

Comparing UML 2.0 Interactions and MSC-2000

Øystein Haugen

Department of Informatics, University of Oslo
oysteinh@ifi.uio.no

Abstract. This paper is a brief comparison between the Interactions of UML 2.0 as defined by the Final Adopted Specification (OMG ptc/03-07-06) and MSC-2000 as defined by Z.120 (ITU 1999). The comparison investigates whether UML 2.0 has serious shortcomings relative to MSC-2000. The paper also discusses whether MSC-2000 is still needed in the future or should be retired.

Introduction

Message Sequence Charts emerged from the SDL (ITU-T Specification and Description Language) community leading to its first ITU-T recommendation in 1992 [1]. Later there have been revisions of MSC in 1996 [2] and in 2000 [3].

UML 1.0 appeared in 1999 [4] and it did have some simple sequence diagrams similar to those found in MSC-92. UML went through small revisions leading to UML 1.5 in 2003. Still over the last three to four years a major revision of UML has taken place leading to UML 2.0 [5] which will become an available technology from OMG (Object Management Group) in 2004. In UML 2.0 also sequence diagrams (or Interactions) have been thoroughly revisited and revised.

Having led the work towards MSC-2000 as ITU-T Rapporteur, and towards UML 2.0 Interactions as the editor of that chapter, we think it may be of interest to give a comparison between the two. For obvious reasons many of the same requirements have affected the direction of the developments, but the differences in standardization style between ITU-T and OMG have also had influence [6, 7]. The first version of this paper has been input to the ITU SG17 discussions, but we thought the comparison could also be useful for a wider audience.

To read this paper it is helpful to have good knowledge of either MSC or UML sequence diagrams. This paper is not a tutorial for any of these languages.

The paper is organized as follows. First we give a comparison table that will give the readers a quick summary of the differences between MSC and UML Sequence Diagrams, and present the terms used. Then we present the diagrams and show examples of similarities and differences. We go in some detail into the important fragment concepts. Details of messages are commented before the areas of data and time are considered at greater length. Finally there is a summary and conclusion.

Comparison table

The comparison table is intended to give overview of the different central concepts in the field of Interactions / MSC such that those only familiar with one of the languages can see what terms have been used in the other language.

MSC-2000	UML 2.0	Comments
MSC (Message Sequence Chart)	Interaction (Sequence Diagram; Communication Diagram; Timing Diagram; Interaction Overview Diagram)	The individual scenarios. MSC and UML have different approaches to language.
Event	EventOccurrence	
MSC Document	Class (or Collaboration)	The context of the scenarios.
HMSC	Interaction Overview Diagram	Conceptually they are scenarios, but with a different syntax.
Instance	Lifeline	Notice that a lifeline refers to a property (part) of a composite structure, while the instance is a part of a structure
Message	Message	Both distinguish between asynchronous and synchronizing messages
Method call	Operation call	
Method (area)	ExecutionOccurrence	
Action	ExecutionOccurrence	
Suspension (area)	No direct counterpart	
Gate	Gate	In UML we have only message gates, while in MSC there are also general ordering gates.
No direct counterpart	Interaction fragment	See Interaction fragment
Inline expression	Combined Fragment	UML 2.0 has introduced more operators.
Coregion	Coregion	In MSC this is a basic concept from 1992, but in UML 2.0 this is only presented as a shorthand for the par-operator. No semantic difference.
MSC reference	Interaction Occurrence	The ability to refer to another interaction. See also Referring another

		interaction / MSC diagram
Decomposition	PartDecomposition	How the aggregate hierarchy of the structure is reflected in interactions/MSCs..
General Ordering	General Ordering	UML does not have general ordering gates.
Condition (global state)	Continuation	This concept is basically a label being a syntactic way to combine pieces of the specification that are distributed
Condition (predicate)	Interaction Constraint	
Relative time	Duration	
Absolute time	Time	
Time measurement	TimeObservationAction, DurationObservationAction	
Timer	No counterpart	

Diagrams and concepts

MSC, in the tradition of the ITU languages, considers the graphical syntax tightly coupled to the concepts. In UML, however, one has tried (not always successfully) to distinguish between the concept and the presentation forms. The latter approach resembles what was done for SDL, where there was an abstract syntax on which the semantics was explained. In UML the abstract syntax is defined by what is called a metamodel. The metamodel is described in a subset of UML itself called MOF (Meta Object Facility)

MSC has two graphic forms, simple message sequence charts (**Fig. 1**) and high-level message sequence charts (**Fig. 4**). MSC also has a supposedly equivalent textual form.

In UML the concept referred in this paper is called *Interaction*. UML Interactions come in several graphic forms with different foci. The most expressive form is the *sequence diagram* (**Fig. 2**) and every concept of Interactions can be expressed in sequence diagrams.

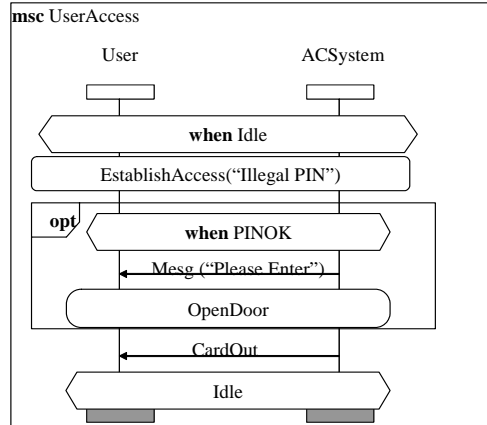


Fig. 1. Simple MSC-2000 diagram

The **Fig. 1** shows an MSC-2000 diagram which is a piece of the specification of an access control system. We see initial global condition, a reference, messages, an optional inline expression including yet another initial condition, a message and a reference. The diagram ends with a setting condition.

An almost exact UML counterpart is shown in **Fig. 2**.

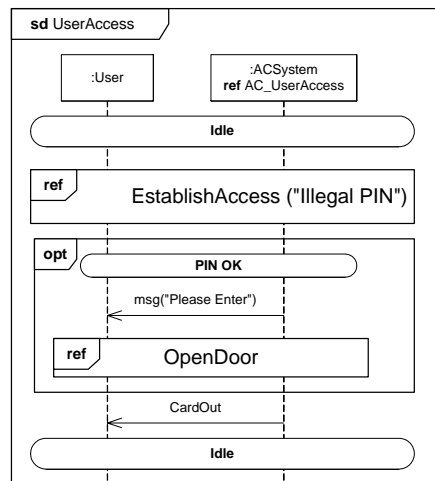


Fig. 2. Simple UML 2.0 Sequence Diagram

A continuation is at the top of the diagram followed by an interaction occurrence and a combined fragment that includes another continuation, a message and another interaction occurrence. The whole diagram ends with a continuation. The reader may

notice that continuations do not distinguish between setting and guarding as is done with MSC conditions.

Then there is the *communication diagram* (**Fig. 3**) that gives an overview of how simple communication goes between the lifelines. It is overloaded on a composite structure diagram (which closely resembles an SDL-96 block diagram [8]) where the messages are numbered and shown on the connectors (communication lines). In UML 1.x the communication diagram was called a collaboration diagram, but the term “collaboration” was inadequately overloaded and it was decided to rename the type of diagram that actually described an interaction. In UML 2.0 “collaboration” is a term for a special kind of classifier – a kind of generic class concept.

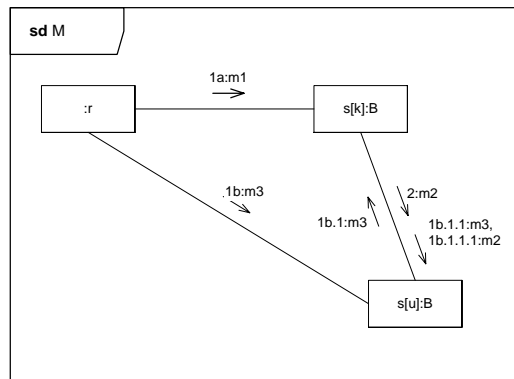


Fig. 3. UML 2.0 Communication Diagram

UML 2.0 has also a counterpart for the HMSC diagrams (**Fig. 4**), and it is called *Interaction Overview Diagram* (**Fig. 5**). For detailed interactions, the specification of each individual event is important, but there is often a larger picture where the general control flow is the most significant. For this purpose HMSC is a graphical form of MSCs that can be referred from simple MSCs and can themselves refer simple MSCs.

The HMSC in **Fig. 4** has the same semantics as the plain MSC in **Fig. 1**. We notice that the conditions serve as nodes in the flowgraph. Other nodes are MSC references and the start and end symbols (triangles). In addition there is a circle that only serves as a join for graphical lines.

HMSC and Interaction Overview Diagrams are quite similar. The UML 2.0 variant can also have inline interaction diagrams (of any kind) as nodes. This is slightly more general than MSC.

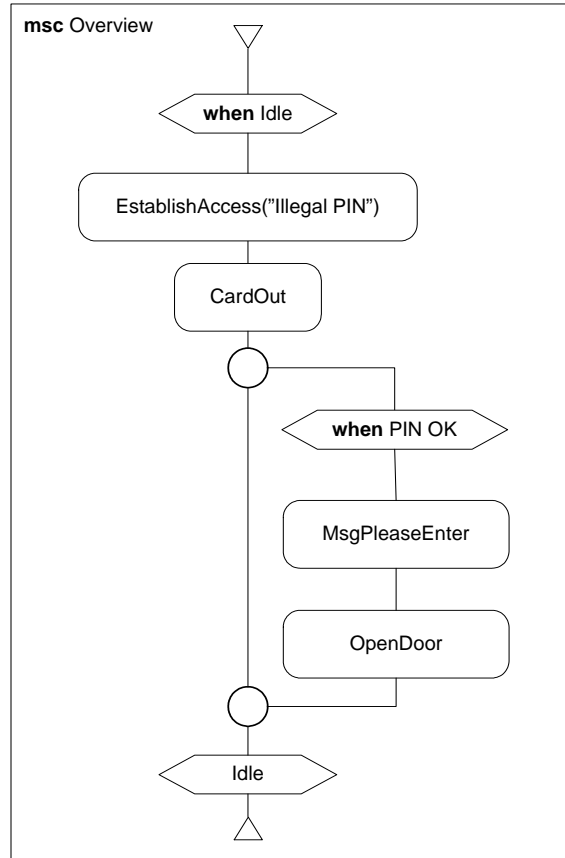


Fig. 4. HMSC – High Level MSC

We see this in **Fig. 5** where continuations come in place of MSC conditions, and interaction occurrences or inline sequence diagrams replace the MSC references. Since the UML Interaction Overview diagrams follow the UML activity diagram syntax there are other supplementary symbols to describe branching, start and end.

In UML 1.x there was already activity diagrams that on a very rough abstraction level served the same purpose as HMSC. Therefore it was natural to try and reuse notation from activity diagrams when designing the UML 2.0 Interactions counterpart of HMSC (**Fig. 5**).

Still UML 2.0 Interaction Overview Diagrams are understood as Interactions rather than activities. This is important since activities are understood through Petri-net-like semantics while Interactions are understood through trace semantics.

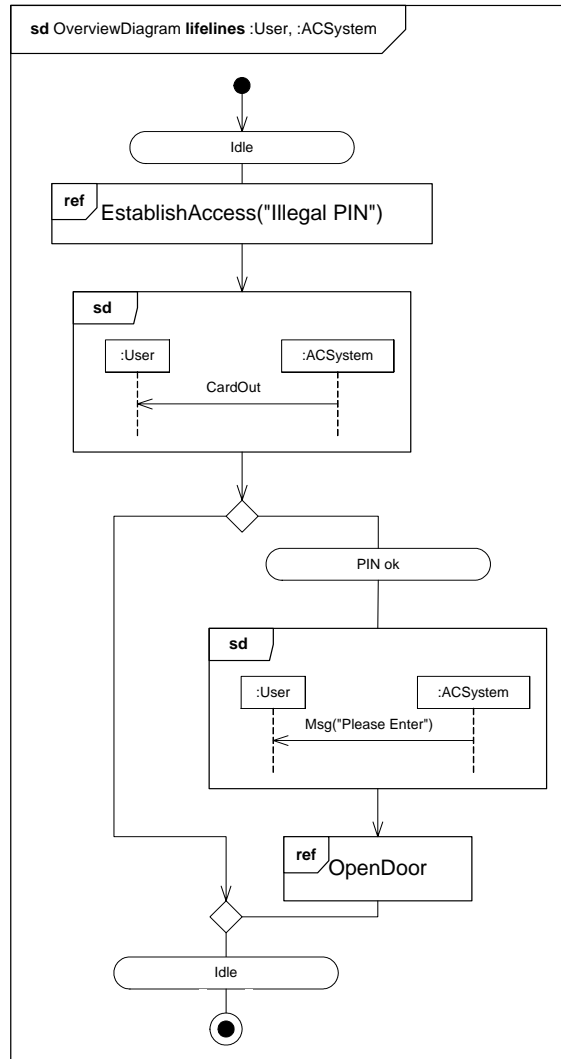


Fig. 5. UML 2.0 Interaction Overview Diagram

Finally UML 2.0 has the timing diagram included for the purpose of focusing on timing issues. It is an interaction diagram even though the timing concepts have been made applicable not only for interactions.

In UML concepts may even exist without syntax, which means that it is up to the tool how to present the concept to the user – and this will often result in values given in a dialogue box. In MSC on the other hand, every concept has concrete syntax.

In UML 2.0 unlike in UML 1.x, the diagrams have a frame and a name like in MSC. UML has a textual form (XMI) which is the standardized format for model exchange.

Interaction fragment

The UML concept Interaction fragment is only implicitly present in MSC. The more specific concept of MSC Inline Expression corresponds to UML 2.0 Combined Fragment.

This is a minor point, but introducing “interaction fragment” makes it a little easier to explain the semantics in a compositional way. The interaction fragment is the central recursive concept.

Combined fragments / inline expressions

The following table is a summary of operators of combined fragments. The reader will have to look up the detailed definition of an operator in the standards.

MSC-2000	UML 2.0 Interactions	Comments
Loop	Loop	iteration construct
Opt	Opt	optional behavior
Seq	Seq	weak sequencing
Par	Par	parallel merge
Alt	Alt	alternatives
Exc	Break	Exception as a special case of alternative. There is no semantic difference between MSC and UML here.
	Strict	For traces where the order is given by vertical coordinates also between lifelines
	Neg	Negative traces. Introduces a more complex semantics
	Critical	Critical region
	Ignore/consider	Filtering of message types
	Assert	A way to define what has to happen in a given situation

Furthermore in UML 2.0 it is possible to combine operators directly in the operator area as shorthand for nesting.

The operators are as the table above shows almost identical, but UML 2.0 has added a few. **Neg** and **assert** are introduced to make sequence diagrams more suitable to express requirements that are more absolute. The **neg** operator defines those traces that should not occur, and the **assert** defines the traces that should (mandatory) occur at a given point in the scenario. These operators are intended to bring the essence of Live Sequence Charts [9] into the compositional semantics of UML 2.0 interactions.

In the future we will probably see even more operators. One suggestion has been to distinguish between specifying potential alternatives and mandatory alternatives [10]. Others want to make interactions able to describe Java exceptions properly.

Conditions / Continuations

Already in MSC-92 we had the concept of “conditions”. The term indicated that we were talking about predicates that had to be true for some behavior to take place. This turned out to be a slightly incorrect intuition and in MSC-2000 the condition concept came in two variants – the MSC-92 variant, which is really a label, and the guarding condition (predicate).

UML did already in its version UML 1.x have constraints and guards. The constraints could be put anywhere in the model, and guards could be put on messages to indicate when the executions could follow that message.

In UML 2.0 we have in the spirit of MSC-2000 included both the labeling variant of conditions called “continuations” and the predicate, guarding variant called “interaction constraints”.

Disregarding the difference in terms, the concepts are comparable. The UML 2.0 concept “continuation” is deliberately made more narrow than the MSC-2000 concept “global state condition”, but in practice they will serve the same purpose, namely to combine parts of the specification that are distributed for other reasons. Typically such combination is needed when the branching of control occurs within one diagram, and this branching should be continued in another diagram without having to check again for the same condition

Referring another interaction / MSC diagram

To refer to another interaction from within a diagram is one of the most important structuring mechanisms, and the first to be asked for by the users.

The concept of MSC reference of Z.120 and Interaction Occurrence of UML 2.0 are almost identical. In its basic form they can both be understood by substituting the referred diagram into where the reference was.

MSC also features “reference expressions” where the text of the MSC reference can designate an expression like an inline expression. This may be understood as textual shorthand for more voluminous graphics. UML does not have a direct counterpart of this feature.

Context of the scenarios

MSC defines a concept of its own “MSC Document” to define the context of an MSC (the individual scenarios). This context defines a composite structure of the instances playing in the MSC. The MSCs of the MSC Document are divided in *defining* and *utilities*. The defining MSCs are those intended to define the semantics of the MSC

Document while the utilities are merely used directly or indirectly by the defining ones.

This is very similar to UML where the context may be a class or a collaboration. The latter is for more generic interactions. A class or a collaboration have a composite structure restricting the possible lifelines of the interactions. The composite structure of classifiers in UML is an innovation from version 2.0 inspired by SDL block diagrams and ROOM [11] structures. Composite structures may have ports on the edges and they may also be represented by lifelines in interactions [12]. To tie Interactions closely to the composite structure was a significant simplification step relative to earlier versions of UML where the participants of Interactions had lived their own life relatively unaffected by those specified in other parts of UML.

In UML there is no distinction between defining and utility behaviors, but a class may define a “classifier behavior” which is that designated behavior of an object of that class. This would be similar to a defining MSC. A classifier behavior in UML may be any behavior, which means it can be a state machine or an activity.

In UML, the class (collaboration) has a number of other purposes than serving as the context for interactions, and as such defines the bridge between the interaction part of the language and other parts like state machines and activities. With MSC, it is necessary to assume a mapping between concepts in MSC and corresponding ones in SDL. This mapping is, however, mostly trivial.

Decomposing the structure

In MSC, the MSC Documents represent an aggregate hierarchy of structure, and in UML 2.0 classes and collaborations may have composite structures which also represent an aggregate structure.

The natural question is what happens to a scenario specified for a high aggregate level when the constituents are decomposed.

The concept in MSC and that of UML are designed to be as similar as possible since the UML concept was modeled according to the MSC counterpart.

As a graphical option, in UML 2.0 lower aggregate levels can be shown directly inline under a Lifeline representing a higher aggregate level. Thus a sequence diagram may contain lifelines on different aggregate levels, which is impossible in MSC.

Messages

UML 2.0 and MSC do not have exactly the same tradition with respect to what kind of scenarios they have been used to describe. While UML has a tradition of using interactions to describe the control flow of a sequential program, MSC considers a set of entirely concurrent instances where the method calls are considered remote procedure calls.

This difference in tradition does make some difference in requirements, but neither language should have any restriction limiting their usage in this respect.

Technically MSC considers method calls and replies a concept area of its own, UML 2.0 considers operation calls as just one kind of message. However, in both

cases the semantics is given by the traces of the events leading from the initiation of the operation call (MSC: method call) to the reception of the reply.

The difference in tradition is also highlighted by UML usage of interactions where also passive objects are depicted as lifelines. In MSC passive objects would always be modeled as variables of the active instances.

Therefore what UML users sometimes describe using messaging, MSC users would only consider as operations on variables. In the UML community we have therefore experienced a very strong need to have data guards for alternatives that cover more than one lifeline since it is in practice possible to determine that the covered lifelines are never concurrently executing.

Suspension area

The suspension area is a part of the lifeline where no events should appear (with certain strict restrictions). This is given explicitly and is not always the case when there are method calls. This is because method calls can appear from different sub-lifelines if the lifeline represents a composite structure. That would mean that even when there is a method call the whole object will not have to wait for the reply.

In UML 2.0, the suspension region concept was considered, but removed at a late stage of the process.

Data concepts

The approaches to data in MSC and in UML are both similar and different.

In both languages it is considered desirable that the users can choose to apply the data manipulation language of his preference rather than a standard data-manipulation sub-language of the modeling language. This is contrary to SDL (Z.100), which does have a data language of its own.

Data in MSC-2000

The data concepts of MSC-2000 is characterized by a rather elaborate scheme designed to give the user the necessary freedom of expression without sacrificing precision and formality.

In order to let the users write the data portions without being hampered by all kinds of escape notation, there is a way to define parts of the syntax of the preferred data language within MSC-2000. Through declarations it is possible to define characters for parentheses etc. It is recognized that most data languages have a nested structure where parenthesizing is important. It is possible to declare syntax for nestable as well as non-nestable parentheses. Through these “meta” declarations of syntax, it is possible for the user to write data expressions in his favorite data language without any extra wrapping syntax, and still the MSC analyzer can extract the appropriate data strings in a general way.

For the MSC analyzer to “kick down” to analyzing the extracted data strings, a number of interface functions are defined. Some of these functions controls the static requirements of the data language such that the MSC analyzer can apply these functions which must be defined for the data language used.

Finally there are a few functions that are required to define the dynamic semantics of the combined MSC and Data language. These functions represent a way to parameterize the semantics of the MSC language modulo different data languages.

The ITU has standardized such a binding between SDL data language and MSC in Recommendation Z.121 [13]. This formally combines the data aspects of SDL and MSC tightly together.

Data in UML 2.0

Data is in principle an integrated part of the UML modeling language, but there is little UML-specific rules for concrete syntax. Most of this must be found through examining the chapters on Actions (and Activities) in UML 2.0.

There is no UML concrete syntax for actions, and there is no counterpart to the language specific way in which the syntax can be defined within the language as one can find in MSC.

There has been talk about defining concrete syntax for an action language, but nothing concrete has come out of this yet.

Data summary

MSC has a reasonably well founded formalization of how different action/data languages can be combined with core MSC, but this is a relatively elaborate scheme not easily conceived by the users. Furthermore the MSC approach to data is not directly aligned with SDL (representing the more imperative style of modeling within the same tradition). The connection to SDL is taken care of through defining the necessary interface functions for SDL as required by MSC described above. This task is done but still not implemented in any publicly available tool.

Formally associating data with the lifelines of the interaction is necessary to produce any formal analysis or model transformation for practical purposes. In practice both MSC and UML 2.0 tools are adapting programming languages (such as Java or C) as their action language. The connection with the rest of the standard language is done ad hoc and not necessarily following the principles of the standards.

Time Concepts

MSC-2000 defines time concepts and mechanisms to define constraints on absolute and relative time related to events. This is also the case in UML, but in UML time constraints are found not in the section on Interactions, but in the section on Common Behavior intended to hold for all of UML, not only for the Interactions.

For the user MSC-2000 and UML 2.0 will appear as close to identical when it comes to specifying timing constraints.

Some users of UML will want to use a more elaborate time model than the simple one included with UML 2.0 proper. There is a profile (i.e. extension) of UML 1.4 called “UML Profile for schedulability, performance and time specification” that comprises a more comprehensive understanding of time. One important point is that the simple time model is close to assuming one global time. There will be an update of that profile to match UML 2.0.

Timers are not included in UML 2.0, but they exist in MSC. In UML 2.0 the user is recommended to use separate lifelines within Interactions to model timers. This does work reasonably well for Interactions, but there is definitely the argument that timer is such a common concept that it deserves to become a concept of the language. This has resulted in that e.g. the UML Testing Profile [14] augments UML 2.0 by timers and time zones.

Generics

MSC-2000 has the possibility to parameterize the charts with data, instances, message types and timers. In UML 2.0 the Interactions are also general Behaviors that may have normal value parameters corresponding to data parameters of MSC-2000.

Message types as well as instance parameters must be dealt with in UML 2.0 through the mechanism of template parameters. Lifelines as parameters are not discussed in UML 2.0.

Timers are not included as separate concepts of the UML language.

Formal Semantics

It has been customary to argue that UML has no (formal) semantics. And from MSC people it has been commonplace to note that MSC does have a formal semantics. Both of these statements are dubious. The practical difference between UML and MSC regarding formality is not as big as the SDL/MSC community likes to pretend. The determining interpretations come in both languages from reading the informal specification and adjusting it to the situation where it is going to be applied. It is still the case that MSCs / Interactions are used more for illustration and discussion than formal requirements specification and verification [15].

Historically, in the mid-1990'ies there was a lot of exploratory academic work on MSC [16-20], while recently academics in general are turning more towards UML for the same opportunistic reasons as does the industry. We will probably see contributions in the UML community that corresponds well with results in the SDL/MSC community in the previous century [10].

Even though the semantics of MSC may be slightly more formally defined through the early work on MSC-96 [21] and the precision of the Z.120 standard, than the UML 2.0, the average user will not notice.

Discussion

The purpose of this paper was to examine whether there is a need for both MSC and UML 2.0 Interactions. Should the ITU retire MSC, or should MSC become a profile of UML 2.0?

The question is clearly not only to be settled on technical grounds. We have seen no tools yet, neither for the full MSC-2000 nor UML 2.0 Interactions. MSC-2000 has been around for about 4 years, why should there suddenly be a renewed interest from the tool vendors? Possibly the reason lies in the challenge from UML 2.0?

Technically the following statements summarize the main issues:

1. MSC-2000 is a language in its own right. This is an advantage and a disadvantage since MSC on its own is seldom enough.
2. UML 2.0 Interactions are dependent upon other parts of UML 2.0. This means in principle that if you choose UML 2.0 Interactions over MSC, you will need to take the whole UML 2.0 to go with it. This would in practice mean also substituting SDL with UML 2.0 in your development process.
3. Assuming that tools existed for both MSC-2000 and UML 2.0, there are few technical reasons for choosing one before the other provided scenarios are your only interest.
4. There may be need for UML profiles that focus on extending Interactions and also making specific choices for the semantic variation points. We have already the forthcoming UML Testing Profile [14]. A specific MSC profile of UML 2.0 could add the innovative data mechanism which possibly could make it easier to handle Interactions formally.

Conclusion

MSC-2000 and UML 2.0 Interactions are very similar, which is exactly what was expected and intended.

Proper tool support and how those tools integrate with SDL and UML 2.0 respectively, will determine which language will survive. There is also the possibility that the languages will co-exist and cross-pollinate as we have seen SDL and UML do over the last years.

From a pure market view it seems probable that the next couple of years will choose a winner. If the UML community can reach real code generation and machine-supported verification, their market position will make them the winners. On the other hand if UML users are unable to reach the level of automatic support that is commonplace in the SDL community, UML will vanish. Whether new products and languages will take their place or SDL/MSD will again be fashionable, remains to be seen.

References

1. ITU, *Z.120*, in *Message Sequence Charts (MSC)*, E. Rudolph, Editor. 1993, ITU-T: Geneva. p. 36.
2. ITU, *Z.120*, in *Message Sequence Charts (MSC)*, E. Rudolph, Editor. 1996, ITU-T: Geneva. p. 78.
3. ITU, *Z.120*, in *Message Sequence Charts (MSC)*, O. Haugen, Editor. 1999, ITU-T: Geneva. p. 126.
4. OMG, *OMG Unified Modeling Language 1.4*. 2000, Object Management Group.
5. OMG, *Unified Modeling Language: Superstructure 2.0*. 2003, OMG.
6. Haugen, Ø. *Converging MSC and UML Sequence Diagrams*. in *Beyond the Standard UML'99 - The Unified Modeling Language*. 1999. Fort Collins, CO, USA.
7. Haugen, Ø. *From MSC-2000 to UML 2.0 - The Future of Sequence Diagrams*. in *10th International SDL Forum*. 2001. Copenhagen: Springer.
8. ITU, *Z.100*, in *Addendum to Recommendation Z.100: CCITT Specification and Description Language*, A. Sarma, Editor. 1996, ITU: Geneva. p. 31.
9. Damm, W. and D. Harel. *LSCs: Breathing Life into Message Sequence Charts*. in *FMOODS'99*. 1999.
10. Haugen, Ø. and K. Stølen. *STAIRS - Steps To Analyze Interactions with Refinement Semantics*. in *UML 2003*. 2003. San Francisco: Springer-Verlag.
11. Selic, B., G. Gullekson, and P.T. Ward, *Real-Time Object-Oriented Modeling*. 1994: Wiley.
12. Haugen, Ø., B. Møller-Pedersen, and T. Weigert, *Structural Modeling with UML 2.0*, in *UML for Real*, L. Lavagno, G. Martin, and B. Selic, Editors. 2003, Kluwer Academic Publishers: Boston. p. 369.
13. ITU, *Z.121*, in *SDL data binding to MSC*. 2003, ITU-T: Geneva. p. 9.
14. OMG, *UML Testing Profile*, I. Schieferdecker, Editor. 2003, OMG.
15. Haugen, Ø. *Using MSC-92 Effectively*. in *SDL'95 with MSC in CASE. Proceedings of the Seventh SDL Forum*. 1995. Oslo, Norway: North-Holland, Elsevier.
16. Baeten, J.C.M. and S. Mauw. *Delayed choice: an operator for joining Message Sequence Charts*. in *FORTE 94*. 1994. Bern, Switzerland.
17. Mauw, S. and E.A. van der Meulen, *Generating tools for Message Sequence Charts*, E.U.o.T. Philips, Editor. 1994, ITU-TS: Geneva. p. 34.
18. Mauw, S. and M.A. Reniers, *An algebraic semantics of Basic Message Sequence Charts*. *The Computer Journal*, 1994. **37(4)**.
19. Mauw, S., *The formalization of Message Sequence Charts*. CN&ISDN, 1996(June 1996): p. 1643-1659.
20. Mauw, S. and M.A. Reniers, *Operational Semantics for MSC'96*, in *Tutorials of the Eighth SDL Forum SDL'97: Time for Testing - SDL, MSC and Trends*, A. Cavalli and D. Vincent, Editors. 1997, Institut national des télécommunications: Evry, France. p. 135-152.
21. ITU, *Z.120 Annex B*, in *Formal Semantics of Message Sequence Charts*, S. Mauw, et al., Editors. 1998, ITU-T: Geneva. p. 76.