Consiglio Nazionale delle Ricerche

*F*ormal verification of OMT dynamic models

Alessandro Fantechi, Stefania Gnesi

B4-09
ago-1997

S.T.A.R.
Servizio Tecnografico Area di Ricerca del CNR - Pisa
dicembre '97 324,134

Formal verification of OMT dynamic models

Alessandro Fantechi[1] and Stefania Gnesi[2]

[1] Dipartimento di Sistemi e Informatica, Università di Firenze, Italy
[2] IEI - C.N.R., Pisa, Italy

Abstract. This paper proposes to apply model checking to the verification of the consistency between diagrams produced in the analysis of a system following the OMT methodology. It is then shown how this verification technique can be integrated in a commercial CASE tool.

1 Introduction

Industrial acceptance of formal methods is hampered by their notational difficulty and by the amount of theoretical knowledge often needed to apply them. Rigorous, though informal, methods, often based on graphical notations, have been instead more and more widely and successfully applied in industry. A graphical notation is able to vividly represent some intuition on the description of a problem at the analysis stage and still is able to guide the following design in a rigorous manner, to a stage in which important pieces of code can be drawn directly from such pictures and the associated textual annotations. Many so called "CASE" tools are able to support this development trajectory, from the informal analysis stage to the rigorously generated code. The possibility of generating code from pictures reveals that picture themselves should have a formal definition, in order to be able to define a code generation algorithm; often CASE tools check the semantic consistency of pictures before generating the code, not allowing any code production if the pictures have no precise semantic meaning; the development process should be able to refine the initial informal pictures of the analysis phase into more precise ones at the end of the design phase.

The precise definition of the meaning of pictures can be taken as the basis to apply techniques developed in the formal methods field to increase the verification and validation capabilities of commercial CASE tools.

In particular, we consider the OMT methodology [16], in which three models (Object, Dynamic, Functional) of a system are defined at the analysis phase, all represented by graphical notations, which are then refined in later design stages until SQL or Object-Oriented program structures can be generated.

A formalization of OMT methodology has been already considered in [2, 4, 5]. In [6], a formalization of the Object and Dynamic models using the B method [1] is proposed. This formalization allows reachability analysis to be performed by theorem proving on the Abstract machines which formalize the State Transition diagrams of the OMT Dynamic model.

Model checking [7] is an established techniques to prove properties of state machines, and provides an effective alternative to theorem proving in all the

cases in which the complexity of the system in terms of number of states is not exceedingly great. Being an algorithmic technique, it can be applied without involving any human formal reasoning, and this should be well accepted in the industrial practice.

In this paper we show how model checking can be applied to prove algorithmically properties on the state diagrams of the OMT dynamic model, and in particular, how such algorithms can be embedded in a CASE tool so to hide any formalities to a user, letting him/her to express properties only by reasoning on the pictures drawn by him/herself.

2 OMT methodology - the Dynamic model

The OMT methodology is based on the definition, at the analysis phase, of three models of a system (Object, Dynamic, Functional), which show different viewpoints of the same system.

The Object model describes a system as a collection of classes with their relations (aggregation, inheritance, or any other relation): therefore the Object model is able to describe the structure of the system, especially in terms of data and their relationships.

The Dynamic model describes the behaviour of the system, both in terms of the internal behaviour of the single classes (through state machine definitions) and in terms of the interactions between classes and between the system and the external environment.

The Functional model describes, by means of Data Flow Diagrams, the functionality of the operations of the system, without reference to which class and within which behaviour is performing them.

The kind of the application dictates the relative importance of one of the three models over the others. While, for example, data-base oriented applications will be described mainly by the Object model, control-intensive applications will put significant emphasis on the Dynamic model.

In this paper we are especially concerned with the Dynamic Model, and therefore targeted to control-intensive applications.

The Dynamic model of a system is composed of three kinds of diagrams:

- Class Communication diagram, which shows the topology of the interactions between the classes of the system;
- State Transition diagrams, one for each interacting class. They represent the behaviour of classes as automata where transitions between states model the interactions with other classes; Actually, OMT State Transition diagrams are based on Statecharts [13] in which a single state can be decomposed in several substates.
- Event Trace diagrams, also called Scenarios, which are important interaction schemes, each involving a subset of the classes of the system.

Actually, scenarios are the first diagrams drawn to define the Dynamic model. For their intuitive appearance, they are a suitable mean to fix once for all the

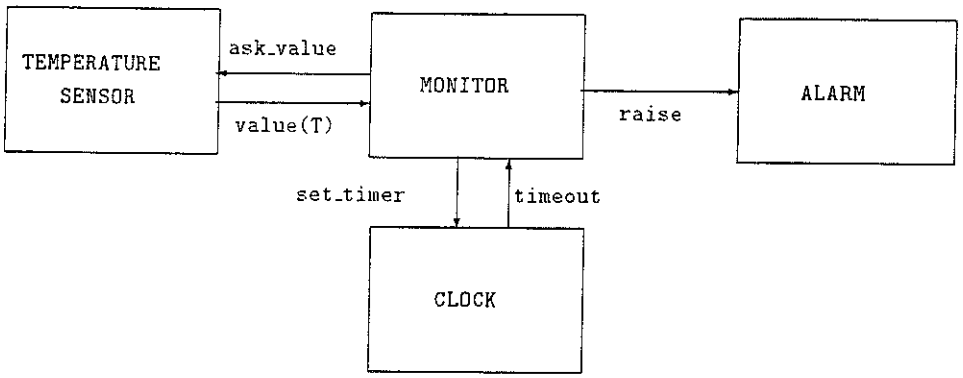

Fig. 1. The Class Communication diagram of a monitoring system

main interaction schemes in the early specification phases (see the examples of Fig. 2 and 3 for a system composed of a monitor, a temperature sensor, a clock and an alarm device).
The later definition of Class Communication diagrams and State Transition diagrams which together contribute to a complete definition of the behaviour of the system, will have to comply with the defined scenarios. Here the validation problem we aim to solve comes: do the Class Communication diagrams and the State Transition diagrams comply with the Event Trace diagrams?
To give an example, consider the Class Communication diagram of Fig. 1 of the example monitoring system, and the State Transition diagram given in Fig. 4 that represents a possible behavior of the monitoring system. The question that can arise is: is this behaviour compatible with the Event Trace diagrams of Fig. 2 and 3

To give an answer to questions like this a solution can be found in already established formal verifcation methods, techniques and tools. In particular in this paper we will use for this purpose the verification methodology supported by the JACK environment [12].

3 The model checking facilities of the JACK environment

The idea behind the JACK environment[3] was to combine different specification and verification tools, independently developed at different sites. A first experiment in building verification tools, based on existing ones, is described in [8]. Following this attempt, the JACK environment has been developed exploiting

[3] Detailed information about JACK and its component tools are available at http://rep1.iei.pi.cnr.it/Projects/JACK

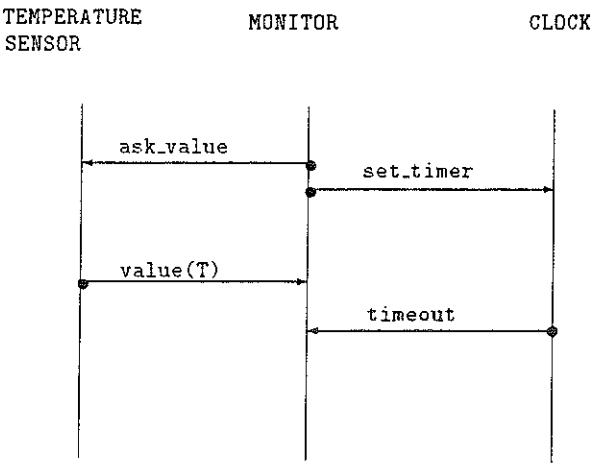

Fig. 2. An Event Trace diagram for the monitoring system

the FC2 format [3, 14] which has been proposed as a standard format for representing automata. In this format an automaton is defined by means of a set of tables which keep the information about state names, arc labels and transition relations between states. One of the main objectives of JACK was to provide an environment in which a user can choose from several specification and verification tools by a simple, user-friendly graphic interface and create a general system for managing any tool that has an input or output based on FC2 format files. Such tools can easily be added to the JACK environment, thus extending its potential.

Some of the tools within JACK allow a system specification to be built. This can be done both by entering a specification in a textual form as a process algebra term and by using sophisticated graphical procedures to build a specification as an automaton or as a net of automata. Then, model checking techniques [7] are used to verify safety and liveness properties of a system. The action based version of CTL [11], called ACTL [9], is used in JACK to formalize properties of concurrent systems. ACTL is a temporal logic that is suitable for describing the behaviour of systems that perform actions during their working time. In fact, ACTL embeds the idea of "evolution in time by actions" and is suitable for describing the various possible temporal sequences of actions that characterize a system behaviour (i.e. ACTL is a *branching time* temporal logic).

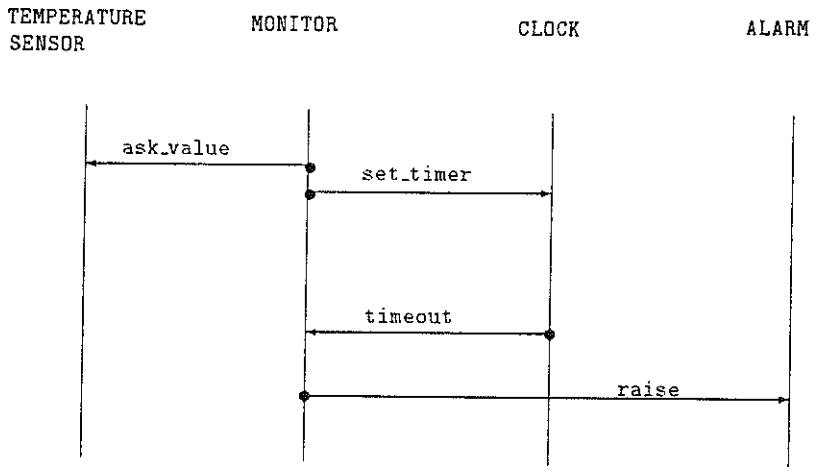

Fig. 3. Another Event Trace diagram for the monitoring system

4 Verifying OMT diagrams within JACK

To be able to use the verification facilities of JACK , the OMT diagrams have to be transformed into Fc2 automata that are its required input format.
This can be done in a straightforward way.
The topology given by the Class Communication diagram can be directly mapped into a Fc2 network topology, since Fc2 already is able to describe a "network" of interacting processes, each described in its turn by a State Transition diagram (an automaton).
The State Transition diagrams can be directly written as Fc2 automata. In Statecharts, a single state can be decomposed in substates: this require a more complex, but still feasible, translation into fc2, which will explode every state of the State Transition diagram into the constituent states. For even simple, but not plain, Statecharts, this translation can produce a so called "state space explosion", since the number of states of the produced model can grow exponentially with the number of subcomponents. Several techniques and tools can be used to afford this problem, but this issue is not discussed further in this paper.
Event Trace diagrams are then expressed as a temporal logic formulae. Actually, Event Trace diagrams represent conditions that in some way constrain the behaviour of a system. A logical formula, in our case an ACTL formula, well express this kind of constraints. To be more precise, an Event Trace diagram express the existence of a path of interactions and it has a direct correspondence with an ACTL formula which complies with the scenario. That is, if the event trace gives the sequence `act1, act2, .., actn`, the corresponding formula will be:

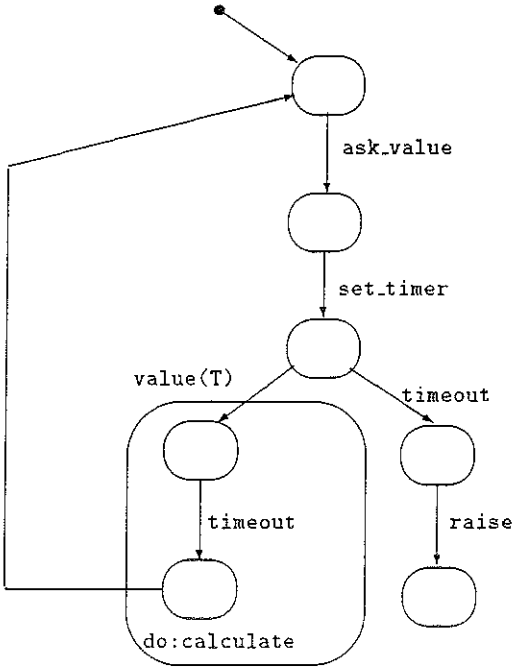

Fig. 4. A State Transition diagram

```
<act1><act2> .... <actn>true
```

That means: "there exist a path in which the actions `act1, act2, .., actn` are performed in sequence".
For example, the formula corresponding to the event trace of Fig. 2 is the following:

```
<ask_value>
   <set_timer>
     <value(T)>
         <timeout>true
```

To answer the question: "is the system specification compliant with the given scenario?" we have to perform the model checking of the related formula on the state model of the system obtained by considering as observable only the actions which are indicated in the event trace. It can be observed that both in the state transition diagrams and in the event trace diagram the interaction names may convey some value.

5 Architecture of an enhanced CASE tool

To provide in a transparent way to an OMT user the outlined verification facilities, we propose to enhance a commercial CASE tool with the verification modules (as the ones taken from JACK), by building the following "interface" modules:

- translator from Class Communication Diagrams (given in the internal CASE tool format) into FC2 networks;
- translator from State Transition Diagrams into FC2 automata;
- translator from Event Trace Diagrams into ACTL formulae;
- invocator of the global automaton generator, under the observability conditions required for the scenario under examination, and of the model checker.

At this regard, we can observe that we have limited ourselves to a quite simple analysis of the compliance to the scenario. For example, a safety requirement could indicate that a certain scenario should not occur. This is the case of the "undesired scenario" of Fig. 5 for the considered example, which expresses that the alarm can be raised even though the asked value has been received in time. The validation of the safety requirement amounts to prove by model checking that the formula associated to the undesired scenario is not true for the system.

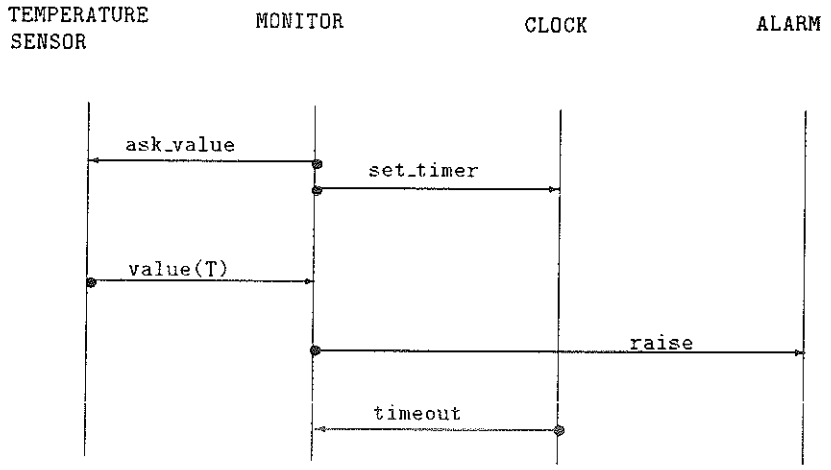

Fig. 5. Another Event Trace diagram

6 Conclusions

The use of a model checker of a powerful temporal logic such as ACTL means that actually only a fraction of this power (essentially, the existential linear fragment) is used in the proposed scenario validation. On the other hand, the verification power given to the user is limited by the notation (Event Trace diagrams) available for the expression of validation questions.
We can observe that we are using Event Trace diagrams as a visual query language, that is, as a visual representation of the temporal logic formula that it is asked to hold. Visual representation of temporal logic formulae has already been studied [15, 10], using more complex graphic notations, able to express the richness of the logic, but maybe less immediate as a visual tool: anyway, they deserve attention for a possible inclusion in diagram-based CASE tools, together with their verification tools.
Further work will be in the direction of investigating what could be the optimal compromise between the expressive power of the (visual) formalisms proposed and their easiness of use in the industrial practice.

References

1. J.R. Abrial. "The B Book. Assigning Meanings to Programs", Cambridge University press, 1996.
2. R. H. Bordeau and B.H.C. Cheng, "A formal semantics for Object Model Diagrams", IEEE Transactions on Software Engineering, 10, 1995.
3. Bonali, A., Ressouche, A., Roy, V., De Simone, R., "The FC2TOOLS set," Workshop on Computer-Aided Verification (CAV'96), Lecture Notes in Computer Science 1102 , Springer-Verlag, 1996.
4. T.C. Hartrum and P.D. Bailor, in "Teaching Formal Extensions of Informal-Based Object-Oriented Analysis Methodologies, pp.389-409, Computer Science Education, 1994.
5. F. Hayes and D. Coleman, "Coherent Models for Object-Oriented Analysis", OOPSLA'91, pp.171-183, 1991.
6. E. Bertino, D. Castelli, F. Vitale, "A formal representation for State Diagrams in the OMT methodology", SOFSEM'96, LNCS 1175, 1996.
7. E.M. Clarke, E.A. Emerson, A.P. Sistla. Automatic Verification of Finite State Concurrent Systems using Temporal Logic Specifications. *ACM Transaction on Programming Languages and Systems*, vol.8, n. 2, 1986, pp. 244-263.
8. De Nicola, R., Fantechi, A., Gnesi, S., Ristori, G., "An action-based framework for verifying logical and behavioural properties of concurrent systems," Computer Network and ISDN systems, 25, February 1993, pp. 761–778.
9. De Nicola, R., Vaandrager, F.W., "Action versus State based Logics for Transition Systems." Proceedings Ecole de Printemps on Semantics of Concurrency, Lecture Notes in Computer Science 469, Springer-Verlag, 1990, pp. 407–419.
10. A.Del Bimbo, L.Rella, E.Vicario, Visual Specification of Branching Time Temporal Logic, VL95, IEEE Symp. on Visual Languages, Darmstadt, Germany, 1995.
11. Emerson, E.A., Halpern, J.Y., "Sometimes and Not Never Revisited: on Branching Time versus Linear Time Temporal Logic," Journal of ACM, 33 (1), January 1986, pp. 151-178.

12. Gnesi, S., "A Formal Verification Environment for Concurrent Systems Design" ENTCS, Vol.6, 1997, to appear.
13. D. Harel, "On visual formalisms", Comm.ACM, 31 (1988), 514-530.
14. E. Madelaine and R. De Simone. The fc2 reference manual. Technical report, INRIA, 1993.
15. L. Moser, Y. S. Ramakrishna, G. Kutty, P. M. Melliar-Smith and L. K. Dillon, "A Graphical Environment for Design of Concurrent Real-Time Systems", ACM Transactions on Software Engineering and Methodology, January 1997.
16. J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, "Object-Oriented Modeling and Design", Prentice Hall, Englewood Cliffs, 1991.

This article was processed using the LaTeX macro package with LLNCS style