Towards a hybrid approach for supervising interactive adaptive systems

Damien Mondou, Armelle Prigent, Arnaud Revel, Nicolas Rempulski

Towards a hybrid approach for supervising interactive adaptive systems

Damien Mondou, Armelle Prigent, Arnaud Revel, Nicolas Rempulski

1 Research area

The analysis of users' behavior (for example the visitors of a website or tourists using a dedicated mobile application) tackle a common problem: the rapid extraction of path's patterns. These patterns will go back to a user profile characterization and their behaviors. The goal is to achieve a dynamic content adaptation according to these behaviors and the specific context of navigation. Various works were interested in how to acquire and dynamically analyze navigation datas through contents by extracting relevant information about users's interest or motivation and proposing dynamic adaptations rules relevant for the specific situation. Two approaches allow to address this problem: one based on **machine learning techniques** (i.e. reinforcement algorithms) for extracting representative information on the basis of past navigations logs, the other is based on the **dynamic observation of the behavior's** user and a set of scenario of modification rules.

Therefore our work intends to develop a generic framework, based on a formal model, for analyzing users' behavior and dynamic content adaptation using a hybrid approach combining learning and real-time observation/scenario modification.

1.1 State of the art

There are already a huge number of approaches related to the dynamic adaptation for interactive storytelling. We can classify them into three classes. First, **scripted approaches** [Vega and L., 2003], [Rempulski et al., 2009] are based on an expert system controlling the user experience. The scripted approaches generate the narrative that is closest to the expectations of the latter based on the observation that is made and compared to a set of predefined rules. Modeling such an architecture is a complex task as all the possible paths for the user must be defined by the designer. **Oriented agent approaches** aim to the emergence of narratives based on a set of agents' behaviors [Mateas and Stern, 2003]. The modularity of this way of modeling the system makes it easier to design but this approach makes it more difficult to control the quality and consistency of the generated model, what scripted approaches guarantee by their very structure. Finally, **hybrid approaches** try to find the balance between the user's freedom, and thus a high level of adaptation obtained with oriented agent approach, and the respect of the designer's frame (designed with scripted approaches). [Brian Magerko, 2003] offers such an hybrid interactive storytelling system, where the author specifies a generic framework without limiting the user's actions. The quality of the execution is carried out by a manager that dynamically analyzes the behavior of users to detect unanticipated behaviors.

The major difficulty stands in the representation of all the possible interactions between the user and the system. The model must garantee a high level of modularity, reusability and want be scalable enough to meet potential changes. In addition, supervision models requires having a formal model from which

it is possible to extract the most relevant paths. Moreover, as one of our purpose is to mix **machine learning** (by adding or removing interactions for example) and dynamic formal analysis, our execution framework needs to manipulate a modular and easily alterable model depending on the execution context. Moreover, we need a model that is robust enough for a large number of states and transitions (as we are working on interactive systems). [Li et al., 2013] describes *NeTA*, a model based on nested timed automata. A set of automata is stored in a stack and the system behaves as the present automaton on the top of the stack. Time passes uniformly in all the automata. This approach is an extension of timed automata presented in [Ah and Dill, 1994] by adding a switching context. This context happens by the actions *"push"* and *"pop"* or by changing the top of the stack automaton. A second extension of timed automata is presented in [David, 2003]. Hierarchical timed automata allow to model systems easily with a large number of states and transitions. Automata are encapsulated in localities which can themselves be encapsulated in other localities. This model respects the semantics of timed automata by the passage of transitions and to model a system in depth but does not offer a modular modeling as proposed by our model.

1.2 Contribution

The first question is, which model can we use to represent and supervise a large interactive system without being confronted to the hard task of modeling complex system and to overcome state space explosion problem? Here, we intend to extend the work of [Rempulski, 2013] offering a representation based on I/O automata of complex systems. The model handles many entities, offering the designer a paradigm based on a simplified model such as modularity, multiple-inheritance and reuse of components.

The second question is, how to combine in a single system supervision, management of operations (**top-down** approach) and integration into the pilotage rules determined on the basis of the observations of the actual user's behavior (**bottom-up** approach)? The **bottom-up** approach allows, after a learning phase, to extract relevant information by analyzing an interactive system during its very execution. The reinforcement learning techniques [Russell and Norvig, 2003] offer an iterative process to find an ideal sequence to achieve a goal in a given environment. The **top-down** approach aims to produce a representation of a system from a profile (user or context). [Wang et al., 2010] proposes a new language, ADAM, to specify adaptive structures. These structures are defined through three dimensions: the spatial dimension, the temporal dimension and a dimension managing the adaptive (location, quality of contents and techniques of interaction), and will be used to generate content in adequacy with the user's profile. Also based on the user profile, [De Virgilio, 2012] offers a modeling language, AML for the adaptive design of web applications at the conceptual level. Thanks to the basic primitives, the response and the structure of the information provided to the user is adapted to the execution context of the application.

Thus, unlike conventional approaches proposing an adaptation based on a model and a set of rules to re-scripting the activity, we wish here to offer a hybrid approach based on top-down control handset a bottom-up analysis. The aim is to bring these two approaches and to propose an iteration loop between learning and formal representation to improve the knowledge and dynamically into the model by relevance feedback. The different application areas have each specific features. Thus, if in the context of the narrative, interaction between entities is important, applying the model to interactive experiences in cultural sites requires increasing the model for the consideration of the spatial dimension.

2 A situation based model : *#Telling*

In this section, we present the situation based model that has been initiated in [Rempulski, 2013] and that we aim to extend and improve in the **bottom-up** part of our system. It has been combined with a tool developped at the L3I lab. *#Telling* is a software platform, developed in C#, allowing the designer to create a modular system, control its execution and perform property checks.

2.1 A three-layer framework

In our approach, modeling a system is divided into three parts:

- We first define the system abstractly, as reusable components. Thus, the designer defines a set of actants, with their behaviors, grouped into situations. This step creates **the abstraction layer**.
- By instantiating the actants into actors and situations into scenes, the designer creates **the implementation layer**.
- Finally, the system dynamical description is defined by orders the scenes previously created, by defining a set of plots. The scenes execution order is then created and constitutes the **dynamic layer**.

Figure 1 shows the order in which items have to be created. Here, the designer has defined three actants *A1, A2* and *A3*. The situation *Sit1* is composed of actants *A1* and *A2* and the situation *Sit2* of actants *A2* and *A3*. An actant can appear in many situations.

In the second layer, these actants are then instantiated in an entity called an Actor. Three actors are defined: *Act1* implementing the roles of actants *A1* and *A2* by multiple inheritance, the actant *Act2* implementing the role of the actant *A2* and the actant *Act3* implementing *A3*. The scene *Sc1* is an instantiation of the situation *Sit1* and is composed of actor *Act1*. The scene *Sc2* is an instantiation of the situation *Sit2* and is composed of actors *Act2* and *Act3*.

Finally, the designer arranges the scenes by creating three *plots* represented by the graph of the expected execution frame (from scenes to scenes). The execution order of the system can be the *Plot P1(Sc1)* $\rightarrow$ *Plot P2(Sc1)* $\rightarrow$ *Plot P3(Sc2)* or *Plot P1(Sc1)* $\rightarrow$ *Plot P3(Sc2)*.

This method presents two advantages. First, the design is based on three layers that helps modularity and reuse. The abstraction layer helps the designer to constructs generic entities (actants) and a generic way of combining them. The second one (instantiation layer) is devoted to the implementation of generic entities and supports modularity by multiple inheritance of actants into concrete actors and combinations of actors in a concrete situation of execution (so called a scene). Finally, the dynamic layer helps organizing those concrete scenes into execution schemes. This approach constitutes a great support for extensible systems and reuse of entities for modeling. The second contribution concerns the dynamic construction of the systems global finite state automata from behaviors. Here, the designer has only to describe atomic behaviors for actants. The automata executing situations is constructed automatically.

In order to improve the system, we have produced an example of a lift. This example, helps us to test the gain for modeling task (using the modularity paradigm to add floors) and to test the limit of automatic construction of the global system with *#Telling* (state space explosion). For this system, represented in Figure 2, we can define the actants *User, Cabin, Door, ControllerDoor* and *ControllerShifting*. These five actants could be grouped into two differents situations: *Floor0* and *Floor1*. By instantiating this situation into scene and actants into actors, we could define two plots, one per floor.

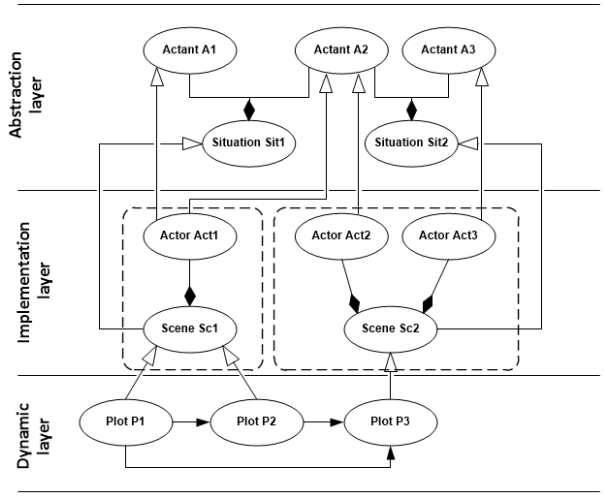

Figure 1: Modeling a system in *#Telling*.

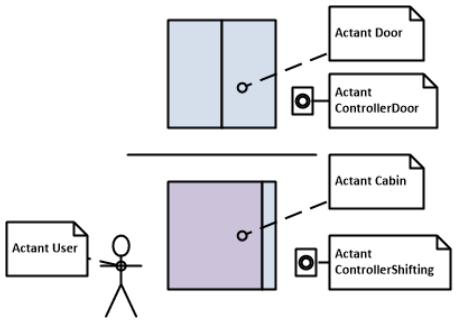

Figure 2: System for managing a two-floors lift.

2.2 Designing abstract entities

The **abstraction layer** defines an abstract model of the system. It defines the prototypic roles (actants) and the generic situations. Actants define a set of behaviors which are the atomic elements of our model. They represent an action making the actant evolve. Our model construction paradigm is based on these behaviors composition. The abstraction layer is defined by the tuple (A, S, V) with A is a set of actants, S is a set of situations and V is a set of variables.

Each entity defined in the system contains its own automata (whose construction is performed automatically), which can be synchronized to the other entities when they are grouped into situation.

2.3 Implementing entities in the story

Entities defined in the abstraction layer are generic. This second layer, called **instantiation layer**, is used to implement generic entities defined in the abstraction layer into concrete entities. Thus, actants are instantiated by *actors* and situations by *scenes*. This mechanics of instantiation allows to create several actors from a single actant model and allows the multiple inheritance (an actor implements the behavior of several actants). We define the instantiation layer by the following tuple (A_{cr}, S_c) with A_{cr} is a set of actors and S_c is a set of scenes.

The **actor** implements the role of one or more actant(s). Multiple inheritance, shown in Figure 3, allows the designer to specialize an actor according to the behavior of the different actants it implements.

2.4 Supervising the model

The **dynamic layer** defines the execution ordering of scenes declared in the instanciation layer. Here, the scenes are encapsulated in an entity: the **plot**. This overcoat specifies the elements necessary for the scheduling execution of the scene. The plot is defined by the tuple (sc, G, S_{sup}) with $sc \in S$ a scene of the instanciation layer, G a set of constraints and S_{sup} a set of successors plots.

A **constraint** imposes a value to the state vector of a specific actor, before the completion of the scene. It is defined by the tuple $(a, g(V_a))$ with $a \in A_{cr_s}$ an actor of the referenced scene s and $g(V_a)$ a constraint on the state vector of a.

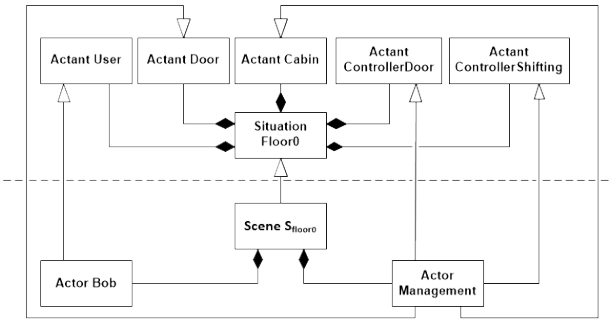

Figure 3: Principle of inheritance.

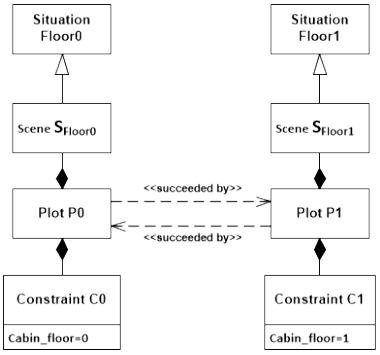

Figure 4: Plots representation.

We define two scenes S_{floor0} and S_{floor1} composed of *ControllerDoor, ControllerShifting, Cabin, Door* and *User*. Then, we define Two plots *P0* and *P1*. *P0* is composed of the scene S_{floor0}, and the plot *P1* is composed of the scene S_{floor1}. The plot *P0* is succeded by *P1* and has constraints indicating that the cabin is to floor 0. The plot *P1* is succeded by *P0* and has constraints indicating that the cabin is to floor 1. Figure 4, represents this case.

3 Results

We return to our example of lift modeling. We present in Figure 5 the evolution of the number of states and transitions according to the number of floors. Give results, we see that without appropriate tools, modeling a complex system is a difficult and tedious task.

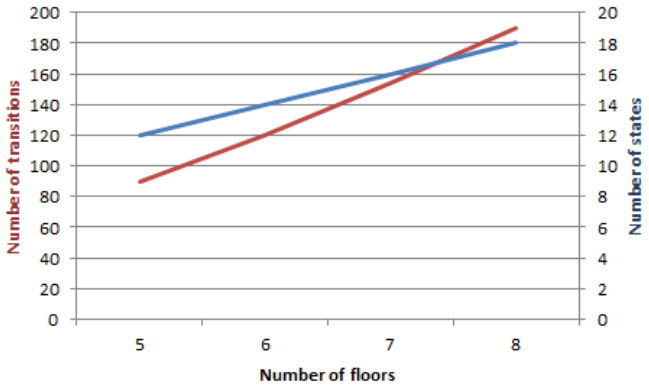

Figure 5: Evolution of the number of states and transitions.

The application domain of our work concerns mainly interactive experiences in museums. Our approach will be validated by the realization of games proposed by Nao robots piloted by *#Telling* in partner museums. With this intention, connection works with *#Telling*, our modeling tool, to Choregraphe, the management tool of the robot Nao, are ongoing. This connection will allow the designer to design both the architecture of the games it will offer to the museum's visitors and also the associated behaviors of the Nao robot.

4 Conclusion

In this paper, we have presented a model to facilitate the formal representation of complex systems. This model allows the automatic construction of systems based on simple behavior description. As an example, we have showned here the lift model. Now, we want to explore the topic of interactive experiences in museums piloted by the Nao robot. In this perspective, we wish to integrate a machine learning module to learn new behaviors for the system and dynamically add to the model. We also integrate time constraints whatsoever in the detection context or decision making (integration of the semantics of time UPPAAL [Larsen et al., 1997])

References

[Ah and Dill, 1994] Ah, R. and Dill, D. L. (1994). A theory of timed automata. *Thoretical Computer Science 126*, pages 183–235.

[Brian Magerko, 2003] Brian Magerko, J. L. (2003). Building an interactive drama architecture. *In First International Conference on Technologies for Interactive Digital Storytelling and Entertainment*, pages 226–237.

[David, 2003] David, A. (2003). *Hierarchical Modeling and Analysis of Timed Systems*. PhD thesis.

[De Virgilio, 2012] De Virgilio, R. (2012). Aml: a modeling language for designing adaptive web applications. *Personal and Ubiquitous Computing*, 16(5):527–541.

[Larsen et al., 1997] Larsen, K. G., Pettersson, P., and Yi, W. (1997). Uppaal in a nutshell. *Int. Journal on Software Tools for Technology Transfer*, 1:134–152.

[Li et al., 2013] Li, G., Cai, X., Ogawa, M., and Yuen, S. (2013). Nested Timed Automata. *Formal Modeling and Analysis of Timed Systems*, pages 1–15.

[Mateas and Stern, 2003] Mateas, M. and Stern, A. (2003). Façade: An experiment in building a fully-realized interactive drama.

[Rempulski, 2013] Rempulski, N. (2013). *Synthèse dynamique de superviseur pour l ' exécution adaptative d ' applications interactives*. PhD thesis.

[Rempulski et al., 2009] Rempulski, N., Prigent, A., Courboulay, V., Perreira Da Silva, M., and Estraillier, P. (2009). Adaptive Storytelling Based On Model-Checking Approaches. *International Journal of Intelligent Games & Simulation (IJIGS)*, 5(2):33–42.

[Russell and Norvig, 2003] Russell, S. J. and Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition.

[Vega and L., 2003] Vega, S. N. and L. (2003). A Petri Net Model for the Analysis of The Ordering of Actions in Computer Games. In *GAME ON 2003*.

[Wang et al., 2010] Wang, C., Wang, D.-Z., and Lin, J.-L. (2010). Adam: An adaptive multimedia content description mechanism and its application in web-based learning. *Expert Systems with Applications*, 37(12):8639 – 8649.