Compatibility Analysis of System Behavior Described by Message Sequence Charts

Wolfram Hardt, Mirko Caspar* and Matthias Glockner**

Abstract

Dynamic behavior of automotive electronic control units (ECUs) can be specified by different models as ASCET, Matlab Simulink or Message Sequence Charts (MSCs). MSCs are often used by automotive OEMs (Original Equipment Manufacturer) for specifying dynamic behavior of ECUs as black box models. In this paper we present a method which allows a tool based compatibility check of MSCs independently from the structure. Compatibility means, that parts of the behavior of two MSCs are equivalent. Therefore, the MSCs are transformed to nondeterministic automata with ε -transition which then be transformed to deterministic automata. These are the inputs for a heuristical backward compatibility checking algorithm. Furthermore, we present some representative results of runtime evaluation for our approach, which is used for backward compatibility checking of ECUs by now.

Keyword: Compatibility Analysis, System behavior, Massage Sequence Chats

1. Introduction

Due to the complexity of the MSC-Syntax [9], MSCs can represent equivalent dynamic behavior by different structural designs. Figure 1 presents two MSCs (MSC_1, MSC_2) with the identical instance p_1 and the identical events $E=y_1,y_2,y_3,x_1,x_2,x_3$. MSC_1 has one Alternative-Operator (alt) with three sections. MSC_2 has one Optional-Operator (opt) which contains an Alternative-Operator with two sections.

MSC_1 and MSC_2 differ only in the structural design but describe a similar behavior. So, the comparison of the structural designs of the MSCs will not result in a correct prediction for the question, if the specified behavior of the MSC is compatible. An alternative for analyzing the compatibility is the comparison of the event-sequences. An event-sequence is one specific order of events. MSC_1 and MSC_2 describe three alternative event-sequences (Figure 2). MSC_1 has an Alternative-Operator with three Sections and so three alternative event-sequences are possible. MSC_2 has an Optional-Operator with an internal alternative-Operator. The 1st and 2nd possible event-sequence occur, when the "Optional-Operator=True" and so the two alternative sections of the Alternative-Operator can be passed. The 3rd event-sequence can occur, when the "Optional-Operator=False" and is not passed through. A comparison of the event-sequences of MSC_1 and MSC_2 shows now that for each possible sequence of MSC_1 an identical sequence of MSC_2 and vice versa can be found. So we can conclude, that MSC_1 and MSC_2 describe the same behavior. They are called compatible.

This paper describes a method for comparing the specified behavior of MSCs which is used regarding backward compatibility. In section 2 we discuss related work shortly and define the (backward) compatibility of MSCs in section 3. Our concept and the detailed process for comparison of MSCs are described in section 4 and 5. Details about the implementation and the performance of our approach can be found in section 6.

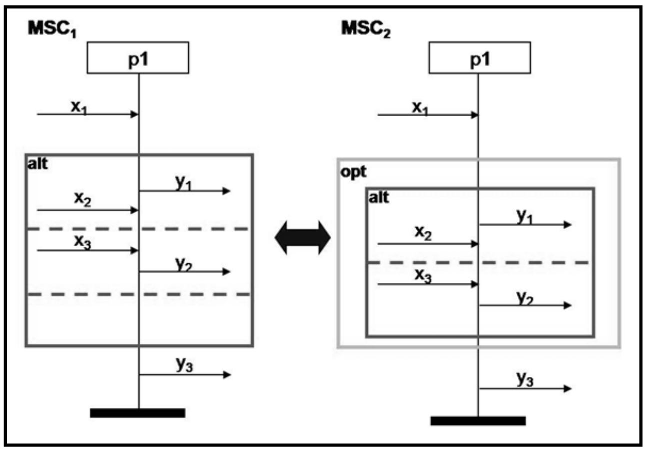

Figure 1 *: MSC1 and MSC2 have a different structural*

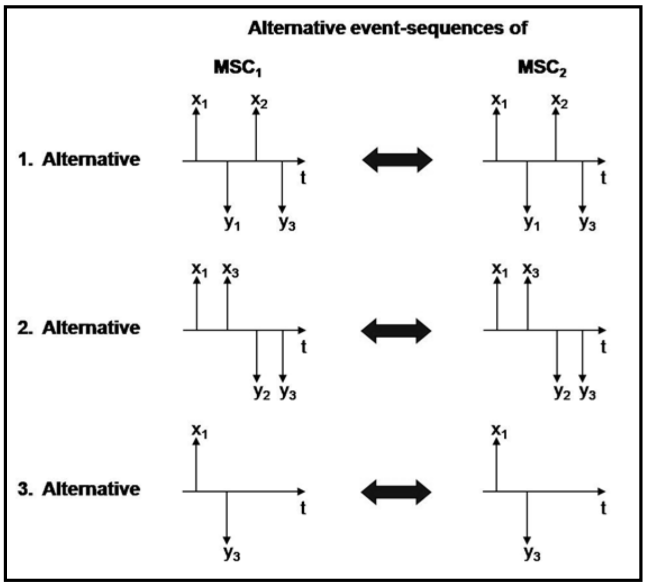

Figure 2 *: Possible event-sequences of the MSC1 and the MSC2 of Figure 1*

*Chemnitz University of Technology, Faculty of Computer Science, 09111 Chemnitz, Germany

** BMW AG, Forschungs– und Innovationszentrum,80788 Munchen, Germany

2. Related Work

MSCs are used for specifying dynamic behavior, interaction sequences and test cases [6][13]. The specified behavior itself is often analyzed based on finite automata [7][10][11]. The method described in this paper is an expansion of [10]. While [10] generates a full behavioral description based on multiple MSC scenarios, this paper focuses on an abstract comparison of MSCs. Nevertheless, synergies could be used and expanded. For instance, specific transformation rules for nested MSC inline-expressions were introduced

3. Definition of compatibility for MSCs

To define the compatibility of MSCs it's necessary to define a MSC first.

Definition 1 [5]**:** A Message Sequence Chart (MSC) is a tuple (E, P, L, λ, <, tgt) where:

- E: set of events, $E = S \cup R$ where S is a set of send events and R is a set or receive events
- P: finite set of instances
- L: finite set of labels
- λ: $E \rightarrow L \times I$ maps events to instances and labels, $i(E) \subseteq E$: $(\forall e \in i(E) \Rightarrow \lambda(e) = (l,i) \wedge$
  $\forall e \in E \setminus i(E), \lambda(e)=(l,j): j \neq i)$, $i, j \in I$
- < is a set of transitive total orders relation $<_i$
  $<_i \subseteq (i(E) \times i(E))$, $i \in I$
- tgt: $S \rightarrow R$, function that maps send events to receive events

A sequence of message labels $(l_1, l_2, \ldots, l_n)$ (event sequence) describes the behavior of an instance i $\Leftrightarrow$

- label is assigned to an event in i
  $\forall j \in [1,n]: \exists e \in i(E): \lambda(e)=(l_j,i)$
- total order in i is hold
  $\forall j \in [1,n-1]: (l_j, l_{j+1}) \in <_i$

The behavior of the MSC is a set Z of label sequences, which define all possible behaviors of all instances.

Intuitively, two MSCs are compatible, when they have the same behavior. At first we define the backward compatibility of two MSCs:

Definition 2: $MSC_2=(E_2, P_2, L_2, \lambda_2, <_2, tgt_2)$ is backward compatible to $MSC_1=(E_1, P_1, L_1, \lambda_1, <_1, tgt_1)$ (we write $MSC_1 \Leftarrow_c MSC_2$) $\Leftrightarrow$

$\forall z_i = (l_1, l_2,\ldots, l_n) \in Z_1 \Rightarrow \exists z_j = (k_1, \ldots,k_m) \in Z_2: n = m$
$\wedge\ \forall i \in [1, n]: l_i = nc(k_i)$

The name-conversion function nc: $L \rightarrow L$ maps the labels of one MSC to the labels of another and helps to solve problems with different namespaces for events in different MSC. The relation between the events E and the labels L are given by λ.

In other words, MSC_2 is backward compatible with MSC_1, if every event-sequence of MSC_1 is also part of MSC_2 (see Figure 2).

Definition 3: MSC_1 and MSC_2 are compatible $\Leftrightarrow$ $(MSC_1 \Leftarrow_c MSC_2) \wedge (MSC_2 \Leftarrow_c MSC_1)$

4. Concept

In order to get a prediction about backward compatibility of ECUs, all MSCs of ECU_1 and ECU_2 must be compared. For explaining the method, this paper limits the comparison on just two MSC scenarios. The comparison of further MSC scenarios is equivalent. For more examples and details see [4].

In order to compare MSCs, the MSCs must be transferred to a normalized form. This paper suggests a transformation into automata. The transformation of MSCs in automata is reasonable, because a single MSC instance specify analog to an automaton also an input/output-behavior [9]. Therefore the instances of MSCs have to be decomposed and transferred separately into automata.

In Figure 3 two MSCs are shown, which are to be compared. After the decomposition of the MSC every instance is transformed into a finite automaton. The automaton can be reduced by methods of the automaton theory (ε-elimination, determination, minimization) [8]. Through a comparison of the generated automata of MSC_1 and MSC_2 a statement related the backward compatibility of MSC_2 to MSC_1 can be generated

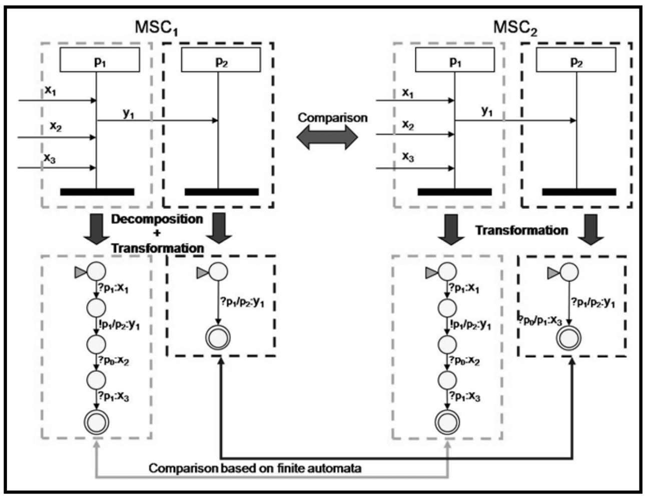

Figure 3 *: Comparison of MSCs bases on automata*

5. MSC comparative method

The suggested method is composed into the following steps (see also Figure 4):

1. Structural specification: Message Sequence Charts are designed with an MSC-Editor and exported to an ASCII-File.
2. Transformation into XML-description: Based on a MSC-Metamodel the ASCII-Files are converted to MSC-XML-Instances. This steps aims to have a defined representation of the MSC, since the syntax of the ASCII files depends on the used editor.
3. Transformation into automata: MSC-XML-Instances are transformed into finite automata (FA).
4. Determination of automata: Finite automata (FA) are converted through ε-elimination and determination into deterministic finite automata (DFA).
5. Compatibility analysis based on automata: Comparison of the deterministic finite automata (DFA(MSC_1), DFA (MSC_2)) and analysis regarding backward compatibility.

In this paper the steps 3-5 are described in detail. Explanations of the other steps can be found in [4].

5.1 Transformation into automata

The MSC-XML-Instances out of step 2 are the basis for all further calculations. The transformation of MSCs into automata (step 3) can be structured as follows:

1. Decomposition of MSCs into instances p_i

2. Transformation of the decomposed instances p_i into finite automata. Throughout the transformation three different automata-types can occur:

- nondeterministic finite automaton (NFA)
- nondeterministic finite automaton with ε-transitions (NFA - ε)
- deterministic finite automaton (DFA)

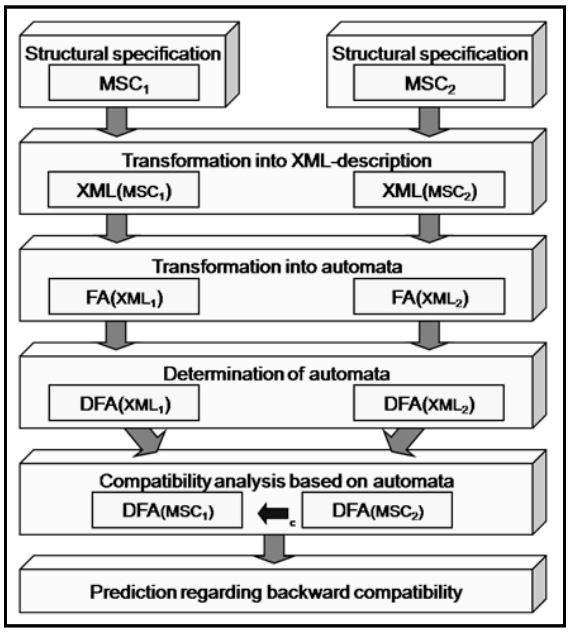
Structural specification
MSC1
Structural specification
MSC2
Transformation into XML-description
XML(MSC1)
XML(MSC2)
Transformation into automata
FA(XML1)
FA(XML2)
Determination of automata
DFA(XML1)
DFA(XML2)
Compatibility analysis based on automata
DFA(MSC1)
c
DFA(MSC2)
Prediction regarding backward compatibility

Figure 4 *: Steps of the MSC comparative method*

5.1.1 Decomposition of MSCs

Figure 5 presents a MSC with two instances. Each instance is transformed separately into an automaton. Therefore the MSCs have to be decomposed by the instances. In order to make sure that the information, which events are exchanged between which instances, does not get lost, the events get prefixes in the style (!|?<source>/<destination>:). These prefixes are separated by colon and are set in front of the event-name:

- The symbols ! (sending) and ? (receiving) describe the direction of communication analog to [10]
- The allocation, which instance sends an event and which instance is receiving an event, is marked by the notation <source>/<destination>

In Figure 5 an event y_1 is sent from instance p_1 to instance p_2 . During the decomposition, this event gets the prefix $!p_1/p_2:y_1$

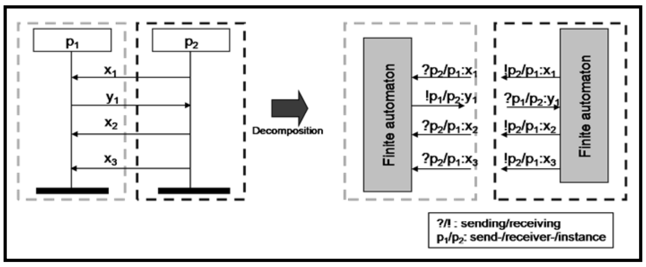

Figure 5 *: Decomposition and use of the defined notation*

5.1.2 Transformation into finite automata

After the decomposition every instance pi will be transferred separately into a finite automaton. Therefore the instances are traversed top down and the MSC-elements are transformed into related finite automata based on specific transformation rules.

Finite automata $A = (Q, \Sigma, \delta, q_0 , F)$ consist of states $Q=\{q_i , ...\}$, transitions δ and an input alphabet Σ. Furthermore the initial-state q_0 and the finite states $F=\{q_n , ...\}$ are explicitly marked. To transform complex MSC-inline expressions as Loop- or Alternative-boxes into automata, the input alphabet is expanded with ε-transitions [15][8].

For each MSC-element a specific transformation rule is necessary. This paper will exemplify the transformation rules in case of the Alternative-Operator. All other transformation rules for the MSC-instructions can be found in [4]. The transformation rules are modular, recursive and hierarchical structured, so that also nested MSC-inline expressions (e.g. Alternative-Operator within Optional-Operator) can be transformed. Each MSC-instruction gets transformed separately into a related automaton. Afterwards the automata are conducted by concatenation or aggregation [12][2][3].

During the transformation of an Alternative-Operator each section is considered separately. The content of each section (e.g. events) gets transformed based on the related transformation rules. Afterwards the created automata of the sections are conducted by aggregation with ε-transitions (Figure 6). The result of the transformation is a NFA-ε. The aggregated automata branch out overε-transitions and represent the alternatives event-sequences. In Figure 7 an example is presented. After receiving the signal x_1 two alternatives are possible: 1) Either the signal y_1 is sent or 2) the signal x_1 is received.

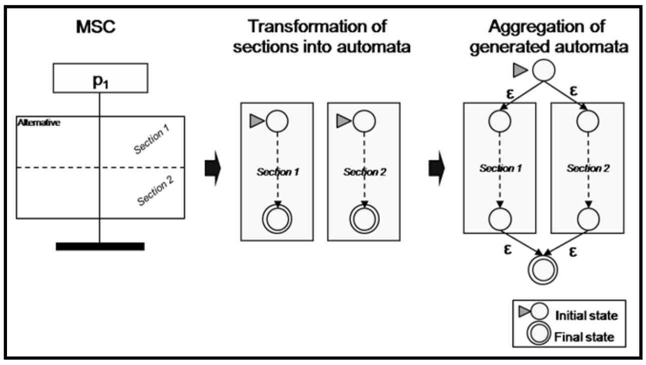

Figure 6 *: Transformation of an Alternative-Operator*

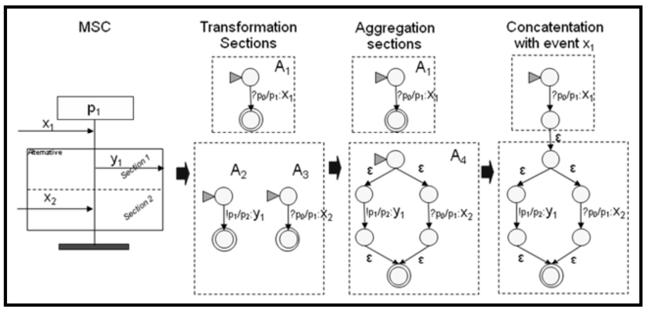

Figure 7 *: Example for the transformation of an Alternative-Operato*

During the transformation phase: At first the event x_1 is transformed into an automaton A_1. Afterwards the Alternative-Operator is transformed. Therefore the content (nested events) of Section 1 and Section 2 are transformed in the automata (A_2, A_3). These two automata (A_2, A_3) are aggregated to A_4. A_4 is the related automaton of the Alternative-Operator. At least the automaton A_1 (of event x_1) and the automaton A_4 (of the Alternative-Operator) are conducted by concatenation.

5.2 Determination of automata

Deterministic automata are the precondition for the heuristic comparison method of this approach. The automata theory [8] describes methods for the determination of finite automata. This is done in two steps:

1. ε-elimination: Elimination of all ε-transitions. Every NFA-ε is converted into a NFA.
2. Determination: Conversion from NFA to DFA.

A positive side effect is that the ε-elimination reduces the complexity and the size of the automata. A full reduction of the automata leads to a minimal size of the automata.

This can be reached by using the method minimization [8]. In [1] the calculation time of different minimization-methods are evaluated. For the method described in this paper the minimization is not mandatory.

5.3 Compatibility analysis

On basis of deterministic automata the backward compatibility will be analyzed. In this chapter a developed heuristic algorithm is described. Figure 8 shows two MSCs (MSC_1 and MSC_2), which were transformed into correspondent deterministic automata (A_1, A_2). There are only two automata, because each MSC has just one instance: $MSC_1 \rightarrow A_1$ and $MSC_2 \rightarrow A_2$.

For the comparison of the generated automata a heuristic algorithm is needed. Powerful commercial tools already exist, but these are quite expensive and furthermore these do not consider compatibility constraints.

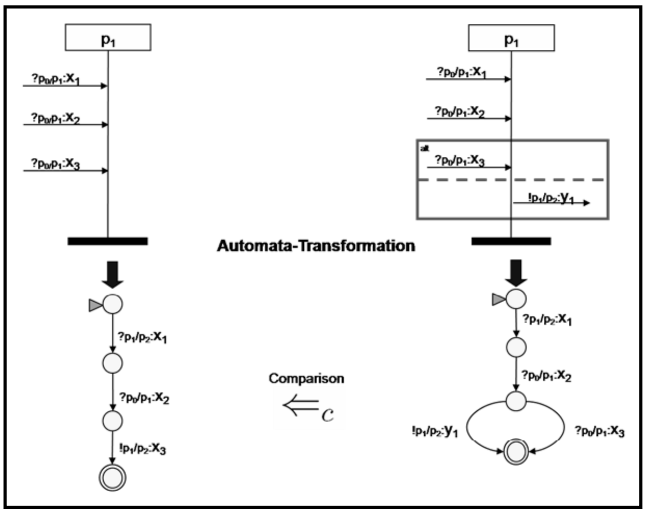

***Figure 8** : Analysis of backward compatibility based on automata*

This paper describes only the heuristic algorithm, the usage of the compatibility constraints can be read in [4]. Based on Definition 2, the heuristic algorithm has to handle following restrictions for the corresponding automaton:

• The set of infinite automata M_1 of MSC_1 has to be subset of the set of infinite automata M_2 of MSC_2.

• The set of transitions $\delta(M_1)$ of M_1 has to be backward compatible to the set of transitions $\delta(M_2)$ of M_2

For testing, if the set of automata M_1 is a subset of M_2, all automata of M_1 are singularly compared with all automata of M_2. The heuristic algorithm is explained based on Figure 8.

The two automata A_1 and A_2 should be compared. Starting from the initial state, the automata are traversed in parallel and all states and transitions get compared. If in automaton A_1 a finite state is reached and in automaton A_2 not, the comparison aborts and the automata are incompatible.

During the comparison of two transitions, first the prefixes are compared (here: $?p_1/p_2$:). These must be equivalent. Afterwards the events get compared. These must be equivalent. Due to modeling variations (e.g. different modelers) the names of events may differ. To resolve such inconsistencies a compatibility matrix can be defined [4], which implements the function nc mentioned in Definition 2.

The traversed states of A_1 and A_2 were marked in tuples as traversed. In this way loops can be recognized and the traversing can be aborted, if a marked tuple should be repeatedly traversed. The parallel traversing is running until the automaton A_1 is completely traversed or an incompatibility is detected. If the automaton A_1 is completely traversed without any abort, the automaton A_1 is a subset of A_2. A_2 is so backward-compatible to A_1.

6. Results

All introduced transformation rules and algorithms were implemented as Java-Software-Modules on the Software-Platform CAMP (Common Application Module Platform). CAMP provides an platform, on which SW-Modules can be developed and executed [14][16].

6.1 Comparison of transformation rules

Figure 9 shows how many states, transitions and ε-transitions were created during the transformation of different MSC-constructs. Therefore test cases with the constructs Event, Condition, Optional, Alternative, Parallel, Loop and Coregion were defined. Each test case consists of 6 Basic-MSC-elements (Events, Conditions) and of related Inline-Expressions. In the figure is visible that for the Optional-Operator additional ε-transitions and for the Alternative-Operator also additional states were added. During the transformation of the Parallel-Operator the sections of the Operator are transformed separately into automata and afterwards a product automaton is built [4]. In order to reduce the complexity of the product automaton, the ε-elimination is executed before. That's why no ε-transitions are existing here. In case of the Loop-Operator, the created automaton of 6 events is concatenated 6 times because of the parameters <0,6> (which means that the loop is executed at least 0 times and at most 6 times). Therefore the automaton owns 6 times more states, transitions and ε-transitions than events. Coregion is the most complex construct. Here are 1000 times more elements created than for Event.

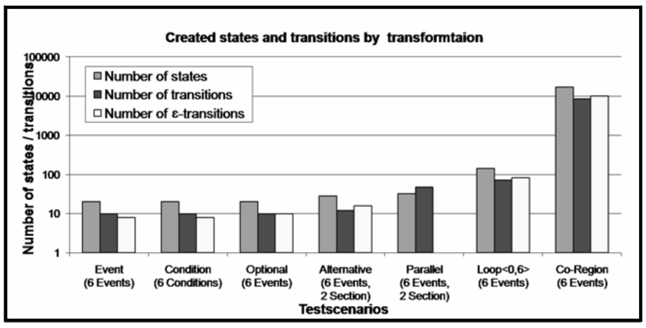

***Figure 9**: Created states and transitions by the transformation*

6.2 Determination on generated automata

After the transformation, the automata were determined. This means the ε-elimination and the determination were executed. Through the elimination of ε-transitions the set of automaton elements were strongly reduced. To show this effect, the set of states and transitions (incl. ε-transitions) is compared with the set of states and transitions after the determination (Figure 10). It is visible that the transitions and the states are reduced by 50% because of the ε-elimination. Since the generated deterministic automata are the input for the comparison algorithm, its calculation time can be reduced too.

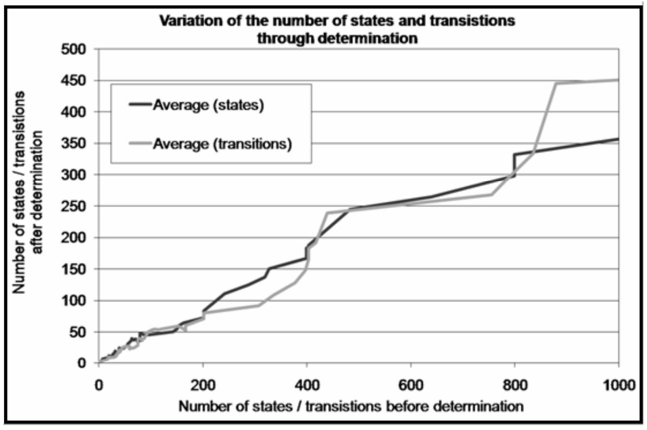

***Figure 10**: Variation of the number of states and transitions through determination*

6.3 Comparison of calculation times

In Figure 11 the calculation time for the transformation, the determination and for the compatibility analysis are compared. For this figure the calculation time in [ms] of many specific test cases were measured and compared with the set of states and transitions (incl. ε-transitions). In the chart it is visible, that the calculation time of the automata-transformation longs last up to 3500 elements. Starting from 3500 elements the complexity of the determination (incl. ε-elimination) is increasing strongly. The calculation time for the compatibility analysis is, compared to the others, quite short. The transformation of two equivalent MSCs with a Loop <0,6> and 6 Events (Figure 9) takes just 218ms. Hereby 288 states and 308 transitions (incl. ε-transitions) were created. The determination (incl. ε-elimination) itself requires 314ms. As the integrated method for the systematic comparison is fully implemented, the efficiency and the validity of the approach could be reproducible proved on several test cases. We were also able to demonstrate that the approach fullfils industrial requirements by solving an industrial real-world problem.

7. Summary

This paper described an integrated method for analyzing dynamic behavior, described by MSCs, regarding backward compatibility. Therefore (backward) compatibility was defined first. Further the integrated method was explained stepwise. The transformation from MSCs in automata was shown in detail. The related transformation rules are an extension of the approach [10] and can transform nested MSC-constructs. Also the heuristic algorithm for comparative analysis of automata was explained. The described approach has been implemented and was evaluated by analyzing the results of some examples. The developed tools are used in practice by an automobile manufacturer.

8. References

[1] M. Almeida, N. Moreira and R. Reis, "On the performance of automata minimization algorithms", *Technical report, Departamento de Ciencia de Computadores,* 2007.

[2] B. Braatz, "Vorstellung 3. Meilenstein, Deterministische Automaten, Konstruktionen", *Lecture, TU Berlin,* December. 2006.

[3] B. Braatz, "Konstruktion und Vergleiche auf Automaten", *Lecture, TU Berlin,* Juni 2007.

[4] M. Glockner, "Methoden zur Analyse von Rückwärts-kompatibilität von Steuergeräten", *PhD thesis, Chemnitz University of Technology,* 2008.

[5] S. Uchitel, J. Kramer and J. Magee, "Detecting implied scenarios in message sequence chart specifications", *In Proceedings of the joint 8th ESEC and 9th FSE,* pp.74-82, 2001.

[6] J. Grabowski, "Test Case Generation and Test Case Specification with Message Sequence Charts", *PhD thesis, University of Bern,* 1994.

[7] J. G. Henriksen, M. Mukund, K. N. Kumarm, M. Sohoni, and P. Thiagarajan, "A theory of regular MSC languages", *Information and Computation,* vol. 202, no. 1, pp. 1–38, 2005.

[8] J. E. Hopcroft, R. Motwani and J. D. Ullman, "Einführung in die Automatentheorie, Formale Sprachen und Komplexitätstheorie", Addison-Wesley, edition 2, 2002.

[9] ITU, "Message Sequence Chart (MSC)", *ITU-T,* chpt. Z.120, 2000.

[10] I. H. Krüger, "Distributed System Design with Message Sequence Charts", *PhD thesis, Technische Universität München,* 2000.

[11] N. H. Lee and S. D. Cha, "Generating a reduced finite state machine from concurrent scenarios using static partial order method", *Journal of research and practice in information technology,* vol. 36, no. 3, pp. 145–156, 2004.

[12] G. Peters, "Endlicher Automat, Reguläre Grammatik und Regulärer Ausdruck", *Lecture,* 2007.

[13] I. Schieferdecker and J. Grabowski, "The graphical

format of ttcn-3 in the context of MSC and UML", *LNCS,* vol. 2599, pp. 233–252, 2003.

[14] A. Scholand, "Konzeption und Implementierung einer Java-basierten Integrationsplattform für Software Module", *Student Thesis, University of Paderborn,* 2003.

[15] U. Schöning, "Theoretische Informatik - kurzgefasst, Spektrum", edition 4, 2001.

[16] M. Visarius, J. Lessmann, W. Hardt, F. Kelso and W. Thronicke., "An xml format based integration infrastructure for ip based design", *In Proceedings of the 16th Symposium on Integrated Circuits and Systems Design (SBCCI),* pages 119–124, September. 2003.