastic models (e.g. [26,4]), the IMC model does not require a strict alternation between actions and delays. It is therefore permitted to have several successive actions not separated by a delay in between. It is also permitted to have several delays interspersed between actions. This is practically useful: by combining several exponential distributions one can define a more general class of distributions, so-called *phase-type distributions*. Concretely, each CTMC fragment with an *absorbing* state (i.e., a state without `rate`-successors) can be used to represent a phase-type distribution, which describes the time needed to reach the absorbing state from the initial state. For instance, the following example:

$$\ldots \circ \xrightarrow{A} \circ \xrightarrow{\texttt{rate}\ 10} \circ \xrightarrow{\texttt{rate}\ 10} \circ \xrightarrow{\texttt{rate}\ 10} \circ \xrightarrow{B} \circ \ldots$$

  expresses that the occurrence of action B after witnessing action A is delayed by an Erlang-3 distribution. This is an important feature, as phase-type distributions can approximate arbitrary distributions arbitrarily close [32].

There is a subtle, but important difference between the LTS and IMC models. In the LTS model, given an action A and two states S_1 and S_2, there is *at most one* transition labelled by A going from S_1 to S_2. It is not possible to have several identical transitions between the same states, because transitions are usually defined by a relation over *States* $\times$ *Actions* $\times$ *States*. Technically, it would be easy to allow identical transitions by using a *multirelation* over *States* $\times$ *Actions* $\times$ *States* instead. But this is not the standard approach, as the usual means of observing LTSs (bisimulations, μ-calculus, SOS rules that define the semantics of process algebraic operators used to compose LTSs) only check for the existence of transitions and, thus, would not make any difference between one and several identical transitions.

The situation is different in the stochastic setting. Multiplicity of identical transitions is making a difference in the case of delay transitions. Given a rate λ and two states S_1 and S_2, the co-existence of two transitions labelled "`rate` λ" expresses that there are two competing ways to reach S_2. According to this so-called *race interpretation*, which is widely used to explain the behaviour of Markov chains over time, these two delay transitions could be merged into a unique transition "`rate` 2λ" that cumulates their rates.

Concretely, in our approach, LTSs and IMCs are encoded in the BCG (*Binary Coded Graphs*) file format. BCG is a compact format for storing very large LTSs. It plays a pivotal role in the CADP tool set, which provides programming interfaces and a comprehensive collection of software tools to handle BCG files. The BCG format can handle identical transitions according to the multirelation semantics because, for time efficiency reasons, transitions are stored inside the BCG format as a list-like data structure, without checking for duplicates.

2.2 Using LOTOS to Express Interactive Markov Chains

Although it is possible to specify performance aspects directly at the IMC level, this is not always suitable for complex systems, which are more easily described using higher level languages. Our approach is based on the LOTOS process algebra, which we briefly present hereafter.

LOTOS is a formal description technique for specifying communication protocols and distributed systems at a high abstraction level and with a strong mathematical basis. Its definition [27] features two parts.

The *data part* is based on the theory of algebraic data types. It allows the definition of data structures described by *sorts*, which represent value domains, and *operations*, which are mathematical functions defined on these domains using algebraic *equations*. Sorts, operations, and equations are grouped in modules called *types*, which can be combined together using importation, renaming, parameterisation, and actualisation. The underlying semantics is that of initial algebras.

The *behaviour part* combines the best features of the pioneering process algebras, notably Milner's CCS and Hoare's CSP. It is used to describe concurrent processes that synchronise and communicate by rendezvous message-passing. LOTOS has a small set of basic operators (sequential composition, non-deterministic choice, guard, parallel composition, etc.), which can be combined together to express complex behaviours. The semantics of LOTOS is defined operationally in terms of (finite or infinite) LTSs. We refer to [7,35] for further reading.

As LOTOS is mainly intended for functional aspects (data and behaviours), it provides no built-in support for quantitative time nor performance modelling. It is worth noticing that the recent E-LOTOS standard [28], which introduces quantitative time, still lacks support for performance aspects. In particular, a concept like *randomness* or *probability* has not been included.

At this point, we are confronted to a crucial choice: either designing a new process algebra containing stochastic extensions (as done with TIPP [15], PEPA [26], or EMPA [4]), or taking LOTOS as is and extend it orthogonally with stochastic features. The former approach requires to develop a whole set of new tools, which we want to avoid for time/cost reasons. We therefore chose the latter approach, so as to reuse existing tools already available for LOTOS, in particular the CÆSAR.ADT [11] and CÆSAR [13] tools of CADP.

CÆSAR.ADT and CÆSAR are two complementary LOTOS to C compilers, the former for the data part, the latter for the behaviour part of LOTOS. The C code generated by these compilers is then used by other CADP tools for various purposes: simulation, random execution, on the fly verification, test generation, etc. Additionally, CÆSAR can generate the LTS corresponding to a LOTOS specification, if of finite size. This LTS is encoded in the BCG format and can be verified using bisimulations and/or model-checking of μ-calculus or temporal logic formulas.

Extending LOTOS with stochastic constructs would imply deep changes in the existing compilers in order to cope with delay transitions. Still guided by pragmatism, we found a lighter approach, which does not modify the syntax of LOTOS and requires no change in the CÆSAR.ADT and CÆSAR compilers. The principle is the following. Starting from a LOTOS specification whose functional correctness has been already verified, the user should, at every place in the LOTOS specification where a Markov delay λ_i should occur, insert an action Λ_i, where Λ_i is a new LOTOS *gate* (i.e., action name) expressing a communication with the external environment. The user should declare as many new gates Λ_i as there exists different rates λ_i. It is also possible to declare a single new gate Λ to which different parameter values will be associated (e.g., "Λ `!`i").

To ensure that introducing Markov delays does not corrupt the functional behaviour of the original specification, one can check that the LOTOS specification obtained after hiding the Λ_i gates (i.e., renaming these gates to τ) is equivalent to the original LOTOS specification modulo a weak equivalence (e.g., branching equivalence), or that both satisfy the same set of properties expressed in temporal logic or μ-calculus.

After the special gates Λ_i have been inserted in the LOTOS specification, CÆSAR and CÆSAR.ADT are invoked as usual to generate the corresponding LTS. This LTS is then turned into an IMC (still encoded in the BCG format) by replacing all its action transitions Λ_i with delay transitions "`rate` λ_i". This is done using the BCG_LABELS tool of CADP, which performs hiding and/or renaming on the labels attached to the transitions of a BCG file, according to a set of regular expression and substitution patterns specified by the user.

Our approach operates in two successive steps, first generating an Lts parameterised with action names Λ_i, then instantiating the Λ_i parameter with actual Markov delays. This is practically useful, as one often needs to try several values for each rate parameters when evaluating the performance of a system. With our approach, the highest cost (generating the parameterised Lts) occurs only once, while the instantiation costs are negligible in comparison.

One might wonder whether this two step approach is theoretically sound. For most Lotos operators (sequential composition, non-deterministic choice, process instantiation, etc.), there is no problem because the Imc model has been designed as an orthogonal extension of standard process algebra [19,23,20]. Yet, two points must be clarified:

– As regards parallel composition, there are various possible semantics for the synchronisation on a common action [25,20]. To avoid any ambiguity, we do not allow synchronisation on the special gates Λ_i. It is the user's responsibility not to synchronise these gates. For the same reason, the Lotos parallel operator "`||`", which forces synchronisation for all visible gates, should be avoided as well.

– With respect to the above discussion on multirelation semantics for transitions, it is true that the standard semantics of Lotos [27] is defined in terms of Ltss, contrary to stochastic process algebras, which rely (explicitly or implicitly) on multirelation semantics. However, Lotos could equally well be equipped with a multirelation semantics without disturbing its sound algebraic theory, given that both standard and multirelation semantics cannot be distinguished by strong bisimulation.

  Concretely, if a Lotos specification contains identical transitions (e.g., "Λ; `stop [] ` Λ; `stop`"), a Lotos compiler such as Cæsar can generate an Lts with one or two Λ-transitions, both solutions being equivalent modulo strong bisimulation; the number of Λ-transitions will mainly depend on the degree of optimisations done by the compiler internally. The user can safely avoid this issue by using, instead of Λ, two different gate names Λ_1 and Λ_2, which will be later instantiated with the same Markov delay.

There is another approach to extend a Lotos specification with stochastic timing information, besides the direct insertion of Markov delays in the specification text. This alternative approach is based on the use of *specification styles* [36] for Lotos, and especially the *constraint-oriented* style, which allows to refine the behaviour of an existing Lotos process by synchronising it with one (or several) concurrent process(es) expressing a set of temporal constraints on the ordering of actions. It has been suggested in [23] that the constraint-oriented style can be used to incorporate Markov delays (or even more complex phase-type distributions) between the actions of a Lotos specification, without modifying the specification text itself; see also [6] for a similar suggestion. Following this idea, a general operator for expressing time constraints compositionally has been proposed in [19, Section 5.5]. In this paper, we will illustrate both approaches, i.e., both the direct insertion of Markov delays in the Lotos text (see Section 3) and the superposition of time constraints specified externally (see Section 4).

2.3 Minimisation of Interactive Markov Chains

After generating an Lts from a Lotos specification and converting this Lts to an Imc by instantiating Markov delays with their actual values, the next step of our methodology consists in *minimising* this Imc, i.e., aggregating its state space. This minimisation is based on the (closely related) notions of bisimulation (on Lts) and *lumpability* (on Ctmcs), and is of interest for at least three reasons:

- It brings the Imc to a minimal number of states, still retaining its essential properties; this improves the efficiency of performance evaluation tools applied later to the minimised Imc;
- It replaces all delay transitions between a given pair of states by a single transition that cumulates the rates of these transitions; in particular, it removes identical transitions, so that multirelation semantics is no longer needed after minimisation.
- It may reduce (or even eliminate) nondeterminism, a concept not supported by performance evaluation algorithms; however, nondeterminism is not guaranteed to vanish after minimisation.

Although minimisation is practically useful, a lack of tool support to minimise large Imcs or Ctmcs has been identified (e.g., in [23] where the minimisation tool used could not handle more than 4,000 states). To account for this, we developed a software tool called Bcg_Min (3,000 lines of C code) for minimising Ltss and Imcs encoded in the Bcg format:

- As regards Ltss, Bcg_Min performs efficient minimisation with respect to either strong or branching bisimulation. According to independent experts, Bcg_Min is "*the best implementation of the standard [i.e., Groote & Vaandrager] algorithm for branching bisimulation*" [18]. Using Bcg_Min we have been able to minimise an Lts with 8 million states and 43 million transitions on a standard Pc.
- As regards Imcs, Bcg_Min implements both *stochastic strong bisimulation* and *stochastic branching bisimulation.* In a nutshell, stochastic strong (resp. branching) bisimulation combines lumpability on the delay transitions with strong (resp. branching) bisimulation on the action transitions. Consequently, Bcg_Min can be used to minimise Ctmcs modulo lumpability. A formal definition of stochastic strong bisimulation and stochastic weak bisimulation (a variant of stochastic branching bisimulation) can be found in [19].

Apart from Ltss, Ctmcs, and Imcs, Bcg_Min can handle a wide range of other models, including *(i)* stochastic models containing transitions labelled by (action, rate) pairs, which allows to minimise Tipp [15], Pepa [26], and Empa [4] models modulo strong equivalence and Markovian bisimulation, *(ii)* probabilistic systems containing transitions labelled by action, probabilities, and/or (action, probability) pairs, which allows to minimise discrete time Markov chains (and various probabilistic transition systems) modulo lumpability (respectively probabilistic bisimulation), and *(iii)* Markov decision processes [33], which can be minimised modulo lumpability. We refer to the Bcg_Min manual page[1] for a detailed description of the features of Bcg_Min.

2.4 Compositional Generation of Interactive Markov Chains

Both functional verification and performance evaluation are confronted to the well-known *state explosion* problem, which occurs when state spaces or Markov chains become too large for being generated exhaustively. As regards functional verification, the Cadp tool set provides various strategies to address the state explosion problem, one of these being *compositional generation* (also known as *compositional minimisation*), see e.g. [16]. This approach consists in dividing the system into a set of concurrent processes, then generating the Ltss corresponding to these processes, minimising these Ltss using an equivalence relation (such as strong or branching bisimulation), and finally combining the minimised Ltss in parallel so as to generate the Lts of the whole system.

Compositional generation has been adapted to performance evaluation, both in the context of Ctmcs, where bisimulation is known to agree with the notion of lumpability [26], and in the context of Lotos and Imcs [23]. Compared to [23], our approach is novel in several respects:

- Using the Bcg_Min tool, which did not exist at the time of [23], we are now able to minimise Imcs effectively.
- To compute the Imc corresponding to a set of Imcs combined together using Lotos parallel composition operators (without synchronisation on delay transitions as mentioned above), we resort to the Exp.Open tool[2] developed by Laurent Mounier. The Exp.Open tool is also used to combine a Lotos specification with a set of Imcs expressing delays to be incorporated in a constraint-oriented style.
- Finally, we take advantage of Svl [12,31], a new scripting language for compositional and on-the-fly verification. Svl provides a high-level interface to all Cadp tools (including Cæsar, Cæsar.adt, Bcg_Labels, Bcg_Min, Exp.Open, etc.), thus enabling an easy description and execution of complex performance studies.

2.5 Numerical Analysis of Interactive Markov Chains

After constructing a minimised Imc, the last step of our methodology consists in applying performance evaluation analysis algorithms, so as to compute interesting performance metrics out of the model. To analyse the Imc models, one can use either model checking algorithms, such as those implemented in Etmcc [22] or Prism [30][3], or more standard analysis algorithms for Ctmcs, such as

[1] http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html

[2] http://www.inrialpes.fr/vasy/cadp/man/exp.open.html

[3] Imc models containing nondeterminism require rather involved algorithms as described in [33,9]

those available in the TippTool [21] developed at the University of Erlangen-Nuremberg. Note however that in general the Imc models contain nondeterminism, and thus one needs rather involved algorithms as described in [33,9].

We decided to stick to standard Ctmc analysis algorithms. A connection of the TippTool analysis engine to the Bcg format was developed, which enables the use of the TippTool to carry out analysis of (moderate size) Imcs generated using Cadp. This connection allows to study the time dependent (*transient*) behaviour, as well as the long run average (*steady-state*) behaviour of a model. Transient analysis uses a numerical algorithm known as *uniformisation*, while steady-state analysis is carried out using either the *power*, *Gauss-Seidel* or *SOR* method; see [34] for a thorough introduction to these algorithms.

3 The SCSI-2 Bus Arbitration Protocol

To illustrate our approach, we consider an industrial case-study brought to our attention by Massimo Zendri while he was working in the Vasy team. This case-study is about a storage system developed by Bull in the early 90's. This system consists of at most 8 *devices* (7 hard disks and one disk controller) connected by a bus implementing the Scsi-2 (*Small Computer System Interface*) standard [2]. Each device is assigned a unique Scsi number between 0 and 7.

During the testing phase, Bull engineers discovered potential starvation problems for disks having Scsi numbers smaller than the Scsi number of the disk controller. Practically, this problem was solved by instructing system manufacturers to install the controller with the Scsi number 0 systematically. In parallel, research was initiated to understand the issue. This problem was first modelled by Massimo Zendri, who developed a Markovian queueing model to study performance issues [37]. Later, the functional aspects of the Scsi-2 bus arbitration protocol were formalised in Lotos by Hubert Garavel, with an emphasis on modelling arbitration concisely using Lotos multiway rendezvous. This Lotos specification[4] served as a basis for model-checking verification by Radu Mateescu (thus, enabling to discover the starvation problem mechanically) and automated test generation by Solofo Ramangalahy. See also [5] for a discussion of fairness issues in the Scsi-3 bus arbitration protocol. In the present paper, we complement these functional verification efforts by enhancing the Lotos model so as to study performance issues.

In the Scsi-2 system, the controller can send randomly to the disk n a message "`CMD !`n" (*command*) indicating a transfer request (read/write a block of data from/to the disk). After processing this command, the disk sends back to the controller a message "`REC !`n" (*reconnect*). We do not model the detailed contents (e.g., type or data) of these messages. The `CMD` and `REC` messages are stored in eight-place Fifo queues (see Figure 1). Since we abstract from the message contents, it is sufficient to model these queues as simple counters.

Arbitration mechanism. The `CMD` and `REC` messages circulate on the Scsi bus, which is shared by all devices. To avoid access conflicts, the Scsi-2 standard

[4] See `http://www.inrialpes.fr/vasy/verdon` for details

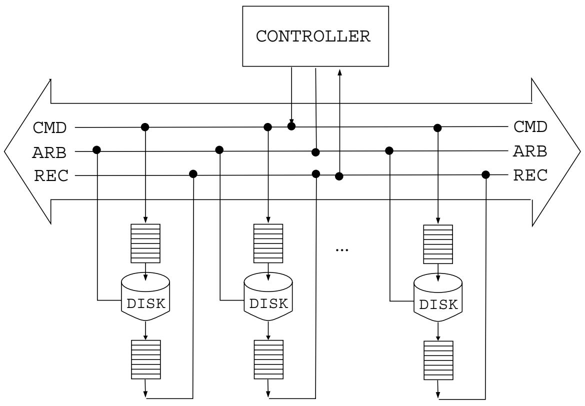

Fig. 1. Architecture of the Scsi-2 system.

defines a bus arbitration policy ensuring that at any time at most one device is allowed to access the bus. Before sending a message over the bus, each device must first request and obtain exclusive bus access. Arbitration is based on fixed priorities: if several devices want to access the bus simultaneously, the device with the highest Scsi number is granted access. Arbitration is also decentralised: contrary to other bus protocols (e.g., Pci) there is no centralised arbiter responsible for granting bus access. To ensure exclusive access in a distributed way, the arbitration mechanism is physically implemented by eight electrical wires, the voltage level of which (high or low) can be consulted by all devices. Each wire is owned by a particular device, and is set to high voltage when this device requests bus access. Before using the bus, each device examines the eight wires' voltage level during a certain amount of time (the *arbitration period*) to ensure that no other device with a higher Scsi number has its wire set to high voltage.

Modelling the Scsi-2 arbitration policy in a precise, concise, yet understandable way is a challenge, especially for languages providing binary communication paradigms only (such as Fifo queues, remote procedure calls, or binary synchronisations).

Yet, this problem can be solved elegantly using the advanced features of Lotos (namely, multiway rendezvous with value negotiation based on pattern-matching). Assuming that the arbitration period is short enough, arbitration can be modelled by a single, eight-party rendezvous between all devices on a gate named `ARB`. During every arbitration period, all devices must synchronise to indicate whether they request bus access or not. Syntactically, each device must propose an action of the form "`ARB ?W:WIRE [C`$_n$`(W,`n`)]`", where n is the Scsi number of the device, where variable `W` of type `WIRE` is an eight-tuple $(w_0, w_1, \ldots, w_7)$ of booleans corresponding to the voltage levels on the wires[5], and where predi-

[5] The boolean values *false* and *true* correspond to low and high voltage, respectively.

cate $\mathtt{C}_n(\mathtt{W}, n)$ belongs to a set of three possible constraints relating W and n. These three constraints are: *(i)* the constraint $\mathtt{C\_PASS}(\mathtt{W}, n) := \neg w_n$ is true iff device n does not request the bus; *(ii)* the constraint $\mathtt{C\_WIN}(\mathtt{W}, n) := w_n \ \wedge \ \neg \bigvee_{i=n+1}^{i=7} w_i$ is true iff device n requests the bus and succeeds to be the highest priority competitor; *(iii)* the constraint $\mathtt{C\_LOSS}(\mathtt{W}, n) := w_n \ \wedge \ \bigvee_{i=n+1}^{i=7} w_i$ is true iff device n requests the bus but fails to gain access. When the eight devices synchronise together on gate ARB, their individual, distributed constraints are combined into a logical conjunction $\bigwedge_{i=0}^{i=7} \mathtt{C}_i(\mathtt{W}, i)$, which determines a unique solution W agreed by all the devices unanimously.

Disk devices. Each disk is described as an instance of a generic LOTOS process (noted DISK) parameterised by the SCSI number N, the number L of CMD messages waiting to be processed in the disk's input FIFO queue (initially, L = 0), and by a boolean variable READY which is true iff the device has processed a CMD message and is ready to send the result back to the controller (initially, READY = *false*). The behaviour of the DISK process is a nondeterministic selection between five branches: *(i)* the disk may receive a CMD message and increment L (a flow control mechanism implemented in the controller avoids overflows in the disks' input queues); *(ii)* if the disk is not ready, it may take part in the arbitration mechanism without requesting the bus, which enables lower priority devices to access the bus; *(iii)* if the disk is not ready and if its input queue is not empty, it may process a command stored in the queue (which takes a Markov delay noted "MU !N"), then decrement L and become ready; *(iv)* and *(v)* if the disk is ready, it requests the bus repeatedly until it is granted; once successful, it sends a corresponding REC message and returns to its non-ready state.

```
process DISK [ARB, CMD, REC, MU] (N:NUM, L:NAT, READY:BOOL):noexit :=
   CMD !N;
      DISK [ARB, CMD, REC, MU] (N, L+1, READY)
   []
   ARB ?W:WIRE [not (READY) and C_PASS (W, N)];
      DISK [ARB, CMD, REC, MU] (N, L, READY)
   []
   [not (READY) and (L > 0)] ->
      MU !N; (* Markov delay inserted here *)
         DISK [ARB, CMD, REC, MU] (N, L-1, true)
   []
   ARB ?W:WIRE [READY and C_LOSS (W, N)];
      DISK [ARB, CMD, REC, MU] (N, L, READY)
   []
   ARB ?W:WIRE [READY and C_WIN (W, N)];
      REC !N;
         DISK [ARB, CMD, REC, MU] (N, L, false)
endproc
```

Controller device. The controller is described by a LOTOS process (noted CONTROLLER) parameterised by the SCSI number NC of the controller and by two variables PENDING and T. PENDING contains the SCSI number of the disk to which the controller has to send a CMD message (initially, PENDING = NC, which means that the controller is idle). T is a table (i.e., an array) used for flow control, so as to avoid overflow of the disks' input queues. The n-th element of T (noted "VAL (T, n)'', where n is a SCSI number different from NC) stores the number of commands waiting to be processed by disk n, i.e., the difference between the number of "CMD !n" messages sent and the number of "REC !n" messages received by the controller. ZERO denotes the initial value of the table, with all elements equal to 0. INCR (T, n) and DECR (T, n) denote the table T in which the n-th element is incremented or decremented, respectively.

As with the disk, the behaviour of the CONTROLLER process is a selection between five branches: *(i)* if the controller is idle, it may take part in the arbitration mechanism without requesting the bus; *(ii)* if the controller is idle, it may also select (nondeterministically) some disk N with less than eight unprocessed commands and assign N to PENDING; in practice, this selection is triggered by a transfer request sent to the controller by its external environment; we introduce a Markov delay noted "LAMBDA !N" in order to model the load stress imposed on the controller; *(iii)* and *(iv)* if the controller is not idle, it requests the bus repeatedly until it is granted; once successful, it sends a CMD message to the disk indicated by PENDING, then increments T accordingly and returns to its idle state; *(v)* the controller may receive REC messages and decrement T accordingly.

```
process CONTROLLER [ARB, CMD, REC, LAMBDA] (NC:NUM, PENDING:NUM,
                                             T:TABLE) : noexit :=
   ARB ?W:WIRE [(PENDING == NC) and C_PASS (W, NC)];
      CONTROLLER [ARB, CMD, REC, LAMBDA] (NC, PENDING, T)
   []
   (
   choice N:NUM []
      [(PENDING == NC) and (N <> NC)] ->
         [VAL (T, N) < 8] ->
            LAMBDA !N; (* Markov delay inserted here *)
               CONTROLLER [ARB, CMD, REC, LAMBDA] (NC, N, T)
   )
   []
   ARB ?W:WIRE [(PENDING <> NC) and C_LOSS (W, NC)];
      CONTROLLER [ARB, CMD, REC, LAMBDA] (NC, PENDING, T)
   []
   ARB ?W:WIRE [(PENDING <> NC) and C_WIN (W, NC)];
      CMD !PENDING;
         CONTROLLER [ARB, CMD, REC, LAMBDA] (NC, NC, INCR (T, PENDING))
   []
   REC ?N:NUM [N <> NC];
      CONTROLLER [ARB, CMD, REC, LAMBDA] (NC, PENDING, DECR (T, N))
endproc
```

System architecture. The architecture of the SCSI-2 system is described by composing in parallel the seven disk processes and the controller process. All these processes synchronise together using an eight-way rendezvous on the ARB gate. The disks communicate with the controller using binary rendezvous on gates CMD and REC. Although the seven disks are competing with each other for achieving a rendezvous on gates CMD and REC with the controller, the "$!n$" parameters associated to these gates allow to identify the corresponding disk. Finally, as explained in Section 2.2, the MU and LAMBDA gates must not be synchronised.

```
(
DISK [ARB, CMD, REC, MU] (0, 0, false)
|[ARB]|
DISK [ARB, CMD, REC, MU] (1, 0, false)
|[ARB]|
  ...
|[ARB]|
DISK [ARB, CMD, REC, MU] (6, 0, false)
)
|[ARB, CMD, REC]|
CONTROLLER [ARB, CMD, REC, LAMBDA] (7, 7, ZERO)
```

4 Performance Model Aspects

The SCSI-2 specification as introduced above incorporates already some timing parameters, namely the Markov delays LAMBDA and MU. This section motivates the timing characteristics of the model. It further discusses the approach followed to generate and analyse the model numerically, together with some interesting performance figures we obtained.

4.1 SCSI-2 Timing Parameters

Based on the timing parameters given in definition of the SCSI-2 architecture, we identified three parameters as most relevant for a performance study.

– The Markov delay LAMBDA put in the controller models the load (transfer requests issued by the controller) that stimulates the whole SCSI-2 system. It is the main parameter we vary in our experiments.
– The Markov delay MU put in the disk corresponds to the *disk servicing time*, i.e., the time needed by an individual disk to fetch or store the requested data. The mean servicing time depends on the size of the data blocks to be transferred, and also varies from one disk manufacturer to another. Its value ranges from 1500 μs to about 4500 μs [37].
– Finally, the *bus inter-arbitration time* (or *bus delay*, for short) determines the delay between two consecutive bus arbitration periods. This delay is minimally 2.5 μs and depends on the amount of data transmitted on the bus after an arbitration.

To incorporate the bus delay into the SCSI specification, we use the constraint-oriented style mentioned earlier. As the bus delay elapses between any two consecutive ARB actions, it will be incorporated by running the SCSI system in parallel with an additional, very simple process BUS, which forces any two consecutive ARB actions to be separated by a Markov delay NU:

```
process BUS [ARB, NU]:noexit :=
  ARB; NU; BUS [ARB, NU]
endproc
```

Both the Scsi system and the BUS process are synchronised on gate ARB. Note that this approach allows one to experiment with different, phase-type distributed delays in a flexible way, such as with an Erlang-5 distributed delay:

```
process BUS_5 [ARB, NU]:noexit :=
  ARB; NU; NU; NU; NU; NU; BUS_5 [ARB, NU]
endproc
```

We carried out several experiments with such delays, and found that as long as the mean value of the distributions used stays unchanged, the influence of the distributions on the numerical results is marginal. As regards the LAMBDA and MU delays, experimenting with other distributions is not so straightforward, because any change in the distribution implies a change in the Lotos specification, and hence a proof obligation that the functional behaviour is still as intended. The constraint-oriented style reliefs this burden, since it preserves the functional behaviour: the resulting Lts obtained after parallel composition is branching bisimilar to the original one provided that the Markov delays are hidden (i.e., renamed to τ) [19].

4.2 Performance Results

Among the studies we performed, we here focus on the behaviour of a Scsi-2 system under heavy load, since the system exhibits some interesting aspects of unfairness in extreme situations. Note that due to the distributed priority mechanism governing the bus arbitration protocol, the system can not be expected to behave perfectly fair under all circumstances.

We study a system with 3 disks. The load imposed on the system varies between 10 and 800 requests per seconds and per disk. Unless otherwise stated, we assume the average servicing time of the disks to be 2,500 μs, and the bus delay to range between 2.5 μs and 2,500 μs.

First, we study a system in which the controller is assigned the Scsi number 7, and observe the throughput of each disk under increasing load. The resulting throughputs are plotted in Figure 2, for four different bus delay parameters. The left plot shows the high priority disk 2, and the right one shows the low priority disk 0. We observe that the bus bandwidth is shared in a load dependent way, and we further observe that the higher the bus delay, the lower the throughputs of the disks. Interestingly, the lower disks' throughputs may collapse if the bus delay is very long and load is heavy. The high priority disk does not exhibit such a phenomenon. This reveals the unfairness of the arbitration mechanism.

To study this phenomenon further, we analyse the effect of the controller Scsi number on the throughputs of the high and low priority disk. Figure 3 plots the throughputs of the low and high priority disks under extreme bus delays. If the controller is in the highest position (Scsi number 7), we find back one of the scenarios studied in Figure 2: the high priority disk dominates the low priority disk, and makes the throughput of the latter collapse. If on the other hand, the

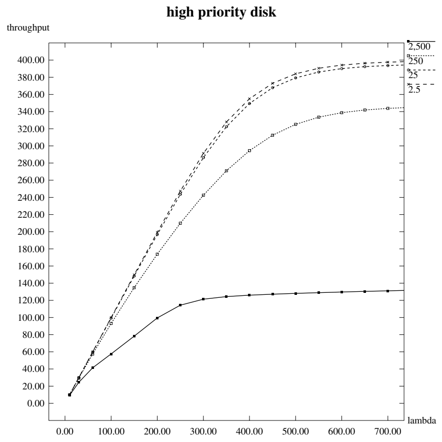

Fig. 2. Throughput of disk 2 (left) and disk 0 (right) under increasing load with bus delay ranging from 2.5 μs (dashed) to 2.5 ms (solid), and controller having number 7.

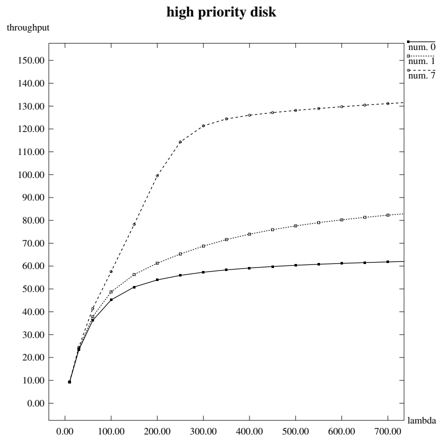

Fig. 3. Throughput of high priority disk (left) and low priority disk (right) under increasing load with bus delay 2.5 ms, and controller having lowest (solid), middle, and highest (dashed) number.

controller is in the lowest position (Scsi number 0), the achieved throughputs of high and low priority disk are rather balanced, and in particular the low priority throughput does not degrade nor collapse.

This study allows us to draw the conclusion that assigning Scsi number 0 to the controller makes the system balanced. Otherwise, disks in a position lower than the controller are disfavoured. This conclusion is in line with the experimental observations made by the Bull engineers; our studies allow a quantification of the influence of the disk position on the throughput.

4.3 An SVL Session with CADP

This section discusses how the Markov chains under study are generated from the LOTOS specification using the CADP toolbox. To explain how we proceed, we list below the main fragment of the SVL-script used to distill the lumped Markov chain used for the plots in Figure 2.

```
"scsi.bcg" = branching reduction of                                    (1)
                total rename "ARB !.*" -> ARB  in
                   hide CMD, REC in
                      "scsi.lotos";


"model.bcg" = hide all but LAMBDA, MU, NU in                           (2)
                ("scsi.bcg" |[ARB]| "erlang.lotos":BUS [ARB, NU]);




% for SPEED in .4 2 4 40 400                                           (3)
% do
   % for LOAD in .01 .03 .06 .1 .15 .2 .25 .3 .35 \
   %             .4 .45 .5 .55 .6 .65 .7 .75 .8
   % do
      % BCG_MIN_OPTIONS="-rate"
        "res-$SPEED.bcg" = branching reduction with bcg_min of         (4)
                              total rename "NU" -> "rate $SPEED",
                                 "MU !0" -> "DISK_L; rate .4",
                                 "MU !1" -> "DISK_M; rate .4",
                                 "MU !2" -> "DISK_H; rate .4",
                                 "LAMBDA !.*" -> "rate $LOAD" in
                                    "model.bcg";
        % seidel -v $LOAD "res-$SPEED.bcg"                             (5)
     % done
% done
```

During step (1) the transition system of the SCSI specification is generated, the CMD and REC gates are hidden as they are not needed in subsequent processing, and the arbitration events are uniformly renamed into a new action named ARB. Then, the resulting state space is minimised according to branching bisimulation, and stored in a file named "scsi.bcg".

Step (2) incorporates the bus delay via the process BUS [ARB, NU] taken from file "erlang.lotos". Afterwards, all gates are hidden, except those corresponding to Markov delays (i.e., LAMBDA, MU, and NU). The result is stored in file "model.bcg".

Step (3) initiates two nested loops that compute a two-dimensional matrix of performance results. The outer loop varies the SPEED parameter, which is the inverse of the bus delay expressed in milliseconds, ranging from $1/2.5$ μs to $1/2.5$ ms. The inner loop varies the LOAD parameter, imposing between 0.01 and 0.8 requests per millisecond on each disk.

Step (4) instantiates, for each pair (SPEED, LOAD), the Markov delays LAMBDA, MU, and NU present in file "model.bcg" with concrete values. The resulting IMC is then minimised using Bcg_Min according to stochastic branching bisimulation, which eliminates nondeterminism. This results in a Markov chain stored in file "`res-$SPEED.bcg`".

Step (5) calls the TippTool solver `seidel`, a numerical solution engine implementing the Gauss-Seidel linear equation solver for Markov chains. It computes the equilibrium (steady-state) probabilities for the states of the Markov chain. From these probabilities, `seidel` calculates the transition throughputs for each Markov delay marked with a distinguished label. These labels have been incorporated into the transition system in step (4); they indicate a high (`DISK_H`), medium (`DISK_M`), or low (`DISK_L`) priority disk being active.

The largest state space produced during the execution of the Svl script is the Lts generated from "`scsi.lotos`", which has 56,169 states and 154,752 transitions. The size of the Markov chains solved (i.e., files "`res-*.bcg`") ranges from 10,666 to 17,852 states.

5 Concluding Remarks

This paper has presented a practical methodology for studying the performance of a concurrent system, starting from an already verified functional specification of this system. Compared to prior works on stochastic Petri nets and stochastic process algebras, our approach is original in several respects:

– We have chosen not to design a new formalism to model stochastic systems, because the effort required to develop appropriate software tools would have been very high. Instead, we reuse a non-stochastic process algebra (Lotos), which we adapt to the stochastic framework by introducing a few additional operators (such as relabelling, restriction, time constraints, and minimisation). This approach provides the user with a high-level language (Lotos) to describe both control and data aspects (contrary to, e.g., the TippTool, which only supports a subset of Lotos without data structures). Furthermore, existing Lotos tools can be used to perform functional verification before undertaking performance analysis.
– To translate Lotos specifications into labelled transition systems, we use the Cæsar.adt and Cæsar compilers of the Cadp tool set. To perform relabelling, we also reuse an existing Cadp tool, Bcg_Labels. Our major development effort is Bcg_Min, an efficient tool implementing several minimisation algorithms for ordinary, stochastic, and probabilistic transition systems. Bcg_Min plays a central role in connecting the Cadp tools to the stochastic setting, and supports the compositional approach proposed in [23], in which concurrent processes are generated, then minimised separately so as to handle large state spaces.
– In order to automate the performance studies, in which stochastic parameters are varied in multiple dimensions, we take advantage of the scripting language Svl. Originally developed for compositional verification of non-stochastic systems, Svl is also useful in the stochastic settings, and provides convenient means to integrate the various tools transparently.

We have presented an application of these principles to an industrial problem: the Scsi-2 bus arbitration protocol, which we managed to model elegantly using the expressiveness of Lotos multiway negotiated rendezvous. After verifying the functional correctness of the Lotos specification using the Cadp tools, we turned this specification into a performance model, which we analysed automatically by combining the Cadp tools and the solution engine of the TippTool. This performance study allowed us to quantify the unfairness of the Scsi-2 bus arbitration protocol, and to show how the respective disk thoughputs depend on the Scsi number assigned to the controller. These results are in line with the experimental observations on the real Scsi-2 disk system.

As regards future work, more efforts are foreseen on the model solution side. So far, we are resorting to the TippTool, but in a near future we shall investigate model checking approaches to Markov models, notably by linking the Etmcc Markov chain model checker [22] to Cadp. Also, Mtbdd- or Kronecker-based Markov chain representations [30,8] are promising directions to enable the analysis of even larger models, in combination with our compositional approach.

Acknowledgements. We are grateful to Massimo Zendri for bringing the Scsi-2 example to our attention, and to Moëz Cherif (formerly at Inria/Vasy) for helping us to develop the Bcg_Min tool. We are also grateful to Frédéric Lang (Inria/Vasy) and the anonymous referees for their remarks about this paper.

References

1. A. Marsan, G. Balbo, and G. Conte. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Trans. on Comp. Sys.*, 2(2), 1984.
2. ANSI. Small Computer System Interface-2. Standard X3.131-1994, American National Standards Institute, 1994.
3. M. Bernardo, W.R. Cleaveland, S.T. Sims, and W.J. Stewart. TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems. In *Proc. FORTE'98*, IFIP, North-Holland, 1998.
4. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Th. Comp. Sci.*, 202:1–54, 1998.
5. D. Bert. *Preuve de propriétés d'équité en B : Preuve de l'algorithme d'arbitrage du bus SCSI-3.* In *Proc. AFADL'2001* (Nancy, France), pages 221–241, June 2001.
6. L. Blair, G. Blair, and A. Andersen. Separating Functional Behaviour and Performance Constraints: Aspect-Oriented Specification. Technical Report MPG-98-07, Computing Department, Lancaster University, 1998.
7. T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Comp. Netw. and ISDN Sys.*, 14(1):25–59, 1988.
8. P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of memory-efficient Kronecker operations with applications to the solution of Markov models. *INFORMS J. on Comp.*, 13(3):203–222, 2000.
9. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proc. Symp. on Logic in Computer Science*, 1998.

10. D. Ferrari. Considerations on the Insularity of Performance Evaluation. *IEEE Trans. on Softw. Eng.*, SE–12(6):678–683, June 1986.
11. H. Garavel. Compilation of LOTOS abstract data types. In *Proc. FORTE'89*, pages 147–162. IFIP, North-Holland, 1989.
12. H. Garavel and F. Lang. SVL: A Scripting Language for Compositional Verification. In *Proc. FORTE'2001*, pages 377–392. IFIP, Kluwer Academic, 2001. Full version available as INRIA Research Report RR-4223.
13. H. Garavel and J. Sifakis. Compilation and verification of LOTOS specifications. In *Proc. PSTV'90*, pages 379–394. IFIP, North-Holland, 1990.
14. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling. In *Proc. TOOLS'94*, 1994.
15. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Tutorial Proc. PERFORMANCE '93.* Springer, LNCS 729, 1993.
16. S. Graf, B. Steffen, and G. Luettgen. Compositional Minimization of Finite State Systems. *Formal Asp. of Comp.*, 8(5):607–616, 1996.
17. H. Garavel, F. Lang, and R. Mateescu. An Overview of CADP 2001. INRIA Technical Report RT-254, December 2001.
18. J.F. Groote and J. van de Pol. State space reduction using partial τ-confluence. In *Proc. MFCS'2000*, pages 383–393. Springer, LNCS 1893, 2000.
19. H. Hermanns. *Interactive Markov Chains.* PhD thesis, Universität Erlangen-Nürnberg, 1998. revised version to appear as Springer LNCS monograph.
20. H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Th. Comp. Sci.*, 274(1-2):43–87, 2002.
21. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Perf. Eval.*, 39(1-4):5–35, January 2000.
22. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov Chain Model Checker. In *Proc. TACAS'2000*, pages 347–362, Springer, LNCS 1785, 2000.
23. H. Hermanns and J.P. Katoen. Automated compositional Markov chain generation for a plain-old telephony system. *Sci. of Comp. Prog.*, 36(1):97–127, 2000.
24. O. Hjiej, A. Benzekri, and A. Valderruten. From Annotated LOTOS specifications to Queueing Networks: Automating Performance Models Derivation. Decentralized and Distributed Systems (North Holland), 1993.
25. J. Hillston. The Nature of Synchronisation. In *Proc. PAPM'94*, Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg. pages 51–70, 1994.
26. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
27. ISO/IEC. LOTOS — A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. International Standard 8807, ISO - Information Processing Systems - Open Systems Interconnection, 1988.
28. ISO/IEC. Enhancements to LOTOS (E-LOTOS). International Standard 15437:2001, ISO - Information Technology, 2001.
29. A. Marsan, A. Bianco, L. Ciminiera, R. Sisto, and A. Valenzano. A LOTOS Extension for the Performance Analysis of Distributed Systems. IEEE/ACM Trans. on Networking, 2(2), 151–164, 1994.
30. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic Symbolic Model Checking with PRISM: A Hybrid Approach. In *Proc. TACAS'2002*, pages 52–66, 2002, Springer LNCS 2280.

31. F. Lang. Compositional Verification using SVL Scripts. In *Proc. TACAS'2002*, pages 465–469, 2002, Springer LNCS 2280.
32. M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models–An Algorithmic Approach.* The Johns Hopkins University Press, 1981.
33. M.L. Puterman. *Markov Decision Processes.* John Wiley, 1994.
34. W.J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.
35. K. J. Turner, editor. *Using Formal Description Techniques – An Introduction to ESTELLE, LOTOS, and SDL.* John Wiley, 1993.
36. C. Vissers, G. Scollo, M. van Sinderen, and E. Brinksma. Specification styles in distributed systems design and verification. *Th. Comp. Sci.*, 89(1):179–206, 1991.
37. M. Zendri. *Studio ed implementazione di un modello del bus SCSI.* Laurea thesis, Politecnico di Milano, Facoltà di Ingegneria, Dip. di Elettronica, 1992.