An approach for the verification of the temporal consistency of NCL applications*

S. Yovine
CONICET-UBA
syovine@dc.uba.ar

A. Olivero
UNSAM
aolivero@unsam.edu.ar

D. Monteverde
UBA
dm4e@dc.uba.ar

L. Córdoba
UNSAM
lcordoba@unsam.edu.ar

G. Reiter
UNSAM
greiter@unsam.edu.ar

ABSTRACT

NCL is used to write interactive applications for DTV. This paper presents the first steps of our approach towards the formalization of the semantics of NCL. We use Time Petri Nets (TPN) as target formalism for giving mathematically precise meaning to NCL. We rely on Visual Timed Scenarios (VTS) to graphically specify the behavioral properties the NCL application should satisfy. We sketch the method to translate NCL programs into TPN models and to verify VTS properties on the obtained model. We illustrate the approach with "O Primeiro João" from `club.ncl.org.br`.

1. INTRODUCTION

Considering the important economical and social impact of IDTV, related to the forseen development of t-commerce, t-learning, social TV, etc., it is imperative to have a productive development process which ensures the correctness and efficiency of the content to be broadcasted. This means that the timing requirements imposed by the concurrent execution, communication and synchronization of the components (audio, video, user, etc..) participating in the broadcasted software must be consistent. The inherent complexity of the mechanisms offered by the Nested Context Language (NCL)[1] standard turns infeasible verifying temporal consistency without the assistance of appropriate automated frameworks. However, there are currently no tool-supported rigorous methods for developing NCL applications. The available tools only validate the syntactical correctness and test simple applications, ignoring fundamental aspects related to ensuring temporal consistency. To address this problem, this work proposes a formal methodology and associated tools for verifying whether the NCL application meets specified timing requirements.

Formal verification reduces software production costs by capturing design errors early in the development process. It is not opposed to testing, but complements it by ensuring that certain properties hold at some abstraction level. Moreover, formal semantics of programming languages enable using model-based testing which greatly improves the time and quality of testing by automating the process of test-case specification and generation. Last but not least, formal semantics is very important for developing correct compilers and runtimes.

A way of enhancing the usability of formal techniques in model-driven system design and analysis consists in resorting to visual languages capable of visually presenting application semantics in a clear, precise way, specially in the context of event-based systems, such as interactive multimedia applications. Following this idea, in this paper we adopt Visual Timed Scenarios (VTS) [6] as a language for specifying behavioral properties of applications described in NCL. In order to make possible the verification of these properties using available tools we devise a compositional translation from NCL to Time Petri Nets (TPN). This allows the use of existing model-checking tools for verifying whether the translated NCL application satisfies the VTS property.

This work is an instance of the diagram depicted in Fig. 1, which shows the architecture of the GAMBETAS project, orchestrated around two subprojects. The goal of SP1 is to study different formalisms for mathematically expressing the meaning of NCL in the best possible way. SP2 is in charged of extending VTS with appropriate concepts to deal with NCL-specific properties for IDTV. This paper presents the preliminary work carried out with TPN as modeling language and plain VTS as property specification language. It sketches a compositional translation from NCL to TPN and resorts to VTS2TINA [13] and TINA [3] for model-checking.

Outline The paper is structured as follows. Sec. 2 recalls the syntax and semantics of VTS and illustrates it by means of an example from `club.ncl.org.br`. Sec. 3 briefly reviews Time Petri Nets (TPN) and the approach for checking whether a TPN satisfies a VTS scenario. Sec. 4 presents the translation of NCL into TPN and it applies the verification approach to the NCL application "O Primeiro João". We

*Partially supported by Ministerio de Planificación Federal, Inversión Pública y Servicios, Argentina, under project "Ginga-oriented Automated Methodology for Better Embbeded Television Application Software (GAMBETAS)", and PICT PAE 02287.

[1] `http://www.ncl.org.br`

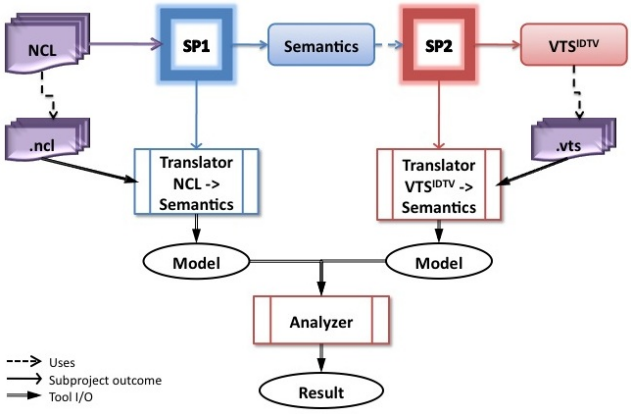

Figure 1: Project architecture and tool flow

close the paper with related and future work.

2. VISUAL TIMED SCENARIOS

Visual Timed Scenarios (VTS) [6][2] is a language to describe *event patterns*, which can be regarded as simple, graphical depictions of predicates over traces (time-stamped executions), constraining expected behavior. It basically features annotated partial orders of relevant events, denoting a (possibly infinite) set of matching traces. Violation of verification goals for models such as freshness, bounded response or event correlation can naturally be expressed in VTS.

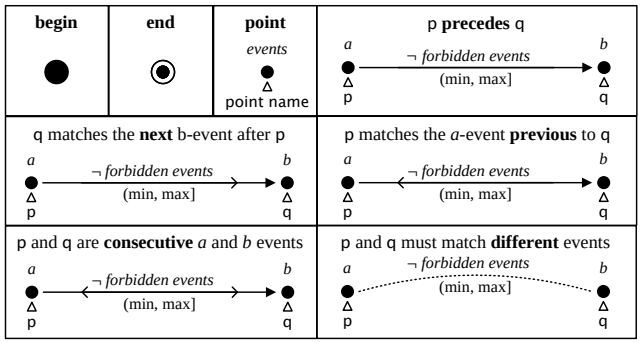

Figure 2: VTS graphical notation

Fig. 2 shows the VTS graphical notation used in this paper. The basic elements are points connected by lines and arrows. Points are labeled by sets of events, meaning that the point stands for an occurrence of one of the events in an execution. A big full circle stands for the *begin* of the execution, and two concentric circles correspond to its *end*. Triangles below points are used to display optional point names, needed for the formal notation. VTS can represent *precedence relations* and *temporal distances* between points; and sets of events which are *forbidden* between them. An arrow between two points specifies a *precedence* relationship. Arrow labels specify *forbidden events* between points. A double forward arrow means "the next" occurrence of the event of the destination point (i.e., shorthand for labeling the arrow with the destination's label). A double backward arrow meas "the previous" occurrence of the event of the source point (i.e., shorthand for labeling the arrow with the source's label). A dashed line linking two points expresses a *temporal distance* between them. Dashed lines can also be labeled with forbidden events. VTS can also identify the *first* or the *last* event in a set. The graphical representation for the first event in a set is a point linked to each point in the set by dotted lines ending in small empty circles. The notation for the last event uses full circles. VTS has more primitives which increase its expressive power. The interested reader is referred to [6].

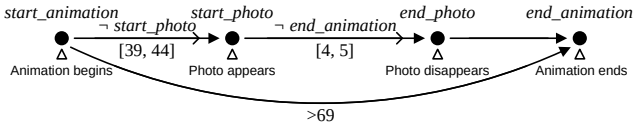

Figure 3: VTS scenario from "O Primeiro João"

Here, we informally introduce VTS through a simple, yet illustrative, example. Consider the VTS scenario depicted in Fig. 3. It specifies a behavior observed when running the interactive NCL application "O Primeiro João". The scenario expresses that the "animation" video lasts more than 69s. Between 39s and 44s after animation starts, a photo appears, and there is no other photo starts inbetween. It disappears between 4s and 5s later, before animation ends. It is important to stress that this scenario has been obtained without looking at the actual NCL document.

The scenario of Fig. 3 does not involve any user interaction. In "O Primeiro João" there are several examples of that, one is precisely the component `interactivity` which enables and disables interactivity through the INFO button. The first two scenarios of Fig. 4 express that interactivity is switched off and on by pressing the INFO button. The third scenario states that it is possible to repeatedly set on interactivity without setting it off. Clearly, this scenario is NOT satisfied in our example.

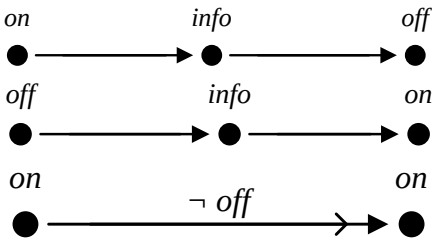

Figure 4: Example of scenarios with user interaction

The reader is referred to [6] for a formal definition of the syntax and semantics of VTS. Intuitively, the semantics of VTS assigns to each scenario a set of traces satisfying it. Labeled points represent events in the traces, they can match a particular position in a trace if the event in that position is among the allowed events associated to the point.

A *matching* is a mapping between points in a scenario and positions in a trace, exhibiting how the trace satisfies the scenario. A trace σ *satisfies* a scenario $\mathcal{S}$ (noted $\sigma \vDash \mathcal{S}$)

[2] `http://lafhis.dc.uba.ar/vts`

iff there exists at least one matching between them. For instance, the sub-trace (*start_anim*, 12.4) (*start_photo*, 53.1) (*end_photo*, 57.9) (*end_animation*, 83.2) has a matching with the scenario of Fig. 3. However, if the time stamp for event *end_photo* is 57, there is not matching, because it violates the time restriction [4,5].

In [6] the notion of *conditional* scenario was introduced. This allows expressing universal properties. The verification of conditional scenarios is done by translating them into a set of existencial scenarios that violate the property. An example of conditional scenario is depicted in Fig. 5. It expresses that each time interactivity switches from off to on (antecedent, in black), then the user must have pressed the INFO button in between (consequent, in gray).

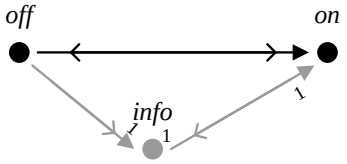

Figure 5: Conditional scenario

3. TIME PETRI NETS

Time Petri Nets (TPN) [4] are a widely used formalism for timed systems. TPN extend Petri nets with temporal intervals associated with transitions. Fig. 6 summarizes the graphical notation for TPN.

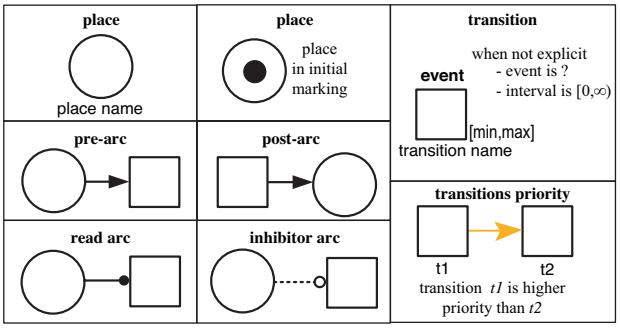

Figure 6: TPN graphical notation

Two TPN can be combined by parallel composition into one TPN where transitions with the same label are merged. The parallel composition between two TPN, $\mathcal{N}_1$ and $\mathcal{N}_2$, is denoted as $\mathcal{N}_1 \| \mathcal{N}_2$. See [4] for the formal definition.

Fig. 7 depicts the TPN of part of "O Primeiro Joao". It has been obtained from the NCL document by applying the translation procedure from NCL to TPN of section 4.

3.1 Model-checking

The problem of checking whether a system under analysis (SUA) modeled as a TPN $\mathcal{N}$ satisfies a VTS scenario $\mathcal{S}$ is solved in the following way. The tool VTS2TINA presented in [13] translates $\mathcal{S}$ into a TPN (observer) $\mathcal{T}_\mathcal{S}$ which recognizes matching traces. The TPN $\mathcal{T}_\mathcal{S}$ is composed with the TPN $\mathcal{N}$ of the SUA to check whether a matching execution exists, by using available model checking tools for TPN, e.g., TINA [3]. Specifically, the model-checking problem consists in verifying whether there exists an execution that reaches a state where place `accept` of $\mathcal{T}_\mathcal{S}$ is marked, and remains marked thereafter. See [13] for further details.

4. VERIFICATION OF NCL

In order to complete the verification method, we need a procedure to translate a NCL application to a TPN model. For the "O Primeiro João" case study, the TPN showed in Fig. 7, modeling the behavior defined by the NCL document, have been constructed manually, but systematically applying the rules described hereinafter. We are currently developing a tool to generate TPN automatically from NCL documents. In this section, we restrict to a subset of NCL comprising media components and connectors. In particular, we focus on the common library of causal connectors defined in the file causalConnBase.ncl distributed with the case study. The main idea is to translate each connector (onBeginStart, onBeginStartDelay, etc.) and each media (audio, video, music, user interaction, etc.) as a single TPN component. This approach allows to construct the TPN of the NCL document in a compositional way and to reuse translations of frequent pieces of code (typically connectors). For lack of space, we only show parts of NCL syntax. The reader is refered to the NCL manual and full example code for details.

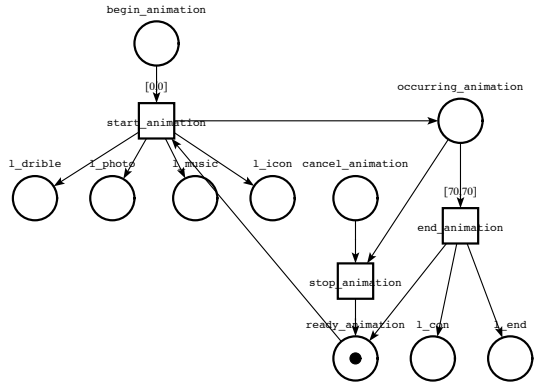

Figure 8: TPN of the media component `animation`

Fig. 8 shows the TPN of the media component `animation`. There are 4 basic places[3], namely: `begin`, `ready`, `occurring`, and `cancel`. Initially, there is a token in `ready`. Transition `start` is fired immediately when a toke is put in `begin`. This is done by transition `init` (Fig. 7). When `start` fires, it puts a token in `occurring`, modeling that the media component is executing. It also puts tokens in places associated with links bound to the "onBegin" role of the media. In this case, these are `l_drible`, `l_photo`, `l_music`, and `l_icon`. Transition `end` models the natural end of the media component, that is, when the player associated with the media ends playing. In the case of animation, this happens 70 seconds after start, which is modeled by the interval $[70, 70]$ attached to `end`. Transition `end` puts a token back to `ready`, meaning that the media can be started again. It also puts tokens in places associated with links bound to the "onStop" role of

[3]For readability, we omit the suffixes `_animation`, etc., when clear from context

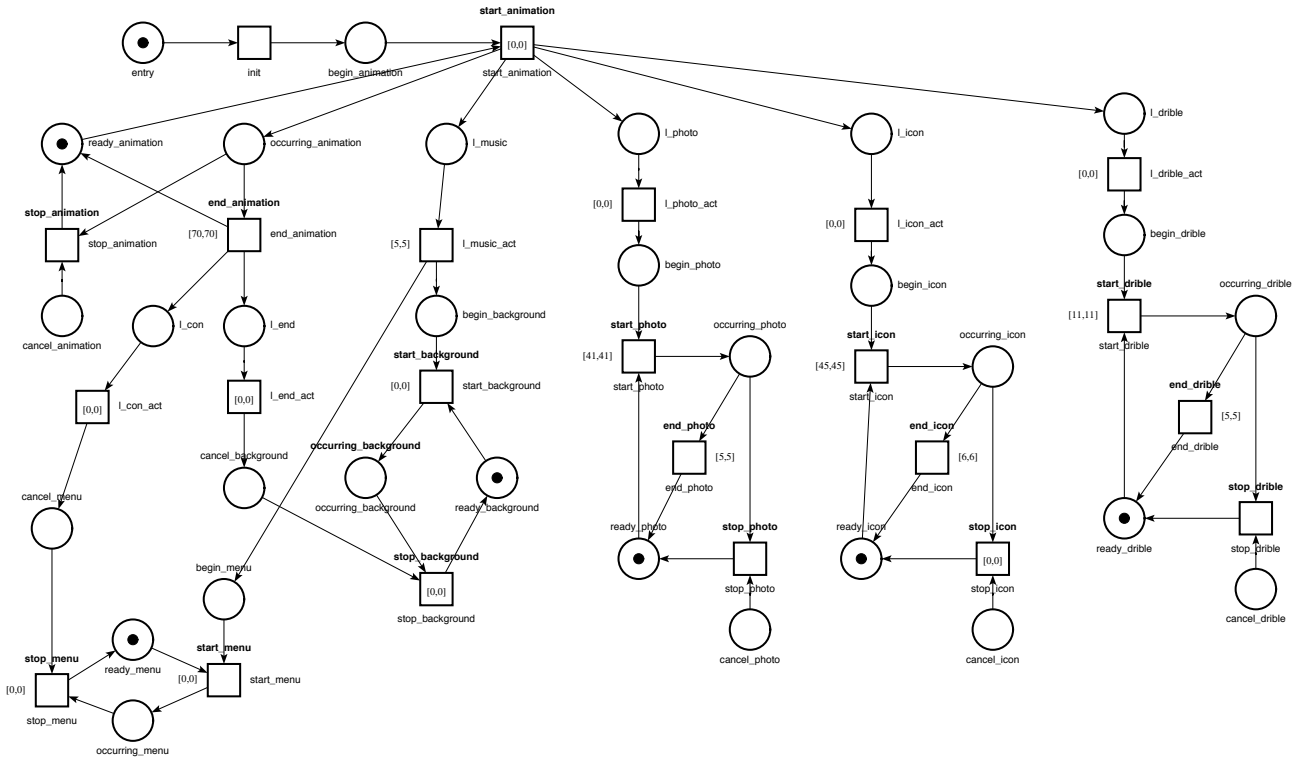

Figure 7: TPN of a part of "O Primeiro João"

the media. In this case, they are `l_con` and `l_end`. Place `cancel` is used to put a token to stop the media component through a link binding role "stop".

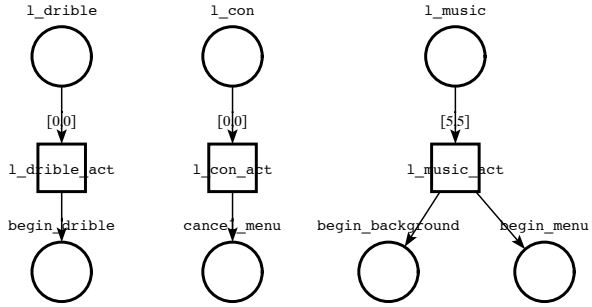

Figure 9: TPN of three links

Consider the following NCL link from "O Primeiro João":

```
<link id="lMusic" xconnector="conEx#onBeginStartDelay">
<bind role="onBegin" component="animation"/>
<bind role="start" component="background">
<bindParam name="delay" value="5s"/></bind>
<bind role="start" component="menu">
<bindParam name="delay" value="5s"/>
</bind></link>
```

Fig. 9 (right) shows the TPN of this link. It starts components background and menu with a delay of 5 seconds. The one on the left models the link

```
<link  id="lDrible" xconnector="conEx\#onBeginStart">
```

by immediately starting component drible. The one in the middle models the link

```
<link xconnector="conEx#onEndStop">
<bind role="onEnd" component="animation" ... />
<bind role="stop" component="menu"/></link>
```

by immediately stops component menu. Fig. 10 shows an example of translation of a switch node, e.g., the menu component of "O Primeiro João". Notice that the TPN of the switch is not part of the TPN shown in Fig. 7.

We have sketched the translation rules for media components, switch nodes, links and causal connectors. The TPN of the NCL document is therefore obtained in a modular way. We have illustrated the overall idea with our case study. We are ready now to carry out the verification of the desired properties on the NCL application. In our experiment, we use the VTS scenario of Fig. 11, which extends the scenario shown in Fig. 3 by adding the media icon and including a superposition condition between it and the media photo.

We proceed as was described in Sec. 3, by running TINA on the TPN obtained by composing the TPN of the NCL document shown in Fig. 7, and the TPN of the VTS scenario of Fig. 11 obtained with the tool VTS2TINA. It takes a few seconds to generate the TPN of the VTS scenario and verify the composed net. The verification result is `true`, meaning

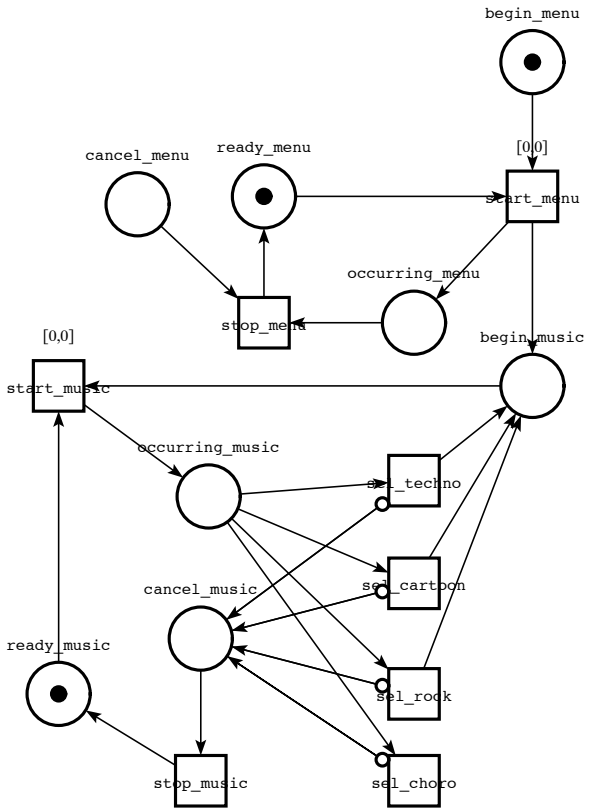

Figure 10: TPN of the switch node "menu"

that all the time constraints on the medias are satisfied and the superposition between two of them effectively occurs.

4.1 Interactivity

The behaviors presented so far did not consider interaction with the user. To illustrate how interactivity is handled, we resort to the `interactivity` component of the running application. Fig. 12 shows part of the TPN of this component[4]. Here, the interaction is reduced to pressing button INFO. For this, the TPN has two distinguished places, namely: `sel_info` and `enabled_info`. The former is to be connected with the TPN modeling the behavior of the user. An example of such behavior is shown in Fig. 13. In this case, the INFO events have a minimal inter-arrival time of one second. Besides, for an INFO event to be taken into account, the `enable_INFO` place has to be marked, that is, the `interactivity` component has to be waiting for it, and the `sel_INFO` place has to be empty, that is, there must not be a pending previous INFO event. For a discussion on different strategies for modeling the interaction of a reactive application with its environment, the reader is referred to [15].

We have verified that the first two scenarios of Fig. 4 and the conditional scenario of Fig. 5 are satisfied in the TPN model of the NCL application "O Primeiro João". We have also checked that the third scenario of Fig. 4 does *NOT* hold, as expected.

[4]Recall that arrows between transitions are priorities.

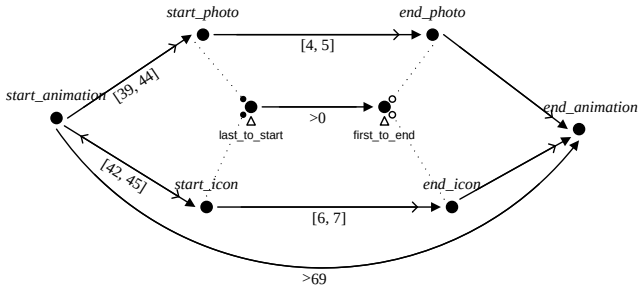

Figure 11: VTS scenario from "O Primeiro João"

5. RELATED WORK

There has been very little work on the specific topic studied by this paper. The main reason for this is the recent adoption of Ginga-NCL for ITVD. The only, and very preliminar, research paper we are aware of about checking temporal consistency of NCL documents is [9]. That article proposes a translation of version 1.0 of the underlying modeling framework of NCL, namely Nested Context Model (NCM), into Timed Automata (TA) [2]. Such work focuses on the validation of NCL 1.0 and 2.0, less expressive than 3.0. Moreover, there is no available tool implementing the approach nor reported case studies. Other previous works relate to SMIL (Synchronization Multimedia Integration Language), versions 1.0 and 2.0, less expressive than NCL 3.0. For instance, [11] presents the editing tool Madeus, based on temporal constraint networks [7], weaker than TPN, [10] proposes a verification framework for SMIL 1.0 based on Argos [12], and [14] proposes a semantics of SMIL 2.0 based in RT-LOTOS and model checking based on TA. None of these tools are currently available. Earlier, [8] developed Timed Stream Petri Nets for modeling multimedia systems and [16] extended timed automata in the same direction. There is no tool support for such modeling formalisms. More recently, [5] proposed a verification technique for SMIL based on Petri Nets and [17] used the PN-based tool SAM for checking temporal consistency of a multimedia document. Also, [1] used TA with deadlines (introduced in [16]) for model-checking a multimedia greetings card. Those works only present examples without giving formal semantics to any supporting language.

6. CONCLUSIONS AND FUTURE WORK

This paper proposes an approach and tool support for verifying complex properties on NCL documents, encompassing temporal consistency. For this, we devised a TPN-based semantics of a significant subset of NCL. In our framework, properties are expressed with the visual language VTS. Verification is carried out by using the tool VTS2TINA [13] to generate a TPN from a VTS scenario, and then using TINA [3] to check whether the TPN of the NCL document satisfies the scenario. The approach has been illustrated with the case study used for teaching NCL.

From the preliminary experimental results obtained with this case study, we believe VTS is an adequate formalism for specifying complex timing properties of interactive multimedia applications, implemented in NCL, in an intuitive and compact way. We intend to extend VTS in order to be able to express other kinds of properties, comprising layout,

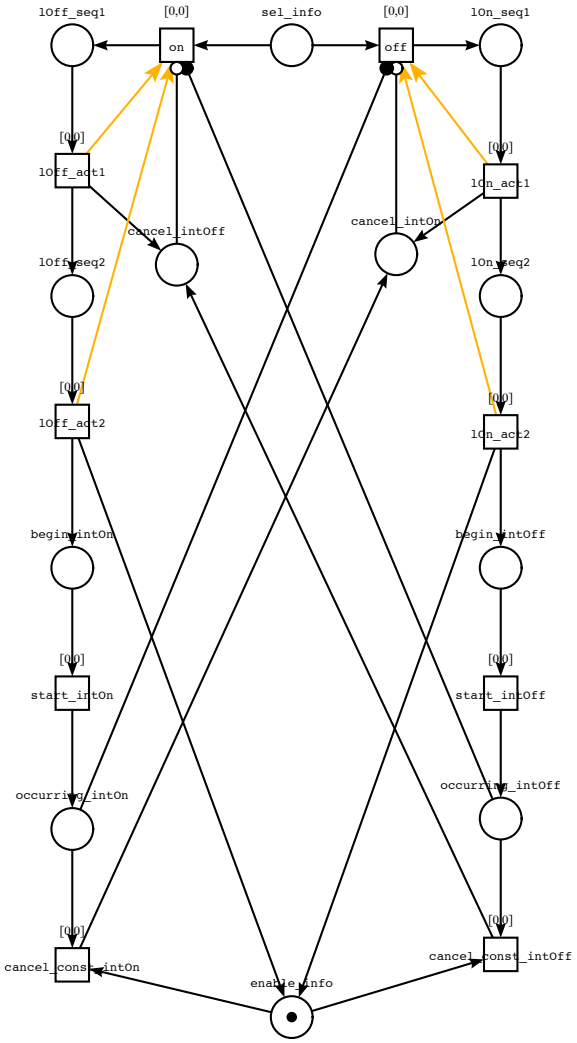

Figure 12: TPN of component `interactivity`

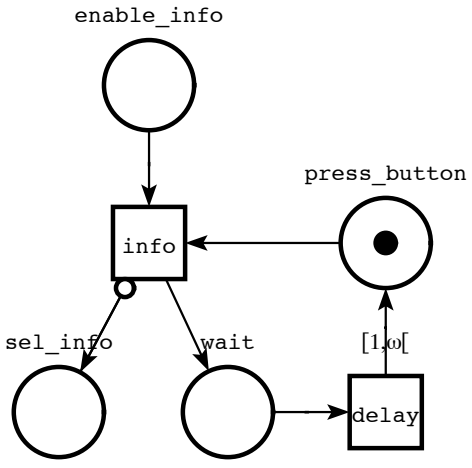

Figure 13: User behavior

transitions, etc. We are currently working on the automatization of the translation from NCL to TPN, the extension of the current semantics to a larger subset of NCL (including interaction with devices and lua-based scripting), and the study of other formalisms to determine which is the best suited for verifying NCL documents.

7. REFERENCES

[1] K. Altisen, G. Gossler, A. Pnueli, J. Sifakis, S. Tripakis, and S. Yovine. A framework for scheduler synthesis. In *IEEE RTSS'99*.

[2] R. Alur, D. Dill. A Theory of Timed Automata *TCS* 126(2): 183-235 (1994).

[3] B. Berthomieu and F. Vernadat. Time Petri Nets Analysis with TINA. In *(QEST'06)*, 2006.

[4] B. Berthomieu and F. Vernadat. State Space Abstractions for Time Petri Nets. In *Handbook of Real-Time and Embedded Systems*, 2007.

[5] S. Bouyakoub, A. Belkhir. H-SMIL-Net: A Hierarchical Petri Net Model for SMIL Documents. 10th Int. Conf. on Computer Modeling and Simulation, p. 106-111, 2008.

[6] V. Braberman, N. Kicillof, A. Olivero. A Scenario-Matching Approach to the Description and Model Checking of Real-Time Properties. *IEEE TSE*, 31(12), 2005.

[7] R. Dechter, I. Meiri, J. Pearl. Temporal Constraint Networks. *Artificial Intelligence* 49:61-95, 1991.

[8] M. Diaz, P. Sénac. Time Stream Petri Nets: A Model for Timed Multimedia Information. Application and Theory of Petri Nets, 1994, p. 219-238

[9] M. Fagundes Felix, E. Hermann Haeusler, L. F. Gomes Soares. Validating Hypermedia Documents: a Timed Automata Approach. TR, PUC-Rio Inf. MCC21/02.

[10] M. Jourdan. A formal semantics of SMIL: a web standard to describe multimedia documents. *Computer Standards and Interfaces* 23(5):439-455, November 2001, Elsevier.

[11] M. Jourdan, N. Layaida and L. Sabry Ismail. MADEUS: an authoring environment for multimedia documents. IEEE Int. Conf. on Multimedia Computing and Systems, 1997.

[12] F. Maraninchi. Argonaute: Graphical Description, Semantics and Verification of Reactive Systems by Using a Process Algebra. LNCS 407, p. 38-53, 1989.

[13] D. Monteverde, A. Olivero, S. Yovine, and V. Braberman. VTS-based specification and verification of behavioral properties of AADL models. ACES-MB'08, Toulouse, France, 2008. Also available as TR, DC. FCEN. UBA.

[14] P. Sampaio, C. Lohr, J. P. Courtiat. An integrated environment for the presentation of consistent SMIL 2.0 documents. ACM Symp. on Document Engineering, 2001.

[15] J. Sifakis, S. Tripakis, S. Yovine. Building models of real-time systems from application software. Proceedings of the IEEE 91(1): 100-111 (2003).

[16] J. Sifakis, S. Yovine. Compositional Specification of Timed Systems. STACS'96, LNCS 106, p. 347-359.

[17] H. Yu, X. He, Y. Deng, L. Mo. Formal analysis of real-time systems with SAM. ICFEM'02, LNCS 2495.