STUDIA UNIV. BABEŞ–BOLYAI, INFORMATICA, Volume **XLVI**, Number 2, 2001

ACTIVECASE - TOOL FOR DESIGN OF CONCURRENT OBJECT-ORIENTED APPLICATIONS

DAN MIRCEA SUCIU

Abstract. Object-oriented concurrent programming is a methodology that seems to satisfy nowadays requirements for complex application development. Issues like inheritance anomalies or developing of object models that integrate in a natural way concurrent programming elements with object-oriented concepts was intensely analyzed in literature.

Construction of a consistent modeling mechanism that ameliorates the inheritance anomalies as much as possible represents the main goal of our research work ([13], [14]). This paper presents the implementation of this modeling mechanism into a CASE tool for analysis and design of concurrent object-oriented applications. Developing specific scalable statecharts for behavior modeling of active objects and automatic code generation are subsequent issues that are attend to validate the executability of our mechanism.

Key words: CASE tools, object-oriented concurrent programming, reactive systems, statecharts.

1. Introduction

CASE (*Computer Aided Software Engineering*) tools are software products able to support medium or large application development. This support is realised by automating some of the activities made in an analysis and design method. If we agree that one of the main goals of an analysis and design method is code generation and that we should obtain automatically a high rate of application code, it is obvious that an efficient use of a method cannot be made without an associated CASE tool.

Typically, the translation of a complex analysis/design model into a programming language takes a long period. A model is called *executable* if this translation can be made automatically. The automatization of the translation process allows running a prototype of an application immediately after building its model.

The executability is an important feature of *scalable statecharts* [13], allowing the automatization of active objects implementation based on their behavioral

2000 *Mathematics Subject Classification.* 68N30.

1998 *CR Categories and Descriptors.* D.2.3 [**Software**] : Software Engineering – *Coding Tools and Techniques*; D.2.7 [**Software**] : Software Engineering – *Distribution, Maintenance and Enhancements* .

models. Furthermore, the executability offers support for simulation, testing and debugging of active object execution at the same level of abstraction like the built model.

The paper describes the architectural and functional features of a CASE tool designed for modeling, developing and simulation of concurrent object oriented applications. This tool, called ActiveCASE, is complete original and supports active objects behaviour modeling through scalable statecharts formalism as described in [13] and [14]. In addition, ActiveCASE allows concurrent class structure specification, active object behavior modeling and source code generation.

Section 2 presents the meta-model of class diagrams and scalable statecharts implemented in ActiveCASE.

A detailed description of tool functional features is shown in section 3.

Section 4 validates the modeling capacity and executability of scalable statecharts describing the development and modeling process of an application for traffic control on a rectangular track. This sample exploits all scalable statecharts features described in [13] and [14].

2. The meta-model of scalable statecharts

In this section is presented in detail a static meta-model of scalable statecharts. This meta-model is used in scalable statecharts implementation in ActiveCASE (figure 1).

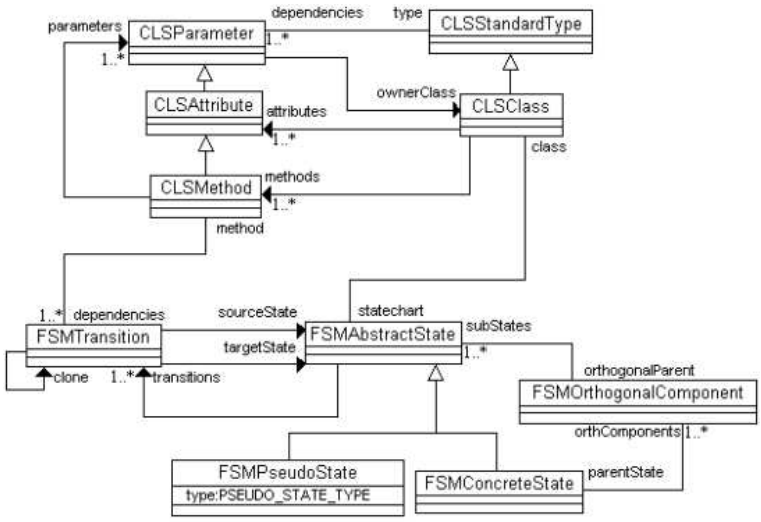

Figure 1. The meta-model of class and state diagrams used in ActiveCASE

Because a state diagram is associated with a class, ActiveCASE supports also a primitive class diagram editor. This kind of diagrams is not an important feature of ActiveCASE, because the tool focuses on behavioural models. However, ActiveCASE can be easily interfaced with other existing CASE tools. The classes that models class diagrams are: **CLSClass** (models a class), **CLSStandardType** (models primitive types like integer, float, string, boolean etc), **CLSParameter**, **CLSAttribute** and **CLSMethod** (model the properties and operations of a specific class).

In ActiveCASE simple states are viewed like composed states with zero substates, and a non-concurrent state like a state, which contains one orthogonal component. This manner of considering state diagrams allows the elimination of redundant classes from meta-model. In the same time, this representation semantically unifies the concepts of simple state, composed state and orthogonal state. Figure 2 shows in a graphical manner the relationships between entities of scalable statecharts.

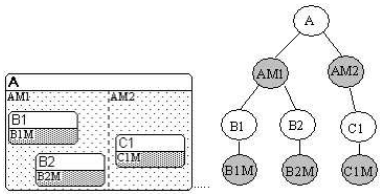

FIGURE 2. Graphical representation of scalable statecharts

FSMAbstractState abstract class models all kind of states defined in Level 2 scalable statecharts (SS^2). An object of **FSMAbstractState** class contains a list of (incoming and outgoing) transitions (modeled by **FSMTransition** class) and a list of orthogonal components (modeled by **FSMOrthogonalComponent** class). In ActiveCASE there are two state categories: pseudostates (initial, final, history - modeled by **FSMPseudoState** class)and concrete states (modeled by **FSMConcreteState** class). A concrete state contains at least one orthogonal components.

Another important element of our model is that a class has not associated a state diagram or a state machine, but a concrete state. This concrete state represents the parent (root) of all states that describes the behavior of associated class objects. This particular state will have the same name as the modeled class, and its invariant corresponds to the consistency condition imposed on class objects. Practically, the concept of state diagram is not used anymore, and the behavior of objects is described through a state hierarchy.

The auto-transition from **FSMTransition** class level assigns to each transition a 'clone' used in scaling (minimizing or maximizing) composed states. In figure

3 is presented a sample where this double transition is useful. The transitions labeled with m1 and m3 link states from different nesting levels. When *State2* is minimized, its sub states will be 'hidden' and the same thing will happen with their transitions.

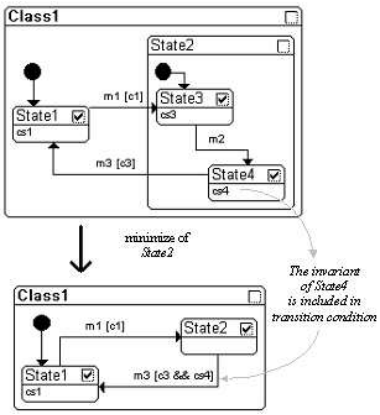

FIGURE 3. Cloning transitions

Therefore, is necessary to introduce supplementary transitions that will preserve the behavioral model. These transitions double the original transitions, and they link states *State1* and *State2*. Also, 'clone' transitions do not influence the source code generation. Their launch conditions are conjunctions between the original condition and the invariant of source state (for the transition labeled with *m3*, its clone will have attached the condition *c3* && *c4*, where && is logical AND operator from C++ programming language).

3. ACTIVECASE ARCHITECTURE

The ActiveCASE tool has three main components:

- *ActiveCASE.exe* - main application, used for editing class and scalable statecharts diagrams and source code generation,
- *StateControl.ocx* - component used for specific statecharts display,
- *ActiveStatechart.dll* - component used in simulation of active objects behavior during execution of a generated application.

ActiveCASE is a tool for modeling of active objects behavior and offers support for analysis, implementation and testing phases of life cycle of an application. ActiveCASE application allows editing primitive class diagrams and scalable statecharts, and has a C++ source code generator (Figure 4).

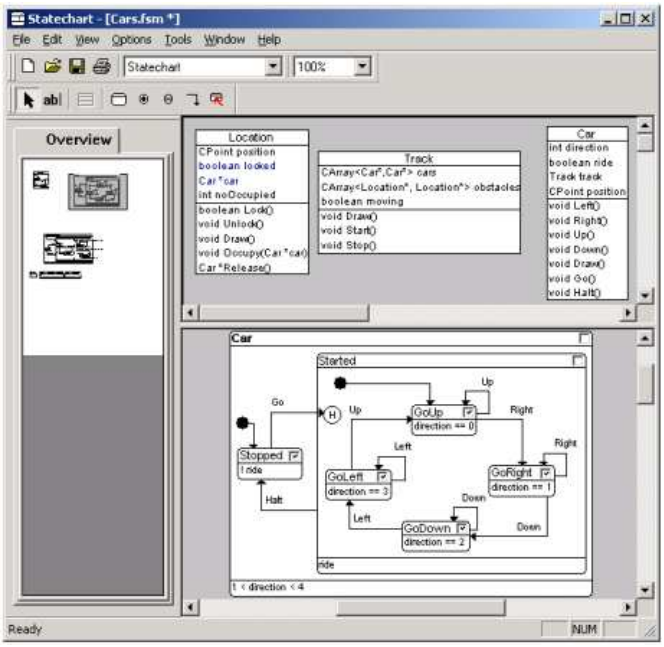

FIGURE 4. ActiveCASE capture with graphical editors of class and states diagrams

All modeled classes are sub-classes of a special class called ActiveObject. This class has attributes and operations for handling states and transitions and for interactions with simulation component.

The component used for simulation allows the visualization of concurrent objects execution during the execution of an application generated by ActiveCASE environment.

4. Modeling an application for traffic simulation

In this section is presented a sample application for traffic simulation on a rectangular track. This sample application uses all features provided by ActiveCASE tool and all new elements introduced by scalable statecharts.

Figure 5 shows the class diagram that models an application for traffic simulation on a rectangular track. A **Track** object contains a bi-dimensional array of locations and has a set of associated cars. The three operations of **Track** class allow displaying a **Track** object on screen and to start or stop all its associated cars.

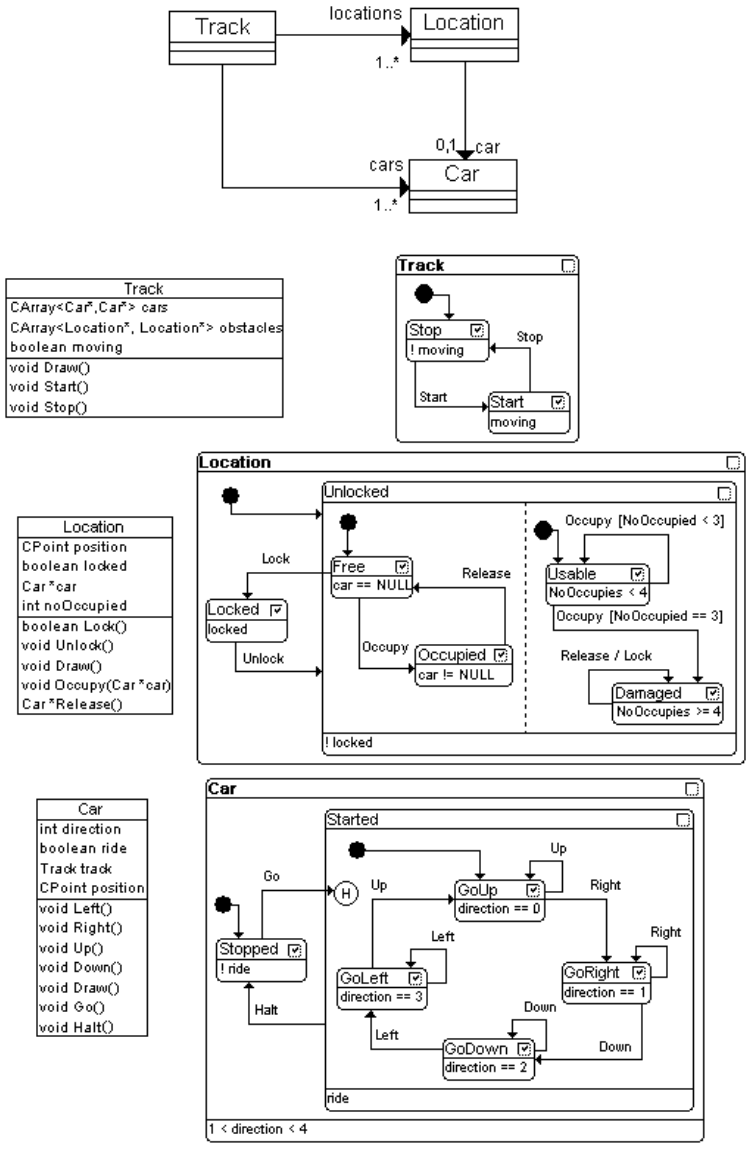

FIGURE 5. State diagrams for **Track**, **Car** and **Location** classes

All diagrams presented in figure 5 are made with ActiveCASE tool. Using ActiveCASE environment the implementation code for all three classes was generated

and attached to a Visual C++ project. The intervention of developer is necessary only for creating a **Track** object and for attaching to it a desired number of cars (**Car** objects). In addition, the developer can implement code for graphical representation of all objects.

Figure 6 shows a test of "Maşină roşie" active object behavior using the simulation of its execution using scalable statecharts.

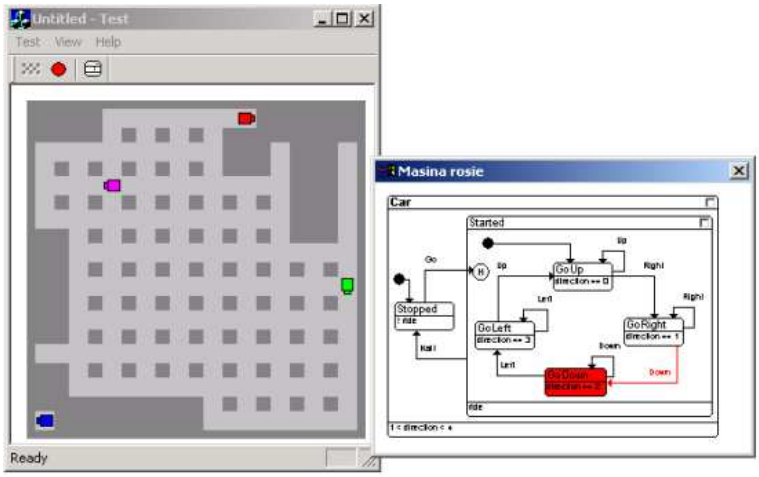

FIGURE 6. The simulation of "Maşină roşie" object behavior

5. CONCLUSIONS

The process of concurrent object oriented applications development is laborious. As we stated in the previous sections, the conceptual differences between different concurrent object oriented programming languages make difficult to translate applications from one language in another. In the same time, testing and debugging these applications is more complicated than for sequential applications. Therefore modeling these applications through a unitary set of concepts and notations and testing and debugging them at models level increase the quality of developed applications and decrease the maintenance effort.

The goal of ActiveCASE tool is to automate some steps of the developing process for concurrent object oriented applications. Its main features are:

- flexibility in modeling active objects behavior through scalable statecharts that cover most of concurrent object models;
- high level of internal concurrency specification;
- the source code generator is adaptable to any concurrent object oriented programming language;

- offers support for active objects behavior simulation at run-time;
- imposes an implementation discipline, which ameliorates reuse anomalies.

References

[1] F. Barbier, H. Briand, B. Dano, S. Rideau, "The Executability of Object-Oriented Finite State Machines", Journal of Object-Oriented Programming, SIGS Publications, 4 (11), pp. 16–24, jul/aug 1998

[2] Michael von der Beeck, "A Comparison of Statecharts Variants", Formal Techniques in Real-Time and Fault-Tolerant Systems, L. de Roever and J. Vytopil (eds.), Lecture Notes in Computer Science, vol. 863, pp. 128–148, Springer-Verlag, New York, 1994

[3] S. Cook, J. Daniels, "Designing Object Systems - Object-Oriented Modelling with Syntropy", Prentice Hall, Englewood Cliffs, NJ, 1994

[4] Bruce Powel Douglas, "UML Statecharts", Embedded Systems Programming, jan. 1999, available at http://www.ilogix.com/fs_prod.htm

[5] D. Harel, A. Naamad, "The STATEMATE Semantics of Statecharts", ACM Transactions on Software Engineering and Methodology, 5 (4), pp. 293–333, 1996

[6] D. Harel, E. Gery, "Executable Object Modeling with Statecharts", IEEE Computer, 30 (7): 31–42, Jul. 1997

[7] David Harel, Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, vol.8, no. 3, pp. 231–274, June 1987

[8] Object Management Group, OMG Unified Modeling Language Specification, ver. 1.3, June 1999 available on Internet at http://www.rational.com/

[9] Z. Manna, Mathematical Theory of Computation, McGraw-Hill, 1974

[10] Michael Phillipsen, Imperative Concurrent Object-Oriented Languages, Technical Report TR-95-049, International Computer Science Institute, Berkeley, Aug. 1995

[11] Marian Scuturici, Dan Mircea Suciu, Mihaela Scuturici, Iulian Ober, Specification of active objects behavior using statecharts, Studia Universitatis "Babes Bolyai", Informatica, Vol. XLII, no. 1, pp. 19–30, 1997

[12] Dan Mircea Suciu, Reuse Anomaly in Object-Oriented Concurrent Programming, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLII, no. 2, pp. 74–89, 1997

[13] Dan Mircea Suciu, Extending Statecharts for Concurrent Objects Modeling, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLIV, No. 1, pp. 37–44, 1999

[14] Dan Mircea Suciu, Using Scalable Statecharts for Active Objects Internal Concurrency Modeling, Studia Universitatis "Babes-Bolyai", Informatica, Vol. XLV, No. 2, pp. 67–76, 2000

Department of Computer Science, "Babeş-Bolyai" University, 1 M. Kogălniceanu St., RO-3400 Cluj-Napoca, Romania

E-mail address: tzutzu@cs.ubbcluj.ro