Connecting Message Sequence Charts

Frans A. Meijs,
Philips Research Labs Eindhoven

SAM 2000, June 26
RWB-506-pr-0001

Author is member of the Interoperability cluster in the Information and Software Technology sector.

He has been working last years on the subject of networked home appliances, with side steps to methodology.

The problem

- During design of concurrent systems, MSC-s with local conditions were used as scenarios (for avoiding the emergence of spurious behaviour). Basic MSC syntax was used.
- Exploding number of MSC-s when attempting to describe complete behaviour

•The systems were: An MHEG presentation engine (communication between engine's components), and DVB Broadcast Interaction Channel (communication between server and consumer’s station).

•Reason for using MSC-s in this phase of development was our experience that unintended behaviour arises easily when only SDL diagrams or stateCharts are used. MSC complements SDL here by making the event ordering explicit.

•Local conditions were used because we found that considering message exchange alone is in many cases insufficient for keeping track of the required behaviour - especially exception handling - and because selection of proper local conditions is a major step towards proper implementation. For the rest the basic MSC syntax was used.

•When doing so, the problem occurred that we were obliged to do unwieldy things like drawing 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 MSC-s to be drawn exploded because we had to keep track of state variables with many possible values.

Essentials of our improved method

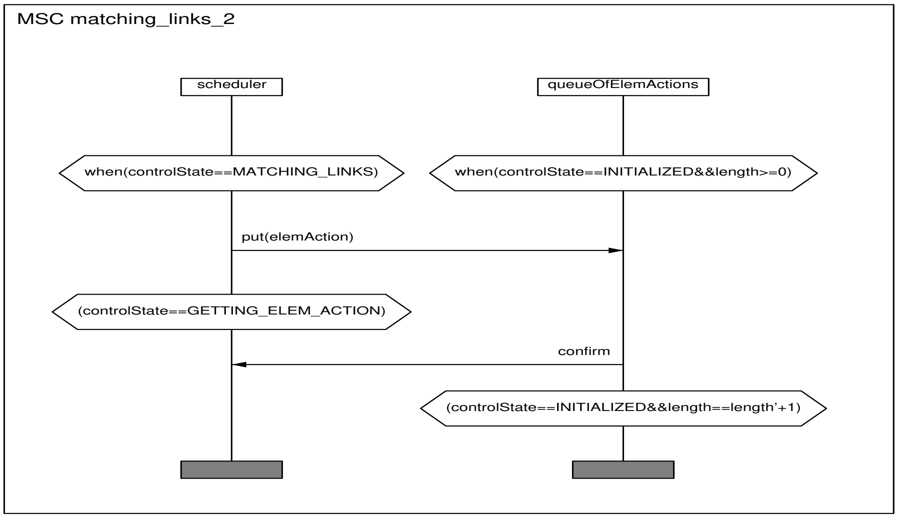
- Use of conditions as pre- and post-condition of basic actions.
- Basic actions are used for composing all behaviour, like an alphabet is used for assembling a language.
- Each MSC is normalised such that it represents a basic action (transaction).

•A method was needed which would avoid the explosion problem. Also, we wanted to stay as close as possible to the method we were using already, to keep MSC-s clean and simple.

•Given these requirements, one of the options which looked promising was a further exploitation of the local condition, making it to become a pre- or post-condition of a basic action, and where a set of these basic actions can be used for composing the total behaviour, like an alphabet is used for assembling a language. An action is a basic one when it is undisruptable, otherwise it could be decomposed into still smaller actions.

•Now we would like each MSC represent a *transaction* - i.e. an undisruptable action involving one or more system components. The set of transactions is a concise base from which all behaviour of the system can be assembled. We define an MSC representing a transaction as being *normalised*. We decomposed the original scenarios until we got a set of normalised MSC-s.

The normalised MSC



•Here we show a normalised MSC. To obtain each instance representing an undisruptable action, the following constraints are satisfied:

• Each transition class is represented at least by a pre-condition and a post-condition. When any of these items is missing, no transition class is represented. This ensures that no transition can be defined above the uppermost condition of an instance, nor below the bottom one.

• A condition at the bottom of an instance may connect to a condition at the top of an instance. No other connections should be possible.

Essentials of the method, contd.

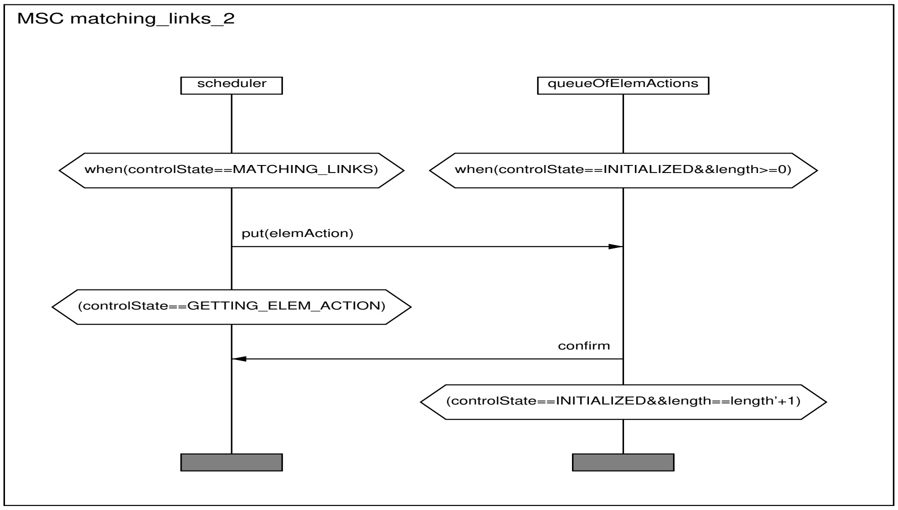
- Behaviour is sequentially composed by connecting transactions. Connection is governed by pre- and post-conditions.
- Transaction = undisruptable set of transitions.
- By introducing pre- and post-conditions, an instance may represent a *class* of transitions instead of a single transition.

•Assembly of behaviour is done by connecting - i.e. sequentially composing - transactions. Connection can be done when the post-conditions of the current transaction intersect with the pre-conditions of a candidate one. A simple tool makes the connectabilities explicit, given the pre- and post-conditions. Details are given shortly.

•We assume that each instance represents the behaviour of an Extended Finite state Machine. Usually, an EFSM is defined by transitions from one state vector to another. The transition is the basic action of the EFSM. An undisruptable set of transitions makes up a *transaction*.

•There is another useful aspect of the pre- and post-condition: An instance with a pre-condition and a post-condition gives us the possibility to describe a *class* of transitions. A class is a set whose elements satisfy a common constraint. By describing classes of transitions instead of individual transitions we get another means for getting a concise description of behaviour.

The normalised MSC, contd.



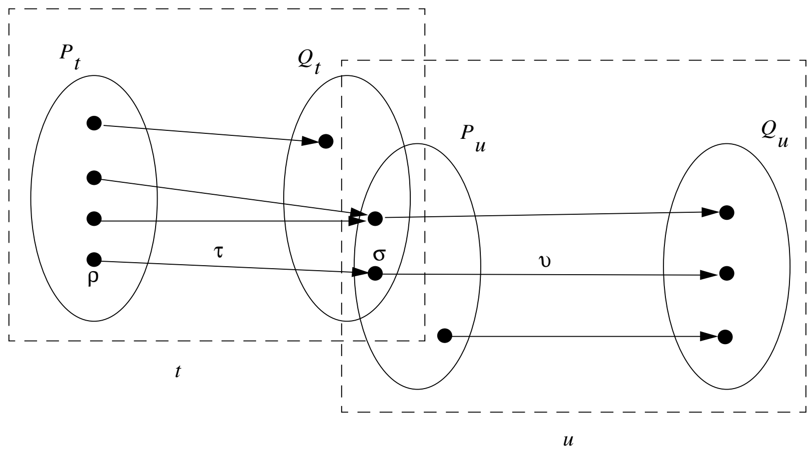
•In the normalised MSC, the *when*- clauses are the pre-conditions, the expressions between parentheses in the bottom conditions are the post-conditions. We used the syntax of conditional expressions in the ``C" language.

•The post-condition of an instance may contain *primed* state variable names - i.e. post-fixed with an apostrophe - representing the state the variable had when interpreting the pre-condition of the instance. This is one way of defining a class of transitions. Without the primed variable we had to write an MSC for each value of *length* (or to use an action box with imperative text). The other way is by defining inequalities or disjunctions of equalities.

Structure of the rest of this presentation

- Some details:
  - Coupling of transitions.
  - Connectability of entire MSC-s.
  - The MSC connectability diagram
- Concluding remarks

Connectable conditions



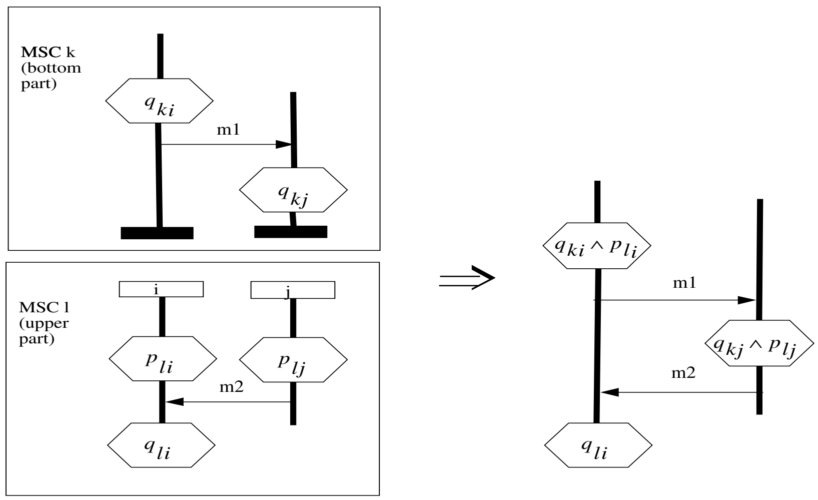
•Now we discuss the basic theory about connection. In our context, a *class* is a set whose elements satisfy a particular constraint.

•In the example, *t* and *u* each denote a transition class, *Pt* and *Pu* each denote a set of state vectors defined by the pre-condition of *t* and *u* respectively, and Qt and Qu each denote a set of state vectors defined by the respective post-conditions.

•A transition 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.

•When for a given post- and pre-condition the sets they define have a non-zero intersection, we say that the conditions are *connectable* and that the sets are *coupled*. In this case there is at least one state where a subsequent transition is possible.

Connectability of MSC-s



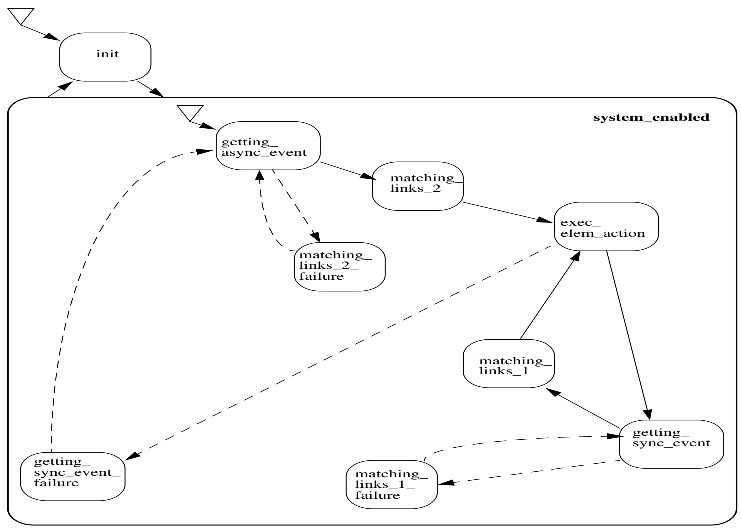
•Now we consider all EFSM-s in a given system. We assume that the normalised MSC-s have an instance for each EFSM in the system. Also, we assume the environment of a system to be represented by an explicit instance, and that the set of MSC-s is *complete,* i.e. all valid combinations of (transaction) instances have been included. This allows us to make the closed world assumption: “No other behaviour than the specified one will actually occur”.

•When an MSC interpreter arrives at a post-condition (e.g. q_k), all MSC-s of the system are searched for connectable pre-conditions, i.e. which evaluate to *true* for the instantaneous state vector.

•Two MSC-s are connectable when all instances in the MSC-s are connectable, as shown in the example. Connection is represented by temporarily merging the conditions: They are replaced by a single condition, having the conjunction of the original expressions. The conjunction represents the intersection of the state sets.

•By making the pre-conditions as weak as possible, we minimise the number of MSC-s. By making the post-conditions as strong as possible, we minimise the number of couplings.

A bonus: The MSC Connectability Diagram



•Given a set of normalised MSC-s, we used a prototype tool for generating a table of the connectabilities. This tool was also used for generating a MSC connectability diagram (MCD).

•This diagram proved to be invaluable in detecting missing or unwanted connectabilities: 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. Arrows representing weak connectabilities are dashed.

•Referring to the set of normalised MSC-s as an alphabet, the MCD is the corresponding syntax diagram.

•An MCD can be drawn in a simple ``flat" way, but in case of global reset states an explosion of arrows emerges. A nested way of drawing suits such cases better. As an example, the figure represents the behaviour of an MHEG hyper-document presentation engine. Initial points are indicated by triangles. The nested way of drawing describes elegantly that an arrow can be drawn from any node in the *system_enabled* box to the *init* node, reducing the number of reset arrows from 8 to 1.

•Note that the MCD is not an HMSC. It also does not have the same purpose: It is just an analysis aid. The set of normalised MSC-s proper is sufficient for describing the complete behaviour.

Conclusion

- Each MSC represents a (class of) transaction
- Connectability indicates how transactions may be sequentially composed.
- Builds on MSC 2000, orthogonal to HMSC.
- More concise description of behaviour w.r.t. scenarios.
- TBD: Constructing transition diagrams

•Summarising: We have arrived at a method where each MSC represents a class of transactions, and where the connectability of the MSC-s indicates how transactions may be sequentially composed.

•We use the MSC 2000 data-oriented guard as pre-condition and a small extension to the condition syntax for the post-condition.

•The method is orthogonal to the method supported by the High-level MSC notation as the latter is of a higher level of abstraction: In HMSC-s global conditions are used for connecting MSC-s. As such the HMSC notation supports a top-down way of development. In contrast, our method addresses local conditions in individual MSC-s, where connections between MSC-s 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.