15 Tutorial on MSC-96

MSC-96 Tutorial

Introduction

MSC-96 is the ITU standard for Message Sequence Charts which was finalized in 1996 [110]. MSC-96 is backward compatible with MSC-92 [105] and adds a number of features which makes MSC-96 more versatile and powerful than MSC-92.

In this tutorial we take as our starting point the same example which has been used for the MSC-92 tutorial and show how MSC-96 could be used to specify the same problem.

In order not to repeat ourselves too much we will not introduce again all features also found in MSC-92. For those readers not familiar with MSC-92 it is probably wise to start by reading the MSC-92 tutorial. The concepts of timers and instance creation and instance stop have not changed in MSC-96. The concepts of condition and submsc of MSC-92 have been slightly modified (See Restrictive conditions (p.15-6) and Figure 15-10 "General order relation" (p.15-15)) without making the interpretation of old diagrams drastically different when interpreted as MSC-96 diagrams.

The new features include: MSC references, MSC expressions, gates, HMSC, general ordering and substitution. In addition a number of clarifications to the interpretation of MSC has been made. To let the reader get some idea about what the new language mechanisms are, we list brief explanations of the improvements:

- MSC References (p.15-4) make it possible to refer from within one MSC to other MSCs.
- MSC operators (p.15-10) combine MSCs to express alternatives, parallel merge and loops.
- Gates (p.15-17) serve as flexible connection points between references/expressions and their surroundings.
- HMSC diagram (p.15-5) – High Level MSC is a new kind of MSC diagrams introduced for better overview of MSC documents.
- General Ordering (p.15-14) improves the ability for MSC to express partial orders between events more precisely than strict order or no order.
- Substitution (p.15-23) makes MSC more general and may be used to simulate such object-oriented features as inheritance and virtuality.

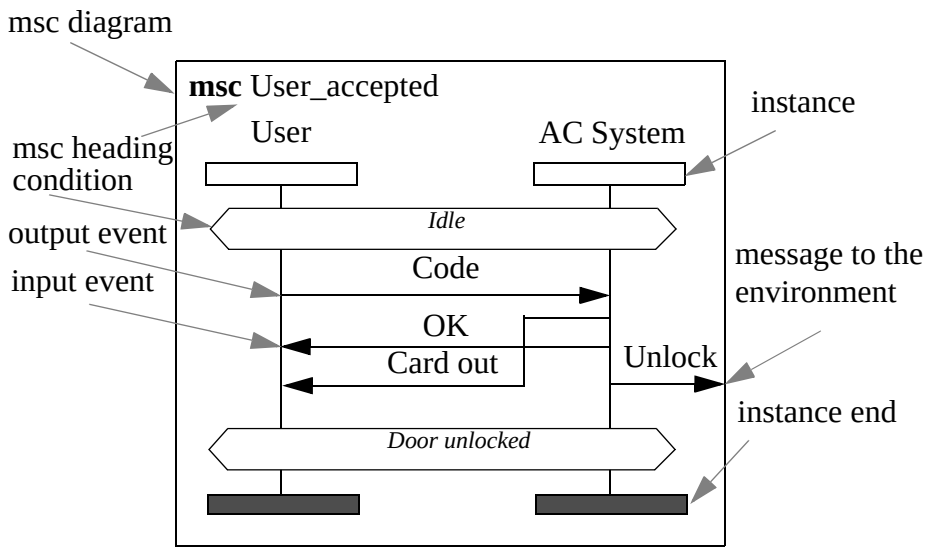
Basic MSC in a nutshell

Before we dig into MSC-96, we should re-introduce the example and repeat the basic concepts of MSC.

Our example is the Access Control System and we start by specifying a number of MSCs to describe the interaction between one user and the system as a whole. Our first case is when the User is accepted for access.

Figure 15-1: Basic MSC

Open figure



The MSC diagram in Figure 15-1 (p.15-3) shows a case where the User presents a Code to the AC system and the system responds by ejecting the card and displaying an OK message. These two messages change order on their way to the User. The display of OK overtakes the mechanical releasing the card. Furthermore the AC system will unlock the door which is considered to be outside the specified case.

This MSC is a plain MSC-92 diagram, but its communication with its environment can be interpreted in MSC-96 as what shall be known as a gate.

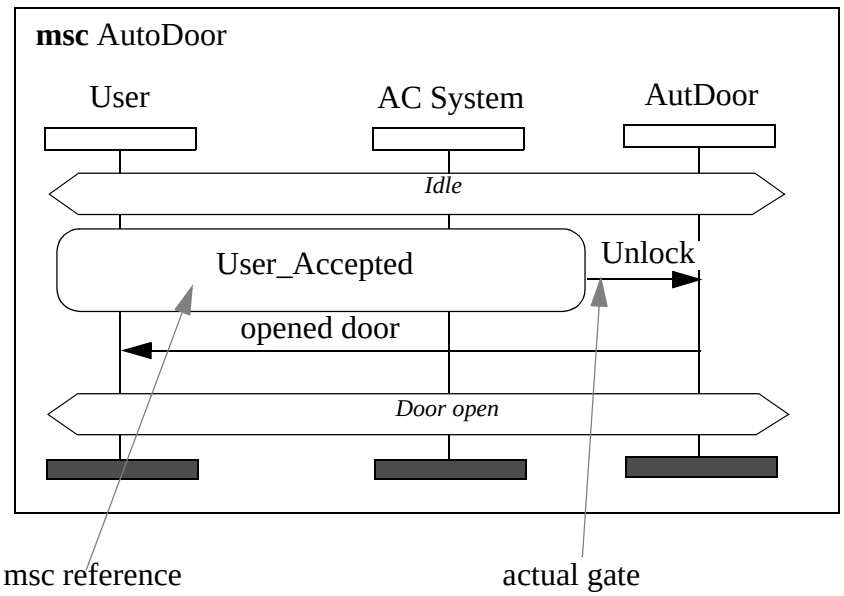
MSC References

In almost all description languages such as programming languages and specification languages there is a way to isolate subparts of the description in a separate named construct and then refer to this name from other places in the overall description. In Algol we have procedures, in C we have functions, in Simula and C++ there are classes and in Ada there are packages. In MSC there are MSCs which can be referred from other MSCs.

Assume that the scenario where the user is accepted is a part of a larger context where there is an automatic door. When the door is unlocked it automatically opens.

Figure 15-2: MSC reference

Open figure



The MSC reference symbol is a box with rounded corners. We also notice that the message *Unlock* out of User Accepted is consistent in Figure 15-1 (p.15-3), where it is a message to the frame and in Figure 15-2 (p.15-4), where it is a message from the MSC reference.

MSC documents

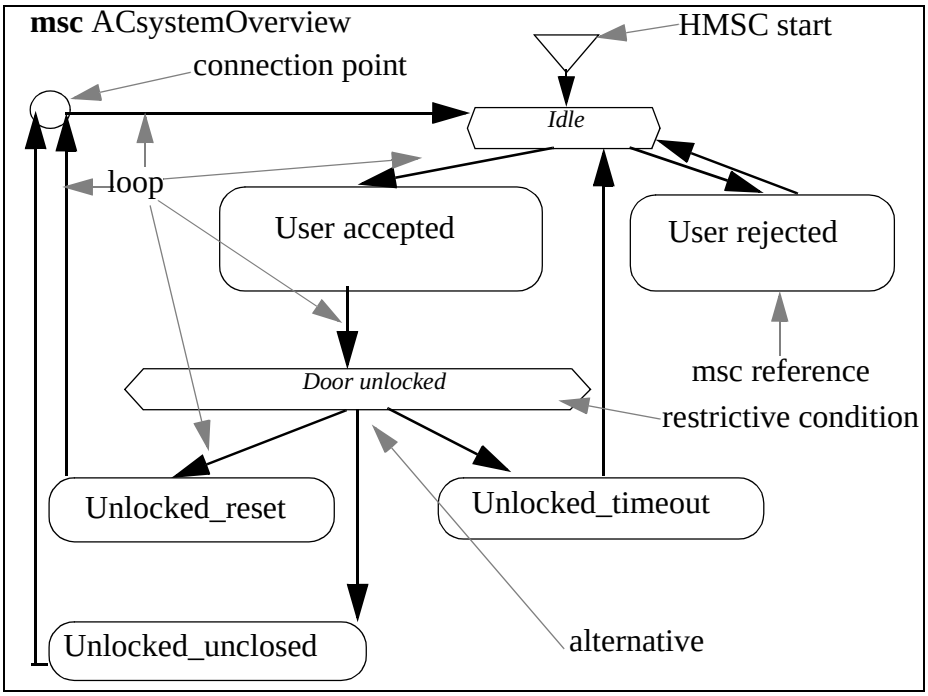
We recall from MSC-92 that a set of MSCs is called an MSC document. In MSC-92 there was no language construct to keep the overview of an MSC document. In our MSC-92 methodology we presented two informal ways to keep the overview of the MSC document. On approach was called the Road Map. The other approach was a table of variants.

In MSC-96 the road map has been carried further into what we call High Level MSC (HMSC). An HMSC diagram is also an MSC diagram and as such references to it may appear in other HMSCs or in plain MSCs.

An HMSC has no instances, it consists of nodes and branches. The nodes of the graph are the individual MSCs (which are combined in sequence or to alternatives) and conditions. For simplicity also connection points are included to minimize the number of parallel branches.

Figure 15-3: HMSC diagram

Open figure



This HMSC gives an overview of the MSC document. The HMSC start symbol defines the entry of the graph. Here the condition *Idle* will hold in the beginning. The conditions are restrictive in an HMSC. This will be covered in Restrictive conditions (p.15-6).

When more than one branch leads from a condition (or directly from the MSC reference) this represent an “alternative” meaning that one of the branches will occur.

When a sequence of branches (and nodes) together form a cycle, this represents a loop. In Figure 15-3 "HMSC diagram" (p.15-5) we have several loops described by cycles of the graph.

HMSCs may also have an HMSC end, but in this HMSC there are no ends, but this is shown in other diagrams like Figure 15-7 "Parallel merge" (p.15-11).

All together we say that HMSC describes *MSC expressions*. In the sequel we shall see that MSC expressions can be described in two more styles.

Restrictive conditions

From MSC-92 we recall that condition symbols were used to describe “system states”, situations which could be labelled. Informally the conditions were used to combine different MSCs. In MSC-96 the HMSCs and plain MSCs with MSC references are used to formally give the overview of the MSC document and how the individual MSCs are combined.

This leaves some room for the conditions to describe some redundancy. The idea is that a condition in an HMSC describes a requirement for the MSCs referenced in its vicinity.

In Figure 15-3 "HMSC diagram" (p.15-5) there are two conditions and the condition *Idle* is restrictive wrt. the MSC User_accepted by demanding that it must start by the condition *Idle*. Furthermore User_accepted must end in the condition Door_unlocked.

The restrictive conditions of HMSC apply only to global conditions which means conditions which hold for all instances in the MSCs.

A full definition of the sets of initial and final conditions to any MSC or MSC expression is given in Z.120, but it is too lengthy to give here. For the semantic restrictions attached to HMSC conditions see definition.

HMSCs and plain MSCs

HMSCs are meant for overview as the message passing instances are omitted. On the other hand the possible courses of action are more easily seen in an HMSC than in a plain MSC.

Plain MSCs are better when it comes to understanding which instance did what and the details of message passing.

Still it is possible in many cases to choose between the two forms to describe the same reality.

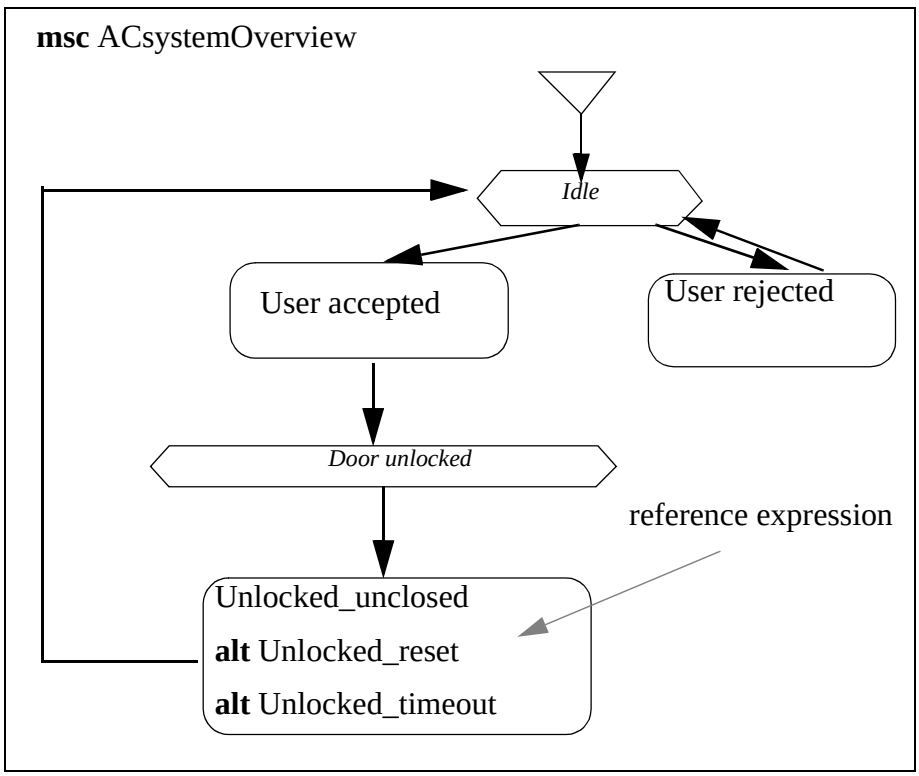
Reference expressions

The HMSC shown in Figure 15-3 "HMSC diagram" (p.15-5) shows that three alternatives may follow the condition *Door unlocked*. Following the execution of one of these three alternatives, the ACSystemOverview will return to the *Idle* condition.

What happens after the *Door unlocked* condition may be seen as one unit which has three alternatives. The unit contains an expression involving MSC references. Naturally we describe such an expression inside a reference symbol by a textual notation. Therefore it is called a *reference expression*.

Figure 15-4: Reference expression

Open figure



In Figure 15-4 (p.15-7) we present an alternative expression as the description of the execution following the *Door unlocked* condition.

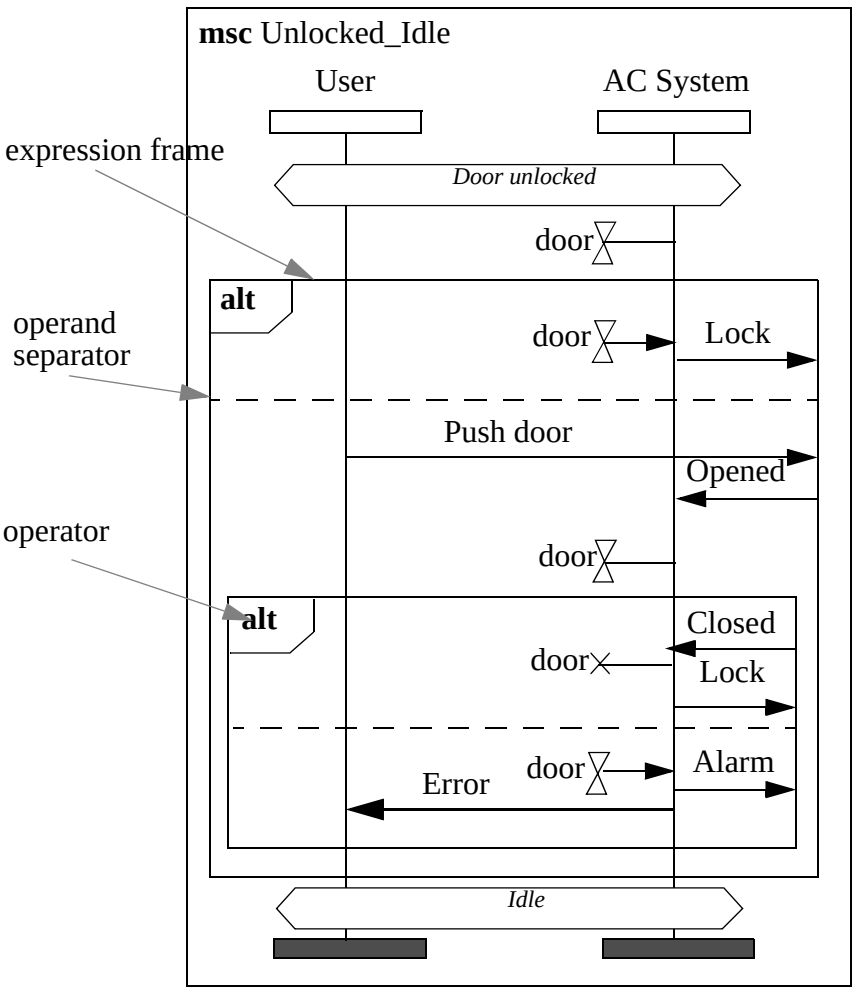
Inline expressions

We have shown how MSC expressions can be described by HMSCs and by reference expressions. It is also possible to describe MSC expressions in plain MSCs.

We will use the same situation as before describing the execution following the *Door unlocked* condition.

Figure 15-5: Inline expression

Open figure



We notice that the expressions are enclosed by an expression frame. The operands are separated by a dashed separation line, and the operator is depicted in the left upper corner of the expression frame.

We see that inline expressions can be nested by simple geometric nesting. Our inline expression in Figure 15-5 "Inline expression" (p.15-8) is not identical to the reference expression in Figure 15-4 "Reference expression" (p.15-7), but it describes the same situation. The reference expression was not nested.

Gate propagation

Some of the messages are sent to or received from the frame. Such messages are known from before as representing gates and this is also the case with inline expressions. The gates are then called expression gates.

As a practical shorthand when frames are nested, propagation of gates is defined. The rule is very simple: whenever a gate is not continued on the other side of a frame, the gate propagates to the next enclosing frame.

In Figure 15-5 "Inline expression" (p.15-8) the gates to/from the expression frames all propagate to the frame of the MSC diagram. In Figure 15-12 "Gates" (p.15-18) we have given an artificial example which also shows the propagation of a message gate.

In principle gates may propagate all the way to the “outermost environment” and represent messages to/from the surroundings of the whole system.

Exceptions and options

The alternative expressions are very common. Among alternative expressions there are two special cases which seem to occur so frequently that they deserve special treatment.

Firstly is the situations where at certain points during the execution there is a choice (or alternative) between a main course of action and an exceptional one. If the exceptional one is followed, normally some kind of recovery or termination is performed, while if the main, normal course is followed the execution may get involved in any complexity of continuations. If we had used the plain nested alternative inline expressions, that would have lead to a large number of nested expression frames. The reader would probably find this cumbersome and confusing.

Therefore MSC-96 defines a special syntax for exceptional alternatives such that either the exception is chosen or the rest of the enclosing diagram is chosen.

Furthermore there are situations where some actions may or may not happen. If they happen, the overall execution will continue after the optional action sequence just like it would without the optional sequence. Such sequences are called options and are equivalent to alternatives where the second operand is empty.

Figure 15-6: Exceptions and options

Open figure

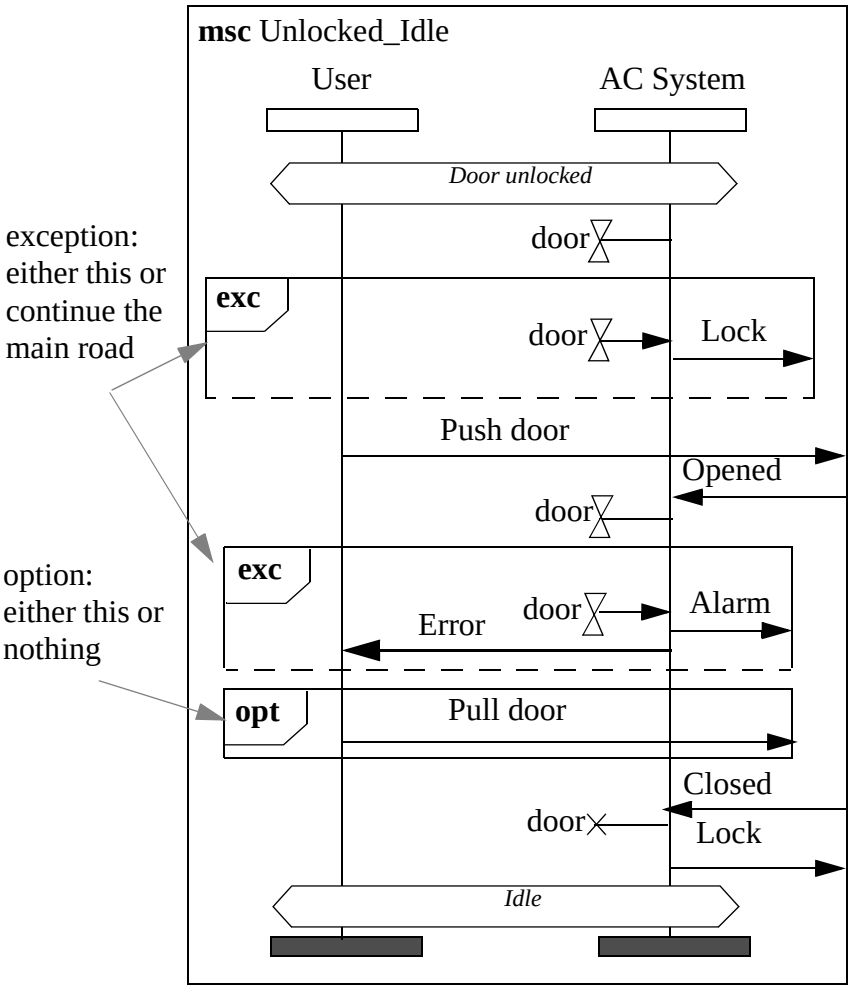


Figure 15-6 "Exceptions and options" (p.15-10) shows basically the same situation as Figure 15-5 "Inline expression" (p.15-8), but we can see that the graphical nesting is not necessary. To show a case of option, we have added an optional pulling of the door as opposed to letting it slam by itself. The branches defined by the exceptions do not have any final condition. To make the situation more equal that of Figure 15-5 "Inline expression" (p.15-8) we should have added the *Idle* condition at the end of both exceptions in Figure 15-6 "Exceptions and options" (p.15-10).

MSC operators

We have given examples of alternative-expressions by HMSC, by reference expressions and by inline expressions. Although important alternative-expressions are not the only kinds of expressions.

MSC-96 offers also parallel merge operator and a loop operator. Implicitly there is also the sequencing operator (which is actually explicit in reference expressions).

Alternative

It may seem obvious that the operator **alt** describes a point of decision where one of several alternative courses of action can be followed. This is also the case as it was pointed out in Figure 15-3 "HMSC diagram" (p.15-5).

However, if the alternatives have a common preamble, i.e. their first part look the same, the point of decision is deferred to the point where the alternatives differ. In practice this distinction of where/when the point of decision is, is of little significance, but formally it plays a certain role.

Parallel merge

Parallel merge is used to describe situations which are independent of each other. The interpretation is that possible sequences are all interleavings of the involved events, such that the partial event orders implied by each of the operands are retained.

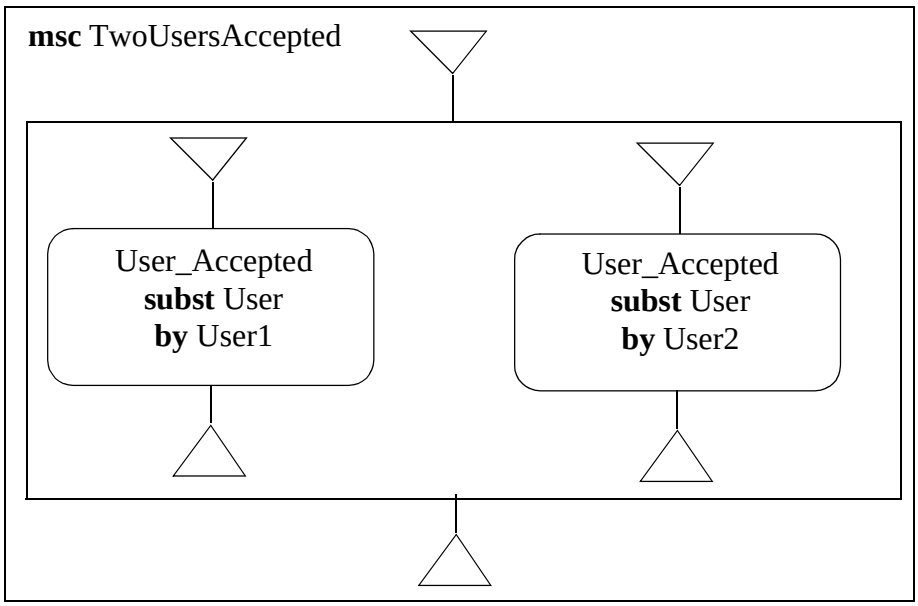
Assume in our example that the Access Control System is approached by two different users at approximately the same time. We have Figure 15-1 "Basic MSC" (p.15-3) which shows a successful approach by one User, but how can the parallel approach of two Users be described?

In Figure 15-7 "Parallel merge" (p.15-11) we have described how two Users in parallel approach the AC System and get accepted.

We have used simple substitution of instance names to make a distinction between User1 and User2 while the AC System is the same instance in both operands of the parallel merge. Substitution is introduced in Substitution (p.15-23).

Figure 15-7: Parallel merge

Open figure



Loop

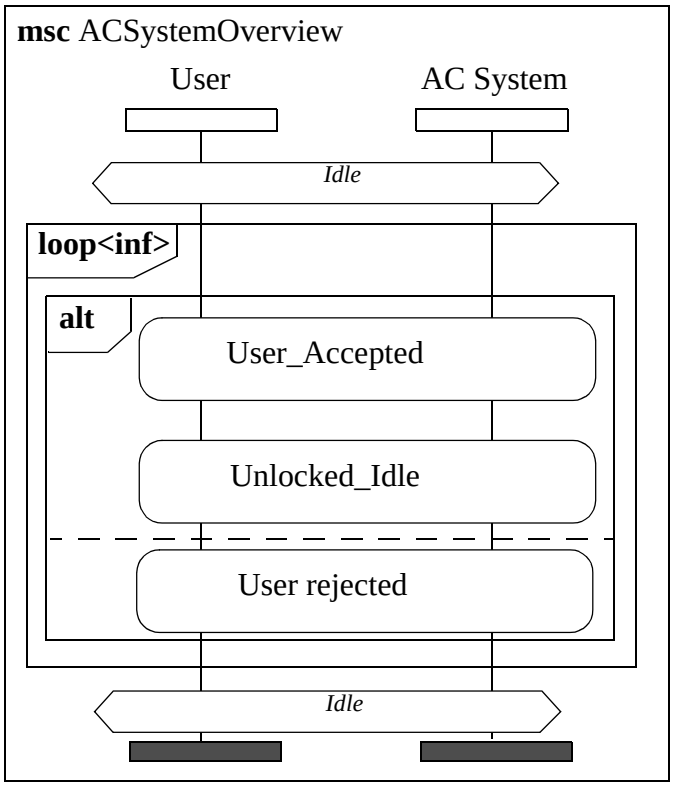
We have mentioned that the HMSC in Figure 15-3 "HMSC diagram" (p.15-5) implicitly describes loops. We may also explicitly describe loops by inline expressions. The equivalent situation is shown in Figure 15-8 "Loop expression" (p.15-12) where also some more information can be included.

The loop operator can be appended loop boundaries in arrow brackets. In Figure 15-8 "Loop expression" (p.15-12) the keyword **inf** designates the loop boundaries. **inf** means that the loop will go infinitely. We could have specified any two natural numbers as a pair of lower and upper bound of repetitions. Either of the two numbers can be **inf** which means infinity. If the upper bound is lower than the lower bound the loop will execute zero times.

If no loop boundaries are given, the default is <1,**inf**>.

Figure 15-8: Loop expression

Open figure



Sequence

Implicitly in the diagrams and explicitly in reference expressions there is a sequencing operator. The reader should notice that the sequencing operator of MSC is the weak sequencing which means the following.

Let A and B be weakly sequenced in that order. Let A and B share the instance I. Then weak sequencing means that all events on I of A will come before all events on I from B. For events on instances which are not shared by A and B the order is arbitrary (just like for parallel merge). And even stronger, if A and B share I and J, still there may be events in B on J occurring before events on I in A!

We show this by an example. In Figure 15-8 "Loop expression" (p.15-12) we have references User_Accepted and Unlocked_Idle which are sequenced. In User_Accepted there is an output event of *Unlock* on instance *AC System*. This may if we follow the definitions closely also appear after the output of *Push door* in Unlocked_Idle. In practice this is also correct as the *User* may very well try and push the door before it is unlocked.

MSC-96 does not include a strong sequencing operator.

The reader should also notice that an MSC reference has only one entry and one exit. Thus there are no ways to continue possible alternatives of User_Accepted into Unlocked_Idle (there are no alternatives in these diagrams here).[1]

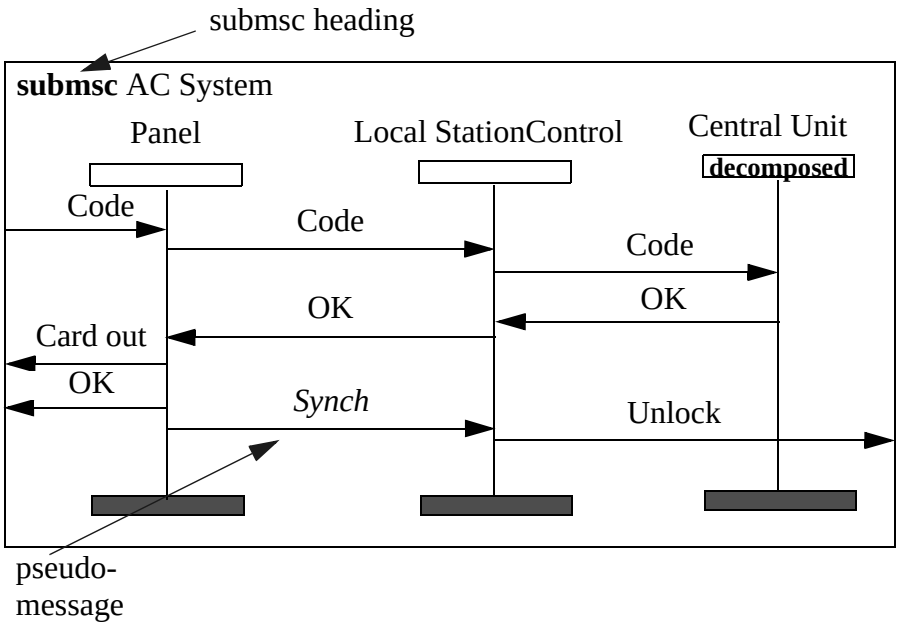
1. The need for multiple entries and exits of MSC references has been acknowledged, but in MSC-96 no solution to the language design was found.

General Ordering

We recall from our MSC-92 tutorial that decomposition requires that the message interface of the decomposing diagram shall correspond to that of the decomposed instance.

Figure 15-9: Submsc

Open figure



In Figure 15-9 (p.15-14) the pseudo-message *Synch* makes the ordering of the interface events strict. This is necessary since the interface is to correspond to an ordering expressed on one instance line. Without the pseudo-message *Synch*, we have in MSC-92 no way to express that output of *OK* precedes output of *Unlock*. In MSC-92 on one instance either a strict ordering or no ordering (coregion) could be expressed.

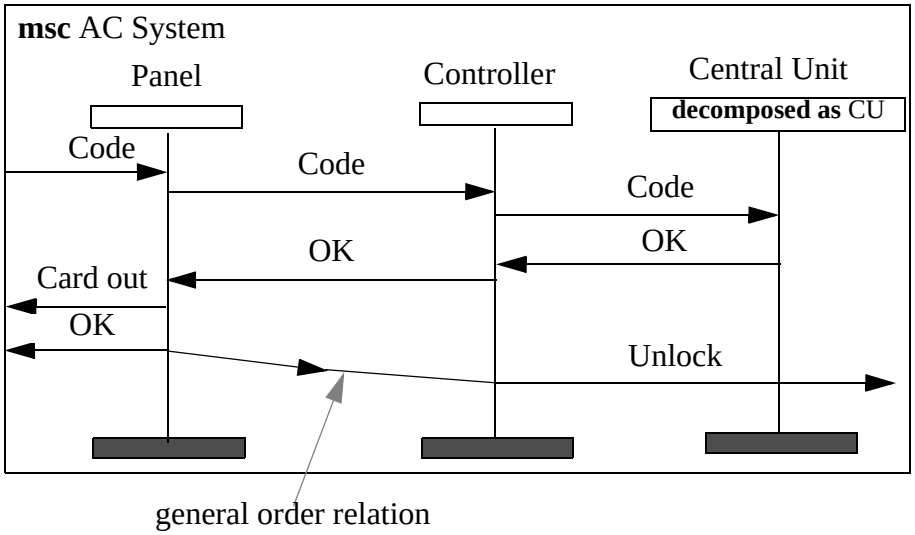
General ordering between events on different instances

In MSC-96 we are more flexible as the general ordering mechanism can be used to express any partial ordering among events. We show the decomposed AC System in the shape of MSC-96.

Notice that the keyword **submsc** has disappeared in MSC-96 and that the diagram where the decomposition can be found, can be explicitly expressed.

Figure 15-10: General order relation

Open figure



Here we have substituted the pseudo-message by a formal general order relation between the output of OK on Panel and the output of Unlock on Local Station Control.

The general order relation expresses that the event at the beginning of the arrow must happen before the event at the end of the arrow.

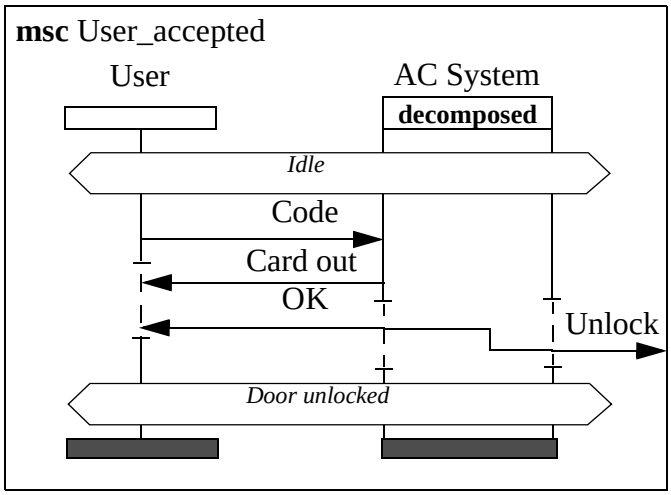
The general order relation symbol is an arrow with the arrow head somewhere between the endpoints of the line. The line which connects the two events may be formed in any shape, it may be curved, jagged or straight.

General ordering between events on the same instance

The general order feature can also be applied within one instance. Then the events connected by a general order relation should be within the same coregion. Sometimes the column form of an instance is well suited when a general ordering is needed within a coregion.

Figure 15-11: General order relation in one instance

Open figure



In Figure 15-11 (p.15-16) we show that there is also another notation within the column form for general ordering where the line segments are either vertical or horizontal and the middle arrowhead is optional. An event A (here: output of *OK*) is before another event B (here: output of *Unlock*) iff[1] the segments from A to B are all non-increasing in height and at least one segment is decreasing.

Notice also in Figure 15-11 (p.15-16) that the decomposition diagram has not bee explicitly specified. The default is that the decomposition diagram has the same name as the decomposed instance.

General ordering between events in different MSCs

General order relations can also end in a gate and thus events in one MSC can be ordered relative to an event in another MSC. Since general order relations have no message name to distinguish them from each other, the order gates must be named explicitly.

An order gate definition is shown in Figure 15-12 "Gates" (p.15-18).

1. if and only if

Gates

In the real world a “gate” is a point of interface between something inside and something outside. The inside may be a mansion or a garden which is well fenced in letting no access be possible other than through the gates. A messenger boy who wants to deliver a message from somebody on the outside to someone inside will have to address himself to the appropriate gate where the message will be taken care of by the employees of the mansion or garden. The messenger will have to rely on the communication lines inside the mansion. He has only made sure that the message has been delivered to the correct gate.

Gates in MSC as shown in Figure 15-12 "Gates" (p.15-18) are very much the same way. For a surrounding MSC diagram, the Mansion is just a reference M where the gates *out_s1* and *h* are known. How the internal communication of M is, has no effect on the communication between the Mansion M and its surroundings. When the messenger boy turn up at gate *out_s1*, he will get an *s1* message which he delivers to instance *k* in the surroundings according to the MSC *G*.

Now it turns out that the surroundings of M is an estate G which has its defined gates *in_s2* and *g* plus the propagated *h* from *M*.

The purpose of gates is to serve as connection points such that messages (and order relations) connected to an MSC reference are associated with the correct message/order relation of the MSC definition referred by the MSC reference.

We have implicitly had a look at two different kinds of gates:

1. Message gates,
2. Order gates.

Furthermore we have seen that there are different categories of where the gates are:

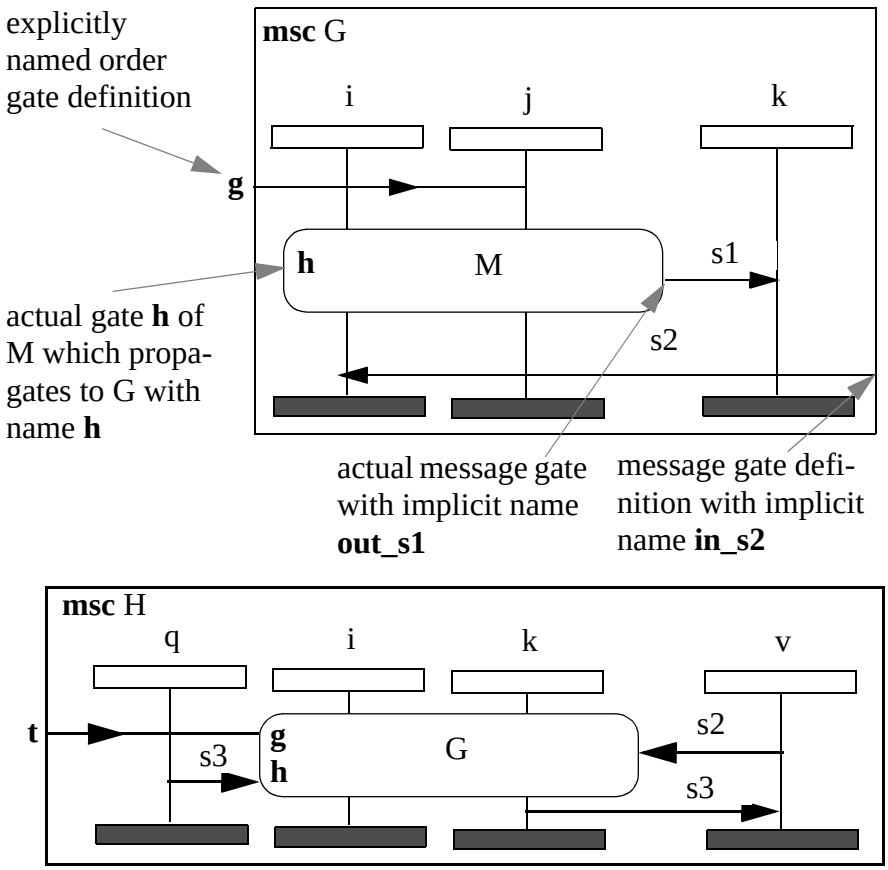
1. Gate definitions on MSC diagram frames,
2. Actual gates on MSC references,
3. Gate definitions and actual gates on MSC inline expressions.

Finally we have mentioned that gates may be explicitly named (in the case of order gates) or they may be unnamed (or actually implicitly named). If a gate is unnamed, it is in fact implicitly named by the message name and the direction of the message through the interface.

Ambiguous gate names are allowed, but actual gates with the same name are considered to be identical. In effect this means that ambiguous gates can be allowed if they represent the interface with the outermost environment.

Figure 15-12: Gates

Open figure



In the MSC H of Figure 15-12 (p.15-18) we see how a reference to G could look. There are some points to notice:

- The gate **h** has been propagated from M through G and applied here.
- The gate **g** is connected to the gate definition t. If we wanted we could have omitted this connection and **g** would have propagated to H with name **g**.
- in_s2 is an implicit actual gate which matches well with the implicit name of gate definition of G.
- The instance *j* is not present in H. The rule is that instances need not be present when misunderstanding regarding event sequences cannot occur. Here also instance *i* could have been omitted.

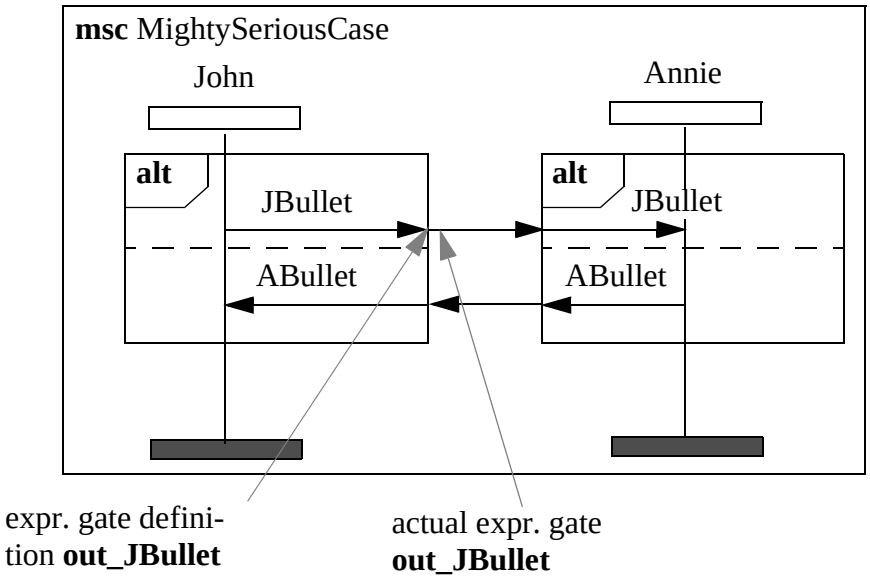
Inline expression gates

In Figure 15-12 "Gates" (p.15-18) there are examples of different variants of expressing gates. We have, however, not presented any inline expression gates. The special thing with inline expression gates is that they are both gate definitions and actual gates. Seen from the inside, the expression gates are definitions, while seen from the outside the gate is an actual connection point.

Alternative-expressions combined with gates lead to some slightly complicated and somewhat counter-intuitive cases.

Figure 15-13: Inline expression gates

Open figure



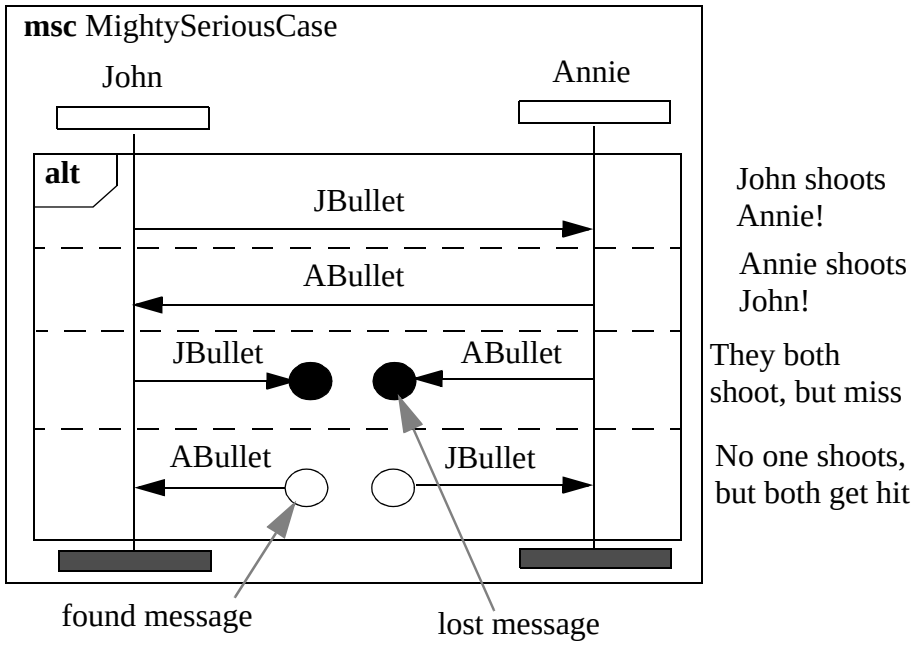
In Figure 15-13 (p.15-19) we have a situation where the gate **out_JBullet** occurs in only one of the alternatives on the left side, and the connected gate **in_JBullet** occurs only in one alternative of the right side.

What is the interpretation of this?

First we should stress that in alternative-expressions, all gate definitions are implicitly thought to be present in every alternative. John and Annie both have two alternatives. In Figure 15-13 (p.15-19) John's alternatives both have the *out_JBullet* and the *in_ABullet*. John's and Annie's alternatives are independent of each other! This means that there are four different cases which form the interpretation of the MightySeriousCase, the duel between John and Annie. These alternatives are spelt out in Figure 15-14 (p.15-20).

Figure 15-14: Interpreting expression gates

Open figure



The reader may wonder why the two last alternatives of Figure 15-14 (p.15-20) are valid interpretations of the MSC of Figure 15-13 (p.15-19). The clue lies in the fact that the two alternative expressions of Figure 15-13 (p.15-19) are independent meaning that selecting one alternative of the left one has no effect on the possible selections of the right one. Since the two expressions have each two alternatives the total number of combinations is four.

The two first alternatives are obvious, the two options where one duelist shoots the other. The third situation may also seem plausible after some consideration, both duelists shoot almost concurrently and the description does not give any room for any of the two being hit because each of them may *either* shoot or get hit, but not both.

The last situation where both get hit, but neither of them shoot, is slightly more difficult to accept. We may all see the western comedy setup where both duelists fall down as a consequence of a nut cracking or some other virtual bullet. Formally in our MSC the situation is a valid interpretation because whether John is hit is *not* dependent upon whether Annie shoots.

This latter case highlights the problem: the actual inline expression gates describe no selection of possible alternatives, they merely connect incomplete messages and order relations.

MSC-96 does not give any mechanisms to define “guards” for different alternatives. A “guard” for alternatives could be that “to get hit, the corresponding shot must have been fired”.

Having analyzed the MightySeriousCase and the formal understanding of what the first MSC actually described, we come to realize that John (and symmetrically Annie) did not have only two choices, either to shoot or to get hit. There are more possibilities, at

least to shoot and to get hit, and possibly also to get hit before shooting. This may serve as an example of how thorough analysis based on the formal semantics[1] may clarify shortcomings of a specification.

1. We applied knowledge of how the formal semantics would have handled this case.

Incomplete Messages

The MSC Figure 15-14 "Interpreting expression gates" (p.15-20) which spelled out the interpretation of the Mighty Serious Case also showed the need for explicit description of incomplete messages. Incomplete messages are messages where either the output or the input are absent or unknown.

There are two different kinds of incomplete messages, found and lost messages. Again the lost messages are much easier to accept than found messages, but mathematically as well as pragmatically they are symmetrical.

Lost messages are depicted by a “black hole” at the arrow head symbolizing the black hole into which the message disappears. The black hole may also be associated with an identifier which indicates the instance or gate which was the target of the message, but to where it never arrived.

Found messages are symmetrical and depicted by a “white hole” symbolizing a source of new messages.

That messages disappears, i.e. that no input can be observed even though an output has been issued, is commonplace in communication systems. Almost equally commonplace are incidents of noise and electronic errors such that messages are received which no instances agree to have sent. We all know the car alarms that go off all the time indicating that some alarming message has been received even though no alarming incident has actually taken place.

When a message has been lost, we may or may not know to where it was heading. MSC-96 offers the possibility to associate an instance identification or gate name with the black hole to indicate the supposed target.

Symmetrically an instance identification or gate name may be associated with the white hole.

In subsequent phases of the specification such found or lost messages may be more closely examined and described. The reason for the sudden alarm (found message) may be due to improper tuning of the sensor. Such closer examination causes a more precise (and sometimes more detailed) description which on a more coarse level is not needed.

Substitution

Every description language with high ambitions must have mechanisms to handle generalizations. For MSC-96 the gates are means to obtain flexible connections between MSCs, but still the insides of the MSCs are fixed.

Substitution is a way to make MSCs more general as every MSC can be seen as a pattern where message names and instance names as well as MSC names can be exchanged. Substitution is comparable to parameterization, where any message name, instance name or MSC name are formal parameters.

Substitution in MSC-96 is similar to, but not equivalent to, macros. Only specific semantic units are substituted such that the basic semantics of an MSC is not drastically changed by a substitution as may be the case with macros where any lexical unit is eligible for substitution.

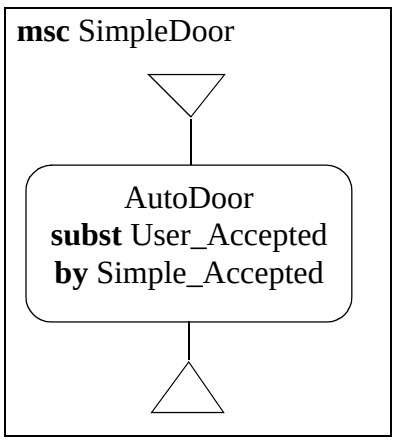
Substitution of message names and instance names is very parallel to context parameters of SDL or type parameters of C++. The overall semantics is not changed, but the MSC may appear in different contexts when the substitution has been applied.

Substituting MSC names simulating object orientation

Substitution of MSC names on the other hand resembles virtuality of object orientation. Internal behaviour of an MSC is changed when an MSC name of an MSC reference is changed (substituted).

Figure 15-15: Substitution

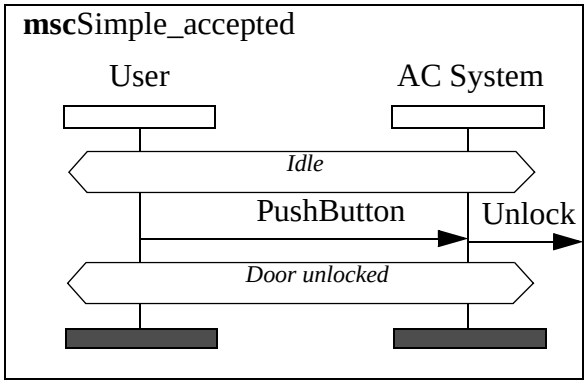
Open figure



In Figure 15-15 "Substitution" (p.15-23) we have the most trivial case of MSC name substitution. In AutoDoor (see Figure 15-2 "MSC reference" (p.15-4)), the somewhat complicated User_Accepted MSC is used by an MSC reference. In the SimpleDoor the User_Accepted is substituted by the very simple Simple_Accepted.

Figure 15-16: Simple Accepted (substitutee)

Open figure



In Figure 15-16 "Simple Accepted (substitutee)" (p.15-24) we see that the unlocking of the door is due to a pushing of a button rather than a lengthy protocol of card insertions and PIN typing.

In Figure 15-15 "Substitution" (p.15-23) we also see an example of an HMSC with an end symbol.

In object-oriented terms this situation would have been modelled by specifying User_Accepted as virtual in AutoDoor. Then a specialization of AutoDoor named SimpleDoor would be specified to inherit from AutoDoor, but redefining User_Accepted. The Simple_Accepted MSC in Figure 15-16 "Simple Accepted (substitutee)" (p.15-24) would constitute the body of the redefined User_Accepted in SimpleDoor.

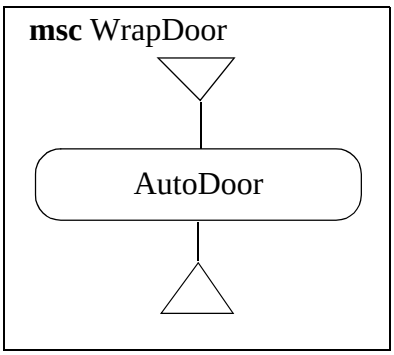
There may be different opinions about whether a more pure object-oriented notation would have been better than the substitution notation. Substitution is more flexible as virtuality is not specified in advance and it is more in harmony with the substitution of messages and instances. On the other hand the concept hierarchies of object orientation are not so easily conceived.

Substitution propagates through MSC references

To fully understand the effects of substitution, the reader should bear in mind that substitution carries on to the MSCs being referred. In our example case, this is of no significance, but in other cases the designer should keep this effect in mind.

Figure 15-17: Substitution propagation

Open figure



Take Figure 15-17 (p.15-25) as a start:

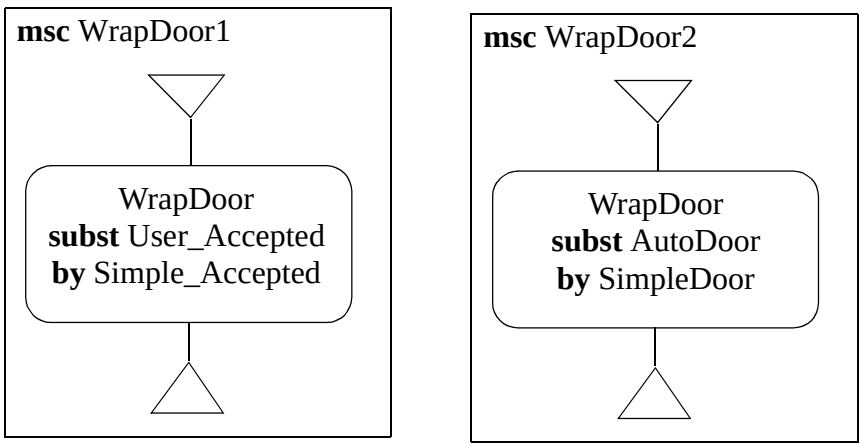
It has no specific meaning other than for showing the propagation of substitution to wrap the AutoDoor with one more reference layer.

Now we apply two different substitution strategies.

In Figure 15-18 (p.15-25) the WrapDoor1 substitution will propagate down through the reference to AutoDoor and as such become equivalent to substituting AutoDoor by SimpleDoor as shown in WrapDoor2

Figure 15-18: Substitution propagation (2)

Open figure



Substitution restrictions

The reader may already have spotted some possible restrictions to the substitution scheme.

Message parameters

Due to time pressure during the finalizing of MSC-96, substitution of message parameters was not properly addressed. The syntax restricts message substitution to replace message names by message names. The parameters are not mentioned. This will probably be modified shortly, but for the official Z.120 this restriction holds.

MSC expressions

It is also very tempting to substitute an MSC name by an MSC expression. For lack of time and due to syntactic problems this has not been allowed. MSC substitution is limited to replacing an MSC name by another. But there is also the special predefined name of an empty MSC, namely **empty** which can be used as either source or target for the substitution.

MSC-96 – its benefits and challenges

Benefits

MSC-96 gives you better overview

Overview diagrams in the form of HMSC has been formally introduced giving the designer and the readers better overview of the overall structures of an MSC document.

Inline expressions have been introduced to give a better overview of small variations within plain MSCs.

MSC-96 offers improved ways to combine MSCs

MSC references have been included to let MSCs have more layered structure. Combined with gates, MSC references offer a flexible way to reuse MSCs and to connect them.

MSC reference expressions give a flexible and compact way to express variability with plain MSCs.

MSC-96 offers improved generalization mechanisms

MSC gates offer a flexible and practical interface definition mechanism. With the rules for implicit naming of gates and their propagation to enclosing MSCs simple MSCs become powerful building blocks in larger contexts. Even old MSC-92 diagrams become building blocks with gates when referenced in MSC-96 diagrams.

Substitution adds more generalization power. Object orientation and parameterization may be simulated.

MSC-96 gives you added expressive power

The general ordering feature makes it possible to express any partial order of events.

Challenges

The biggest challenge of MSC-96 is to prove its worth in specifying more complete and more precise reactive systems than was possible by MSC-92 without appearing complex and difficult to learn and read by its users.

Another challenge is to get good tools early enough to prove their worth before MSC-96 has become obsolete.

It is our hope that MSC-96 may serve well as a vehicle for requirement engineering and as a communication language for different groups of people involved in real-time reactive systems.

Strongholds and shortcomings

MSC can be used as a communication means between different types of personnel. MSC-92 has (and MSC-96 will have) a formal semantics. MSC is supported by powerful CASE tools. MSC is being used by a number of users in a number of large projects throughout the world.

MSC-96 offers strong structuring mechanisms without sacrificing its simplicity.

MSC describes message interaction. Describing algorithms or database structures is impractical in MSC.

Expressing explicit time and duration cannot be handled properly. Still structured comments may provide helpful additions to MSCs to describe such time requirements.

MSC has no mechanisms to express temporal logic such that safety and liveness requirements can only be described via MSC if extra information and special interpretation of MSC is applied. When MSCs are interpreted as requirements it is often difficult to know whether they express requirements to the system (the interacting instances) or to the environment.

MSC-2000

It may look strange to talk about MSC-2000 before all MSC users know about and use MSC-96, but any language which is considered finished, is a dead language. The MSC group of ITU SG 10 has pointed out a series of important aspects of the language which are being studied.

Non-functional properties

Many requirements of reactive systems include duration requirements or limits on performance such as error rates etc. MSC-2000 should be able to express these aspects as a part of the language. There has been some attempts in international projects to come up with some notation, but none of them were mature enough to be included in MSC-96.

Methodology

MSC has become popular in many different application areas. MSC together with object-oriented development is a hot issue to consider. It may be reasonable to add to or modify the language to accommodate for effective software engineering methods. Furthermore test case generation from MSC (and SDL) has been done. It may be desirable to let MSCs include test verdicts or other evaluative descriptions. The intimate relationship between MSC and SDL should also be studied e.g. to produce a common semantic base applicable for effective consistency checking.

Data

MSC has no formal data concept. If MSCs are used more constructively, the need to express data more precisely will be of absolute importance.

Grammar

MSC-96 has (like MSC-92) both a graphical and a textual grammar. There is no abstract grammar in MSC-96 Z.120 as it was in MSC-92 and the informal semantics is given relative to the textual grammar. The formal semantics will also take the textual grammar as its starting point.

Experiments have been done to see if the graphical grammar could be made more formal. The MSC-96 graphical grammar has only informal descriptions of spatial relations. If the graphical grammar can be made more precise the need for a textual grammar which is different from a mere translation from the graphical one seems to disappear.

Condition Conditions in MSC-96 have improved relative to the definition in MSC-92, but still there are obvious improvements which have been requested. A strong global condition concept which is different from a condition where all instances are included “by accident”. Furthermore many users have wanted general predicates in conditions rather than a list of labels as MSC-96 offers.

Others Other language issues which have been raised, but which have not found its conclusions are:

1. remote procedure,
2. synchronous communication,
3. grouping of instances,
4. hierarchy of messages,
5. additional MSC operators such as disruption and interrupt,
6. total ordering of events,
7. gates in HMSC.

MSC-96 Methodology

MSC-96 is a language which supersedes MSC-92 wrt. expressiveness and power. Still the standard interpretation is that an MSC document represents a set of message sequences which represent *possible* sequences in the system under consideration. In the MSC-92 methodology we introduced MSC documents where the interpretation was that the message sequences were *not possible*. With MSC-96 we may in some cases introduce the third interpretation that the MSC document covers *all possible* sequences which may happen in the system under consideration.

Also with MSC-96 the company strategy and the categorization of the MSCs are important for the awareness and focusing of the MSC production. This issue was properly covered also in the MSC-92 methodology.

In this MSC-96 methodology we shall focus on the adaptation of the general property modelling technique when using MSC-96 as the vehicle for formal description.

Making more precise descriptions

Assuming that there is already some description of the service in prose, we would like to formalize this in MSC-96. How do we go about doing it?

Even though MSC-96 is a formal language, MSC-96 diagrams may have comments and they may be annotated by informal, but important prose. How can the amount of important, informal information be decreased or eliminated? By eliminating such informal information more formal validation techniques can be applied.

Formalize With the prose description as starting point, make MSCs which have the active objects of the prose description as instances. In the domain property model, such active objects are roles.

Model communication actions by messages. Give rather course sketches of the message sequences possibly leaving out all messages or actions which may blur the overview picture.

Narrow The general operation of narrowing is to restrict the possible interpretations of the description. MSC-96, however, is a language which are very explicit about which sequences it covers. While in other languages it may be difficult to overview the runs covered by a certain construct, MSC-96 is more intuitive. The MSC diagrams express explicitly the covered runs. Therefore narrowing is not very applicable with MSC-96.

One possible act of narrowing may be the adding of more general ordering relations in a coregion.

Supplement Supplementing, on the other hand, is very applicable. Since an MSC may describe a (more or less) finite set of sequences, more MSCs will describe a larger set of sequences.We will often start by describing some normal cases, and thereafter some exceptional and erroneous cases will be described.

In MSC-96 the erroneous and exceptional cases can often be described as additions to the MSCs where the normal runs are described. Through alternative constructs like **opt**ion and **exc**eption, normal MSCs can be enriched into covering all legitimate runs.

Making more detailed descriptions

Even when we have a formal MSC-96 document, we may have reasons to go into greater detail. The instances may consist of smaller instances, the messages may actually be a whole protocol, and the MSC references must be resolved by defining the MSC diagrams which they refer to.

Furthermore when the magnifying glass is applied, entirely new aspects may be revealed which on the more coarse level were insignificant, but which on a more detailed level proves to be significant.

Decompose

Decomposition in MSC is meant to be achieved by the **decomposed** mechanism where an instance of one MSC diagram (called the “decomposed instance”) is spelt out in greater detail in another MSC diagram (called the “decomposition diagram”) (See Figure 15-11 "General order relation in one instance" (p.15-16) and Figure 15-10 "General order relation" (p.15-15)).

Even though the principle of decomposition is a simple one and seem to be well supported in MSC-96, there are some points which should be carefully handled.

1. The engineer should be aware of how well the tool supports decomposition (Tool support (p.15-31)).
2. The environment of a decomposition diagram should be exactly the boundaries of the decomposed instance. Instances of the upper level MSC containing the decomposed instance should not reappear in the decomposition diagram (MSC environment (p.15-32)).
3. Decomposition of instances and MSC references are orthogonal, but this orthogonality must be carefully expressed (Decomposition and MSC references (p.15-33)).
4. Conditions of the upper level diagram should be matched by corresponding conditions in the decomposed diagram (Decomposition and Conditions (p.15-34)).
5. Make sure to let decompositions define one tree structure of instances (The instance hierarchy (p.15-35)).

We shall address these questions one by one and give our advice.

Tool support

The support for decomposition in tools may vary from little or no support to advanced consistency warnings. It is important for the engineer to know what language rules and methodological guidelines he will have to check himself without the aid of the tool. Here is a checklist for tool support:

- *Support for decomposition diagrams* at all. Unfortunately this is not automatically true.

- *Static check of messages* to/from the decomposed instance compared with the corresponding communication in the decomposition diagram.
- *Dynamic check of the communication*. This means to make sure that the sequence described on the decomposed instance in the upper level diagram is *exactly* the sequence resulting from the communication within the decomposition diagram.
- *Check of aggregate hierarchy of instances*. The tool may check that there is an underlying tree structure of decomposed instances. See The instance hierarchy (p.15-35) for a more thorough explanation.

MSC environment

MSCs are used to describe interaction between instances. It may seem arbitrary which instances are considered “in the environment” and which instances are considered “inside the situation”. Often we would prefer to let many instances appear inside the situation where many individual entities of the environment can be described by individual instances rather than using the frame to describe the environment.

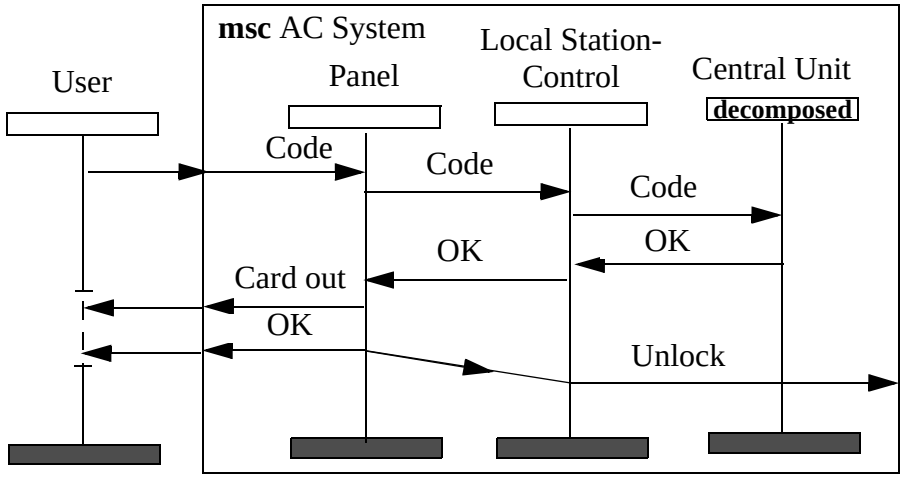
In Figure 15-10 "General order relation" (p.15-15) we may wonder from where the *Code* message comes and to where the *Card out*, *OK* and *Unlock* messages are to be sent. We could have been tempted to including the User instance also in the MSC AC System. What we then would have done is to expand the AC System instance in a new MSC, and if both the old MSC and the new MSC are in the same MSC document, there is an overlap of situations which should be kept consistent, but where MSC as a language does not help.

Our firm advice is never to repeat instances in decomposition diagrams from upper level diagrams, and rather to use messages to/from the environment and supply the gates with adequate names or comments.

For tool makers it could be a good idea to provide possibilities to draw instances outside the diagram frame as shown in the non-standard diagram Figure 15-19 (p.15-33). Such instances would constitute gate end-point constraints as we find in SDL. This option should be attractive also for those who make their MSCs with a general drawing tool and not a dedicated MSC editor.

Figure 15-19: Gate endpoint constraints in MSC (dialect)

Open figure

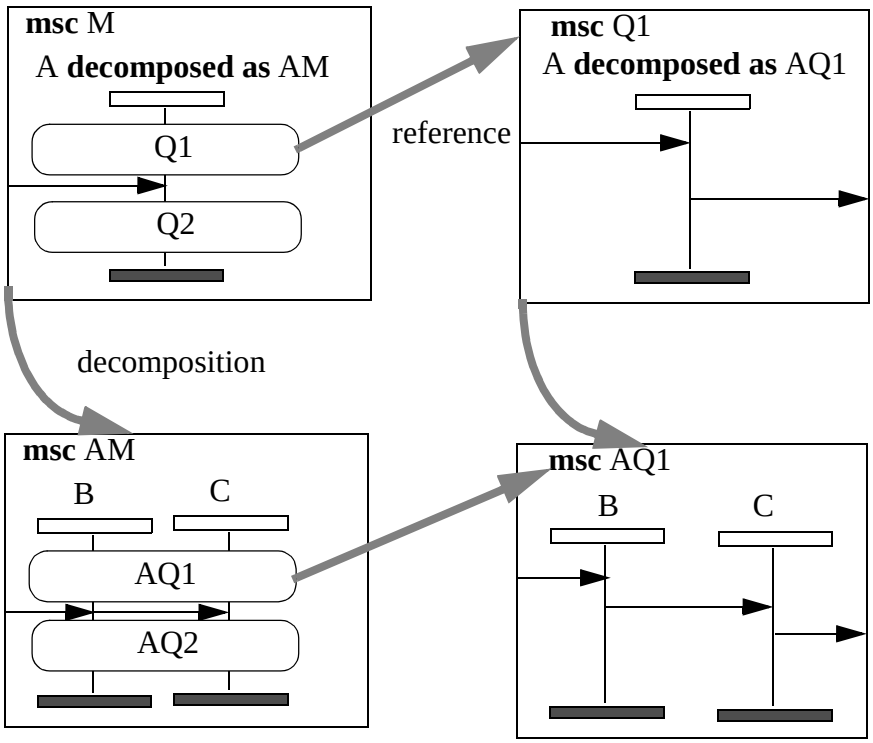


Decomposition and MSC references

Decomposition combined with MSC references will quickly demand a faithful methodological approach in order to keep the decompositions consistent with the MSC references. In Figure 15-20 (p.15-34) we present schematically the principle for consistency between decompositions and MSC references.

Figure 15-20: Decomposing MSC references

Open figure



The following principles hold. Whenever an instance which is covered by an MSC reference is decomposed, the decomposition should show the same structure of MSC references as the decomposed instance. The MSC references of the decomposition refer to decompositions of the instance of the referred MSC. A proper naming convention of MSCs should be provided such that the name itself describes the hierarchy of decompositions which it represents.

By following this fairly simple principle, there is no need for a more complicated consistency control procedure.

Decomposition and Conditions

Conditions and decomposition are problematic only because conditions themselves are still somewhat problematic. The main area of concern is the scope of the condition. A condition is defined by its name *and* its covered instances. When a condition covers all instances of an MSC, is it then “global” meaning that it can be considered to cover *all* instances in the whole MSC document? The MSC language definition hardly gives an adequate answer to this, but a normal interpretation is that it is a global condition if every instance in all MSCs it appears are covered. What if instance A is decomposed into an MSC with instances B and C. Does a global condition covering A also cover B and C? From what we earlier said, the reasonable interpretation is that the global condition also covers B and C (since it is supposed to cover all instances). Then a reasonable requirement is that global conditions of the decomposed instance should reappear in the

decomposed MSCs. For verification and consistency purposes the global conditions are the interesting ones, while for MSC to SDL conversion purposes local conditions can also be used favorably.

A local condition of a decomposed instance should reappear in the decomposition diagram as a condition shared by all the instances of the decomposition.

The instance hierarchy

Finally we address the need for some consistency between different decompositions of the same instance in different MSCs. The decomposition of an instance represent a definition of part of the aggregate hierarchy of instances. In Figure 15-10 "General order relation" (p.15-15) the instance *AC System* has been decomposed into instances *Panel*, *Local Station Control* and *Central Unit*. In Figure 15-20 "Decomposing MSC references" (p.15-34) the instance *A* is decomposed into *B* and *C*.

Our first question is whether all decompositions of an instance must contain the same set of sub-instances? In principle it may be possible to find cases where different component sets could be feasible. In a situation where the actual implemented actors are even smaller, intermediate levels may be aggregated in different ways. In our methodology, however, we hold the view that there is *one* underlying aggregate tree-structure of instances.

The second question then is whether all decompositions must show exactly the same set of instances. Our methodological answer to this is that as long as one underlying aggregate structure can be deduced from the decompositions of the whole MSC document, there is no need for instances which are not involved in the communication to be shown in decompositions.

An example of this may be the situation where in our *Access Control* system, the *User* will always escape from the inside to the outside of the *Access Zone* by only pressing a plain button which will unlock the door. This situation can be described with the *AC System* as one instance communicating with the *User*. In this situation, however, a decomposition of *AC System* need not show *Central Unit*, because the *Central Unit* is not involved in this situation at all. Still the underlying aggregate structure will have *Central Unit* as one component of *AC System*.

The designer should also be careful not to skip levels in the decomposition which would make it more difficult to deduce the underlying aggregate hierarchy.

The aggregate hierarchy of instances should match the aggregate hierarchy of the corresponding object model.

Breaking Down

While we in Decomposition defined a hierarchy of instances, breaking down means to define a hierarchy of communication concepts or protocols. There are two MSC concepts used for this: messages and MSC references.

There is no mechanism in MSC to break down a message. Still we all know that messages are on different levels often modelled by the OSI layers. While low level communication may be necessary to achieve a detailed understanding, the more upper level messages are better for coarse overviews.

Since there is no language mechanism to handle this, we must distinguish between a set of different cases.

1. The MSC document as such may be divided into layers which match protocol layers such that broken down messages do not occur in the same MSC document as the aggregated message (Layers of MSC documents (p.15-36)).
2. A message (type) of one MSC may be broken down by an MSC. The original message may be considered an MSC reference to the result of the breaking down (Message as MSC reference (p.15-36)).
3. A message (type) of one MSC can be broken down be an MSC, but the original message cannot be seen directly as an MSC reference (Messages as merged protocols (p.15-37)).

Layers of MSC documents

These cases require different description techniques. The first case where the MSC document is layered into separate new MSC documents can be applicable in situations where the layers are for very different use. Possibly the upper layers are used for marketing and documentation, while the lower ones are used in design. Their inter-consistency is preferable, but not vitally important. Documentation MSCs may “cut a few corners” without violating the overall principles and spirit of the solution.

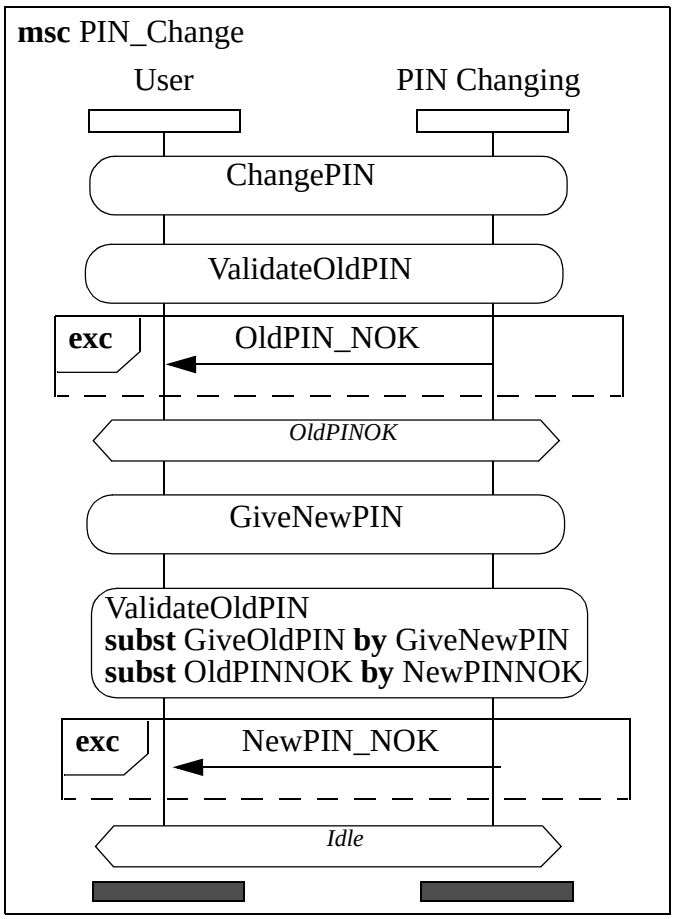
Message as MSC reference

When the message can be understood as an MSC reference, this is exactly what we advise to do: substitute the message by an MSC reference to the broken down protocol. Unfortunately the substitution mechanism in MSC cannot be used since messages can only be substituted by messages. The change must normally be done manually for all places where this message (type) occurs. The resulting MSC document keeps the layered structure, both overview and detail are taken good care of. The only disadvantage compared with the MSC document strategy above is that the direction of the communication primitive is lost since MSC references have no direction.

In our example there is an MSC for PIN_Change which include a message ChangePIN which is broken down in the MSC ChangePIN. The merged MSC diagram is shown in Figure 15-21 "Modified MSC diagram" (p.15-37)

Figure 15-21: Modified MSC diagram

Open figure



The transformation from message to MSC reference may of course also have effect on decomposition as pointed out in Decompose (p.15-31).

Messages as merged protocols

Technically there is an important difference between messages and MSC references. While messages may overtake other messages (see message overtaking in MSC-92), no such thing is defined for MSC references. If we have two messages which are involved in message overtaking and they are both subject to breaking down, the final result is not obvious. While a message has one sender and one receiver, a protocol may have messages going both ways. Extra instances may also be introduced in the breaking down (see also Reveal (p.15-38)). The actual meaning of breaking down two such messages *may* be to allow the parallel merge of the two protocols. This is normally not the case, however, that the freedom is that wide. Rather the fact is that the detailed meaning must be spelled out manually for that specific situation. In the descriptions broken down messages must be substituted not by an MSC reference, but by the content of the MSC diagram which defines the breaking down. The message overtaking of the original indi-

cates a “personal” merge of the two diagrams for the situation. The original messages will become historical (forgotten) if the MSC document layer approach cannot be applied.

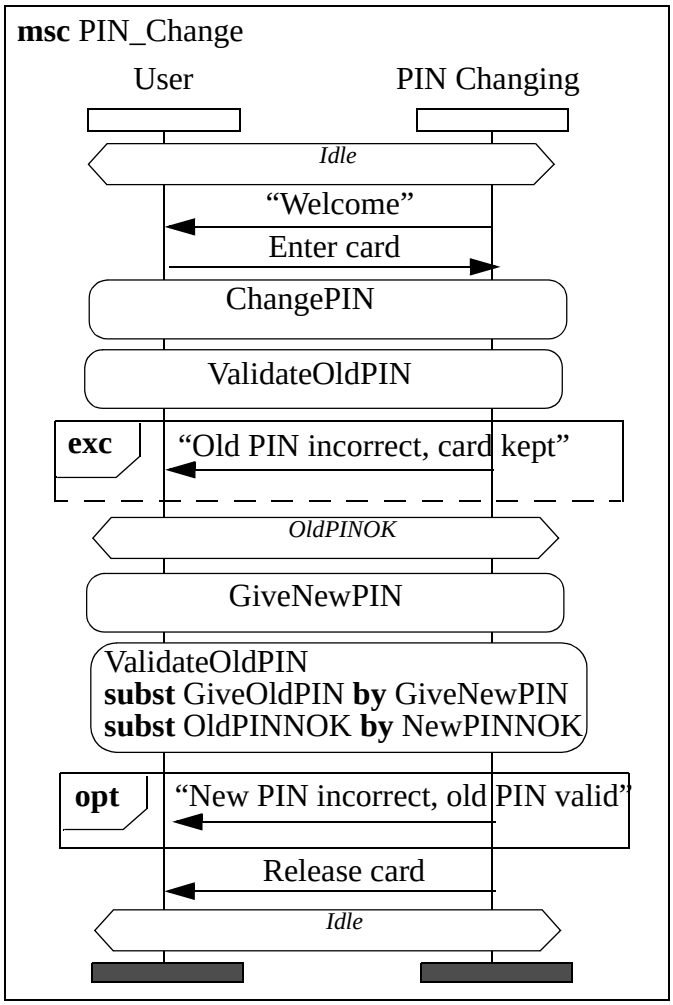
Reveal

As pointed out in the general property modelling principles, we seem to have overlooked the entering of the card (and its return). This may be considered revealing significant new aspects which has not been covered by decomposition and breaking down.

Having revealed the entering and returning of the card, the modified diagram appears in Figure 15-22 (p.15-38)

Figure 15-22: Revealed ‘card’ in PIN_Change

Open figure



Distillery

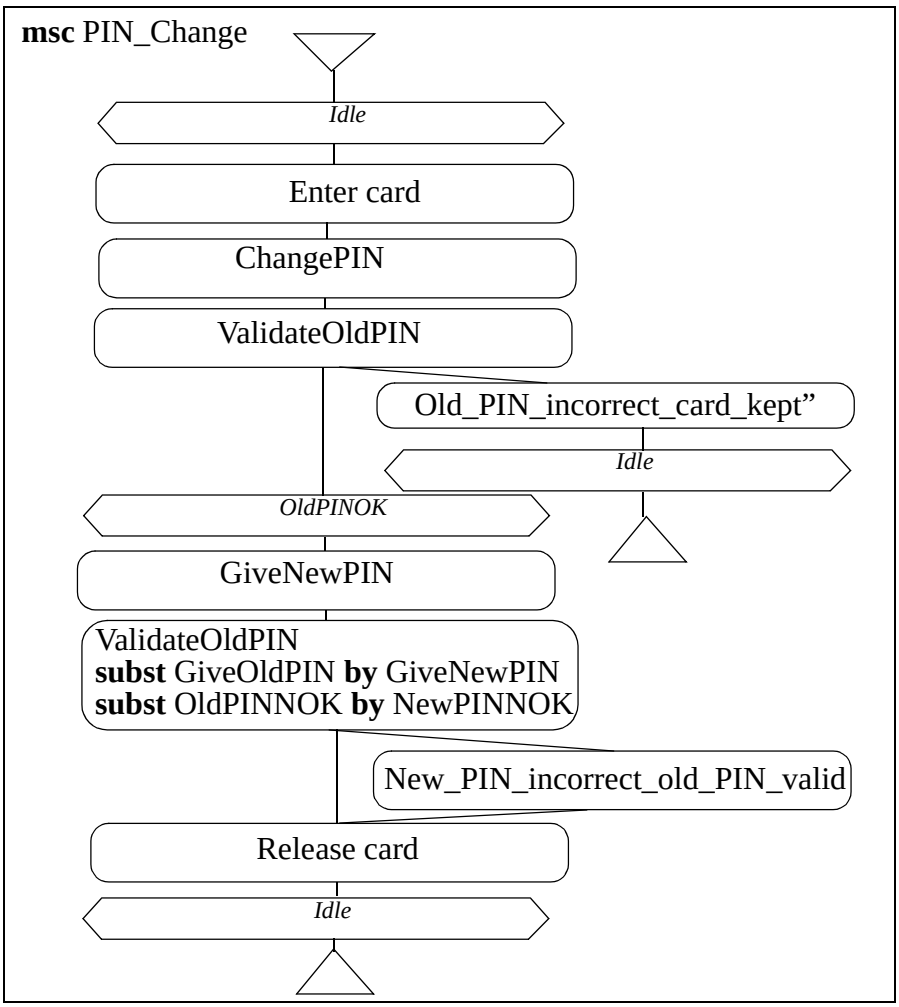
Having applied the different techniques of refinement, we come to a point where we want to organize the description such that it appears layered. MSC has two orthogonal layering mechanisms: MSC referencing and decomposition. The finalizing of the description also means to make sure that the use of these layering mechanisms are optimal wrt. the problem.

In Figure 15-22 (p.15-38) we have mixed MSC references and messages. It is a question whether we should wrap the few messages in MSCs such that this top level overview MSC consists of merely MSC references representing individual, but fairly high level concepts. Such concepts lend themselves well to substitution when variants of the service shall be devised.

Having done so, we are left with an MSC with MSC references and with instances. We should then consider whether it gives an even better overview and future flexibility by letting this top level MSC become an HMSC where the instances have disappeared. In Figure 15-23 (p.15-39) we can see what this could look like.

Figure 15-23: HMSC of PIN_Change

Open figure



MSC-96 in domain modelling and design

There are differences between domain modelling and design modelling. In this section we shall see how these differences manifest themselves in MSC.

The biggest technical difference between the two kinds of modelling is the entity orientation. During the domain modelling, the focus is service- and role-oriented, while during design, the focus is service, but object-oriented.

Roles (in the domain) are often service providers. Since their nature is to be played by some object (in design) defined through a “synthesis” process, it is not so usual to decompose roles into “sub-roles”. Consequently decomposition is an activity which is not much present during domain modelling.

On the other hand, during design, decomposition becomes very important. Furthermore there should be close relations between the domain model and the design model. The services identified during the domain modelling should reappear in the design model. Since roles may be played by different classes of objects, the corresponding service MSCs should apply in different contexts. For this we may preferably use substitution of instances where the role instances are substituted by object instances.

When we combine decomposition with substitution, we realize a challenge for the commutative scheme of Figure 15-20 "Decomposing MSC references" (p.15-34). On one hand we want to describe the service preferably one place (in the domain model where instances are roles), but on the other hand we want to decompose the instances (objects) of the design model which appear from substituting roles by objects. We have the choice of either breaking the commutativity scheme or to decompose the substituted roles and in turn substitute these sub-roles. The latter strategy is the preferable one as it keeps consistency throughout the descriptions.

Methodological Rules for the description by MSC-96

We have presented the principles for using MSC-96 in a stepwise fashion. In this section we summarize our findings by giving explicit guidelines and rules for the design.

Formalizing

- *Service orientation*. Make one MSC per service.
- *Role orientation*. Instances of the MSCs in the domain are roles.
- *Normal cases*. Focus first on the normal cases and make them formal.

Narrow

- *Actions and comments*. Try and minimize the use of informal text which is actually meaningful on its own. See if actions and comments can be expressed (also) by messages.
- *General ordering*. Scrutinize coregions and make sure that the allowed variability of sequencing is the desired one.

Supplement

- *Exceptions and errors*. Supplement the normal cases by cases expressing exceptional and erroneous situations. Use alternative-, exceptions- and option- mechanisms of MSC-96.

Decompose

- *Environment*. Use the frame to describe the environment for better reuse capabilities. Use good gate names or comments to describe the connection points. Informally instances outside the frame can be utilized (non-standard MSC).
- *Underlying aggregate hierarchy of instances*. Use the decompose-mechanism to define the aggregate hierarchy of instances. The hierarchy should be one tree structure. Do not skip aggregate levels in the decomposition. The instance tree structure should match a corresponding structure from the object model.
- *Decomposition consistency.* In order to keep a simple consistency between decompositions and MSC references the following principles should be kept:
  - The structure of MSC references on an instance A in MSC M shall be retained in the decomposition of A.
  - MSC references in the decomposition of A refers to decompositions of A in diagrams referred to by MSC references on A in the original MSC M.
  - Thus MSC references and decompositions make up a commutative scheme as shown in Figure 15-20 "Decomposing MSC references" (p.15-34).
- *Conditions*. Use global conditions to describe important system states. These will formalize restrictions on component MSCs when used in HMSC diagrams. Whenever a global condition covers a decomposed instance, the condition shall also appear in the decomposition.

Breaking down

- *Layers of MSC documents*. When the description of the services are made on very different abstraction levels for very different purposes (and possibly for very different people), it is possible that keeping the ultimate formal connection between the MSCs requires too much effort. The solution to this is to make more than one MSC document, but where each MSC document represents a complete understanding by itself. The relation between MSC documents is established informally or by a separate MSC document describing the mapping between message types in the upper level MSC and protocols (MSCs) on the lower level.
- *Messages become MSC references*. Sometimes what appears as one message turns out to be a somewhat more complicated protocol. Such messages may be substituted by MSC references to a diagram showing the protocol.
- *Messages are expanded*. When the messages which turn out to be protocols cannot be substituted by MSC references due to e.g. message overtaking or other sequence merge problems, the messages should be expanded by the contents of the protocol. The merge problems must be handled manually.

Reveal

- *"Under the carpet"*. Consider details which have been pushed aside in earlier phases. As the richness in detail has increased, the significance of "forgotten" details will also increase. Reconsider the aspects which have been pushed under the carpet.

Distillery

- *Layering*. The purpose of the distillery is to make sure that the descriptions are organized in a layered manner. The upper layer should be such that it can be understood by itself in its own universe of concepts. The relation which defines the layering should be explicit and well defined. In MSC-96 MSC references and decomposition constitute such relations.
- *HMSC*. High level MSC can often be used for top level overviews. In HMSC instances are eliminated and conditions are explicitly global and restrictive.

List of figures

List of definitions

Actual gate

The message gates are used when references to the MSC are put in a wider context in another MSC. The actual gates on the MSC reference are then connected to other message gates or instances. Similar to gate definitions, actual gates may have explicit or implicit names.

A message gate always has a name. The name can be defined explicitly by a name associated with the gate on the frame. Otherwise the name is given implicitly by the direction of the message through the gate and the message name, e.g. "in_X" for a gate receiving a message X from its environment.

<actual gate area> ::=

<actual out gate area> | <actual in gate area> |

<actual order out gate area> | <actual order in gate area>

<actual out gate area> ::=

<void symbol> [**is associated with** <gate identification>]

is attached to <msc reference symbol>

[**is attached to** { <message symbol> | <lost message symbol> }]

Note: The <actual out gate area> is attached to the open end of the <message symbol> or <lost message symbol>.

<actual in gate area> ::=

<void symbol> [**is associated with** <gate identification>]

is attached to <msc reference symbol>

[**is attached to** { <message symbol> | <found message symbol> }]

Note: The <actual in gate area> is attached to the arrow head end of the <message symbol> or <found message> symbol.

<actual order out gate area> ::=

<void symbol> [**is associated with** <gate identification>]

is attached to <msc reference symbol>

is followed by <general order area>

<actual order in gate area> ::=

<void symbol>[**is associated with** <gate identification>]

is attached to <msc reference symbol>

is attached to <general order area>

Alternative

The **alt** operator defines alternative executions of MSC sections. This means that if several MSC sections are meant to be alternatives only one of them will be executed. In the case where alternative MSC sections have common preamble the choice of which MSC section will be executed is performed after the execution of the common preamble.

Condition

A condition describes either a global system state (global condition) referring to all instances contained in the MSC or a state referring to a subset of instances (nonglobal condition). In the second case the condition may be local, i.e. attached to just one instance.

<condition area> ::=

<condition symbol> **contains** <condition name list>

is attached to {<instance axis symbol>*} **set**

Connection Point

The connection points are introduced to simplify the layout of HMSCs and have no semantical meaning.

Connection points are nodes which make it possible to reduce the number of branches since several parallel branches with the same start and end give no additional meaning.

<node area> ::=

<hmsc reference area> | <connection point symbol>

| <hmsc condition area> | <par expr area>

Environment

Environment is the surroundings of an MSC. When the MSC is placed in a wider context my using MSC references, the communication with the environment from inside the MSC diagram should match the communication with the MSC reference which references it.

The environment is represented by the diagram frame.

Communication with the environment goes through gates.

General order relation

A general order relation is a binary relation between two message events. It defines a sequencing between the two events which otherwise would not have been defined.

General order relations may also be completed via gates. An order gate connects general order relations of an MSC diagram with an event of another MSC diagram. Order gates must be explicitly named.

HMSC start

The graph describing the composition of MSCs within an HMSC is interpreted in an operational way as follows. Execution starts at the <hmsc start symbol>. Next, it continues with a node that follows one of the outgoing edges of this symbol.

Incomplete messages (lost and found)

The loss of a message, i.e. the case where a message is sent but not consumed, may be indicated by a black hole.

Symmetrically, a spontaneously found message, i.e. a message which appears from nowhere, can be defined by a white hole.

<incomplete message area> ::=

{ <lost message area> | <found message area> }

{ **is followed by** <general order area> }*

{ **is attached to** <general order area> }*

<lost message area> ::=

<lost message symbol> **is associated with** <msg identification>

[**is associated with** { <instance name> | <gate name> }]

is attached to <message start area>

NOTE: The <lost message symbol> describes the event of the output side, i.e. the solid line starts on the <message start area> where the event occurs. The optional intended target of the message can be given by an identifier associated with the symbol. The target identification should be written close to the black circle, while the message identification should be written close to the arrow.

<found message area> ::=

<found message symbol> **is associated with** <msg identification>

[**is associated with** { <instance name> | <gate name> }]

is attached to <message end area>

NOTE: The <found message symbol> describes the event of the input side (the arrowhead) which should be on a <message end area>. The instance or gate which supposedly was the origin of the message is indicated by the optional identification given by the text associated with the circle of the symbol. The message identification should be written close to the arrow part.

Input event

An input event designates the consumption of a message. Normally there is a corresponding output event. The input event follows after the corresponding output event in time.

<message in area> ::= <message in symbol>

is attached to <instance axis symbol>

is attached to <message symbol>

<message in symbol> ::= <void symbol>

The <void symbol> is a geometric point without patial extension. The <message in symbol> is actually only a point which is on the instance axis. The end of the message symbol which is the arrow head is also pointing on this point on the instance axis.

Instance

An instance is an interacting entity of an MSC. Events are on instances and they are ordered according to their position on the instance from top to bottom. An instance has an instance head and an instance end or a stop. Between these there is the instance axis which may be either a single vertical line or a column defined by two vertical lines.

<instance area> ::=

<instance head area> **is followed by** <instance body area>

<instance head area> ::= <instance head symbol>

is associated with <instance heading>

[**is attached to** <createline symbol>]

<instance heading> ::=

<instance name> [[:]<instance kind>][decomposition>]

<instance body area> ::= <instance axis symbol>

is followed by {<instance end symbol>|<stop symbol>}

Loop (HMSC)

A loop in HMSC occur when branches and nodes form a cycle. There are no restrictions on how such cycles should appear.

MSC diagram

A Message Sequence Chart, which is normally abbreviated to MSC, describes the message flow between instances. One Message Sequence Chart describes a partial behaviour of a system.

An MSC describes the communication between a number of system components, and between these components and the rest of the world, called environment. For each system component covered by an MSC there is an instance axis. The communication between system components is performed by means of messages. The sending and consumption of messages are two asynchronous events. It is assumed that the environment of an MSC is capable of receiving and sending messages from and to the Message Sequence Chart; no ordering of message events within the environment is assumed.

<msc diagram> ::=

<msc symbol> **contains**

{ <msc heading> { <msc body area> | <mscexpr area> } }

MSC heading

The Message Sequence Chart heading consists of the Message Sequence Chart name.

<msc heading> ::=

msc <msc name>

MSC reference

MSC references are used to refer to other MSCs of the MSC document. The MSC references are objects of the type given by the referenced MSC.

MSC references may not only refer to a single MSC, but also to MSC reference expressions. MSC reference expressions are textual MSC expressions constructed from the operators alt, par, seq, loop, opt, exc and subst, and MSC references.

The actual gates of the MSC reference may connect to corresponding constructs in the enclosing MSC. By corresponding constructs we mean that an actual message gate may connect to another actual message gate or to an instance or to a message gate definition of the enclosing MSC. Furthermore an actual order gate may connect to another actual order gate, or an orderable event or an order gate definition.

<msc reference area> ::= <msc reference symbol>

contains { <msc ref expr> [<actual gate area>*] } **set**

is attached to { <instance axis symbol>* } **set**

is attached to { <actual gate area>* } **set**

Operator

The **alt** operator defines alternative executions of MSC sections. This means that if several MSC sections are meant to be alternatives only one of them will be executed. In the case where alternative MSC sections have common preamble the choice of which MSC section will be executed is performed after the execution of the common preamble.

The **par** operator defines the parallel execution of MSC sections. This means that all events within the parallel MSC sections will be executed, but the only restriction is that the event order within each section will be preserved.

The **loop** construct can have several forms. The most basic form is "**loop <n,m>**" where n and m are natural numbers. This means that the operand may be executed at least n times and at most m times. The naturals may be replaced by the keyword inf, like "**loop <n,inf>**". This means that the loop will be executed at least n times. If the second operand is omitted like in "**loop <n>**" it is interpreted as "**loop <n,n>**". Thus "**loop <inf>**" means an infinite loop. If the loop bounds are omitted like in "loop", it will interpreted as "**loop <1,inf>**". If the first operand is greater than the second one, the loop will be executed 0 times.

The **opt** operator is the same as an alternative where the second operand is the empty MSC.

The **exc** operator is a compact way to describe exceptional cases in an MSC. The meaning of the operator is that either the events inside the <exc inline expression symbol> are executed and then the MSC is finished or the events following the <exc inline expression symbol> are executed. The exc operator can thus be viewed as an alternative where the second operand is the entire rest of the MSC.

Output event

An output event designates the output of a message. Normally there is a corresponding input event. The output event must come before the corresponding input event in time.

<message out area> ::= <message out symbol>

is attached to <instance axis symbol>

is attached to <message symbol>

<message out symbol> ::= <void symbol>

The <void symbol> is a geometric point without patial extension. The <message out symbol> is actually only a point which is on the instance axis. The end of the message symbol which has no arrow head is also on this point on the instance axis.

Reference expression

MSC references may not only refer to a single MSC, but also to MSC reference expressions. MSC reference expressions are textual MSC expressions constructed from the operators **alt**, **par**, **seq**, **loop**, **opt**, **exc** and **subst**, and MSC references.

The **alt**, **par**, **loop**, **opt** and **exc** operators are described in definition of operator. The **seq** operator denotes the weak sequencing operation where only events on the same instance are ordered.

The **subst** operation is a substitution of concepts inside the referenced MSC. Message names are substituted by message names, instance names by instance names and MSC names by MSC names.

Restrictive condition

Four static restrictions are related to conditions in HMSCs:

•If an <msc reference> is immediately preceded by a <condition symbol>, with an associated set of <condition name>s, then this set must be a subset of the set of initial conditions of the <msc ref expression> associated with the <msc reference>.

•If an <msc reference> is immediately followed by a <condition symbol>, with an associated set of <condition name>s, then this set must be a subset of the set of final conditions of the <msc ref expression> associated with the <msc reference>.

•If an <par expr area> is immediately preceded by a <condition symbol>, with an associated set of <condition name>s, then this set must be a subset of the set of initial conditions of the <par expr area>.

•If an <par expr area> is immediately followed by a <condition symbol>, with an associated set of <condition name>s, then this set must be a subset of the set of final conditions of the <par expr area>.