Validating Hypermedia Documents: a Timed Automata Approach

Marcelo Fagundes Felix

e-mail: felix@inf.puc-rio.br

Edward Hermann Haeusler

e-mail: hermann@inf.puc-rio.br

Luiz Fernando Gomes Soares

e-mail: lfgs@inf.puc-rio.br

PUC-RioInf.MCC21/02 Agosto, 2002

Abstract: This paper presents a formal approach to validating hypermedia documents, a case of real-time and reactive computing application, aiming to extend these ideas to the context of architectural design. The main aspect of hypermedia validation concerns to the so-called temporal consistency analysis problem: a document is consistent if it is possible present it in a way that all event synchronization constraints can be fulfilled. The validation technique consists on providing formal semantics to hypermedia documents via network of timed-automata, in a compositional way. Therefore, obtained a behavioral model of the document, we can pass onto model-checking phase. The consistency analysis consists on to formulate an adequate set of temporal logic formulas that will be checked against to the automata model. The specifications of the hypermedia documents that we are considering are written using the modeling concepts of NCM (Nested Context Model [2][7]), a conceptual model to hypermedia documents, and we choose UPPAAL [5][6] as the model-checking tool to implement the our hypermedia validation model. This work constitutes too a preliminary investigation, via a case study, about how we can integrate formal validation, using model-checking techniques, with software architecture design.

Keywords: hypermedia documents, formal validation, model checking, software architecture.

Resumo: Este trabalho apresenta uma abordagem formal para a validação de documentos hipermídia, visto como um caso particular de sistema reativo de tempo-real e objetiva aplicar essas idéias ao contexto do projeto arquitetural. O principal aspecto da validação hipermídia diz respeito ao assim chamado "problema da consistência temporal": um documento é dito temporalmente consistente se é possível apresentá-lo de tal forma que todas as condições de sincronia dos eventos envolvidos possam ser atendidas. A técnica consiste em prover uma semântica formal composicional para documentos hipermídia, baseada em autômatos temporizados, que nos permita obter um modelo comportamental do documento. De posse desse modelo derivado é possível validar o documento usando um model-checker. A análise de consistência consiste em formular um conjunto adequado de fórmulas em lógica temporal, que expressem a consistência do documento, para então testar se elas são ou não verdadeiras naquele modelo de autômatos obtido. As especificações de documentos hipermídia que estamos considerando são escritas usando os conceitos de modelagem estabelecidos em NCM (Nested Context Model), um modelo conceitual para documentos hipermídia. Para implementar nosso modelo de validação, escolhemos o Model-Checker UPPAAL. Este trabalho pode ser visto como uma investigação preliminar para um estudo de caso, sobre como integrar validação formal, mais especificamente model-cheking, com projeto arquitetural.

Palavras-chave: hipermídia, validação formal, model-checking, arquitetura de software.

1.Introduction

In the last years most of the work on software development has focused on the notion of components and the very way they are to be integrated in a whole system. This approach gave raise to some frameworks for dealing with domain specific issues on one side, and, to some general purpose languages for describing systems based on components, on the other side. The later works are best represented by the so-called Architecture Description Languages (ADL) like DARWIN, WRIGHT, ACME and others. One can argue that the profit of having a well-structured system might contribute for a better validation of the system itself. In general, the main advantage is to allow the task of validation to be compositional, i.e., the validation of the system would be a function of the validation of its components, where this function reflects the semantics of the architectural operators involved in the system's description. The authors of this article believe in architecture of software as a way of providing a better framework for validation of software. There are other works [1] that show us how integrate formal validation and architectural design and, likewise, the focus is on formal validation, that is, proving the satisfaction of properties against the model induced by the system, either by means of proving theorems about the theory represented by the system or by means of model checking these properties in the model itself. This article intends to show the advantage of having some architectural principles in designing systems when looking towards formal validation. It presents a detailed study on an architectural hypermedia system's design approach showing how to obtain a formal validation layer for the kind of architectural principles involved. The emphasis is on model checking of properties. It is worthwhile noticing that there are other works on formal validation of Hypermedia systems [3], however, they do not address architectural principles from its background. Some of these works do not contain enough explicit detail and formal description in order to the authors of the present work be able to withdraw any conclusion about an implicit use of architectural operators besides those imposed by the corresponding design model.

From the architectural point of view, systems based on hypermedia documents[1] constitute an interesting case study due to complex nature of data involved in. Besides, this kind of system can

[1] From now on, we will use the term "document" on the place of "hypermedia document", just for simplicity.

be seen as a particular case of reactive real-time application. It is a well-established fact that the testing activity inside the life cycle of these systems is a very hard one. The main goal of our present work is the formal validation of the real-time component of hypermedia systems. As we know, real-time systems are reactive systems in which the time deadlines and timely execution are essential requirements to be fulfilled. The current experience has showed us that, in general, old-fashioned development methods are not suitable to handle the main aspects of such systems. Indeed the non-determinism imposed by the temporal properties introduces difficulties to the traditional testing phase, especially that based on systematic test techniques. In order to overcome such troubles, the timed-automata modeling and the symbolic model-checking approach has been successfully used to validate real-time systems as can be seen in [6][8]. It is well-established fact that to invest in formal validation phase reduces the whole cycle of corrective test, bringing the checking of faults to the initial phase of design. This strategy minimizes the over-all cost of development. In this work, we choose to apply this same method to the temporal validation of hypermedia systems, taking into account that the reactive component is imposed by the highly interactive nature of these systems.

The approach that we will present consists on three distinct phases: first, the hypermedia author (designer) needs to model the document (system) using an authoring tool (design tool). This tool automatically generates code in a modeling language conceived to express the model semantic concepts (hypermedia documents). Now, we can translate this description in a timed automata model semantically equivalent. Finally, the last phase, consists on model-checking the automata using an adequate model-checker. For this, the designer needs to formulate a set of properties that the modeled system must satisfy. The first step was already implemented on HyperProp [11], an authoring and formatter tool developed at PUC-Rio. So we concentrate ourselves on the latter steps. In [3][4][12][13], an algebraic specification based approach, using RT-LOTOS (a temporal extension of the LOTOS process algebra formalism), was applied as a validation model to the hypermedia temporal consistency problem. For the sake of validation, this method consists on to translate from the hypermedia document specification to the RT-LOTOS formalism and then to translate from this algebraic specification to the timed automata formalism. Finally, over the timed automata model so obtained, it is possible to proceed to the validation step using some model-checking tool. Differently, by our approach, we derive a timed automata model directly from the

document specification. One of the objectives of our work is to investigate how we can provide a practical formal validation layer integrated to the hypermedia authoring tools (that can be seen as a particular case of architectural design tool). It is interesting to stress that, from a wider point of view, these ideas can be naturally extended to provide a formal validation layer to real-time software architecture design, in particular, as well as, to any software architecture design, in general.

The article is structured on the following way. In section 2, we present the concepts of a model for hypermedia data, namely the Nested Context Model (NCM) and we show an example of document modeled according to the NCM concepts. We choose it among the several models because it is a sufficiently complex and it is compositional, that is, it makes use of modular abstraction as one of the design principles. In the section 3 we introduce an extension of timed automata to cope with broadcast signals and we use this model to provide semantics for hypermedia models. In section 4 we show how to use the timed automata semantics introduced in section 3 in order to validate documents using the UPPAAL model checker [5]. Some conclusions and comments about future works are given on the section 5.

2.The Compositional Hypermedia Model

2.1. A informal presentation

A compositional hypermedia model is a conceptual model for hypermedia documents that allows functional abstraction in a compositional way. So, a composition of documents is a document too. As such one, the model must provide a representation for the document based on simple and structured data objects, as well as, some way of specify the presentation of the whole document and its parts. Roughly speaking, the model has static and dynamic parts. In the static component, the model is based on the node concept that is used to encapsulate and to structure information. In the dynamic part of the model, the events and state machines are the central concepts. Now, to

introduce the terminology, we describe bellow some of the main basic concepts of the NCM model[2].

> ***Content node***: entity that has a media content (video, audio, text or image), a set of anchors and a set of descriptors.
> ***Composition node***: entity used for grouping in a node a set of nodes and their relationships.
> ***Anchor***: defines a fragment of information. Information can be a set of *media units* (inside content nodes) or a set of relationships (a structure inside a composition node). To completely define a node presentation we need to say what will be presented. The anchors define it.
> ***Descriptor***: keeps all the information about the node presentation (output device, edit method, start and end methods, window position, time duration, etc).
> ***Event***: any kind of occurrence in time, not necessarily instantaneous. An event is necessarily associated to a node. There are at least three types of them: selection event, presentation event and assign event. The selection event is associated with the user selection of a node anchor. The presentation event is, as the name says, a kind of event related to the exhibition of an anchor of a node. The last type mentioned is a kind of event associated to the change of the value of some attribute previously established on a node. Events constitute the main concept in the dynamic part of the model.
> ***Link***: entity that establishes (causal and temporal) relationships among events. It defines what conditions about some events must be accomplished to trigger some actions over other events.

The structured definition of documents is desirable as it carries built-in concepts of modularity, encapsulation and abstraction. The NCM is an object-oriented model that allows logically structure the inner components in a document by the use of nested compositions. On the NCM, the relationships between the components can be modeled using compositions and links. It is possible to define an acyclic graph that indicates the control flow of the document presentation, using compositions for encapsulating and links to tie components together. Both of them constitute a way to define temporal relationships among the components. A composition defines a certain pre-established pattern of structural relationship that can be viewed as a temporal one, as for example, the sequential or parallel presentation of the internal components [10]. Any other kind of complex relationship among the events (user interaction, synchronism of presentations or changing node attributes) can be modeled using links.

---

[2] The NCM intends to be a complete model to systems based on hypermedia documents. For our purposes we just expose the concepts that are fundamental to comprehension of the article.

So, a hypermedia document modeled in NCM can be seen as a graph of compositions[3]. Each composition holds its nodes (media contents or another composition) related each other by the links (representing how these pieces of information are related on the space and time). Each link in a NCM document specifies a temporal or spatial relationship among the nodes (more precisely, events defined by the anchors of nodes). For this, the links indicate a set of source terminal points, a set of target terminal points and a meeting point composed by a condition part and an action part. It is this pair condition-action that describes how the events associated to the presentation of nodes are related one another. The condition part is a logical sentence about the set of source events and the action part is a sentence indicating how to change the state of each target event. It is not our interest to make a profound analysis of these NCM syntactic constructs. Indeed, for effect of temporal validation we are concerned with the expressive power of these mechanisms aiming the dynamic aspects of the model.

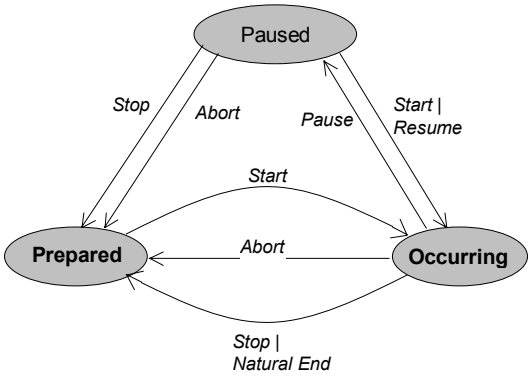


Figure 1 – Events state machine

Action	Transition
Start(E)	If *State(E)*∈ {*prepared, paused*} Then *State(E)*← *occurring;* *Start the presentation of 1st media unit* Else, *no transition occurs*.
Stop(E)	If *State(E)*∈ {*occurring, paused*} Then *State(E)*← *prepared;* *E.Occurrences++;* *E.Repetitions-- { iteration control }* Else, *no transition occurs*.
Pause(E)	If *State(E)=occurring* Then *State(E)*← *paused* Else, *no transition occurs*.
Resume(E)	If *State(E)=paused* Then *State(E)*← *occurring* Else, *no transition occurs*.
Abort(E)	If *State(E)*∈ {*occurring, paused*} Then *State(E)*← *prepared;* *E.Repetitions* ← *0;* Else, *no transition occurs*.

Figure 2 – State changing table

The dynamic part of the model encompasses the events related to the presentation of the nodes defined on the static part of the model. Among the layers of the model, the representation layer is that where the events are effectively represented. The objects in this stratum result from to bide

[3]The term "context" is used in NCM for reference to compositions. For our purposes we does not distinguish one of other and we use composition as a kind of modular abstraction.

the static objects[4] from the model to the respective dynamic version of it. The document specification provides to the authoring tool all the information about how to instantiate the objects on the representation layer. For each event on this layer there is an instance of the generic state machine (Figure 1). With these state machines to each event, the authoring tool can to run the presentation of a document specified, controlling the events flow according to condition/action pairs expressed by the links. The actions and associated state changing for the events can be seen on the Figure 2.

2.2. The Representation Objects Language

In order to give a semantic to documents presentation, we need to define a formal language to express the main conceptual objects of the NCM model. The Figure 3 presents the abstract syntax of this language that we call *Representation Objects Language*[5], ROL for short. In this language, *composition node* (represented by the metavariable X) is the main syntactic concept.

Metavariables

X: Composition	Δ: Descriptor	Ψ: Signal	Ξ: Anchors
Γ: Nodes	K: Condition	I: Identifier	N: Number
Λ: Links	A: Action	Φ: Attribute	T: Type
E: Events	Σ: State	Ω: Operator	M: Media

Abstract Syntax

X ::= **comp** I Γ E Λ
Γ ::= **term** I E | **comp** I Γ E Λ | Γ_1 Γ_2
E ::= **ev** I Δ | **pub ev** I Δ | E_1 E_2
Ξ ::= λint | I
T ::= **prs** | **sel** | **att**
Δ ::= **desc** Ξ T M int int | **desc null**
M ::= **audio** | **video** | **text** | **image**
Λ ::= **link** I K ⇒ A | Λ_1 Λ_2
K ::= I@Σ | I.Ψ | Φ(I) Ω int | K_1 **and** K_2 | K_1 **or** K_2 | ¬ K | K ⊕ int | **true**
Σ ::= **PREPARED**| **OCCURRING**| **PAUSED**
Ψ ::= **start** | **stop** | **resume** | **pause** | **abort**
Φ ::= **Occurrence** | **Repetition** | I
Ω ::= **=|<>|<|<=|>|>=**
A ::= **start**(I) | **stop**(I) | **pause**(I) | **resume**(I) | **abort**(I) |
A_1**||**A_2 | A_1**;**A_2 | A_1**,**A_2 | **delay**(int)

Figure 3 – ROL abstract syntax

[4] The static part of the model compose two levels, namely, storage and data objects layers.
[5] One node can have many NCM descriptors, but after the binding that creates the representation objects (dynamic version of it), each event has just one ROL descriptor.

Note that we need to represent the nested structure on the language for name visibility reasons directly related to compositionality of the model. Inside the main composition we have other nodes. A node Γ can be a content or another composition node, identified by the **term** or **comp** keywords, respectively. On this representation level a content node has a name and its associated events list. Each event has an identifier name, an anchor and the type (selection, presentation or attribution[6]). Linguistically, a link object is identified by its name and defines a *meeting point* that is used to connect the *source events* (events over which a trigger condition is established) to *target events* (events to which an action is fired). Note that an event is semantically viewed as a state machine so that a meeting point trigger condition is basically established by two cases: the permanency or not in a certain state, and the state changing.

Observing the abstract syntax, conditions can be complex or atomic. The abstract syntax to conditions defines a complex condition as $\mathbf{K ::= K_1}$ **and** $\mathbf{K_2 \mid K_1}$ **or** $\mathbf{K_2 \mid \neg K \mid K \oplus N}$, meaning the usual logical conjunction, disjunction and negation and the delayed condition, a kind of parametric condition that is true at the present instant if **K** was true **N** time units ago. The next three syntactic patterns, $\mathbf{I.\Psi \mid I@\Sigma \mid \Phi(I)\ \Omega\ N}$, are the atomic cases for conditions and are respectively evaluated true on the following cases: when the **Ψ** transition on the **I** event happens (on the associated controller machine), when the **I** event stays on the state **Σ**, and when a certain attribute **Φ** of the **I** event (e.g.: number of occurrences) holds the relation **Ω** with **N**. Likewise, meeting points can have complex action expressions according to the syntax $\mathbf{A ::= A_1||A_2 \mid A_1;A_2 \mid A_1,A_2 \mid}$ **delay**(**N**). The $\mathbf{A_1||A_2}$ syntactic construction denotes the parallel execution of the two actions started together, that is, $\mathbf{begin(A_1)=begin(A_2)}$. The complete sequencing, expressed by $\mathbf{A_1;A_2}$, where $\mathbf{end(A_1)<=begin(A_2)}$, indicates a compound action where the second action needs to wait the termination of the first action. The $\mathbf{A_1,A_2}$ expression means a kind of weak sequencing, where $\mathbf{begin(A_1)<=begin(A_2)}$. The expression **delay**(**N**) is a delay action and means the waiting for **N** time units. The atomic actions are used to produce different kinds of state changing over the event state machine to which it is applied. These are the main aspects of the representation objects language. Follow, we show an example that uses these core features.

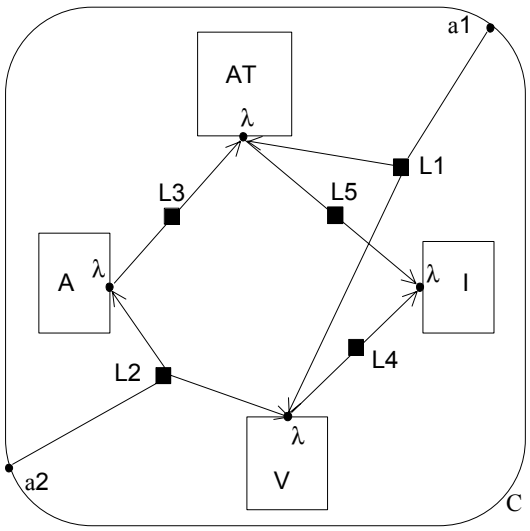
---

[6] For simplicity questions, in this work we do not take in effect the NCM attribution events.

2.3. A document specification

A diagrammatic schema helps us to visualize the document structure (Figure 4). The document is a *composition node* **C** made up of four components, all of them, *content nodes*: an audio **A**, a video **V**, an image **I** and a node identified **AT** that can be presented as an audio or text (on the descriptor there is information about the way that the content of a node will be effectively presented).

The composition can be started from two possible points: the *anchors* **a1** and **a2**, into the scope of **C**. The choice of one of them will put the presentation at two different scenarios. If **C** starts from **a1**, the nodes **AT** and **V** will be started at the same time. The nodes **AT**, as an audio, and **V** will last 10sec and 8 to 9 seconds, respectively. When the composition **C** is started from **a2**, then the nodes **A** and **V** are started together, with duration 7 seconds and 8-9 seconds, respectively. The presentation event of the content node **I**, identified for **E8**, is started by the **L4** link. As we can see on the figure 4, its meeting point says that the presentation must be started just on the end of the **E3** or **E5** events, respectively, the **V** presentation, by **L1,** and the **V** presentation, by **L2**. As the duration of this presentation is not defined on its descriptor, it is necessary that some link control the end of the **E8** event. This is accomplished by the **L5** link that has in its meeting point the control information: when **E2** or **E7** ends, the **E8** event must be stopped. So, the image **I** will be stopped just at the end of the **AT** presentation started from **L1** or **L3**.




```
comp C
        term AT ev E2 desc λ prs audio 10 10
                ev E7 desc λ prs text 6 6
        term V  ev E3 desc λ prs video 8 9
                ev E5 desc λ prs video 8 9
        term A ev E6 desc λ prs audio 7 7
        term I ev E8 desc λ prs image ∞ ∞

        ev E1 desc a1 sel null
        ev E4 desc a2 sel null

        link L1 E1.start ⇒ start(E2) || start(E3)
        link L2 E4.start ⇒ start(E5) || start(E6)
        link L3 E6.start ⇒ delay(3);start(E7)
        link L4 E3.stop or E5.stop ⇒ start(E8)
        link L5 E2.stop or E7.stop ⇒ stop(E8)
```

Figure 4 – A ROL document description

3. A Timed-Automata semantics to ROL

In order to apply our validation approach, we provide a timed-automata semantics to ROL, the language that we are using to describe a subset of NCM models. We will define a function that structurally maps each ROL syntactic concept to a corresponding broadcaster timed-automata concept. This function ([[ ]]), following the denotational semantics tradition, is compositionally defined. The B-timed-automata semantics obtained as image, by this function, can be used to validate properties about the original NCM model described in ROL.

But, how we can do that? First of all, it is important to do a question: what is the simplest concept from which we can define a mapping from NCM to a network of B-timed-automata? To answer this question it is enough to remember that NCM specifications describe how a set of multimedia events must be controlled. We consider that events constitute this simplest semantic concept. So, by means of a translation schema, each NCM event specified (selection or presentation) will give place to one B-timed-automata. To adequately decorate each one of these machines with the synchrony data among them, we need to extract, from the NCM model, all causal and temporal relationships existing between the events associated to the nodes. In fact, we add to the timed-automata model, some automata, that we call *synchrony machines*. These machines play the synchrony control among the linked events. Finally, the parallel composition of all machines, so obtained, will naturally capture the concurrent presentation semantics for the whole document specified. We will use the document modeled at the previous section to exemplify how we can translate (automatically) a document model into a network of B-timed-automata that can be used as input to the UPPAAL model-checker tool.

3.1. The B-timed-automata (Broadcaster Timed-Automata)

In this article we are using a different version of timed-automata, as it is defined at the literature. Our version extends the classic timed-automata model by allowing a signal (**a!**) to be acknowledged by more than one process enabled to the receiving of this signal (**a?**). In fact, all process waiting for an **a** signal (indicated **a?**) will acknowledge the **a!** signal in a synchronous way. In the UPPAAL timed-automata, process communication is like CCS, where only two

processes can synchronize at time. There is a UPPAAL mechanism that allows to simulate B-timed-automata: a sequence of two signals connected by a *committed* dummy state.

3.2. The Compositionality Concept

All NCM document is a composition node[7]. As we have said, composition nodes constitute a natural structuring mechanism to model documents in a modular way. In a document, each component can be a terminal node, with some kind of media content, or a composition node that group inside it a number of other related components. So, for we give a compositional semantics based on timed-automata, first we need to give timed-automata semantics for each simple event indicated on the links, and then, find a way to compose them, following the structure defined by the compositions. For accomplish this, the *composition node* concept needs to be naturally compositional. To our purposes we will only work with compositions that owns a well-defined interface[8].

3.3. A Semantics to Events

Events constitute the central semantic concept of any event-driven hypermedia model. As we know, each kind of NCM event has a typical state machine to model them. So, for we give a timed automata semantic to NCM models we need to translate events into state machines, so that state machines can to communicate each other according to the relationships among the events in the model. Indeed, each event in a model will give place to simple B-timed-automata on the semantic model. The NCM provides three kinds of events, namely: presentation, selection and attribution. On this work, we will only consider presentation and selection events, once that attribution events constitute just an auxiliary kind of control mechanism.

---

[7] A particular case of this is that of a simplest document as a text for example. In this case, the document is not a composition node but, in fact, it is a terminal node.

[8] Originally, NCM do not provide a mechanism to ensure compositionality. This was supplied by ROL with *public events* and *private events* mechanisms.

3.3.1. Types of Anchors

How we know, anchors indicate what part of content is to be presented. However, there are two kinds of anchors: λ anchors and the user defined anchors denoted by an identifier. The first indicates the whole content of a node and, the second, names a piece (segment) of a content, as defined by user.

3.3.2. Presentation events

Let **ev E desc Ξ prs M d D** be a presentation event of the anchor **Ξ** of a node **X**. This event consists on the presentation of the *media segment* (sequential pieces of information) referenced by the anchor **Ξ**. The presentation may last a minimum of **d** seconds and a maximum of **D** seconds. We can define presentation events for two kinds of nodes: composition nodes and content nodes. The first one is a kind of node that has no media content. In fact, it keeps information about the structure of document. So, its content is a set of nodes and links in such way that presentation consists on the exhibition of graph structure where the clickable anchors are available to user selection. Using this interface, the user can select what node will be presented in the sequel. The second possible case is the presentation of a content node (a node that has a real media content to be presented). Here, we also can use two kinds of anchors to indicate what media segment will be played: the λ anchor to present the whole content and an user identified anchor to indicate what part of content will be presented. The figure 1 shows the generic state machine for this kind of event.

3.3.3. Selection events

Let **ev E desc Ξ sel M d D** be a selection event. The selection of a certain region named by the anchor **Ξ** originates an implicit presentation event of the same region. The user needs to see what he/she is going to select. This presentation shows the region **Ξ** inside the content of the node where the event was defined. The minimum and maximum duration is respectively the minimum and maximum quantity of time used to present the region defined by the anchor. In fact, all times we have a selection event, we have a presentation event associated. For example, the region **Ξ** can be an icon that when pressed (selected) will originate a control signal to be used on another place.

So, the icon needs to be presented to the user for allow the choice. If this presentation had duration then the icon would be selected only at this time interval. If no duration has been set, then the choice can be done at any time. The figure 4 shows us two selection events, respectively over the anchors **a1** and **a2**. There are only two states at the generic selection machine: prepared and occurring.

3.4. The Resulting Timed Automata

The meaning of a NCM document specification is the meaning of the most external composition node. Basically, we provide semantics to a composition assigning timed-automata to each one of its components. This semantics is structured on components themselves. The machine generated for events represents the life cycle of presentations, just as the machine for the links models the synchronism control. As we saw, the link, seen as connector, is an entity used to relate nodes, seen as components, and to define the so-called source and target events, seen as ports. So, the machines generated to each event must be decorated using the causal and temporal information taken from the links[9]. In order to get a better understanding, let us exemplify how to apply the network of timed-automata generating procedure, for the ROL document specification given on the section 2.3. That specification shows us a composition made up of five links connecting together five components: the composition C and four content nodes. The network of B-timed-automata resulting from the specification can be seen on the next sequence of figures.

The first schema (Figure 5) shows us four B-timed-automata communicating each other. There is a control machine for each event (E1, E2 and E3) and a synchronizer machine that implements the link L1. The event machines follow the pattern[10] given by the generic sate machine and the link machine establishes the synchrony among the events using the broadcaster mechanism.

[9] Each link possesses exactly one meeting point that keeps the information about causal and temporal relations between that events tied together.
[10] For readability reasons, we choice to simplify the machines formally defined in appendix.

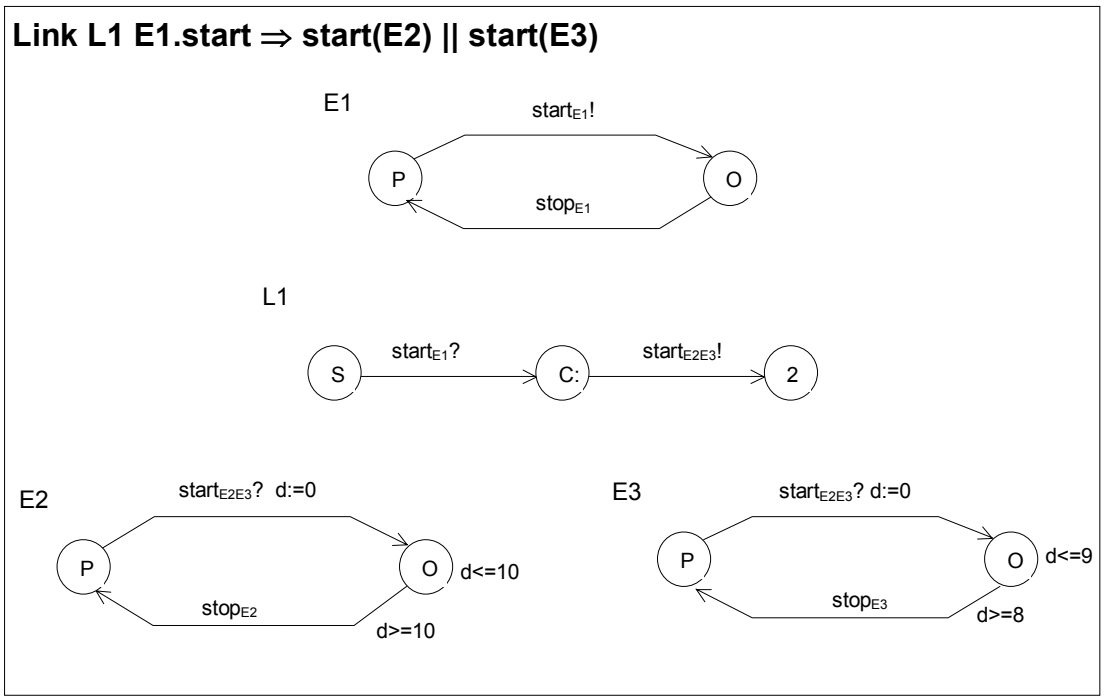


Figure 5 – The L1 synchronizer and E1, E2 and E3 machines.

The second schema (Figure 6) shows us a similar network of B-timed-automata. There is a control machine for each event (E4, E5 and E6) and a synchronizer machine that implements the link L2. Note that the duration of each event is informed on the descriptor of the event. The third schema (Figure 7) shows us a delayed action on the E7 event. For better understanding, this schema is a simplification over the timed-automata as defined by the semantic in the appendix. The last two schema have a **or** connective joining two atomic conditions. It is easy to see how this connective can be implemented as a synchronizer (L4 and L5) using different channels on two different transitions.

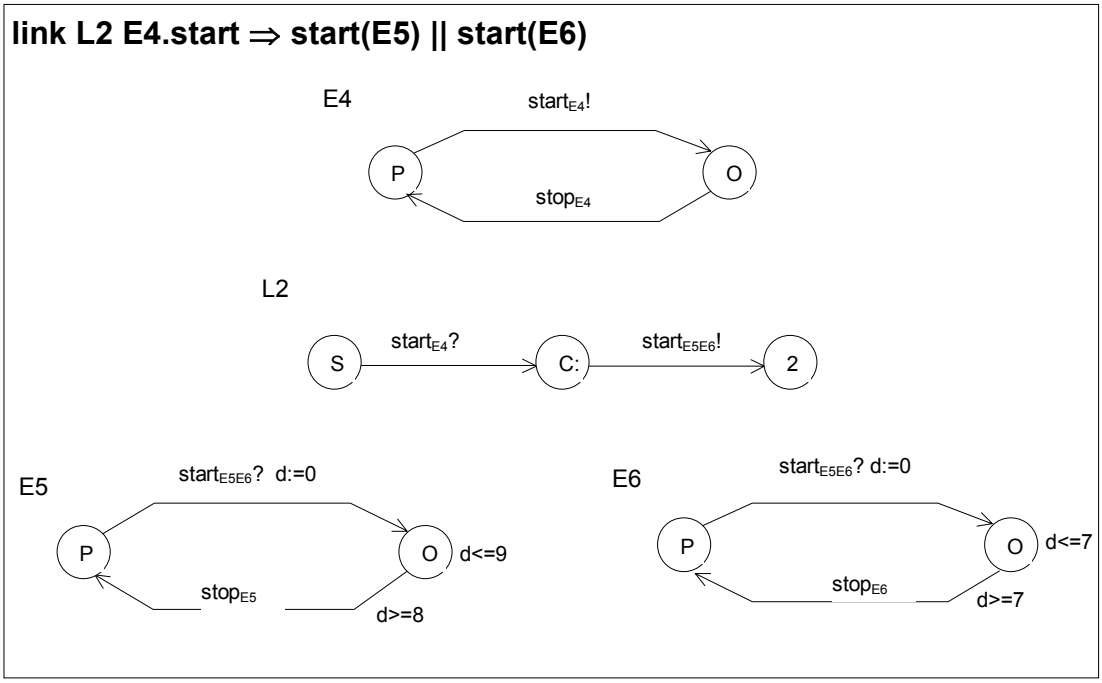


Figure 6 – The L2 synchronizer and E4, E5 and E6 machines.

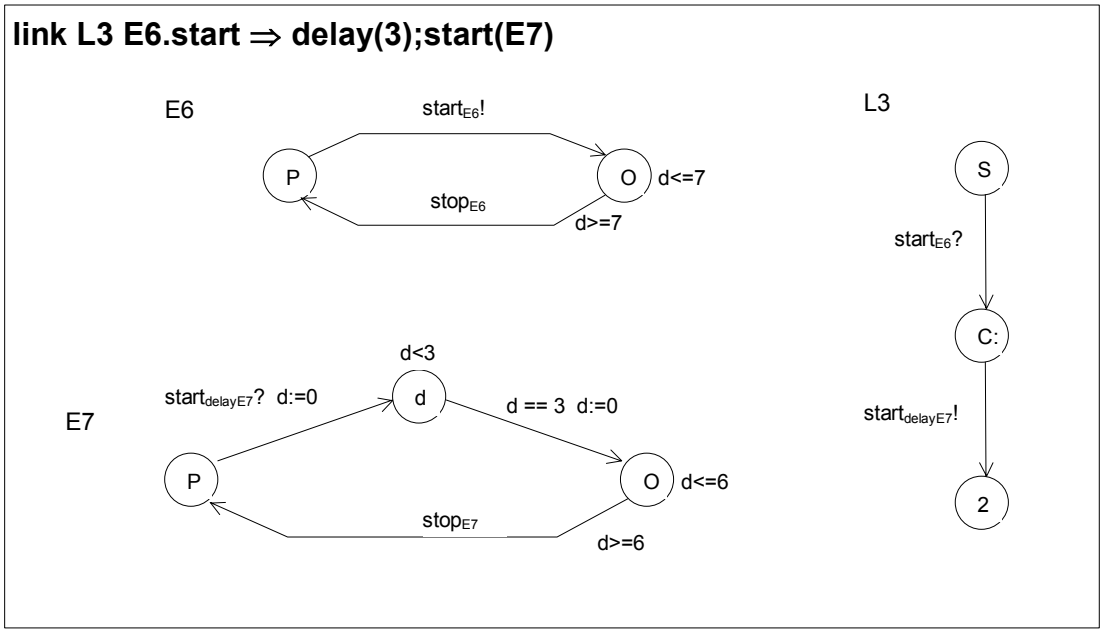


Figure 7 – The L3 synchronizer and E6 and E7 machines.

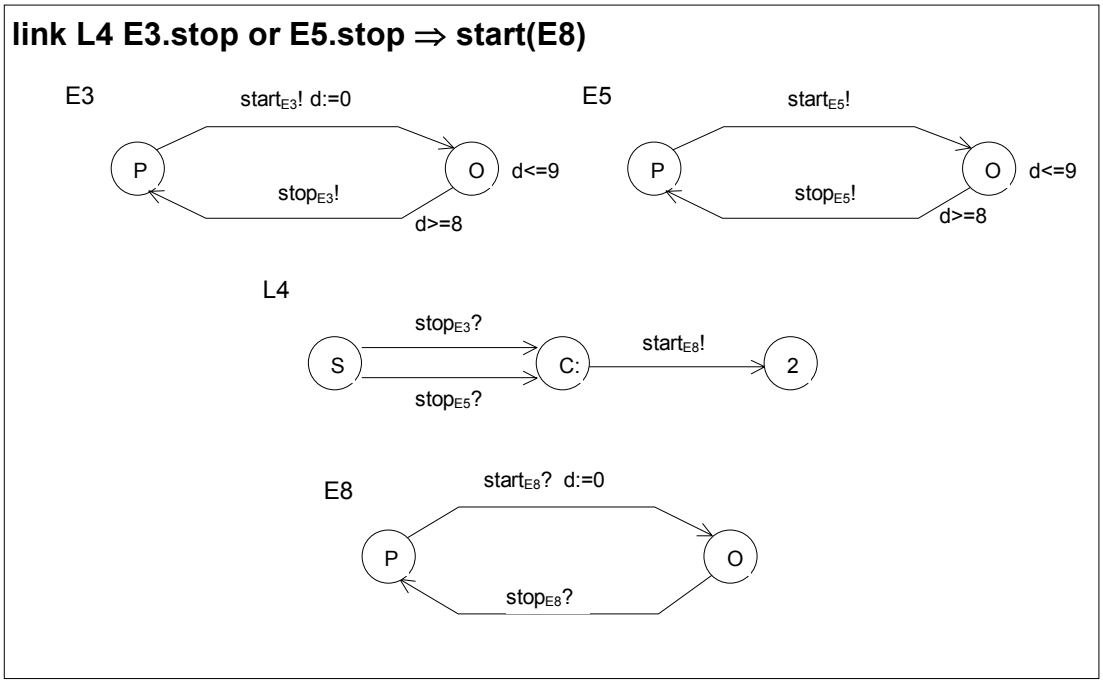


Figure 8 – The L4 synchronizer and E3, E5 and E8 machines.

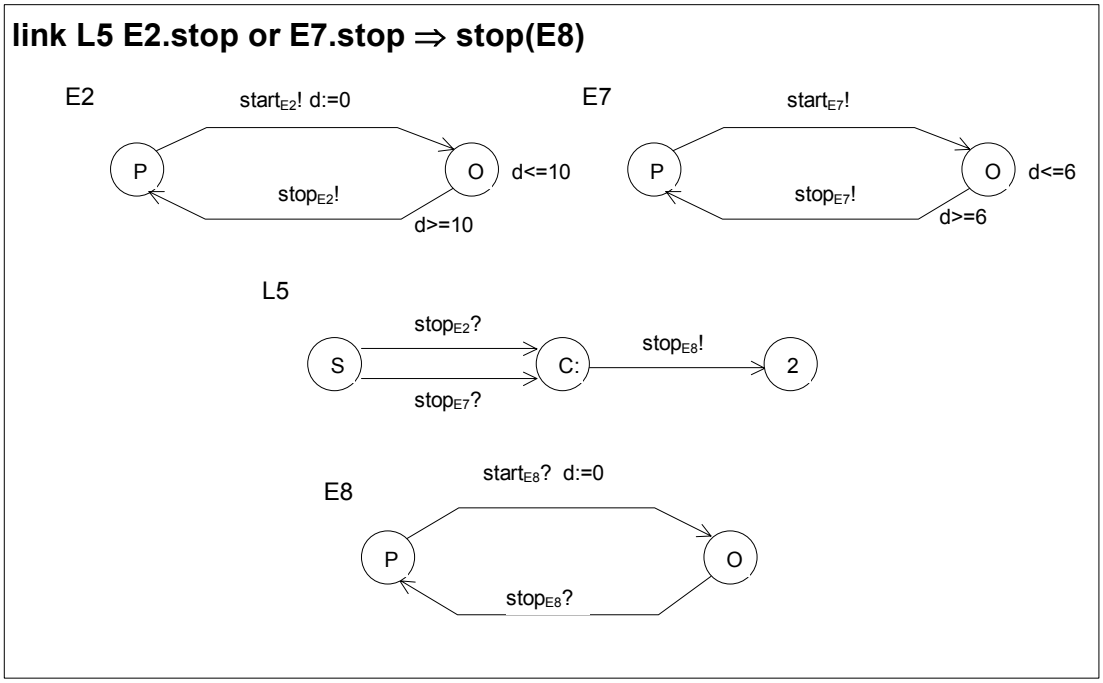


Figure 9 – The L5 synchronizer and E2, E7 and E8 machines.

4. The Validation Process

After the generation of the network of Timed Automata using the translation schema above, we can proceed to the validation process properly said: temporal consistency is adequately expressed in modal logic and the timed automata model is checked against the consistency property using some model-checker. If the property is true on the automata model, then the model is validated otherwise, a counter-example is given to show why the property is not true on the model. This process can be repeated until a model to satisfy the property.

4.1. The UPPAAL Model-Checker

UPPAAL[11] is a tool set suitable for modeling, simulation and validation of real-time systems [14][15]. From UPPAAL point of view, it is assumed that a typical real-time system is a parallel composition of non-deterministic timed sequential processes, communicating each other over channels and global variables. An UPPAAL model consists on a network of timed-automata where each timed-automata, modeling a system component, is a finite-state machine extended with real-valued clock and data variables.

Briefly speaking, UPPAAL have three parts: a basic programming language for timed automata, a simulator and a model-checker. With this language, it is possible, using a textual notation, to model the system as a network of timed-automata. It is important stress that, by the semantics of UPPAAL model, each automata in the network specified, runs concurrently with all the others. The second part, the simulator, has an important role on the process of modeling. With it, it is possible to run the model since the early stages providing an inexpensive way of error detection. It is important to reach the model-checking phase, properly saying, with a reasonable, trivial-error free model in hands. Stripping away the inessentials, we have a very simple set of features, available in UPPAAL, to express the behavior of real-time systems. Follows a summary of the most important concepts.

[11] Developed at Uppsala University (Sweden) and Aalborg University (Denmark).

Network of Timed Automata is the main structure of UPPAAL modeling. Any set of timed automata is naturally joined in a network by means of a synchronous product built-in operation;
Guards express conditions on the clocks and integer variables that need to be satisfied in order to the edge to be taken. The default guard of an edge is *true*;

Reset operations can be defined on the edges in order to control the set of variables defined on the machine. These operations consist on simple integer expression assignments;

Channels and Synchronization are the available communication mechanism between the machines of the network. Just like CCS, communication on channels occurs as two-process synchronization;

Committed locations and urgency are flow control mechanisms used to define the permanence time on the locals (states of the machines). If nothing is said, a machine can stay on the same state for an undefined time.

In UPPAAL, the transitions defined on the automata can be non-deterministic, that is, a state can be left on by more than one transition when there is more than one trigger condition enabled. This characteristic is essential to modeling of reactive systems. In these systems, there are circumstances where is necessary be capable to react to several stimulus came from the environment at the same instant. Some criteria to choice what transition must be taken use to be postponed to implementation. In true, at the most cases, an UPPAAL model is entirely deterministic. Even so, there is one case where the non deterministic choices could be useful: when we are describing a system not fully implemented, that is, a system wherein some design choices must still be left to the programmer or to another later phase on the developing.

In order to apply the UPPAAL model-checking to the hypermedia consistency analysis problem, we must to know how obtain an UPPAAL automata network from the NCM document specification. We can do this using, as much as possible, a translational schema. Such schema gives, to each part of a NCM specification, a semantic preserving UPPAAL implementation. One interesting fact about this approach is that this translation can be performed at any moment in the modeling process, since the begin until the last version of the document model. So, it is possible to practice an incremental validated-modeling to hypermedia documents.

4.2. Model-Checking with UPPAAL

The UPPAAL model for a document specified on NCM must include an environment component where the document is presented. The user must be modeled as a simple machine to simulate the interaction with the document components. After we add this component to the UPPAAL model early obtained, we can proceed with the verification phase properly said. When we submit the network timed automata to UPPAAL, the deadlock verifier automatically carries out the first step in the model validation. If the model is deadlock free, then it is verified by simulation. The Figure 10 shows us this phase being running on a UPPAAL model of our example.

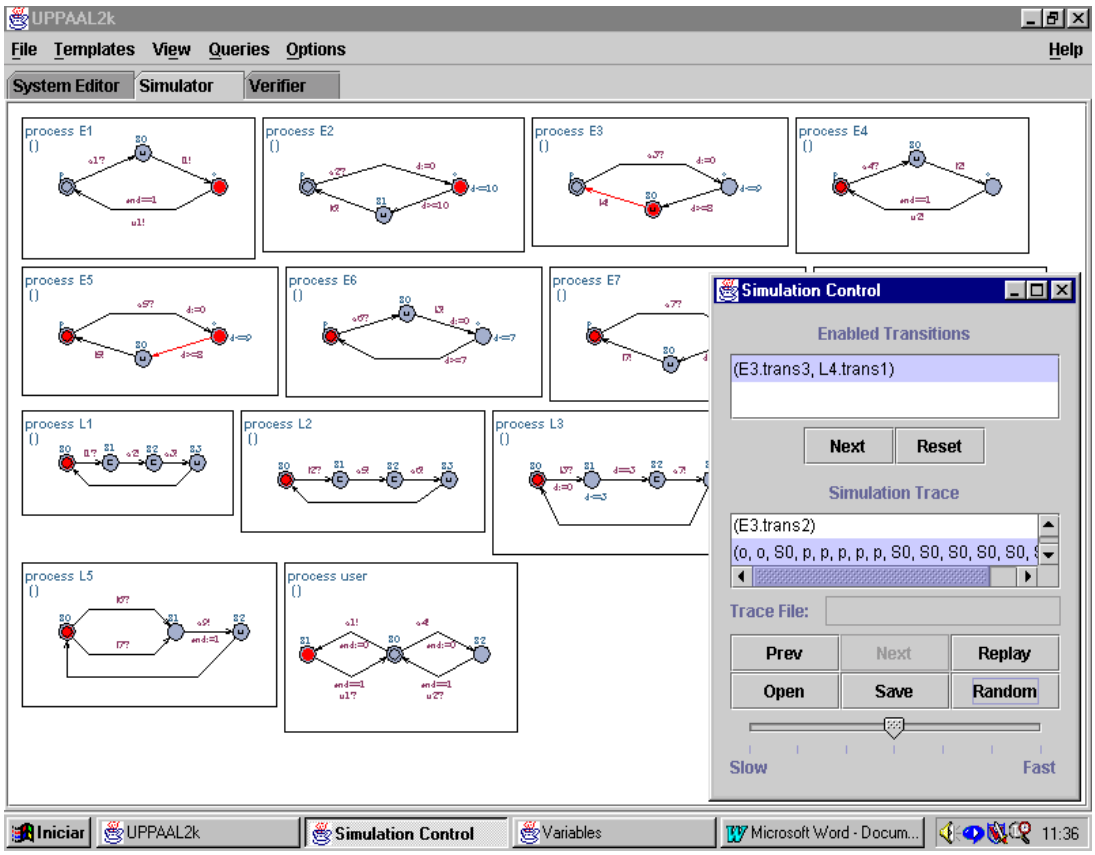


Figure 10 – The UPPAAL verifier simulating our model.

It is important to stress that this UPPAAL model constitute an implementation of that schema given in section 3.4. The timed-automata language available with the tool does not allow simultaneous synchronization by mean of the same channel. This feature must be simulated in UPPAAL timed-automata language. Some others aspects concerns to the flow control were added

to the UPPAAL model too (like urgency on channels and committed locations). This phase of the process give to the user a general view of the modeled system by an exhaustive exploration of its behavior. Ending this stage, the model is supposed to be free of the trivial modeling mistakes and we can go on to the model checking properly said. Some reachability properties must be formulated and checked against the model in order to guarantee that we have a temporal consistent document. Let us see a checking section for exemplify how we can apply the approach.

4.2. A validation session using UPPAAL

In this phase, we need to formalize the system requirements in the UPPAAL temporal-logic language. This language allows to express reachability properties and some cases of bounded liveness, in particular, if combinations of states and constraints (on clocks and integer variables) can be reached from the initial configuration of the system. The properties that can be analyzed by the model-checker has the following syntactic formation:

φ ::= $\forall\Box\ \beta \mid \exists\Diamond\ \beta \mid \forall\Diamond\ \beta \mid \exists\Box\ \beta$

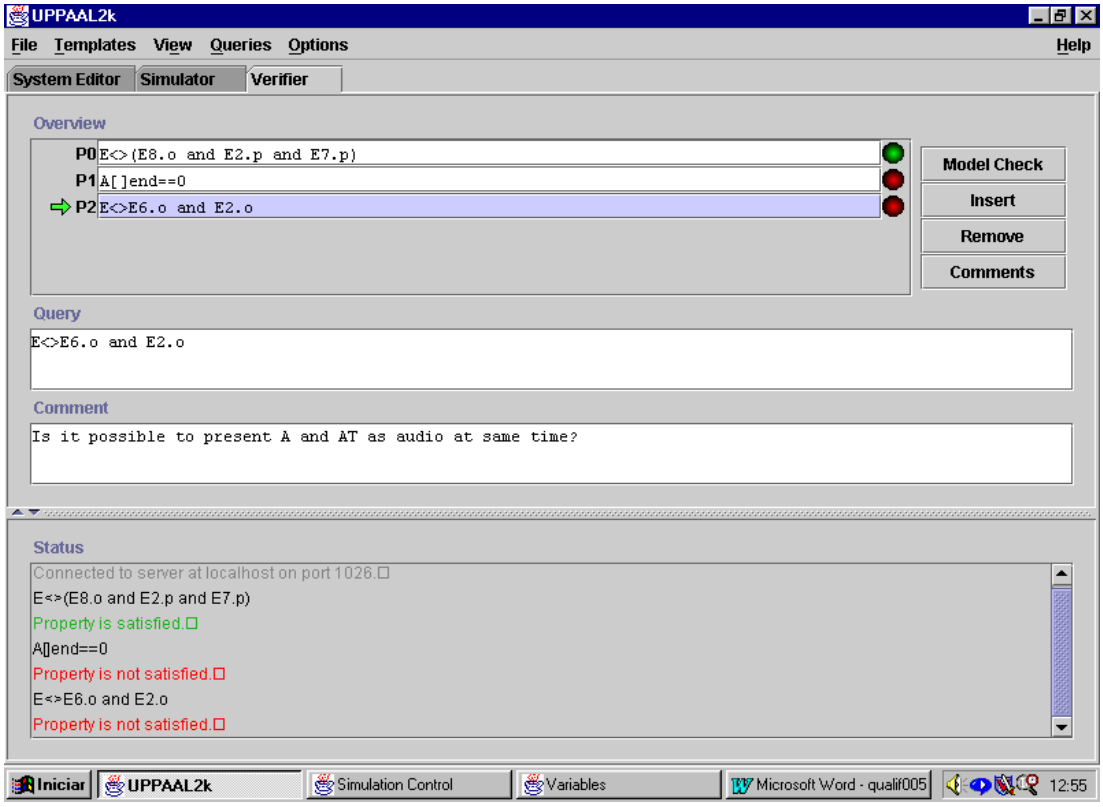
β ::= atom $\mid \beta \wedge \beta \mid \beta \vee \beta \mid \beta \rightarrow \beta \mid \neg\beta$

atom ::= IdMachine . State | ClockExpression | IntegerExpression

The modal operators $\Box$ and $\Diamond$ have the usual meaning. In this language, $\forall\Box$ means "for all paths it is necessary", $\exists\Diamond$ means "there is a path where it is possible", $\forall\Diamond$ means "for all paths it is possible" and $\exists\Box$ means "there is a path where it is necessary". There are three types of atomic formulas: over machine states, over the value of a clock variable or about the value of an integer variable. We are interested to check for temporal consistency of documents. For this, we need to formulate a sentence (or a set of sentences) that captures the semantic notion of consistency. Using the automata model derived from our example, we run the UPPAAL verifier using the following sentences:

```
∃◇ (E8.o and E2.p and E7.p)
∀□ end==0
∃◇ (E6.o and E2.o)
```

The first modal formula means: “There is a system configuration where the image I is being presented while AT, as audio or text, is not being presented?”. The second formula denotes that for all paths it is necessary the variable end to be set on 0, that is, there is not a possibility of presentation finish. The results can be seen on the Figure 11. Note that, for this last question the model-checker answer NO, meaning that, in this case, the document is consistent.



5. Conclusions and Future Works

Hypermedia documents constitutes a well-established case of reactive system for which non-traditional testing techniques have been proposed. The hypermedia authoring tools so far implemented have no general well-accepted solution to testing and consistency problems. To integrate some formal approach to these authoring tools so that could be transparently used by hypermedia authors seems to be a promising task. Our approach intends to do this role. Some others applications, based on the high interactiveness and implemented using some kind of concurrent event controller, can be well treated following the same directions just pointed (e.g.: virtual-reality applications). Finally, our formal approach can also be taken as one providing timed-

automata semantic to the class of the event-driven concurrent systems (hypermedia documents are a case of). However, one question remains without a practical answer: how we can automatically extract, from the validation process, the necessary set of corrections to fix the troubles found on the just verified model? We presume that if we use a technique that maps from the document specification directly to a timed automata model, we can get, on the inverse way, the set of the changes indicated by the counter-examples traces obtained in the model-checking phase. So, we claim that our timed-automata based approach is a natural way to provide a formal validation layer to hypermedia authoring tools, allowing to the document author to find out an inconsistency and fix it using the information given by the counter-example traces model-checked directly on the state machine model. Besides, the timed-automata model is much more close to the hypermedia description languages so that it seems to be a more natural semantic model to event-driven hypermedia models.

To extend these ideas to architectural description languages, providing a formal validation layer to architectural design, we need to identify in the ADL the mechanisms available to describe how components interact each other. Causal relations among the nodes in a document embody this correspondent concept for the event-driven hypermedia models: the nodes are components; the links are the connectors and the meeting points keep the information about how the components are related. Each component needs to be modeled as a timed automata and each connector needs to be modeled as a synchronizer machine. It is important to stress that, as more expressive the description language describes the component behavior, as more complex the timed automata generated and the validation complexity are. In spite of this pragmatic aspect, this translation phase can ever be automatically done if the ADL´s semantic would be rigorously defined.

6. Appendix

6.1. The translated UPPAAL model

```
chan startE1, startE4,
     E1start, startE2, startE3,
     E4start, startE5, startE6,
     E6start, startE7,
     E3stop, E5stop, startE8,
     E2stop, E7stop, stopE8;
chan u1,u2;
int chain, E8atP;
process E1{
state p, o, S0;
init p;
trans
o -> p {guard chain==1; sync u1!; },
p -> S0 {sync startE1?; },
S0 -> o {sync E1start!; };}
process E2{
clock d;
state p, o{d<=10};
init p;
trans
p -> o {sync startE2?; assign d:=0; },
o -> p {guard d>=10; sync E2stop!; };}
process E3{
clock d;
state p, o{d<=9};
init p;
trans
p -> o {sync startE3?; assign d:=0; },
o -> p {guard d>=8; sync E3stop!; };}
process E4{
state p, o, S0;
init p;
trans
o -> p {guard chain==1; sync u2!; },
p -> S0 {sync startE4?; },
S0 -> o {sync E4start!; };}
process E5{
clock d;
state p, o{d<=9};
init p;
trans
p -> o {sync startE5?; assign d:=0; },
o -> p {guard d>=8; sync E5stop!; };}
process E6{
clock d;
state p, o{d<=7}, S0;
init p;
trans
o -> p {guard d>=7; },
p -> S0 {sync startE6?; },
S0 -> o {sync E6start!; assign d:=0; };}
process E7{
clock d;
state p, o{d<=6};
init p;
trans
p -> o {sync startE7?; assign d:=0; },
o -> p {guard d>=6; sync E7stop!; };}
process E8{
state p, o;
init p;
trans
p -> o {sync startE8?; assign E8atP:=0; },
o -> p {sync stopE8?; assign chain:=1,
E8atP:=1;};}
process L1{
state S0, S2, S1, S3;
commit S2, S1, S3;
init S0;
trans
S0 -> S1 {sync E1start?; },
S1 -> S2 {sync startE2!; },
S2 -> S3 {sync startE3!; },
S3 -> S0 {};}
process L2{
state S0, S1, S2, S3;
commit S1, S2, S3;
init S0;
trans
S0 -> S1 {sync E4start?; },
S1 -> S2 {sync startE5!; },
S2 -> S3 {sync startE6!; },
S3 -> S0 {};}
process L3{
clock d;
state S0, S1{d<=3}, S2, S3;
commit S3;
init S0;
trans
S0 -> S1 {sync E6start?; assign d:=0; },
S1 -> S2 {guard d==3; },
S2 -> S3 {sync startE7!; },
S3 -> S0 {};}
process L4{
state S0, S1, S2;
commit S1, S2;
init S0;
trans
S0 -> S1 {sync E3stop?; },
S0 -> S1 {sync E5stop?; },
S1 -> S2 {sync startE8!; },
S2 -> S0 {},
S1 -> S1 {guard E8atP == 0; };}
process L5{
state S0, S1, S2;
commit S1, S2;
init S0;
trans
S1 -> S2 {sync stopE8!; assign chain:=1; },
S0 -> S1 {sync E2stop?; },
S0 -> S1 {sync E7stop?; },
S2 -> S0 {};}
process user{
state S0, S1, S2;
init S0;
trans
S0 -> S1 {sync startE1!; assign chain:=0;
},
S0 -> S2 {sync startE4!; assign chain:=0;
},
S1 -> S0 {guard chain==1; sync u1?; },
S2 -> S0 {guard chain==1; sync u2?; };}
system user,
       E1, E2, E3, L1,
       E4, E5, E6, L2,
       E7, L3,
       E8, L4,L5;
```

6.2. A Rigorous Semantics to ROL

Semantic Functions:

$\mathcal{X}$: X→ TA $\mathcal{G}$: Γ→ ⟨TA, ρ⟩ , where ρ: Id → Id ∪ {nil} ∪ ρ[12]

$\mathcal{E}$: E→ ⟨TA, ρ⟩ $\mathcal{L}$: Λ→ TA

$\mathcal{K}$: K→ TA $\mathcal{A}$: A→ TA, where TA is the class of all Network of Timed Automata.

$\mathcal{X}$[[**comp** I Γ E Λ]] = $\mathcal{G}$[[Γ]]↓1 ‖ $\mathcal{E}$[[E]]↓1 ‖ $\mathcal{L}$[[Λ]]$_{(\ I\rightarrow [[\Gamma]]\downarrow 2\ \cup\ I\rightarrow [[E]]\downarrow 2\ )}$

where $<x_1,x_2,...,x_n>\downarrow i = x_i$, $1 \le i \le n$.

$\mathcal{G}$[[$\Gamma_1\Gamma_2$]] = ⟨ $\mathcal{G}$[[Γ_1]]↓1 ‖ $\mathcal{G}$[[Γ_2]]↓1 , $\mathcal{G}$[[Γ_1]]↓2 ∪ $\mathcal{G}$[[Γ_2]]↓2 ⟩

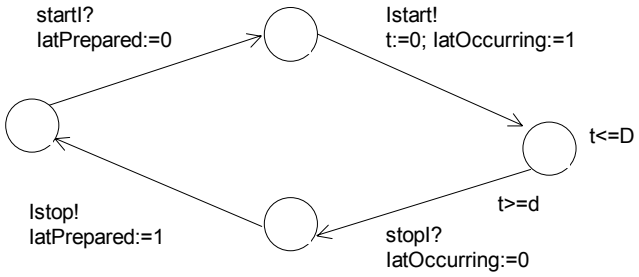
$\mathcal{G}$[[**comp** I Γ E Λ]] = ⟨ $\mathcal{L}$[[Λ]]$_{[[\Gamma]]\downarrow 2\ \cup\ [[E]]\downarrow 2}$ ‖ $\mathcal{E}$[[E]]↓1 ‖ $\mathcal{G}$[[Γ]]↓1 , I→$\mathcal{G}$[[Γ]]↓2 ∪ I→$\mathcal{E}$[[E]]↓2 ⟩

$\mathcal{G}$[[**term** I E]] = ⟨ $\mathcal{E}$[[E]]↓1 , I→$\mathcal{E}$[[E]]↓2 ⟩

$\mathcal{E}$[[E_1E_2]] = ⟨ $\mathcal{E}$[[E_1]]↓1 ‖ $\mathcal{E}$[[E_2]]↓1 , $\mathcal{E}$[[E_1]]↓2 ∪ $\mathcal{E}$[[E_2]]↓2 ⟩

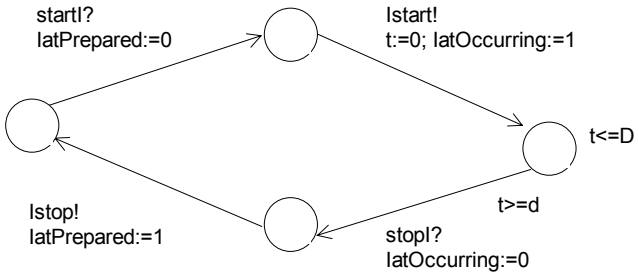
$\mathcal{E}$[[**pub ev** I Ξ T **desc** d D]] = ⟨ TA_I , ⟨ I ⟩ ⟩

where TA_I =



$\mathcal{E}$[[**ev** I **desc** Ξ T d D]] = ⟨ TA_I , nil ⟩

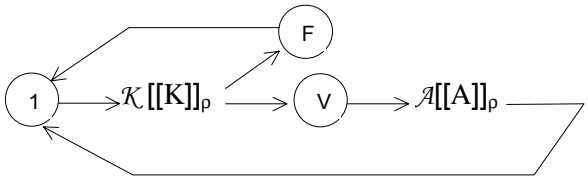
where TA_I =



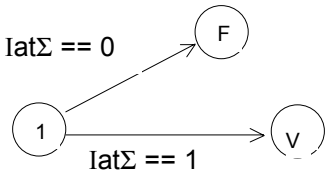
[12] This indeed is a least fixpoint definition in the context of the Domain Theory (Denotational Semantics).

$\mathcal{L}[[\textbf{link}\ I\ K \Rightarrow A]]_\rho = TA_I$ [13]

where $TA_I =$



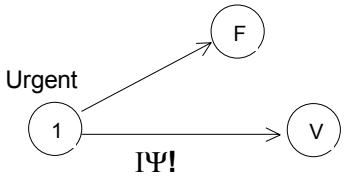
$\mathcal{K}[[I@\Sigma]]_\rho =$



, if $E(I)\rho$

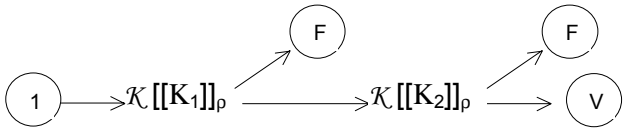
where $E(I)\rho = \begin{cases} \text{true, if } \exists X \in Id\ .\ \rho(X) = I \\ \text{false, if } Im(\rho) \subseteq Id \\ \bigvee_{X \in Id} E(I)_{\rho(X)} \text{ , else} \end{cases}$

$\mathcal{K}[[I.\Psi]]_\rho =$

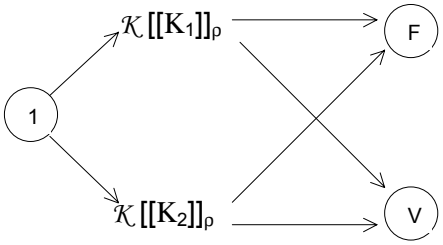


, if $E(I)\rho$

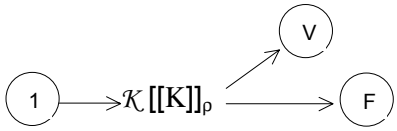
$\mathcal{K}[[K_1\ \textbf{and}\ K_2]]_\rho =$

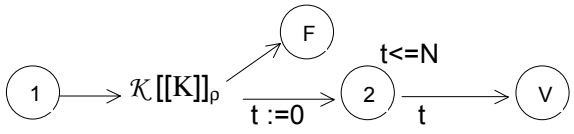


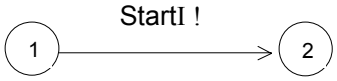
$\mathcal{K}[[K_1\ \textbf{or}\ K_2]]_\rho =$



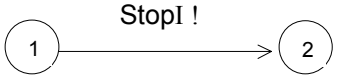
[13] For readability reasons, we don't overload the notation (ρ environment function) for handling the events name visibility. The event names in a same link must be visible in the same scope.

$\mathcal{K}[[\neg K]]_\rho =$ 

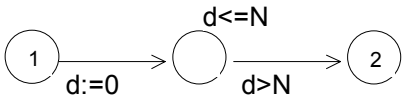
$\mathcal{K}[[K \oplus N]]_\rho =$ 

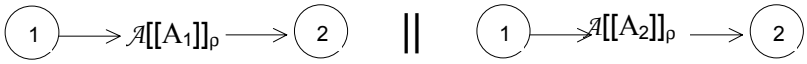
$\mathcal{A}[[\textbf{start}(I)]]_\rho =$ 

, if $E(I)\rho$.

$\mathcal{A}[[\textbf{stop}(I)]]_\rho =$ 

, if $E(I)\rho$.

$\mathcal{A}[[\textbf{delay}(N)]]_\rho =$ 

$\mathcal{A}[[A_1 \| A_2]]_\rho =$ 

$\mathcal{A}[[A_1;A_2]]_\rho =$ 

$\mathcal{A}[[A_1,A_2]]_\rho =$ 

7.References

[1] R.Allen,; D.Garlan “Formalizing architectural conection”. Proceedings of Sixteenth International Conference on Software Engineering, May 1994.

[2] Casanova, M.A. ; Tucherman, L.; Lima, M.J.; Rangel Netto, J.L.; Rodriguez, N.R.; Soares, L.F.G.; “The Nested Context Model for Hyperdocuments”. Proceedings of Hypertext’91. Texas. December 1991.

[3] Courtiat,J.P.; De Oliveira,R.C.; "Proving temporal consistency in a new multimedia synchronization model". ACM Multimedia'96, November.

[4] Courtiat,J.P.; De Oliveira,R.C.; "A Reachability Analysis of RT-LOTOS Specifications". ACM Multimedia'96, November.

[5] Kim G.Larsen, Paul Pettersson, Wang Yi. UPPAAL in a Nutshell. International Journal on Software Tools for Technology Transfer; 1997.

[6] Kim G.Larsen, Paul Pettersson, Wang Yi. Diagnostic Model-Checking for Real-Time Systems. 1997.

[7] Soares, L.F.G.; “Modelo de Contextos Aninhados V2.2”; Technical report, TeleMídia Lab/PUC-Rio; 1997.

[8] R.Alur, D.L.Dill. Automata-theoretic Verification of Real-Time Systems. John Wiley; 1996.

[9] R.Alur, D.L.Dill. A theory of timed automata. Theoretical Computer Science, 126: 183-235; 1994.

[10] Rodrigues, Leandro M. , Rodrigues, Rogério F., Muchaluat-Saade, Débora, Soares, Luiz F.G. Acrescendo Facilidades NCM a Documentos SMIL – VI SBMIDIA, 193-212; 1999.

[11] Soares, Luiz F.G., Rodrigues,Rogério F., Muchaluat-Saade, Débora, Modeling, Authoring, Formatting Hypermedia Documents in the HyperProp System , ACM Multimedia Systems Journal, Vol.8; No.2. March 2000, pp118-134.

[12]Santos,C.A.S.;Soares, L.F.G.; Souza, G.L.; Courtiat J.P.; "Design methodology and formal validation of hypermedia documents". ACM Multimedia'98, Bristol, September.

[13] Santos,C.A.S.; Courtiat J.P.;Soares, L.F.G.;Souza, G.L.; "Formal specification and verification of hypermedia documents based on the Nested Context Model"

[14] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.R. “Nested Composite Nodes and Version Control in an Open Hypermedia System Revisited”. I Prêmio Compaq de Estímulo à Pesquisa e Desenvolvimento em Informática, São Paulo, Brasil. November 1996; pp 99-116.