

Behavioural analysis of *UML* models based on active objects

Sébastien Gérard, Pantxo Amorena, François Terrier et Jean-Pierre Gallois

LETI (CEA - Technologies Avancées) DEIN - CEA/Saclay

F-91191 Gif sur Yvette Cedex France

Phone: +33 1 69 08 62 59 Fax: +33 1 69 08 83 95

Sebastien.Gerard@cea.fr; Francois.Terrier@cea.fr; Gallois@albatros.saclay.cea.fr;

1 Introduction

In an industrial context that pushes to improve productivity, reduce development costs while preserving high levels of reliability, specification verification becomes more and more critical in the design process of large systems. It is well known that the earlier errors are detected within the development process, the lower will be the technical and financial impact to correct them. Specification verification constitutes thus a tremendous economical challenge to day.

As a matter of fact, specification validation as well as software testing is not yet fully mature. Though significant theoretical advances have been achieved in laboratories, few practical tools have reached the industrial market place. The validation work results in tedious hand made analysis achieved by highly skilled engineers and thus induces for the developed product a cost and delay increasing, often considered as too heavy relatively to the overall project.

Developing techniques and tools to efficiently support engineers in charge of such specification validation constitutes thus a major challenge whose economical impact goes far beyond the domain of safety critical applications development.

Specification validation can be handled in several ways via formal techniques implementation:

- q Theorem proving and model checking have been proven successful mainly for the validation of critical parts of systems (e.g. based on formal method such as *VDM* [D. Andrews et al. 1991], *B* [J.-R. Abrial 1996] or *Z* [J. Jacky 1997]) and for the validation of communication protocols (e.g. *LOTOS* [P. H. J. v. Eijk et al. 1989], *ESTELLE*, *SDL* [A. Olsen et al. 94]). However, these techniques still require for model validation high level skills from the user who must be aware of formal methods fundamental concepts. Moreover, the state explosion problem remains a major drawback for the analysis of large systems though some of their limitations are currently falling down with the use of specific symbolic techniques associated to well suited data structures (*BDD* and extensions [R. E. Bryant 1986]).
- q Automatic test generation is another way to tackle the problem of systems verification. Among the numerous work on the subject [E. Brinksma 1988], [C. Jard et al. 1989], [J. C. Fernandez et al. 1996], [M. Phalippou 1994], one must distinguish structural testing aimed at evaluating the control structure of an application from conformance testing whose goal is to verify that an implementation is conform to its specification. In that latter, the principle consists in building tests sets from the specification and then to replay them on the implemented system. We won't come back on the different possible ways for implementing such tests sets since our purpose is upstream from this. Indeed, one must validate the specification itself before testing conformance. A good paradigm for verifying specification is structural testing.

The most well known structural testing is the cover test that attempts to establish all possible execution paths from a system's description. It is an efficient test criteria for verification since it permits to establish if a property is true through all possible execution paths of an application.

One team of our laboratory works especially on this subject and has developed a tool called *AGATHA* [J. P. Gallois et al. 1997]. This one offers a framework and provides a methodology to incrementally verify applications specifications by applying a verification technique similar to test integration techniques for software. Various abstraction and composition operations permit to obtain simplified views of the system and to focus the analysis on subparts of particular interest. At the beginning of the project, *AGATHA* was a tool set designed to be connected to existing specification environments such as *ASA+* from Verilog. Its inputs are system specifications described in a *LTS* (Labelled Transition System) formalism from which it computes a symbolic execution tree that can easily be analysed by users with graphical views. This formalism is a restriction of the *LSA+* language resulting from the *ESTELLE* norm. This language is dedicated to the specification of systems based on parallel automata communicating with « rendez-vous » (the *LSA+* language, [J. P. Gallois et al. 1999]). Ongoing developments will ensure interfacing with other formalisms such as *SDL (ref ITU)* and *StateCharts (Harel87 et RTS'2000)*.

Formal approaches seem to be promising in the context of specification verification. But their main drawback is to be difficult to tackle because of their underlying mathematical formalism. This point has also contributed to heavily impair their using within industrial projects. Engineers prefer often to apply more operational techniques even if they are not so reliable than formal ones. Objects oriented approach is one of such operational techniques and their intrinsic features such as, encapsulation and inheritance, are well appreciated by developers. Moreover, the incoming of *UML*, as a standard language for object oriented modelling, was a major factor in the fact that they are now more and more used within industrial projects.

A second team of our laboratory focuses its activity on modelling aspects of the system development. It is particularly involved in OO domain and tries applying them to model real-time systems. To achieve this purpose, a framework prototype, called *ACCORD/UML*, has been developed. This one is composed of the definition of a *UML* profile dedicated to automotive system development. This profile is equipped through extensions added to the *UML* general tool

Objecteering from the Softeam society. These extensions allow to generate automatically the code of an application regarding to their real-time aspects.

The idea was so to take advantage of the different, but complementary, abilities of both teams, in terms of specification modelling and in terms of specification validation. Our purpose is also to make the test generator using as open and automatic as possible in order the designer doesn't need to know, neither how the AGATHA tool is running, nor its specific formal input-language.

2 General principles of the work

The Figure 1 illustrates the test generation process from UML models:

- q the designer invokes the tool called, UML2LSA+, which translates UML models specification into a specification written in LSA+;
- q AGATHA analyses the specification and supplies its results under the form of a textual file. This file contains the description of the symbolic execution tree that represents all possible execution paths of the analysed system. Indeed, an execution path represents a state series that depends on the input values applied to the system. The set of possible value ranges for each input value implying an execution path defines its called Path Condition (PC);
- q in order to make the results file more readable, it is analysed via a second tool called, SeqDiagGenerator, which constructs one sequence diagram per possible symbolic execution path identified by AGATHA;
- q the designer can also study these sequence diagrams and fix the errors in his/her models;
- q this process can be repeated until the model is correct.

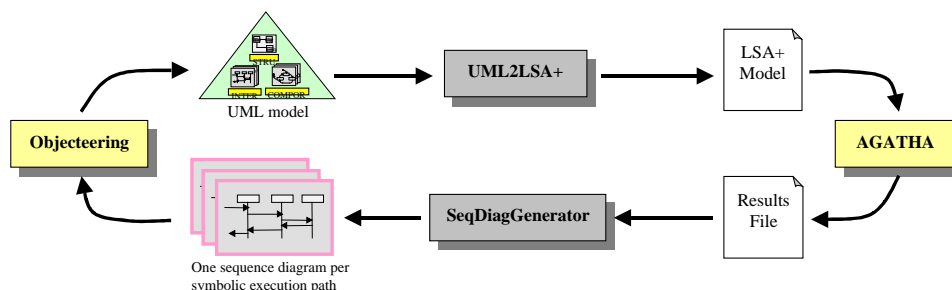


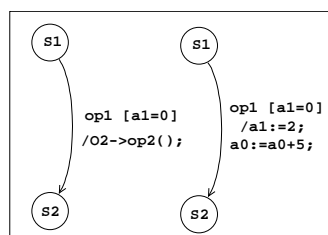
Figure 1 : Principle of the AGATHA behavioural analysis of UML models.

3 Principles of the UML2LSA+ tool

AGATHA relies on a minimal set of the LSA+ language. So in order to have a translator as simple as possible, we have first limited the possibility of the UML state machines specification to be as closest as possible of the LTS formalism analysable by AGATHA. In a further stage, the idea is to translate more general UML specification into this minimal subset.

So the restrictions of UML state machine semantics we made are following:

- q simple states and simple transitions only;
- q only CallEvent triggers;
- q one CallAction or a sequence of assignment actions per transition;
- q no action on states (i.e. no Activity, Entry or Exit actions).



The execution semantics description of UML state machines relies on the definition of an underlying abstract machine implementing the state machine. It is made of three parts:

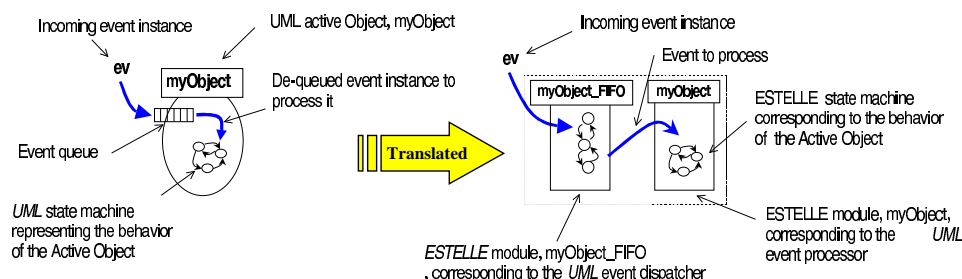
- q an event queue storing the incoming events till they are consumed;
- q an event dispatcher that selects and dequeues the event instances contained in the event queue;
- q an event processor that consumes the event instances supplied by the event dispatcher.

In the other hand, we have to respect some important points, active objects of an application communicate through asynchronous message passing and yet according to AGATHA assumption on LSA+, the communication mode between LSA+ modules is based on the « rendez-vous ». This constraint has lead us to the following choice. Each active object of an application is translated into two LSA+ modules (cf. Figure 2):

- q the first one is equivalent to the event processor defined in the abstract machine of UML. It is responsible for consuming the event instances received by the object. It is very similar to the UML state machine specification describing the behaviour of the object.
- q the second one represents the queuing mechanism of active object communication. It will insure asynchronous communication mode between objects of an application, that has to be non -stopping, while the communication

through « rendez-vous » on which relies AGATHA is stopping. Indeed with « rendez-vous », the caller is stopped until the callee takes into account the request of the calling module. So, this second module stores all the incoming event addressed to the modelled object and dequeues them.

Moreover, execution semantics of *UML* stat machines is widely based on the RTC assumption. That means that an event instance may be dequeued and dispatched only when the processing of the previous current event instance is ended. For that, regarding to the *UML* RTC assumption, this requires a synchronisation mechanism between the two modules (events processor and events dispatcher) to insure the respect of the RTC assumption. Finally, the event dequeuing policy adopted for our study is of type FIFO (cf. Figure 2).



This approach starts from a detailed specification written with a high level language, *UML*, and transforms it automatically in a formal abstract model. So, a user has at his/her disposal the power of the *UML* language to specify her/his applications, and the power of formal approaches to validate his/her specifications. But formal techniques involved in the validation process are fully transparent for its user.

The analysis realized by *AGATHA* on *UML* models supplies in an exhaustive and formal way all the symbolic execution paths of the specification. So, the identification of all possible paths allows the user to study issues linked to real-time behaviour as deadlock, livelock... Moreover, he or she can also formally evaluate for each paths the resource used in terms of time and memory. Therefore, it is possible to tackle issues such as time and memory consuming in an automatic and formal way.

5 References

- [J.-R. Abrial 1996] J.-R. Abrial, "The B-Book: Assigning Programs to Meanings", Cambridge University Press Pub., Ed., pp. 850, 1996.
- [D. Andrews et al. 1991] D. Andrews, D. Ince, "Practical Formal Methods with VDM", McGraw Hill Pub., Ed., pp. 450, 1991.
- [E. Brinksma 1988] E. Brinksma, "A theory for the derivation of tests", Protocol Specifications, Testing and Verification, IFIP, Elsevier Science, pp. 63-74, North-Holland, 1988.
- [R. E. Bryant 1986] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation", Vol. 8, pp. 677 - 691, 1986.
- [P. H. J. v. Eijk et al. 1989] P. H. J. v. Eijk, C. A. Vissers, M. Diaz, "The formal description technique LOTOS", Elsevier Science Pub., Ed., 1989.
- [J. C. Fernandez et al. 1996] J. C. Fernandez, C. Jard, T. Jérón, L. Nedelka, C. Viho, "Using on-the-fly verification techniques for the generation of test suites", CAV, Springer-Verlag, Vol. LNCS 1102, 1996.
- [J. P. Gallois et al. 1997] J. P. Gallois, A. Lanusse, "Le test structurel pour la vérification de spécifications de systèmes industriels", in Génie Logiciel, Vol. 46, pp. 145-150, 1997.
- [J. P. Gallois et al. 1999] J. P. Gallois, A. Lapitre, "Analyse de spécifications industrielles et génération automatique de tests", ICSEA, Paris, 1999.
- [J. Jacky 1997] J. Jacky, "The Way of Z: Practical Programming with Formal Methods", Cambridge University Press Pub., Ed., 1997.
- [C. Jard et al. 1989] C. Jard, T. Jérón, "On-line model-checking for finite linear temporal logic", Automatic Verification Methods for Finite State Systems, International Workshop, Springer-Verlag, Vol. LNCS 407, pp. 275 -285, Grenoble, France, 1989.
- [A. Olsen et al. 94] A. Olsen, O. Faergemand, B. Moller-Pedersen, R. Reed, J. R. W. Smith, "Systems Engineering Using SDL-92", Elsevier Science Pub., Ed., North-Holland, 94.
- [M. Phalippou 1994] M. Phalippou, "Test sequence using Estelle or SDL structure information", FORTE'94, Berne, 1994.