Unified model of interaction: use cases and scenarios engineering

Abdeslam Jakimi and Mohammed El Koutbi

Labo SI2M, École Nationale Supérieure d'Informatique et d'Analyse des Systèmes (ENSIAS), Agdal, Rabat, Morocco

Summary
Scenarios and use cases have been used interchangeably in several works meaning partial descriptions. In this paper, we suggest a requirement engineering process that composes (merge) use cases/scenarios and yields a formal specification of the system in form of a high-level Petri net. Use cases and scenarios are acquired in form of diagrams as defined by the Unified Modeling Language (UML). These diagrams are composed and transformed into Petri net specifications and merged to obtain a global Petri net specification capturing the behavior of the entire system. From the global specification, a system prototype can be generated and embedded in a user interface builder environment for further refinement. Based on end user feedback, the use cases and the input scenarios may be iteratively refined. The result of the overall process is a specification consisting of a global Petri net, together with the generated and refined prototype. This paper discusses some activities of this process. The need of a unified model of interaction (UMI) is also discussed at the end of this paper.

Key words:
Use Case/Scenario engineering, Use Case/Scenario composition, Model transformation, UML, Unified Model of Interaction.

1. 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. 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 UI, 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). In step three, the CPNs describing one particular use case are integrated into one single CPN, and the CPNs obtained in this way are linked with the CPN derived from the use case diagram to form a global CPN capturing the behavior of the entire system. Finally, in step four, a system prototype is generated from the global CPN and can be embedded in a UI builder environment for further refinement [5, 6].

In our approach, we aim to model separately the use case and the scenario levels. We also want to keep track of scenarios after their integration or composition. Thus, we need a PN class that supports hierarchies as well as colors or objects to distinguish between scenarios in the resulting specification. We adopted Jensen's definition of CPN [7] which is widely accepted and supported by the designCPN tool [8] for editing, simulating, and verifying CPNs. Object PNs could also being used, but CPNs are largely sufficient for this work.

Section 2 of this paper gives a brief overview of the scenario aspects. Section 3 gives a general idea of the UML diagrams relevant to our work. In Section 4, the four activities leading from use cases / scenarios to formal specifications and executable prototypes are detailed. Section 5 discussed the need of unified model of interaction that gives a unique syntax to express. Finally, the section 6 concludes the paper and provides an outlook of future work.

2. 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 [9] proposed a framework for the classification of scenarios according to four aspects: the form, contents, the goal and the cycle of development (Figure1).

Manuscript received December 5, 2011
Manuscript revised December 20, 2011

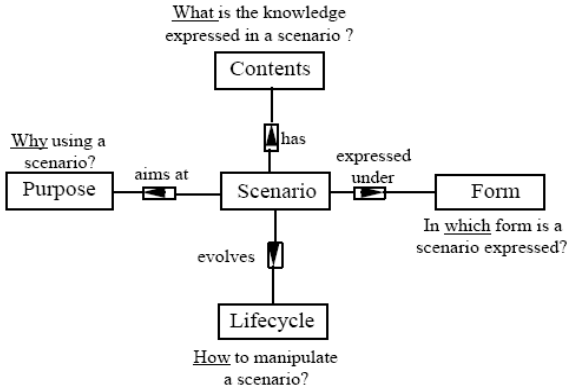

Fig. 1: Aspects of scenarios [5]

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.

3. 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 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 (SequenceDs) or collaboration diagrams (CollDs). Both types of diagrams rely on the same underlying semantics. Conversion from one to the other is possible .

3.1 Use case diagram

Some authors [10,11] 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 UsecaseD 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 2. 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. Other relations between use cases are detailed in [12, 13].

Figure 2 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.

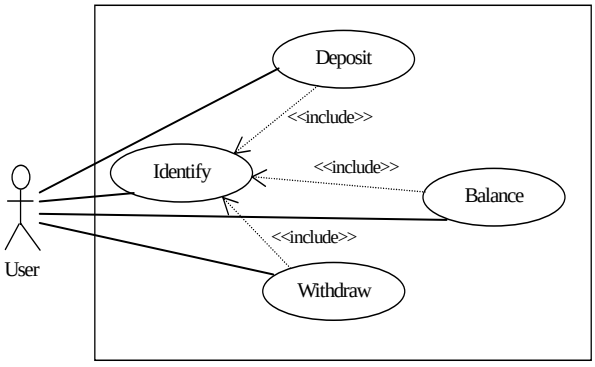

Fig. 2. ATM use case diagram.

3.2 Sequence diagram

We chose to use sequence diagrams (SequenceDs) because of their simplicity and their wide use in different domains. A SequenceD 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 SequenceD 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 3), 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.

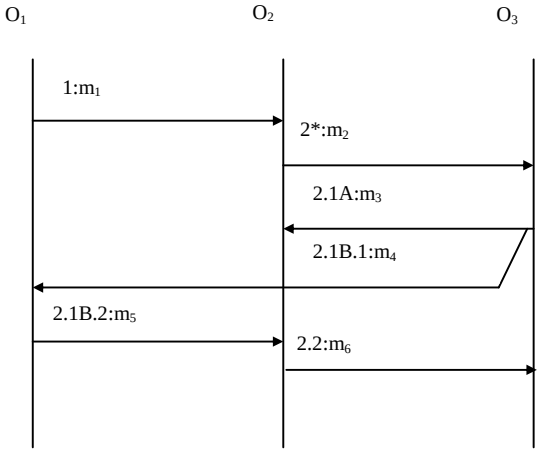

Fig. 3 : Example of a SequenceD

4. Use case and scenario engineering

In this section, we give an overview of the iterative process that derives a formal specification for the system from use cases and scenarios. Figure 4 presents the sequence of activities involved in the proposed process.

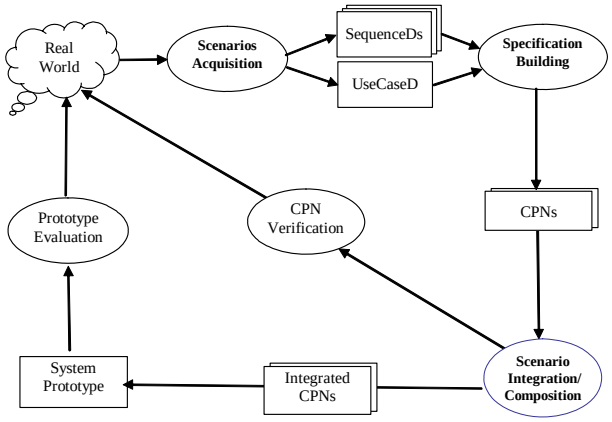

Fig. 4: Activities of the proposed process

In the Scenario Acquisition activity, the analyst elaborates the UsecaseD, and for each use case, he or she elaborates several SequenceDs corresponding to the scenarios of the use case at hand. The analyst then uses some composition operators as defined in section 4.3 to capture interaction at different levels: use cases, scenarios and messages [14]. The Specification Building activity consists of deriving CPNs from the acquired UsecaseD and SequenceDs and composes them to obtain a global CPN with three levels of hierarchy. 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 [15, 16].

In the following subsections, we will focus on the three first activities this process: scenario acquisition, specification building, and composition of UML scenarios.

4.1 Scenarios acquisition

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

Scenarios of a given use case are classified by type and ordered by frequency of use. We have considered two types of scenarios: normal scenarios, which are executed in normal situations, and scenarios of exception executed in case of errors and abnormal situations. Figures 5 and 6 give examples of SequenceDs corresponding to the scenarios regularIdentify and errorIdentify.

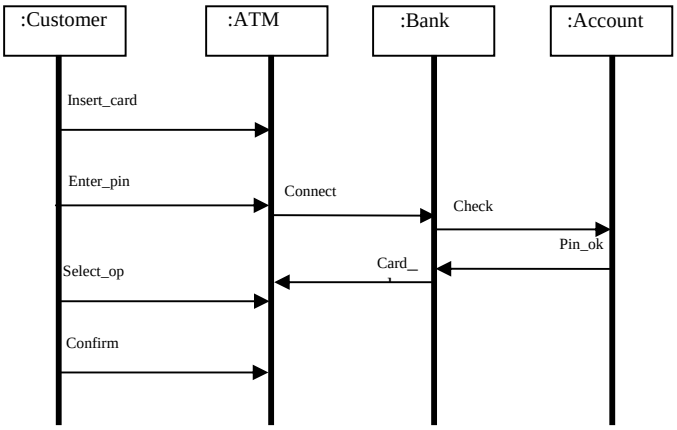

Fig. 5: Scenario regularIdentify of the use case Identify.

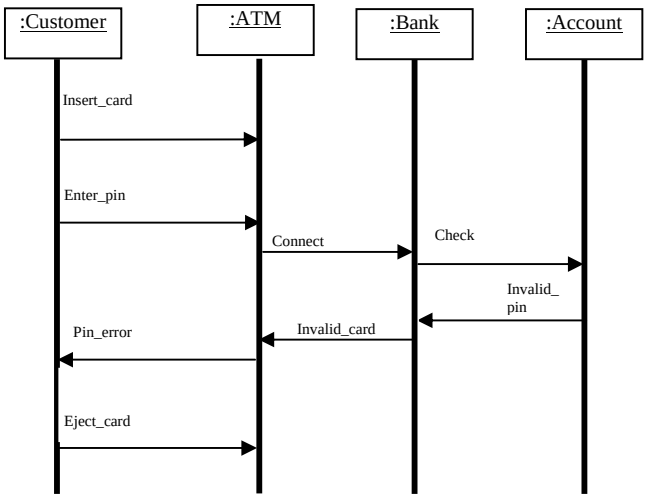

Fig. 6: Scenario errorIdentify of the use case Identify.

To obtain a global description of a given service (use case) of the system or the description of the whole system, an operation of integration or composition between use cases and/or between scenarios is needed.

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.

In this paper, 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.

To explain more how these operators act on interaction objects (section 4.3).

4.2 Specification Building

This activity consists of deriving a hierarchical CPNs from both the acquired and composed use cases and all of the SequenceDs. The obtained CPN will have three levels of hierarchy: the first level captures use cases interactions, the second level describes scenario interactions of the same use case and the third level shows interactions between messages within a given scenario. These derivations are explained below in the subsections Use case specification and scenario specification.

4.2.1. Use case specification

The CPN corresponding to the UsecaseD is derived by mapping use cases into transitions. A place Begin is always added to model the initial state of the system. After a use case execution, the system will return, via an back to its initial state for further use case executions. The place Begin may contain several tokens to model concurrent executions. Figure 7 depicts the CPN derived from the ATM system's UsecaseD (Figure 2) based on the following composition [Identify ; (Withdrawal | Deposit | Balance)*].

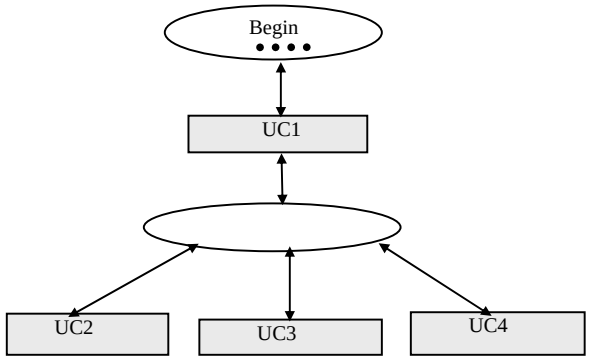

Fig. 7: CPN first level of hierarchy (corresponding to use case interactions).

UC1: Identify, UC2: Withdrawal , UC3: Deposit, UC4: Balance

4.2.2. Scenario specification.

Each use case (a transition in the CPN above) is expended in a CPN handling relations between its scenarios. Suppose that UC2 is described by three scenarios SC1, SC2 and SC3 composed as follow: (SC1; SC2) || SC3. The UC2 will be expended as shown in figure 8.

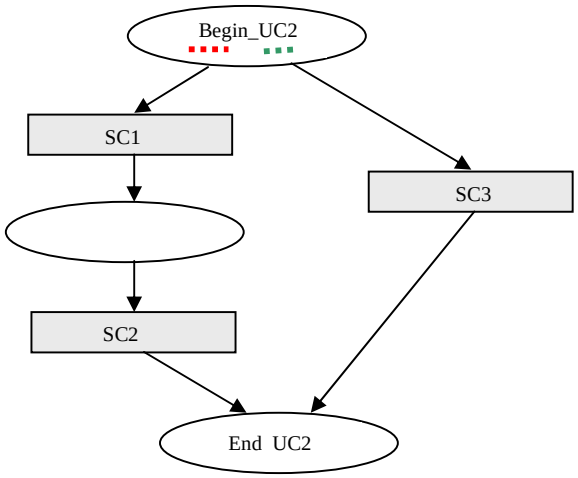

Fig. 8 : CPN second level of hierarchy (corresponding to scenario interactions).

The third level concerns scenarios. For each scenario of a given use case, we first derive the CPN structure, then the CPN semantic is built by the help of the analyst. The CPN structure is automatically obtained from the graph representing the sequence of messages in the scenario by adding places between each pair of sequential messages. Figure 9(a) gives an example of such graph derived from the scenario of Figure 3, and Figure 9(b) shows the inserted places.

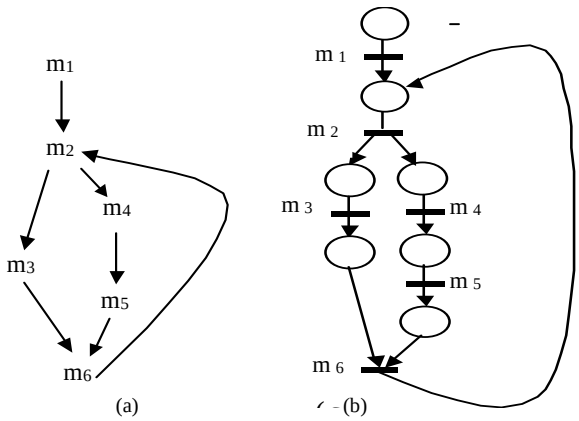

Fig. 9: (a) graph of messages, (b) Structure of CPN third level of hierarchy (corresponding to message interactions).

For the CPN semantic, the analyst can build an associated table of object states from the scenario by following the exchange of messages from top to bottom and identifying the changes in object states caused by the messages. This can serve to label places in the derived CPN. It can also been used to verification issues to check the coherence between scenarios.

To each scenario, we assign a distinct color to track it in the composed CPN. All scenarios of the same use case will have the same initial place which we call Begin_UC2 in figures 8. This place will contain several tokens with different colors of the linked scenarios : color1 for SC1 and color3 for SC3. At the end of SC1 transition, the token will change its color to color2 corresponding to the scenario SC2 and it can begin its execution.

4.3. 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.

Figure 10 gives an overview of the merging algorithm based on scenarios represented in the form of sequence diagrams.

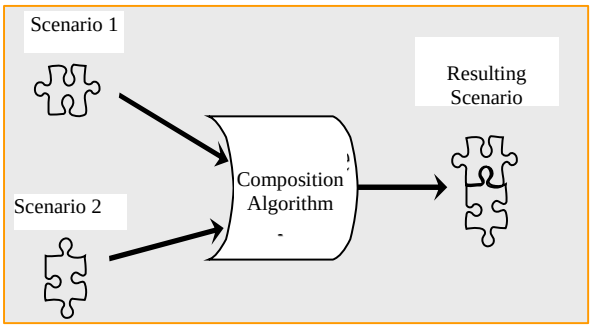

Fig. 10: Composing UML Scenarios.

In this section, we will describe the sequential, conditional, concurrency and iteration operators which defines a relation of precedence between two scenarios.

4.3.1 The sequential operator

This operator is the simplest one to implement. The interactions between objects (or actor and objects) of two SequenceDs are ordered in such a way that the interactions of first SequenceD (sd1) will occur before those of the second SequenceD (sd2). To compose sequentially two SequenceDs, 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 SequenceD 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 SequenceD sd2 before merging them in sdf.
- Add to sdf objects that only belong to sd2.

An example of composing sequentially two scenarios sd1 and sd2 is shown in Figure 11.

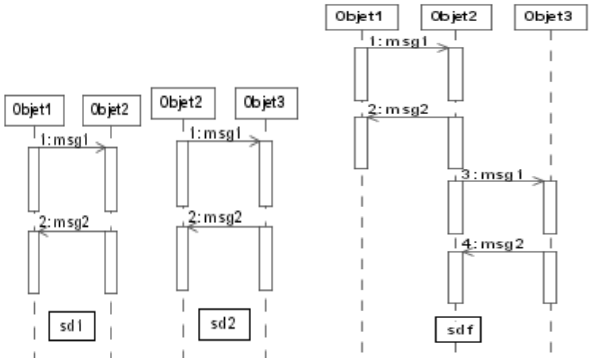

Fig. 11: (a) SequenceD sd1, (b) SequenceD sd2 and (c) Resulting SequenceD sdf = sd1 ; sd2.

4.3.2 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 SequenceD sd1 and the complement [NonX] will allot the second SequenceD sd2. This operator gathers two scenarios into one SequenceD with keeping the conditions behind messages in the resulting SequenceD. The figure (Figure 12) shows how we can merge two scenarios sd1 and sd2 with the conditional operator (sdie = sd1 ? sd2). Note that the sequence numbers of sd2 must be updated as in the case of the sequential operator.

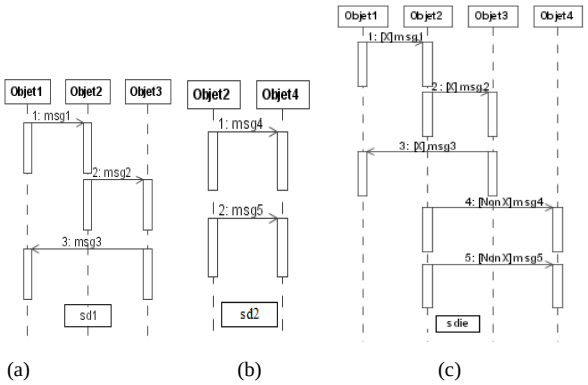

Fig. 12: (a) SequenceD sd1, (b) SequenceD sd2 et (c) Resulting SequenceD sdie = sd1 ? sd2.

4.3.3 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. In this work, 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 SequenceDs that will be merged by the concurrent operator (||):

- 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.
- Compose sequentially the updated SequenceDs sd1 and sd2.

Figure 13 shows an example of a concurrent composition of two scenarios in form of SequenceDs.

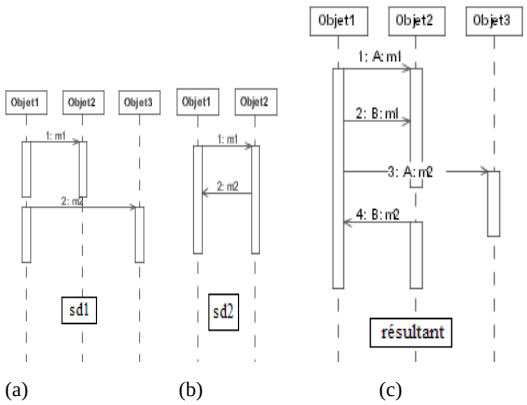

Fig. 13: (a) SequenceD sd1, (b) SequenceD sd2 et (c) Resulting SequenceD sdr = sd1 || sd2.

4.3.4 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 SequenceD. 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).

4.3.5 Tool support

To implement the four operators described above, we have used the Eclipse environment, the TogetherJ [17] plug-in for UML modeling and the application programming interface (API) JDOM for XML manipulation. Figure 14 gives a picture of how these tools have been used in this work.

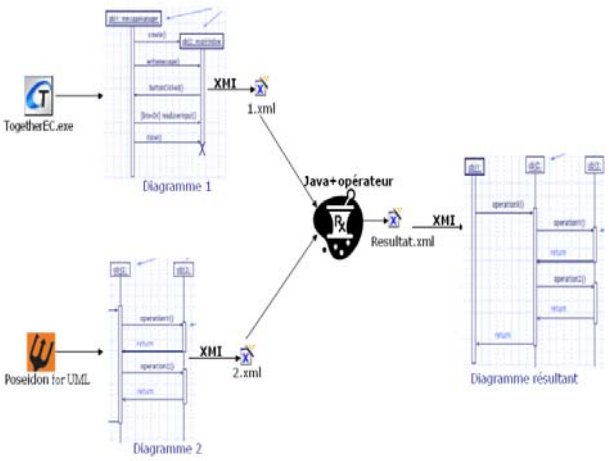

Fig 14. : Tool support for scenario composition.

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 14.

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.

5. Unified Model of Interaction

The UsecaseD captures services offered by the system and interactions between these services with the external actors. Interactions in a UsecaseD are modeled using some limited relations such as uses and extends. These relations give only a simplified view of interactions that can really exist between services given by a system. For example, in the case of the system of the Automatic Teller Machine (ATM), the use case Identification is used by three others use cases: Withdrawal, Deposit and Balance. Within the relation uses, interactions between the four services of the system are not precisely defined. Really after executing the use case Identification, the customer can repeatedly carry out one of the three other use cases (Deposit, Withdraw and Balance). Another example of limitation of expressiveness using use case relations, constraints as ‘it is forbidden to execute the Withdraw use case after the Deposit one; it is forbidden to

Withdraw more than three times' can not be now easily expressed.

These kinds of constraints are not actually supported in UML. More detailed relations are needed in the use case model [12]. We think that artifacts provided by the UML standard in sequence diagrams with some extensions can be used to model interactions between use cases. Thus, we can use the same interaction operators found in the sequence diagram: like sequence (;), condition (if), choice (|), iteration (*), and concurrence (||). For example, [Identification; (Withdrawal | Deposit | Balance)*] express naturally that the use case Identification is executed then it is followed by an iteration of one of three other use cases. Graphical notations can also be used to express easily these relations. The meta-model of interactions in sequence diagrams can be generalized to use cases, scenarios and messages. This will gives to developers a hierarchical simplified view of interactions at different levels of abstraction.

A Unified Model of Interaction (UMI) will help in better expressing interaction between usecases, sequence diagrams and messages using one kind of diagram. The UMI will be very useful during the step of code generation from UML models. The UMI will also be a great support to automate the operation of code from static and dynamic models (class, use case and scenario diagrams).

6. 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 discussed the need of unified model of interaction that gives a unique syntax to express interactions at different levels: use case, scenario and message.

As future work, we prospect to study the possibility of code generation from in form of web services usecases/scenarios 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] M. Elkoutbi, and R.K. Keller. " User Interface Prototyping based on UML Scenarios and High-level Petri Nets," Application and Theory of Petri Nets 2000 (Proc. of 21st Intl. Conf. on ATPN), Aarhus, Denmark, June 2000. Springer. LNCS 1825, pp. 166-186.

[7] K. Jensen., Coloured Petri Nets, Basic concepts, Analysis methods and Pratical Use, Springer, 1995.

[8] designCPN: version 4, Meta Software Corp. <http://www.daimi.aau.dk/designCPN>.

[9] 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.

[10] I. Jacobson Use cases—yesterday, today, and tomorrow. Software Syst. Model. 2004, 3 210–220.

[11] G. Génova and J. Lorens. The emperor's new use case. J. Object Technol. 2005, 4 81–94.

[12] P. Metz, J. O'Brien, W. Weber. Specifying use case interaction: clarifying extension points and rejoin points. J. Object Technology. 2004, 3 87–102.

[13] J.M. Almendros-Jiménez and L. Iribarne. Describing Use-Case Relationships with Sequence Diagrams. Oxford Journals, the Computer Journal, 2007 50(1):116-128.

[14] 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.

[15] M. Elkoutbi, "User Interface Engineering based on Prototyping and Formal Methods" PhD thesis, Université de Montréal", Canada, March 2000.

[16] A. Jakimi, A. Sabraoui, E. Badidi, A.Salah and M. El Koutbi, "Using UML Scenarios in B2B systems", in the proceedings of ICCCE'08, Malaysia, 2008.

[17] Borland, "Together", www.borland.com /together.

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.

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.