Component Interface Description Using HyperMSCs and Connectors

Peter Graubmann
Siemens AG, München
peter.graubmann@mchp.siemens.de

Ekkart Rudolph
Technische Universität München
rudolphe@informatik.tu-muenchen.de

Jens Grabowski
Med. Universität zu Lübeck
grabowsk@itm.mu-luebeck.de

Abstract

Modelling of complex systems with Message Sequence Charts requires several extensions in order to arrive at sufficiently transparent and manageable descriptions. Two extensions of major importance are introduced within this paper: on the one side, extended High Level MSCs denoted as HyperMSCs are allowed to contain MSC references with hypertext-like inscriptions or in expanded form as detailed MSCs; on the other side, MSC connectors are introduced in form of MSCs representing high level communication patterns between MSC components. The introduction of MSC connectors may be viewed as a generalisation of the gate concept and as a completion of the MSC language for communicating operator expressions. Moreover, MSC connectors can be employed quite generally as a communication description on a higher level of abstraction - a structural language construct which obviously is still missing in the MSC standard. These new concepts allow a system modelling based on stepwise refinement starting with HyperMSCs, decomposed instances and MSC connector communication

1. Motivation

The ITU standard language Message Sequence Chart (MSC) [16][17] is generally accepted as a central visual modelling technique for the description of the dynamic behaviour of systems, in particular of the communication behaviour. As such, MSC is equally popular in the telecommunication area where it is used dominantly in combination with SDL [2], as in the field of object oriented modelling in form of the OO variant UML - Sequence Diagrams [6][12]. While the MSC standard language MSC-2000 [17] is perfectly tailored for the description of sets of sample behaviour, in case of a more comprehensive description it easily leads to specifications that are difficult to handle and also lack transparency. However, incoherent collections of sample behaviour are not sufficient for an adequate description of complex systems. Thus, in today's system development, comprehensive behaviour descriptions are demanded, if not for the interaction of complete systems, so at least for identifiable, self-contained parts, as, e.g., for interface protocol-, test case-, and Use Case descriptions [1].

As a consequence, there is the danger that, in applications, MSC descriptions increasingly may appear to be over-complicated and that the MSC language is becoming less attractive at least for the modelling of complex systems. It is the main purpose of this paper to propose certain extensions of the MSC language in order to preserve the transparency and simplicity of MSC specifications also on an advanced level of system modelling which essentially refers to the usage of High Level MSCs.

High Level MSCs (HMSCs) [10][13] are widely appreciated as a means to present the dynamic system behaviour on a higher level of abstraction. As such, HMSCs have been compared to roadmaps since they may be viewed as a means to navigate within sets of basic MSC descriptions. Nevertheless, the connection between High Level MSCs and basic MSCs, at present, is unsatisfactory since, in case of many small separate MSC reference definitions, the specification becomes unmanageable [5][13]. Within this paper, hypertext-like mechanisms are proposed for HMSCs, in order to render possible the arbitrary folding and unfolding of MSC references, thus providing the transition from a high level HMSC view to basic MSCs in a smooth manner. Thus, eventually also the strict borderline between HMSCs and basic MSCs is disappearing.

Another deficiency is that HMSCs do not support a component oriented view, i.e., HMSCs, at present, are not tailored for the high level description of individual system components which communicate with each other via communication protocols. A first attempt towards a component oriented view has been provided in [10], however, the introduction of gates for HMSC within this paper can be seen only as a first step and in general proves to be not completely appropriate. Such partitioning mechanisms, however, are crucial for a system modelling which is based on stepwise refinement of components and their communication behaviour. Certain refinement mechanisms are provided in the MSC standard by the concept of 'decomposed instances' [13][17] which is quite related to HMSCs. However, a corresponding convincing and sufficiently general concept for the refinement of the communication behaviour is still lacking though some proposals have been made into this direction [4][8]. MSC references [13][17]

may be used only in very simple cases as is shown in Chapter 3. Within this paper, we propose MSC connectors for that purpose. MSC connectors quite generally mirror the fact that today's system development is component oriented. The communication between parts of a system, i.e., its components, follows interaction patterns which are expressed in the component oriented world as software connectors [3]. MSC connectors allow to define recurring interaction patterns and to apply them jointly with the other MSC abstractions, like High Level MSCs, MSC references, and decomposed instances. The HyperMSC concept of folding and unfolding can be conveniently carried over to MSC connectors. MSC connectors have been introduced recently in an informal way for some special cases [5]. Within this paper, they are described for the first time in full generality being defined themselves in form of MSCs.

The concepts of HyperMSCs and MSC connectors have been essentially stimulated by the development of a graphical format for TTCN-3 [14][15]. Practice has shown that a naive translation of TTCN test cases into MSC descriptions does not lead to diagrams that are sufficiently transparent. Experiments with the employment of inline expressions and HMSCs have led quite naturally to the idea of HyperMSCs. In addition, the translation of TTCN-3 test components leads to MSC test component descriptions which have to be merged by a join operation. For an appropriate specification of such a join operation, an MSC connector has been proposed which allows to describe the exchange of co-ordination messages among test components on a suitable level of abstraction.

Another source for the development of the MSC connectors and the HyperMSC mechanism stems from considerations with respect to system family engineering where the composition of (mostly pre-developed) components and their variants is prevalent. In this context, it becomes essential to clearly define and describe the protocols that are associated with the component interfaces. For system families, protocols describing the interaction of their parts become valuable assets in the design management space when it comes to identifying which component is appropriate to be re-used in a particular context. MSC connectors are well suited to support the development of system families. So, at present, the investigation of MSC connectors is carried out within the ITEA projects ESAPS (Engineering Software Architectures, Processes and Platforms for System Families) [7] and its continuation CAFÉ (From Concept to Application in System Family Engineering) funded by the German Ministry for Education and Research.

Within Chapter 2, the concept of HyperMSCs is presented. Chapter 3 describes the MSC connector concept in the context of decomposed instances. The next chapter, Chapter 4, explains the concept by means of the well known Inres protocol. Chapter 5 shows that the old MSC message gate concept is elegantly subsumed by the connector concept and, eventually, Chapter 6 provides conclusion and outlook.

2. Advanced Visual System Modelling with HyperMSCs

The MSC language contains several constructs for behaviour composition and refinement – decomposed instances, MSC references, MSC reference operator expressions, inline expressions and HMSCs [8][13][17]. There is a lot of overlap between these constructs, however, the variety is justified by different application areas and different forms of visualisation. Nevertheless, a more unifying view would be desirable together with the possibility to easily switch between different representations. In particular, there is an obvious gap between HMSCs and the rest of the MSC language which disturbs the homogeneity of the language. Though all of these structural concepts have proven to be most fruitful for sets of sample behaviour descriptions, they are not yet perfectly suited for advanced visual modelling of more comprehensive behaviour descriptions [5][6][15]. A more unifying standpoint eventually leads to the concept of HyperMSCs which also provides a suitable visualisation means. To apply this concept most fruitfully, a corresponding advanced tool support appears to be mandatory.

As a first step, we want to recall the relation between MSC operator expressions and inline expressions. As can be seen in Figure 1, inline expressions describe just unfolded MSC reference operator expressions.

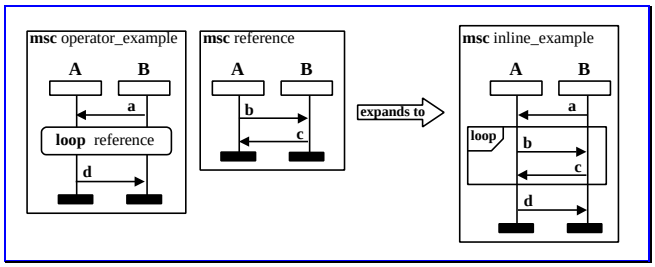

Figure 1 Expansion of an MSC operator expression by means of an inline expression.

Whereas in case of MSC operator expressions the unfolding is well established (except for the operator **seq)** there is no corresponding possibility in the MSC language in case of simple MSC references. Some SDL- and MSC-tool manufacturers, however, have soon discovered the deficiency in the language and consequently provided a corresponding inline definition for MSC references. It should be pointed out, that this is not a pure tool issue but demands a real extension of the language if the expanded

form still contains the MSC reference symbol. In Figure 2, such an unfolding of an MSC reference is provided.

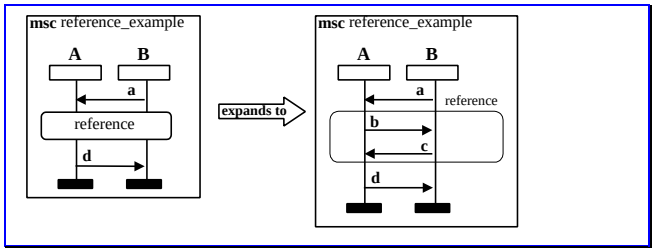

Figure 2 Unfolding of an MSC reference within the embedding MSC.

The expressiveness of this MSC reference inline expansion becomes more evident in case of HMSCs [5][9]. HMSCs describe the composition of MSCs in form of a graph with MSC references and conditions as nodes. This way, they abstract from instances and messages which are not shown in the diagram. Each MSC reference points by means of the reference name to another MSC in the MSC document which defines the meaning of the reference, i.e., each reference symbol can be seen as a placeholder for an MSC diagram which has to be defined somewhere else in the MSC document. In case of many fairly small MSC reference definitions, such a representation soon becomes quite complex and in practice is difficult to handle. The inline expansion of MSC references turns out to be useful particularly in such cases [5][13].

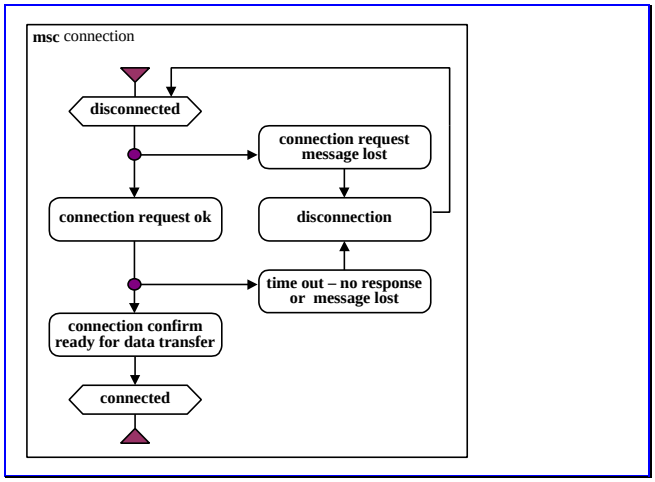

Figure 3 HyperMSC for connection establishment. All MSC references are folded into the respective MSC reference symbols.

Within HMSCs, several expanded MSC references may be combined to one coherent expanded MSC reference with the connection points being shifted to the borderline of the MSC reference. This way, a convincingly transparent representation is obtained even in case of many alternatives and loops [5][6] (see Figure 3 and Figure 4).

The folding and unfolding of HMSCs can be compared with electronic roadmaps where certain parts can be selected and expanded in detailed form whereas the folded parts are presented in an abbreviated form as overview. Because of the analogy to hypertext-like mechanisms we have chosen the name HyperMSC for such extended HMSCs. MSC reference symbols in HyperMSCs may either contain hypertext-like descriptions or, in its expanded form, detailed MSCs.

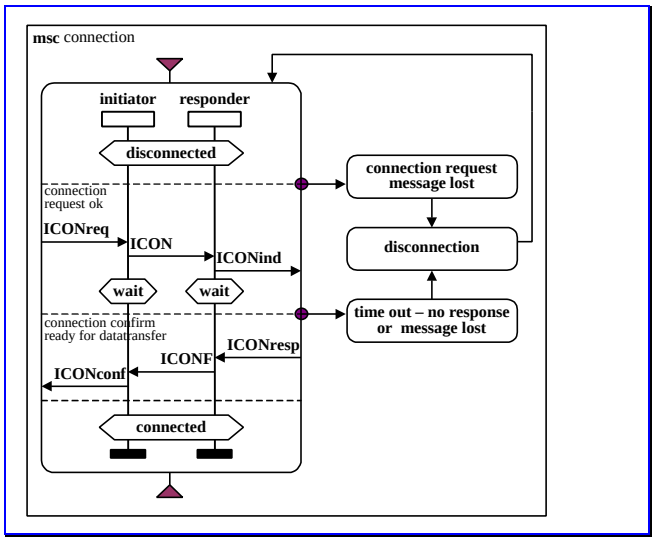

Figure 4 HyperMSC of Figure 3, where the path describing a "successful connection" is expanded.

In the context of HyperMSC, special attention has to be paid to the concept of decomposed instances. This concept is useful for the description of components which can be further decomposed by means of a refining MSC. However, it can be alternatively modelled by a corresponding HMSC containing one MSC reference which points towards this refining MSC (see Figure 5). Within this paper we use the strong relationship between decomposed instances and HMSCs for the purpose of inline expansions of decomposed instances.

It should be noted that the decomposed instance "A" in Figure 5 is represented graphically by drawing the instance axis with a double line. This appears to be more intuitive than the keyword **decomposed** and is also stimulated by a corresponding connector symbol introduced in Chapter 3.

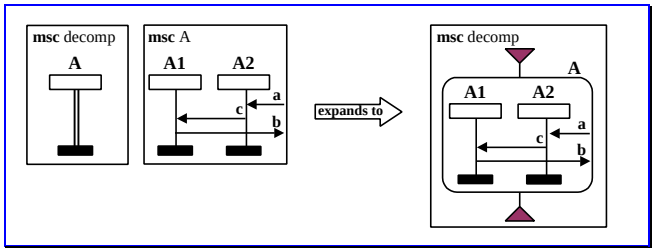

Figure 5 Inline expansion of a decomposed instance “A” with refining MSC “A” by means of an HyperMSC.

3. System Development with MSC Connectors and Decomposed Instances

The refinement of MSCs normally is based on the concept of decomposed instances on the one side and on the refinement of messages on the other side, whereby one message in the next step of refinement may be replaced by a whole set of messages describing a complex communication pattern [4][8]. Obviously, messages and instances are treated on a different footing. The concept of MSC connectors that is introduced in this chapter, in a certain respect can be viewed as the message counterpart to decomposed instances. MSC connectors somehow represent high level messages and therefore are described graphically as double line message arrows (see Figure 6). MSC connectors in general may describe a complex communication behaviour which itself is best defined by means of an MSC. MSC connector definitions are distinguished from MSCs by the keyword **connector**. Otherwise they resemble MSCs with only a few additional constructs. A connector may be unidirectional, indicated by a single arrow head into the direction of the one-way communication, or it may be bi-directional as indicated by a double arrow head.

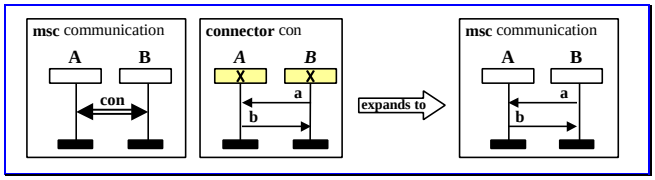

Figure 6 Basic connector definition.

MSC connectors contain external connector instances which represent abstractions of the entities to which the connector shall be connected. External connector instances show a distinguished instance header: in the textual representation they are identified by the keyword **external**, in the graphical representation they show an “**x**” in their instance head symbol. As a first simple example see Figure 6 with the MSC “communication”. The two instances “A” and “B” communicate via the connector “con” which is defined within the connector MSC “con”. Obviously, the connector “con” just means a compact description of the exchange of the messages “a” and “b”. In case where the connector is joined to plain instances, it plays the same role as an MSC reference since both constructs are placeholders for the defining MSC to which they are pointing. Also both, MSC connector definitions and MSC reference definitions, may contain further instances in addition to the connecting instances of the embedding MSC.

Connectors, however, play a much more sophisticated role if they describe the communication behaviour between decomposed instances or MSC references. In this case, the events on the external connector instances in the connector definition have to match (synchronise) with corresponding events on the instances to which the connector is attached. This matching mechanism is closely related to the join operation introduced in [8].

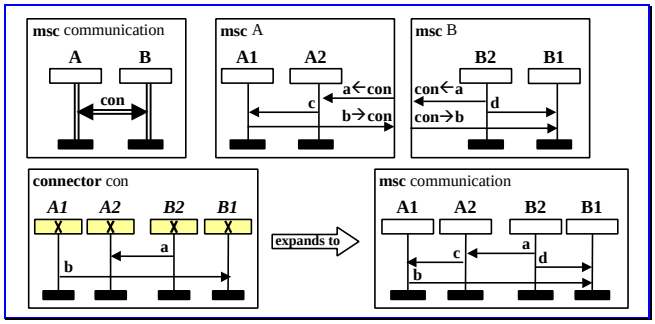

Figure 7 Connector communication between decomposed instances.

Figure 7 shows an MSC with the decomposed instances “A” and “B” where decomposed instances are indicated with doubled lines for their instance axes, just in analogy to the double lined arrow of the connector. Each decomposed instance is defined by a corresponding MSC reference. The MSC connectors between decomposed instances bundle the messages which are crossing the environment of the refining MSCs. This is more clearly visualised in Figure 8, where the decomposed instances “A” and “B” are shown in expanded form as HyperMSCs. Thereby, it has been used that decomposed instances can be equivalently represented as HMSCs.

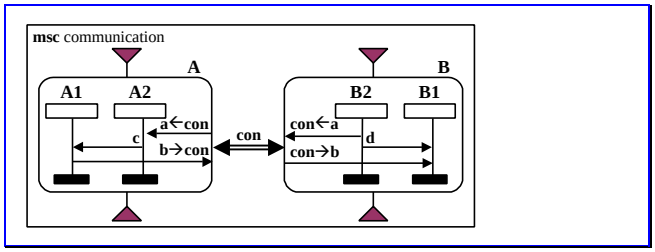

Figure 8 Decomposed instances expanded in form of HyperMSCs.

Within these refining MSCs “A” and “B”, connector pointers are added to message names in order to indicate to

which connector a message is sent (<message name>→ <connector name>, e.g., a→con), or from which connector a message is received (<connector name>→<message name>, e.g., con→a). Obviously, the messages "c" and "d" do not occur in the connector definition. However, if one tries to replace the connector by a description using an MSC reference, the messages "c" and "d" have to be included in the MSC reference definition. This demonstrates already a characteristic feature of MSC connectors which merely focus on the communication behaviour in contrast to MSC references that in general also contain internal behaviour. Obviously, MSC connectors and MSC references play a different role in system modelling. For the description of the communication behaviour on a higher level of abstraction, MSC connectors are demanded since they abstract from internal events. Beyond that, MSC connectors rather are intended to describe abstract communication patterns than concrete message sequences. This becomes evident in case of the probably most common MSC connector, namely the FIFO connector (see Figure 9).

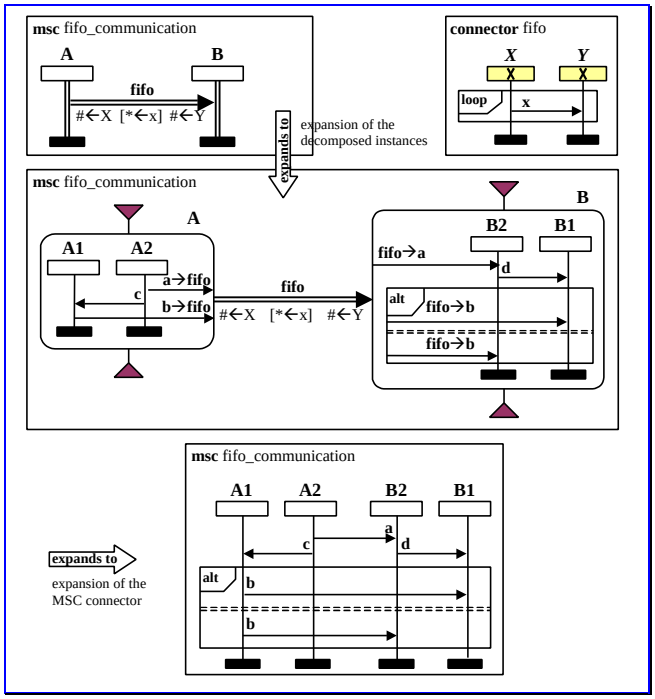

Figure 9 Definition and use of a FIFO connector.[1]

In the definition of the FIFO connector (connector "fifo"), the external connector instance "X" represents the sender, the external instance "Y" the receiver role. We have used a mapping of the external instances to the general instance identifier '#' which indicates that all instances involved in MSC reference "A" and "B" may be sender or receiver, respectively. We have also used the mapping to the general message identifier "*" which indicates that all messages sent to or received from the connector may be matched, i.e., in the situation of Figure 9, both the messages "a" and "b" are conveyed through the connector "fifo". The example demonstrates the generality of the MSC connector concept. In practise, of course, a predefined FIFO connector will be used by means of the keyword **FIFO**.

The above mentioned mapping mechanism for external instances can be generalised to select exactly which external instance of an MSC connector is to be associated with an instance in an MSC reference in cases where it is not obvious by name identification. In a similar form, message mapping makes it possible to express in detail which message in the connector definition is to be identified with the actual messages in the MSC references.

4. The Inres Example

The Inres service [2] is an abridged version of the Abracadabra service. The service is connection oriented. A user who wishes to communicate with another user via the service must first initiate a connection before exchanging data. For simplification purposes, the service is not symmetrical. The Initiator-user can initiate a connection and later send data. The Responder-user can accept the connection or reject it. After acceptance, it can receive data from the initiating user. Initiator and Responder have to use the underlying Medium-service for their communication. The Medium-service is unreliable and data may be lost.

In the following, we describe the "Connection establishment and Disconnection phase" in Inres.

A connection establishment is initiated by the Initiator-user with an ICONreq message to the Initiator (i.e., to the sender component in the Inres protocol). Within the MSC diagrams, Initiator-user and Responder-user will be represented by the environment. The Initiator then sends a CR (connection request) to the Responder (i.e., the target component to which data shall be transmitted). If the Responder receives a CR from the Initiator, the Responder-user gets an ICONind message. The Responder-user can respond with ICONresp or IDISreq. ICONresp indicates its willingness to accept the connection. The Responder thereafter sends a CC (connection confirm) to the Initiator. If the Initiator receives a CC, it sends an ICONconf to its user and the data transmission can be started. An IDISreq from the Responder-user results in the sending of a DR (disconnection request) to the Initiator. If the Initiator receives a DR from the Responder, the Initiator sends an IDISind message to the Initiator-user. If the Initiator receives nothing at all in response to a CR after time T, CR

[1] A type concept – not detailed here due to lack of space – would be needed to avoid misunderstandings when one connector definition is instantiated several times within the same diagram (cf. the similar problem for MSC references).

is sent again. If, after four attempts, still nothing is received by the Initiator, the Initiator sends an IDISind to the Initiator-user.

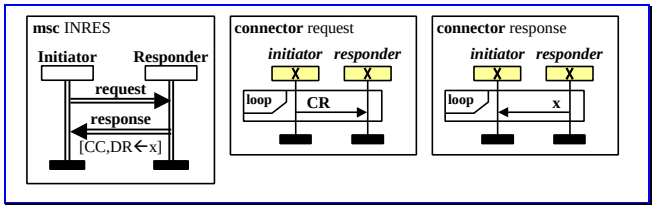

Figure 10 Inres – basic connector description.

In the following, the notation clearly distinguishes between decomposed instances (indicating the components of the Inres protocol) and their corresponding refining instances by using capital initial letters for the names of the decomposed instances ("Initiator", "Responder") and lower case initial letters for instance names of refining instances ("initiator", "responder"). Thus, the decomposed instance "Initiator" is refined by the instance "initiator" and the decomposed instance "Responder" by the instance "responder", whereby, in this case, the refinement is provided merely in form of a complex behaviour description instead of a splitting of instances.

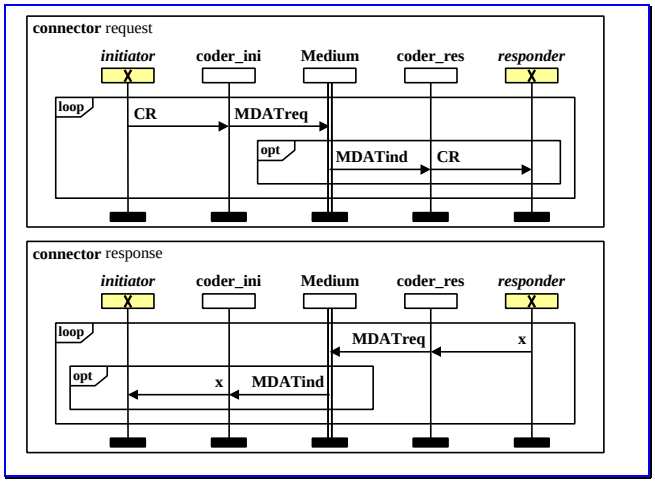

Figure 11 Inres – connector description taking into account the unreliable Medium.[2]

In Figure 10, first the simpler case of a connection set-up is modelled where the Medium-service is omitted and the messages are directly exchanged between "Initiator" and "Responder".

In Figure 11, a connector definition is provided which takes the unreliable medium into account. This example demonstrates that MSC connectors are capable to describe also more complex message interfaces. In particular, the connectors contain non-external connector instances ("coder_ini", "coder_res", "Medium"). This example also indicates how a system may be refined by modifying the connector definition without changing the decomposed instances.

The two connectors "request" and "response" can be unified by using a further abstraction. Obviously, the connection "request" can be mapped onto the connection "response" if we use the abstraction of the concrete message identifier "CR" to "x" and exchange instance "initiator" and "responder" on the one hand, and instance "coder_ini" and "coder_res" on the other hand. This leads to the following unifying abstract connector "con" (see Figure 12). The further abstraction is in line with the concept of reusability which is a central idea behind the MSC connector concept.

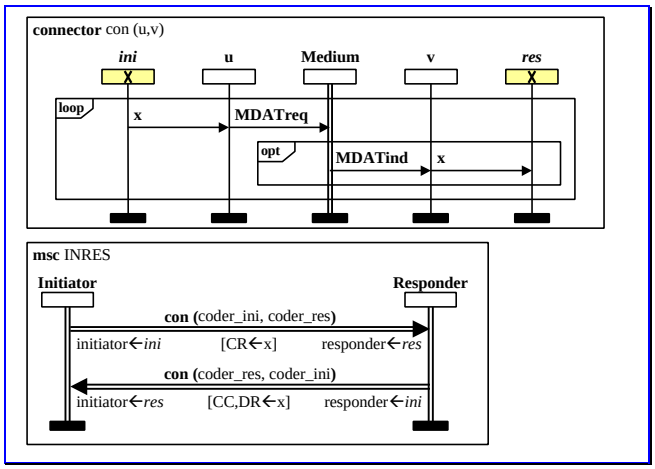

Figure 12 Unification by means of the abstract connector "con".[3]

Within Figure 13, the decomposed instances "Initiator" and "Responder" are shown in expanded form.

The traces described by the HMSCs "Initiator" and "Responder" have to synchronise via the MSC connectors "request" and "response" (which are defined in Figure 11). Figure 14 demonstrates this for the initial step of the Inres protocol by synchronising the initial MSC references "con-req" in HMSC "Initiator" and "conind" in HMSC "Responder". In the beginning, only the instance "initiator" is ready to receive the message "ICONreq" from the Initiator-user (i.e., from the environment) and to send a connection request to the "Responder". Depending on the unreliable medium, the "Responder" may receive the connection response or not and send a message "ICONind" to the Receiver-user (which is also represented by the environment).

[2] In order to avoid ambiguities, in a complete description also the scope of x with respect to the mapping onto values "CC" and "DR" has to be defined. Here, the scope of the binding of x is restricted to one loop. Scoping rules are not detailed further due to the lack of space.

[3] Note the parameterisation of the connector definition and the mapping of the connector's external instances (*ini*, *res*) onto the instances initiator and responder in the respective decomposed instances which is defined by the expressions "initiator←*ini*", etc.

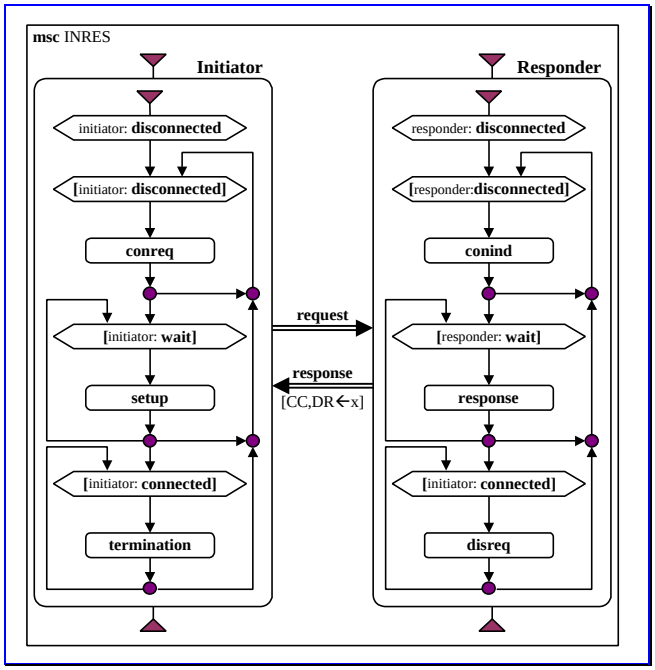

Figure 13 Expansion of the decomposed instances “Initiator” and “Responder”.

The specification of the connection establishment and disconnection phase can be extended to the inclusion of the data transfer in a similar manner. Because of its relative simplicity, the Inres protocol is ideally suited to demonstrate the capacity of the connector concept for the specification of complex communication interfaces. In fact, the MSC connectors “request” and “response” comprise already an amazing number of concrete communication cases due to the possibility of several repetitions of the connection request in case of time-out. Obviously, a comprehensive specification using basic MSCs would be not manageable, however, selected representations in form of basic MSCs can easily be derived from the high level description.

5. Subsuming the MSC Message Gate Concept under MSC connectors

Message gates in the MSC standard language are used to define connection points for messages with respect to the interior and exterior of MSC references and inline expressions. In simple cases, i.e., MSC references without operator expressions, the interpretation of gates is straightforward. In case of intricate operator expressions, an unambiguous and convincing interpretation has not yet been given [9], cf. also the respective discussion in [11].

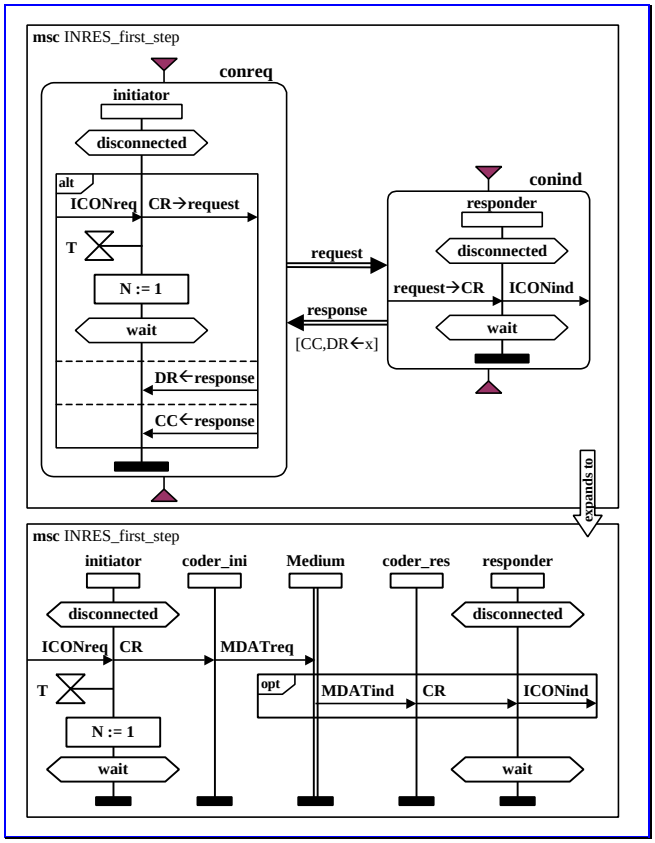

Figure 14 Expansion of the first step (MSC references “conreq” and “conind”) of the Inres specification in Figure 13.

Using the MSC connector concept, a mapping from gates to a basic connector can be provided which eventually also leads to a general and unambiguous gate interpretation. The basic connector definition is provided by a simple message connection.

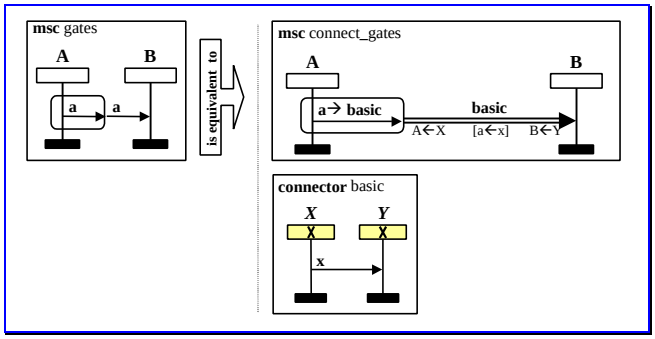

Figure 15 Replacement of an HyperMSC gate by an MSC connector.

Within Figure 15, a gate is replaced by a basic connector and assigning the message name of the ‘gating’ message to the connecting message in the basic connector definition. As such, the message gate construct can be viewed as a shorthand notation in case of the basic connector.

The same gate interpretation by means of a basic MSC connector can be used also in case of operator expressions. As an example, we use the message communication of an **opt** expression [9] (see Figure 16).

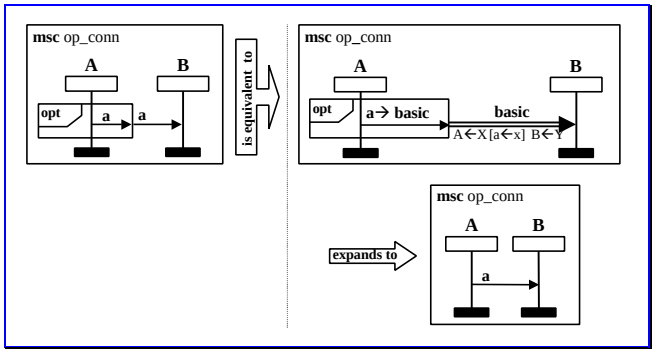

Figure 16 Replacement of a gate in an operator expression by an MSC connector.

6. Conclusion and Outlook

Within this paper, for the first time the MSC connector concept has been introduced in full generality by employing MSCs for the connector description itself. The newly introduced MSC connector concept makes a component oriented system modelling by means of MSCs possible in addition to the traditional system oriented view. Such a component oriented view is demanded, e.g., in system family engineering and particularly for the development of modern test specification languages based on TTCN-3 and MSC. Within this paper, the MSC connector concept has been applied to a protocol specification which demonstrates that MSC may become a universal modelling language by means of this new extension. Though the range of applicability of MSC with connectors appears to be nearly unlimited, the main focus is still on the specification of Use Cases, component interaction schemes (interface protocols) and test cases. As such, the extended MSC language is not intended to replace process oriented languages like SDL or State Charts but rather to provide a smooth transition from a communication oriented view to a process oriented view.

7. References

[1] M. Broy, I. Krüger: *Interfaces – Towards a Scientific Foundation of a Methodological usage of Message Sequence Charts.* In: Formal Engineering Methods ICFEM'98 (J. Staples, M.G. Hinchey, Shaoying Liu, editors), IEEE Computer Society, 1998.

[2] J. Ellsberger, D. Hogrefe, A. Sarma: *SDL - Object oriented Language for Communication Systems.* Prentice-Hall.1997.

[3] A. Egyed, N. Metha, N. Medvidovi: *Software Connectors and Refinement in Family Architectures*. In: Proceedings of the 3rd Int. Workshop on Software Architectures for Product Families, Las Palmas de Gran Canaria, Spain, March 15-17, 2000.

[4] A. Engels: *Message Refinement –Describing Multi-level Protocols in MSC.* In: Proceedings of the 1st SAM Workshop (Y. Lahav, A. Wolisz, J. Fischer, E. Holz, editors), Berlin, June 1998, Informatik-Bericht Nr. 104, Humboldt-Universität Berlin.

[5] J. Grabowski, P. Graubmann, E. Rudolph*: HyperMSCs with Connectors for Advanced Visual System Modelling and Testing.* In: SDL Forum 2001 – to appear.

[6] P. Graubmann, E. Rudolph: *HyperMSCs and Sequence Diagrams for Use Case Modelling and Testing*. In: UML2000, 3rd International Conference on The Unified Modeling Language (A. Evans, S. Kent, B. Selic, editors), 02-06 October, 2000, York, UK, Springer 2000.

[7] P. Graubmann, R. Wasgint: *Methods for Interface Annotations and Component Selection*. SAG-WP2-0106-16, ESAPS internal report, 2001.

[8] I. Krüger: *Distributed System Design with Message Sequence Charts*, PhD Thesis, Techn. Universität München, 2000.

[9] S. Loidl, E. Rudolph, U. Hinkel: *MSC'96 and Beyond-a Critical Look*. In SDL'97 Time for Testing-SDL, MSC and Trends, Proceedings of the 8th SDL Forum in Evry, France (A. Cavalli, A. Sarma, editors), North Holland, Sept. 1997.

[10] S. Mauw, M. A. Reniers: *High Level Message Sequence Charts*. In: SDL'97 - Time for Testing-SDL, MSC and Trends, Proceedings of the 8th SDL Forum in Evry, France (A. Cavalli, A. Sarma, editors), North Holland, Sept. 1997.

[11] M. A. Reniers: *Message Sequence Chart: Syntax and Semantics.* PhD Thesis, Eindhoven Univ. of Technology, 1999.

[12] E. Rudolph, J. Grabowski, P. Graubmann*: Towards a Harmonization of UML-Sequence Diagrams and MSC.* In: SDL'99 - The Next Millennium, Proceedings of the 9th SDL Forum in Montréal, Québec, (R. Dssouli, G.V. Bochmann, Y. Lahav, editors), Elsevier Science B.V., Amsterdam, 1999.

[13] E. Rudolph, J. Grabowski, P. Graubmann: *Tutorial on Message Sequence Charts (MSC-96).* Forte/PSTV'96. Kaiserslautern, Germany, October 1996.

[14] E. Rudolph, I. Schieferdecker, J. Grabowski: *Development of an Message Sequence Chart/ UML Test Format*. In: Proceedings of FBT'2000 - Formale Beschreibungstechniken für verteilte Systeme, Lübeck, Germany (J. Grabowski, S. Heymer, editors). Shaker-Verlag, Aachen, 2000.

[15] E. Rudolph, I. Schieferdecker, J. Grabowski: *HyperMSC - A Graphical Representation of TTCN*. Proceedings of the 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM'2000), Grenoble, France, June, 26 - 28, 2000.

[16] ITU-T Rec. Z.120 (MSC-96): *Message Sequence Chart (MSC*). (E. Rudolph, editor), Geneva, 1996.

[17] ITU-T Rec. Z.120 (MSC-2000): *Message Sequence Chart (MSC).* (O. Haugen, editor), Geneva, 1999.