Data Source: INSPEC Record type: Conference paper Accession number: 7145320 INSPEC Abstract number: C2002-02-6110F-024
Editor(s): Kim, M.; Chin, B.; Kang, S.; Lee, D. Author affiliation: Dept. of Electr. & Comput. Eng., Concordia Univ., Montreal, Que., Canada
Conference Title: Formal Techniques for Networked and Distributed Systems. FORTE 2001 IFIP TC6/WG6.1-21st International Conference on Formal Techniques for Networked and Distributed Systems Inclusive page numbers: 19-34
Publication date: 2001 Publication year: 2001 Country of publication: USA Place of publication: Norwell, MA, USA
Material identity number: XX-2001-01853 Standard book number: 0 7923 7470 3 Publisher: Kluwer Academic Publishers
Conference date: 28-31 Aug. 2001 Location: Cheju Island, South Korea Number of references: 18

Controlled indexing: conformance testing; formal specification; specification languages; systems analysis; telecommunication computing
Uncontrolled indexing: stepwise design; Message Sequence Charts; use cases; software process automation; formal language; telecommunications specification; stepwise refinement; internal system component interactions; system architecture; automatic validation; conformance relation; text interpretation; system design; implementation errors; informal notations; design specification Copyright Statement: Copyright 2002, IEE

STEPWISE DESIGN WITH MESSAGE SEQUENCE CHARTS*

Ferhat Khendek[1], Stephan Bourduas[1], Daniel Vincent[2]

[1]Department of Electrical and Computer Engineering, Concordia University
1455, de Maisonnneuve W., Montréal (P.Q.), Canada H3G 1M8
E-mail: {khendek, s_bourdu}@ece.concordia.ca
[2]France Telecom R & D, Lannion, France

Abstract Use cases are useful in various stages of the software process. They are very often described using text that has to be interpreted by system designers. This could lead to implementation errors. Another drawback of using such informal notations is that automating the process of moving from use cases to design specification is difficult, if not impossible. It would be beneficial to represent use cases in an unambiguous way, thereby reducing the probability of misunderstanding and allowing for automation of various activities in the software process. Message Sequence Charts (MSC) is a formal language and widely used in telecommunications for the specification of the required behaviors. In this paper, we use MSC for describing use cases and we propose an approach for stepwise refinement from high-level use cases in MSC to design MSCs that contain more details about the internal components of the system and their interactions. The refinement steps are done by the designer and guided by the system architecture. For each step, the newly obtained MSC is validated automatically against the previous MSC using a conformance relation between MSCs.

1. INTRODUCTION

Distributed software systems, like any software system, go through requirement, design, implementation and testing phases. Ensuring the quality of such software systems from the initial stages is a challenging task.

UML use cases are becoming the standard form for requirements specification. An UML use case describes some functionality offered by a system as perceived by the user or an external actor of the system [1, 2, 9]. The user sees the system as a black box that responds to inputs with a specified output. Use cases are very often specified using a combination of text and diagrams that must be interpreted by the system designers and translated into a more concrete representation. It would be beneficial to represent use cases in an unambiguous way from the start, thereby reducing the probability of misunderstanding, and enabling the use of tools. MSC [3, 4, 11] is an excellent candidate as discussed in [10].

* Partially supported by France Telecom R&D.

MSC and SDL (Specification and Description Language) [5, 6] are widely used for telecommunication software engineering. In a previous work [7, 8] we have developed an approach and a tool for generating SDL specifications from a MSC and a given architecture. The given MSC is seen as a design MSC where the internal behavior of the components of the system is given. In this paper, we introduce a new approach for refining use cases specified with MSC into design MSC in a stepwise manner. The resulting MSC is then used as input for our MSC to SDL translation approach.

In our approach, a use case model is developed through interactions with the system designers and the customers as described in [1, 2, 9]. The result of this represents the functional requirements of the system under construction. Once the use cases have been agreed on, the designers must specify the architecture of the system. Guided by this architecture, the use cases are then refined in a stepwise manner. We distinguish between horizontal and vertical refinements. Horizontal refinement consists of adding new messages or actions to existing MSC axes. Vertical refinement consists of decomposing an axis into at least two other axes following the architecture of the system. The enriched MSC must conform to the previous (or parent) MSC. We therefore define a conformance relation between MSCs that is used to validate refinement steps made by the designer.

The rest of this paper is organized as follows. In Section 2, we discuss the modeling of use cases with MSC. Section 3 introduces the notion of conformance between MSCs and discusses the related work. In Section 4, Subsection 4.1 describes our stepwise refinement approach for bMSCs, while Subsection 4.2 extends this approach to HMSCs. In Section 5, we apply our approach to the ATM (Automatic Teller Machine) example, before concluding in Section 6.

2. DESCRIBING USE CASES WITH MSC

Jacobson introduced the concept of use cases in the Object-Oriented Software Engineering (OOSE) method [9]. Use cases are now part of the UML standard. A use case is an abstract description of some desired functionality provided to the user of a system. The use case description sees the system as a black box that generates some output for a given input. The fact that they are not formalized allows for flexibility in the early stages of development. This same advantage can turn into a liability if inexperienced designers are expected to interpret the use cases and then produce a design specification. In [10], the authors propose a method for formalizing use cases using MSCs. We follow this approach for the modeling of use cases.

2.1 bMSC Use Cases

bMSCs are simple diagrams that capture the interactions between system components, and between system components and the environment. For use case modeling, bMSCs are used to show the interactions between the system and the environment only as illustrated in Figure 1. The user sees a “black box” represented by a single process instance in the MSC. The axis represents the system boundary that interacts with the user. In this example, the user sends message “a” to the system, and the system reacts by sending message “b” to the user.

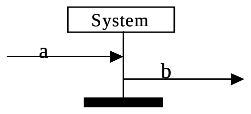


Figure 1. A bMSC use case.

2.2 HMSC Use Cases

HMSC allows for a more abstract view of the system [10, 11]. HMSCs improve the readability of the system by hiding low-level details and showing graphically how a set of MSCs can be combined. They are viewed as roadmaps composed of MSCs using sequential, alternative and parallel operators. HMSCs are seen as directed graphs where the nodes are: start symbol, end symbol, MSC reference, a condition, a connection point, or a parallel frame.

For simple use cases, a single bMSC is sufficient to show the behavior, but complex behaviors such as conditional executions or alternatives can lead to large and unreadable bMSCs. For these complex use cases, it is easier to use several bMSCs and then combine them using a HMSC use case.

A use case describes one possible usage of a system. Most systems will have many functions that will result in many use cases. In order to be able to organize many use cases and to allow a designer to view and navigate them with ease, a HMSC that references the HMSC use cases could be used to show an even higher level of abstraction. Using three level of abstraction allows us to specify the use cases for an entire system. Figure 2 shows how the various levels relate to each other.

The system level expresses the functional view, as a set of use cases, of the system by using a top-level HMSC. The structure level describes each use case using one HMSC without going into details. The basic level shows the interactions between the system and the environment using bMSCs. A bMSC at that level cannot refer to a HMSC as allowed in the standard [3]. A bMSC at the basic level does not contain alternatives or optional behaviors. Any alternative or optional behavior can be represented using HMSC at the structure level. In this paper, we restrain the HMSC language to the weak sequencing operator, alternative operator and iteration. The parallel operator is not taken into consideration.

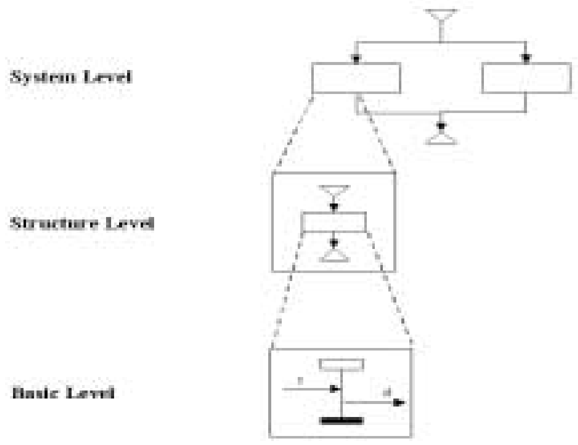


Figure 2. Three levels of abstraction for use case specification.

3. CONFORMANCE RELATION FOR MSC

In this section, we first introduce the definition of conformance between bMSCs, and then we extend this definition to HMSC.

3.1 bMSC

The bMSC refinement process consists of several steps. We refer by M_k to the bMSC obtained at step k. In order to preserve, throughout the refinement process, the semantics of the original use case defined by the user, each newly obtained bMSC should preserve the semantics of its parent bMSC. In other words, bMSC M_n must preserve the behavior described by bMSC M_{n-1}. M_n must have the events of M_{n-1} and for these events M_n preserves the orders defined in M_{n-1}. The new messages and events introduced in M_n are not constrained. Informally, we say that a bMSC M_2 preserves the behavior of (or *conforms to*) a bMSC M_1, if and only if for each axis A_1 in M_1 there is a corresponding set of axes $\{A_{21}, A_{22}, \ldots, A_{2n}\}$ in M_2 with all events of A_1 included in $\{A_{21}, \ldots, A_{2n}\}$, and all the orders defined between events in M_1 are preserved in M_2.

For a formal definition of the conformance relation, we need a formal definition of a bMSC. For this, we follow the definitions in [12, 13].

Definition 1 (bMSC). A bMSC is a tuple <V, <<, P, M, L, T, N, m>, where

- V is a finite set of events,
- $<< \subseteq V \times V$: is a transitive and acyclic relation,
- P is a set of processes,
- M is a set of message names,
- L: V $\rightarrow$ P is a mapping that associates each event with a process,
- T: V $\rightarrow$ {send, receive, local} defines the types of each event as a send, receive or local,
- N: V $\rightarrow$ M maps every event to a name
- m: a partial function that pairs up send and receive events.

The relation << is defined between the sending event (s_m) and the receiving event (r_m) of message "m" as $s_m << r_m$, and between events e_1 and e_2 in the same axis, $e_1 << e_2$, if e_1 appears before e_2.

Definition 2 (Conformance for bMSCs). A *bMSC M2* = $<V_2, <<_2, P_2, M_2, L_2, T_2, N_2, m_2>$ *conforms to a bMSC M1* = $<V_1, <<_1, P_1, M_1, L_1, T_1, N_1, m_1>$, if and only if there exist an injective mapping $\Gamma: V_1 \rightarrow V_2$ and a surjective function $\phi: P_2 \rightarrow P_1$ such that:

- $L_1(e) = \phi(L_2 (\Gamma(e)))$
- $T_1(e) = T_2 (\Gamma(e))$
- $N_1(e) = N_2 (\Gamma(e))$
- if $m_1(e) = f$ then $m_2(\Gamma(e)) = \Gamma(f)$
- if $e <<_1 f$ then $\Gamma(e) <<_2 \Gamma(f)$

Our conformance relation is similar to the matching relation defined in [12, 13]. However, there are two differences. The first one is related to the MSC semantics. In fact, we do not distinguish between the visual order and the enforced order as it is done in [12, 13]. The visual order being the transitive and reflexive closure <<* of <<, while the enforced order depends on the communication architecture where some visual orders may not hold, because of race conditions [14]. In the formal semantics of MSC (for bMSC [15]), the visual order has to be enforced. The question "if an architecture allows for that order or not" is another issue. The second difference as mentioned earlier in our conformance relation, an axis in bMSC Mn may correspond to a set of axes in bMSC Mn+1. In other words, we allow for one-to-many relationship between axes, contrary of the one-to-one relation in [12, 13].

Mauw and Reniers have proposed a refinement relation for interworkings [16]. This refinement consists of decomposing an instance into its constituents and adding internal messages between these constituents. Beside the semantic issues, our conformance relation can be seen as a combination of the refinement relation in [16] and the matching relation in [12, 13]. In this paper, the terms "refinement" and "conformance" are not synonymous. With the term "refinement" we only mean developing further a MSC, without automatically ensuring conformance.

For the illustration of the conformance relation, let us consider the bMSCs in Figure 3. A mapping Γ between M1 and M2 that associates each event to itself and a function ϕ that associates each process to itself satisfy the conditions in Definition 2. bMSC M2 conforms to bMSC M1. We have more messages and events in M2, but these events preserve the orders defined in M1. The bMSC M3 also conforms to M1. In this case, the function ϕ associates P21 and P22 to P2. In the case of bMSC M4, we cannot find a mapping of events that preserves the orders defined in M1. In fact, reception of message "x" by P21 and reception of "y" by P22 are not ordered as specified in P2 in M1.

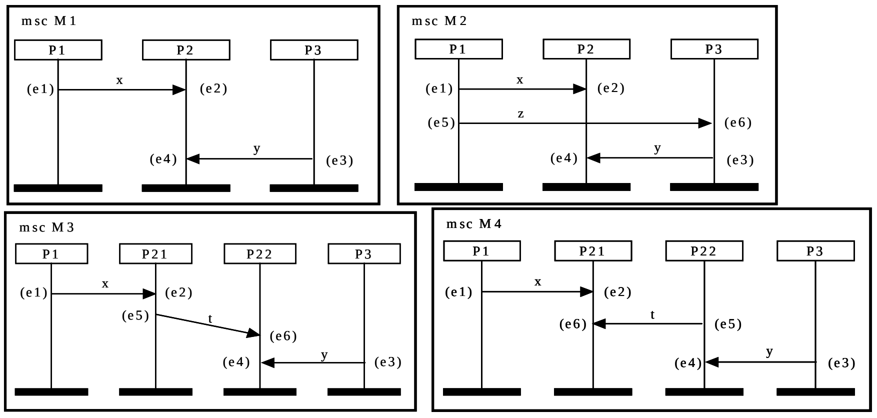


Figure 3. Examples of conformance and non-conformance between bMSCs.

3.2 HMSC

In order to build complex behaviors from simple behaviors, the MSC standard [3] has defined weak sequential, iteration, alternative and parallel compositions of MSCs. Informally, weak composition of two bMSCs can be seen as a concatenation of the axes of the common processes in the bMSCs.

Definition 3 (Weak composition of bMSCs). Given two bMSCs, $M1 = <V_1, <<_1, P_1, M_1, L_1, T_1, N_1, m_1>$ and $M2 = <V_2, <<_2, P_2, M_2, L_2, T_2, N_2, m_2>$, with $V_1 \cap V_2 = \emptyset$, $M1 \text{ o } M2 = <V_1 \cup V_2, <<, P_1 \cup P_2, M_1 \cup M_2, L_1 \cup L_2, T_1 \cup T_2, N_1 \cup N_2, m_1 \cup m_2>$, where $<< = <<_1 \cup <<_2 \cup \{(e_1, e_2)$, such that $L_1(e_1) = L_2(e_2)$, $e_1 \in V_1$ and $e_2 \in V_2\}$.

From the syntactical point of view, a HMSC defines a roadmap where MSCs are combined using weak sequential, iteration, alternative and parallel compositions. As mentioned in Section 2, we do not take into account the parallel operator in this paper. From a semantic point of view, we define a HMSC as a potentially infinite set of alternatives of (infinite) weak sequential composition of bMSCs.

Definition 4 (HMSC). A HMSC H_1 is a set of sequences seq_i (or bMSCs), with $seq_i = M_{i1} o M_{i2} o M_{i3} o \ldots o M_{iq}$, where M_{ij} is a bMSC, for $j = 1, \ldots, q$ and $i = 1, \ldots, n$.

The set and the sequences could be infinite. The iteration is defined as the repetition, using weak sequential composition, of the same bMSC.

Informally, we say that a HMSC H_1 conforms to HMSC H_2, if and only if for each alternative bMSC M_1 in H_1 there is a bMSC M_2 in H_2, such that M_2 conforms to M_1. M_2 preserves the behavior of M_1. H_2 may contain more alternatives than H_1, but each alternative of H_1 is preserved in at least one alternative of H_2.

Definition 5 (Conformance for HMSCs). Given two HMSCs, $H_1 = \{seq1_i, i = 1, \ldots, n\}$ and $H_2 = \{seq2_j, j = 1, \ldots, m\}$, *$H_2$ conforms to H_1*, if and only if for each $seq1_i$ in H_1 there exist $seq2_j$ in H_2 such that *$seq2_j$ conforms to $seq1_i$*.

The conformance between $seq1_i$ and $seq2_j$ is given in Definition 2. Both conformance relations (for bMSCs and HMSCs) are transitive and reflexive.

4. FROM USE CASES TO DESIGN SPECIFICATION

Typically, designers will use their experience to generate design specifications from textually represented use cases. This method can lead to errors in implementation if the use cases are unclear. How these types of errors can be avoided is the goal of our approach. It enables a designer to specify an initial use case in MSC and refine it incrementally into a MSC design specification. The system architecture plays an important role in the stepwise refinement of MSC use cases. A refined MSC must conform to its parent MSC and follow the underlying system architecture. This system architecture can be represented using either UML or SDL. Our methodology for stepwise refinement of use cases is to be used in conjunction with existing tools for translation of a UML architecture into a SDL architecture [17], and the generation of a SDL specification from a given target architecture and MSCs [8].

4.1 bMSC Refinement Methodology

During the refinement process, we distinguish between vertical and horizontal refinements. Vertical refinement reflects the architectural decisions, while horizontal refinement allows to enrich the behavior of a bMSC. Both types of refinements are used together to incrementally refine a use case bMSC into a design specification. Figure 4 illustrates how both types of refinements are combined to reach a design specification.

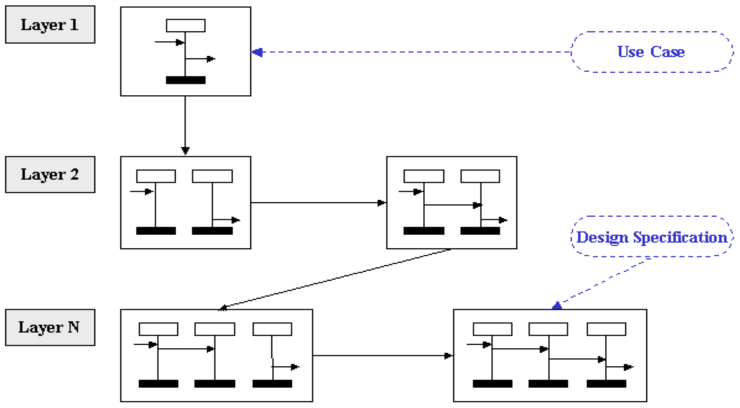


Figure 4. Overall refinement methodology.

The refinement process is dependent on the system architecture. The amount of layers used in the architecture limit the number of vertical refinements. There can only be one vertical refinement per layer. Each vertical refinement step splits a bMSC instance into several instances according to the architecture. The instances that are not decomposed keep the same names. When an instance is decomposed the designer has to distribute its events among the new instances. For each message “m”, the names of the associated events (sending and receiving) are kept unchanged. The messages and events associated with instances that are not refined are kept unchanged and re-generated automatically. In general, a combination of vertical and at least one horizontal refinement is needed in order to generate a MSC that may conform to the previous level.

Horizontal refinement is concerned with adding messages, events and local actions to the bMSC. The designer can add new messages, sending and receiving events, as well as local actions. The messages that can be used in a horizontal refinement are specified in the architecture. A designer cannot introduce messages that are not specified in the architecture at the current level of refinement. Unlike vertical refinement, there is no limit to the number of horizontal refinements that can be performed at any given layer.

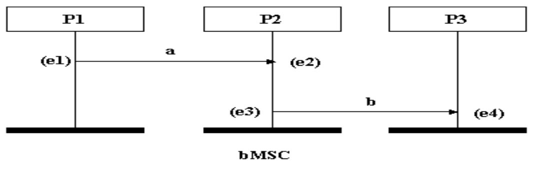
After each horizontal refinement, the designer can check automatically whether the refined bMSC conforms to the previous one. This verification of the conformance relation between bMSCs is implemented with the Event Order Tables (EOT) introduced in [7, 8]. As mentioned earlier, the bMSCs used in this paper do not contain alternative or optional behaviors.

Event Order Tables (EOT)

An EOT for a bMSC is a matrix that shows precedence relationships between events of the MSC. Figure 5 shows a bMSC and its corresponding EOT. Each message between two instances corresponds to two events: sending and receiving. Once the events on a MSC have been labeled appropriately, the EOT can be constructed. Following the MSC semantics and the relation <<, we use two rules to generate the EOT [8]:

- Each instance is totally ordered (except for co-regions where there is no order between events).
- A reception event happens after its matching sending event.

Inspection of each process axis in Figure 5 reveals that the first rule must be applied to P2, yielding the relation $e_2 << e_3$. Applying the second rule to the MSC of Figure 5 yields $\{e_1 << e_2, e_3 << e_4\}$. The transitive closure $<<^*$ of << allows for the construction of the EOT. For instance, $e_1 << e_2$ and $e_2 << e_3$ implies $e_1 << e_3$. A cell in the EOT is set to true if the row event occurs before the column event.



	e1	e2	e3	e4
e1		T	T	T
e2			T	T
e3				T
e4				

Event Order Table

Figure 5. A bMSC and its EOT.

Definition 6 (Inclusion for EOTs). We say that *EOT T_1 is included in EOT T_2*, if and only if

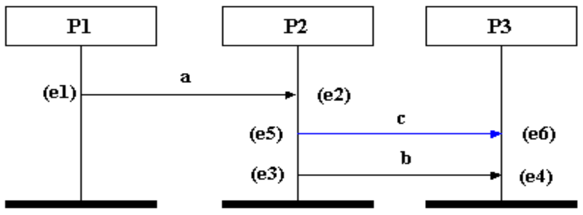
- all the events of T_1 are in T_2, and
- for each pair of events (e_i, e_j) if $e_i << e_j$ in T_1 then $e_i << e_j$ in T_2.

Provided the relation, enforced by the bMSC refinement approach (and tool), between instances and events in bMSC M_{k+1} and bMSC M_k, we say that M_{k+1} conforms to M_k if and only if EOT of M_k is included in EOT of M_{k+1}.

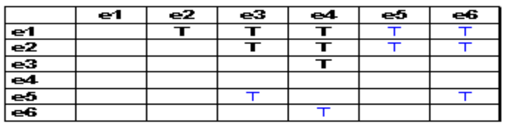
Proposition 1 (Conformance of bMSCs using EOTs). Given bMSCs, M_k and M_{k+1}, provided that events in M_k are mapped into themselves in M_{k+1}, M_{k+1} conforms to M_k if and only if EOT of M_k is included in the EOT of M_{k+1}.

Proof. The proof is straightforward. Our refinement approach maps events in M_k into themselves in M_{k+1}. The orders defined in M_k are preserved in M_{k+1} if and only if the EOT of M_k is included into the EOT of M_{k+1}. **[End]**

Figure 6 shows a bMSC that conforms to the bMSC in Figure 5, the message “c” sent from P_2 to P_3 has been added. In order to preserve the original event labels, the event labels e_5 and e_6 have been used for the new events.



Refined bMSC



	e1	e2	e3	e4	e5	e6
e1		T	T	T	T	T
e2			T	T	T	T
e3				T		
e4						
e5			T			T
e6				T		

Event Order Table

Figure 6. Example of conformance verification using EOTs.

The EOT in Figure 5 specifies an ordering of events that is maintained by the refined bMSC. In fact, the EOT in Figure 5 is included in the EOT in Figure 6. The old relations are bolded in Figure 6; as long as these are not violated, the new bMSC conforms to its parent bMSC.

4.2 Stepwise Refinement of HMSCs

A HMSC is composed of a number of bMSCs following a certain roadmap. To enrich a use case HMSC, we keep the same roadmap and we refine the bMSCs. A HMSC is refined according to the following rules:

1. Keep the roadmap unchanged.
2. For vertical refinement, all bMSCs are vertically refined at the same time so that each bMSC has the same number of instances.
3. For horizontal refinement, we refine horizontally the referenced bMSCs. Each bMSC can be refined horizontally independently of the others.

Vertical refinements are done automatically according to the system architecture and the current level of abstraction. The user refines bMSCs horizontally using the method described previously and messages described in the architecture at the current level of abstraction. Unfortunately, the conformance of the refined bMSCs to their corresponding bMSCs does not lead automatically into the conformance of the refined HMSC to the original HMSC. In fact, orders between events in a given axis may not hold anymore when the axis is decomposed and the events distributed among the refining axes as shown in Figure 7. Indeed, in the HMSC M_1 o M_2, the sending of “x” always precedes the sending of “y”, but this not the case in the HMSC M_1’ o M_2’, because of the distribution of these two events. In this example, M_1’ conforms to M_1 and M_2’ conforms to M_2, but M_1’ o M_2’ does not conform to M_1 o M_2.

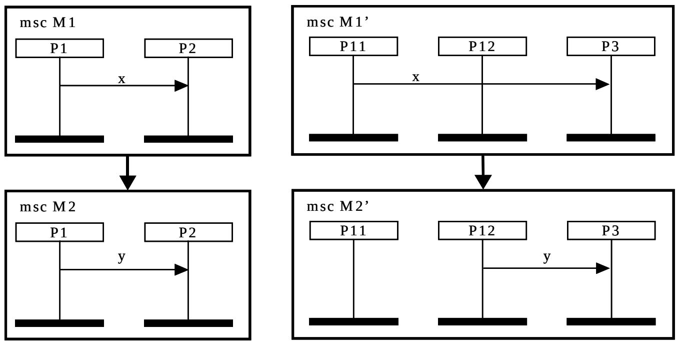


Figure 7. Non-conformance between HMSCs.

Theorem 1 (Conformance between HMSCs).
Given a HMSC H_1 and its refinement H_2, (with the same roadmap), if

- each bMSC in H_2 conforms to its corresponding bMSC in H_1, and
- in the HMSC H_2, for each pair of bMSCs, M_1’ and M_2’, such as M_1’ o M_2’ in H_2, for each set of axes $\{A_{11}, \ldots, A_{1n}\}$ in M_1’ and M_2’ resulting from the decomposition of A_1 (in M_1 and M_2 in H_1), the order between the last event of A_1 in M_1 and the first event of A_1 in M_2 is preserved,

then H_2 conforms to H_1.

Proof. Consider two HSMCs, H_1 and its refinement H_2 obtained following the rules mentioned above. From a semantic point of view, each HMSC is a set of alternatives of concatenation of bMSCs. Roadmaps of H_1 and H_2 are identical, each bMSC in H_1 has a corresponding bMSC in H_2. Any sequence of concatenation seq_1 in H_1 has its corresponding sequence of concatenation seq_2 in H_2, using corresponding bMSCs in H_2. In order to show that H_1 conforms to H_2, we have to show that seq_2 conforms to seq_1. For that, we only have to prove that if M_1’ conforms to M_1, M_2’ conforms to M_2, and the second condition of the theorem holds, M_1’ o M_2’ conforms to M_1 o M_2.

M_1’ conforms to M_1 means that orders in M_1 are preserved in M_1’. M_2’ conforms to M_2 also means that orders of M_2 are preserved in M_2’. M_1 o M_2 (before decomposition of axes) imposes orders on events in the same axes and these orders have to be preserved in M_1’ o M_2’. The second condition of the theorem ensures the preservation of these orders in M_1’o M_2’. Therefore, M_1’ o M_2’ conforms to M_1 o M_2. **[End]**

During the refinement of HMSC use cases, we check the conformance between the new HMSC and its parent HMSC by checking the conformance between each pair bMSCs and the second condition of the theorem above. When a bMSC in the new HMSC does not conform to its corresponding bMSC in the parent HMSC, the refinement is not accepted. New refinements and modifications have to be applied to the new bMSC and the conformance checked again. In some cases, the designer has to add new messages in the refined HMSC to ensure that the second condition of the theorem holds. The tool can also add automatically “dummy” messages to satisfy this condition of the theorem. Notice that in our refinement approach the conformance of bMSCs is a sufficient condition and not a necessary one. However, in order to ensure the conformance between HSMCs, we enforce conformance between pairs of bMSCs. The study of all the possible necessary conditions is left for future investigations.

5. APPLICATION

Automation (tool) reduces the likelihood of human error and increases the confidence in the generated specifications.

5.1 Tool

We have developed a MSC refinement tool. The use cases are specified with HMSC and the system architecture is specified using UML (or SDL directly). The stepwise refinement of MSCs is used in conjunction with other methodologies that have been automated, namely:

- UML to SDL architectural translation [17].
- Generation of SDL specification from a given SDL architecture and MSC [7, 8]. The SDL specification preserves the behaviors specified in the MSC and is free of any design error such as deadlocks.

Figure 8 shows how the MSC refinement tool can be combined with existing tools. The shaded region represents the MSC refinement process. A tool guides the user through the refinement process and verifies that the resulting MSCs conform to the original use cases and follow the system architecture.

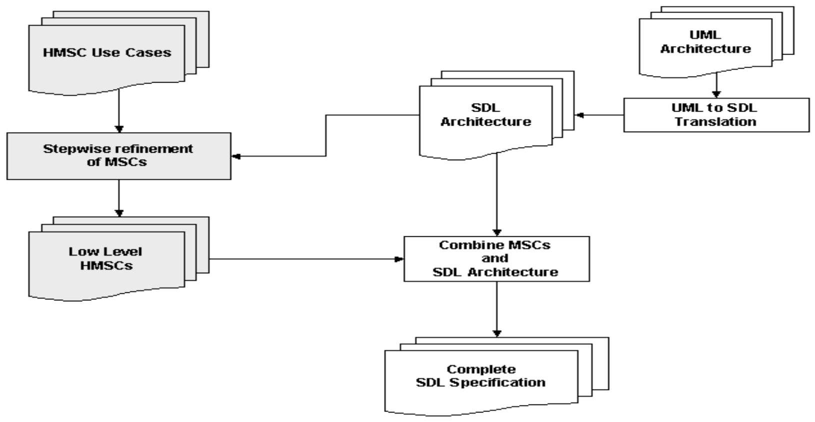

HMSC Use Cases
UML Architecture
SDL Architecture
UML to SDL Translation
Stepwise refinement of MSCs
Low Level HMSCs
Combine MSCs and SDL Architecture
Complete SDL Specification

Figure 8. A toolkit for developing SDL and MSC specifications.

The MSC refinement approach (and tool) can also be used in the front-end of the SDL specification enrichment approach introduced in [18].

5.2 Example

This section illustrates our refinement approach through an example. Figure 9 shows a system architecture representing an ATM machine using SDL. There are two processes, namely “ATM” and “BANK”. There is a channel that allows the user of the system to interact with the ATM process instance, and another channel allows for the two process instances to interact with each other. The signals these processes exchange with each other and their environment are specified in the architecture.

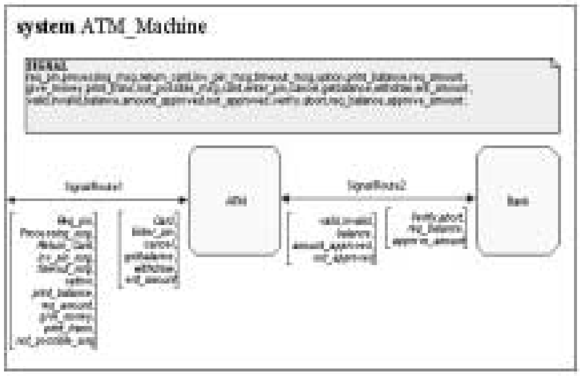


Figure 9. ATM system architecture in SDL.

Two functions provided by an ATM machine are the ability to withdraw and deposit money. Both these functions have a common starting point. The user must first insert a card and enter the correct PIN before a transaction can occur. Figure 10 shows the system and structure level HSMCs.

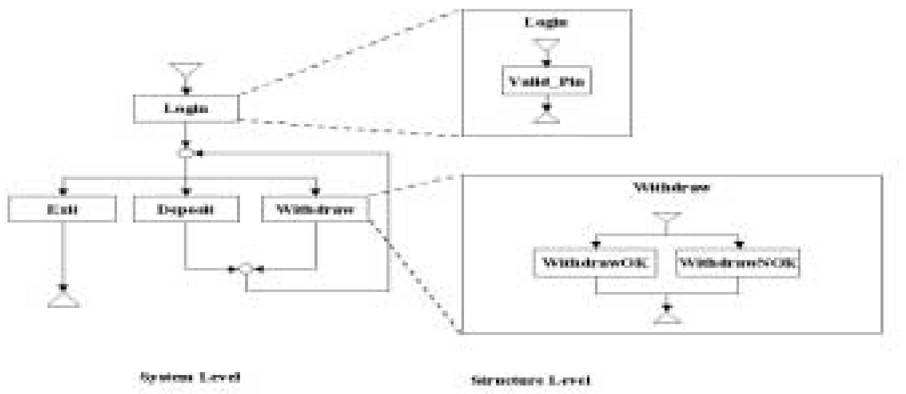


Figure 10. System and structure level HMSCs for the ATM example.

The system level HMSC shows three different use cases, "Exit", "Deposit" and "Withdraw". It also shows that each use case has the common behavior denoted by the "Login" block. The structure level shows more details. The "Login" HMSC has only one bMSC, "Valid_Pin". Later on, the system designer can decide to add an "Invalid_Pin" scenario to the "Login" HMSC that would not affect the rest of the system. The "Withdraw" HMSC shows that the withdraw use case can have two results, "WithdrawOK" and "WithdrawNOK". For illustration purpose, we will perform refinements on "WithDrawOK" only. The bMSC "WithdrawOK" in Figure 11.a is first refined vertically into the bMSC in Figure 11.b. Note that there is no new behavior added, only a splitting of the instance "ATM_Machine" to reflect the architecture of Figure 9.

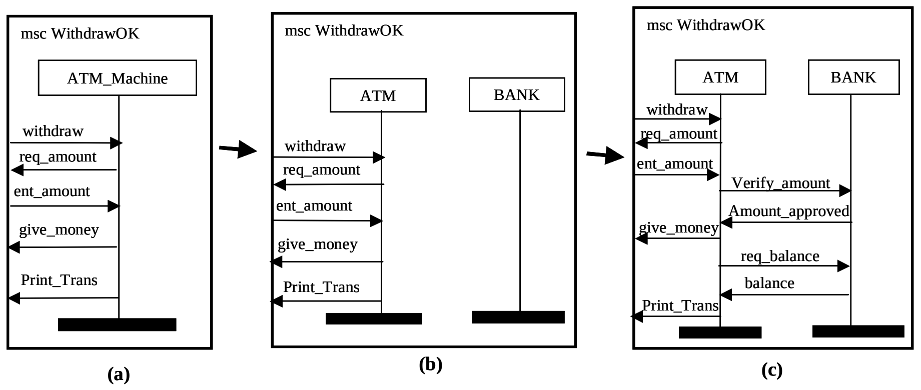


Figure 11. Vertical and horizontal refinements of "WithdrawOK".

Once the vertical refinement has been performed, the designer can now enrich the bMSC. A horizontal refinement is shown in Figure 11.c. The new MSC conforms to the original MSC in Figure 11.a.

Now suppose that the ATM block of Figure 9 was further decomposed into three separate processes: *Dialogue*, *MoneyHandler* and *Printer*. With our refinement process, the MSC of Figure 11.c is first refined vertically to show the decomposition of the ATM block. This refinement is followed by a horizontal refinement with the addition of messages. The resulting MSC, that conforms to the MSC in Figure 11.c and therefore to the initial use case in Figure 11.a, is shown in Figure 12.

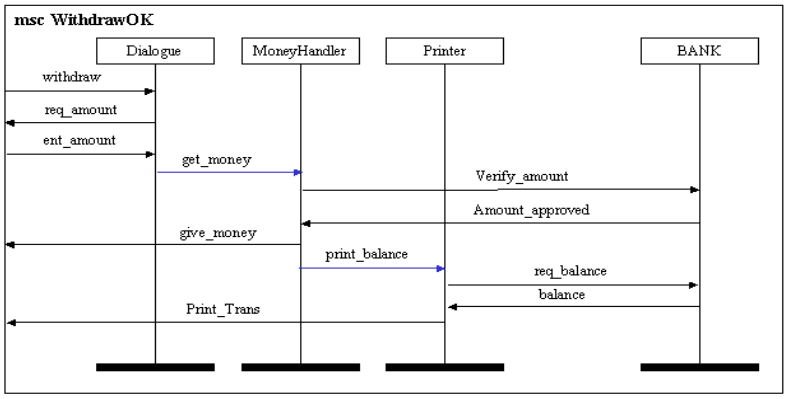


Figure 12. Another refinement of “WithdrawOK”.

While horizontal refinements are being done separately for each bMSC of the HMSC. The designer can verify the conformance of each new bMSC (HMSC) against its corresponding bMSC (HMSC) in the previous stage.

6. CONCLUSION

Use cases are a good way to capture user requirements. They are abstract and are often described using textual documents that can be misinterpreted later on by designer. The reason for this is that designers apply their experience to translate use cases into a design and implementation. A structured way of going from use cases to specification would benefit the software development process by detecting and avoiding errors early on, and by enabling easier software comprehension. This will reduce development and maintenance costs. MSCs are well suited for this because they can show a system at a high level of abstraction, or they can be used for low-level specifications. They are intuitive, easy to learn and formally defined.

The goal of this project is to create a methodology and a CASE tool to semi-automate the process of refining a use case HMSC into a design specification. The MSC refinement tool is used as a front-end tool for an existing set of tools for translating UML architecture specifications to SDL architecture specifications, and from MSC specifications to SDL specifications. This set of tools aims at improving the quality of the software as well as shortening the development process.

Other issues such as conformance relations that allow for enrichment and modification of roadmaps are under consideration.

Acknowledgements: We would like to thank Loïc Helouët from France Telecom for his very helpful comments.

REFERENCES

1 OMG, "*UML version 1.3*", June 1999.
2 H. E. Eriksson and M. Penker, "*UML Toolkit"*, John Wiley & Sons Inc., 1998.
3 ITU-T, Z.120, "*Message Sequence Charts – MSC'2000*", November 1999.
4 S. Mauw, M. A. Reniers and T.A.C. Willemse., "*Message Sequence Charts in the Software Engineering Process*", Report 00/12, Department of Computer Science, Eindhoven University of Technology, 2000.
5 ITU-T, Z.100, "*Specification and Description Language – SDL'2000*", November 1999.
6 J. Ellsberger, D. Hogrefe and A. Sarma, "*SDL: Formal Object Oriented Language for Communicating Systems*", Prentice-Hall, 1997.
7 M.M. Abdalla, F. Khendek and G. Butler, "*New Results on Deriving SDL Specifications from MSCs"*, Proceedings of SDL Forum'99, Montréal, Canada, June 21-25, 1999.
8 G. Robert, F. Khendek and P. Grogono, *"Deriving an SDL Specification with a Given Architecture from a Set of MSCs"*, Proceedings of SDL Forum'97, Evry, France, Sept. 1997.
9 I. Jacobson et al, "*Object-Oriented Software Engineering*", Addison-Wesley, 1992.
10 M. Andersson and J. Bergstrand, *"Formalizing Use Cases with Message Sequence Charts"*, Masters Thesis, Lund University, Sweden, 1995.
11 E. Rudolph et al, "*Tutorial on Message Sequence Charts (MSC'96)*", Proceedings of FORTE/PSTV'96, Kaiserslautern, Germany, Oct. 1996.
12 A. Muscholl and D. Peled, "*Analyzing Message Sequence Charts*", Proceedings of SDL And MSC Workshop (SAM'2000), Grenoble, France, June 26-28.
13 A. Muscholl, D. Peled and Z. Su, "*Deciding properties of message sequence charts*", Proceedings of FoSSaCS'98, LNCS 1378, Springer, 1998.
14 R. Alur, G. Holzmann and D. Peled, "*An analyser for message sequence charts*", Software Concepts and Tools, 17(2), 1996, pp. 70-77.
15 S. Mauw and S. A. Reniers, "*An algebraic semantics for Basic Message Sequence Charts*", The Computer Journal, 37(4), 1994, pp. 269-277.
16 S. Mauw and M. Reniers, "*Refinement in interworkings*", Proceedings of CONCUR'96, LNCS 1119, Springer, 1996.
17 S. Bourduas, F. Khendek and D. Vincent, "*From MSC + UML to SDL*", Technical Report, Concordia University, November 2000.
18 F. Khendek and D. Vincent, "*Enriching SDL Specifications with MSCs*", Proceedings of SDL And MSC Workshop (SAM'2000), Grenoble, France, June 26-28, 2000.