# *Modeling and Simulation of Object Based Software Systems*

# DESIGN AND VALIDATION OF OBJECT-ORIENTED SOFTWARE VIA MODEL INTEGRATION

A.Rasse, J-M.Perronne, B.Thirion

MIPS Laboratory, LSI Group, UHA
ESSAIM, 12 rue des frères Lumière, 68093 Mulhouse Cedex, France
{a.rasse, jm.perronne, b.thirion}@uha.fr

## ABSTRACT

Due to their increasing complexity, software development and maintenance have become a difficult task. It is therefore necessary to consider a rigorous process for software design which integrates the different design phases in a unified manner. This is the aim of the present paper which proposes a coherent approach based on models that guarantee the development of validated applications. From *analysis models*, the approach helps to obtain both *validation models* which can be exploited with existing model checking tools and specific *implementation models* which conform to the validated models.

## KEYWORDS

Object Oriented Modeling, UML, Model Transformation, Model Checking, Design Patterns

## 1 INTRODUCTION

Modeling control software systems has become more and more difficult due to their increasing complexity. To allow the development of reliable applications, such complexity must be controlled and design accuracy must be ensured at all levels. The bottom up Object Oriented methods based on reusable entities help to understand this complexity and to make the design process easier [4]. Despite the undeniable advantage in terms of productivity and 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, since their failures may have economic, material or human consequences. A series of approaches try to introduce more formal aspects and semantics into the *Unified Modeling Language* (UML) [10] specification of systems [7,8,1] 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 planned approach depends on the composition and transformation of models to make checking and reliable implementation possible. (figure 1).

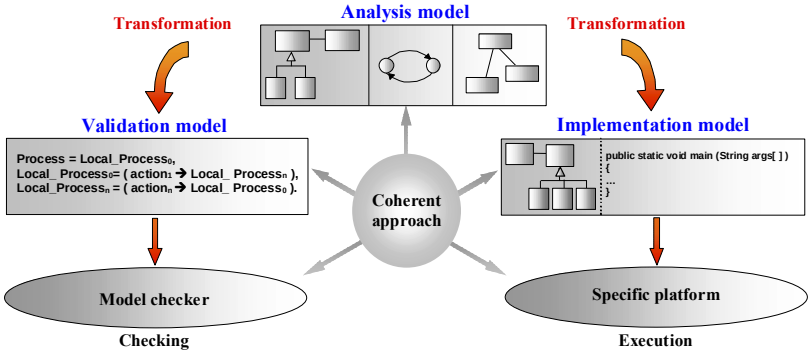



figure 1: Conceptual representation of the approach proposed

The present approach consists of five parts:

- an object oriented analysis in which the structural aspect is divided into two conceptual levels: the resource objects and the behavioral objects. This provides a simple means to identify, then isolate - in distinct classes - the entities of a domain and the behaviors applied to these entities. The abstraction and organisation of the behaviors obtained through this modeling process make them easier to understand and specify.
- a model of the dynamic aspect. In UML [10], Statecharts are commonly used to model the reactive behaviour of models. However, the organisation obtained through structural modeling allows the use of simpler, more precise formalism. Here, finite state machines present an appropriate notation to capture, formally the behaviors associated with each behavioural class.
- a particular configuration of the system in order to obtain the specific behaviors required.
- the validation of the behavioral model obtained earlier. The finite state machines are translated into process algebra called *Finite State Processes* (FSP) [7]. This leads to a validation model which can be exploited with the *Labeled Transition System Analyzer* (LTSA) model checking tool [7].
- implementation which agrees with the specifications. To this aim, this paper proposes a translation of the validated model based on the recurring use of design patterns [5] to ensure reliable implementation.

This paper is divided into four parts. The first part presents the running example of a legged robot which will be used for the present approach. The next sections describe the *analysis, validation* and *implementation models* respectively.

## 2 RUNNING EXAMPLE

The system used to illustrate the present approach is an omnidirectional hexapod robot called Bunny [11] (figure 2.a). This mobile platform requires an efficient appropriate control architecture for the integration of a number of coordinated functions. Only the locomotion function will be considered here: a leg moves in a cyclic way between two positions *aep* (anterior extreme position) and *pep* (posterior extreme position) (figure 2.b). 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 decentralised control; the local behaviors obtained with local controllers (LC) are applied to a leg (L) and a global controller (GC) coordinates the local behaviors.

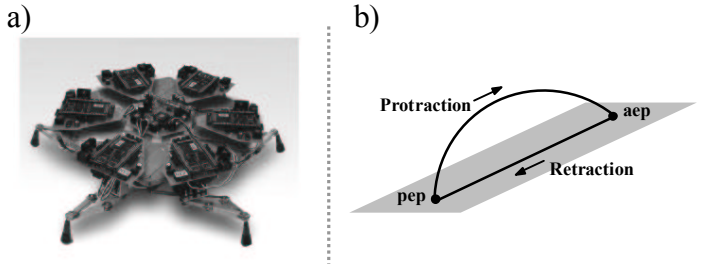



figure 2 : a) Bunny b) cycle of a leg

This system is the source of a number of problems concerning concurrence, synchronization, or decentralized control. So, to ensure robustness and flexibility in locomotion, Bunny must satisfy a set of progress and safety rules. According to the progress rules, all the legs must continue to move, whatever the possible execution traces of the system. Bunny's control software is representative of a class of software systems which must be validated to avoid any problems in their exploitation.

# 3 ANALYSIS MODEL

The *analysis model* consists in finding a robust and adapted structure that fits the system to be modelled. Based on simple, key entities, this structure must be easy to handle and organized in such a way that a complex macroscopic behavior can be created. Since it was standardized by the *Object Management Group* (OMG) [9], the UML has become a standard for the description of software systems [6]. In the present approach, *the analysis model* consists of three parts (figure 3): a structural, a behavioral and a configuration part. The structural aspect helps to organize the abstractions (classes) of the model. The behavioral one describes the behaviors associated with the structure and the configuration part establishes the links between the instances from the abstractions. The specification of these links is necessary for the description of a particular application.

## 3.1 Structural aspects

Through different levels of abstraction, the specification of the structural aspects helps to better understand the organization and the interactions within a system. The UML class diagram is the representation which is best adapted to the structural organization of the abstractions. The structural aspect of the present models is based on a two-level conceptual model [12], where so-called behavioral objects control objects considered as resources (figure 3.a). The classes describing the resources represent the elements necessary for the description of a system. The classes describing the behaviors control the resources in their state space. This separation helps to isolate and abstract the behaviors of the objects to which they apply and so, to simplify their specification. The concept can also be generalized insofar as a behavior/resource association can be considered as a new resource which is, itself, controlled by another behavior.

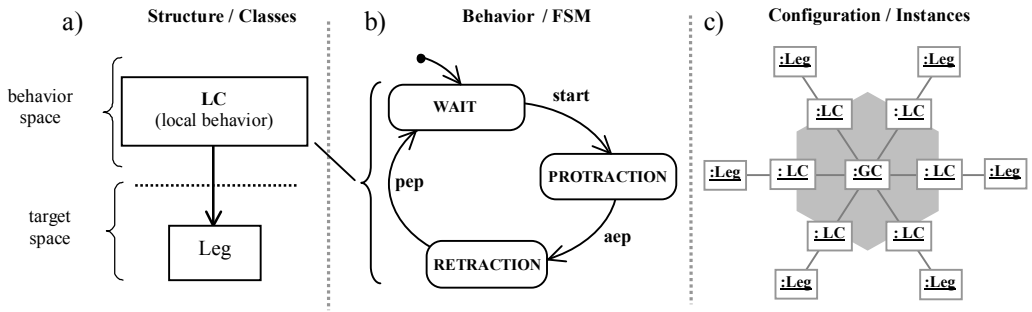



figure 3: analysis model

## 3.2 Behavioral aspects

Each behavioral class is associated with a finite state machine. These finite state machines specify the dynamic aspect of each elementary entity of the system in the form of event/action sequences. Figure 3.b shows the discrete behavior of a leg equipped with its local controller. To coordinate the legs and so keep the platform stable, the global controller supervises each local controller by allowing (or not) its walking cycle. It is the synchronization of the *start* actions and the concurrent execution of all the local controllers and of the global controller which provide the global behavior of the system. The specification of the global controller will not be detailed here. The locomotion algorithm which has been obtained must be validated to guarantee the above mentioned properties (§ 2). Only the behavioral aspects which have been specified in this way will be validated, as will be seen in the next section.

### 3.3 Configuration aspects

From the structural and behavioral aspects, the specification of a configuration helps to define a particular application. The instances from the abstractions (classes) are composed in such a way that they provide the specific behavior defined by the requirements. During this composition, the behaviors (finite state machines) are combined and synchronized. The UML object diagram in figure 3.c illustrates the object configuration used for the locomotion function.

## 4 VALIDATION MODEL

The aim of all validation tools is to make software design reliable and to assure the designers that their specifications actually correspond to the requirements [3]. 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. 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 with a process algebra notation called *Finite State Processes* (*FSP)* [7] based on the semantics of the *labeled transition system* (LTS). So, a system is structured by a set of components whose behaviors are defined as finite state machines. This formalism which is commonly used in the field of checking provides a clear and non ambiguous means to describe and analyse most aspects of finite state process systems [2,3]; it allows the use of the *LTSA* model checker [7]. The behaviors of the present *analysis model,* described in the form of finite state machines, immediately find a correspondence (figure 4.a) with the FSP notation and their translation then becomes obvious. The local behaviors of the *validation model* result from all the behavioral classes and all the associated finite state machines. Figure 4.b represents the FSP translation of the behavior of the *local controller* (LC) class graphically described by its finite state machine in figure 3.a.

| Analysis model | Validation model |
|---|---|
| configuration | parallel composition<br>*instance_1* ‖ *instance_2* |
| classes | processes |
| instances | process labeling<br>*instance_name : type_name* |
| FSM event | action<br>*a* → |
| FSM state | local process<br>*P=(a → P).* |

a) mapping from analysis model concepts to validation model

```
LC          = WAIT,
WAIT        = (  start    -> PROTRACTION ),
PROTRACTION = (  aep      -> RETRACTION  ),
RETRACTION  = (  pep      -> WAIT        ).
```

b) behavioral description of a LC in FSP

figure 4: FSP translation

The global behavior is obtained from all the instances of these elementary components and all their interactions within a particular configuration (§ 3.3). In FSP, a process labeling ($lc_i$:*LC*) provides multiple instances of elementary components, which agree with the instances of the behavioral classes of the present *analysis model*. A set of six local controllers processes ($lc_i$) is thus created, in which the labels of the actions (*start, aep, pep*) will be prefixed with the label of the particular local controller. The *ROBOT* global behavior (figure 5) is expressed as a parallel composition ( || ) of the local ($lc_i$) and global (*gc*) controllers. These are executed concurrently and synchronized on the *start* action using the FSP relabeling operator ( / ).

```
||ROBOT =  ( lc1:LC || lc2:LC || ...|| gc:GC ) / {gc.start_lc1 / lc1.start,.....}.
```

figure 5: global behavior in FSP

This *ROBOT* behavioral model will be validated by the LTSA model checker. This tool allows the interactive simulation of the different possible execution scenarios of the model specified. To do so, the user selects the different actions he wishes to control. This interactive exploration allows him to improve his confidence in the coherence between the expected behaviors and the models which describe them. This first non exhaustive type of validation, can be complemented by a search for properties violation. In the *validation model* proposed, great attention will be given to the progress properties which asserts that "*something good eventually happens*". Indeed, adapted locomotion requires above all, the recurrent motion of each leg. It must then be checked that each local controller will always be able to carry out its walking cycle. To do so, it is necessary to check the occurrence of the *start* actions for each local controller and their infinitely repeated execution. This property is called *progress Cycle_lc1 = {lc1.start }* in LTSA. If this property is violated by the model, the analyzer produces the sequence of actions that leads to the violation. The designer can then modify his model according to the results obtained.

## 5 IMPLEMENTATION MODEL

The formal specification techniques and the use of model checking tools do not prevent model mismatch during the development cycle. The *Design Patterns* [5] representing generic implementation models have been proposed to bring an explicit, proven solution to some recurring design problems. Among these design patterns, the *State pattern* gives a solution which is commonly used for the implementation of finite state machines. In the present *analysis model*, the behavior of each component is described as a finite state machine. It is therefore pertinent to associate each component with the implementation which corresponds to the *State design pattern*. Because it is similar to finite state machines, this pattern helps to obtain a code which conforms to the behaviors specified in the *analysis model* and checked by the *validation model*. The implementation of the *State design pattern* for the behavior of a leg and of its local controller is shown in figure 6.

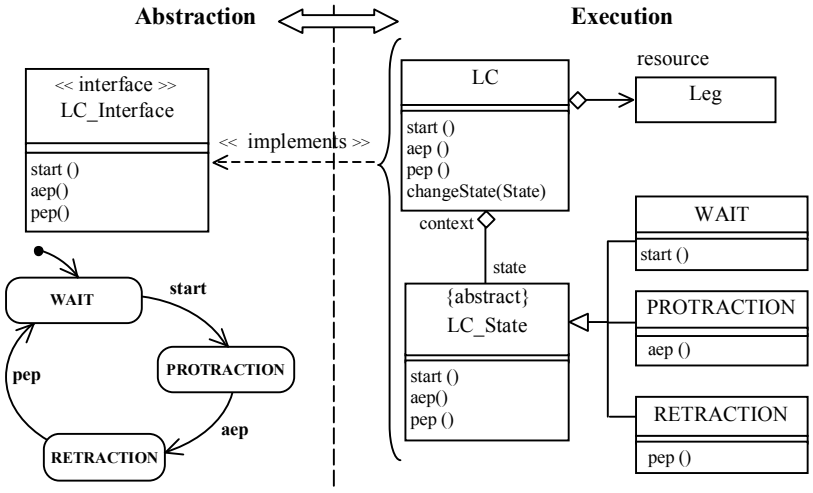



figure 6: implementation of a LC using the *State design pattern*

This implementation diagram consists of a number of elements including:

- An *LC_Interface* which defines all the possible actions of a component (alphabet of its finite state machine).
- The Local Controller class (*LC)* which exploits the abstraction of the leg as a resource by giving it a behavior described as a succession of states. It implements the *LC_interface* and lets a local object called *state* perform the specific behaviors. This local object represents the current state of the local controller and changes according to the transitions inherent in its behaviour (*aep, pep*, or *start* ).

- The *LC_State* class which implements, in an abstract way, the behavioral *LC_Interface* and represents the parent class of all the states of a local behavior. Each particular state (*Wait, Protraction, Retraction)* implements the specific behavior associated with the state of the component. Each of these subclasses only defines the actions/transitions that are associated with them and the call of the corresponding methods causes the adaptation of the state of the local controller.

In this way, the *State design pattern* provides a safe means to produce a translation of an abstraction (the *analysis model* ) into its implementation (the *implementation model*).

## 6 CONCLUSION AND PERSPECTIVES

This paper has shown the feasibility of a rational method for software design by proposing a model based approach (*analysis, validation* and *implementation models*). It depends on an object oriented architecture with two conceptual levels and formal specifications based on finite state machines. From the information (object configuration and behavior) contained in the present *analysis model,* a *validation model* is obtained which fits the specified behavior. An *implementation model* adapted to this specification is obtained using the *State design pattern*, while conforming to the validation performed previously. Each model corresponds to the semantics of finite state machines which reduces the gaps between the different models. The present approach thus allows the coherent transition between heterogeneous models, ensuring the rational integration of the different phases in software development. The current work will be followed by the implementation of transformation models aiming at a systematic, or even automatic translation, between the different models proposed. The concepts of model transformation will eventually make the development process easier and even more reliable.

## REFERENCES


[1] L.Apvrille, P. de Saqui-Sannes, C.Lohr, P.Sénac, J-P.Courtiat (2001). A new UML Profile for Real-time System Formal Design and Validation, UML'2001, Toronto, Canada.

[2] A. Arnold (1994), *Finite Transition System*, Prentice Hall, Prentice Hall.

[3] B.Bérard, M.Bidoit, A.Finkele, F.Laroussinie, A.Petit, L.Petrucci, and P.Schnoebelen (2001). *Systems and Software Verification. Model-Checking Techniques and Tools*. Springer

[4] G.Booch (1994). *Object-oriented Analysis and Design with Applications*, Second Edition, The Benjamin/Cummings Publishing Company Inc, Redwood City, California.

[5] E.Gamma, R.Helm, R.Johnson, J.Vlissides (1995). *Design Patterns – Elements of Reusable Object Oriented Software*, Addison Wesley, Reading, Massachusetts.

[6] H. Gomaa (2000). *Designing Concurrent, Distributed and Real Time Application with UML*. Addison Wesley, Reading Massachusetts.

[7] J. Magee, J. Kramer (1999). *Concurrency. State Models & Java Programs*. John Wiley & Sons, Chichester, UK.

[8] E.Mikk, Y.Lakhnech, M.Siegel, G.J.Holzmann (1998). Implementing statecharts in PROMELA/SPIN, *in proceedings of 2nd IEEE workshop on Industrial-Strength Formal Specification Techniques,* IEEE Computer Society Press, p 90-101.

[9] Object Management Group, http://www.omg.org/

[10] Object Management Group. (2003). OMG Unified Modeling Language Specification, Version 1.5, http://www.omg.org/docs/formal/03-03-01.pdf

[11] B.Thirion, L.Thiry (2002). Concurrent programming for the Control of Hexapode Walking, *ACM Ada letters*, vol. 21, N°1, pp.12-36.

[12] L.Thiry, J.M.Perronne, B.Thirion. Patterns for Behavior Modeling and Integration, *Computer in Industry*, Elserier Ed, to appear.

# ASSESSING THE MODIFIABILITY OF TWO OBJECT-ORIENTED DESIGN ALTERNATIVES – A CONTROLLED EXPERIMENT REPLICATION

Ignatios Deligiannis[1], Panagiotis Sfetsos[1], Ioannis Stamelos[2], Lefteris Angelis[2]
Alexandros Xatzigeorgiou[3], Panagiotis Katsaros[2]

[1]Dept. of Informatics, Technological Education Institute of Thessaloniki, Greece
{igndel, sfetsos@it.teithe.gr} http://sweng.csd.auth.gr/htmls/people_files/deligiannis.html
[2]Department of Informatics, Aristotle University of Thessaloniki, Greece
{stamelos, lef@csd.auth.gr}
[3]Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
{achat@uom.gr}

## ABSTRACT

This paper presents a replication study of a controlled experiment, investigating the impact of many design characteristics on one of the most desirable quality factors, *modifiability*. Two alternative design structures were used; a responsibility-driven (RD) versus a control-oriented "mainframe" (MF) design. Two groups of undergraduate students participated, each performing on one of the two designs. The subjects designed, implemented in Java, and tested a set of three maintenance tasks in order to assess the degree of their understanding, effort, and performance. The results indicate that the RD version due to its delocalised structure, exhibited higher correctness, better extensibility, and design stability, than the MF version. In order to provide an objective assessment of the differences between the two versions, a considerable number of metrics were used on the delivered solutions, quantifying separately each produced design's characteristics.

## KEYWORDS

Object-Oriented, experiment, maintainability, design, metrics.

## 1 INTRODUCTION

The necessity for change is indissolubly related with the development process, and it results from the fact that with the progress of time, additional knowledge is accumulated both from software developers, as well as from customers using the software. The Object-Oriented (OO) paradigm, by providing a number of mechanisms, such as inheritance and encapsulation, has been claimed to enhance understandability, due to increased cohesion of class structures, and manage more efficiently complexity. Hence, it is considered having particular affinity to *modifiability*, which is one of the most important quality factors by which the OO paradigm aims at reducing the maintenance costs of a system. This was also supported by a number of empirical studies [1].

Hence, a considerable number of researchers and practitioners in Software Engineering field, have turned their concerns in even more effectively applying the OO mechanisms and techniques, aiming at producing even more understandable and maintainable software, i.e. more qualitative systems. Particularly the designers, often facing the dilemma of choosing among different available design structure solutions, have focused on design quality and the various

design characteristics from which it stems. They aim mainly at some most desirable quality factors, such as understandability, performance, correctness and reusability.

The role of empirical research is to provide empirical evidence with proper grounding, whether and to what extent the above-mentioned aims and quality factors have been met. Experimental replication, that is an empirical study conducted under identical conditions as a basic experiment, is desirable in empirical investigation for several reasons: a) it can help us to know how much confidence we can place in the results of the experiment; b) it enables us to estimate the mean effect of any experimental factor [2]; c) the results of an experiment can be more easily generalised than those of a case study.

## 2 DESIGN OF THE EXPERIMENT

The research reported in this paper is a replication study of a controlled experiment carried out by Arisholm et. al. [3]. The main goal of the study is to investigate to what extent design characteristics affect the *modifiability* of OO software. Thus, to investigate how design characteristics affect modifiability, we need mainly to focus on structural characteristics, since they describe more precisely the structure of a system. In order to achieve an assessment of these characteristics, we need a set of proper metrics, since only them can provide an objective assessment.

In this study, two alternative designs of a system were used, implementing the same functionality. They were written in exactly the same manner, namely, programming style, naming conventions, and documentation. Similar skilled subjects performed a given set of change tasks. The experiment investigates the same hypotheses as the initial study, using similar settings. However, it deviates from the original experiment in three points: a) *Java* was used, instead of *Mocca* programming language; b) the subjects carried out the modification tasks realistically; implementing in code, compiling and running, separately for each requested task; c) a wide range of metrics were applied upon the produced code. This was considered particularly useful, since we believe by applying a considerable number of metrics, available in the literature, and capturing major design characteristics, a more accurate and detailed assessment of the delivered solutions could be obtained. This provides an in depth understanding of what design characteristics mostly affected the subjects' behavior and therefore modifiability.

The ***hypotheses*** tested were the following:

(H1) *Change effort*: The RD design requires more change effort than the MF design.
(H2) *Learning curve*: The RD design has a stronger learning effect (one learns more easily from previous tasks) than the MF design.
(H3) *Correctness*: The solutions for the RD design contain more errors than the MF design.
(H4) *Change complexity*: The RD design has higher change complexity than the MF design.
(H5) *Structural stability*: The RD design has better structural stability than the MF design.

The statistical test will attempt to reject the null hypothesis, which is the opposite of H1 to H5.

Forty-three students were used as participants to perform modification tasks on the two system designs, MF and RD. All were undergraduate at the second semester of their studies, at the Dept. of Information Technology of the Technological Education Institute of Thessaloniki, Greece, enrolled in the class of OO Programming. Each lecture was supplemented by a practical session using the Java programming language. They were randomly assigned to two groups, 22 performed in MF and 21 in RD design.

*Experimental material***:** The application used was a system implementing a 'Coffee Machine' system [4]. Each system documentation included: a) a description of its functionality, followed by a figure showing the logical parts of the machine; b) the Java code, consisting of 11 and 16 classes, and 286 and 391 lines of code, for the MF and RD version respectively. The two

designs were classified according to Coad and Yourdon's quality design principles [5, 6]. Thus, the MF design, not adhering to them, is considered to be the "bad" design, while the RD design, which adhered to the principles, is considered to be the "good" design.

*Experimental tasks*: The participants were asked to perform three modification tasks. Each task was coded, compiled, and tested realistically, as it happens in practice. This was actually one of the differences with the original study. The main purpose on the test was to motivate the subjects to produce solutions of good quality, before proceeding to the next task.

*Procedures*: No time-constraint was specified to complete all the tasks. However, they were told to perform as quickly and correctly as possible, hence they were requested to mark their performance time. Additionally, they were given a debriefing questionnaire to complete, expressing their subjective opinion concerning a number of issues.

*Experimental Design*: We applied statistical analysis on the collected data to interpret the results in order to draw meaningful conclusions from the experiment. The choice of the design affects the data analysis and vice versa. In this study, the type of design applied is a *randomized within-subjects design* [7]. The independent variable is the *design principles* under investigation. The experiment context is characterized as *multi-test within object study* ([7] p. 43). An advantage of using such a design is that it simplifies the statistical analysis. The participants were assigned into two groups as mentioned above.

*Dependent variables*: For hypotheses testing the following variables were examined:

- *Change Effort* – Time in minutes for each completed task.
- *Correctness* – The completed task was examined whether it worked correctly or not.
- *Learning Curve* – The normalised difference in effort to understand the last change (*c3*) versus the first change (*c1*) for each subject, for design MF and RD respectively.

$$\text{Learning Curve}(d) = \frac{\text{Understand}(d, c1) \ - \text{Understand}(d, c3)}{\text{Understand}(d, c1) + \text{Understand}(d, c3)}, d \in \{\text{RD, MF}\}$$

- *Subjective Change Complexity* – Measured from the questionnaire by two of participants' subjective answers, regarding *Solution difficulty,* and *Solution confidence.*
- *Structural Stability* – The impact changes have on the design, measured by the differences in the average values of the measures before and after each change.

## 3 EXPERIMENTAL RESULTS

To interpret the results, we consider examining the following factors: a) participants performance time; b) each task required modifications based on measurements extracted from participants' solutions. Hence, eight structural metrics, capturing structural design characteristics, were applied, provided by the Together© CASE tool.

Considering Task 1, all data shows that the two designs required similar treatment, hence both groups performed almost similarly. Both groups had to add similar trivial code in two classes, a method call, and a new method. Task2, required adding new functionality mainly based on inheritance. However, it differed between the two groups. MF design, with its centralised structure, was more demanding, mainly due to its procedural approach. RD group, by its OO structure, providing more reuse efficiency, led to better performance. Task 3, which required performing a 'check' before proceeding to 'produce' a requested drink, was more difficult for RD group to accomplish. MF group was forced to apply a procedural approach in a centralised structure, as in task2, where the most of the functionality was captured in one class by using many *if*-statements. On the other side, RD group with its decentralised structure faced a 'delocalised plan' [9] approach (OO technology's "weak point"), since the required 'check' captured a long message path between five classes.

To make a scientific statement with a reasonable degree of confidence, the significance level for the hypotheses test were set to $\alpha$=0,1.

***Change Effort (H1)***: *Hypothesis H1 is not supported.* Total effort shows no significant difference between the two groups (*t*-test, *p*=0,69) (Fig.1). However, considering each task separately, task2 showed a significant difference against MF group, while task 3 against RD group.

***Learning Curve (H2)*****:** *Hypothesis H2 is not supported.* Data shows no significant difference between the two groups (*t*-test, *p*=0,863) (Fig.2). In total, MF group required more time to understand than RD group. Particularly, task2 is considerably more difficult to understand for MF group, mainly due to procedural structure (many 'ifs') in one method. In contrast, RD group understood more easily due to reusability efficiency. Considering task 3, MF group understood more easily because they had to work with the same method as in task2 (learning effect), while RD group's functionality was split in 5 classes.

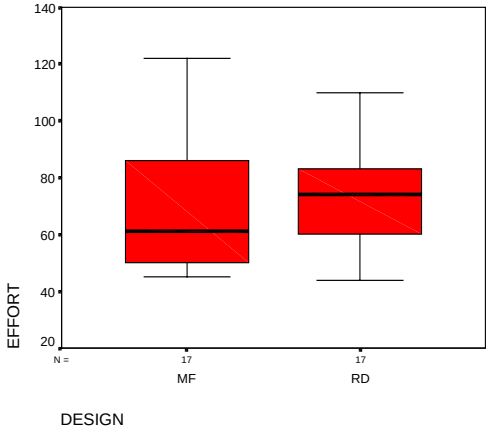



*Figure 1.* Performance effort

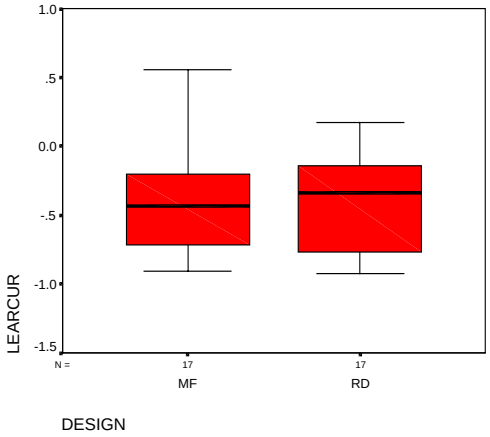



*Figure 2.* Learning curve

***Correctness (H3)*****:** *Hypothesis H3 is not supported.* RD group performed significantly more correctly than MF group (*t*-test, *p*=0,09) (Fig.3).

***Subjective Change Complexity (H4)*****:** *Hypothesis H1 is supported.* Considering *confidence,* Task1 showed no difference between both groups. In Task2 and 3, MF group participants indicated greater confidence than RD group. Considering *difficulty,* Task1 and 2 showed no considerable difference, while Task 3 showed significant difference against RD group (*t*-test, *p*=0,669) (Fig.4).

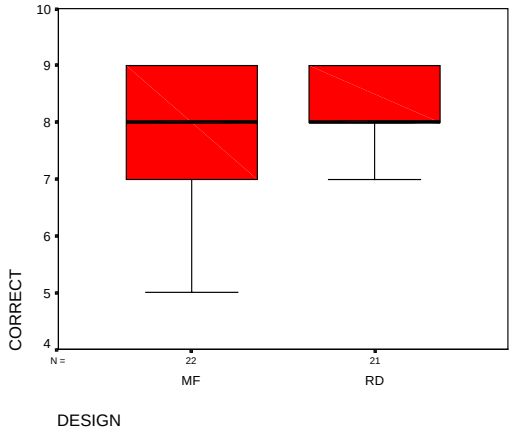



*Figure 3.* Correctness

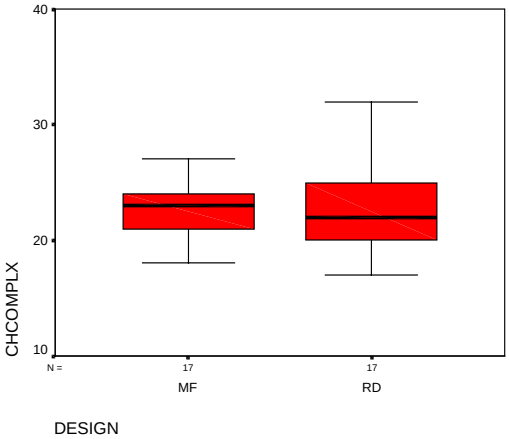



*Figure 4.* Subjective Change Complexity

***Structural Stability (H5)*:** *Hypothesis H1 is supported.* Data extracted from 8 metrics values indicated a significant difference between the two groups (Wicoxon, $p$=0,069, $t$-test, $p$=0,114) (Fig.5). The selected metrics were the following: Coupling Between Objects (CBO), Response for a Class (RFC), Weighted Method Per Class (WMPC1 &2) [10], Lines Of Code (LOC), Method Invocation Coupling (MIC) [8], No Of Attributes (NOA) [11], No Of Operations (NOO) [12]

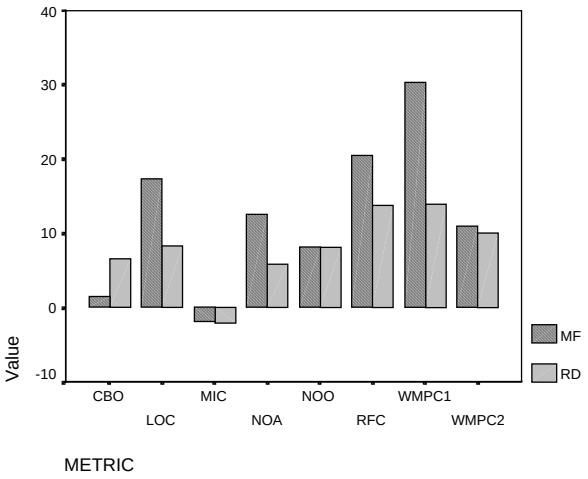



*Figure 5.* Structural Stability

.

### 3.1 Comparing the results with the original study

To compare the results between the two studies, we first have to consider the conditions under which they were conducted, since their impact determines subjects performance and therefore to results. Three were these differences, discussed later in section 4: a) programming language used, b) implementation manner, and c) metrics set applied to solutions. However, the two studies differ in hypothesis 1.

## 4 THREATS TO VALIDITY

This section discusses threats to validity and the attempt to alleviate them. The *external validity* of this study depends basically on the population selection, the choice of the design alternatives, and the choice of modification tasks. Considering the *internal validity,* this may be threatened for example by, unclear experimental materials, ambiguous questions, and skewed group assignments. An ultimate means to improve the validity of the original study, according to its authors, is by replication. Hence, we focused on those threats that were related to the differences between the original and our study. These are the following:

*Programming language, Java vs. Mocca*: In this study *Java* was used, while in the original study *Mocca* programming language was used. We consider Java currently is the most widely used programming language [13]. Also the participant students in this study had good knowledge of it. However, this could lead to differences between the two studies.

*Coding realistically vs. Pen and Paper*: The changes were coded using an advanced editor, while in the original study pen and paper were used. As the original study authors mentioned, using a computer one provides a number of advantages. We consider the main advantage is that subjects perform closely to realistic situations that happen in industry. However, there is a risk, due to students' not professional experience. Namely, one could spend a long time in a trivial task, when coding, compiling, and running a program, mainly caused by a "misunderstandable" compiler message. This practice could also lead to differences between the two studies. To alleviate this threat, an instructor was available

*Selection.* This is the effect of natural variation in human performance. Volunteer students were used, considered more motivated and suited for the task than the whole population. Hence the selected group is not representative for the whole population. Our concern was to select the most capable of the students from the course, offering them a degree bonus for their participation. An additional reason for this kind of selection was that we considered excluding those not really willing to participate, because they might not perform properly.

## 5 CONCLUSIONS

Examining the results, the following conclusions could be drawn: In general, design structure seems to play a major role in modifiability; Since, it is a creative process, guided by design principles could lead to quality solutions; a) *Extensibility*, in this study captured by task2, is more efficiently and effectively applied on a decentralised structure, better providing understanding and reusability conditions. b) 'Delocalised plan' is a potential drawback within OO paradigm. However, we believe that it could be to some extent alleviated by applying proper design techniques. As such, we consider some design patterns found in literature [14] [15], which could guide us to more effective members placement, hence yielding a more cohesive and less coupled design; c) RD design supports correct solutions.

The results give us the motivation for further research on a task solution basis. This could provide the necessary information to gain a better understanding of each task structure and implementation difficulty. Hence, we will consider first separating each task implementation, where a proper set of metrics would be applied providing metric values before and after task completion. These metrics values compared to both objective performance values (time spent), as well as subjective values (questionnaire), could help us to more precisely explain which structural design characteristic and to what extent affect subjects understanding and therefore the ability to perform efficiently and effectively.

## ACKNOWLEDGEMENTS

The authors thank the students who participated in this study, at the Dept. of Informatics at Technological Education Institute of Thessaloniki. Also, we thank the anonymous reviewers for providing useful comments.

## REFERENCES

1. Deligiannis, I., Shepperd, M.,Webster, S.,Roumeliotis, M., *A Review of Experimental Investigations into Object-Oriented Technology.* Empirical Software Engineering Journal, 2002. 7(3): p. 193-231.
2. Fenton, N. and S.L. Pfleeger, *Software Metrics, A rigorous & Practical Approach, Second Edition.* International Thompson Computer Press, 1997.
3. Arisholm, E., D. Sjoberg, and M. Jorgensen, *Assessing the Changeability of two Object-Oriented Design Alternatives - a Controlled Experiment.* Empirical Softw. Engineering, 2001. 6: p. 231-277.
4. Cockburn, A., *The Coffee Machine Design Problem: Part 1 & 2.* C/C++ User's Journal, 1998(May/June).
5. Coad, P. and E. Yourdon, *Object-Oriented Design*. first ed. ed. 1991: Prentice-Hall.
6. Coad, P. and E. Yourdon, *Object-Oriented Analysis*. second ed. 1991: Prentice Hall.
7. Wohlin, C., Runeson, P., Host, M., Ohlsson, M., Regnell, B., Wesslen, A., *Experimentation in Software Engineering - An introduction*. 2000: Kluwer Academic Publishers.
8. Marinescu, I., R., *An Object Oriented Metrics Suite on Coupling. Universitatea*. 1998.
9. Wilde, N., Mathews, P., and Ross, H., *Maintaining Object-Oriented Software.* IEEE Software, 1993 (Jan): p. 75-80.
10. Chidamber, S. and C. Kemerer, *A Metrics Suite for Object Oriented Design.* IEEE Trans. Softw. Eng., 1994. 20(6): p. 476-493.
11. Henderson-Sellers, B., *Modularization and McCabe's Cyclomatic Complexity.* Communications of the ACM, 1992. 35(12): p. 17-19.
12. Chen, J.-Y. and J.-F. Lu, *A new metric for object-oriented design. Information and Software Technology*, 1993. 35(April): p. 232 - 240.
13. Goodley, S., *Java on course to dominate by 2002*. 1999, http://www.vnunet.com/News/87054.
14. Gamma, E., Help, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*. 1995: Addison -Wesley.
15. Larman, C., *Applying UML and Design Pattrens*. 2002: Prentice Hall PTR.

# Transparent Modelling Of Objects' Evolution*

Dimitrios Theotokis[1], Anya Sotiropoulou[1], Georgios Gyftodimos[2]

[1]Department of Computer Science and Technology, School of Science and Technology, University of Peloponnese, GR 22100 Tripolis, Grecce, Fax +30 2710 372160
{dtheo,anya}@uop.gr
[2]Department of Informatics and Telecommunications, National and Kapodistrian University of Athens, Panepistimiopolis, Ilissia, GR 15784 Athens, Greece, Fax +30 210 7275214
geogyf@di.uoa.gr

## ABSTRACT

Supporting behavioural changes and in particular unanticipated ones is essential for achieving the behavioural evolution of objects. Object evolution is an important feature for war game simulation systems. This is the case because a unit's behavioural landscape may be altered during the course of a simulation either by virtue of the simulation itself of by user intervention. For instance, an armoured vehicle unit may become a scout and later an artillery observation unit. In this article we present how objects' evolution can be modelled and realised transparently using a role-based system as part of an High Level Architecture/Run-Time Infrastructure (HLA/RTI) federation.

## KEYWORDS

Behavioural changes, Behavioural landscape, High Level Architecture, Object evolution, Role modelling

## 1 INTRODUCTION

Although, the object-oriented paradigm better models the real world, it is widely accepted that it does not provide the adequate infrastructure for modelling the evolution of real world objects [5,7,8,14]. The static nature of the inheritance hierarchy compounded by issues related to the common ancestors dilemma, code scattering and tangling as well as the invariability of the self-reference are the reasons for which the evolution of objects and classes alike is inhibited [12]. Viewing information systems and in particular war game simulation systems as "living systems" [10] one can identify that object evolution is an essential feature such systems should support. Object evolution becomes important - if not essential - in military simulation systems and in particular battlefield simulation. This is attributed to the need for the accommodation of change, which in many occasions is unanticipated. In what follows we address the issue of objects' behavioural evolution in terms of roles, which are considered as behavioural adjustments or modifications of objects' initial behavioural specification. A framework that supports role-based object evolution, namely ATOMA [13,14], is utilised on top of an HLA/RTI [1] federation in order to propagate behavioural changes throughout a federation.

The remaining of this article is organised as follows: The benefits obtained from the use of roles as a means of accommodating behavioural changes is presented in section 2. In section 3 a description of the ATOMA framework and run-time system is given focusing on the modelling and implementation of behavioural changes in terms of roles. A brief comparison with other role-based approaches is also given. In section 4 the HLA/RTI architecture is presented in brief. Next, in section 5, the proposed approach is presented providing a description of how behavioural changes are accommodated throughout an HLA/RTI federation. Finally, section 6 concludes the paper and addresses future research directions on behavioural evolution, within an HLA/RTI federation.

*This research was supported in part by Pythagoras program (MIS 89198) co-funded by the Greek Goverment (25%) and the European Union (75%).

## 2 MODELLING BEHAVIOURAL CHANGES

According to Sowa [9] conceptual structures are natural types, "that relate to the essence of entities" and role types "that depend on an accidental relationship to some other entity". For instance, in a battlefield simulation an armoured vehicle is a natural entity. Viewed as a scout unit or as an artillery observation and control centre can depend on the needs of the simulation and the availability of resources. As such, a scout unit or an artillery observation and control centre are roles that can be played by an armoured vehicle. Roles may be played when the accidental relationship becomes effective, as many times as it becomes so, and more importantly they can be played in association with each other. In other words, an armoured vehicle may be at the same time a scout and an artillery observation and control unit. In fact, a natural entity may play as many roles as are applicable in a given context. For example, if the need occurs the same armoured vehicle may also become an infantry transport unit. Role acquisition and abolishment may be the result of human intervention or the result of changes in the context that the entity is found. Such changes may occur either because the internal state of the object changes, for instance the hill-climbing ability of a particular unit may be reduced when the unit receives an enemy fire hit, or because an object is viewed by other objects from another perspective, e.g. a unit becomes a scout unit.

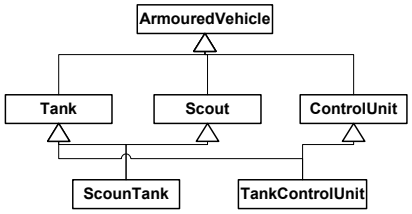



**Figure 1: An example**

In traditional object-oriented implementations and models all different behavioural scenarios must be explicitly modelled in terms of "kind-of" and "part-of" hierarchies or graphs. For instance, given the class hierarchy illustrated in Figure 1, if a tank unit is required to behave as a scout unit or as an artillery control unit, the corresponding subclasses must be *a priori* available. This imposes several problems:

1. All possible combinations of behavioural landscapes must be available during the design phase.
2. The resulting design introduces entities that do not really exist in the real world *per se*, but are only present to assist in resolving possible functional requirements
3. Class explosion results as the different behavioural landscapes played by natural entities (real-world entities) increases.
4. The resulting classes contain extrinsic behavioural characteristics thus introducing both code scattering and tangling.
5. There is no provision to accommodate unanticipated behavioural changes, except for redesigning the entire hierarchy to provide the necessary amendments.
6. For each particular behavioural requirement an instance must be created resulting in object schizophrenia, since the same unit is represented by two or more instances of different classes.
7. Deletion related problems: two or more instances of the same real world entity may co-exist, for instance a particular tank may exist on its own accord, but may also exist to model a scout vehicle. Apart from violating the underlying semantics related to instance existence, instance deletion (destruction) becomes inconsistent. When removing the instance of the tank-scout vehicle the tank part of it still remains as an instance of the tank class part. Moreover, if the instance of the tank is removed, due say to its destruction by enemy fire, the tank part of the tank-scout instance still remains.

The issue that causes the aforementioned problems is related to the lack of concern separation. It becomes evident that acquirable behaviour is time constraint and does not identify an object by virtue. It is only a means to an end, not the end itself as it accommodates for a temporary need. It is thus, and according to Sowa [5], a role type. Roles exist independently of the natural types (classes) and augment their behaviour when the need occurs and only then.

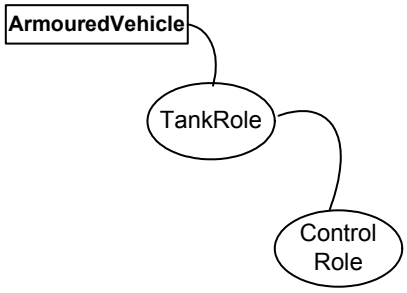



**Figure 2: Behavioural evolution**

Figure 2 illustrates the behavioural evolution of the problem area shown in Figure 1, in terms of roles. Class ArmouredVehicle represents the generic class for all armoured vehicles, whether tanks, scout units or artillery control units. The notions of a tank and artillery control unit are modelled as roles, which are assigned to instances of the class ArmouredVehicle, according to the requirements that exists at a given time.

To this extent the notion of roles employed here does not differ from those proposed by Gottlob et al. [4], Steinmann [10], Albano et al. [2], Bachman [3], Kristensen [6], amongst others. However, the realisation of roles and their underlying semantics present a number of issues addressed in the following section.

## 3 ROLES IN THE ATOMA FRAMEWORK AND RUN TIME SYSTEM

Under the ATOMA framework and run-time system classes and roles are two separate first class objects. Classes can exist on their own, while role instances exist in conjunction with the class instances they behaviourally modify. Role hierarchies are not supported by the framework at least at the definition layer. Role specialization is achieved by considering a role as a behavioural modification of an existing role. Key to the framework and run-time system is a three-layered architecture. The description layer provides the definitions and descriptions of classes and roles. The composition layer employs the mechanisms that enable the behavioural modification of classes in terms of roles, while the provision layer contains instances of both classes and roles.

What follows is a list of the key characteristics of roles and role playing under the ATOMA framework and run-time system:

1. A role comes with its own attributes and behavioural definition.
2. Roles depend on relationships with classes or class instances.
3. An object may play different roles at the same time.
4. An object may play the same role several times.
5. A role may be assigned to or removed from an object dynamically.
6. The removal of a role implies that all roles that may have been added to this role are also removed.

From an implementation perspective the behaviour of a role is not weaved into the object it behaviourally modifies. Instead dynamic proxies are constructed in order to achieve the behavioural modifications of an object by a role.

Role addition and activation are two separate procedures. A role may be added to an object but may remain inactive until it is explicitly activated. Role activation can occur either because an event triggering the role occurs, or because a condition holds or due to a combination of both. In this light role handling in the ATOMA framework differs from those in [4, 10].

## 4. HLA/RTI

The High Level Architecture provides an architecture for modeling and simulation. The intent of this architecture is to foster the interoperation of simulations and the re-use of simulation components. HLA is defined by three concepts:

- The object model templates
- The Runtime Infrastructure (RTI)
- The HLA compliance rules

The RTI and compliance rules are unchanged across all HLA-compliant simulations. However, each group of interacting simulations, or federation, must define a basis for the exchange of data and event between simulations.

## 4.1 Object Model Templates

The format and content of this basis is defined by the Object Model Templates (OMT). The definition of the Federation Object Model (FOM) is one part of the process of creating an HLA compliant federation. The Object Model Templates are used to describe the objects that will exist in the federation, the object attributes (the data that describe the state of the object), and the interaction that may occur between the objects in the federation.

## 4.2 The RTI

The RTI is a collection of software that provides commonly required services to simulation systems. These services fall into five categories:

- Federation Management
- Declaration Management
- Object Management
- Ownership Management
- Time Management

For the interested reader a more detailed description of HLA/RTI can be found in [1].

### 4.2.1 Definition of RTI federation

An HLA/RTI federation is the combination of a particular FOM, a particular set of federates and, and the RTI services. A federation is designed for a specific purpose using a commonly understood federation object model and a set of federates that may associate their individual semantics with that object model [1].

# 5. REALISING BEHAVIOURAL CHANGES IN A FEDERATE BASED SIMULATION

In order to accommodate for the behavioural evolution of objects participating in a distributed federate based simulation we have employed a role-based system, namely ATOMA [14], as a intermediate layer between each federate and its HLA/RTI component. Each federate's behavioural landscape is controlled by the role-based system. Role addition and removal occurs at this layer and is seemingly propagated to each federate via the HLA/RTI component. Transportation across the federation is achieved, as streams of byte-codes, which are resolved by each federate's ATOMA framework component. Figure 3 depicts schematically the architecture. Upon receipt of a behavioural modification the HLA/RTI component forwards the modification to the ATOMA layer, which is responsible for applying the behavioural change to the corresponding object in the federate leyer, for which it holds a representation.

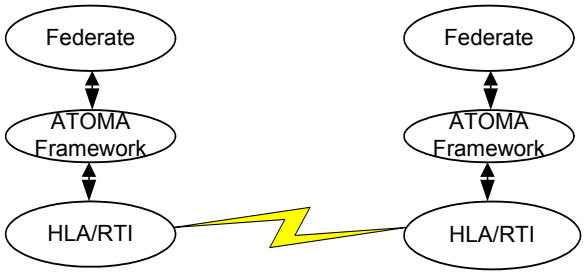



**Figure 3. System Architecture**

For example assume that within federate A an instance of a tank is assigned the role of a scout unit according to the semantics governing the Atoma run-time system described in section 3. This change is propagated to all the federates participating in the federation. Upon receipt of the appropriate message/event each federate's HLA/RTI component forwards this change to the federate's ATOMA layer, which applies the behavioural change to the appropriate object.

In order to achieve this functionality the Federation Object Mode (FOM) provides the necessary description for a behavioural modification. This description consists of an object whose attributes contain the following information:

1. The object that will be behaviourally modified. This is actually the object cross-federation unique identity.
2. The actual code that realises the behavioural modification. This is an array of bytes that is resolved to a role by the ATOMA layer, which is also responsible for applying the change to the corresponding object in each federate.
3. An indication whether the role should be activated immediately or not.
4. The condition(s) and event that will activate the role of a later time. This information is optional.

In terms of the FOM definition as depicted by a federation FED tile this information is structured as follows:

```
(class BehaviouralChange
      (attribute Recipient)
      (attribute RoleName)
      (attribute Code)
      (attribute Activation)
      (attribute ActivationRole)
)
```

When a role is to be activated at a later time this is achieved in terms of a message transmitted via the interactive mechanism supported by the RTI infrastructure. The message has the following structure:

```
(class RoleActivation reliable
      (parameter RoleName)
      (parameter Recipient)
)
```

## 6. CONCLUSIONS

The use of role-based systems for modelling and realizing the evolution of objects' behavioural landscape provides a natural and intuitive approach for realising "living" information systems. Role carry the semantics of a changing world and in doing so aid in the modelling of the system that abide to change, particularly when considering the time dimension. Roles under the ATOMA framework differ from other proposed role-based systems in two ways: The first one, although an implementation related one, bares some important semantic issues. An object does not directly relate to the roles it plugs at a given time. It only relates to a proxy for all its roles. Consequently, it is only responsible for delegating behaviour related requests, that it cannot handle, to its proxy, which in turn directs them to the corresponding roles. This provides a multi-role invocation scheme a characteristic present in real world "living systems". The second on concerns the activation semantics for roles. Existing role systems handle role activation at the same time of role acquisition and role deactivation at the time of role removal. Under the ATOMA run-time system these two notions are separate, thus providing a degree of flexibility missing in most role-based systems. Coupled with HLA/RTI such a system supports the behavioural evolution of a federate-based military simulation system, thus providing a more realistic representation of the simulated objects.

## REFERENCES


[1] DMSO https://sdc.dmso.mil/

[2]A.Albano, R.Bergamini, G.Ghelli, & R.Orsini, An object data model with roles, in: R.Agrawal, S.Baker, D.Bell (Eds) Proceedings of the 19th International Conference on VLDB, Morgan Kaufmann, Dublin; 1993 pp.39-51

[3]C.W.Bachman & M.Daya, The role concept in data models, in Proceedings of the 3rd International Conference on VLDB, 1977;pp.464-476
[4]G.Gottlob, M.Schrefl & B.Röck, Extending object-oriented systems with roles, ACM TOIS 14(3) (1996) pp. 268-296
[5] G.Kiczales. E.Hilsdale, J.Hugunin, M.Kersten, J.Palm &W.Griswold. (2001) An overview of AspectJ. In Proc. Of 15th. ECOOP, LNCS 2072, p. 327-353, Springer-Verlag,
[6] B.B.Kristensen, Object-oriented modelling with roles in J.Murphy, B.Stone (Eds) Proceedings of OOIS 95, 18-20 December 1995, Dublin, Springer 1996, pp.57-71
[7] M.Mezini (1998) Variational Object-Oriented Programming Beyond Classes and Inheritance, Kluwer Academic Publishers
[8] H.Ossher & P.Tarr. (2000) Multi-Dimensional Separation of Concerns and the Hyperspace Approach. In Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. Kluwer.
[9] J.F.Sowa. (1984) Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, New York
[10]D.Stamoulis, D.Theotokis, D.Martakos, & G.Gyftodimos. Ateleological Development of Design Decisions Independent Information Systems. In Adaptive Evolutionary Information Systems, Editor: Nandish Patel, IDEA Publishing Group, ISBN 1-59140-034-1.
[11]F.Steinmann On the representation of roles in object-oriented and conceptual modelling, Data & Knowledge Engineering 35(2000) 83-106
[12]A.Taivalsaari (1996). On the notion of Inheritance, ACM Computing Surveys, Vol 28 Number 3, pp 438-479
[13]D.Theotokis. G.D.Kapos, C.Vassilakis, A.Sotiropoulou & G.Gyftodimos, Distributed Information Systems Tailorability: A Component Approach in Proceedings of the IEEE Workshop on Future Trends on Distributed Computing, Cape Town, 1999, pp. 95-101
[14]D.Theotokis. (2003) Approaching Tailorability in Object-Oriented Information Systems through Behavioural Evolution and Behavioural Landscape Adaptability to appear in the Journal of Applied Systems Studies special issue on “Living Evolutionary and Tailorable Information Systems: Development Issues and Advanced Applications

# SIMULATION METAMODELING FOR THE DESIGN OF RELIABLE OBJECT BASED SYSTEMS

Panagiotis Katsaros Lefteris Angelis Constantine Lazos

Department of Informatics
Aristotle University of Thessaloniki
54124 Thessaloniki, Greece
{katsaros, lef, clazos}@csd.auth.gr
http://delab.csd.auth.gr/~katsaros/index.html

## ABSTRACT

Replication is a suitable approach for the provision of fault tolerance and load balancing in distributed systems. Object replication takes place on the basis of well-designed interaction protocols that preserve object state consistency in an application transparent manner. The published analytic performance models may only be applied in single-server process replication schemes and are not suitable for schemes composed of miscellaneous policies, such as those arising in object based systems. In this work we make use of a simulation metamodeling approach that allows the comparative evaluation of composite fault tolerance schemes, on the basis of small size uniform experimental designs. Our approach opens the possibility to take into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading). We provide results in terms of a case system study that reveals a dependence of the optimal adjustments on the system load level. This finding suggests the device of dynamically adjusted fault tolerance schemes.

## KEYWORDS

object replication, fault tolerance, simulation metamodeling

## 1 INTRODUCTION

Distributed object applications and services are composed of a number of objects with instances that interact to accomplish common goals. When designing and deploying such an application there are many decisions to be made with regard to the composition of objects into processes, the distribution of processes across nodes, the threading policies for the processes and the appropriate software replication for the provision of the required levels of fault tolerance and/or load balancing. All these decisions affect the resulted quality of service (QoS).

In this paper we provide a quantitative technique for the selection of a composite replication scheme that assures the required fault tolerance effectiveness. A scheme's effectiveness is determined by the response times of the fault-affected service requests, which often have to conform to a specified service quality level. The proposed simulation-based evaluation takes place on the basis of a simulation metamodeling approach that allows us to take into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading). Metamodeling is used for the prediction of system performance and its optimum operating conditions.

In the design of fault tolerance, different candidate policies may be considered and their optimum operating conditions is the unique criterion making feasible their comparison. The reason lies in the mechanisms used in replication protocols for minimizing loss of computation in the presence of faults.

One such mechanism well known as checkpointing saves the objects' states from time to time on stable storage. Another mechanism is the object state transfer (synchronization) of a live or recovered object to the states of the other replicas. In both cases the participating objects have to be operationally quiescent, i.e. not to be in-between an invocation service or to be blocked, because of a synchronous invocation of another object. In the course of a state transfer, the received invocations cannot be processed, before the end of it. Thus, placement of checkpoints and state transfers has a major impact on the perceived quality of service: excessive checkpointing (state transfers) result in performance degradation, while deficient checkpointing (state transfers) incurs expensive recovery. Moreover, in schemes composed of multiple policies for the constituent objects, checkpointing performance also depends on the structural dependencies imposed by the objects' invocation flows.

In [6] we used the term "tightest effective checkpoint intervals" for the checkpoint (state transfer) placement that yields a scheme's optimum effectiveness. We also provided a heuristic decision-making procedure that converges to the forenamed checkpoint placement. This procedure takes advantage of an initially unknown number of simulation runs that depend on the number of interacting objects. Its efficiency is bound to the observed effectiveness variations, because it trades the gains of a checkpoint interval reduction, against the overhead imposed to the vast majority of service requests (which are not affected by the occurred faults). This trade-off is suitable for the choice of a minimal cost checkpoint placement, in respect to a specified service quality level. However, it is not an efficient way for determining the required tightest effective intervals.

In the present paper this problem is bypassed by the use of small size uniform experimental designs with an a priori known number of runs.

In related work we can refer only to the work published in [8], where the authors propose a hybrid mathematical programming and analytic evaluation algorithm for a different trade-off problem: to determine process replication or threading levels, such as to avoid unnecessary queuing delays for callers or unnecessary high consumption of memory. Their approach while more efficient compared to simulation metamodeling, does not allow taking into account different design concerns in a combined manner (e.g. fault tolerance combined with load balancing and multithreading).

On the other side, the most recent work regarding the estimation of the optimal checkpoint interval is the one published in [2]. This work concerns with minimizing the program execution time and maximizing the effectiveness in a single node in a mobile environment with handoffs. The whole approach fits to a different computational environment and as other analytic models is also not suitable for the evaluation of schemes composed of miscellaneous policies.

Section 2 constitutes a short introduction to object-based fault tolerance. Section 3 outlines the core evaluation approach. Section 4 introduces the used case system study and summarizes the obtained results. Finally, the paper concludes with a discussion on future research prospects.

## 2 FAULT TOLERANCE IN OBJECT BASED SYSTEMS

Fault tolerance for object-based systems has been recently standardized [10] in a plain specification (OMG FT-CORBA) of robust support for applications that require a high level of reliability. To render an object fault tolerant, several replicas of the object are created and managed as a single object group. The OMG FT-CORBA standard allows the definition of appropriate fault tolerance properties, for each constituent object group. The supported strategies include request retry, redirection to an alternative server, passive (primary/backup) replication and active replication.

The provided support [9, 4, 11] offers protection against object faults that do not recur after recovery. Some of them may be hardware dependent (e.g. insufficient memory) and others may be attributed to media failures, power outages, human lapses, catastrophic events, the use of local timers, the use of multithreading etc. The faults conform to the fail-stop model, which means that objects fail by crashing, without emission of spurious messages. There are not any assumptions about the network topology or the protocols making up the interprocess communication service, except that communication is accomplished through loss less FIFO channels. Network partitioning and commission faults are not addressed.

In active replication all the group replicas execute each request independently, but in the same order. Checkpointing on a regular basis is not required. In passive replication only one object replica - the primary - executes the methods invoked on the group. Checkpointing takes place on a regular basis. In the presence of a fault, a backup replica is promoted to be the new primary. The state of the new primary is restored to the last checkpointed state of the old primary and the requests logged since the last saved checkpoint are then reapplied.

A fault tolerance scheme for an object-based system may be composed of miscellaneous policies. The term miscellaneous refers to the replication styles for the objects that comprise the system, with possibly different fault tolerance properties (e.g. different checkpoint placement). The simulator described in [5] allows us to realistically model the interaction effects regarding:

- the simultaneous resource possession, caused by the synchronous, often nested object invocations, which block the callers, until they get a reply,
- the hardware resource contention, as a result of the chosen replica placement,
- the load and the blocking costs caused by the recurrent checkpointing activities and the state transfers between replicas of the same group and
- the load, caused by a replica restart (repair) or re-invocation of the logged requests, according to the OMG FT-CORBA specification [10].

## 3 THE CORE EVALUATION APPROACH

The core evaluation approach allows the comparison of (combined) fault tolerance (and load balancing) schemes, with at least one recurrent checkpointing activity. A scheme's effectiveness is measured by the mean of the fault-affected requests' response times. We consider a service request to be affected by the occurred faults if

- its dispatch to the assigned service object is delayed or
- its dispatch causes the generation of synchronous invocations that are queued somewhere in the system,

as a result of at least one detected object fault. Regarding actively replicated objects a synchronous invocation is considered to be affected by an object fault, if it is delayed because of blocking to enforce operational quiescence and/or a state transfer for an object replica recovery.

More frequent checkpoints are considered to be effective, when they result in a reduction of the fault-affected requests' response times. If there is no chance of further improvement for all possible interval reductions in a vector of $n$ intervals - where $n$ the number of objects with a recurrent checkpointing activity -, this vector specifies the tightest effective checkpoint intervals.

For a (combined) fault tolerance (and load balancing) scheme, the tightest effective intervals determine the minimum response times that the scheme may achieve for the fault-affected service requests. We say that this vector characterizes the scheme's optimum effectiveness for the applied object fault model. These checkpoint intervals constitute the unique criterion that makes feasible the comparison with other schemes composed of possibly different replication policies, checkpoint placement mechanisms and/or load balancing strategies.

The tightest effective intervals were found by metamodeling [7] based on a small size uniform experimental design with an a priori known number of runs. The uniform experimental designs [3] have been suggested for computer and industrial experiments specifically for cases where the underlying relationships are unknown. They are space-filling designs with experimental points scattered uniformly on the domain. They can explore complicated nonlinear relationships between the response variable and the factors with a reasonable number of runs and have been proved robust to the underlying model specifications. A great number of them has been tabulated in [12], where we also selected the design used in our work.

The checkpoint intervals that were found to be significant were set to the values minimizing the fault-affected response times. For the non-significant ones or when multiple minima were detected, the values minimizing the fault-unaffected response times (lowest cost fault tolerance) were selected.

## 4 A CASE SYSTEM STUDY

The system model used (Figure 1) in the case study is comprised of four (4) interacting state owning objects (`obj1`, `obj2`, `obj3`, `obj4`) and four (4) stateless service objects (instances of the class `SrvRequestAccepting`). Received class-1 and class-2 requests are assigned to the available service objects in a random probabilistic fashion with equal probabilities.

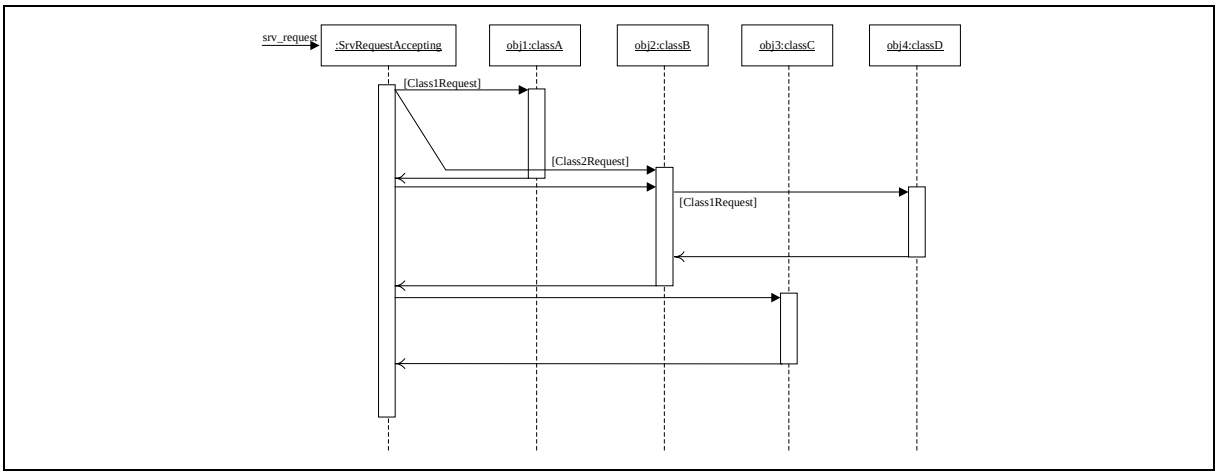
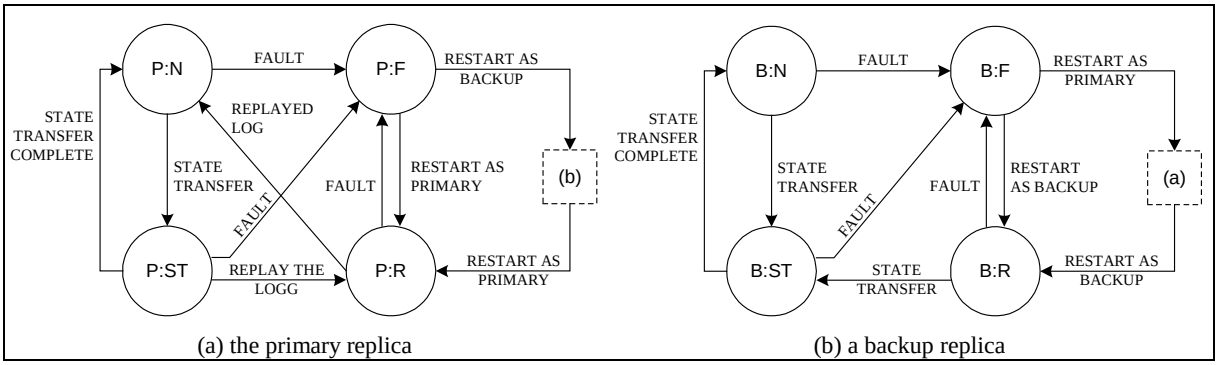



Figure 1. Message sequence for the objects of the case system




(a) the primary replica (b) a backup replica

Figure 2. Passive replication with one primary and one backup object per group

Objects are replicated according to the passive policy of Figure 2, with one primary and one backup per object group. We consider the concurrent execution of the two replicas that alternate in the primary and the backup roles, as shown respectively in the state transitions of (a) and (b). State transfers (for the state owning objects) correspond to backup state updates for a live backup and always result in a persistent checkpoint. They take place on a regular basis according to the specified checkpoint placement. Thus the applied composite schemes result in checkpoint placements specified as quadruples. Generally, in any composite scheme, only the constituent policies that are based on regular checkpointing contribute to the number of experiment's factors. This justifies the applicability of the proposed tightest effective intervals based evaluation in real size applications and services.

State transfer transitions (P:N→P:ST, B:R→B:ST) are also used as a state update of a recovered backup to make feasible a potential replacement of the primary in case of fault. When a failed primary is detected, the corresponding backup replaces it and a replica restart is then scheduled to occur (P:F→(b)) for the new backup. A recovering primary (P:R) either restarts or executes the invoked requests having logged since the last checkpoint. A recovering backup (B:R) either restarts or waits for a state transfer (B:R→B:ST), to become operational (B:N). Each replica restart restores it to the last checkpointed object state. According to the OMG FT-CORBA specification [10], re-invoked requests are detected and the same operations are not performed more than once.

The state transitions of Figure 2 are used in two distinct fault tolerance schemes:

- one with load-dependent checkpoint intervals (LDSC) and
- another one with periodic checkpoint intervals (PSC),

for the objects, which own state. The first scheme assumes a specified number of serviced invocations between checkpoints and the second one results in a fixed time interval between them. Resource consumption for the performed state transfers depends on the object state sizes and the required CPU time (state transfer speeds in sec/KB).

The four (4) service objects (`obj0, obj5, obj6, obj7`) are placed to the available nodes as specified in Table 1. Computational resource consumption is exponentially distributed both for the invoked requests and for the performed state transfers. Collocated object replicas are processed in a processor sharing discipline and each of them is placed on a separate object server. Finally, Table 1 summarizes the used parametric object fault model.

Table 1. System model parameters

| | | | | |
|---|---|---|---|---|
| service objects: | objX:SrvRequestAccepting (X=0, 5, 6, 7) | | | |
| load balancing: | random probabilistic with equal probabilities of request assignment to service object X | | | |
| class 1 request arrivals: | exponential with rates (sec) | 2.6 | 2.4 | 2.2 |
| class 2 request arrivals: | exponential with rates (sec) | 2.6 | 2.4 | 2.2 |

| | obj1:classA | obj2:classB | obj3:classC | obj4:classD |
|---|---|---|---|---|
| object state size (KB): | 0.9 | 1.1 | 0.8 | 0.6 |

| object replicas: | rep10 obj1 | rep11 obj1 | rep20 obj2 | rep21 obj2 | rep30 obj3 | rep31 obj3 | rep40 obj4 | rep41 obj4 | repX0 objX | repX1 objX |
|---|---|---|---|---|---|---|---|---|---|---|
| class 1 service (exp.) | 0.52 | 0.57 | 0.6 | 0.6 | 0.83 | 0.83 | 0.32 | 0.32 | 0.2 | 0.2 |
| class 2 service (exp.) | - | - | 0.28 | 0.28 | 0.83 | 0.83 | - | - | 0.2 | 0.2 |
| reinvoked requests (exp.) | - | - | - | - | - | - | 0.1 | 0.1 | - | - |
| state transfer speed -sec/KB (exp.) | 0.8 | 0.8 | 0.6 | 0.6 | 0.6 | 0.6 | 0.8 | 0.8 | - | - |

| object replicas placement: | | | | |
|---|---|---|---|---|
| process node 1 | rep00 (obj0) | rep51 (obj5) | | (stateless) service object |
| process node 2 | rep01 (obj0) | rep50 (obj5) | | (stateless) service object |
| process node 3 | rep11 (obj1) | rep21 (obj2) | rep40 (obj4) | state owning object |
| process node 4 | rep10 (obj1) | rep20 (obj2) | rep41 (obj4) | state owning object |
| process node 5 | rep30 (obj3) | | | state owning object |
| process node 6 | rep31 (obj3) | | | state owning object |
| process node 7 | rep60 (obj6) | rep71 (obj7) | | (stateless) service object |
| process node 8 | rep61 (obj6) | rep70 (obj7) | | (stateless) service object |

| fault rarity (r): | 21600 sec | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| object replicas: | repX0 objX | repX1 objX | rep10 obj1 | rep11 obj1 | rep20 obj2 | rep21 obj2 | rep30 obj3 | rep31 obj3 | rep40 obj4 | rep41 obj4 |
| fault process (exp.) | 2*r | 2*r | 2*r | 2*r | r | r | r | r | 2*r | 2*r |
| restart times (exp.) | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 | 23.0 |
| fault monitoring interval - periodic (sec) | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 | 15.0 |

Table 2. Optimal checkpoint placements and fault-affected response times (effectiveness)

| request arrival rates (class-1 and class-2) | | fault tolerance scheme | checkpoint interval obj1 | checkpoint interval obj2 | checkpoint interval obj3 | checkpoint interval obj4 | selected optimal placement | class-1 fault-affected | class-2 fault-affected | indicative QoS improvement |
|---|---|---|---|---|---|---|---|---|---|---|
| 2.2 | 2.2 | LDSC-based | **100** requests | **100** requests | **100** requests | **100** requests | **100 100 100 100** | 115.2 sec | 115.737 sec | 88 % |
| 2.2 | 2.2 | PSC-based | **97** sec | **97** sec | **97** sec | **97** sec | **97 97 97 97** | 129.291 sec | 131.020 sec | 83 % |
| 2.4 | 2.4 | LDSC-based | non significant | non significant | **12** requests | non significant | 70 26 **12** 70 | 35.2019 sec | 34.4863 sec | 39 % |
| 2.4 | 2.4 | PSC-based | **97** sec | **23** sec | **23** sec | **97** sec | **97 23 23 97** | 34.1743 sec | 33.8495 sec | 33 % |
| 2.6 | 2.6 | LDSC-based | non significant | **12** requests | **12** requests | non significant | 56 **12 12** 56 | 26.2596 sec | 25.6086 sec | 46 % |
| 2.6 | 2.6 | PSC-based | non significant | **23** sec | **23** sec | non significant | 60 **23 23** 60 | 24.7089 sec | 24.2021 sec | 41 % |

For the four checkpoint placements (factors) and for three levels (namely 12, 56 and 100 for the LDSC-based scheme and 23, 60 and 97 for the PSC-based one) per factor, the chosen uniform design required only nine (9) runs in each load case. Table 2 summarizes the obtained optimal checkpoint placements, the selected optimal placement (the one with the minimum fault-unaffected response times) and the resulted fault-affected response times (effectiveness). PSC-based fault tolerance is equally effective to the LDSC-based one, in moderate load levels (arrival rates: 2.4 and 2.6), but falls short in high load levels (arrival rate: 2.2). The last column quantifies an indicative response time improvement, compared to the worst-case adjustments from those included in the experimental designs. Appropriate checkpoint placement results in response time reductions of up to 88%, for the heavy loaded system cases.

Finally, the tightest effective intervals based evaluation reveals the load levels (arrival rate: 2.2) for which the effectiveness of the tested schemes can be unacceptable. This behavior is related to the resulted queue lengths in the available service objects. Large queue lengths cause late request dispatching and in case of an object fault, the entire population of the queued requests contributes to the obtained mean. If a load level with the forenamed behavior cannot be excluded, this suggests the use of active replication in one or more objects. Table 3 summarizes the fault tolerance costs for the selected optimal checkpoint placements.

Table 3. Fault tolerance costs for the optimal checkpoint placements

| request arrival rates (class-1 and class-2) | | response times without f. t. (class-1) | response times without f. t. (class-2) | % overhead for the fault-unaffected requests (LDSC-based, class 1) | % overhead for the fault-unaffected requests (LDSC-based, class 2) | % overhead for the fault-unaffected requests (PSC-based, class 1) | % overhead for the fault-unaffected requests (PSC-based, class 1) |
|---|---|---|---|---|---|---|---|
| 2.2 | 2.2 | 19.0543 sec | 18,1990 sec | 14.5 % | 15.3 % | 28.7 % | 29.8 % |
| 2.4 | 2.4 | 9.7565 sec | 8.9463 sec | 25.7 % | 27.7 % | 25.9 % | 27.6 % |
| 2.6 | 2.6 | 7.1186 sec | 6.3596 sec | 22.5 % | 24.5 % | 18.8 % | 19.6 % |

## 5 CONCLUSION

For (combined) fault tolerance (and load balancing) schemes, simulation metamodeling on the basis of small size uniform experimental designs allowed us to identify the significant checkpoint intervals and the optimal checkpoint placements with the minimum possible cost. The identified tightest effective intervals constitute the unique criterion that makes feasible the comparison of schemes composed of possibly different replication policies, checkpoint placement mechanisms and/or load balancing strategies.

Future research prospects include the development of a UML reliability-modeling framework like the one of [1] and the device of a coherent multiobjective optimization method, for fault tolerance performance evaluation.

## REFERENCES


[1] Balsamo, S. & Marzolla, M. (2003). Simulation modeling of UML software architectures, *Proc. of the European Simulation Multiconference*, Society for Computer Simulation, Nottingham, 562-567

[2] Chen, X. & Lyu, R. (2003). Performance and effectiveness analysis of checkpointing in mobile environments, *Proc. of the 22nd IEEE Symposium on Reliable Distributed Systems (SRDS 03)*, Florence, Italy, 131-140

[3] Fang, K. T. & Lin, D. K. J. (2003). Uniform experimental designs and their applications in industry, *Mathematics Department Technical Report* No. 296, Hong Kong Baptist University

[4] Felber, P., Guerraoui, R. & Schiper, A. (2000). Replication of CORBA Objects, *Distributed Systems, Lecture Notes in Computer Science 1752*, Springer Verlag, 254-276

[5] Katsaros, P. & Lazos, C. (2003). A simulation test-bed for the design of dependable e-services, *WSEAS Transactions on Computers*, WSEAS Press, 4/2, 915-919

[6] Katsaros, P. & Lazos, C. (2004). Optimal object state transfer – recovery policies for fault tolerant distributed systems, *Proc. of the International Conference on Dependable Systems and Networks (DSN 04)*, IEEE Computer Society - IFIP, Florence, Italy

[7] Katsaros, P., Angelis, L. & Lazos, C. (2001). Applied multiresponse metamodeling for queuing network simulation experiments: problems and perspectives, *Proc. of the 4th EUROSIM Congress on Modelling and Simulation*, EUROSIM, Delfts, The Netherlands

[8] Litoiu, M., Rolia, J. & Serazzi, G. (2000). Designing process replication and activation: a quantitative approach, *IEEE Transactions on Software Engineering*, 26/12, 1168-1178

[9] Narasimhan, P., Moser, L. E. & Melliar-Smith, P. M. (2002). Strong replica consistency for fault-tolerant CORBA applications, *Journal of Computer Systems Science and Engineering*

[10] OMG (2001). Fault tolerant CORBA, *Object Management Group*, TC Doc. 2001-09-29, September 2001

[11] Szentiványi, D. & Nadjm-Tehrani, S. (2002). Building and evaluating a fault-tolerant CORBA infrastructure, *Proc. of the Workshop on Dependable Middleware-Based Systems (WDMS'02), International Conference on Dependable Systems and Networks (DSN 2002)*, Washington, DC, USA

[12] Uniform Design web pages (2000). http://www.math.hkbu.edu.hk/UniformDesign/

# AN RT-UML MODEL FOR BUILDING FASTER-THAN-REAL-TIME SIMULATORS*

Dimosthenis Anagnostopoulos[1], Vassilis Dalakas[2],
Georgios-Dimitrios Kapos[1], Mara Nikolaidou[2]

[1]Harokopio University of Athens, 70 El. Venizelou Str, 17671, Athens, Greece
{dimosthe, gdkapos}@hua.gr
[2]University of Athens, Panepistimiopolis, 15771, Athens, Greece
{vdalakas, mara}@di.uoa.gr

## ABSTRACT

Faster-than-real-time simulation (FRTS) is widely used for training, control and decision making purposes. FRTS experimentation proves to be rather demanding, requiring a consistent specification for developing such systems. This paper presents guidelines for an implementation framework, based on an industry standard, the Unified Modeling Language (UML). In particular, using the OMG UML Profile for Schedulability, Performance and Time Specification (abbreviated by Real-Time UML or RT-UML), specific timing attributes can be included in the derived UML model, which makes FRTS independent of the application examined. Thus, the implementation of relative program modules can be analyzed and realized, following the guidelines of this model, ensuring the reliability of the results within predetermined time frames.

## KEYWORDS

Faster-than-Real-Time Simulation, RT-UML, Simulation Methodology, Systems Analysis.

## 1 INTRODUCTION

FRTS is used when attempting to reach conclusions for the behavior of a real system in the near future [3]. In this type of simulation, model execution is concurrent with the evolution of the real system. Thus, the advancement of simulation time must occur faster than real world time. Furthermore, FRTS implementation becomes more demanding, due to the hard requirements real time systems have for interacting with other agents [4].

In [1] a conceptual methodology for FRTS was described, aiming at providing a framework for conducting experiments dealing with the complexity and such requirements. The following simulation phases have been identified: *modeling*, *experimentation* and *remodeling*. During experimentation, both the system and the model evolve concurrently and are put under monitoring. Data depicting their consequent states are obtained and stored after predetermined, real-time intervals of equal length, called *auditing intervals*. In the case where the model state deviates from the corresponding system state, remodeling is invoked. This may occur due to system modifications, which involve its input data, operation parameters and structure. To deal with system modifications, remodeling adapts the model to the current system state. When model modifications are completed, experimentation resumes. Remodeling can also be invoked when deviations (expressed through appropriate statistical measures) are indicated between the system and the model due to the stochastic nature of simulation, even when system parameters/components have not been modified.

Experimentation phase thus comprises monitoring, that is, obtaining and storing system and the model data during the auditing interval, and auditing, that is, examining a) if the system has been

*This research was supported in part by Pythagoras program (MIS 89198) co-funded by the Greek Goverment (25%) and the European Union (75%).

modified during the last auditing interval (system reformations), b) if the model no longer provides a valid representation of the system (deviations), and c) if predictions should be used in plan scheduling. Evidently, if conditions (a) or (b) are fulfilled, remodeling is invoked without examining condition (c).

Specific measures are monitored to determine whether system reformations have occurred. The variables used to obtain the corresponding values are referred as monitoring variables. Auditing examines *monitoring variables* corresponding to the same real time points (i.e. the current system state and simulation predictions for this point) and concludes for the validity of the model.

To achieve a consistent transition from the analysis of FRTS systems to the implementation of the corresponding program modules, a detailed and multi-facet specification is provided here, using UML [5, 6]. The descriptive capabilities of distinct types of UML diagrams are utilized to specify different aspects of FRTS systems: distinct entities and their roles, overall down to detailed logic of FRTS system, synchronized communication, and data specification. Furthermore, in the proposed specification we use elements from the RT-UML [7]. This profile, also used in [2], enables the detailed specification of critical time and synchronization requirements for FRTS components and the overall performance evaluation. Therefore, a detailed and integrated specification for FRTS systems is given, leading to standardized implementations of such systems that meet strict time requirements. Implementation may also be facilitated with the use of tools that support code generation given a UML model. Construction and execution of FRTS can now be performed assuring the validity of the results.

In section 2 we review RT-UML used in the specification of FRTS systems. An overview of the model, emphasizing on the identification of the discrete roles for actors and entities within FRTS, is presented in section 3. Due to the large extend of the detailed FRTS specification, section 4 provides only sample diagrams that specify how FRTS components implement their functionality in terms of events, activities, and actions, all of which have precise time orientation. Finally, in section 5, some conclusions are drawn.

## 2 RT-UML OVERVIEW

In UML, system modeling is based on different kinds of diagrams providing views of three main aspects of the system: structure, dynamic behavior, and management. In this paper, we define the structural and behavioral characteristics of FRTS with *use case*, *class*, *activity*, and *sequence diagrams*.

The UML diagrams just mentioned, do not provide the required degree of precision (regarding timing issues) for the specification of FRTS. Thus, we propose the use of RT-UML, which enhances UML diagrams. RT-UML does not propose new model analysis techniques, but it rather enables the annotation of UML models with properties that are related to modeling of time and time-related aspects. Therefore, timing and synchronization aspects of FRTS components are defined and explained in terms of standard modeling elements. RT-UML has a modular structure that allows users to use only the elements that they need. It is divided into two main parts (General Resource Modeling Framework and Analysis Models) and is further partitioned in six subprofiles, dedicated to specific aspects and model analysis techniques. To give emphasis on time and concurrency aspects of FRTS systems, one is able to use elements only from the General Time Modeling and General Concurrency Modeling subprofiles.

Each subprofile provides several stereotypes with tags that may be applied to UML models. A stereotype can be viewed as the way to extend the semantics of existing UML concepts (activity, method, class, etc.). For example, a stereotype can be applied on an activity, in order to extend its semantics to include the duration of its execution. This is achieved via a new tag added to the activity, specifying the execution duration. Stereotypes define such tags and their domains.

The proposed FRTS model consists of RT-UML-enhanced diagrams, which are annotated according to the conventions used in the RT-UML profile specification and its examples [7]. Stereotypes applied to classes in class diagrams are displayed in the class box, above the name of the class (a in Figure 1). However, when tag values need to be specified for a certain stereotype, a *note* is also attached (b in Figure 1). In sequence diagrams, event stereotypes are displayed over the events, while method invocation and execution stereotypes are displayed in *notes* (c in Figure 1). In activity diagrams, *notes* are also used to indicate the application of a stereotype on an activity, state or transition (d in Figure 1).

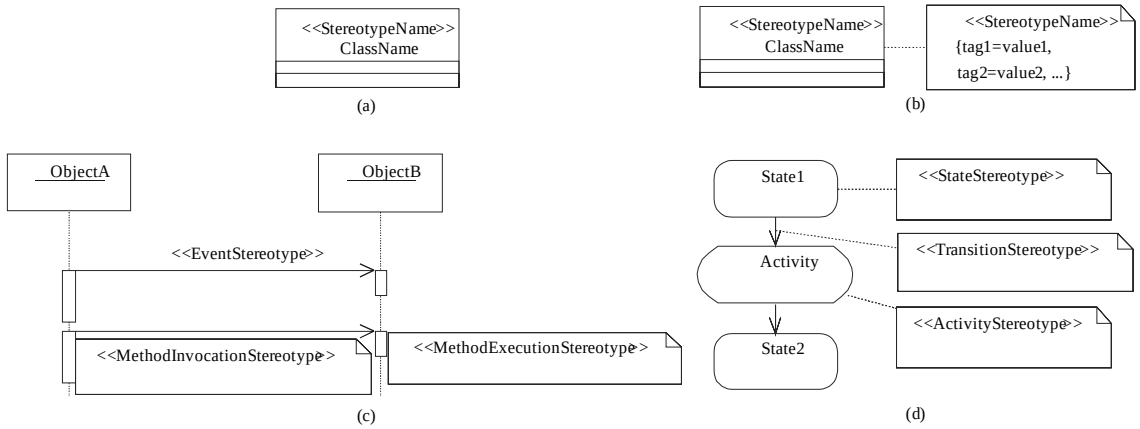



Figure 1. RT-UML notation

The RT-UML stereotypes used in the proposed FRTS model are briefly discussed here. In class diagrams of this paper we use the *CRconcurrent* and *RTtimer* stereotypes. *CRconcurrent* is used for classes of objects that may be executed concurrently. The method invoked when the object moves to "executing" state is specified with the *CRmain* tag. *RTtimer* models timer mechanisms. It defines two tags: tag *RTduration* specifies the time period after which the timer produces an event, while *RTperiodic* indicates whether the timer is periodic or not. *RTaction* is used in activity diagrams for methods, specifying the time instance they start (tag *RTstart*) and their duration (tag *RTduration*).

## 3 FRTS: A HIGH-LEVEL DESCRIPTION

This section is an overview of FRTS systems in terms of UML constructs. The use case diagram depicted in Figure 2 presents the entities involved in FRTS. Both the system and the model are separate from the main module of FRTS and can be viewed as distributed systems. System environment (SE) represents the actual system, as well as a surrounding mechanism which is responsible for performing monitoring of the real system. We consider it as a separate entity that interacts with the FRTS system. Model environment (ME) includes the model and its execution environment (MEE), while the FRTS System process is the software module responsible for controlling FRTS. Finally, the user is the actor that enables the whole process, defining the case study.

In particular, the user provides the experiment specifications and manages the FRTS System process by starting or stopping the experiment. System and model environment entities provide raw system data and raw model data, respectively. How these values are collected and stored in both environments is not of our concern. We examine only the interchange of data. The FRTS System process performs Auditing to identify potential deviations between the model and the system. In case such a deviation is indicated exceeding a respective remodeling threshold, remodeling is invoked (Remodeling), which results in the construction of a new model that replaces the one currently used (Model management).

The sequence diagram in Figure 3 emphasizes the communication between the entities described in the previous diagrams (User, FRTS System, ME and SE), in terms of message exchange. Initially, the user provides the experiment specifications (SetExperimentSpecifications) and starts the process. Thus, the FRTS System starts system monitoring (message to System Environment), initializes and starts the model, and starts model monitoring (messages to ME). In periodic, predefined time instances Audit (or state audit) is invoked. The model is then paused and the values of monitoring variables are retrieved from both the SE and ME (with GetSystemMonitoringInformation and GetModelMonitoring Information). Depending on the result of auditing the model is resumed (valid audit) or remodeling is performed and the old model is deleted (invalid audit).

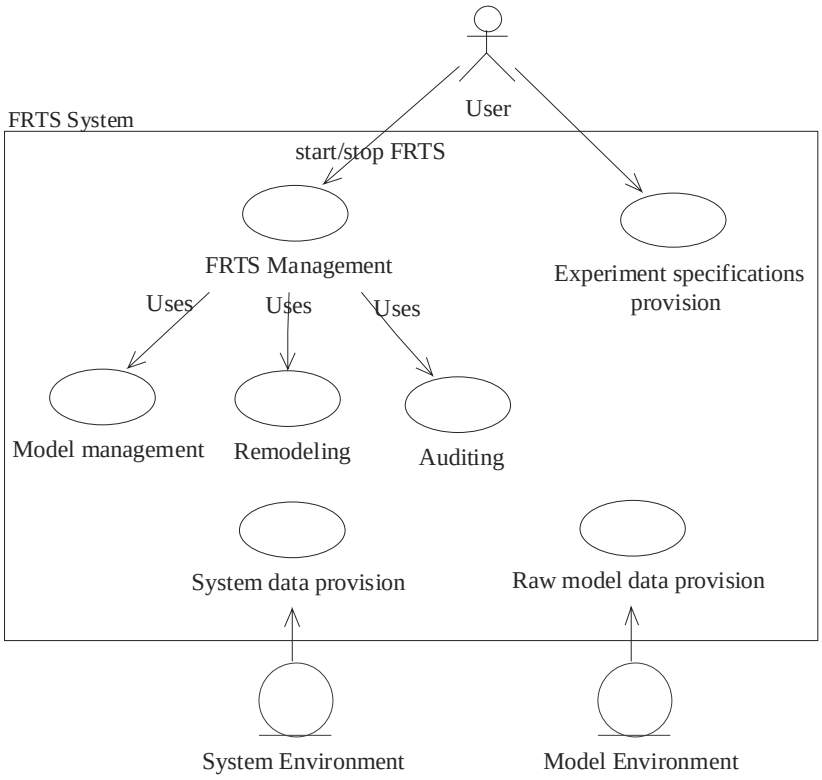



Figure 2. FRTS detailed use case diagram

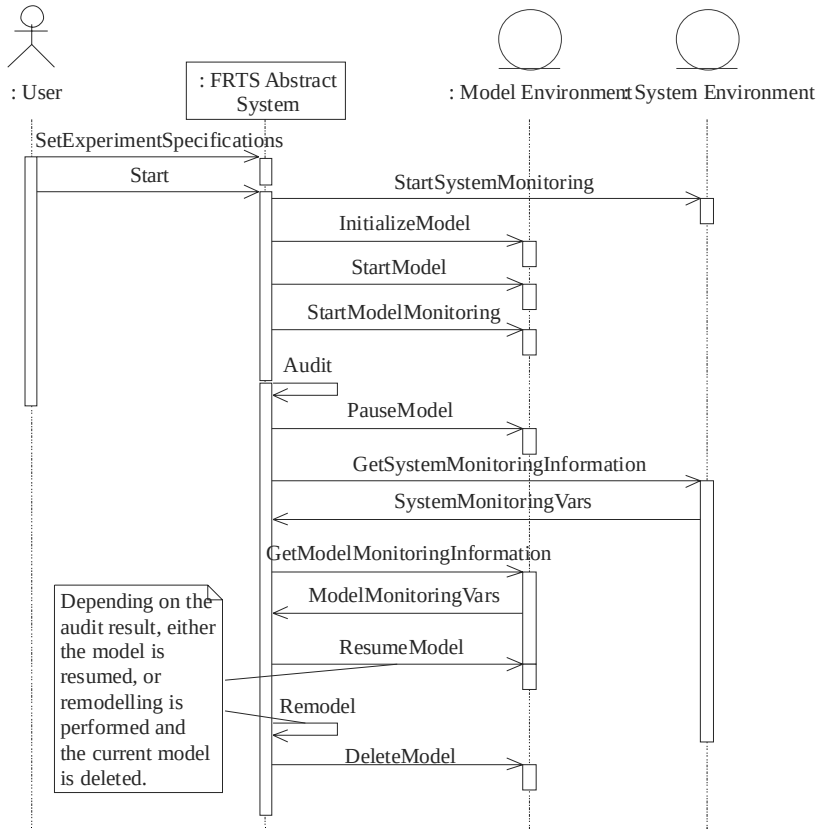



Figure 3. Main sequence diagram

## 4 FRTS System Specification

The FRTS system design is based on a set of classes (*Context*, *Control*, *Timer*, *StateAuditor*, *Auditor*, *Remodeler*, and *UserInterface*) and interfaces (*IAuditor*, *Monitor*, and *SystemMonitor*, *ModelExecutionEnvironment*), depicted in the class diagram of Figure 4. The *Context* is used for storing the experiment specifications, references to the system monitor and the model environment, and monitoring variable values used for state auditing. The *Control* initiates the FRTS process and the *Timer* is responsible for producing *StateAudit* and *Audit* events. *StateAuditor*, *Auditor*, and *Remodeller* are responsible for performing the homonymous operations. Both *StateAuditor* and *Auditor* classes implement the *IAuditor* interface. *Monitor* models the abstract concept of a variable values monitor, which is extended by interfaces *SystemMonitor* and *ModelExecutionEnvironment*. No classes are specified for the system monitor and the model environment, since they are not part of the FRTS system. FRTS components require only communication interfaces with the system monitor and the model environment. Class *UserInterface* is simply the means for introducing user requests and data.

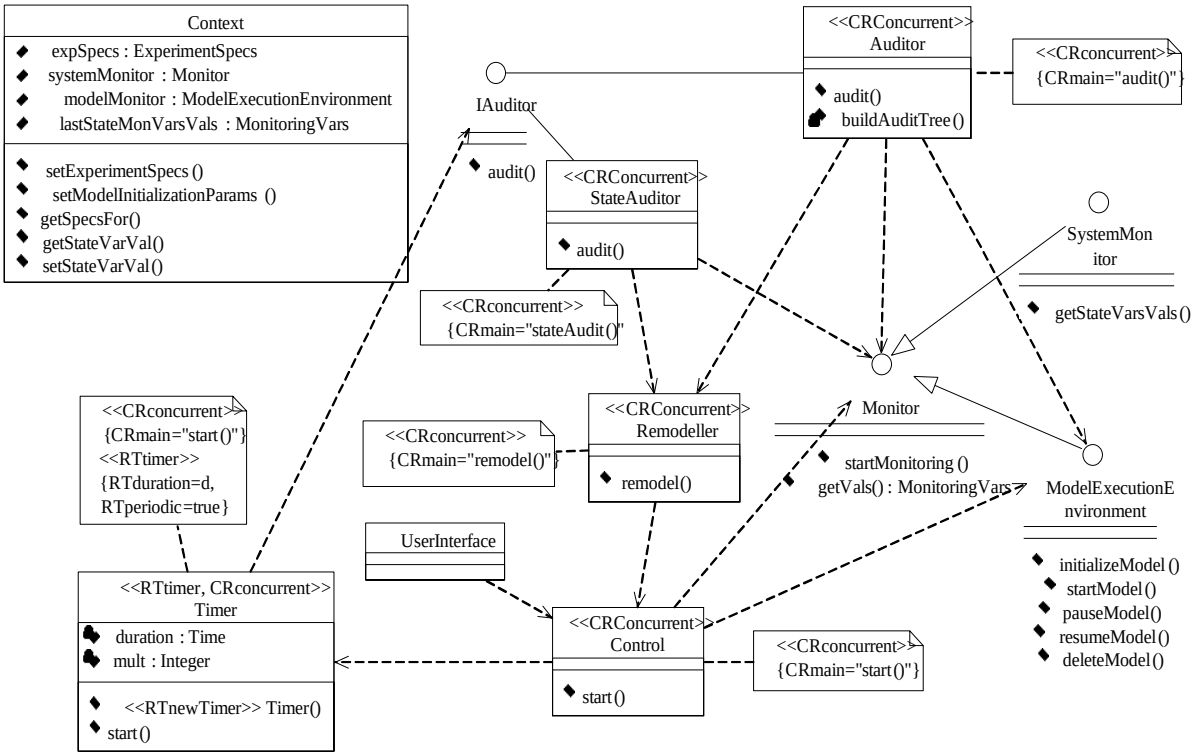



Figure 4. The main FRTS system classes

Classes *Control*, *Timer*, *StateAuditor*, *Auditor*, and *Remodeler* are intended to run on separate threads and therefore have the *CRconcurrent* stereotype. Objects of each of these classes operate independently and occasionally simultaneously. The *CRmain* tag of *CRconcurrent* stereotypes indicates the method that is executed when objects of each class are activated. Class *Timer* is a periodic producer of events, as indicated by the *RTtimer* stereotype.

The activity diagram of Figure 5 specifies the functionality of the *start()* method of *Control*. Each of the activities is annotated with the appropriate *RTaction* stereotype note. These are used to specify the duration of the activities. The lower part of each activity defines the actions executed (*do/*) or messages sent (*do/^*). Overall duration of method *start()* may be calculated as an amount of *6*b+c ms*, where *b* is the time needed for a basic operation to be performed (arithmetic operation, method invocation, etc.) and *c* is a parameter that depends on the specific FRTS application and the experiment specification. Overall duration of *start()* refers to the duration from the time instance when the user sends a *start()* event until everything has been initialized and the *Timer* has been started.

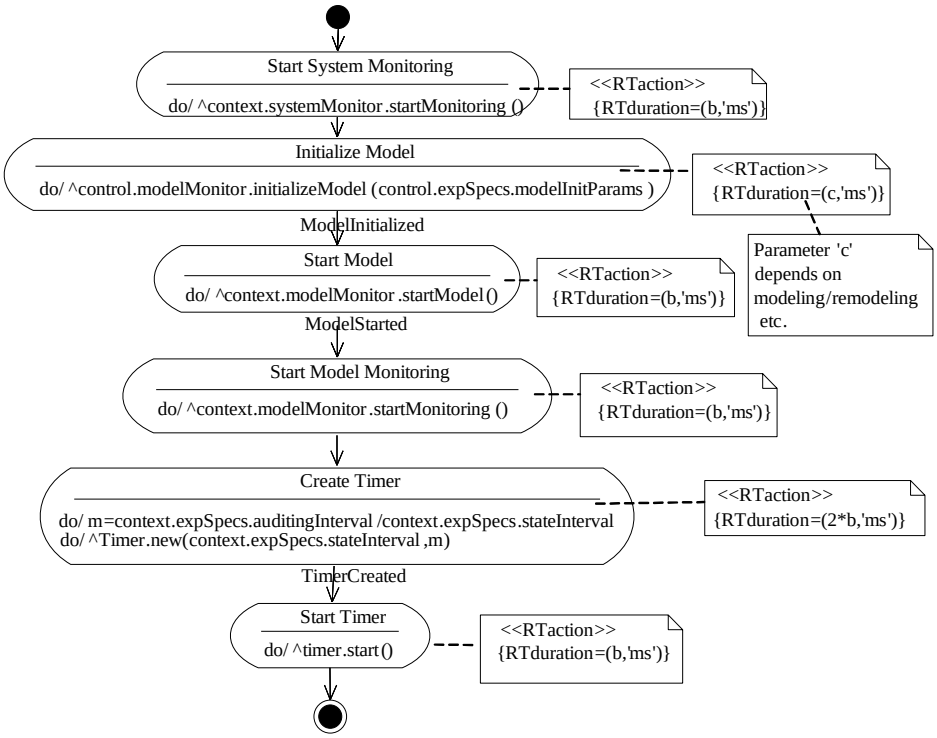



Figure 5. Activity diagram for method *start* of class *Control*

## 5 CONCLUSIONS

In FRTS model execution is concurrent with the evolution of the real system. Thus, FRTS implementation becomes more demanding, due to the hard requirements real time systems have for interacting with other agents. To achieve a consistent transition from the analysis of FRTS systems to the implementation of the corresponding program modules, a detailed and multi-facet RT-UML is provided in this paper. The descriptive capabilities of distinct types of diagrams are utilized to specify different aspects of FRTS systems: distinct entities and their roles, overall down to detailed logic of FRTS system, synchronized communication, and data specification. A detailed and integrated specification for FRTS systems is given, leading to standardized implementations of such systems that meet strict time requirements. Therefore, construction and execution of FRTS can be performed assuring the validity of the results.

## REFERENCES

[1] Anagnostopoulos, D., Nikolaidou, M., & Georgiadis, P. (1999). A Conceptual Methodology for Conducting Faster Than Real Time Experiments. *Transactions of the Society for Computer Simulation International*, 16/2, 70–77.

[2] Bertolino, A., Marchetti, E., & Mirandola, R. (2002). Real-Time UML-based Performance Engineering to Aid Manager's Decision in Multi-project Planning, in: *The Proceedings of the Third International Workshop on Software and Performance (WOSP)*, (pp. 251–261), ACM Press, Rome.

[3] Cleveland, J. *et al.* (1997). Real Time Simulation User's Guide. NASA, Langley Research Center: Central Scientific Computing Complex.

[4] Fishwick, P., & Lee, K. (1999). OOPM/RT: A Multimodelling Methodology for Real-Time Simulation. *ACM Transactions on Modelling and Computer Simulation*, 9/2, 141–170.

[5] *OMG Unified Modeling Language Specification*, v1.5, on-line at http://www.omg.org/docs/formal/03-03-01.pdf

[6] Rumbaugh, J., Jacobson, I., & Booch, G. (1998). *The Unified Modeling Language Reference Manual*, Addison Wesley.

[7] *UML Profile for Schedulability, Performance, and Time Specification*, v1.0, on-line at http://www.omg.org/docs/formal/03-09-01.pdf