4th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001)
Magdeburg, Germany - May, 2-4, 2001

Composing Real-Time Objects: A Case for Petri Nets and Girard's Linear Logic

François Taïani , Mario Paludetto
LAAS-CNRS, 7 av. du Colonel Roche,
F-31077 Toulouse CEDEX 4, France
Tel.: +33 / 561 336 260, Fax: +33 / 561 336 936
{francois.taiani, mario.paludetto}@laas.fr

Jérôme Delatour
ESEO, 4 rue Merlet de la Boulaye,
BP 926, F-49009 Angers CEDEX 01, France
Tel.: +33 / 241 866 790, Fax : +33 / 241 879 927
jerome.delatour@eseo.fr

This article was presented at the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2001) held in Magdeburg, Germany, on May, 2 - 4, 2001. It has been published in the proceedings of the aforementioned conference under the following reference:

F.Taïani , M. Paludetto, J. Delatour , *Composing real-time Objects: a Case for Petri Nets and Girard's Linear Logic*, Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'2001), Magdeburg (Germany), 2-4 Mai 2001, pp.298-305

4th IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing (ISORC 2001)
Magdeburg, Germany - May, 2-4, 2001

Composing Real-Time Objects: A Case for Petri Nets and Girard's Linear Logic

François Taïani , Mario Paludetto
LAAS-CNRS, 7 av. du Colonel Roche,
F-31077 Toulouse CEDEX 4, France
Tel.: +33 / 561 336 260, Fax: +33 / 561 336 936
{francois.taiani, mario.paludetto}@laas.fr

Jérôme Delatour
ESEO, 4 rue Merlet de la Boulaye,
BP 926, F-49009 Angers CEDEX 01, France
Tel.: +33 / 241 866 790, Fax : +33 / 241 879 927
jerome.delatour@eseo.fr

Abstract

Object and component technology play an ever-increasing role in the development of real-time distributed applications. Those systems are characterized by the fact that the temporal compatibility of the different objects that are brought together is a condition for success.

In this paper, we propose an approach to validate the interoperability of object interfaces with respect to their temporal properties. This approach is based on a recent execution time calculation technique dedicated to concurrent environments. In this article we propose a simpler computation framework for this technique, and we show how it can quite advantageously be adapted to distributed real time OO systems.

1. Introduction

The benefits of object orientation for the design of complex systems are now quite unanimously recognized. Initiatives such as CORBA (Common Object Request Broker Architecture) and UML (Unified Modeling Language) provide evidence of the ever-growing acceptance of the object paradigm, even in such fields as organization and workflow management [*Corba00, *UML99].

Originally issued from AI and simulation, the object concepts were first used in the industry for the development of data-processing systems. Those systems either had no concurrency, or contained only loosely synchronized components. Because of the strong influence of those sequential systems on the evolution of object-orientation, most of the early object oriented languages did not integrate any concurrency primitives.

One of the important trends of the last decade was to *bring back* objects to the design of distributed applications, in which concurrency management stays in front position [Selic94]. Since that turn many approaches have been proposed to extend the OO (Object Oriented) paradigm with concurrency mechanisms. Actor Languages and Meta Object Protocols are some of them [BriGue96].

Today, commercial OO frameworks are available which specifically target the development of distributed systems, such as CORBA and DCOM (Distributed Common Object Model). Those products have given rise to an increasing number of real life applications that typically run in a timely constrained environment. One of today's research challenges consists in improving the real-time services of OO middleware, such as scheduling, time predictability, and Quality of Service [SchKu00, Polz+99]. Simultaneously, and as the range of temporal features supported by OO systems increases, the question arises as how these systems should be validated.

In this paper we try to address a part of this validation problem by considering the temporal expectations an object may impose on its environment. Our idea is that given an abstract description of the objects composing a system, a development environment should be able to check if and under which conditions the temporal constraints of each object can be met.

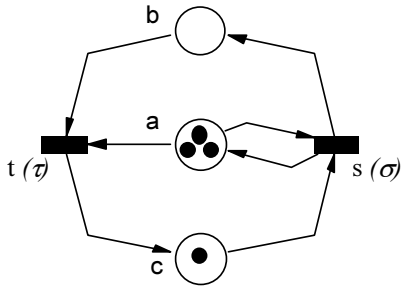
Our line of attack consists in adapting to OO software a duration computation technique that was recently proposed in the field of plant production and multimedia scheduling [Prad+00, Prad+99, GiPrV97]. This technique associates Petri nets with Girard's Linear Logic. (Except for its name, Girard's Linear Logic has no connections with the Linear Temporal Logic (LTL) [Lam80], commonly used in Model-Checking [Merz00]). One of its main benefits lies in its ability to handle unsynchronized and distributed local partial states, which appears to be a crucial point to limit the state explosion. This technique furthermore produces symbolic variable expressions, which can for instance be used to customize a COTS-based architecture.

The remainder of the paper is organized as follows: First, in section 2, we introduce linear logic and the execution time calculation technique that was originally developed by R. Valette, and B. Pradin-Chézalviel. In the

same section, we propose a notable simplification of this technique that we have developed. Then in section 3, we show how linear logic can be applied to object oriented development. We come back on the notion of timely constrained object interfaces, and show on a case study how the method we propose can be applied to the calculation of Worst Case Execution Times (WCET) in distributed real-time object oriented systems. Eventually, we conclude on our work in section 4, and present the future developments we plan.

2. On Girard's Linear Logic, PNs and Time

Linear logic was introduced by J.Y. Girard at the end of the 80's [Girard87]. It is a kind of mutant form of the classical boolean logic in which *truths get lost once they are used* [Girard90]. Because of this, linear logic is remarkably well suited for the study of distributed systems with shared resources, particularly in conjunction with Petri Nets.



Bracketed Greek letters represent transition durations.

$$A(0)\otimes A(0)\otimes A(0)\otimes C(0)\ ;\quad [A\otimes B \multimap C\ ]_t(\tau),\ [A\otimes C \multimap B\otimes A\ ]_s(\sigma)$$

Figure 1: A bi-state counter and its linear logic interpretation.

In this paper we adopt the Petri Net interpretation of Linear Logic proposed by F. Girault [Girault97, GiPrV97] along with its temporally annotated extension developed by R. Valette, B. Pradin-Chézalviel, and L.A. Künzle [Prad+99]. For lack of space we restrict ourselves to the linear logic fragment needed for our approach. Interested readers are referred to the many articles of Girard [Girard87, Girard90, Girard95], as well as to [Gehlot92] and [Girault97] about the connection between linear logic and Petri nets.

Linear logic manipulates resources called *atoms* (noted **X**, **Y**, **Z**, **…**), which may be accumulated using a *times* operator, noted ⊗. For instance **Z⊗Z⊗X** is a linear proposition that accumulates two atoms of type **Z** and one of type **X**. Petri nets are interpreted in linear logic by associating each place of the net (noted **x**,**y**,**z**...) with one atom type. Tokens that are present in one place are represented by atoms of the type associated with that place, and markings by an accumulation (⊗) of atoms.

Each atom / accumulation is moreover annotated with a time stamp representing its creation date. For instance $\mathbf{Z(d_Z)}$ stands for a token in place **z** that was produced at time $\mathbf{d_Z}$. Using this notation, the initial marking of the net shown on figure 1 will be denoted by:

$$(A\otimes A\otimes A\otimes C)(0)$$
$$\left(\equiv A(0)\otimes A(0)\otimes A(0)\otimes C(0)\right) \qquad (1)$$

System dynamics is expressed in linear logic using the connector ⊸ (*linear implication*). For example in figure 1, **[A⊗B⊸C](τ)** states that one token can be produced in place **c** by consuming one token from **a** and one token from **b**. The annotation **(τ)** means that such a production requires τ time units. The implication **[A⊗B ⊸ C](τ)** is itself a resource, i.e. the possibility of consumption / production expressed by the implication disappears once it has been used. Note that all time stamps and durations may be symbolic.

According to those interpretation rules and for simplicity, we will also name *marking* the linear accumulation of atoms that represent a marking, and *transition* a linear implication with *markings* on both of its sides.

On the net of figure 1, let's now assume that we want to study the evolutions of the system in which each transition **s** and **t** only fires once. To reach that goal, we start with the linear representation of the initial marking, and with one instance of each of the linear implications representing the transitions **s** and **t** (equation *(2)*).

$$A(0)\otimes A(0)\otimes A(0)\otimes C(0)\ ;\quad [A\otimes B \multimap C\ ]_s(\tau),\ [A\otimes C \multimap B\otimes A\ ]_t(\sigma) \qquad (2)$$

The execution time calculation proposed by Valette and, Pradin-Chézalviel was originally defined in the more general context of sequent calculus [Girard95]. Here, the authors offer to replace this powerful but somewhat complex calculus with a lighter rewriting system. The rules we propose work on propositions like the one presented in equation *(2)*. A rewriting step on such a proposition fundamentally corresponds to the firing of a transition. For instance, the consecutive firings of transitions **t** and **s** on equation *(2)* are denoted with the following rewriting steps (noted ⇝):

$A(0)\otimes A(0)\otimes \mathbf{A(0)}\otimes \mathbf{C(0)}$; $[\mathbf{A}\otimes\mathbf{C} \multimap \mathbf{B}\otimes\mathbf{A}\ ]_t(\sigma)$,
$[A\otimes B \multimap C\ ]_s(\tau)$

$\rightsquigarrow$

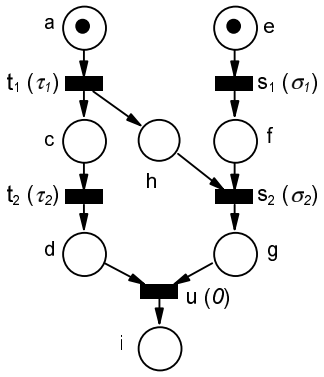
$A(0)\otimes \mathbf{A(0)}, A(\sigma)\otimes \mathbf{B}(\sigma)$; $[\mathbf{A}\otimes\mathbf{B} \multimap \mathbf{C}\ ]_s(\tau)$

$\rightsquigarrow$

$A(0), A(\sigma), C(\sigma+\tau)$

Bold transitions are fired in the next step.
Bold atoms are consumed in the next step.

Figure 2: A rewriting sequence on the net of fig.1



Bracketed Greek letters represent transition durations.

Figure 3: Asynchronous communication between two processes

Note that, in figure 2, transitions **t** and **s** are consumed as they fire. Here, the transient nature of linear implications ($\multimap$) allows us to finely control the depth of our exploration walk.

From those rewritings, we conclude on the result that in the net of figure 1, the time needed from the marking (**a**:3, **c**:1) to the marking (**a**:2, **c**:1) is $\sigma + \tau$. This result is, of course, straightforward due of the intentional simplicity of the chosen net.

In order to better understand the advantages of linear logic, figures 3, and 5 represent nets containing more internal concurrency. Figure 3 shows a process cooperation: The processes **a;c;d** and **e;f;g** run concurrently, and synchronize through message passing on channel **h**. Eventually, they join in a common process **i**. Figure 4 contains the rewriting sequence based on linear logic for that net. This sequence yields a symbolic duration formula that covers all possible traces from marking $\mathbf{A}\otimes\mathbf{E}$ to marking **I**.

The example of figure 3 actually induces a partial order that could not be expressed in a traditional algebra based on sequence and parallel operators (noted ";" and "|" respectively) without extra precedence relations. As a consequence, a duration based on such an algebra expression would be necessarily over-pessimistic in this case.

Abbreviations:
$T_1 = A \multimap C\otimes H$ $S_2 = F\otimes H \multimap G$
$T_2 = C \multimap D$ $U = D\otimes G \multimap I$
$S_1 = E \multimap F$

$^{(0)}$ $A(0)\otimes E(0)$; $T_1(\tau_1), S_1, T_2, S_2, U$
$^{(1)}\rightsquigarrow \mathbf{A(0)}, E(0)$; $\mathbf{T_1(\tau_1)}, S_1, T_2, S_2, U$
$^{(2)}\rightsquigarrow (C\otimes H)(\tau_1), \mathbf{E(0)}$; $\mathbf{S_1(\sigma_1)}, T_2, S_2, U$
$^{(3)}\rightsquigarrow (C\otimes H)(\tau_1), F(\sigma_1)$; $T_2(\tau_2), S_2, U$
$^{(4)}\rightsquigarrow C(\tau_1)\otimes H(\tau_1), F(\sigma_1)$; $T_2(\tau_2), S_2, U$
$^{(5)}\rightsquigarrow \mathbf{C(\tau_1)}, H(\tau_1), F(\sigma_1)$; $\mathbf{T_2(\tau_2)}, S_2, U$
$^{(6)}\rightsquigarrow D(\tau_1+\tau_2), \mathbf{H(\tau_1)}, \mathbf{F(\sigma_1)}$; $\mathbf{S_2(\sigma_2)}, U$
$^{(7)}\rightsquigarrow \mathbf{D(\tau_1+\tau_2)}, \mathbf{G\big(\sigma_2+max(\sigma_1, \tau_1)\big)}$; $\mathbf{U(0)}$
$^{(8)}\rightsquigarrow I\big(max\big[\tau_1+\tau_2, \sigma_2+max(\sigma_1, \tau_1)\big]\big)$; .

Bold transitions are fired in the next step.
Bold atoms are consumed in the next step.

Figure 4: Rewriting Sequence for the net on fig. 3

Nevertheless, in the case of figure 3, linear logic is certainly not the simplest solution to compute an execution time. Actually, a straightforward structural transformation of timing information associated with each transition into a '+', if they build a sequence, and into a *'max'*, if they belong to *parallel* segments, would actually yield the same result.

But such a structural approach falls short with what we would call *multithreaded* nets like the one presented in figure 5. On figure 5, **s** structurally depends on **t** for its firing. However, because of the particular considered marking, **t** and **s** may actually fire in parallel. Linear logic can detect this concurrency, which a plain structural (*max*, +) analysis of the net would not.

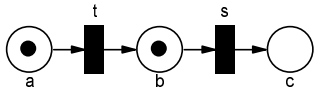


Figure 5: t and s structurally build a sequence. Yet they can fire concurrently.

One central reason why linear logic can uniformly handle such different cases as those of figures 1, 3, and 5 is that it works on local partial states. Both the structure and the dynamic state of a system are taken into account when considering event concurrency [Girault97]. The ability of linear logic to handle such heterogeneous cases as those just presented is what actually clearly differentiates it from other approaches that also compute execution time in concurrent environments, such as

simulation, algebraic analysis, and (*max,* +) event graphs transformations.

More formally, the rewriting sequences we've introduced in figures 4 and 2 are made of propositions of the following form:

<list of markings> ; <list of transitions> *(3)*

The list of transitions of those propositions is a sort of *transition tank*. The rewriting sequence is built by successively taking transitions out of the tank. This occurs either until the tank is empty (for instance on line (8) of figure 4), or until no rewriting rule can be applied any more (dead-lock).

The list of markings in equation *(3)* is what we call a *current step* of the rewriting sequence [Prad+99]. For instance $\mathbf{C}(\tau_1)\otimes\mathbf{H}(\tau_1)$, $\mathbf{F}(\sigma_1)$ is a current step of the sequence on figure 4 (on line 4). The fact that $\mathbf{C}(\tau_1)\otimes\mathbf{H}(\tau_1)$, $\mathbf{F}(\sigma_1)$ appears as a current step on figure 4 does not necessarily imply that the marking $\mathbf{C}\otimes\mathbf{H}\otimes\mathbf{F}$ will happen. Actually, if $\tau_1<\sigma_1$, **F** is produced after **C** was consumed, and the marking $\mathbf{C}\otimes\mathbf{H}\otimes\mathbf{F}$ never occurs. A current step denotes distinct observations of partial states, but says nothing about their possible simultaneity [Prad+99].

The rewriting rules we have developed propagate time according to a *best effort* policy, i.e. transitions which actually *are* fired, are fired at the very moment when the youngest token they consume is produced. This happens for example on figure 4 during the firings of $\mathbf{S_2}$ and **U**.

3. Temporal Verification of OO Interfaces

In section 2, we've presented a symbolic computation technique of trace durations based on Girard's linear logic and Petri nets. Now we offer to apply this technique to the temporal verification of interface compatibility in a real-time OO environment.

Our approach is based on UML / PNO, a development method for OO real time systems that comes from our precedent work. In a first subsection we briefly describe the fundamental constructs and the spirit of UML / PNO. Then we illustrate and discuss our approach with a case study.

3.1. UML / PNO (Petri Net Objects)

UML / PNO [PaDe99, DePa99, DePa98] is a real-time development method based on UML (Unified Modeling Language, [*UML99]), and Petri Net Objects (PNO) [Palu91]. The main idea of UML / PNO consists in describing the different behavioral aspects (specification, design, constraints) of an object using Petri nets. UML / PNO augments UML with Petri nets for specific analysis purposes, a bit like the intermediate formats found in compilers.

UML / PNO does not require the use of a particular kind of Petri nets, which depends on the project progress. Here, as in the previous section, we use t-time Petri nets with plain tokens.

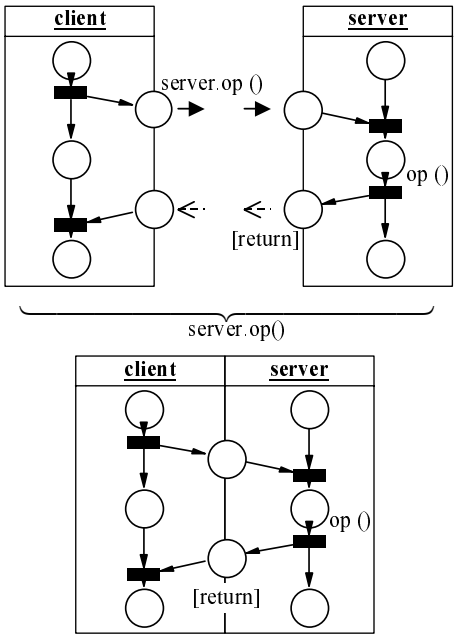


Figure 6: Synchronous method call through asynchronous channels in UML / PNO

In this article we limit object interactions to asynchronous message passing on – possibly – shared channels. Those asynchronous communication channels are modeled with interface places, which are merged when objects are combined together. (See figure 6.)

3.2. Time Constraints and Object Behaviors

In real-time applications, the temporal behavior that an object expects from its environment is a critical feature of that object's behavior, and has to be considered in its interface [Lee00, Selic98, BriGue96, Meyer93]. Taking this point of view, temporal constraints at object boundaries are a sort of extension of the contract-based approach that Meyer initiated. Objects run according to an ongoing *rely / guaranty* principle [HodJo96, Jones96] that keeps inter-object interferences under a *contractual* limit.

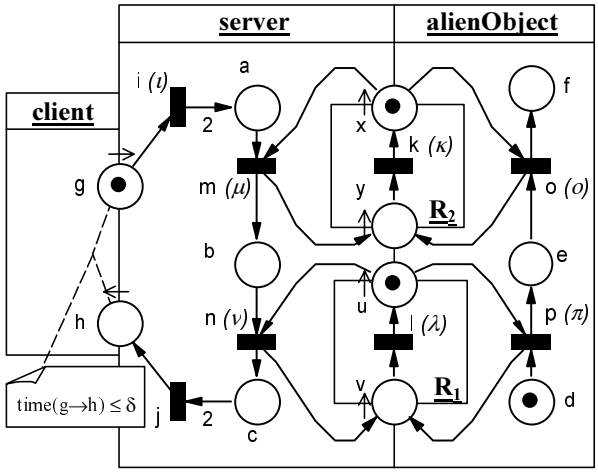
This idea of *temporal contracts* goes along with the ability of many of today's middleware products to explicitly specify precedence, synchronization, or time-out constraints. (For instance, Java, HOOD, and CORBA with their respective keywords **`synchronized`**, **`MTEX`**, and **`SINGLE_THREAD_MODEL`** [Joy+00, Hood95, *Corba00]).

In this paper, we focus more particularly on performance expectations. Our goal is to check that the performance expectations of an object can be met. To do

that, we represent each object's behavior with a Petri net that organizes, and connects together the services the object offers along with the resources and external services it may request. This net also contains local execution times for sequential code segments. Those execution times may either be estimated for components under development, or inferred from the manufacturer's documentation for COTS, or directly computed from the sequential source code if available. (See for instance [GusEr98].)

3.3. Application Example

The figure 7 shows a small distributed object application described with UML/PNO, in which the temporal behaviors of objects have been abstracted by t-time Petri nets.



Bracketed Greek letters represent transition durations.
Interfaces places are indicated with a small side arrow.

Figure 7: Distributed Object Application

In this system, a **client** object (for instance a web-browser) invokes a service from a **server** object (for example a web-server). The **client** interface sets an explicit temporal constraint on the remaining system: The service invoked by the **client** between places **g** and **h** should not require more than δ time units:

$$\text{time}(g \rightarrow h) \leq \delta \qquad (4)$$

We call such a constraint an *Expected Temporal Constraint*, which stresses its precondition nature. Without further information, the service invocation between places **g** and **h** is not guarantied to take less than δ time units. It depends on the environment the **client** object is embedded in. Conversely, the **client** object, and hence the system, can't be expected to behave properly if this condition is not respected.

This service invocation results in the **server** processing concurrently *two* pieces of data (two tokens), which explains the weight 2 between transition **i** and place **a**. The **server** object then successively uses the resources $\mathbf{R_1}$ and $\mathbf{R_2}$ (which may be relational data tables, or other servers) to process both tokens through transitions **m** and **n**.

The **server**'s response to the **client** depends on the availability of resources $\mathbf{R_1}$ and $\mathbf{R_2}$. These resources are shared between the **server** and another object (**alienObject**). Consequently, the execution time of the **client**'s request will depends on the interferences possibly caused by **alienObject** on the object **server** via $\mathbf{R_1}$ and $\mathbf{R_2}$.

Here, some difficulties must be highlighted before we start our analysis. The net resulting from the combination of each object notably differs from those presented in section 2. The net of fig. 7 presents conflicts, which must be considered when applying our rewriting technique. For instance, in the initial marking, transition **p** may fire. **p** also stays in structural conflict with transition **n** through place **u**. If **p** were the only transition to be enabled, **n** would quite obviously be causally dependent on **p** as in fig. 1 of section 2, and the question would not arise. But because transition **i** may fire too, we can't be sure of that, and firing **p** may possibly compel transition **n** to wait for a later token in place **u**, and thus lead to different execution times.

Because we want to compute a Worst Case Execution Time (WCET), we must identify those conflict points, and consider all alternatives whenever they occur. Pradin-Chézalviel, Valette et al. have shown that those conflicts can be automatically identified [Prad+99, Prad+00]. Furthermore, between two conflict situations, the rewriting sequence may be unwounded by applying the rules in any order.

That being said, we may start with the analysis. For lack of space, we will use the following abbreviations for the transitions of the net of figure 7: (figure 8)

$$I =_{def} [G \multimap A \otimes A](\iota)$$
$$J =_{def} [C \otimes C \multimap H](0)$$
$$K =_{def} [Y \multimap X](\kappa)$$
$$L =_{def} [V \multimap U](\lambda)$$
$$M =_{def} [A \otimes X \multimap B \otimes Y](\mu)$$
$$N =_{def} [B \otimes U \multimap C \otimes V](\nu)$$
$$O =_{def} [E \otimes X \multimap F \otimes Y](o)$$
$$P =_{def} [D \otimes U \multimap E \otimes V](\pi)$$

Figure 8: Abbreviations used for the net of fig. 7

According to those abbreviations and to the notation presented in section 2, our analysis of the system of figure 7 starts with the following proposition:

$$(G \otimes X \otimes D \otimes U)(0);\ I,\ 2\times M,\ 2\times N,\ P,\ O,\ 2\times K,\ 2\times L \qquad (5)$$

As explained before, this proposition contains a conflict point: Should **p** use the partial marking **(D⊗U)(0)**

or not? Both alternatives will be explored, and will thus build a classical search tree. Let's first consider the case in which **p** consumes **(D⊗U)(0)**. The rewriting sequence can be developed until either a new conflict point appears or the analysis ends (equation *(6)*).

$^{(0)}$ (G⊗X⊗D⊗U)(0) ; I, 2×M, 2×N, P, O, 2×K, 2×L
$^{(1)}\rightsquigarrow$ **G(0)**, (X⊗D⊗U)(0) ; **I**, 2×M, 2×N, P, O, 2×K, 2×L
$^{(2)}\rightsquigarrow$ (A⊗A)(ι), (X⊗D⊗U)(0) ; 2×M, 2×N, P, O, 2×K, 2×L
$^{(3)}\rightsquigarrow$ (A⊗A)(ι),X(0),**D(0)**,**U(0)** ; 2×M, 2×N, **P**, O, 2×K, 2×L
$^{(4)}\rightsquigarrow$ (A⊗A)(ι),X(0),(E⊗V)(π) ; 2×M, 2×N, O, 2×K, 2×L
$^{(5)}\rightsquigarrow$ (A⊗A)(ι),X(0),E(π),**V(π)** ; 2×M, 2×N, O, 2×K, **2×L**
$^{(6)}\rightsquigarrow$ (A⊗A)(ι),X(0),E(π),U(π+λ) ; 2×M, 2×N, O, 3×K, L
$^{(7)}\rightsquigarrow$ A(ι),A(ι),X(0),E(π),U(π+λ) ; 2×M, 2×N, O, 3×K, L

Bold transitions are fired in the next step.
Bold atoms are consumed in the next step.

(6)

On line 7 of this rewriting sequence (equation *(6)*), we reach a new conflict point. Transitions **o** and **m** stays in conflict about place **X**. Let's first consider the case in which **o** *dominates* **m**. We obtain the sequence shown on equation *(7)*.

$^{(7)}$ A(ι),A(ι),**X(0)**,**E(π)**,U(π+λ) ; 2×M, 2×N, **O**, 2×K, L, J
$^{(8)}\rightsquigarrow$ A(ι),A(ι),(Y⊗F)(π+o),U(π+λ) ; 2×M, 2×N, 2×K, L, J
$^{(9)}\rightsquigarrow$ A(ι),A(ι),**Y(π+o)**,F(π+o),U(π+λ) ; 2×M, 2×N, **2×K**, L, J
$^{(10)}\rightsquigarrow$ **A(ι)**,A(ι),**X(t_{X1}=$_{def}$ π+o+κ)**,U(π+λ),F(π+o) ; **2×M**, …
$^{(11)}\rightsquigarrow$ (B⊗Y)(t_{M1}=$_{def}$ μ+ *max*(ι, t_{X1})),A(ι),U(π+λ),F(…) ; M, …
$^{(12)}\rightsquigarrow$ B(t_{M1}),**Y(t_{M1})**,A(ι),U(π+λ), … ; **M**, 2×N, **K**, L, J
$^{(13)}\rightsquigarrow$ B(t_{M1}),**X(t_{M1}+ κ)**,**A(ι)**,U(π+λ), … ; **M**, 2×N, L, J
$^{(14)}\rightsquigarrow$ B(t_{M1}),(B⊗Y)(t_{M2}=$_{def}$ μ+ *max*(ι , t_{M1}+ κ)),U(π+λ), … ; …
$^{(15)}\rightsquigarrow$ **B(t_{M1})**,B(t_{M2}),**U(π+λ)**,Y(…), … ; **2×N**, L, J
$^{(16)}\rightsquigarrow$ (C⊗V)(t_{C1}=$_{def}$ ν+ *max*(t_{M1}, π+λ)), B(t_{M2}), … ; N, L, J
$^{(17)}\rightsquigarrow$ C(t_{C1}), V(t_{C1}), B(t_{M2}), … ; N, **L**, J
$^{(18)}\rightsquigarrow$ C(t_{C1}), U(t_{C1}+λ), B(t_{M2}), … ; **N**, J
$^{(19)}\rightsquigarrow$ C(t_{C1}), C(ν+ *max*(t_{M2}, t_{C1}+λ)), … ; J
$^{(20)}\rightsquigarrow$ H(*max*(t_{C1}, ν+ *max*(t_{M2}, t_{C1}+λ))), … ;

Bold transitions are fired in the next step.
Bold atoms are consumed in the next step.
Intermediate time variables have been introduced for clarity reasons

(7)

This time the rewriting sequence ends without further conflict and yields the duration formula from equation (8), in where the index *pom²n²* stands for a representative trace of the partial order underlying the search path.

$$t_{pom^2n^2} = max(C1, \nu + max(t_{M2}, t_{C1}+\lambda)) \quad (8)$$
$$= \nu + max(2\mu + \kappa + max(\iota, \pi+o+\kappa), \nu + \lambda + max(\mu+max(\iota,\pi+o+\kappa), \pi+\lambda))$$

For space reasons, we don't reproduce all computation cases corresponding to the other alternatives. As a whole, the search tree contains 6 different paths, which yield six different expressions. The five remaining expressions are given in equation *(9)*. All computations are about the same length as the preceding one, and though they are a bit fastidious to do manually, they remain quite tractable.

$$t_{pm^2n^2o} = \nu + max(2\mu + \kappa + \iota, \nu + \lambda + max(\mu+\iota, \pi+\lambda))$$
$$t_{pmomn^2} = \nu + max(\mu + \kappa + o + max(\mu+\iota+\kappa, \pi), \nu + \lambda + max(\mu+\iota, \pi+\lambda))$$
$$t_{mnpomn} = \nu + \mu + \iota + max(\mu + \kappa + o + max(\kappa, \pi+\nu+\lambda), \nu + 2\lambda + \pi)$$
$$t_{m^2npon} = \nu + \mu + \iota + max(\mu + \kappa, \nu + 2\lambda + \pi)$$
$$t_{m^2n^2po} = \nu + \mu + \iota + max(\mu + \kappa, \nu + \lambda) \quad (9)$$

3.4. Results and Discussion

Coming back on our original requirement,

$$time(g \rightarrow h) \leq \delta,$$

we conclude from equations *(8)* and *(9)* that the temporal constraint expected by object **client** will be satisfied if and only if the following holds:

$$t_{WCET} \leq \delta$$

whereby

$$t_{WCET} =_{def} max(t_{pom^2n^2}, t_{pm^2n^2o}, t_{pmomn^2}, t_{mnpomn}, t_{m^2npon}, t_{m^2n^2po})$$
$$= max(t_{pom^2n^2}, t_{mnpomn})$$

(with the different t_{smthg} defined in equations (8) *and* (9)*)*

(10)

This formula takes all possible cases into account, because the behavior of the system is not known in conflict situations. Due to our lack of knowledge about the system, we thus usually get an over-approximation of the real WCET. Therefore, if we enrich our model with further information, for instance regarding scheduling policies, we can drastically reduce the computation cases. Let's assume that resources $\mathbf{R_1}$ and $\mathbf{R_2}$ are managed with a primitive *first in, first served* scheduling policy. Only two cases must still be considered instead of six, depending on whether $\iota > \pi$ or $\pi > \iota$. The new WCET is then given by the following formula:

if ($\iota > \pi$) $t_{WCET} := t_{pom^2n^2}$

else if ($\pi > \iota$) $t_{WCET} := t_{pm^2n^2o}$

else $t_{WCET} := max(\ t_{pom^2n^2}, t_{pm^2n^2o}\ )$ **//**$\pi=\iota$

(with $t_{pom^2n^2}$ and $t_{pm^2n^2o}$ defined in equations *(8)* and *(9)*). *(11)*

Considering the relative complexity of the example, and the non-triviality of the results, the fact that all computations remained manually tractable is quite interesting. This is, in our opinion, a very promising point regarding the applicability of linear logic to interface verification.

Of course, linear logic is not a silver bullet. It also suffers from the classical state space explosion problem inherent to any verification method. In the preceding example this appears with structurally conflicting transitions. Still, because linear logic does not add arbitrary causality relations, this problem does not show up as early as with other traditional methods, as our example shows. Moreover, the number of cases to consider can be further reduced, and the WCET further refined, by simply enriching the model with an explicit scheduling semantic.

4. Conclusion

In this article, we've focused on the automatic verification of the temporal compatibility of object interfaces in real-time systems. To address this problem we've adapted an execution time calculation technique recently proposed for plant production and multimedia scheduling. [Prad+00, Prad+99, GiPrV97]

In a first part, we've presented the fundamentals of this technique, based on Girard's linear logic and Petri nets. We've also introduced a new rewriting system that considerably alleviates the initial sequent calculus, in which this technique was originally developed.

In a second part, we've shown on an example how this duration computation technique could be adapted for the verification of object oriented real-time systems. Starting from a performance expectation of a client object, we could obtain a symbolic equation constraining the execution times of the different components of the system.

With this example, we could illustrate how efficient linear logic can be for the analysis of distributed and temporally constrained systems. This efficiency is primarily due to the very condensed state representation allowed by linear logic. The notion of *current step* is here essential, and allows considering several partial states separately. Those states progress independently until some explicit synchronization is required. In that way we avoid any arbitrary action interleaving, which actually meets up with the many results already reached on partial orders.

The symbolic nature of the obtained duration formulae, as opposed to plain numerical results, is also remarkable. Those expressions cover all possible action durations via variables, which would not be possible using a stochastic simulation or a Model-Checking technique. This particular aspect seems to be very promising in the context of system development based on Components Off the Shelf (COTS), where sizing is critical.

Finally, the rewriting rules we propose are quite easy to implement, as they directly manipulate symbol strings. By implementing them in an Integrated Development Environment (IDE), a designer would be given the ability to identify performance incompatibilities very early in the development of a Real Time OO System.

A first prototype of linear logic computation has been implemented at LAAS [Kabo00]. In the near future, we plan to use this first tool to study the applicability of our technique on large-scale OO distributed systems.

5. Acknowledgment

The authors would like to warmly thank Robert Valette for his very kind support, and his unbroken patience in answering our numerous questions.

6. References

[BriGue96] J.-P. Briot, R. Guerraoui, Objets pour la programmation parallèle et répartie : intérêts, évolutions et tendances, *Technique et Science Informatiques*, AFCET-Hermès, Paris (F), 15(6):765-800, June 96.

[*Corba00] Object Management Group: *The Common Object Request Broker: Architecture and Specification, Revision 2.4: Oct. 2000*, ftp://ftp.omg.org/pub/-docs/formal/00-10-33.pdf.gz, accessed 2000-11-02.

[DePa99] J. Delatour, M. Paludetto : De HOOD/PNO à UML/PNO : une méthode pour les systèmes temps réel basée UML et objets à réseau de Petri, *2nd Congrès sur la Modélisation des Systèmes Réactifs (MSR'99),* Cachan (F), Ed. Hermès, 1999, pp.223-231

[DePa98] J. Delatour, M. Paludetto: UML/PNO, A Way to Merge UML and Petri Net Objects for the Analysis of Real-Time Systems, In Proc. of the *Object-Oriented Technology and Real Time Systems Workshop (ECOOP'98),* Brussels (B), LNCS n° 1543, Springer Verlag, 1998, pp.511-514

[Gehlot92] V. Gehlot: *A Proof Theoretic Approach to Semantics and Concurrency*, Ph.D. thesis, University of Pennsylvania, 1992.

[Girard87] J.Y. Girard: Linear Logic, *Theoretical Computer Science*, n°50, 1987.

[Girard90] J.Y. Girard: La Logique linéaire, *Pour la Science*, n°150, Apr. 1990, pp. 74-85.

[Girard95] J.Y. Girard: Linear Logic, its syntax and semantics. In *Advances in Linear Logic*, Eds. Girard, Lafont, Regnier, London Mathematical Society, Lecture Notes Series 222, Cambridge University Press 1995.

[GiPrV97] F. Girault, B. Pradin-Chézalviel, R. Valette : A logic for Petri nets. *RAIRO-APII-JESA, Journal Européen des Systèmes Automatisés*, Vol. 31, N. 3, p.525-542,1997.

[Girault97] F. Girault : *Formalisation en logique linéaire du fonctionnement des réseaux de Petri*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, 1997.

[GusEr98] J. Gustafsson and A. Ermedahl : Automatic derivation of path and loop annotations in object-oriented real-time programs. *Journal of Parallel and Distributed Computing Practices*, vol. 1 no. 2, June 1998

[Hood95] Hood Technical Group: *Hood Reference Manual, Release 4.0*, (HRM4-9/22/95), September 1995

[HodJo96] S. J. Hodges, C. B. Jones: Non-Interference Properties of a Concurrent Object-Based Language: Proofs Based on an Operational Semantics, in *Object Orientation with Parallelism and Persistence*, Kluwer Academic Pub. 1996. Eds.: B. Freitag, C. B. Jones, C. Lengauer and H.-J. Schek.

[Jones96] C. B. Jones: Accommodating Interference in the Formal Design of Concurrent Object-Based Programs, in *Formal Methods in System Design*, vol. 8 number 2, pages 105-122, March 1996.

[Joy+00] B. J., G. Steele, J. Gosling, G. Bracha: *The Java Language Specification, Second Edition* (The Java Series) - 544 p. 2nd edition, June 2000, Addison-Wesley Pub Co, ISBN: 0201310082

[Kabo00] W. E. Kabore, *Réseaux de Petri et Logique Linéaire pour la vérification de systèmes concurrents: algorithmes de preuve*, Mémoire de Projet de Fin d'Etude, I.A.I. (Libreville, Gabon), LAAS-CNRS (Toulouse, France). Responsables : R. Valette, B. Pradin-Chézalviel. Toulouse (F), 2000.

[Kindl97] E. Kindler: A Compositional Partial Order Semantics for Petri Net Components. In *Proc. of the 18th International Conference on Application and Theory of Petri Nets, ICATPN 1997*, Toulouse (F), p. 235-252, LNCS 1248, Springer Verlag.

[Lam80] L. Lamport, Sometimes is sometimes "not never" - on the temporal logic of programs. In *Proc. of the 7th ACM Symposium on Principles of Programming Languages*, pp. 175-185, Jan. 1980.

[Lee00] E. A. Lee, What's Ahead for Embedded Software, *Computer*, Sep. 2000, vol. 33, n°9, IEEE Computer Society, pp. 18-26

[Merz00] S. Merz : Model Checking. In *Proc. of the Summer School on MOdelling and VErification of Parallel processes*. (MOVEP'2k), Nantes (F), 19-23 June 2000. Eds.: F. Cassez, C. Jard, B. Rozoy, and M. Ryan. pp.52-70

[Meyer93] B. Meyer: Systematic Concurrent Object-Oriented Programming, in *Concurrent Object-Oriented Programming*, Special Issue of the Communications of the ACM (CACM), vol. 36, n°9, Sep. 1993.

[PaDe99] M. Paludetto, J. Delatour : UML et les réseaux de Petri : vers une sémantique des modèles dynamiques et une méthodologie de développement des systèmes temps réel, Rapport. LAAS n°99511, *Revue L'Objet*, vol. 5, n°3-4, pp.443-467, Dec. 1999

[Palu91] M. Paludetto : *Sur la commande de procédés industriels : une méthodologie basée objets et réseaux de Petri*, Thèse de Doctorat, Université Paul Sabatier, Toulouse, Dec. 1991, n°1071, 270p.

[Polz+99] A. Polze, J. Richling, J. Schwarz, and M. Malek : Towards Predictable CORBA-based Web-Services, in *Proc. of the 2nd IEEE Int. Symposium on OO Real-time distributed Computing (ISORC'99)*, pp. 182-191, St. Malo (F), May 2-5, 1999

[Prad+99] B. Pradin-Chézalviel, R. Valette, L.A. Künzle : Scenario Durations Characterization of t-timed Petri Nets using Linear Logic. *Proc. of the 8th IEEE Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, Zaragoza, Spain, Sep. 6-10, 1999.

[Prad+00] B. Pradin-Chézalviel, R. Valette: Accessibilité de marquage et logique linéaire dans un réseau de Petri t-temporel, in *Journées Formalisation des Activités Concurrentes, FAC'2000*, CERT-IRIT-LAAS, Toulouse (F), May 18-19, 2000, p.123-134

[Selic98] B. Selic : Using UML for Modeling Complex Real-Time Systems, In *Proc. of Languages, Compilers, and Tools for Embedded Systems ACM SIGPLAN Workshop LCTES'98*, Montreal, Canada, June 1998, LNCS n°1474, Springer Verlag.

[Selic94] B. Selic, G. Gullekson, P. Ward: *Real-Time OO Modeling*, John Wiley & Sons, New York, 1994

[SchKu00] D. C. Schmidt, F. Kuhns: An Overview of the Real-Time CORBA Specification, *Computer,* June 2000, vol. 33, n° 6, IEEE Computer Society, pp. 56-63.

[*UML99] UML Revision Task Force, *OMG UML v. 1.3*, Object Management Group, http://www.omg.org/-cgi-bin/doc?ad/99-06-08.pdf, June 1999.