USING UML SCENARIOS IN B2B SYSTEMS

A. JAKIMI, A. SABRAOUI, E. BADIDI, A. SALAH AND M. EL KOUTBI

E.N.S.I.A.S, Laboratory SI2M, University Mohammed V-Souissi,
B.P. 713 Rabat, Morocco

elkoutbi@ensias.ma

***ABSTRACT*:** Scenarios has become a popular technique for requirements elicitation and specification building. Since scenarios capture only partial descriptions of the system behavior, an approach for scenario composition and/or integration is needed to produce more complete specifications. The Unified Modeling Language (UML), which has become a standard notation for object-oriented modeling, provides a suitable framework for scenario acquisition using Use Case diagrams and Sequence or Collaboration diagrams. In this paper, we suggest an algorithmic and tool support for composing and integrating scenarios that are represented in form of sequence diagrams. 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. In this paper, we suggest also to apply the scenario approach to B2B systems (Business to Business). We propose to develop B2B systems as a three activities process deriving formal specifications and code skeletons from UML scenarios. Activities of this proposed process are generally automatic and are supported by a set of developed algorithms and tools.

KEYWORDS : *Scenario engineering, Scenario composition, Scenario integration, Model transformation, UML.*

1. INTRODUCTION

Over the past years, scenarios have received significant attention and have been used for different purposes such as understanding requirements, human computer interaction analysis, specification generation, and object-oriented analysis and design. Notably, they have been identified as a promising technique for requirements engineering.

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 [1] 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.

The UML is an expressive language that can be used for problem conceptualization, software system specification as well as implementation. It covers a wide range of issues from use cases and scenarios to state behaviour and operation declarations. Scenarios describe partial views of the system. To obtain a global view, we need to define an approach to merge or to compose these partial views. We aim, in this paper, to provide an automatic support for this operation of scenarios composition. There are several related works in this field but most of them have addressed this operation using formal techniques (like Statecharts, Petri Nets, Z schemas, etc.). In this paper, we suggest to compose scenarios that describe a given system in a natural way based directly on sequence diagrams.

Section 2 of this paper gives a brief overview of the UML diagrams relevant for our work. Section 3 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 behaviour of that use case. Section 4 discusses the scenario approach applied to B2B systems. Finally, Section 5 provides some concluding remarks and points out to future work.

2. SCENARIOS IN UML

OO analysis and design methods offer a good framework for scenarios. In our work, we have adopted the Unified Modeling Language, which is a unified notation for OO analysis and design. Scenarios and use cases have been used interchangeably in several works meaning partial descriptions. UML [2] 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.

2.1 Use Case Diagram

The UsecaseD [3] 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 (include, extends, etc). An example of the include relation is given in Fig. 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. Figure 1 gives an example of UsecaseD.

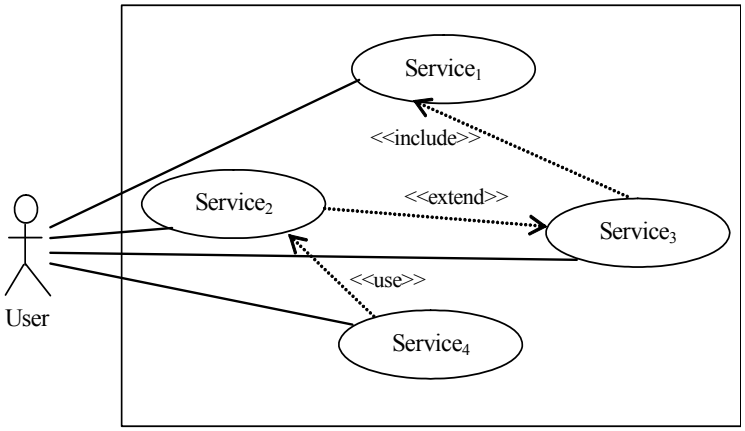


Fig. 1: Example of Use Case diagram.

2.2 Sequence Diagram

In this work, we chose to use sequence diagrams (SequenceDs) because of 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), and
- an iteration indicator * (see message m2) indicating that several messages of the same form are sent sequentially to a single target or concurrently to a set of targets.

In B2B systems, several information systems IS_i interacts between then to realize a scenario. In Fig. 2, objects are prefixed by the name of IS_i where they evolve.

2.3 Constraints in Sequence Diagram

To ease elicitation of non-functional requirements, some new constraints have been defined. In this work, we focus more on time and security requirements that are very important for B2B systems.

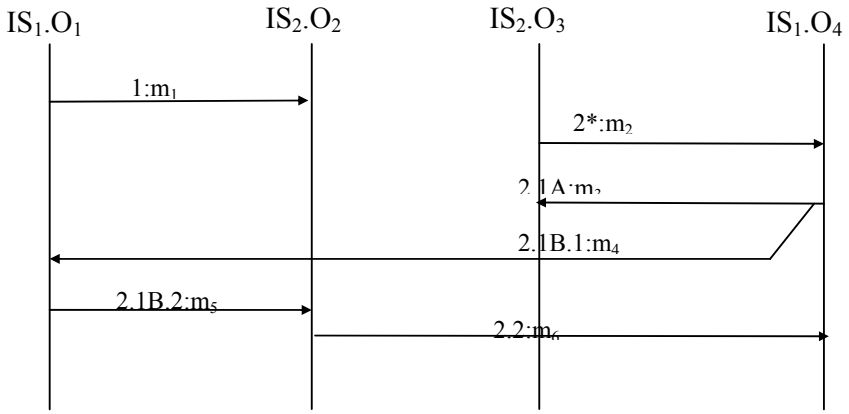


Fig. 2: Example of a SequenceD.

2.3.1 Time Constraints

To model real-time constraints in early stages of development, we defined eight additional constraints [4] which can be are summarized in Table 1. 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 1: 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

2.3.2 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 2: 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 [5, 6] from UML models.

3. COMPOSITION 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 operation of integration will be described in section 4.3. The difficulty of scenarios composition comes in the fact that the scenarios are being described independently one to another. Figure 3 gives an overview of the merging algorithm based on scenarios represented in the form of sequence diagrams.

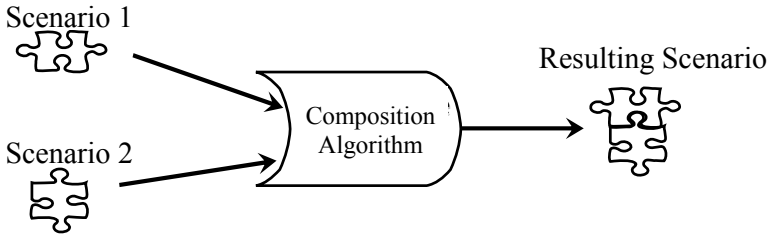


Fig. 3: Composing UML Scenarios.

In this paper, we consider four operators (**;**: sequential operator, ||: concurrent operator, ***** :iteration operator and if-else operator) to compose a set of scenarios that describe a use case of a given system. Our developed algorithms can automatically produce a global SequenceD representing any way of composing scenarios. For example, we can compose three scenarios S_1, S_2 and S_3 to obtain the resulting scenario S_r. $S_r = (S_1 ; S_2 \| S_3)^{*[5]}$, means to compose S_1 and S_2 sequentially, the obtained scenario will be composed concurrently

with S_3, 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.

We can also visualize the scenarios at different levels of detail: IS level where objects of an IS are hidden behind it, object level which is the detailed level, or any set of objects. The abstraction/refinement operator has been developed for this purpose.

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 (sd_1) will occur before those of the second SequenceD (sd_2). 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 sd_f equal to the first sequenced sd_1.
- Calculate the maximum sequence numbers (maxns) in sd_1.
- Add this number (maxns) to all sequence numbers in the second SequenceD sd_2 before merging them in sd_f.
- Add to sd_f objects that only belong to sd_2.

An example of composing sequentially two scenarios sd_1 and sd_2 is shown in Fig. 4.

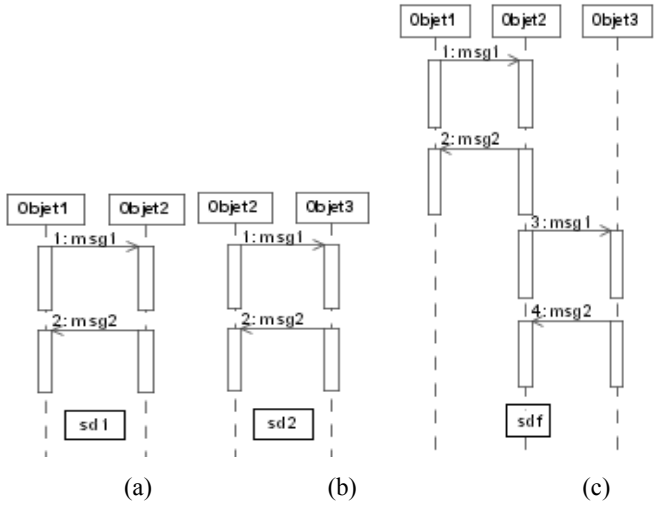


Fig. 4 : (a) SequenceD sd_1, (b) SequenceD sd_2 and (c) Resulting SequenceD $sd_f = sd_1 ; sd_2$.

3.2 The Alternative Operator

This operator acts as the sequential operator. All messages in the second scenario Sc_2 (*IF cond* Sc_1 *ELSE* sc_2) will be prefixed by the condition *cond*.

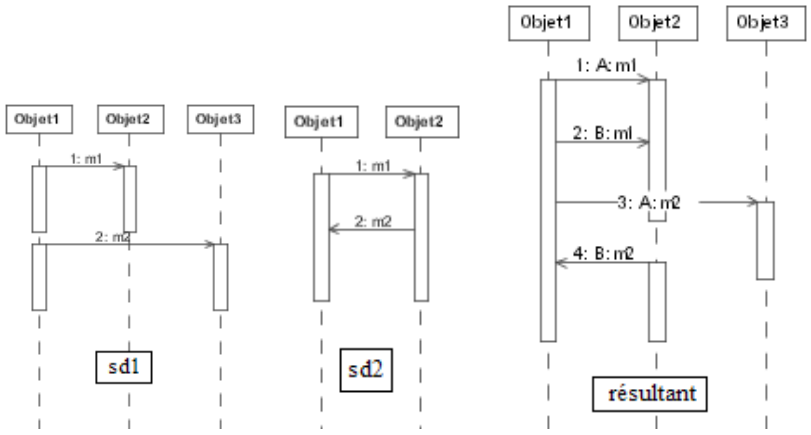
3.3 The Concurrency Operator

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 interaction. 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 approach, 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 sd_1 by adding a letter, that is not yet used in sd_1 or sd_2, representing a new thread of execution.
- Update all sequence numbers of sd_2 by adding a letter, that is not yet used in sd_1 or sd_2, representing the second thread of execution.
- Compose sequentially the updated SequenceDs sd_1 and sd_2.

Figure 5 shows an example of a concurrent composition of two scenarios in form of SequenceDs.



(a) (b) (c)

Fig. 5: (a) SequenceD sd_1, (b) SequenceD sd_2 and (c) Resulting SequenceD $sd_r = sd_1 \parallel sd_2$.

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 the iteration must be propagated globally to all messages of the scenario sd_1. Suppose that sd1 contains two sequential messages m1 and m2. We note that $sd_1 = (1{:}m_1 ; 2{:}m_2)$. If we propagate the iterative condition directly to all

messages of the scenario sd1, we will obtain the resulting scenario sd_r that is equal to ($^{*[3]}1{:}m_1$; $^{*[3]}2{:}m_2$). 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 $sd_r = {}^{*[3]}(1{:}m_1$; $2{:}m_2)$. To solve this problem, we have considered that the scenario sd_1 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, sd_1 can be seen as equal to one message $sd_1 = 1{:}m$ and this message is refined with $1.1{:}m_1$ and $1.2{:}m_2$ ($1{:}m = 1.1{:}m_1$; $1.2{:}m_2$). The resulting scenario sd_r can be seen as equal to $^{*[3]}$ m which is equal to $^{*[3]}(1.1{:}m_1$; $1.2{:}m_2)$.

3.5 The Abstraction/Refinement Operator

This operator offers the possibility to visualize a scenario at different levels of detail. Figure 6 shows an example where all objects of the IS2 is hidden.

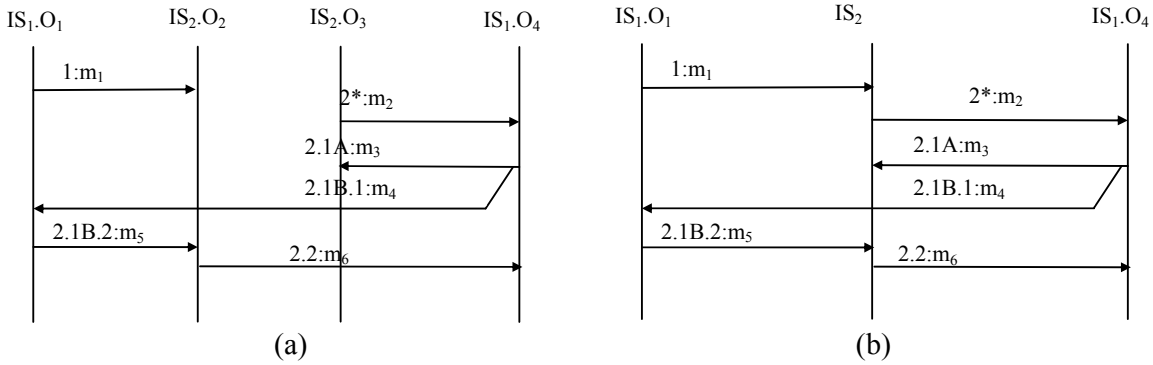


Fig. 6: (a) Detailed view, (b) Corresponding abstract view where objects of IS_2 were hidden.

3.6 Tool Support

To implement the four operators described above, we have used the Eclipse environment, the TogetherJ plug-in for UML modeling and the application programming interface (API) JDOM for XML manipulation. Figure 6 gives a picture of how these tools have been used in this work.

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 Fig. 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.

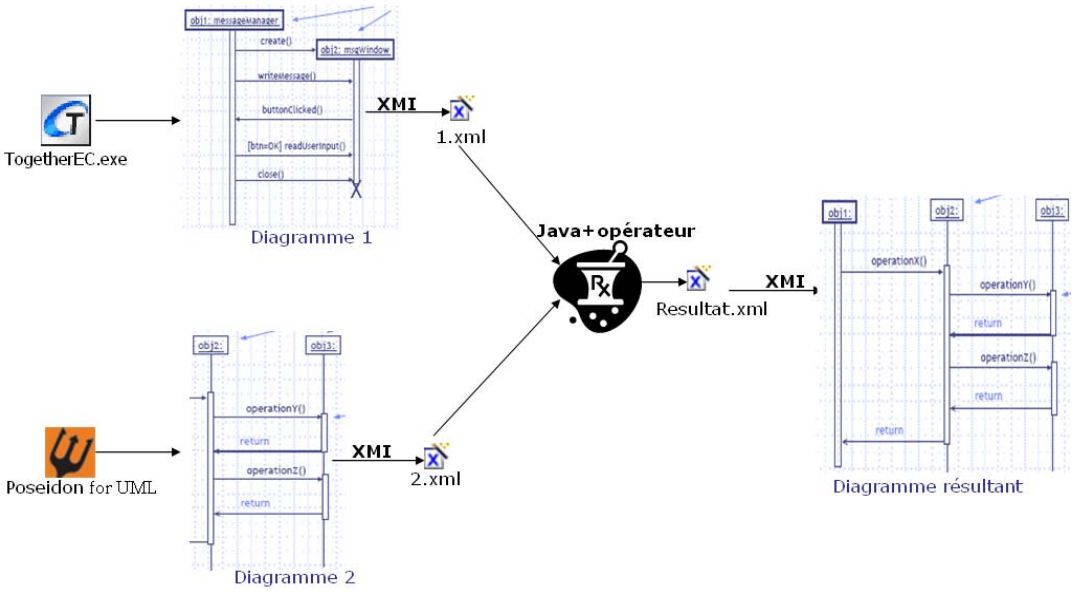


Fig. 7: Tool support for scenario composition.

4. THE SCENARIO APPROACH IN B2B SYSTEMS

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 8 presents the sequence of activities involved in the proposed process.

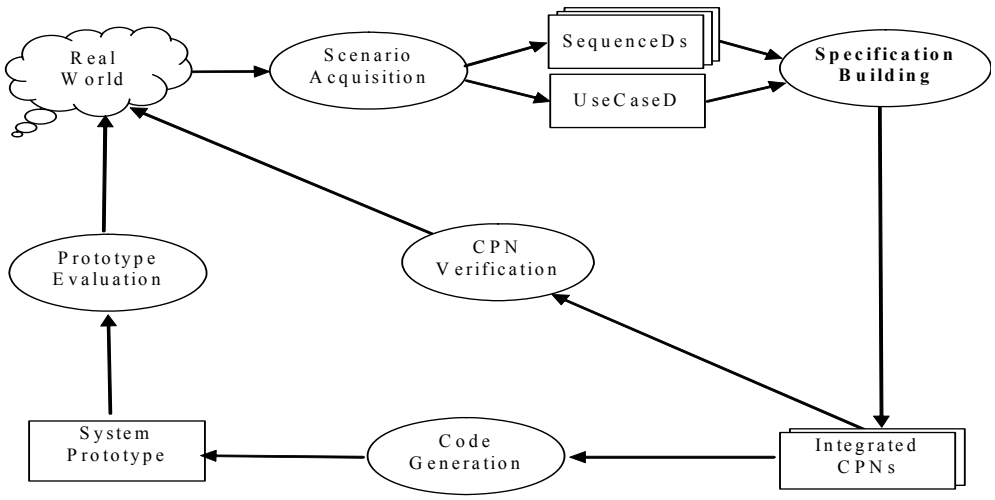


Fig. 8: Activities of the proposed process.

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 3.1 to capture interaction at different levels: use cases, scenarios and messages. The *Specification Building* activity consists on deriving CPNs from the acquired UsecaseD and SequenceDs and compose them to obtain a global CPN with three levels of hierarchy. The Composed CPNs serve as input to both *CPN Verification* and *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. In the following subsections, we will focus on the two first activities this process: scenario acquisition and specification building.

4.1 Scenario 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. 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 behaviors 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.

4.2 Specification Building

This activity consists on deriving 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.

Each use case (a transition in the CPN above) is expended in a CPN handling relations between its scenarios. Suppose that $Service_2$ is described by three scenarios Sc_1, Sc_2 and Sc_3 composed as follow: $(Sc_1; Sc_2) \parallel Sc_3$. The $Service_2$ will be expended as shown in Fig. 9.

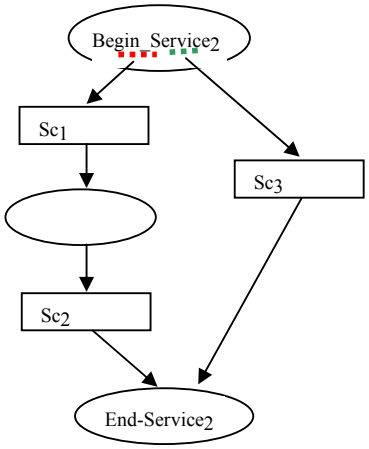
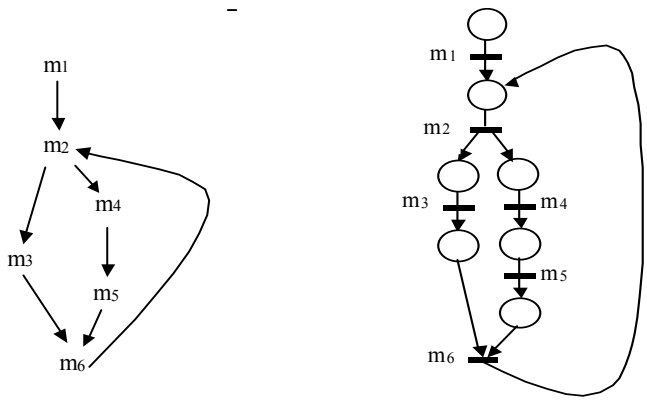


Fig. 9: CPN corresponding to scenario interactions.

The third level concerns scenarios. We first derive the CPN structure, the CPN semantic is added by the help of the analyst. The CPN structure is automatically obtained from the graph of messages (Fig. 6b) in the scenario by adding places between each pair (Fig. 10a) of sequential messages.



(a) (b)

Fig. 10: CPN second 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 by identifying the changes in object states.

4.3 Scenario Integration

In this activity, we aim to merge all CPNs corresponding to the scenarios of a use case Uci, in order to produce an integrated CPN modeling the behavior of the use case. Our algorithm is based on a preliminary version presented in [7]. It takes an incremental approach to integration. Given two scenarios with corresponding CPNs CPN1 and CPN2,

the algorithm merges all places in CPN1 and CPN2 having the same names. The merged places will have as color the union of the colors of the two scenarios. Then, the algorithm looks for transitions having the same input and output places in the two scenarios and merges them with an OR between their guard conditions. In the following, we describe the algorithm in pseudocode, using the "dot"-notation known from object-oriented languages.

Integrating two scenarios whose CPNs have the colors [sc_1] and [sc_2], respectively, will produce a CPN with the list [sc_1,sc_2] as color. The operation of merging follows the steps described below:

```
integrate(uc_cpn,sc_cpn)
   uc_cpn.addPlaces(sc_cpn)
   // adds in uc_cpn places of sc_cpn that do not exist in uc_cpn
   for each t in sc_cpn.getListOfTransitions()
      t' = uc_cpn.LookForTrans(t)
      // t' is a transition of uc_cpn with •t=•t' and t•=t'•
      if (t' does not exist)
         uc_cpn.addtrans(t)
      endif
   end
   uc_cpn.addEdges(sc_cpn)
   // adds to uc_cpn edges of sc_cpn that do not exist in uc_cpn
   uc_cpn.mergeColors(sc_cpn)
   // calculates the new color of the integrated CPN (uc_cpn)
   uc_cpn.putColorsOnPlaces(sc_cpn)
   // all places of the net will have the merged color
   uc_cpn.putGuardOnTransitions(sc_cpn)
   // common transitions will be guarded by the merged color,
   // the others will be guarded by their original colors
   uc_cpn.putVariablesOnEdges(sc_cpn)
   // put on edges variables or token expressions
end merge
```

The resulting specifications derived from scenarios can be verified using exiting tools. We can check scenario specifications before and after integration to detect easily scenarios that cause incoherence in the integrated specification. We used the designCPN [6]. to check and the simulation of the resulting hierarchical CPN.

5. CONCLUSIONS

In this work, we have presented a new approach that produces automatically a global description or specification of a given service of the system or the whole system. The developed algorithms permit to derive the behavior of a use case by simply composing the scenarios describing it. Four operators for composing scenarios have been implemented: sequential, conditional, iterative and concurrent. 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.

We presented the use of the scenario approach in B2B systems. We have discussed the need of a unified model of interaction that gives a unique syntax to express interactions at different levels: use cases, scenarios and messages. Five scenario operators have been implemented: sequential, conditional, iterative, concurrent and abstraction/refinement.

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] 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.

[2] OMG: Object Management OMG. Unified modeling language specification version 2.0: Infrastructure. Technical Report ptc/03-09-15, OMG, 2003.

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

[4] Bennani M., Elkoutbi M., and Nafil K.; Modelling Real-time Aspects using UML Scenarios proceedings of the 3rd International Conference on Software Methodologies, Tools and Techniques, pp. 200-213, Leipzig, Germany, September 2004.

[5] 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.

[6] M. Elkoutbi, Khriss I., R.K. Keller. "Automated Prototyping of User Interfaces Based on UML Scenarios". The Automated Software Engineering Journal, 13, 5-40, 2006.

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