Detecting Implied Scenarios Analyzing Non-local Branching Choices

Henry Muccini

Universitá degli Studi dell'Aquila
Dipartimento di Informatica
Via Vetoio, 1 – L'Aquila*
muccini@di.univaq.it

Abstract. Scenarios are powerful tools to model and analyze software systems. However, since they do not provide a complete description of the system, but just some possible execution paths, they are usually integrated with state machines. State machines may be extracted from scenarios using a synthesis process. We could expect that the synthesized state machine model is "equivalent" to the original scenario specification. Instead, it has been proven that it does not always hold, and state machines may introduce unexpected behaviors, called *implied scenarios*. This paper proves that there is a strict correlation between implied scenarios and non-local branching choices. Based on this result, we propose an approach to identify implied scenarios in High-Level Message Sequence Chart specifications and its application to some specifications. We finally highlight advantages with respect to existent approaches.

1 Introduction

Scenarios describe a temporal ordered sequence of events and are used by researchers and industry practitioners for a variety of purposes and at different phases in the software development process (e.g., [11]).

Although very expressive, this approach has two drawbacks with respect to analysis and validation: *model incompleteness* [19] and *view consistency* [14,16]. In order to mitigate both problems, state machines, synthesized from scenarios, are used to complement scenario specifications. When a synthesis process is applied to a set of scenarios, one expects that the synthesized state machines correctly reflect the scenario specification. What may happen, instead, is that the state-based model synthesized from the system scenarios presents sets of behaviors that do not appear in the scenarios themselves. This result has been initially presented in [1,21] proving that the synthesized state machines may introduce unexpected behaviors called "implied scenarios".

Implied scenarios are not always to be considered wrong behaviors. Sometimes they may be just considered as unexpected situations due to specification incompleteness. Anyway, they represent an underspecification in the Message

* Currently on leave at the University of California, Irvine

M. Pezzè (Ed.): FASE 2003, LNCS 2621, pp. 372–386, 2003.

Sequence Chart (MSC) specification and they have to be detected and fixed before the synthesized state machine is used for further steps.

One of the contributions of this paper is the investigation of the relationship between scenarios and their synthesized state machine, by discovering that there is a close dependence between implied scenarios and non-local branching choice[1] [2]. Based on the result that implied scenarios depend on non-local choices, we propose an approach and an algorithm able to identify implied scenarios in specifications composed by both MSC and High-Level MSC (hMSC) notations. The main advantages are that we apply a *structural, syntactic*, analysis of the specifications as opposed to the *behavioral, model-based* analysis in [21], reducing time and space complexity. Moreover, since we do not need to create the synthesized model, we save computational time and prevent the possibility of state explosion problems. A current limitation of our approach is that it can be applied only to specifications composed by both MSCs and hMSCs while the approach presented in [1] can detect implied scenarios just starting from a set of MSCs. The approach is finally applied to some specifications and compared to others.

The paper is organized as follows. Section 2 presents an overview on MSC and hMSC. Section 3 describes what an implied scenario is and why it is introduced during the synthesis process. Section 4 proposes the approach, how it may be implemented and its application to the Boiler case study. Section 5 points out related work while Sect. 6 presents preliminary results. Section 7 concludes the paper by presenting future work.

2 Message Sequence Charts and High-Level Message Sequence Charts

Message Sequence Chart (MSC) is a graphical (and textual) specification language, initially introduced in the telecommunication industry to model the exchange of messages between distributed software or hardware components. In 1992 it has become ITU standard. The latest version is known as MSC-2000 [12].

A MSC (also called basic MSC, bMSC) describes, in a intuitive way, how a set of parallel processes[2] interact, displaying the order in which messages are exchanged. Simple MSCs are shown in Fig. 1: each rectangular box represents a process, the vertical axes represent the process life line, each arrow defines a communication between the left and the right processes (i.e., send and receive of a message), the arrow's label identify the exchanged message. Messages are communicated asynchronously. Many other features are included in the MSC-2000 specification and are documented in [12,17].

The MSC-2000 semantics can be sketched with the help of Fig. 1:
– the send of a message must occur before its reception. For example, message m1 (in MSC1, Fig. 1a) may be received from process P2, only if m1 has been previously sent by process P1;

[1] as we will see later, a non-local choice is a consequence of an under specification in MSC specifications and can give rise to unintentional behavior.

[2] or "instances", following the terminology used in [12].

– within each process, events are totally ordered according to their position in the component life line. Two messages, m1 and m2, are causally dependent if there is a process P so that m1 is sent to or received from P and m2 is sent to or received from P. For example, the send of message m4 in MSC2 (Fig. 1b) is causally dependent on the reception of message m3 (since P2 is related to both messages); message m4 in MSC3 (Fig. 1c) can be sent and received independently from messages m5 and m6;
– in between the send and the receive of the same message, other messages may be sent/received. For example, m2 in MSC1, Fig. 1a, may be sent before the reception of message m1, since they are non causally dependent.

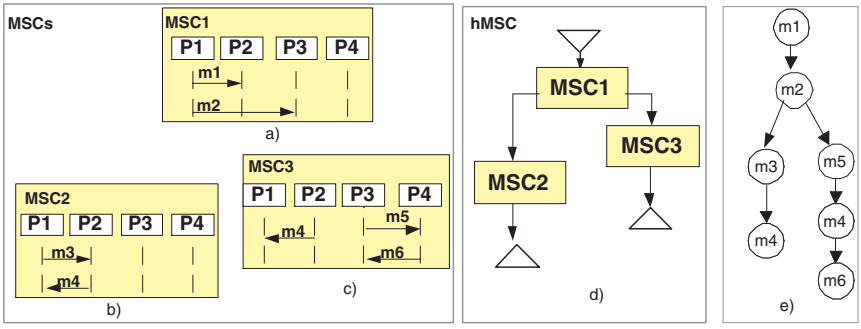

Fig. 1. a) MSCs, d) the hMSC diagram, c) the hMSC graph

To express more complex behaviors and support modularization of MSCs, **High-Level MSC (hMSC)** [12] may be used: they provide operators to link MSCs together in order to describe parallel, sequential, iterating, and non-deterministic executions of basic scenarios. In addition, hMSCs can describe a system in a hierarchical fashion by combining hMSCs within an hMSC.

In Fig. 1d the hMSC notation is shown. The upside down triangle indicates the start point, the other triangles indicate the end points and the rectangles identify MSCs. Figure 1d combines together MSCs in Figs. 1a, 1b, and 1c.

The ITU semantics for unsynchronised hMSCs identifies how MSCs can be combined. Given two MSCs, "MSCa" and "MSCb", and a hMSC in which MSCa precedes MSCb (MSCa -> MSCb):

1. MSCa -> MSCb does not mean that all events[3] in MSCa must precede events in MSCb;
2. an event "b" in MSCb can be performed before another event "a" in MSCa as soon as the processes involved in "b" are not anymore reacting in MSCa;
3. the second event in a generic MSC may be performed before the first one if the processes involved in the first communication are not involved in the

[3] the terms "event" and "action" will be used to identify messages exchanged in the MSCs.

second one. For example, m4 in MSC3 (Fig. 1c) may be sent before m5 in the same MSC.

A hMSC$_G$ graph will be used in the following to represent how actions are related together in the hMSC diagram (Fig. 1e): a node in the graph represents one communication inside the MSCs and it is labeled with the exchanged message name. An edge between node n1 and n2 is drawn when message n1 is exchanged before n2 in the same MSC or n1 is the last node in MSC1, n2 is the first node in MSC2 and MSC2 is reached by MSC1 in the hMSC. Figure 1e draws the hMSC$_G$ for the specification in Fig. 1d. In order to simplify this graph, we assume to deal with synchronous communication. Anyway, the graph can be used to represent bounded asynchronous communication (as in [1]) adding components to act as buffers.

A formal description of sequence chart semantics is out the scope of this paper and interested readers may refer to [12]. With the term *MSC specification* we will refer in the following to the hMSC specification together with its MSCs.

3 Implied Scenarios: The "What" and the "Why"

To explain *why* implied scenarios are introduced, we reuse the example in Fig. 1. The LTS synthesized from the MSC specification is shown in Fig. 2: each dashed box corresponds to one of the processes, the initial state is represented by a solid circle while the final one by a double circle.

As we can see in Fig. 2, m1.m2.m5.m3 is a feasible behavior in the synthesized state machines: P1 sends m1 to P2, then it sends m2 to P3. After the reception of m2, P3 can send m5 to P4. P1 can finally send m3 to P2.

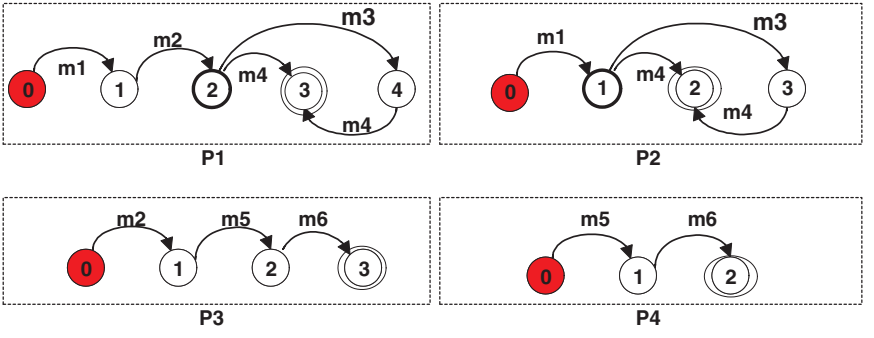

Fig. 2. The synthesized state machine for Fig. 1 specification

If we analyze how the MSC specification in Fig. 1 implements this behavior, we can notice that m1 and m2 are exchanged inside MSC1, then the system proceed to MSC3 with m5. To communicate via m3 (as expected by the

synthesized model), the flow should proceed to MSC2, but *this action is not allowed by the bMSC semantics* previously outlined. In fact, m4 in MSC3 should precede m3 in MSC2 since both of them rely on process P2 (hMSC semantics, second rule, Sect. 2). What we found is a flow of execution, allowed in the synthesized state machine but not specified in the MSC specification, that is, an implied scenario.

Now we will understand the *why* of implied scenarios: analyzing closely the synthesized state machines and the selected path, we can notice that after m2 is sent, (i.e., P1 is in state 2, P2 is in state 1 in Fig. 2) both P1 and P2 become ready to communicate via m3 and m4. After m5 has been exchanged between P3 and P4, P1 and P2 are *still* able to react using m3 while it is not permitted by the hMSC specification. If we take a look back to the MSC specification (Fig. 1) we can notice that this situation may arise when a branch is presented in the hMSC_G graph (node m2 in Fig. 1.e). In a branch, the system can evolve in different directions. Let b1, b2 and b3 be three possible branches. What may happen, in a branch, is that each process freely decides to take one of the possible branches and gets ready to react in that branch. When the system decides to evolve in one direction (e.g., branch b2), all the processes not involved in the first action in b2, do not react. Thus, they are not aware of the system's choice and they are still ready to react as nothing happened. As a result, when the system evolves in b2, those processes are still ready to act as in b1 or b3. This concept is usually known as "non-local branching choice" [2] and arises when there is a branch in the MSC specification, and the first events, in the different branches, are sent by different processes.

In our example, there is a branch in node m2 that generates a non-local choice. In fact, the first message in node m3 is sent by P1 while the first message in node m5 is sent by P3. If the system evolves in MSC3, process P3 sends the m5 message to P4. At this point, P1 and P2, that were not reacting, are still able to communicate via m3 and this communication creates the implied.

To have an implied scenarios these conditions hold: i) there is a non-local branching choice in the MSC specification so that ii) two processes maintain "extra" information[4] that can make them communicate in an unexpected way. We can summarize this section as follows:

– *What is an implied scenario?* An implied scenarios is a behavioral path that can be extracted from the state machine model but does not exist in the MSC specification.

– *Why do implied scenarios exist?* An implied scenario is due to a non-local choice situation in which two processes are enabled to communicate thanks to an extra information they catch. Notice that a non-local choice is not enough to have an implied scenario.

[4] this concept will be better explained later.

4 The Approach and the Algorithm

This section is utilized to describe the approach and the algorithm we propose to detect implied scenarios. Section 4.1 informally presents the overall idea. Section 4.2 refines the overall idea describing how the approach may be implemented by an algorithm which partially reuses existent tools. Section 4.3 shortly presents the *Boiler*[5] (v3) specification described in Fig. 3 and applies the approach to this example. Section 4.4 analyzes the algorithm completeness and correctness.

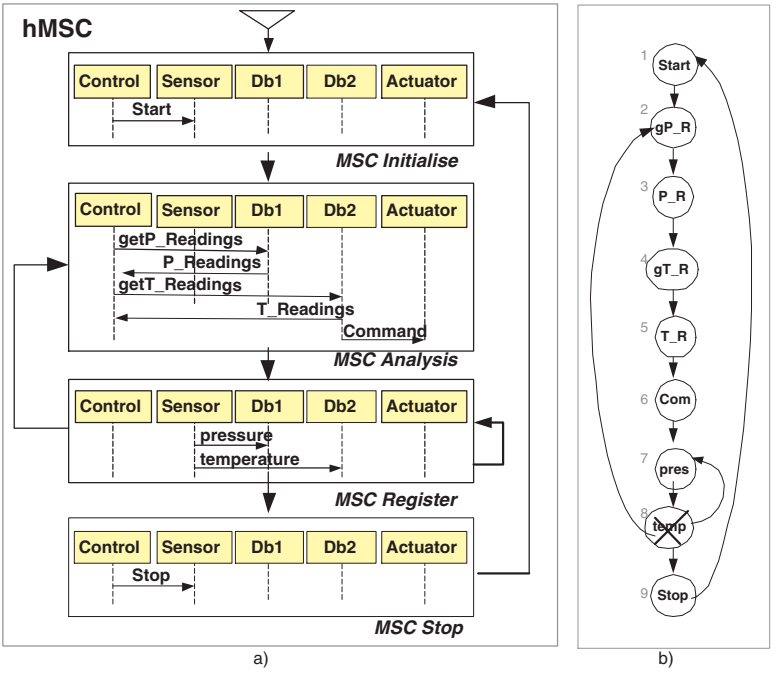

Fig. 3. a) The Boiler (version 3) Example, b) its hMSC_G graph

4.1 High Level Description

Briefly, an implied scenario may be found in the MSC specification when a non-local choice occurs that lets processes keep extra information that is lately used for a communication. Step 1 is used to detect the presence/absence of the non-local choice(s) in the hMSC_G graph. When a non-local choice arises, each process P gets ready to execute all possible branches and its state corresponds to the state P has in all the nodes directly reachable from the non-local choice. Step

[5] This specification comes from [22].

2 detects the state of each process after a non-local choice. Since P maintains this state in all the following hMSC$_G$ nodes where P does not react, we say that P1 keeps an "augmented" behavior that comes from the non-local choice. Step 3 identifies, for each process P and for each node n in the hMSC$_G$, the augmented of P in n. When two processes P1 and P2 share the same augmented behavior e in the same hMSC$_G$ node n, then their interaction generates the implied scenario. Step 4 detects the implied scenario.

4.2 The Algorithm

In this section we describe, using a more precise terminology, how the approach can be implemented through different steps.

Step 1: Non-local Choice Detection
The algorithm proposed in [2] and implemented inside the MESA framework [3] may be applied to detect non-local choices in MSC specifications. Informally, the tool examines the MSCs involved in a choice and verify that they all have the same unique process which sends the first event. If it does not apply, a non-local choice is identified and the hMSC nodes involved in the non-local branch are identified.

Technically, given the MSC specification, a direct graph (called Message Flow Graph (MFG)) is built and analyzed using the algorithm proposed by Ben-Abdallah and Leue [2]. The algorithm visits the graph an identifies the non-local choice nodes. Since the algorithm assumes that each process in each MSC exchange at least one message with other processes (assumption that does not apply in general), it needs to be slightly extended, as already pointed out in [2].

In the context of the hMSC$_G$ graph, we can define a non-local choice node as follows:

Def: *N-L Node*
A node $n0$ in the hMSC$_G$ is a non-local choice node (*N-L* node) if it is a branching node and the messages in DR(n) are sent by different processes.

where:

Def: *Direct Reachability and DR(n) Function*
We will say that nodes $n1$,..., nk are directly reachable from node n if they may be reached from n in through one edge on the hMSC$_G$.
The function DR: $n \longrightarrow$ N will return the nodes N directly reachable from n.

Step 2: State(P,N-L Node) Detection
If a non-local choice is detected, the algorithm needs to identify the state assumed by each process P in the *N-L* node. The following definitions will be useful for next discussions:

Def: *State of P in n and State(P,n) Function*
Let n be a node in the hMSC$_G$ graph and P a process. The state of P in n identifies the messages P can send or receive in n.
The function State: P x $n \longrightarrow s$ will return the state s of P in n.

Using the notation above, we need to compute State(P,N-L node) for each P. In order to do that, we introduce the concept of *maximal continuation*:

Def: *Continuation and C(P,n) Function*
Let $n1$ and $n2$ be nodes in the hMSC$_G$ graph and P a process. $n2$ is a *continuation* of $n1$ for P if i) $n2$ may be reached from $n1$ in through one edge on the MSC$_G$ graph or if ii) there is a $n3$ such that $n3$ is a continuation of $n1$, $n2$ is a continuation of $n3$ and $n3$ does not contain events for process P.
The function C: P x $n \longrightarrow$ N identifies the continuation for process P in node n and returns a set of nodes N.

Informally, $n2$ is a continuation of $n1$, if $n2$ is directly reachable from $n1$ or it is reachable from $n1$ traversing nodes where P does not react.

Def: *Maximal Continuation and MC(P,n) Function*
Node $n2$ is a *maximal continuation* of $n1$ if i) $n2$ is a continuation of $n1$ and ii) for all $n3$ continuations of $n1$, $n2$ is a continuation of $n3$.
The function MC: P x $n \longrightarrow$ N identifies the maximal continuation for process P in node n and returns a set of nodes N.

Informally, $n2$ is a maximal continuation of $n1$ if $n2$ is the first node reachable from $n1$ so that P reacts. (Notice that the concept of continuation and maximal continuation just provided is more fine-grained of that provided in [21] since we deal with hMSC$_G$ nodes instead of hMSC nodes).

Using these definitions we can say that State(P,N-L node) = State(P,MC(P,N-L node)), assuming that State(P,N) = $\cup$ {State(P,n1), State(P,n2),..., State(P,ni)} where n1..ni $\in$ N. Informally, the state a process P has in the N-L node corresponds to the state P has in all the states reachable form N-L so that P can react.

As we said, this state is kept from P until its next reaction in the hMSC$_G$. Technically, State(P,ns) = State(P,N-L node), for each node ns so that $ns \in$ DR(N-L node) and P is not reacting in ns.

Moreover, to extend what we said in Sect. 4.1, if P is not reacting in the N-L node and N-$L \in$ C(P,np), then State(P,np) = State(P, N-L node). This happens because when a process does not react in a hMSC$_G$ node, he becomes ready to react as in the first next state it can react.

We can use the following algorithm to identify the state a generic process P has in the hMSG$_G$ nodes:

Algorithm to Identify State(P,n):

```
Build the hMSC_G graph for the MSC specification under analysis;
\\P is a generic process;
n = N-L node;
NS = {n1, n2, ..., nk};  \\ a set of nodes
FOR each n
    FOR each process P,
       compute MC(P,n) \\ in-breath visit of the graph
       State(P,n) = State(P,MC(P,n)) \\ P assumes the state of its
                                      \\maximal continuation
       NS = DR(n);
       \\ with this function we compute the state P has in the DR(n)
       FOR each P, compute NextState(P,NS);
       \\ with the following statements, we compute the state of P
        \\ in in the predecessors of the N-L node
       IF P does not react in n
         THEN
            FOR each node np so that n = MC(P,np),
                     State(P,np) = State(P,n).


NexState(P,NS)
    {FOR each node ns in NS
         IF State(P,ns) = empty
            THEN State(P,ns) = State(P,n);
                 NexState(P,DR(ns))}
```

Step 3: AB(P,n) Detection

Step 2 identified State(P,n), given a process P and a node n. As we said, it may happen that process P, after a N-L node and due to the non-local choice, can show some unexpected behaviors, called "augmented behavior". Those augmented behaviors are all those actions P can run in n but are not in its maximal continuation. The following definition may help to define this concept:

Def: *Augmented Behavior and AB(P,n) Function*
The augmented behavior of P in a node n is given by State(P,n) - State(P,MC(n)).
The function AB: P x $n \longrightarrow s$ identifies the augmented behavior for process P in node n and returns the augmented state s.

The following algorithm detects, for each process in each node n, the augmented behavior of P in that node.

Algorithm to Identify AB(P,n):

```
FOR each P in the MSC specification
    FOR each node n
        AB(P,n) = State(P,n) - State (P,MC(P,n)).

This information can be stored inside the hMSC_G graph.
```

Step 4: Implied Scenario identification
For each process P in each node n, we know what augmented behavior P has in n. If two processes P1 and P2 are augmented so that event e in AB(P1,n) is in AB(P2,n) too, then a node source of implied scenario is detected. A path that starts from the initial hMSC$_G$ graph node, traverses the N-L node and reaches n is implied if n is in C(Pi,N-L) for i= 1 or 2. The last condition requires that after the $N - L$ node at least one of the two processes do not react before the node source of the implied scenario.

The algorithm follows:

Algorithm to Identify Implied Scenarios Source:

```
n0 = first node in the hMSC_G graph;
p = a hMSC_G graph;
FOR each node n in the hMSC_G graph
    FOR each couple of processes P1 and P2
        IF exists "e" so that "e" is in AB(P1,n) and "e" is in AB(P2,n)
            THEN an implied scenario node is identified.
FOR each implied scenario source n,
    IF n is in C(P1,N-L) or n is in C(P2,N-L)
        THEN p = n0. ...  .N-L. ... .n is an implied scenario.
```

4.3 Application to the Boiler Example

The example we present and analyze in this section has been borrowed from [22]. The system is composed of four different processes communicating through four different MSCs, as shown in the MSC specification in Fig. 3a.

The system works in this way: after the "Control" process starts the system (in MSC *Initialise*), it acquires information about pressure (getP_Readings) and temperature (getT_Reading) from "Db1" and "Db2", respectively (in MSC *Analysis*). When the data have been received, fresh values are stored to the databases (MSC *Register*) and the system can nondeterministically proceed in three ways: the "Control" can get the new data (going back to MSC *Analysis*), the "Sensor" can refresh the data values in the databases (staying on MCS *Register*) or the "Control" can decide to Stop the system (in MSC *Stop*) and go back to the initial configuration (MSC *Initialise*). Figure 3b represents the hMSC$_G$ graph for this example that will be used to apply our approach.

Applying the Ben-Abdallah and Leue algorithm [2], we discovered that node 8 in Fig. 3b is a N-L node.

Applying the second step in our approach, we detect State(P,N-L node) for each process P. The output of this step is summarized in the next table:

Process	**State(P,N-L)**	**N-L ∪ (ns ∪ np)**
Control	{Stop, getP_Reading}	{8} + {6,7}
Sensor	{pressure, Stop }	{8} + {2,3,4,5,6}
Db1	{pressure, getP_Reading}	{8} + {1,9}
Db2	{temperature, getT_Reading}	{8} + {1,2,3,7,9}
Actuator	Command	{8} + {1,2,3,4,5,7,9}

The first column identifies the processes, the second proposes information on the state of each process in the *N-L* node and the last column shows the *N-L* and ns ∪ np nodes obtained applying Step 2 to the Boiler hMSC$_G$. The first row may be interpreted as: process Control in nodes 6, 7 and 8 can react with Stop and getP_Reading.

At this point, for each node $n \in N\text{-}L \bigcup \{ns\} \bigcup \{np\}$, Step 3 allows to identify the augmented behavior for each process P in n. The following table summarizes the results:

Process P	**node n**	**AB(P,n)**
Control	{6,7}	{Stop, getP_Reading}
Sensor	{2,3,4,5}	{pressure, Stop }
Sensor	{6}	Stop
Db1	{1}	pressure
Db1	{9}	{pressure,getP_Reading}
Db2	{1,2,8,9}	{temperature, getT_Reading}
Db2	{3}	{temperature}
Db2	{7}	{getT_Reading}
Actuator	{1,2,3,4,7,8,9}	Command

The first column represents the processes, the second one reports the nodes already found in the first table and the third column calculates the augmentation for P in node n. The first line can be read as follows: process Control in nodes 6 and 7 has some extra behaviors, namely Stop and getP_Reading.

Applying Step 4, we discover that Control and Sensor can communicate through an augmented action in node 6, i.e., Stop ∈ AB(Control,6) and Stop ∈ AB(Sensor,6). Path 1.2.3.4.5.6.7.8.2.3.4.5.6 in the hMSC$_G$ graph in Fig. 3b is implied since it starts from the first node, traverses node 8 (the $N - L$ node), reaches the implied scenario node 6 that is in C(Sensor,8).

4.4 Algorithm Completeness and Correctness

Completeness and correctness of the proposed approach are still under analysis. The approach we propose has been applied, by now, to many of the MSC

specifications available in [22] and several new specifications have been made in order to test completeness and correctness. In all the analyzed specifications, our algorithm has detected the same (and only the) implied scenarios detectable using the approach in [21] (that has been proved to be complete and correct).

An actual limitation (i.e., incompleteness) of our approach is that it works only with specifications composed by both MSCs and hMSCs. Therefore, the approach presented in [1] is more complete, since it can detect implied scenarios just starting from a set of MSCs. An idea to make our approach working without the hMSC specification is to build an MSC_G graph, identifying how the MSCs can interact, and to analyze this new graph with the proposed approach. This is just an initial idea that needs to be evaluated and extended in future work.

5 Related Work

There are some areas of research that may be related to our work: implied scenario detection, consistency checking and model checking.

Implied Scenarios:
The first research on implied scenarios have been proposed in [1,21]: the first paper proposes a polynomial-time algorithm to determine if a set of scenarios are "realizable" through state machines (i.e., if exists a concurrent automata which implement precisely those scenarios) and to synthesize such a realization using a deadlock-free model. If scenarios are not realizable, implied scenarios are detected. In the proposed approach, scenarios are modeled using Message Sequence Charts (MSCs) [12]. The former paper, based on both MSCs and hMSCs, describes an algorithm which generates some safety properties identifying (a simplification of) the exact behavior of the MSC specification, it models this properties using Labeled Transition Systems (LTSs) and checks the conformance of the synthesized LTS with respect to these properties. The algorithm has been implemented inside the Labeled Transition System Analyzer (LTSA) tool [22].

Consistency Checking:
As outlined in [4] consistency checking among multi-perspective or multi-language specifications is not a young field of research. There are different papers proposing different approaches to handle consistency. Some of these are used for checking model consistency.

In [5] the authors propose an approach for multiple views consistency checking. To check the views consistency, they provide a formal definition of the views and their relationships, they define their semantics as sets of graphs and apply an algorithm to check the consistency of diagrams. Filkenstein and colleagues in [4] describe how inconsistencies may be handled using the *ViewPoints* framework and a logic-based consistency handling.

Our work may be related to both of these papers since what we finally do is to check the conformance between a scenario and a state machine. The main

difference is that while in the other approaches these two models may be developed independently, in our case the state machine is obtained by synthesis from the scenario specification.

Model Checking:
Some model-checking based approaches have been proposed to model check scenarios with respect to state-based diagrams [18,6].

Again, these works have still in common with our the intent (i.e., checking the consistency between two different models) but in our case one model is obtained by synthesis from the other.

6 Comparison and Considerations

Comparison with the Approach in [21]:
In our approach, we propose a "structural" analysis which builds the hMSC_G graph and analyze its structure, in order to reveal implied scenarios. In [21], they build a behavioral model of the system, and analyze it. As a result, the complexity of our approach is not dependent on the system, behavioral, complexity, as opposite to [21].

To test the value of our approach, we modified the Boiler (version 3) specification by introducing two concurrently acting "Control" processes. The safety property, built by the LTSA-Implied tool [22] (and implementing the [21] approach) was modeled using an LTS composed by 748 nodes in the original specification and 11717 in the second one, almost 15 times bigger. The application of the approach we proposed, instead, requires to build a graph composed by 9 nodes in the first specification, 18 nodes in the second one. Moreover, the [21] approach needs to build the synthesized state machine, while we do not.

From this analysis, we cannot certainly conclude that our approach is always more efficient than that proposed in [21]. Anyway, we can argue that for concurrent and complex specifications, our approach can reduce time and memory (usage) complexity with respect to the approach presented in [21].

Another advantage of our approach is that with one execution *all* the implied scenarios in the hMSC specification can be discovered and displayed. On the opposite, the LTSA-Implied tool may discover only one implied scenario at a time providing a partial, incomplete, facet of the problem. With our approach, instead, we provide a software engineer a more valuable tool that detects all the possible sources of MSC underspecification at the same time and allows him to fix all the implied behaviors through only one specification refinement.

Implied Scenarios and Non-local Branching Choices:
If an implied scenario is detected, a non-local branching choice (NLC) holds. The opposite is not true, as already pointed out in Sect. 3; an NLC not always causes an implied scenario. In general, it could be enough to detect the NLC and fix it (without using the algorithm proposed in this paper). However, the application of the proposed algorithm guarantees some important advantages.

Every time a NLC is detected and fixed, new problems can be introduced in the MSC specification. Using our algorithm, we can decide to fix only those NLCs that give rise to an implied scenario. Another interesting point is that applying the described algorithm it becomes easier to *localize and fix* the implied scenarios (when they are not desired). In fact, the algorithm we propose uses the algorithm proposed in [2] to identify the non-local choice node(s), discovers how the processes behavior has been augmented due to the non-local choice and outlines which processes have to communicate to generate the implied scenario.

The Message Flow Graph (MFG) is used in [2] and implemented inside the MESA tool [3] in order to detect NLCs. Since the MFG seems to capture more information than the hMSC_G (used in our approach to detect implied scenarios), a MFG could be used to detect both NLCs and implied scenarios. Further research is necessary to understand how our algorithm needs to be modified to work with the MFGs. Eventually, our algorithm could be integrated in the MESA tool in order to provide a complete structural approach to detect both NLCs and implied scenarios.

7 Future Work

Future work goes along different directions: first of all, we need to analyze the completeness of the approach and how it can be extended to cover specification composed of MSCs only. A tool needs to be realized in order to implement the algorithm. A possible alternative may integrate the approach in existing tools, like MESA [3]. Future research will be conducted to analyze the applicability of this approach to UML [15] diagrams: the semantic differences in between sequence diagrams and MSCs and the absence of mechanisms to combine sequence diagrams together need to be carefully analyzed. An interesting future work could also analyze how these unexpected, implied scenarios can be used for testing purposes. In the meantime, we are looking for a real system specification to better evaluate the qualities/weaknesses of the proposed approach.

Acknowledgments. The author would like to acknowledge the University of California Irvine and the Italian M.I.U.R. National Project SAHARA that supported this work, Paola Inverardi for her comments on a draft of this paper and the anonymous reviewers for their interesting suggestions.

References

1. R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. In Proc. *22nd Int. Conf. on Software Engineering*, ICSE2000, Limerick, Ireland.
2. H. Ben-Abdallah and S. Leue. Syntactic Detection of Process Divergence and non-Local Choice in Message Sequence Charts. In Proc. *TACAS'97*, LNCS 1217, pp. 259–274, 1997.
3. H. Ben-Abdallah and S. Leue. MESA: Support for Scenario-Based Design of Concurrent Systems. In Proc. *TACAS'98*, LNCS 1384, pp. 118–135, 1998.

4. A. Filkenstein, D. Gabbay, A. Hunter, J. Kramer, and B. Nuseibeh. Inconsistency handling in multi-perspective specifications. *IEEE Trans. on Software Engineering*, Vol. 20, Number 8, pp. 569–578, August 1994.
5. P. Fradet, D. Le Metayer, and M. Perin. Consistency Checking for Multiple View Software Architectures. In Proc. European Software Engineering Conference (*ESEC/FSE'99*), pp. 410–428, 1999.
6. P. Inverardi, H. Muccini, and P. Pelliccione. Automated Check of Architectural Models Consistency using SPIN. In *ACM Proc. Int. Conference on Automated Software Engineering* (ASE 2001), 2001.
7. K. Koskimies and E. Makinen. Automatic synthesis of state machines from trace diagrams. *Software Practice and Experience*, 24(7), pp. 643–658, 1994.
8. K. Koskimies, T. Männistö, T. Systä, and J. Tuomi. Automated support for OO software. *IEEE Software*, 15(1), pp. 87–94, 1998. Tool Download at: `http://www.cs.tut.fi/~tsysta/sced/`.
9. I. Krüger, R. Grosu, P. Scholz, and M. Broy. From MSCs to Statecharts. *Distributed and Parallel Embedded Systems*, Kluwer Academic, 1999.
10. S. Leue, L. Mehrmann, and M.Rezai. Synthesizing ROOM models from message sequence chart specifications. In *Proc. of the 13th IEEE Conf. on Automated Software Engineering*, 1998.
11. S. Mauw, M.A. Reniers, and T.A.C. Willemse. Message Sequence Charts in the Software Engineering Process. In S.K. Chang, editor, *Handbook of Software Engineering and Knowledge Engineering*, World Scientific Publishing Co., Vol. 1, *Fundamentals*, pp. 437–463, December 2001.
12. Message Sequence Chart (MSC). ITU Telecommunication Standardization Sector (ITU-T). Z.120 Reccomendation for MSC-2000, year 2000.
13. H. Muccini. An Approach for Detecting Implied Scenarios. In Proc. *ICSE2002 Workshop on "Scenarios and state machines: models, algorithms, and tools"*. Available at `http://www.henrymuccini.com/publications.htm`.
14. C. Pons, R. Giandini, and G. Baum. Dependency Relations Between Models in the Unified Process. In Proc. *IWSSD 2000*, November 2000.
15. Rational Corporation. Uml Resource Center. UML documentation, version 1.3. Available from: `http://www.rational.com/uml/index.jtmpl`.
16. G. Reggio, M. Cerioli, and E. Astesiano. Towards a Rigorous Semantics of UML Supporting its Multiview Approach. In Proc. *FASE 2001*, LNCS n. 2029, Berlin, Springer Verlag, 2001.
17. E. Rudolph, P. Graubmann, and J. Grabowski. Tutorial on Message Sequence Charts. In *Computer Network and ISDN Systems*, 28(12), pp. 1629–1641, 1996. Special issue on SDL and MSC.
18. T. Schafer, A. Knapp, and S. Merz. Model Checking UML State Machines and Collaborations. On the *Workshop on Software Model Checking*, Paris, July 23 2001. ENTCS, volume 55 number 3.
19. Second International Workshop on Living with Inconsistency. ICSE'01 workshop, May 13, 2001 Toronto, Canada.
20. UBET. `http://cm.bell-labs.com/cm/cs/what/ubet/`.
21. S. Uchitel, J. Kramer, and J. Magee. Detecting Implied Scenarios in Message Sequence Chart Specifications. In Proc. European Software Engineering Conference (*ESEC/FSE'01*), Vienna 2001.
22. S. Uchitel, J. Magee, and J. Kramer. LTSA and implied Scenarios. On line at: `http://www.doc.ic.ac.uk/~su2/Synthesis/`.
23. J. Whittle and J. Schumann. Generating Statechart Designs from Scenarios. In *Proc. 22nd Int. Conference on Software Engineering* (ICSE'00), 2000.