International Journal of Computer Theory and Engineering, Vol. 1, No. 1, April 2009
1793-8201

An Object-Oriented Approach to UML Scenarios Engineering and Code Generation

A. Jakimi and M. El Koutbi

Abstract—**Object-oriented software development matured significantly during the past ten years. The Unified Modeling Language (UML) is generally accepted as the de facto standard modeling notation for the analysis and design of the object oriented software systems. This language provides a suitable framework for scenario acquisition using use case diagrams and sequence or collaboration diagrams. In this paper, we suggest a requirement engineering process that composes UML scenarios for obtain a global description of a given service of the system and implementation code from the UML use case (service). We suggest four operators: sequential operator, concurrent operator, conditional operator and iteration operator to compose a set of scenarios that describe a use case of a given system. We developed algorithm and tool support that can automatically produce a global sequence diagram representing any way of composing them and to offer a code generation of sequence diagram resulting.**

Keywords—**UML, Sequence diagrams, Scenario engineering, Scenario composition, code generation.**

I. Introduction

Scenarios have been identified as an effective means for understanding requirements and for analyzing human computer interaction. A typical process for requirement engineering based on scenarios has two main tasks. The first task consists of generating from scenarios specifications that describe system behavior. The second task concerns scenario validation with users by simulation and prototyping. These tasks remain tedious activities as long as they are not supported by automated tools.

This paper suggests an approach for requirements engineering that is based on the Unified Modeling Language (UML) [1, 2, 3, 4] and high-level Petri nets. In this paper, we suggest to compose scenarios that describe a given system in a natural way based directly on sequence diagrams. The approach provides an iterative, four-step process with limited manual intervention for deriving a prototype from scenarios and for generating a formal specification of the system. As a first step in the process, the use case diagram of the system as defined by the UML is elaborated, and for each use case occurring in the diagram, scenarios are acquired in the form of UML sequence diagrams and can be enriched with user interface [5], time, security, etc… constraints [4]. In the second step, the use case diagram and all sequence diagrams are transformed into Hierarchical Colored Petri Nets (CPNs) [6, 7]. In step three, the sequence diagrams describing one particular use case are composed into one single sequence diagram, and the sequence diagrams obtained in this way are linked with the single sequence diagram derived from the use case diagram to form a global single sequence diagram capturing the behavior of the entire system. Finally, in step four, a system prototype and code is generated from the global single sequence diagram and can be embedded in a user interface (UI) builder environment for further refinement [5].

Section 2 of this paper gives a brief overview of the scenario aspects. Section 3 offers a general idea of the UML diagrams relevant to our work. Section 4 provides an overview of the iterative process that derives a formal specification and code generation for the system from use cases and scenarios. Section 5 gives an illustration of the tools have been used in this work. Section 6 gives related work and discussion of approach. Section 7 concludes the paper and provides an outlook into future work.

II. Scenario Aspects

Scenarios have been evolved according to several aspects, and their interpretation seems to depend on the context of use and the way in which they were acquired or generated. In a survey, Rolland [8] proposed a framework for the classification of scenarios according to four aspects: the form, contents, the goal and the cycle of development.

The form view deals with the expression mode of a scenario. Are scenarios formally or informally described, in a static, animated or interactive form?

The contents view concerns the kind of knowledge which is expressed in a scenario. Scenarios can, for instance, focus on the description of the system functionality or they can describe a broader view in which the functionality is embedded into a larger business process with various stakeholders and resources bound to it.

The purpose view is used to capture the role that a scenario is aiming to play in the requirements engineering process. Describing the functionality of a system, exploring design alternatives or explaining drawbacks or inefficiencies of a system are examples of roles that can be assigned to a scenario.

The lifecycle view considers scenarios as artefacts existing and evolving in time through the execution of operations during the requirements engineering process. Creation, refinement or deletion are examples of such operations.

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

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

III. USE CASES AND SCENARIOS IN UML

Object oriented analysis and design methods offer a good framework for scenarios. In our work, we adopted the Unified Modeling Language, which is a unified notation for object oriented analysis and design.

Scenarios and use cases have been used interchangeably in several works meaning partial descriptions. UML distinguishes between theses terms and gives them a more precise definition. A use case is a generic description of an entire transaction involving several objects of the system. A use case diagram (Usecase D) is more concerned with the interaction between the system and actors (objects outside the system that interact directly with it). It presents a collection of use cases and their corresponding external actors. A scenario shows a particular series of interactions among objects in a single execution of a use case of a system (execution instance of a use case). A scenario is defined as an instance of a given use case. Scenarios can be viewed in two different ways through sequence diagrams (Sequence Ds) or collaboration diagrams (CollDs). Both types of diagrams rely on the same underlying semantics. Conversion from one to the other is possible.

A. *Use case diagram*

Some authors [9, 10] and the UML reference manual agree that a use case is a high-level description of what the system is supposed to do, whose aim is to capture the system requirements. However, use cases have to be specified, that is, many particular cases of a use case can be described. In other words, if a use case represents a user interaction, many variants of this user interaction can be described.

The Usecase D in UML is concerned with the interaction between the system and external actors. One use case can call upon the services of another use case using some relations (includes, extends, uses, etc). An example of the include relation is given in Figure 1. This relation is represented by a directed dotted line and the label <<include>>. The direction of an include relation does not imply any order of execution.

Figure 1 shows three main use cases: Deposit, Withdraw and Balance (services of the ATM: Automatic Teller Machine) that call on the service of the use case Identify.

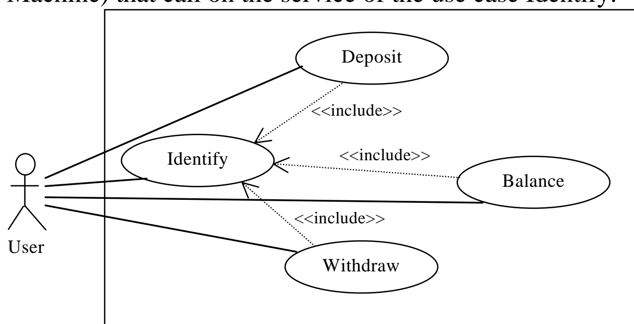

Fig. 1 ATM use case diagram.

B. *Sequence diagram*

We chose to use sequence diagrams (Sequence Ds) because of their simplicity and their wide use in different domains. A Sequence D shows interactions among a set of objects in temporal order, which is good for understanding timing and interaction issues. It depicts the objects by their lifelines and shows the messages they exchange in time sequence. However, it does not capture the associations among the objects. A Sequence D has two dimensions: the vertical dimension represents time, and the horizontal dimension represents the objects. Messages are shown as horizontal solid arrows from the lifeline of the object sender to the lifeline of the object receiver. A message may be guarded by a condition, annotated by iteration or concurrency information, and/or constrained by an expression. Each message can be labeled by a sequence number representing the nested procedural calling sequence throughout the scenario, and the message signature. Sequence numbers contain a list of sequence elements separated by dots. Each sequence element consists of a number of parts, such as: a compulsory number showing the sequential position of the message, and a letter indicating a concurrent thread (see messages (m3, m4 and m5 in figure 2), and an iteration indicator * (see message m2 in figure 2) indicating that several messages of the same form are sent sequentially to a single target or concurrently to a set of targets.

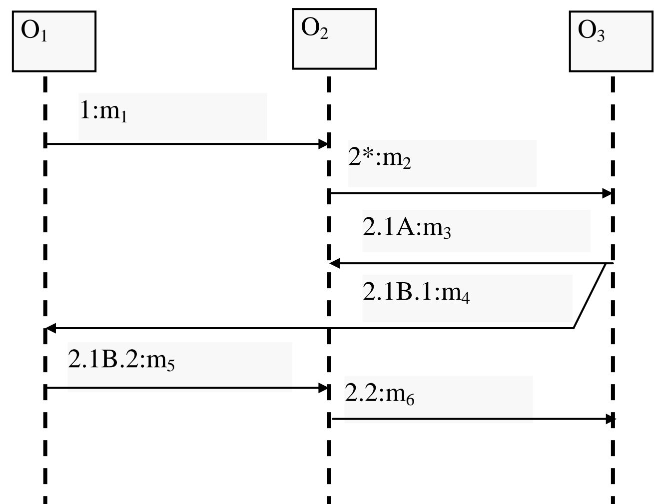

Fig. 2 Example of a Sequence D

IV. SCENARIO ENGINEERING

In this section, we give an overview of the iterative process that derives a formal specification for the system from scenarios and code generation. Figure 3 presents the sequence of activities involved in the proposed process.

In the *Scenario Acquisition* activity, the analyst elaborates the Usecase D, and for each use case, he or she elaborates several Sequence Ds corresponding to the scenarios of the use case at hand. The *Specification Building* activity consists of deriving CPNs from the acquired Usecase D and Sequence Ds and composes them to obtain a global CPN with three levels of hierarchy. In *Scenario composition* activity, the analyst then uses some composition operators as defined in section *C* to capture interaction at different levels: use cases, scenarios and messages [11]. This activity (*Scenario composition*) describes in detail the algorithm of merging several scenarios (of a given use case) in form of sequence diagrams into a global sequence diagram corresponding to the behavior of that use case. During *Code Generation* activity, an object-oriented approach has been proposed to

generate executable implementation code from UML Sequence D in an object-oriented programming language.

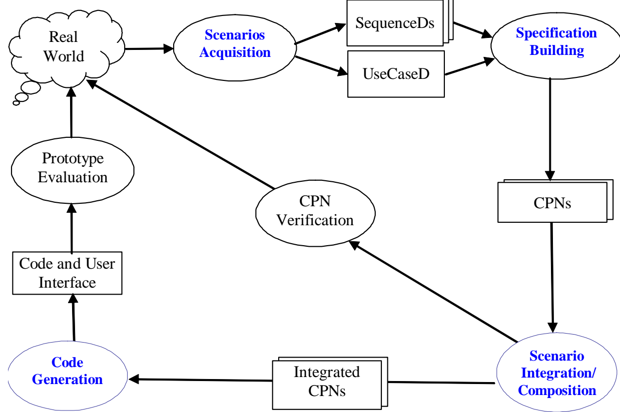

Fig. 3 Activities of the proposed process

The Composed CPNs serve as input to both the *CPN Verification* and the *System Prototype Generation* activities. During Prototype Evaluation, the generated prototype is executed and evaluated by the end user. In the CPN Verification activity, existing algorithms can be used to check behavioral properties [5, 12].

In the following subsections, we will focus on the four first activities this process: scenario acquisition, specification building, scenarios composition and code generation of resulting scenarios.

A. *Scenarios acquisition*

In this activity, the analyst elaborates the Usecase D capturing the system functionalities, and for each use case, he or she acquires the corresponding scenarios in form of Sequence Ds.

The extension of Sequence Ds is made to support time constraints in UML. Beyond the UML standard message constraints found in Sequence Ds, we define eight additional constraints to support time constraints. Note that the UML defines two standard constraints for messages: vote and broadcast. The vote constraint restricts a collection of return messages, and the broadcast constraint specifies that the constrained messages are not invoked in any particular order. UML constraints are generally put between braces. After studying some related work on scenarios that formalize time constraints [13, 14], we propose to extend the UML with the following message constraints: Time constraints and Security constraints

Time constraints

To model real-time constraints in early stages of development, we defined eight additional constraints which can be are summarized in Table I. The four first constraints are applied to a single message, while the remaining time constraints concern two or more messages. This range of time constraints gives a good modelling framework for several communication and distributed real-time systems.

TABLE I: REAL-TIME CONSTRAINTS ASSOCIATED TO MESSAGES IN A SEQUENCED

Constraint	Significance
m{At(a)}	The message m will occur at the time a
m{After(a)}	The message m will occur after the time a
m{Before(b)}	The message m will occur before the time b
m{Between(a, b)}	The message m will occur at the time a, and will takes at most b-a seconds
m_1{Starts(m_2)}	The messages m_1 and m_2 start at the same time
m_1{Ends(m_2)}	The messages m_1 and m_2 finish at the same time
m_1{Equals(m_2)}	The messages m_1 and m_2 start and finish at the same time
m_1{Meets(m_2)}	m_1 starts before the end of m_2

Security constraints

Today, security has become a major issue for information systems (e-business, e-trade, etc). It will be convenient to be able to define and represent these constraints in the step of requirement engineering. We were interested to the major security aspects: authenticity and confidentiality. Authenticity means the proof of identity and confidentiality relates to the privacy of information. Using UML, when a message is sent from a source to a target object, it can carry some information (message parameters). We aim to express that the exchange is private using some encryption algorithms (RSA, AES, 3DES, etc). This can be specified as a parameter of the constraint. The two constraints defined to model security aspects are given below (Table II):

TABLE II: SECURITY CONSTRAINTS

Constraint	Signification
m{Auth}	The message m must be signed by the sender object to proof its identity to the receiver object.
m{Crypt(algo)}	The message content (message parameters) must be encrypted using the algorithm (algo).

These defined constraints are very useful for the purpose of code generation from UML models.

B. *Specification Building (Scenario specification)*

This activity consists of deriving formal specifications from both the acquired Usecase D and interaction diagrams modeling scenarios (Sequence Ds). In our work, the resulting specification captures the behavior of the entire system in form of Petri nets. We consider separately specifications at scenario levels to capture hierarchy in the resulting specification. Indeed, this activity consists of deriving CPNs from the acquired Sequence Ds. We consider one level in building the system specification: the specification at scenario level.

For each scenario of a given use case, we first derive the CPN structure, and then the CPN semantic is built in association with the analyst. The CPN structure is obtained from the graph representing the sequence of messages in the scenario by adding places between each pair of sequential messages. Figure 4(a) gives an example of such graph derived from the scenario of the Figure 4, and Figure 4(b) shows the inserted places.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG

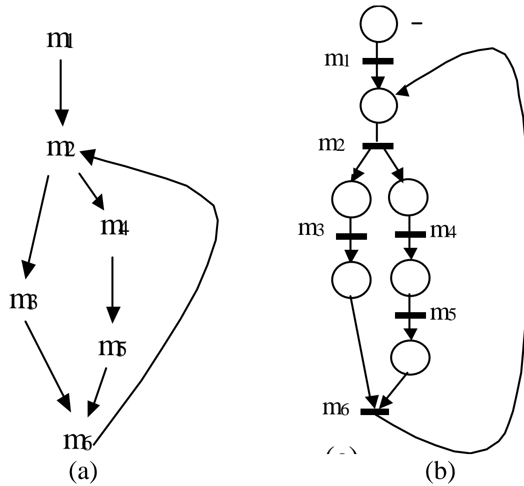

Fig. 4 (a) graph of messages, (b) Structure of CPN third level of hierarchy (corresponding to message interactions)

Note that more detail in Elkoutbi et al [6].

C. Composition of UML scenarios and code generation

In this section, we focus on the composition scenarios and code generation process. Figure 5 gives an overview of the scenarios fusion and the code generation from the result operation of the fusion (Sequence D resulting).

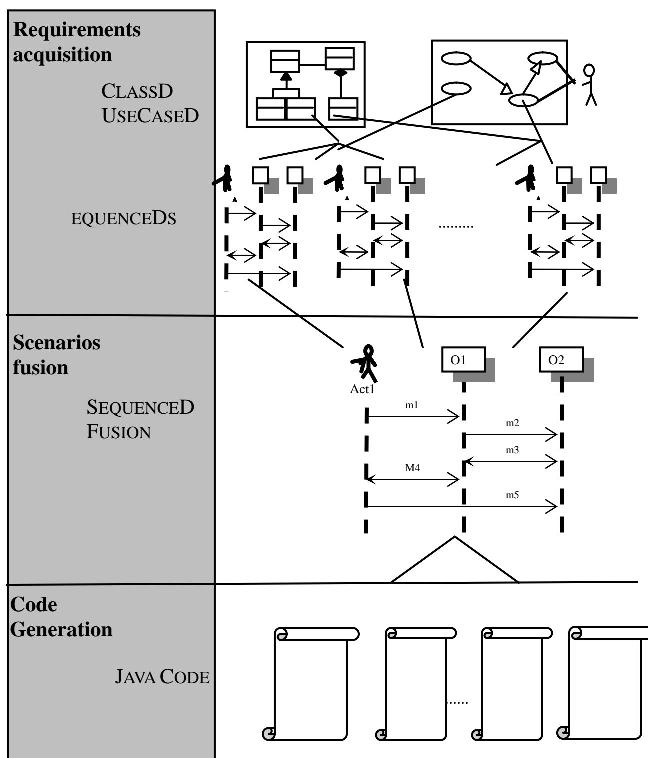

Fig. 5 Process of scenario fusion and code generation

Composition of UML scenarios

UML scenarios are considered as partial descriptions. To obtain a global description of a given service of the system or the description of the whole system, an operation of integration or composition is needed. The difficulty of scenarios composition comes in the fact that the scenarios are being described independently one to another.

The operation of integration looks like a generalization, where the analyst tries to identify and abstract some common parts in the system behaviour.

Composition constructs new behaviours from existing ones. This operation (composition) can be applied to different interaction objects like use cases, scenarios or messages. The difficulty of composition comes from the fact that interaction objects (use cases or scenarios specially) are being described independently one to each others.

Figure 5 gives an overview of the merging algorithm based on scenarios represented in the form of sequence diagrams.

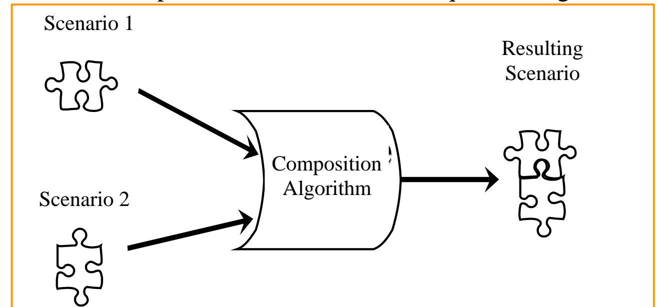

Fig. 6 Composing UML Scenarios

We consider four operators (;: sequential operator, ||: concurrent operator, * :iteration operator and if-else operator) to compose a set of interaction objects that describe a part of a given system. Our developed algorithms can automatically produce a global interaction object representing any way of composing scenarios. For example, we can compose three scenarios S1, S2 and S3 to obtain the resulting scenario Sr. Sr = (S1 ; S2 || S3)*[5], means to compose S1 and S2 sequentially, the obtained scenario will be composed concurrently with S3, then the obtained scenario will be iterated five times. Given a set of scenarios, our algorithms can produce any composing form of the given scenarios. The same operators can be applied to use cases. The scenarios composition is described in detail in [15].

The sequential operator. This operator is the simplest one to implement. The interactions between objects (or actor and objects) of two Sequence Ds are ordered in such a way that the interactions of first Sequence D (sd1) will occur before those of the second Sequence D (sd2). To compose sequentially two Sequence Ds, they need to have at least one common object. The principle of composing two scenarios using this operator is described as follows. Put initially the resulting Sequence D sdf equal to the first sequenced sd1, calculate the maximum sequence numbers (maxns) in sd1, add this number (maxns) to all sequence numbers in the second Sequence D sd2 before merging them in sdf and add to sdf objects that only belong to sd2.

The conditional operator. This operator allows us to define a choice between two possible scenarios when executing a service of the system. In this case, a condition [X] is allotted to the Sequence D sd1 and the complement [NonX] will allot the second Sequence D sd2. This operator gathers two scenarios into one Sequence D with keeping the conditions behind messages in the resulting Sequence D.

The concurrency operator. This operator allows us to define a competition between scenarios. This kind of composition can be used to describe the independence or the interleaving between two sequences of interactions. Two cases have to be considered. The first case, when the two

scenarios have some common objects. The second case relates to two scenarios having different objects acting for separate sub systems. we were interested by the first case which is more complex to implement than the second.

We need to review sequence numbers of the two Sequence Ds that will be merged by the concurrent operator (||).The principle of composing two scenarios using this operator is described as follows. Update all sequence numbers of sd1 by adding a letter, that is not yet used in sd1 or sd2, representing a new thread of execution, update all sequence numbers of sd2 by adding a letter, that is not yet used in sd1 or sd2, representing the second thread of execution and compose sequentially the updated Sequence Ds sd1 and sd2.

The iteration operator. This operator gives the possibility to iterate a given scenario many times. The condition that guards the iteration must be indicated *[cond-iteration] as we do it in an iterative message in a Sequence D. Sdr = sd1*[3] means that the scenario sd1 will be executed three times. The condition of iteration must be propagated globally to all messages of the scenario sd1. Suppose that sd1 contains two sequential messages m1 and m2. We note that sd1 = (1:m1 ; 2:m2). If we propagate the iterative condition directly to all messages of the scenario sd1, we will obtain the resulting scenario sdr that is equal to (*[3]1:m1 ; *[3]2:m2). This means that the message m1 will be iterated three times then the message m2 will do the same. This is naturally different of what we want sdr = *[3](1:m1 ; 2:m2). To solve this problem, we have considered that the scenario sd1 is represented by one abstract message m sent by the first object of the scenario to itself and all concrete messages will be viewed as are refinement of this message m. Thus, sd1 can be seen as equal to one message sd1 = 1:m and this message is refined with 1.1:m1 and 1.2:m2 (1:m = 1.1:m1; 1.2:m2). The resulting scenario sdr can be seen as equal to *[3] m which is equal to *[3](1.1:m1; 1.2:m2).

Code generation

The emergence of UML as a standard for modeling systems has encouraged the use of automated software tools [16, 17, 18, 19, 20] 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 [21] 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.

Based on the partial models of object dynamics, developers then explicitly program object behavior and communications in the target language. Many works [22, 23, 24, 25] 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 final goal of this research is to automatically generate implementation code from the UML SequenceDs 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.

In the operation of code generation from resulting SequenceD that representing a use case of the system, it is necessary to identify the elementary operators of interaction between objects of the diagram.

The general syntax of message in SequenceD is given in the form:

*[CI] [CE] Name := Message (parameters): returned type
* : iteration.; [CI] : iteration condition;
[CE]: send message condition;
Name: name of the object returned

Just the once an interaction identified, the Java code corresponding will be generated according to the type of the interaction. The generation will have to take into all the operations in a scenario (sequence, condition, iteration or concurrence).

The example (figure 6) explains the code generation to part of a resulting Sequence D (service 1) comprising a concurrency interaction.

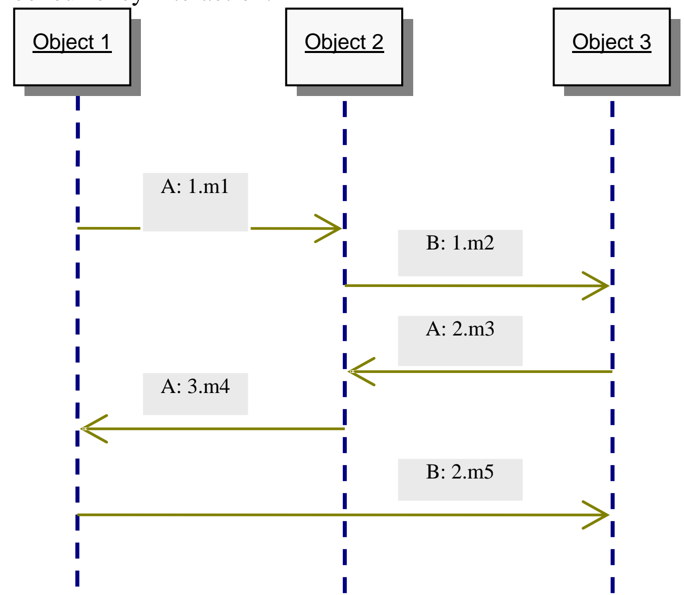

Fig. 7 Resulting SequenceD (service 1).

```
class mySystem {
Thread A = new Thread();
Thread B = new Thread();
public void run(){
If (this == A) service1-A();
If (this == B) service1-B();
}
public void service1-A(){
```


IACSIT
International Association of
Computer Science and Information Technology
WWW.IACSIT.ORG

```
            Objet2.m1();
        Objet2.m3();
            Objet1.m4();
                }
{   Public void service1-B()
            Objet3.m2();
        Objet3.m5();
                }
    Public void service1 () {
            A.start();
        B.start();
                }
                    }
```

V. Tool Support

To implement the four operators described above and explain our automatic code generating system, , we have used the Eclipse environment, the Together J [26] plug-in for UML modeling and the application programming interface (API) JDOM for XML manipulation. Figure 7 gives a picture of how these tools have been used in this work.

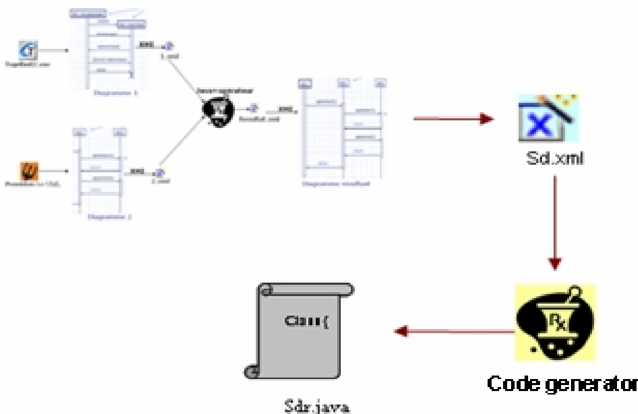

Fig. 8 Tool support for scenario composition and code generation

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 and sequence diagrams. Moreover, our composition algorithm can be used with any plug-in of UML diagrams as shown in figure 7.

Scenarios are first acquired throw the UML diagram plug-in, and then there are transformed in form XML files. These XML files serve as input to our developed composition operators that produce a merged XML file related to the resulting composed scenario. 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 Sequence D resulting (Usecase D or service).

VI. Related Work

In this section we will discuss some approaches for scenarios engineering and implementing UML diagrams.

A. Scenarios Engineering

the work of Koskimies et al. [27] aim to derive a set of specifications for the system or objects from scenarios, whereas all other approaches are interested in synthesizing one description or specification for a service of the system or the whole system. These various approaches differ in the notations that they support for the description of scenarios and specifications. Some, such as Dano [28] and the SCR method [29], use a tabular notation for capturing scenarios, whereas others, such as Whittle and Schumann [30] and ourselves, use SequenceDs or CollDs. In contrast, Deharnais [31] describe scenarios with relations. For the specification description, a variety of notations are used. For example, StateDs are supported by some approaches as by Elkoutbi [5], Koskimies et al. [27], whereas Petri nets are used by Elkoutbi and Keller [5].

One of the most prominent features of our approach is that it supports many kinds of scenarios (sequential, iterative and concurrent), whereas most of the other approaches can handle only sequential scenarios. Most of the related approaches are semi-automatic whereas our approach is fully automatic and offers either algorithmic or tool support.

B. Code Generation

Metz et al [32] proposed an approach to implement statechart diagrams based on switch statement [33]. States are represented as constant attributes, events and actions as methods. Douglass [33] proposed the State Table Pattern to implement the statechart diagrams. States and transitions are modeled as classes. Shlaer and Mellor [34] proposed an implementation of statecharts which is based on a linked list of transitions.

Chow et al. [32] developed two main steps in translating code from dynamic behaviour of the system. Translate an object's state diagram into Java code and Generate method body based on the pre/post condition of an operation and specify the order of language statements based on the message passing sequence in the interaction diagram.

VII. Conclusions

In this work, we have presented a new approach that produces automatically a global specification of the whole system in form of a three level hierarchical CPN. We have also implemented four operators for composing use cases and scenarios: sequential, conditional, iterative and concurrent. We have too automatically generated implementation code from the UML SequenceDs in an object-oriented programming language such as Java. However our approach is general so it can be used to generate the low level code in other object-oriented languages.

As future work, we prospect to study the possibility of code generation from scenarios in from of SequenceDs which will be a good plug-in to add. We plan to generate code from UML diagrams that describe dynamic and non-functional aspects of a system while remaining platform independent.

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] M. Elkoutbi, I. Khriss, R.K. Keller. "Automated Prototyping of User Interfaces Based on UML Scenarios". *The Automated Software Engineering Journal*, 13, 5-40, 2006.
[6] K. Jensen., Coloured Petri Nets, Basic concepts, Analysis methods and Pratical Use, Springer, 1995.
[7] designCPN: version 4, Meta Software Corp. <http://www.daimi.aau.dk/designCPN>.
[8] C. Rolland, C. Ben Achour, C. Cauvet, J. Ralyté, A. Sutcliffe, N.A.M. Maiden, M. Jarke, P. Haumer, K. Pohl, E. Dubois and P. Heymans. "A Proposal for a Scenario Classification Framework". *The Requirements Engineering Journal,* Volume 3, Number 1, 1998.
[9] I. Jacobson Use cases—yesterday, today, and tomorrow. Software Syst. Model. 2004, 3 210–220.
[10] J.M. Almendros-Jiménez and L. Iribarne. Describing Use-Case Relationships with Sequence Diagrams. Oxford Journals, the Computer Journal, 2007 50(1):116-128.
[11] A. Jakimi, A. Sabraoui, E. Badidi, , and Elkoutbi M., " Use Cases and Scenarios Engineering", *(Innovations'07) Proceedings of the IEEE 4th International Conference on Innovations in Information Technology*, November 18-20, 2007, Dubai, United Arab Emirates.
[12] 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.
[13] Dano B., Briand H. and Barbier F. : An Approach based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications, *In proceeding of the Third IEEE International Symposium on Requirements Engineering*, pp.54-64, Annapolis, 1997.
[14] Salah A., Dssouli R. and Lapalme G.. Intégration de scénarios temps-réel en automates temporisés, (*CFIP'2002*), Montréal, Canada, 2002.
[15] A. Jakimi and M. Elkutbi, Unified model of interaction: use cases and scenarios engineering; Accepted in *International Journal of Computer Science and Network Security (IJCSNS)*, VOL.8 No.12, December 2008.
[16] Gentleware AG, Poseidon for UML, http://www.gentleware.com
[17] MicroTOOL, objectiF, http://www.microtool.de /objectif
[18] No Magic Inc. MagicDraw, http://www.magicdraw.com
[19] I-Logix Inc., Rhapsody, ttp://www.ilogix.com.
[20] J. Ali, and J. Tanaka, "An Object Oriented Approach to Generate Executable Code from OMT-Based Dynamic Model", Journal of Integrated Design and Process Science, vol. 2, no. 4 1998, pp. 65-77.
[21] Sun Microsystems Inc., Java Technology, http://java.sun.com
[22] J. Ali, and J. Tanaka, "Converting Statecharts into Java Code", *in Proc. Fourth World Conf.on Integrated Design and Process Technology (IDPT'99)*, Dallas, Texas, USA, 2000.
[23] K.O. Chow, W. Jia, V.C.P. Chan and J. Cao, Modelbased generation of Java code, *Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000)*, Las Vegas, USA, 2000.
[24] Iftikhar Azim 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
[25] G.Pintèr, "Code genration based on Statecharts", Oct. 1, 2003, Budapest, Hungary, Vol. 47, No. 3–4, PP. 187–204 (2003)
[26] Borland, "Together", www.borland.com /together.
[27] Koskimies, K. , Systä, T.; Tuomi, J. and Mannisto, T. "Automatic Support for Modeling OO Software*, " IEEE Software*, Vol. 15 Num. 1, pp. 42-50.
[28] Dano, B.; Briand H. and Barbier F. An Approach based on the Concept of Use Case to Produce Dynamic Object-Oriented Specifications, *In proceeding of the Third IEEE International Symposium on Requirements Engineering*, Annapolis, 1997, pp.54-64.
[29] Heitmeyer C., Kirby J., Labaw B., and Bharadwaj R., "SCR*: A Toolset for Specifying and Analyzing Software Requirements", *Proc. of the 10th Annual Conference on Computer-Aided Verification*, (CAV'98), pp. 526-531, Vancouver, Canada, 1998.
[30] Whittle, J. and Schumann, J. "Generating Starechart Designs from Scenarios, " *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)*, Limerick, Ireland, ACM Press, pp. 314-323.
[31] Desharnais, J.; Frappier, M.; Khédri, R. and Mili, A. "Integration of Sequential Scenarios, " 1998, *IEEE Transactions on Software Engineering,* Vol. 24 Num. 9, pp. 695-708.
[32] P. Metz, J. O'Brien and W. Weber, "Code Generation Concepts for Statechart Diagrams of the UML v1.1*", Object Technology Group (OTG) Conference*, Vienna, Austria, June 1999.
[33] B. P. Douglass, "Real Time UML – Developing Efficient Objects for Embedded Systems", Massachusetts: Addison-Wesley, 1998.
[34] S. Shlaer and S. J. Mellor, "Object Lifecycles Modeling The World in States", Massachusetts: Addison-Wesley, 1992.

Abdeslam Jakimi, was born in morocco, in 1978. He received the diploma (D.E.S.A) in Informatics from Faculty of Sciences Rabat, Mohammed V University, Rabat, Morocco. His current research interests include requirements engineering, user interface prototyping and design transformations, scenario engineering. Email: ajakimi@yahoo.fr, Phone: (+212)67250395.

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.

IACSIT
International Association of Computer Science and Information Technology
WWW.IACSIT.ORG