International Journal of Engineering and Technology Vol. 1, No. 2, June, 2009
1793-8236

Automatic Code Generation FromUML Statechart

Abdeslam Jakimi and Mohammed Elkoutbi

***Abstract*— Object-oriented software development has been widely accepted for its seamless transition property between development phases. In most of the object-oriented approaches for software design, including the use of the standard Unified Modeling Language, different specifications based on multiple models are obtained for describing the different aspects of a system. However, software industry still cannot provide satisfactory solutions to fill the gap between high-level modeling language and programming languages. The emergence of UML as a standard for modelling systems has encouraged the use of automated software tools that facilitate the development process from analysis through coding. In this paper, an approach is proposed to tackle this gap problem. Through mapping between UML and Java, we are able to generate low-level Java code directly from multiple UML statecharts.**

***Index Terms*— Unified Modeling Language, Statecharts, code generation, Java.**

I. INTRODUCTION

Object-oriented software development matured significantly during the past ten years. The Unified Modeling Language (UML) [1, 2, 3, 4] is an expressive language that can be used for problem conceptualization, software system specification as well as implementation. The emergence of the UML as an industry standard for modelling systems has encouraged the use of automated software tools that facilitate the development process from analysis through coding. It covers a wide range of issues from use cases and scenarios to state behavior and operation declarations.

In UML based object-oriented design, behavioral modeling aims at describing the behavior of objects using state machines. A state machine is a behavior that specifies the sequence of states an object goes through during its lifetime in response to events [1]. The UML statechart diagram visualizes a state machine. It contains states, transitions, events and actions. Statechart diagram addresses the dynamic view of a system. It is especially important in modeling the behavior of a class and emphasizes the event-ordered behavior of an object, which is particularly useful in modeling reactive systems. It focuses on changing states of a class driven by events. The semantics and notations used in UML statecharts mainly follow Harel's statecharts [5] with extensions to make them object-oriented [4].

A. Jakimi, , M. Elkoutbi are with ENSIAS, Labo SI2M, university Mohammed V, Rabat, Morrocco. Address: FPE, B..P 512, Boutalamine, Errachidia, Morocco.

This paper suggests an approach for requirements engineering that is based on the UML. In this paper, we suggest to generate the code from the UML statecharts. We have been working on implementing the dynamic behavior of an object-oriented system. Some of the results of our research, which includes a limited treatment of sequence diagrams, have already been published [6]. The present study focuses on implementing statecharts in general and addresses all the important components of statecharts.

Our work proposes a new approach to narrow the gap between UML behavior and an implemented system. The narrowing of gap is achieved by generating low-level Java code from statecharts. The code generation is achieved by creating a mapping between UML statecharts and Java programming language.

The remainder of this paper is organized as follows. In the next section, we present a general idea of the UML Statecharts relevant to our work. Section 3 provides background about various approaches to implement UML behavior. In section 4, we explain our automatic code generation approach to implement statecharts. Section 5 gives an illustration of the tools have been used in this work. Section 6 gives related work and discussion of approach. Finally, in section 7, we summarize and conclude.

II. STATECHARTS IN UML

Object oriented analysis and design methods offer a good framework for behavior [5, 7, 8, 9]. In our work, we adopted the UML, which is a unified notation for object oriented analysis and design.

A UML statechart (state machine) describes the dynamics of a model element as it changes its internal state as the reaction of receiving some external stimuli. UML statecharts can describe the behavior of a classifier (a class) or a behavioral feature (a method of a class).

A state machine is a graph of states and transitions that describes the response of an object to the receipts of events. State machines are used for specifying the full dynamic behavior of a single class of objects. The diagrammatic presentation of a state machine is a statechart diagram. A statechart attached to a class specifies all behavioral aspects of the objects in that class. Figure 1 shows a sample statechart diagram.

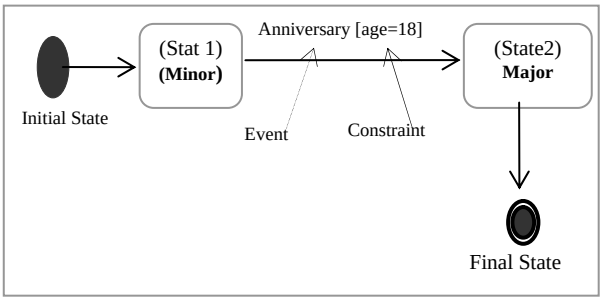

Figure 1. A statechart diagram

Each state models a period of time during the life of an object during which it satisfies certain conditions, performs some action, or waits for some event. A state becomes active when it is entered as a result of some transition and becomes inactive if it is exited as a result of a transition. A transition is a directed relationship between a source state, and a target state indicating that an instance in the source state will enter the target state and performs certain actions when a specified event occurs provided that certain specified conditions are satisfied [1].

III. MAPPING UML BEHAVIOR

Model-system gap is inevitable. This is so because it is normal for one to model from high abstraction level to low abstraction level. In order to narrow the gap, it is necessary to add more detailed information to the UML diagrams. Another reason to the gap existence is that one tends to take the forward engineering approach that is, starting from the problem perspective towards that of implementation

The emergence of UML as a standard for modeling systems has encouraged the use of automated software tools [10, 11, 12, 13] that facilitate the development process from analysis through coding.

There are two major approaches used for object-oriented model based code generation, namely structural and behavioral. The structural approach is based on using models of object structure (static relationships). It generates code frames (such as class interface specifications) from models of static relationships among objects. Class diagrams concepts can be implemented in a programming language supporting concepts like classes and objects, composition and inheritance.

In contrast to the static structural diagram, the main problems in generating Java [14] code covering system behavior are that UML does not have a unified behavioral diagram and many concepts from these diagrams are not supported by Java. As a result, the mapping from the behaviour diagrams to Java code is not as smooth or direct as that in static structural diagram. We will tackle these problems by using each diagram to generate part of the code.

Based on the partial models of object dynamics, developers then explicitly program object behavior and communications in the target language. Many works [15, 16, 17, 18] generate limited skeleton code from such models. The main drawback of this approach is that there is no code generation for object behavior and thus the code generated is not complete.

The pricipal goal of this research is to automatically generate implementation code from the UML statecharts in an object-oriented programming language such as Java. Our code generation approach and tool will help in bridging the gap between the design and development phase and will support the developers in the software development process.

IV. CODE GENERATION FROM STATECHARTS

The generation of Java code from UML models has been met with some degree of success. In addition to the generation of skeleton class code from class diagram, Java codes have been generated from the statechart, the sequence diagram and the component diagram. However, generation of Java source code from all UML diagrams is not yet achievable. And only the first level of UML behavior is handled in the code generation, based on a set of derived behavior translation rules

The UML behavior diagrams include many concepts that are not present in most popular programming languages, like C++ or Java. This means there is not a one-to-one mapping between a UML behavior and its implementation.

A model enables the modeler to work at a higher level of abstraction. The progression from the model to an implemented system is not truly a seamless transition, mainly due to a gap. A model-system gap exists primarily due to the different levels of abstraction.

UML is a modeling language, which consists of semantics and graphical notation. For every element of its graphical notation there is a specification that provides a textual statement of syntax and semantics. Implementing the semantics correctly is a challenging task, as the programming languages do not directly support them. The object-oriented programming languages do not deal with abstract behavior.

This section describes how to generate code from the state diagram. The mapping from the behavior diagrams to Java code is not as smooth or direct as that in static structural diagram.

A. Main Ideas

The following steps will generate code from statechart of an object, including how each element in a statechart will be transformed into source code.

Our approach uses the pre- and post-conditions of the underlying class diagram to generate state names [7]. Given an unlabeled StateD, its state names are identified via the post-conditions of outgoing and the preconditions of incoming events. Note that the events in a StateD correspond to the methods of its underlying class.

Table 1 summarizes the transformation rules for statecharts.

TABLE I: UML TO JAVA TRANSFORMATION

UML	**Java code**
Class	Class
Attribute	Attribute
Operation	Method
State	Scalar variable
Event	Method
Action	Method
Entry/ Exit Actions	Method

B. Code Generation

The UML behavior diagrams include many concepts that are not present in most popular programming languages, like C++ or Java, e.g. events, states, history states etc. This means there is not a one-to-one mapping between a statechart and its implementation.

We illustrate our approach (Mapping statechart to Java code) using an example for simple statechart (Figure 2) with the declarations.

*Attributes of the initial state (state A): a1=0, a2 = 1, a3 = 2 and a4= 3;

* Attributes of the state finale (state B): a1=1 et a2= 2, a3=3 and a4=4.

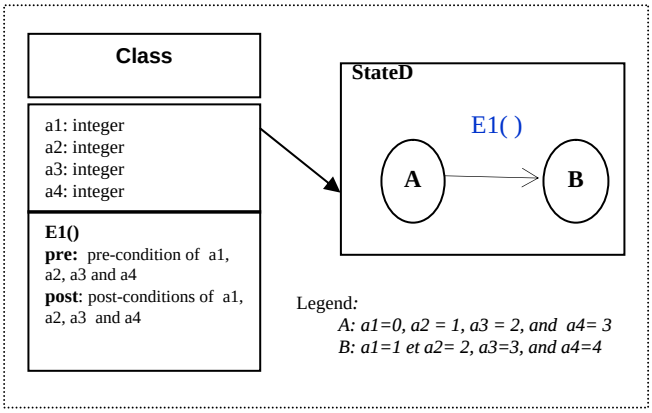

Figure 2. Statecharts of Class N

Consequently, after execution of the transition (E1) since state A towards B, we see a change on the level of the attributes.

Following is the part of Java code for the statechart of Figure 2.

```
class N {      // context class
        private
        final static int a1, a2, a3, a4;      // object attributes
       N ( ) {
              a1=0;  // initial state (State A)
              a2=1;
              a3=2;
              a4=3;
       }
       public void E1() {
                          a1=1;  // (State B)
                          a2=2;
                          a3=3;
                          a4=4;
       }
}
```

V. Tool Support

In this section we are going to present briefly the tool we have developed. This tool reads a UML model, stored in XMI format, and generates Java code. To generate the code from UML behavior and explain our automatic code generating system, we have used the Eclipse environment, the TogetherJ [19] plug-in for UML modeling and the application programming interface (API) JDOM for XML manipulation. Figure 3 gives a picture of how these tools have been used in this work.

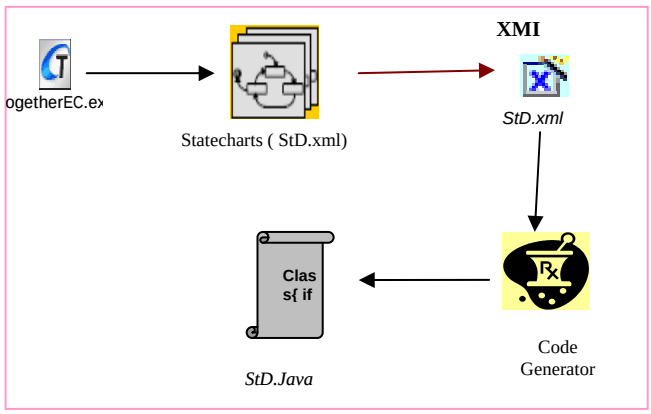

Figure 3. Tool support for code generation from UML behavior

Eclipse has been chosen because of its modular integrated environment of development (IDE). Many modules (plug-ins) are provided by Eclipse and it is very easy to add others developed either by the Eclipse community or by software companies. We used the plug-in for UML diagrams (from Together) which makes it possible for us to create use case, sequence and statecharts diagrams.

statecharts are first acquired throw the UML diagram plug-in, and then there are transformed in form XML files. This XML file can also be imported via the UML diagram plug-in for purposes of visualization and annotation. Finally we develop a code generator for automatic Java code generation from UML statecharts.

VI. Related Work

In this section, we first review some related work in the area of code generation from UML statechart.

Code generation from UML is supported by a variety of methods and tools. Metz et al [20] proposed an approach to implement statechart diagrams based on switch statement [21]. States are represented as constant attributes, events and actions as methods. Douglass [21] proposed the State Table Pattern to implement the statechart diagrams. States and transitions are modeled as classes. Pinter [18] proposed an extension to QHsm called Extended Quantum Hierarchical state machine (EQHsm). They proposed support for actions on transition, concurrency and history state. Kohler et al. [22] presented an approach for code generation from statecharts. Their approach adapts the idea generic array based state-table but uses object structure to represent state-table at runtime. They use objects to represent states of a statechart and attributes to hold the entry and exit actions. Knapp and Merz [23] described a set of tools called Hugo for the code generation of UML statecharts. A generic set of Java classes provides a standard runtime component state for UML statecharts. Every state of a statechart is represented by a separate object, which provides methods for activation, deactivation, initialization and event handling. Events, guards and actions are also implemented as classes.

VII. Conclusions

A new method has been proposed to implement the dynamic behaviour of an application. We have proposed in this paper an UML-based code generation approach. This approach helps developers to transit from the design to implementation phase and to shorten the software development cycle.

Our approach is an object-oriented approach and in the present study we have used Java as the target language. However our approach is general so it can be used to generate the low level code in other object-oriented languages. And the future works of this research include the following areas: optimize generated code, develop an efficient approach to use UML to model the internal working algorithm of the method of objects and the reverse engineering.

References

[1] G. Booch, J. Rumbaugh and I. Jacobson. Unified Modeling Language User Guide. *Addison Wesley*, 1999.
[2] J. Rumbaugh, I. Jacobson and G. Booch. Unified Modeling Language Reference Manual. *Addison Wesley*, 1999.
[3] I. Jacobson, G. Booch and J. Rumbaugh. The Unified Software Development Process, Addison-Wesley, 1999.
[4] Object Management OMG. Uinified modeling language specification version 2.0: Infrastructure. *Technical Report* ptc/03-09-15, OMG, 2003.
[5] D. Harel, "Statecharts: A visual formalism for complex systems", *Science of Computer Programming, vol. 8, no. 3, pp 231-274, Jun. 1987.*
[6] A. Jakimi, M. El Koutbi, " An Object-Oriented Approach to UML Scenarios Engineering and Code Generation" *International Journal of Computer Theory and Engineering (IJCTE*), VOL.1 No 1 April 2009, p38-45
[7] M. Elkoutbi; I. Khriss; and R.K. Keller. "Automated Prototyping of User Interfaces Based on UML Scenarios". *Journal of Automated Software Engineering, 13, 5-40, January 2006.*
[8] K. Jensen., Coloured Petri Nets, Basic concepts, Analysis methods and Pratical Use, *Springer*, 1995.
[9] designCPN: version 4, Meta Software Corp. <http://www.daimi.aau.dk/designCPN>.
[10] Gentleware AG, Poseidon for UML, http://www.gentleware.com
[11] MicroTOOL, objectiF, http://www.microtool.de /objectif
[12] No Magic Inc. MagicDraw, http://www.magicdraw.com
[13] I-Logix Inc., Rhapsody, ttp://www.ilogix.com.
[14] Sun Microsystems Inc., Java Technology, http://java.sun.com
[15] J. Ali, and J. Tanaka, "Converting Statecharts into Java Code", *Proc. Fourth World Conf.on Integrated Design and Process Technology (IDPT'99)*, Dallas, Texas, USA, 2000.
[16] I. A. Niaz and Jiro Tanaka, An Object-Oriented Approach To Generate Java Code From UML Statecharts, *International Journal of Computer & Information Science*, Vol. 6, No. 2, June 2005
[17] I. Majzik, – J. Javorsky –, A. Patricza., Concurrent Error Detection of Program Execution Based on Statechart Specification, *Proc. 10th European Workshop on Dependable Computing*, 1999.
[18] G.Pintèr, "Code genration based on Statecharts", Oct. 1, 2003, Budapest, *Hungary, Vol. 47*, No. 3–4, PP. 187–204.
[19] Borland, "Together", www.borland.com /together
[20] P. Metz, J. O'Brien and W. Weber, "Code Generation Concepts for Statechart Diagrams of the UML v1.1", *Object Technology Group Conference*, Vienna, Austria, June 1999.
[21] B. P. Douglass, "Real Time UML – Developing Efficient Objects for Embedded Systems", Massachusetts: *Addison-Wesley*, 1998.
[22] H. J. Köhler, U. Nickel, J. Niere, and A. Zündorf, "Integrating UML Diagrams for Production Control Systems", in Proc. 22nd International Conf. on Software Engineering (ICSE 2000), Limerick, Ireland, 2000, pp. 241-251.
[23] A. Knapp and S. Merz, "Model Checking and Code Generation for UML State Machines and Collaborations", in Proc. 5th Workshop on Tools for System Design and Verification, Reisenburg, Germany, 2002, pp. 59-64.

Abdeslam Jakimi, received his Masters degree in software engineering in 2004. His current research interests include requirements engineering, user interface prototyping, design transformations, scenario engineering and code generation.
Phone: (00212)667250395.

Mohammed Elkoutbi is a professor at École Nationale Supérieure d'Informatique et d'Analyse des Systèmes in Agdal, Rabat, Morocco. His current research interests include requirements engineering, user interface prototyping and design, and formal methods in analysis and design. He earned a PhD in Computer Science from University of Montreal in 2000.