IADIS International Journal on Computer Science and Information Systems
Vol. 1, No. 1, pp. 30-41
ISSN: 1646-3692

A MODELING FRAMEWORK FOR COMPLEX BEHAVIOR MODELING AND INTEGRATION

Jean-Marc Perronne
MIPS, Université de Haute Alsace
12 rue des frères Lumière, 68093 Mulhouse, France

Alban Rasse
MIPS, Université de Haute Alsace
12 rue des frères Lumière, 68093 Mulhouse, France

Laurent Thiry
MIPS, Université de Haute Alsace
12 rue des frères Lumière, 68093 Mulhouse, France

Bernard Thirion
MIPS, Université de Haute Alsace
12 rue des frères Lumière, 68093 Mulhouse, France

ABSTRACT

The design of the control software for complex systems is a difficult task. It requires the modeling, the simulation, the integration and the adaptation of a multitude of interconnected entities and behaviors. In this context, the present paper describes a modeling framework. It proposes to objectify the behaviors which leads to a two-level modeling process based on three concepts: resources - software images of the controlled system - behaviors applied to these resources, and meta-behaviors, - i.e. means for behavior integration and adaptation. To complete the proposed modeling framework, a coherent model-based approach supported by model-checking tools, that ensures the development of validated applications, is considered. Finally, to illustrate the elements mentioned, this paper uses the control software of a walking robot as a running example.

KEYWORDS

Modeling Framework, Behaviors Modeling, Control Software, UML, Model Transformation, Validation

1. INTRODUCTION

The design of the control software for complex systems is a difficult task (Sanz R. et al, 2001). In particular, it requires means - i.e. concepts, notations and guides - for the integration and

adaptation of a number of local behaviors within the framework of global control. In this context, the present paper proposes a modeling framework which explains how to design complex software systems which are controllers. The basic concept proposed by this paper is that of Behavioral Objects, which consists in reifying the behaviors of a subsystem. This founding principle opens an important field of investigation of complex systems. In particular, it helps to model all the elements considered (subsystems, control laws and interactions) in a uniform way with objects (Thiry L., 2002). The notion of behavioral objects leads to an analysis guided by a two-level architecture that sets up three kinds of entities: resources, behaviors and meta-behaviors. The first conceptual level includes entities which model resources. The resources help to model the structure of the controlled system and to specify the available services to make this structure evolve. The second conceptual level includes entities which model behaviors and allow the control of the previous elements. A behavioral object can then be considered as a resource for behavioral objects of a higher order. These behavioral objects, called meta-behaviors, help to integrate, adapt and coordinate other behaviors; they represent the third concept of the architecture.

2. RUNNING EXAMPLE

The system used to illustrate the present approach (Figure 1) is an omnidirectional hexapod robot (Thirion B., Thiry L., 2002).

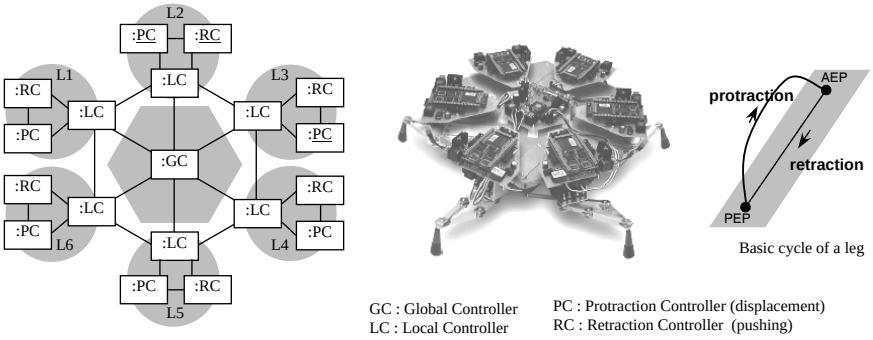


Figure 1. Hexapod Robot.

This mobile platform requires an efficient appropriate control architecture for the integration of a number of coordinated behaviors. Only the locomotion behavior will be considered here. A leg moves in a cyclic way between two positions AEP (anterior extreme position) and PEP (posterior extreme position). A leg is in retraction when it rests on the ground and pushes the platform forward. It is in protraction when it resumes its AEP. The control architecture is based on decentralized control; the local behaviors obtained with local controllers (LC) are applied to a leg (L) and a global controller (GC) coordinates the local behaviors. Each local controller can be decomposed into three simpler controllers: a retraction controller which allows the platform to move, a protraction controller which determines where

and how to reposition a leg and a controller which coordinates the previous two controllers. The proposed modeling framework helps to tackle the complexity of such a system.

3. BEHAVIORAL OBJECTS

The reification of behaviors (Beaudouin-Lafon M., Mackay W.E., 2000), also called objectification, helps to consider a behavior as an object; as such, it can be architectured. A behavior can be defined as a series of transformations applied to the state of a system; the state corresponds to an observable configuration of the system. This reification has numerous advantages. As the behavior is an object, it can also receive messages and be controlled by objects of a higher level; this property is used by meta-behaviors to integrate or adapt different behaviors.

This definition of behavioral objects leads to modeling at a two-level of analysis, Figure 2a. The first level includes all the resources evolving in any state space; resources typically model physical entities. The second level corresponds to the behavioral objects whose role is to control the resources in their state space. In this schema, the laws, the evolution rules or the constraints are systematically separated from the objects governed by these laws (For instance, in Figure 2b, LegBehavior and Leg).

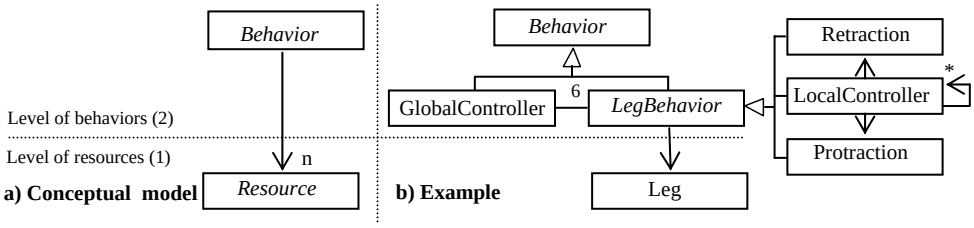


Figure 2. Behavior-Resource conceptual model

3.1 Organization of behavioral objects

The advantage of the proposed concept is to allow the use of the founding principles of object-oriented modeling (Booch G., 1994) for the description and the organization of the behavior space. The relations of classification and aggregation in particular prove important as they help to structure this space and allow the easier synthesis of flexible control software. Hence a network of behavioral objects (Figure 1) which provide a structure for the space of behaviors. Figure 2b proposes an extract of the organization of the behaviors necessary for the control of the legged robot. It notably shows the four main behavior/controller classes used to control this complex system:

- The global controller sets and controls the global motion of the platform.
- The local controller controls the leg motion according to the retraction and protraction phases.
- The retraction controller controls the leg at rest and so contributes to the global motion.

- The protraction controller brings the upheld leg into the position where it can contribute again to the motion of the platform.

The local leg-behavior results from the aggregation of a local controller, a retraction controller and a protraction controller. The global walking behavior is then obtained through the combination of a global controller with six local controllers.

3.2 Meta-behaviors

The previous part has explained how to organize local behaviors that are necessary for the control of a complex system. This part now presents the composition model for the integration of these local behaviors into the context of more global control. It starts from the observation that a behavior is an object; in this sense, it can be considered as a resource and controlled by higher order behaviors (meta-behaviors) (Figure 3).

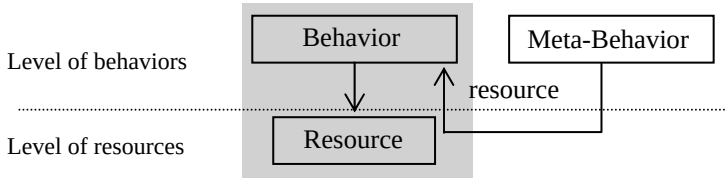


Figure 3. Meta-behavior

Behavioral objects are used to define the dynamic (i.e. control laws) which specifies how the internal state of a resource evolves within a given running mode; meta-behaviors are used to define the metadynamic which describes how the behaviors themselves evolve. A meta-behavior is a behavioral object which activates, inhibits, organizes, combines or adapts other behaviors. Two categories of meta-behaviors are currently identified: combinators and adapters.

3.2.1 Combinators

Combinators are particular behavioral objects in the sense that they can only be applied to behaviors. The most conventional combinators are given by sequential, parallel, repetitive or conditional behaviors, as shown in Figure 4.

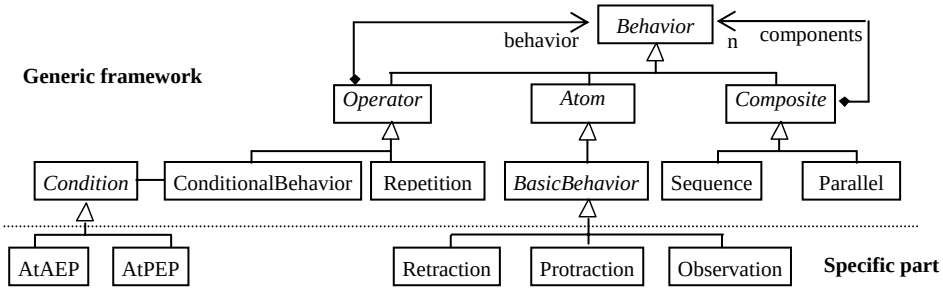


Figure 4. Combinators for the integration of behavioral objects

Figure 4 proposes a family of combinators that can be used for the integration of behavioral objects. These fall into five categories:

- *Basic behaviors* which represent "behavior atoms" to be combined for more complex activities. Retraction and Protraction belong to this category. In general, a basic behavior is directly applied to a resource.
- *Repetition behaviors* are currently used to define cyclic activities. This is the case of controls which must run constantly. For example, the locomotion behavior consists in repeating two behaviors – protraction and retraction – successively.
- *Conditional behaviors* help to add an activation condition (or guard) to a behavior. For example, it must be possible to suspend the retraction behavior when the leg stretch degree reaches its maximum threshold.
- *Sequence behaviors* allow the sequential combination of behaviors following the rule: "if a behavior is completed, then activate the next behavior". For example, the protraction behavior of a leg follows the retraction behavior (Figure 1).
- *Parallel behaviors* allow concurrent behaviors to be performed simultaneously. They play an important part in the control of complex systems where several activities must be carried out in parallel. For example, the global motion of the platform is obtained by combining, in parallel, the motion of the six legs.

3.2.2 Adapters

The concept of meta-behavior can be generalized for the adaptation or reconfiguration of behavioral objects. The modeling framework is refined so that a behavior can adapt/reconfigure another behavior. A behavioral object can be decomposed into two parts: a constant part and a variable or adaptable part which is used as a resource by an "adapter" meta-behavior (Figure 5a). It must be noted that, if need be, the adaptable part can be shared by several behaviors which would then benefit from any adaptation action.

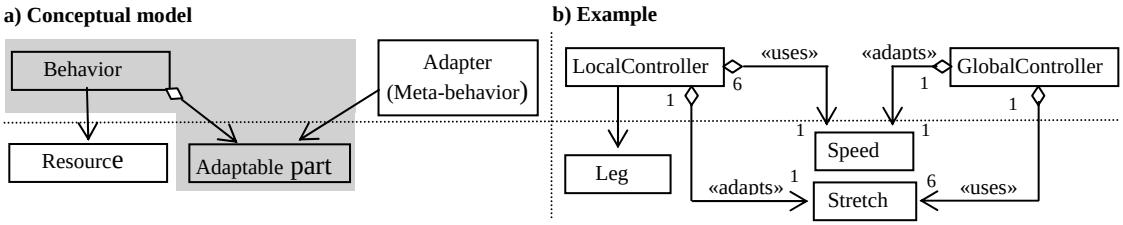


Figure 5. Meta-behavior adapter

An illustration of this principle is the management of the robot's motion. A global controller adapts the global speed of the platform according to the performance of the six local controllers (Figure 5b). The present case is in fact rather complex as it sets up a continuous process (global motion) which adapts the performance of six hybrid processes (leg behavior).

- The global controller uses the stretch degree provided by the leg controllers to adapt the global speed of the platform. Here, speed is the adaptable part of the six local controllers.
- The six leg controllers use the global speed of the platform adapted by the global controller in order to evaluate the control parameters of their legs.

This example has illustrated the case where a process of a higher level adapts a parametric process at a lower level.

4. BEHAVIORS AND VALIDATION

As it separates behavioral objects from resource objects, the proposed modeling framework helps to make the design process of control software easier. Despite its advantage in terms of intelligibility, the occurrence of emerging behaviors makes reliable software production complicated. It then becomes necessary to check and validate the software systems modeled in this way. A series of approaches (Magee J., Kramer J., 1999), (Mikk E., et al, 1998), (Apvrille P., et al, 2001) try to introduce more formal aspects and semantics into UML (OMG, 2003) specification of systems, to allow their validation. Since Object Oriented concepts and model checking techniques have matured, it is becoming possible to establish a design approach based on model driven engineering. The approach depends on the composition and transformation of models to make checking of progress and safety properties as well as reliable implementation possible (Figure 6).

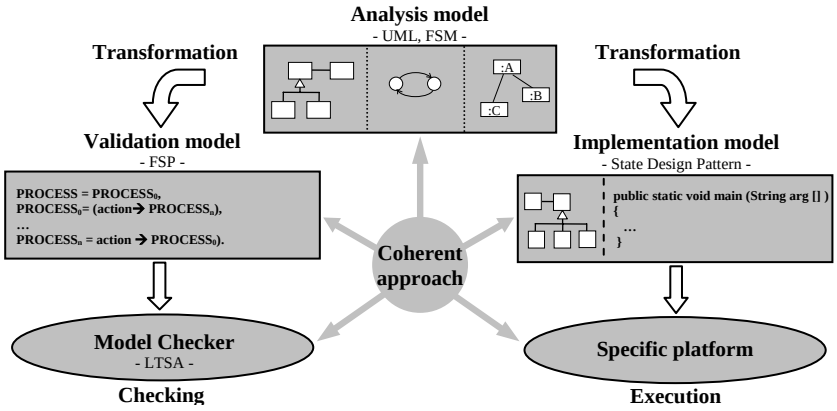


Figure 6. Conceptual representation of the approach proposed

The approach consists of five parts (Rasse A., et al, 2004):

- A structural aspect based on the behavior model (as described above).
- A model of the dynamic aspect. In UML, Statecharts are commonly used to model the reactive behavior of models. However, the organization obtained through structural modeling allows the use of simpler and precise formalism. Here, finite state machines present an appropriate notation to capture, formally the behaviors associated with each behavioral class.
- A particular configuration of the system in order to obtain the specific behaviors required.
- A validation of the behavioral model. The finite state machines are translated into a process algebra called Finite State Processes (FSP) (Magee J., Kramer J., 1999). This leads to a validation model which can be exploited with the Labeled Transition System Analyzer (LTSA) model checking tool (Magee J., Kramer J., 1999).
- Implementation which agrees with the specifications. To this aim, a translation of the validated model based on the recurring use of design patterns can be used to ensure reliable implementation.

The next sections partially describe the proposed approach. In a first time, the sections 4.1 and 4.2 complete the *analysis model* by describing the behavioral and the configuration aspects which are associated with the structure. In a second time, the section 4.3 and 4.4 describe the translation from this *analysis model* into the *validation model* and how this translation is achieved via Transformation Model techniques.

4.1 Behavioral aspects

As described above, the structural aspects have allowed to isolate and abstract - in behavioral classes - the dynamic aspects of each resource. To specify these dynamic aspects, each behavioral class is associated with a Finite State Machine (FSM). The FSMs describe the behavior of each resource in the form of event/state sequences. This choice has been motivated because this simple formalism which has formal semantics is usually used for the behavioral specification and can be easily integrated into a UML design. In accordance with the supervisory control architecture (Ramadge P.J., Wonham W., 1988), higher order behaviors (meta-behaviors) coordinate local behaviors to obtain the global behavior. This coordination is possible by a mechanism of sending/receiving events - concept of synchronization -. So, according to the events synchronized, the global behavior controls local behaviors in their state-space allowed. Figure 7 shows the discrete behaviors of the walking robot, associated with each controller - organized as shown in figure 2 - and one example of synchronization. The *global controller* is not the goal of this example, so its specifications (figure 7) will not be detailed here.

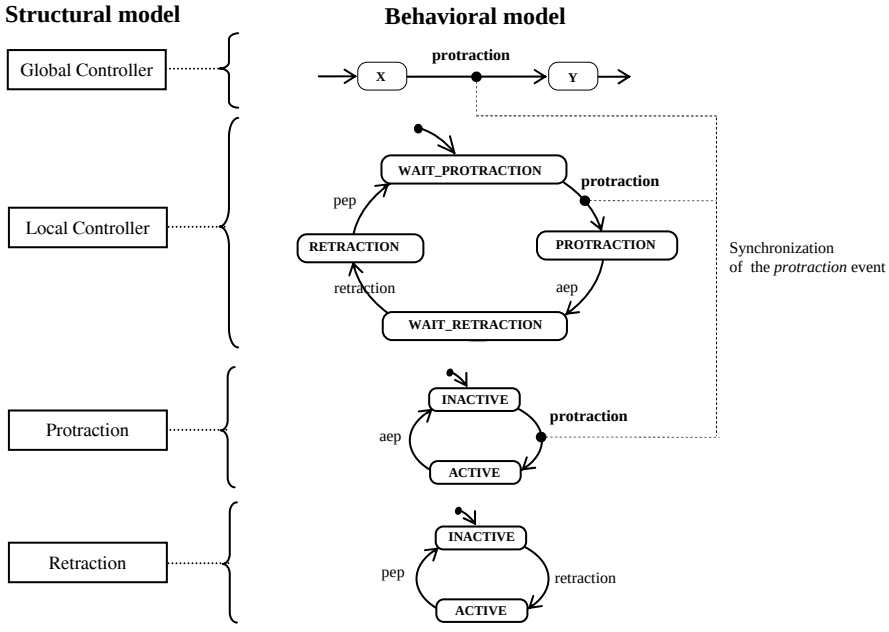


Figure 7. FSM associated with behavioral classes and hierarchy of behavior

4.2 Configuration aspects

The dynamic and structural aspects which have been described so far, propose a family of potential configurations of a system in terms of behavior, class and interaction. However, software design usually requires the use of many instances of the same abstraction. Consequently, to describe a system, it is necessary to know the topology of these instances. In UML, this information is specified with an object diagram (figure 1). This representation refines the structural aspects by specifying the links between the instances and refines the behavioral aspects by specifying the particular messages which are exchanged through these links (concept of synchronization). Thus specified, the configuration helps to understand complex systems and allows the design of a particular application whose behavior must meet the requirements.

The *analysis model* is now fully specified. However, this approach doesn't guarantee the obtaining of a suitable system. So, to make sure the system global behavior is in accordance with the requirements, the FSMs which are associated with the behavioral classes must be validated. To do so, the *analysis model* must be translated into the *validation model* to be checked by an existing model checker. As will be seen in the next sections, FSMs find an equivalence with formal languages used for the validation. Consequently, their use makes the translation easier and so, simplifies the proposed approach.

4.3 Validation

The aim of all validation tools is to make software design reliable and to ensure designers that their specifications actually correspond to the requirements (Bérard B., et al, 2001). Among the checking methods, two major categories can be distinguished: simulation and model checking. These methods are not competitive but complementary. It is sensible to associate them within UML design, so as to bring an effective answer to the numerous checking problems. However, model checking methods require the use of formal methods which provide a mathematical context for the rigorous description of some aspects of software systems. In the present approach, *the validation model* will be expressed using a process algebra notation called *Finite State Processes* or *FSP* (Magee J., Kramer J., 1999) based on the semantics of the *Labeled Transition System (LTS)*. This formalism which is commonly used in the field of checking provides a clear and non ambiguous means to describe and analyze most aspects of finite state process systems (Arnold A., 1994). It allows the use of the *LTSA* model checker (Magee J., Kramer J., 1999) in which a system is structured by a set of elementary components whose behavior is described in FSP. So, the present approach proposes to collect the behaviors specified with FSM in the *analysis model*, then to translate them into FSP. The FSM formalism is also in accordance with the LTS's semantics; consequently, as shown in table 1, they immediately find a correspondence with FSP.

Table 1. Mapping from the *analysis model* concepts to the *validation model*

Analysis model	Validation model
FSM state	local process: P=(a → P).
FSM event	Action prefix: a →
classes	processes
instances	process labeling: instance_name : type_name
configuration	parallel composition: instance_1 || instance_2
synchronization	relabeling operator: a / b

Figure 8 represents the FSP translation of the behavior of the local controller (LC) class graphically described by its Finite State Machine in figure 7

```
LC                 = WAIT_PROTRACTION,
WAIT_PROTRACTION   = ( protraction → PROTRACTION      ),
PROTRACTION        = ( aep        → WAIT_RETRACTION   ),
WAIT_RETRACTION    = ( retraction → RETRACTION        ),
RETRACTION         = ( pep        → WAIT_PROTRACTION  ).
```

Figure 8. Behavioral description of a LC component in FSP

The global behavior is obtained from all the instances of these elementary components (LC, GC, RC and PC) and all their interactions within a particular configuration (figure 1). In FSP, a process labeling (*instance: Component*) provides multiple instances of elementary components, which are in accordance with the instances of the behavioral classes of the *analysis model*. So, a set of six local controllers (lc_i), six protraction controller (pc_i), six retraction controller (rc_i) and one global controller (gc) processes are thus created. The *ROBOT* global behavior (figure 9) is expressed as a parallel composition (||) of these instances, which are executed concurrently and synchronized on their shared action using the FSP relabeling operator (/).

```
||ROBOT =  ( lc1:LC || lc2:LC || pc1:PC || ... || gc:GC ) / {gc.protraction_lc1 / lc1.protraction,.....}.
```

Figure 9. Global behavior in FSP

This *ROBOT* behavioral model is validated by the LTSA model checker. This tool allows the interactive simulation of the different possible execution scenarios of the model specified. This interactive exploration allows the designer to improve his confidence in the coherence between the expected behaviors and the models which describe them. To ensure robustness and flexibility in locomotion, this first non exhaustive type of validation can be complemented by a search for progress or safety properties violation. According to the progress rules, all the legs must continue to move, whatever the possible execution traces of the system. According to the safety rules, all the legs must not be raised at the same time. In the *validation model* proposed, great attention will be given to the progress properties which asserts that "s*omething good eventually happens*". The progress property previously stated consists in checking the occurrence of the *protraction* action for each *local controller* and their infinitely repeated execution. In LTSA, the progress properties are expressed with the *progress* key word (figure 10).

```
progress Cycle_lc2 = {lc2.protraction }.
                …
progress Cycle_lc6 = {lc6.protraction }.
```

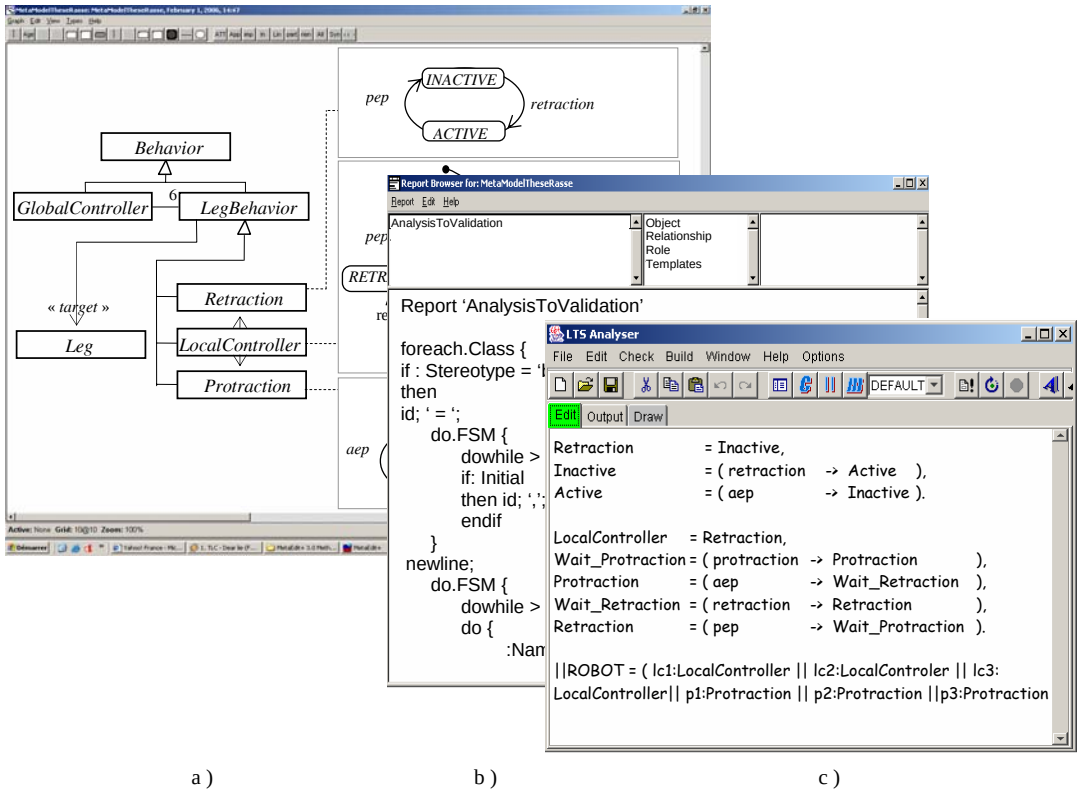
Figure 10. Progress properties in FSP

If this property is violated by the model, the analyzer produces the sequence of actions that leads to the violation. So, in accordance with the result obtained, the designer can modify his *analysis model*, until the obtaining of a suitable control software. This approach allows the coherent translation between an *analysis model* and a *validation model* and thus ensures the validation of the controller behaviors.

4.4 Model Transformation

The model transformations that were intended to produce the validation model in an automated way were prototyped using the MetaEdit tool (MetaEdit, 2005). Its use implies two modeling stages:

- The first stage, at the meta-level, consists:
  - first in describing the entities of the specification meta-model with the meta-meta-model of MetaEdit called GOPRR (Graph, Object, Property, Relation, and Role).
  - then in writing the transformation rules in the form of script (figure 11.b), rules which will have to be applied to the instances of this meta-model in order to obtain the corresponding target models.
- The second stage, at the model level, consists:
  - first in specifying the specification model (figure 11.a) graphically, in accordance with its meta model.
  - then in executing the transformation rules on the entities of this model in order to obtain the entities of the target model (FSP code) (figure 11.c).



a) b) c)

Figure 11. a) Modeling of the specification model in MetaEdit, b) Transformation rules and c) FSP code

The FSP code obtained in this way can directly be analysed with the LTSA tool to ensure that the specification model satisfies the designer's requirements.

As suggested, the aim of the present approach is to produce an executable code for the implementation of validated control software. however even if the joint use of object-oriented methods, checking tools and model transformation techniques makes software development easier and more reliable, it does not guarantee that the implementation conforms with the validation. That is why, the approach presented in this paper is part of a global software development (figure 1) in which the use of an runtime platform – also in conformity with LTS semantic – helps to reduce the semantic gap between the models and thus allows the easier generation of a code in accordance with the specification and validation models.

So, this approach allows the creation of a coherent software development cycle that integrates specification, validation and implementation phases.

5. CONCLUSION

This paper has presented a modeling framework to define control software for complex systems using the example of a controlled hexapod robot. It is based on an architecture model where objects are organized according to two conceptual levels – one for resources and one for behaviors. This proposal reduces the apparent complexity of a system by separating the nature of the entities it is composed from their dynamic. The object-oriented concepts then allow the identification and organization of the different behavior classes. Moreover, some behavioral objects can be considered as resources for behaviors of a higher order (meta-behaviors) so that they can be integrated, combined or adapted. The same notation may then be used to represent the static, dynamic and metadynamic aspects of a system. Finally, the necessity to consider a rigorous process for software design which integrates the different design phases of modeling has been highlighted. To complete the proposed modeling framework, a coherent model-based approach supported by model-checking tools that ensures the development of validated applications is considered. Moreover, an step of implementation, based on the recurring use of design patterns, completes this work and allows an implementation in accordance with the validation (Rasse A., et al, 2004).

REFERENCES

Apvrille P. et al, 2001. A new UML Profile for Real-time System Formal Design and Validation, *Proceeding of 4th International Conference on the Unified Modeling Language*, Toronto, Canada, pp. 287-301.

Arnold A., 1994. Finite Transition System, Prentice Hall, Prentice Hall.

Beaudouin-Lafon M. and Mackay W.E., 2000. Reification, polymorphism and reuse: three principles for designing visual interface, *Proceedings of the working conference on Advanced visual interfaces*, Palermo, Italy, pp. 102-109.

Bérard B. et al, 2001. *Systems and Software Verification. Model-Checking Techniques and Tools*, Springer.

Booch G., 1994. *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, Reading, MA, USA.

MetaEdit, 2005, Domain Specific Modeling with MetaEdit+, http://www.metacase.com/

Magee J. and Kramer J., 1999. *Concurrency. State Models & Java Programs*. John Wiley & Sons, Chichester, UK.

Mikk E. et al, 1998. Implementing statecharts in PROMELA/SPIN. *Proceedings of 2nd IEEE workshop on Industrial-Strength Formal Specification Techniques*, Boca Raton, FL, USA, pp. 90-101.

OMG, 2003. Unified Modeling Language Specification, Version 1.5, http://www.omg.org/docs/formal/03-03-01.pdf.

Ramadge P.J. and Wonham W., 1988. The control of Discrete Event Systems. *Proceeding of the IEEE, Vol. 77, No.1, pp. 81-98.*

Rasse A. et al, 2004. Design And Validation Of Object-Oriented Software Via Model Integration. *Proceedings of EuroSim, Special Session: Modeling and Simulation of Object based Software Systems*, Paris, France, pp. 2-7.

Sanz R. et al, 2001. *Software for Complex Controllers*. In: Karl Astrom, P. Albertos, M. Blanke, A. Isidori, W. Schaufelberger, R. Sanz, ed., Control Of Complex Systems, Springer-Verlag, London, pp. 143-164.

Thirion B. and Thiry L., 2002. Concurrent Programming for the Control of Hexapod Walking, *In ACM Ada Letters*, Vol 21, No 1 , pp. 17-28.

Thiry L., 2002. *Modèles, métamodèles et Objets Comportementaux pour les systèmes dynamiques complexes*, Ph.D. Thesis,Université de Haute Alsace, France.