Connecting Message Sequence Charts

Frans A. Meijs,

Philips Research Laboratories.

Address: Philips Research, Prof. Holstlaan 4, 5656AA Eindhoven,

Tel. +31 402744742, Email: frans.a.meijs@philips.com

Keywords: finite state machine, invariant, Message Sequence Chart, MSC, postcondition, precondition, SDL, semantics

Date: May 30, 2000. Category: Research paper

Abstract

The Message Sequence Chart standard defines a type of connection for (transition-)instances. In this paper a method is presented for further exploiting the connection mechanism. The objective is to get a method which minimises the number of MSCs required for describing the complete required behaviour of a system of communicating components. The method results in a specification style where each MSC represents a particular type of transaction. Two case studies have proven that this approach may be useful during design of systems of concurrently running components, where MSC complements SDL by making the event ordering explicit. The method is orthogonal to the High-level MSC notation as the latter is of a higher level of abstraction.

1 Introduction

During specification work for an MHEG[1] presentation engine we felt the need to use MSCs, for representing the required behaviour of the engine's components as completely as possible. The reason for this was our experience that unintended behaviour arises easily when only SDL diagrams — or equivalent — are used during design. MSC complements SDL here by making the event ordering explicit. When doing so, the problem occurred that we were obliged to draw an MSC for each state of the system, because a general reset command dictated a transition from each state to the initial state. Also — when trying to get a good coverage of the total behaviour — the number of MSCs to be drawn exploded because we had to make state information explicit and there were state variables defined over large sorts, implying many possible values. The same problem occurred during specification work for the Digital Video Broadcast (DVB) Interaction Channel protocol (currently an ETSI standard [2]).

[1] A hyperdocument standard.

So a method was needed which reduced the explosion problem. We were already using the Basic MSC syntax with local conditions for representing the state of individual components, and we wanted to stay as close as possible to this approach. Given these requirements, we found as a solution [14] to make each MSC represent a transaction, i.e. an atomic action involving two or more system components[2], and where an actual system's behaviour is constructed by the connection — a kind of sequential composition — of MSCs. Connection of two MSCs in its turn is obtained by merging conditions. The method we arrived at is orthogonal to the High-level MSC notation as the latter is of a higher level of abstraction: In HMSCs global conditions are used for connecting MSCs, without reference to the contents of the individual MSCs. As such the HMSC notation supports a top-down way of development. In contrast, our method addresses local conditions in individual MSCs, where connections between MSCs are a consequence of the relations between the local conditions, thereby introducing a bottom-up development aspect which may also be useful for analysing existing systems.

According to [15] p.36, one of the applications of the MSC *condition* construct is to represent a particular state of a system's component. The underlying assumption is that each component is of the *extended finite state machine* (EFSM) type, whose state is the vector set up by the individual states of its state variables. This is analogous to the SDL semantics — where an EFSM is identical to a process. Rudolph et al. [16] already indicate connection of instances having identical conditions, and the MSC 2000 [9, 5, 8, 12, 13] standard defines connection by satisfying guards. Kelly [10] defines a pre-condition as being a segment of a synchronous MSC containing messages, conditions and time delays, delimited by a a proprietary syntax. A connection occurs when the instantaneous behaviour matches the pre-condition. Our work builds on the MSC 2000 approach, as it is an exploitation of the "data-oriented guard" feature.

For the rest of this paper, is is assumed that the reader is familiar with the MSC notation and semantics. The paper is organized as follows: First, conditions denoting sets of states are discussed, then the coupling which may occur between transitions described by such conditions, then connectability of transition-instances[3] of a given EFSM, as may arise due to coupling of transitions, and connectability of entire MSCs, as may arise due to connectability of instances. Finally the MSC connectability diagram — which gives the general behavioural view we were striving for— is discussed and some practical work is reported on. Just those formalisms are given which are necessary for properly conveying the ideas.

[2]Here "atomic" stands for "undisruptable".

[3]When ambiguity has to be avoided we will refer to an instance in the MSC sense as a *transition-instance*.

2 Connectability of MSCs

2.1 Conditions denoting sets of state vectors

Let us consider a given EFSM. A type of condition is used, where the expression associated with it defines not just a single state, but a *set of state vectors*. Each element of a state vector is a value which has a 1-to-1 correspondence with a state variable of the EFSM, and each state vector is a possible state of the EFSM.[4] The expression is a constraint — i.e. an invariant — on the state variables of the EFSM,[5] saying that the vector of instantaneous values should be a member of this particular set.[6]. It may refer to state variables of the EFSM — and only of the EFSM — and constants.

A transition-instance containing such a condition denotes a *class* of transitions — instead of a single transition — with at least one transition from or to each element of the state vector set. The class is also characterised by the fact that all transitions in it have the same input event as a trigger, and that all transitions in it map to the same set of output events — if there are any.

Summarising, an EFSM consists in our view of

- A set of state variables, defined over finite sets (sets of values)
- A set of transition classes, each class consisting of
  - A state-vector-set where it maps from
  - A state-vector-set where it maps to
  - An input event (called *trigger*)
  - Optionally, one or more output events
- One of the state-vector-sets is assumed to be the *initial state*; it has to be a singleton (contain a single vector).

For the rest of this paper we will omit the word "vector" as long as there is no ambiguity. To prevent conditions of different EFSMs getting related, For uniqueness, each state variable name is implicitly prefixed by the name of the EFSM.

[4] Each state-vector-set is a special subset of the full state space, each subset having its own defining condition.

[5] This model is actually slightly more general than the EFSM as it treats the state of the EFSM as being the state of just another state variable.

[6] [10] names this an *assertion*.

2.2 Coupling between transitions of a given EFSM

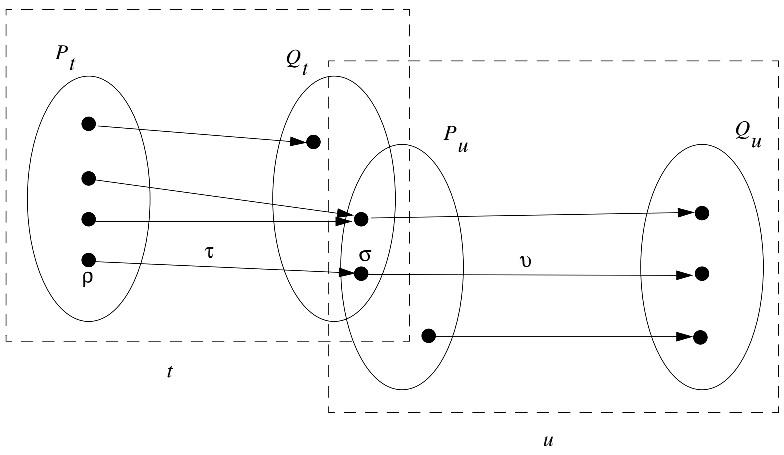

Figure 1: Example of coupling between transition class t and u

From here on we will analyse on two levels of abstraction:

- The *specification-level* where two or more MSCs may be connected to become one MSC denoting the same process.
- The *model-level*, where a hypothetical interpreter maintains a model (of the state variables) of a system, while interpreting a set of MSCs describing the system. This view is identical to the operational semantics.

We will use simple first order predicate calculus extended with binary (dyadic) relations over finite sets. The following definitions are needed when considering the transition classes of a given EFSM, see also figure 1:

ρ and σ each denote an individual state vector.

T denotes the set of all transition classes of the EFSM.

t and u each denote a transition class: $t \in T$ and $u \in T$.

τ and υ each denote an individual transition; $\tau \in t$ and $\upsilon \in u$.

P_t denotes the set of all state vectors from which each transition of t maps.

Q_t denotes the set of state vectors to which each transition of t maps.

$s(x)$ denotes the set of state vectors satisfying constraint x.

p_t is the pre-condition [7] of t, $P_t = s(p_t)$.[7]

q_t is the post-condition of t, $Q_t = s(q_t)$.

A pre-condition may be just *true* (e.g. in the case of a *reset* condition, which is a condition to which a transition may be made from all other conditions).

When two sets of the kind discussed above have a non-zero intersection, we say that they are *coupled*. In this case — and only in this case — there is at least one state where an interpreter can continue asserting transitions. *Connectability* of a given post-/pre-condition pair is obtained when the sets defined by the conditions are coupled. When the interpreter arrives at a post-condition, the instantaneous state vector of the corresponding EFSM is recorded, and all MSCs of the system are searched for a pre-condition which evaluates to *true* for the recorded state vector. From the pre-conditions found, one is selected and the interpreting process continues. On the specification level this is represented by *merging* the involved (MSC-)conditions: they are replaced by a single condition, having the conjunction of the original expressions. It can be proven that the conjunction properly represents the intersection of the sets. The conjunction describes all the states which are allowed to appear at this place, so it is an *invariant*. As the merge is done instantaneously, any events occurring after the pre- or post-condition also occur after the merging action: i.e. a merge of conditions preserves the ordering of events. If no connectable condition is found, the recorded state is a termination state and interpreting halts (or backtracks, if all behaviour has to be found). Formally, using the additional definitions:

I_t denotes the input event from which each transition of t starts,

O_t denotes the set of output events implied by each transition of t.

$\langle \rho, I_t, O_t, \sigma \rangle$ denotes the transition from ρ to σ for certain P_t, Q_t, $\rho \in P_t, \sigma \in Q_t$. υ is defined analogously.

we can define *coupling of two transitions* as

$$c(\tau, \upsilon) =_{def} \exists \rho \in Q_t, \exists \sigma \in P_t : \tau = \langle \_, \_, \_, \rho \rangle \wedge \upsilon = \langle \sigma, \_, \_, \_ \rangle \wedge \rho = \sigma \qquad (1)$$

denoting the fact that for each coupling there is at least one state to which a transition maps as well as *from* which a transition maps. This is also shown in figure 1. The equal signs have highest precedence, the underline characters (_) are a shorthand notation for

[7] This is identical to the *data-oriented guard* in [9].

"don't care": a singleton variable existentially quantified over its set. Definition 1 can be reduced to

$$c(\tau, \upsilon) =_{def} \exists \rho \in Q_t \cap P_u : \tau = \langle \_, \_, \_, \rho \rangle \wedge \upsilon = \langle \rho, \_, \_, \_ \rangle \tag{2}$$

Using definition 2, we lift the notion of coupling to the level of transition classes as follows

$$c(t, u) =_{def} \exists \tau, \exists \upsilon : \tau \in t \wedge \upsilon \in u \wedge c(\tau, \upsilon) \tag{3}$$

denoting the fact there is at least one transition of t which maps to a state from which a transition of u can start. See also figure 1.

There are two types of coupling between transition classes: *strong* and *weak*. For the strong coupling, each state as denoted by the post-condition is identical to a state in the set denoted by the subsequent pre-condition, so interpreting can always proceed to u. Strong coupling is defined as

$$\exists Q, \exists P : c(t, u) \wedge t = \langle \_, \_, \_, Q \rangle \wedge u = \langle P, \_, \_, \_ \rangle \wedge Q \subseteq P \tag{4}$$

For example, in figure 1 we get a strong coupling when $s(p) \cap s(q) \neq \emptyset$ and $s(p) \subseteq s(q)$. In this case an interpreter can always proceed from t to u. For the weak coupling, *not* every "post"- state is identical to a "pre"- state:

$$\exists Q, \exists P : c(t, u) \wedge t = \langle \_, \_, \_, Q \rangle \wedge u = \langle P, \_, \_, \_ \rangle \wedge Q \nsubseteq P \tag{5}$$

and interpreting cannot always proceed to u. Weak coupling is a matter of concern for a designer, as it may lead to termination or deadlock.

2.3 Connectability of transition-instances of a given EFSM

When two disjoint instances are representing coupled transition classes, the instances are connectable. We take care that only conditions at the bottom of instances may be connected to conditions at the top. The latter guarantees the atomicity of each instance, which is one step to obtaining atomic MSCs, as we will discuss later. To obtain this, the following syntactic constraints have to be satisfied:

1. *Each transition class is represented at least by the condition it maps from together with the condition it maps to.* When any of these items is missing, no transition class is represented. This enforces that no transition can be defined above the uppermost condition of an instance, nor below the bottom one.

2. *Only conditions at the bottom or at the top of transition-instances may imply couplings.*

Now the expression associated with a top condition becomes the pre-condition of the instance, and the expression associated with a bottom condition becomes the post-condition of the instance. For a set of MSCs matching the description above, only bottom- and top conditions of instances can be merged. The result is a kind of sequential composition, as shown graphically in figure 2.[8] The instance in MSC k is connectable to the instance in MSC l, because the condition identified by p_k can be merged with the condition identified by q_l. The graphical parts above the top condition and the graphical parts below the bottom condition do not represent a transition and they disappear from the resulting instance. An instance may also connect to itself, which represents the simplest case of cycling, or iteration [3]. Formally, with the additional definitions

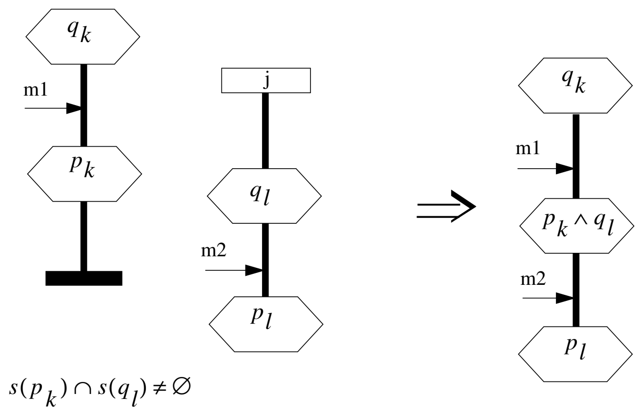

Figure 2: Example of how two instances may be connected

K denotes the set of all MSCs of the subject system. $k \in K$ and $l \in K$ denote an individual MSC.

e_k denotes the instance of the EFSM in MSC k,

e_l denotes the instance of the EFSM in MSC l,

b_k denotes the bottom condition of e_k; $b_k \in e_k$,

h_l denotes the top condition of e_l; $h_l \in e_l$,

[8] Only the relevant parts of the MSCs are shown, and the conditions have got the MSC identifier as second subscript.

and using definition 3, we can define the *connectability of two transition-instances* as

$$c(e_k, e_l) =_{def} \exists t \in T, \exists u \in T, \exists Q, \exists P :$$
$$Q = s(b_k) \wedge P = s(h_l) \wedge t = \langle \_, \_, \_, Q \rangle \wedge u = \langle P, \_, \_, \_ \rangle \wedge c(t, u) \quad (6)$$

We say that instance e_k and e_l are connectable. For connectability purposes, the *stop* of an instance is equivalent to a *disabled* post-condition. From such a condition only a transition to an initial state can be made. The *create* action denotes this.

2.4 Connectability of MSCs

Now we consider all EFSMs in a given system. For this purpose, we have to index all mathematical variables with the index of the EFSM they belong to. The ones we will explicitly refer to are given below:

J denotes the set of all EFSMs in the system. j denotes an individual EFSM; $j \in J$.

e_{kj} denotes the instance of j in MSC k,

e_{lj} denotes the instance of j in MSC l,

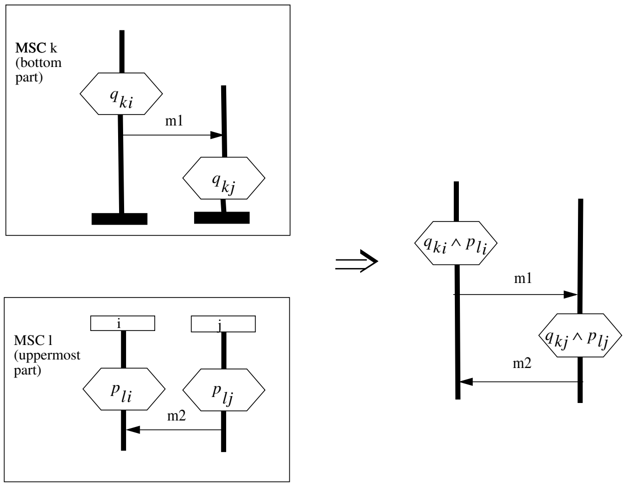

Figure 3: Example of graphically connecting MSCs

In the rest of this paper we assume that each MSC is *normalised*: It contains an instance of each EFSM in the system, and each instance satisfies the constraints on conditions presented above. If the original set of MSCs is not organised this way it can easily be re-arranged to get it as such. Also, we assume the environment of a system to be represented by an explicit instance in the MSCs[9], allowing us to make the *closed world assumption*: No other behaviour than the specified one will actually occur.[10] Two normalised MSCs may be connected to each other when *all* instances in the MSCs are connectable. Figure 3 shows an example. Finally, we assume that the set of MSCs is complete, i.e. all valid combinations of transition-instances have been included. In this way the complete state of the system and its environment is defined at all instances of interpretation — under the assumption that interpreters start at the initial state, of course.

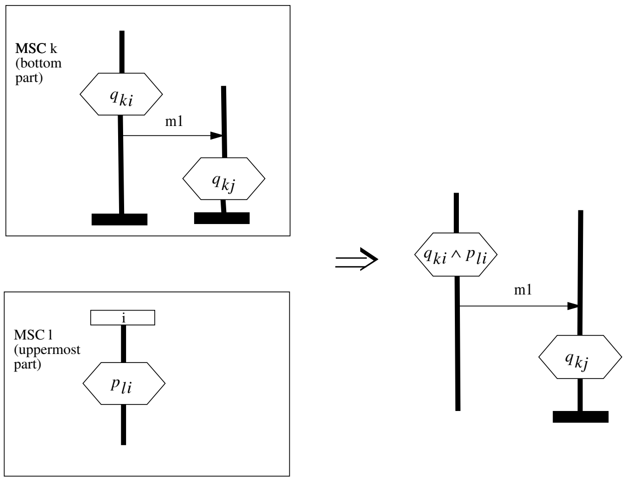

Figure 4: Example of a default connection

Now we relax the rule that a normalised MSC has to contain an instance of each EFSM, by applying the closed world assumption: *If — during interpreting — the instance of a particular EFSM is not defined in an MSC, then that EFSM keeps the state obtained earlier in the interpreting process.* When running into an MSC with an omitted instance, an interpreter has to record the state of the corresponding EFSM, as well as the post-condition interpreted last, and continue the normal interpreting process. As the recorded state always matches the last post-condition encountered, the latter is an imaginary pre-condition of the omitted instance as well as an imaginary post-condition. In the example of figure 4, condition q_{jk} is preserved while interpreting MSC l, and MSC k is connected to l. In this case MSC l has a *default connection* to MSC k for EFSM j. As the imaginary pre-condition is identical to the last post-condition, a default connection represents a strong

[9] This is no restriction, but just another way of representation as long as the environment can also be modelled as a set of communicating transition systems. This assumption is also made in [1].

[10] From this viewpoint, there is no difference between exception handling and regular behaviour: Both have to be specified.

coupling. Using definition 6, we can now define *connectability of MSCs* as

$$c(k,l) =_{def} \forall j \in J, \exists e_{kj} : \exists e_{lj} \Rightarrow c(e_{kj}, e_{lj}) \tag{7}$$

We say that MSCs k and l are connectable. Strong coupling of two MSCs occurs when for each EFSM a strong coupling holds. Weak coupling of MSCs occurs when each transition-instance is connected in some way or another, and when there is at least one transition-instance which is weakly connected. Otherwise, the MSCs are not connected.

Now we extend the interpreter: When it arrives at a post-condition in a particular MSC, it records the instantaneous state vector of the corresponding EFSM and continues until it has interpreted all remaining post-conditions of the MSC. Subsequently, it collects all MSCs which are connectable to the current MSC, and where the weak couplings do not result in termination, given the recorded state. Only after these MSCs have been found the interpreter can make a decision of how to continue. From the MSCs found, one is selected and the interpreting process continues. This way of operation — together with atomicity of the instances — ensures that branching to alternate behaviour occurs only at places where two or more MSCs are connectable, implying atomicity of each MSC, as was an objective of the method.

If there is more than one connection possible then the next MSC is selected in a way which is left undefined at this level of abstraction. If interpreting cannot proceed because there is no connection to a subsequent MSC possible, a termination state or a deadlock is detected.

3 The MSC Connectability Diagram

By introducing the coupling of transition classes, we prevented the number of MSCs from exploding, for the cases we worked on. After that, we could get an overview of the behaviour, which was constructed mechanically from the MSC connection information, in the form of the *MSC connectability diagram.* This type of diagram shows the ways MSCs can be connected: Each node in the diagram represents an MSC, and a connectability relation is represented by an unlabelled arrow, pointing from the post-condition side to the pre-condition side. An MSC connectability diagram can be drawn in a simple "flat" way, but in case of global reset states an explosion of arrows emerges. A hierarchical way of drawing suits such cases better.

An example is shown in figure 5. It represents the behaviour of an MHEG hyper-document presentation engine. Strong connectabilities are represented by solid arrows, weak connectabilities by dashed arrows. Initial points of interpretation are indicated by bullets.

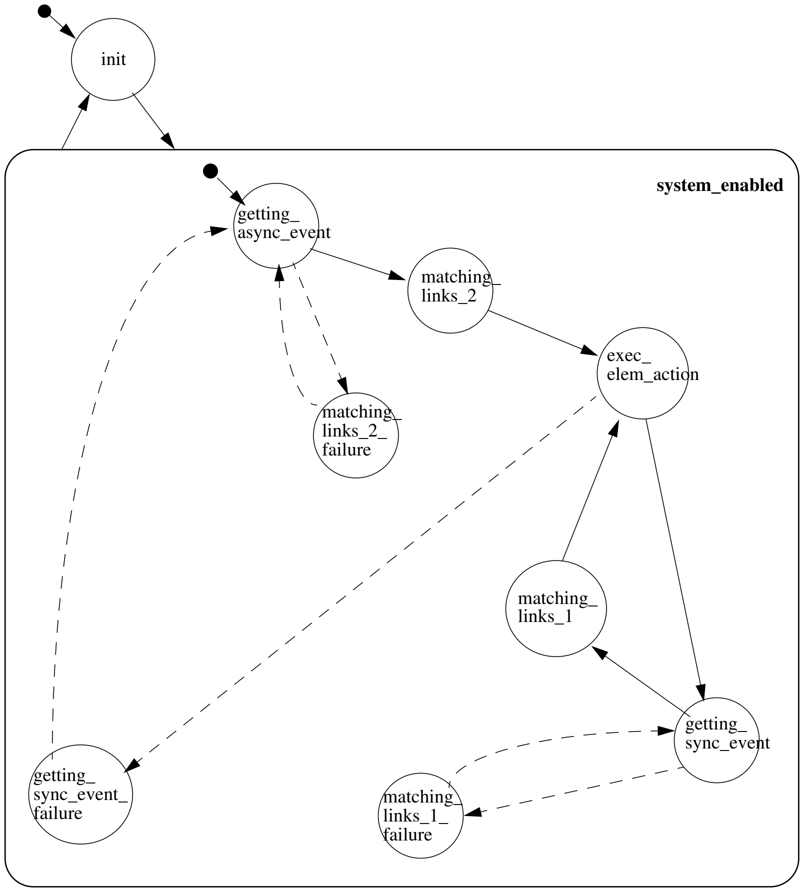

Figure 5: An example of an MSC connectability diagram

An MSC corresponding with one of the nodes is shown in figure 6.[11] For writing the state vector constraints, we used the syntax of conditional expressions in the "C" language. The post-condition of a process-instance may contain primed state variable names — i.e. postfixed with an apostrophe— representing the state a variable had when interpreting the top condition of the instance. This convention is also used in the COLD-K specification language used by a.o. Kelly [10, 4].

An MSC connectability diagram can be seen as a syntax diagram describing a class of regular expressions, where each MSC is a symbol in the alphabet. It can be used for constructing all required behaviour, at an abstract level: For a given system, it should be possible to project each required trace onto its MSC connectability diagram.

In the hierarchical diagram we could elegantly describe that an arrow can be drawn from any circle in the *system enabled* box to the *init* circle, using the transformations shown

[11] The MSC name is implicitly prefixed with "system_enabled".

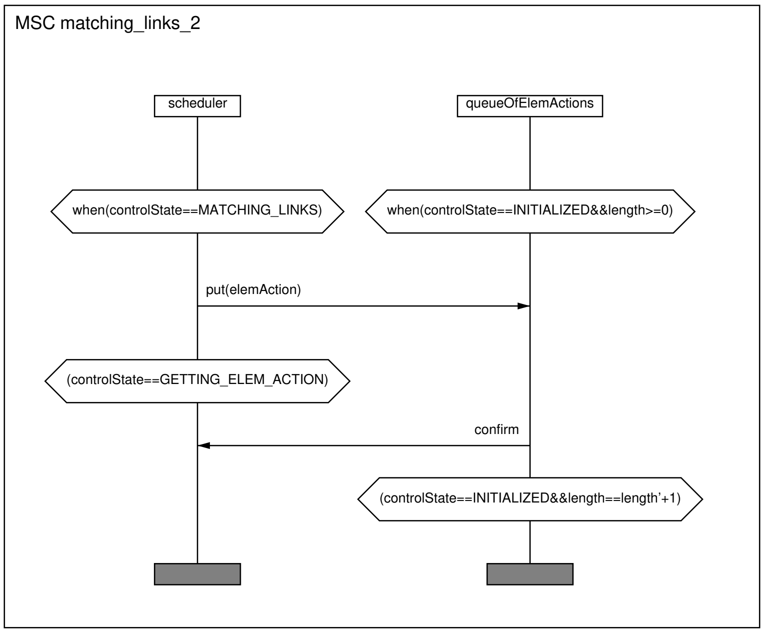

Figure 6: The matching_links_2 MSC

in figure 7. A semantic requirement is that each hierarchical MSC connectability diagram should be expandable into a non-hierarchical one, using the transformations.

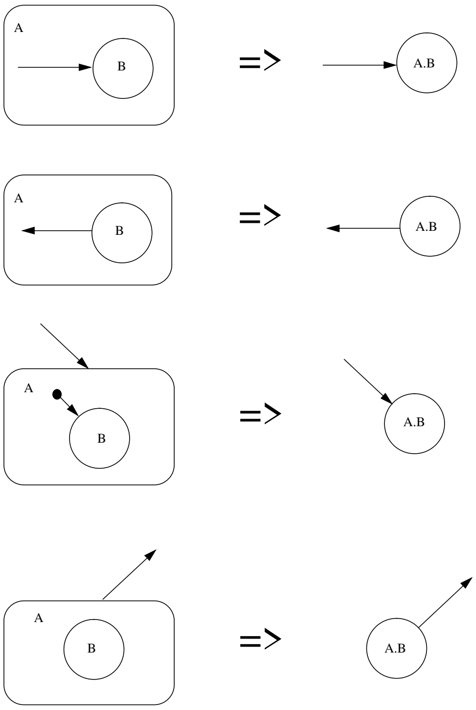

Figure 7: Types of transformations on an MSC connectability diagram

4 Practical Work

We have made a prototype tool kit for constructing an MSC connection diagram, given a set of normalised MSCs. As figure 8 shows, it consists of a translator, translating a textual MSC — as produced by commercial MSC editors — into a set of Prolog facts (which are a kind of existentional proposition). A declarative programming language —like Prolog— was the best choice for the prototype as the rules defining connectability can be easily written in it. A file was prepared with additional declarations about the connectability of the instances in the MSCs.[12] A second program used the facts along with the additional declarations for constructing a table of MSC connectabilities. This table could be produced as plain text but also in Dot [11] format, for mechanically producing a flat MSC connectability diagram. A hierarchical version was constructed from the latter by hand, but there exist methods for mechanising this too.

When designing a system, the pre-conditions p_{t_j} were refined to become *weakest* pre-conditions [7], and the post-conditions q_{t_j} to become *strongest* post-conditions, to minimise the number of cases of possible termination/deadlock.

These techniques and tools have been used for modelling the MHEG presentation engine which is used as example, as well as for developing the Digital Video Broadcast (DVB) Interaction Channel protocol, currently an ETSI standard [2]. They proved to be of great value, especially by making missing and unwanted connectabilities visible.

[12] A comparable approach has been taken in [10] for mapping MSCs into the ARC specification language (a superset of COLD-K).

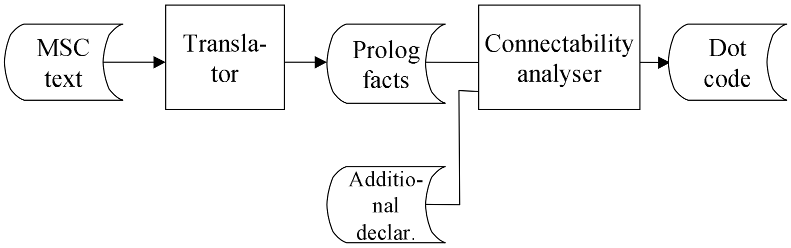

Figure 8: The prototype tool kit

5 Conclusion

We have described a method where each MSC represents a transaction, and where the connectability of MSCs indicates how transactions may be sequenced during operation of an implementation. The method is orthogonal to the High-level MSC notation as the latter is of a higher level of abstraction. Our method directly addresses local conditions in individual MSCs, where connections between MSCs are a consequence of the relations between the local conditions, thereby introducing a bottom-up development aspect which may also be useful for analysing existing systems.

It is hoped that the work presented here helps in making the MSC notation even more versatile than it already is. Engineers working on the design of concurrent systems will especially benefit from the method. Experience has shown that explicit presentation of the event order prevents the introduction of spurious behaviour, and/or the omission of required behaviour. Experience also showed that knowledge of the required states of the various components in a system is an excellent guide for design of a system's general behaviour. Future work might be the incorporation of the MSC coregion construct, and constructing transition(-class) diagrams of a system's components, given a set of normalised MSCs. This work has a relation with [6].

Acknowledgements: The author expresses his great debt to Andre Engels (TU Eindhoven), Loe Feijs (TU Eindhoven/Philips Research), Rob Udink and Roli Wendorf (both at Philips Research), for their help and constructive criticism during the preparation of this paper.

References

[1] Engels,A., Mauw,S., and Reniers,M.A. A hierarchy of communication models for mes-

sage sequence charts. In *FORTE/PSTV'97 proceedings* (1997), Chapman & Hall, pp. 75–90.

[2] ETSI. Digital video broadcasting; interaction channel for CATV. Tech. Rep. ETS 300 800, European Telecommunications Standards Institute, 1998.

[3] Feijs, L. Synchronous sequence charts in action. *Information and Software Technology*, 39 (1997), 583–606.

[4] Feijs, L.M.G., and Jonkers, H.B.M. *Formal Specification and Design.* No. 35 in Cambridge Tracts in theoretical Computer Science. Cambridge University Press, 1993.

[5] Feijs, L.M.G., and Mauw, S. MSC and data. In *Proceedings of the 1st Workshop of the SDL Forum Society on SDL and MSC* (1998), Humboldt-Universitaet zu Berlin. Informatik-Bericht Nr.104.

[6] Graf, S., and Saidi, H. Construction of abstract state graphs with PVS. In *Conference on Computer Aided Verification CAV 97* (1997), LNCS 1254, Springer Verlag, Springer Verlag.

[7] Gries, D. *The Science of Programming.* Springer, 1981. ISBN 3-540-96480-0.

[8] Haugen, O. MSC-2000 highlights. Tech. rep., SDL Forum, 1999.

[9] Haugen, O., et al. ITU recommendation Z.120, message sequence charts. Tech. rep., ITU, nov 1999.

[10] Kelly, D. ARC: A specification language for communicating systems. *Computer Journal 41*, 6 (1998), 386–401.

[11] Koutsofios, E., and North, S.C. Drawing graphs with Dot. Tech. Rep. http://www.research.att.com/sw/tools/graphviz/dotguide.pdf, AT & T, 1996.

[12] Mauw, S., and Reniers, M.A. An algebraic semantics of basic message sequence charts. *The Computer Journal* (1994).

[13] Mauw,S., and Reniers,M.A. Operational semantics for MSC'96. *Computer Networks*, 31 (1999), 1785–1799.

[14] Meijs, F. Message sequence chart enhancements. Tech. Rep. RWB-506-ir-95071, Philips Research Labs Eindhoven, Nov 1995.

[15] Reniers, M. *Message Sequence Chart, Syntax and Semantics.* PhD thesis, Technische Universiteit Eindhoven, 1999.

[16] Rudolph, E., Graubman, P., and Grabowski, J. Tutorial on message sequence charts. *Computer Networks and ISDN Systems 28*, 12 (1996), 1629–1641.