# Translating Message Sequence Charts to other Process Languages using Process Mining

Kristian Bisgaard Lassen[1], Boudewijn F. van Dongen[2], and Wil M.P. van der Aalst[2]

[1] Department of Computer Science, University of Aarhus,
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark.
k.b.lassen@daimi.au.dk
[2] Department of Information Systems, Eindhoven University of Technology
P.O. Box 513, NL-5600 MB, Eindhoven, The Netherlands.
{b.f.v.dongen,w.m.p.v.d.aalst}@tue.nl

**Abstract.** Message Sequence Charts (MSCs) are a well known language for specifying scenarios that describe how different actors (e.g., system components, people, or organizations) interact. MSCs are often used as a starting point for software analysts to discuss the behavior of a system with different stakeholders. Often such discussions lead to more complete behavioral models described by e.g. Event-driven Process Chains (EPCs), UML activity diagrams, BPMN models, Petri nets, etc. The contribution of this paper is to present a method that uses process mining to translate a set of MSCs that represent example scenarios into a complete process model, e.g., represented in terms of EPCs or Petri nets. Our approach takes MSCs and translates them into a special kind event logs. Unlike all known process mining techniques, we use a new approach that uses event logs containing explicit causal dependencies. This allows us to discover high-quality process models. The approach has been implemented in the process mining framework ProM.

## 1 Introduction

*Message Sequence Charts (MSCs)* [26, 22] are a well-known language to specify communication between processes, and are supported by many tools, standards, and approaches, e.g., the Object Management Group (OMG) has decided to adapt a variant of them called Sequence Charts in the UML notational framework [17]. In this paper we look at MSCs that are restricted to only using processes and messages. We do not consider structured language features such as choice and iteration that the UML 2.0 standard introduces, i.e., we consider basic MSCs rather than high-level MSCs. The reason for abstracting from these high-level constructs is that users typically *use MSCs only to model example scenarios* and not complete process models. Therefore, most users do not use routing constructs such a parallel routing, iteration, choice, etc. in MSCs.

When developing a system it is often useful to describe requirements for the system by MSCs were each MSC depicts a single scenario. For example, we have

been involved in a project were many MSCs were generated by the staff at a hospital and software developers to capture the requirements for a new pervasive health care system [25]. The strength of MSCs is that they depict a single scenario using an intuitive notation. Therefore, they are easy to understand. However, this strength can at the same time also be considered a weakness. How does one consolidate several scenarios from the same system? This is far from trivial. For example, two MSCs may be similar up to a certain point, after which they diverge. This point corresponds to a choice if we look at them in combination, but this is not clear just by looking at one of the MSCs. Synchronization points are not discovered just by looking at one diagram, one would again have to consolidate all MSCs. Note that many behavioral patterns can be described implicitly when using multiple MSCs.

Considerable work has been done on the synthesis of scenario-based models such as MSCs (see [24] and the Related Work section for an overview). Existing approaches are very different and typically have problems dealing with concurrency. Moreover, the majority of approaches uses explicit annotations to "glue" MSCs together in a single model. For example, high-level MSCs are being used [16, 27, 28] or there are "precharts", "state conditions" or similar concepts to explicitly relate MSCs [11, 20, 21]. Other problems are related to performance, implied scenarios (i.e., the model allows for more behavior than what has actually been observed), and consistency (e.g., the synthesized model contains deadlocks) [6, 7]. Therefore, it is interesting to apply ideas from process mining [2–4, 9, 12, 14, 23] to the synthesis of MSCs.

The aim of the approach presented in this paper is to generate process models - represented using languages such as Event-driven Process Chains (EPCs), Petri nets, YAWL [1], or even BPEL [8] - based on a set of MSCs. The large variety of languages that we can use to represent models is a result of using our process mining framework ProM [15, 29] which supports a wide variety of process modeling languages (Petri nets, EPCs, YAWL, BPEL, transition systems, heuristics nets, etc.). Using ProM the process mining results can be mapped on any of these languages.

This paper is organized as follows. First we provide some background information required to understand our approach (Section 2). Then, in Section 3, we describe how we actually generate the behavioral models. Section 4 presents an example of an online bookstore where we apply our method and Section 5 discusses related work. Section 6 concludes the paper.

## 2 Preliminaries

### 2.1 Message Sequence Charts

As mentioned in the introduction, several variants of MSC exists, such as UML 2.0 Sequence Charts [17] and Live Sequence Charts [11]. In this paper we focus on MSCs with only two syntactical constructs, i.e., processes and messages. Processes can be used to denote a wide variety of entities ranging from software

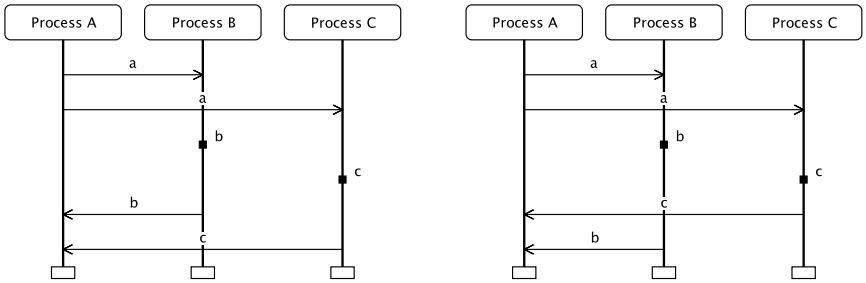


**Fig. 1.** Two example MSCs of three processes communicating.

components and web services to people and organizations. A message is passed from one process to the other and therefore each message has a, (not necessarily different) sender and receiver process. A process has a lifeline representing a sequence of messages that the process is the sender or receiver of.

In Figure 1 we show two examples of the MSCs that we consider. Each of them consists of three processes, `Process A`, `Process B`, and `Process C`. These processes are represented by their name and their lifelines. These lifelines are connected to each other by messages, which are represented by labelled arrows. Internal events (i.e., messages where the sender and receiver are the same process) are represented by a box on the lifeline of the corresponding process, again with a label.

The process shown in both MSCs of Figure 1 starts by the sending of message `a` by `Process A`. Then message `a` is received by `Process B`. Next `Process A` sends a message `a` to `Process C`. The overall process continues like that until `Process C` sends a message `c` that is received by `Process A` and `Process B` sends a message `b` that is received by `Process A`.

### 2.2 Process Mining

The goal of process mining, or more specifically *control flow discovery* is to extract information about processes from transaction logs, such that we can discover the control flow of a process in a model. Process mining always starts from an event log that contains events such that (i) each event refers to an *activity* (i.e., a well-defined step in the process), (ii) each event refers to a *case* (i.e., a process instance) and (iii) events have a *timestamp* implying a total order on them. Table 1 shows an example of a log involving 19 events and 5 activities. This event log happens to also contain information about the people executing the corresponding activities (cf. the orginator field Table 1). Often logs also contain information about data associated to events (e.g., message content).

Figure 2 shows some examples of process mining results that can be obtained using the log of Table 1. This figure clearly shows that process mining is not limited to control flow discovery, i.e., the roles of the actors in the process, as

**Table 1.** An event log (audit trail).

| case id | activity id | originator | case id | activity id | originator |
|---|---|---|---|---|---|
| case 1 | activity A | John | case 5 | activity A | Sue |
| case 2 | activity A | John | case 4 | activity C | Carol |
| case 3 | activity A | Sue | case 1 | activity D | Pete |
| case 3 | activity B | Carol | case 3 | activity C | Sue |
| case 1 | activity B | Mike | case 3 | activity D | Pete |
| case 1 | activity C | John | case 4 | activity B | Sue |
| case 2 | activity C | Mike | case 5 | activity E | Clare |
| case 4 | activity A | Sue | case 5 | activity D | Clare |
| case 2 | activity B | John | case 4 | activity D | Pete |
| case 2 | activity D | Pete | | | |

well as their social relations can be discovered as well. However, in this paper, we only focus on the control flow, which in Figure 2(a) is shown as a Petri net.

Although in this paper we focus on control flow discovery, we make a fundamentally different assumption with regard to the process log, i.e., we say that events are no longer *totally ordered*. Instead, we consider logs, where events are *partially ordered*, since Message Sequence Charts can be considered as partial orders on events (i.e., each event being the sending and receiving of a message). For this, we had to make changes to the very core of the *ProM Framework*, which we introduce in the next section.

### 2.3 The ProM Framework

The (Pro)cess (M)ining framework *ProM* has been developed as a completely plug-able environment for process mining and related topics. It can be extended by simply adding plugins, i.e., there is no need to know or to recompile the source code. Currently, more than 130 plugins have been added. For

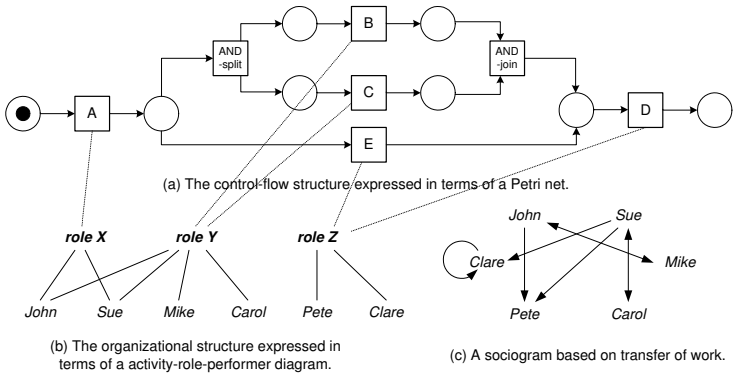


**Fig. 2.** Some mining results from different perspectives, based on the log of Table 1.

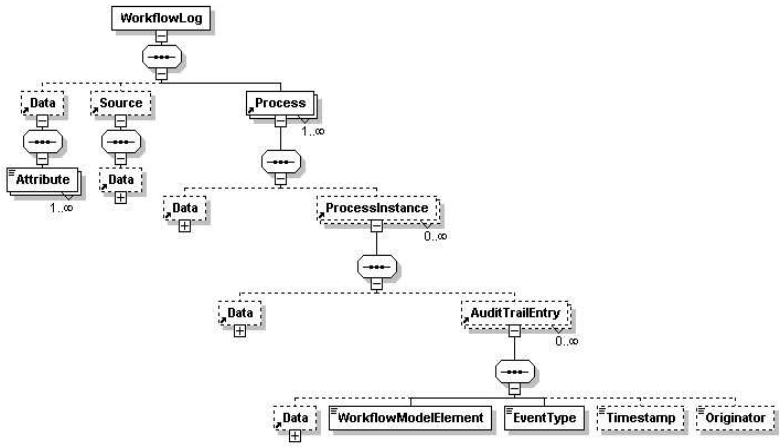


**Fig. 3.** MXML schema Definition.

more information on the ProM framework, we refer to [15, 29] and the web site www.processmining.org. The most interesting plugins in the context of this paper are the mining plugins, that focus on control flow discovery. However, besides mining plugins the architecture of ProM allows for four more types of plugins:

**Mining plugins** which implement some mining algorithm, e.g., mining algorithms that construct a Petri net based on some event log.

**Export plugins** which implement some "save as" functionality for some objects (such as graphs). For example, there are plugins to save EPCs, Petri nets, spreadsheets, etc.

**Import plugins** which implement an "open" functionality for exported objects, e.g., load instance-EPCs from ARIS PPM.

**Analysis plugins** which typically implement some property analysis on some mining result. For example, for Petri nets there is a plugin which constructs place invariants, transition invariants, and a coverability graph.

**Conversion plugins** which implement conversions between different data formats, e.g., from EPCs to Petri nets and from Petri nets to YAWL and BPEL.

ProM uses a standard log format, named MXML as described in [13] for storing process logs, such as the one in Table 1. In the context of the ProMimport framework [19], several adaptors have been developed to map logs from different information systems onto MXML (e.g., Staffware, FLOWer, MS Exchange, MQSeries, etc.). Figure 3 shows the hierarchical structure of MXML. The format is XML-based and is defined by an XML schema (cf. www.processmining.org).

In this paper, we show that we can use the ProM framework to analyze MSCs. However, to take full advantage of the extra information contained in MSCs, which can typically not be found in transaction logs, we had to modify the framework in two ways, which we describe in detail in the next section.

## 3 Generating Process Models from MSCs

This section presents our approach of generating a process model from a set of MSCs. To do so, we first had to construct a mapping from MSCs to transaction logs, i.e., we fixed which part of an MSC refers to an MXML process instance, an MXML audit trail entry, an MXML workflow model element, i.e., all obligatory elements in MXML.

### 3.1 MSCs to MXML

In the first phase of our approach, we take a set of MSCs, all describing the same system and we translate them to one MXML log file. We have chosen to use XMI [18], the Object Management Group (OMG) standard interchange format, as our input format for MSCs, which enabled us to implement the translation in a plugin for the ProMimport framework [19].

In this ProMimport plugin, each MSC is translated into one MXML process instance. The reason for this is simple. Since each MSC describes one possible execution scenario of the system, it corresponds nicely to the notion of a case also referred to as process instance.

Furthermore, within each MSC, all messages are translated into *two* audit trail entries; one referring to the sending of the message and one referring to the receiving of the message. To accomplish this, we made sure that both audit trail entries refer to the same workflow model element (i.e., the message). The audit trail entry that refers to the sending of the message has event type `start`. Receiving audit trail entries have type `complete`.

To incorporate the information about processes in MXML, we use the originator field of each audit trail entry in a trivial way. If `Process A` sends message `a` to `Process B`, then the originator field of the audit trail entry relating to the sending of the message equals `Process A` and the originator field of the audit trail entry relating to the receiving of the message equals `Process B`.

Finally, we add data to each audit trail entry, such that each audit trail entry has a unique label within a process instance. Then, using these labels, we store the set of predecessors and successors that we have observed in the MSC. Consider for example the MSC of Figure 1, where the first three events are: (1) the sending of message `a` by `Process A`, (2) the receiving of message `a` by `Process B` and (3) the sending of message `a` by `Process A`. These events are represented in MXML by the audit trail entries in Figure 4.

The relations between audit trail entries stored in the data part of MXML are built in a trivial way. If an audit trial entry refers to the sending of a message, the preset of that audit trail entry is the event that happened before it on the lifeline of that process. If the audit trail entry refers to the receiving of a message, the preset *also* contains the audit trail entry referring to the sending of the same message. The postsets are build in a similar fashion.

Up to this point, the ProM framework used to order events in a process log based on two criteria, namely the timestamp of each audit trail entry and/or their relative position in the MXML file. Therefore, we had to extend the ProM

```
<ProcessInstance id="id-instance-0">
  <Data>
     <Attribute name="isPartialOrder">true</Attribute>
  </Data>
  <AuditTrailEntry>
     <Data>
       <Attribute name="ATE_id">id1</Attribute>
       <Attribute name="ATE_post">id2,id3</Attribute>
       <Attribute name="ATE_pre"></Attribute>
     </Data>
     <WorkflowModelElement>a</WorkflowModelElement>
     <EventType>start</EventType>
     <Originator>Process A</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <Data>
      <Attribute name="ATE_id">id2</Attribute>
      <Attribute name="ATE_post">id6</Attribute>
      <Attribute name="ATE_pre">id1</Attribute>
    </Data>
    <WorkflowModelElement>a</WorkflowModelElement>
    <EventType>complete</EventType>
    <Originator>Process C</Originator>
  </AuditTrailEntry>
  <AuditTrailEntry>
    <Data>
      <Attribute name="ATE_id">id3</Attribute>
      <Attribute name="ATE_post">id4,id5</Attribute>
      <Attribute name="ATE_pre">id1</Attribute>
    </Data>
    <WorkflowModelElement>a</WorkflowModelElement>
    <EventType>start</EventType>
    <Originator>Process A</Originator>
  </AuditTrailEntry>
```

**Fig. 4.** Example MXML log with partial order stored in attributes.

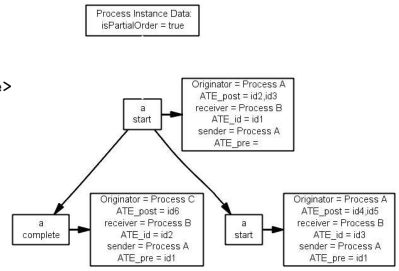


**Fig. 5.** The MXML snapshot of Figure 4 shown in ProM.

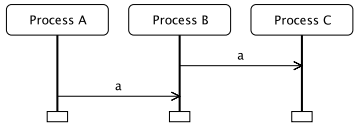


**Fig. 6.** An MSC that leads to problems in the aggregation.

framework to store and read the partial order we represented in the data elements `ATE_id`, `ATE_post` and `ATE_pre` (see Figure 4. Furthermore, to indicate that a process instance is a partial order, we added a data element `isPartialOrder` to the data part of each process instance, allowing us to mix partial and linear orders in one log file.

Figure 4 shows a part of an MXML file illustrating the partial order extension. It contains the `isPartialOrder` flag for the process instance and the identifiers and pre- and postsets for the audit trail entries. Figure 5 shows the same log in ProM, where it is clearly represented as a partial order.

Since the ProM framework is now capable of reading partial orders from MXML files, we needed new plugins for mining partial orders. Fortunately, we could build on the ideas and plugins introduced in [12] and [14].

### 3.2 Process Discovery on Partial Orders

The second and final phase of our approach is to mine a process model using the MXML document that describes the example scenarios, i.e., partially ordered example behaviors of the system. Furthermore, we are interested in constructing

models that describe the behavior of each individual process present in the MSCs. To be able to make a model from the MSCs, we extended an already implemented mining algorithm in ProM.

The Multi Phase Mining plugin, which was introduced in [12] and [14] is an implementation of a multi-stage algorithm. The multi phase approach presented in these papers, basically consist of the following phases, when executed on a totally ordered log file:

1. First the whole log is analyzed and causal dependencies between event are determined in a way similar as presented in [3].
2. Then, each process instance is translated from a linear order of events into a partial order, following the discovered causal dependencies.
3. Finally, these partial orders are aggregated and translated into some model, such as an EPC or Petri net.

In the context of our process discovery problem, this algorithm seems to fit perfectly: We can just skip the first two phases and directly aggregate the partial orders that we have as input. However, there are two important requirements:

**Partial Orders need to be minimal** The aggregation algorithm presented in [14] assumes that the partial orders used as input are minimal, i.e., there are no two paths between two nodes. This requirement is clearly not met by our MSCs, i.e., if `Process A` sends message `a` to `Process B` and then gets message `b` back from `Process B`, i.e., there are two paths between the audit trail entry referring to the sending message `a` and the one referring to the receiving of message `b` (see Figure 1). Therefore, we apply the concept of *transitive reduction* [5] to each MSC, before aggregating them[1].

**Input and Output sets are uniquely labelled** The second requirement for our aggregation algorithm is that no single audit trail entry is preceded or succeeded by two audit trail entries with the same label twice. Consider Figure 6, where the start event of message `a` by `process B` is followed by two complete events of the same message, i.e. the one coming in from `process A` and the one going to `process c`. If we make sure that a situation, we can be certain that the aggregation algorithm produces a correct result.

Before we present the aggregation algorithm, we first consider the abilities of ProM to filter logs and we show how this filtering procedure we adapted for partially ordered log files.

### 3.3 MSC projection

Although the MSCs can be used directly in process mining, our experience is that very often, people are not interested in the overall process, but only in

[1] The idea of transitive reduction is that an edge between two nodes in a graph is removed if there is a different path from the source node to the target node. Since partial orders are a-cyclic, the transitive reduction is unique [5].

parts thereof. For this purpose, ProM is equipped with log filters, which look at each process instance and filter out the required information. For example, to get the first idea about a process, we typically only consider complete events. This process, of removing audit trail entries from each process instance is called *projection*.

Log filters typically remove audit trail entries that are not of interest. If audit trail entries are totally ordered, then this is easy to do. In our case, however, audit trail entries are partially ordered, in which case it is less trivial to remove audit trail entries. In fact, audit trail entries can only be removed if the partial order is first *transitively closed*.

For this purpose, we implemented two log filters in ProM. The first one transitively closed the partial order. Then already existing filters do their filtering and afterward our second plugin transitively reduces the partial order[2]. In Section 4, we show the use of these filters in practice, however, we first introduce the aggregation algorithm and a translation algorithm.

### 3.4 Aggregation and Translation

Under the assumption that our MXML file contains information about partial orders, we can use the *partial order aggregator* to generate a so-called aggregation graph. This partial order aggregator is a new mining plugin in the ProM framework, which accepts an MXML file, but it ignores each process instance for which the `isPartialOrder` flag is not set to `true`. The plugin aggregates the partial orders into an *aggregation graph*.

The aggregation graph that results from this procedure is described in detail in [14]. In essence, an aggregation graph is a straightforward sum over a set of partial orders, with two unique nodes $t_s$ and $t_f$ in such a way that $t_s$ is the only source node and $t_f$ is the only sink node. The labels of nodes and edges represent the number of times it was visited in some partial order. Figure 7 shows the result, after aggregating the two partial order generated from the MSCs in Figure 1. Note that we first projected the MSCs onto the audit trail entries where `process C` is described in the `originator` field.

[2] Due to space limitations, we cannot provide proofs that the transitive reduction of the transitive closure is isomorphic to the transitive closure of the original.

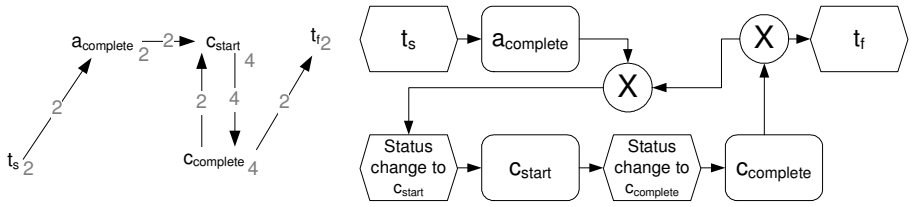


**Fig. 7.** The aggregation graph and EPC of process C after aggregating Figure 1.

Finally, each of these aggregation graphs can be translated into an EPC, i.e., into a human-readable format. This is where the requirements that the MSCs are minimal and that the in- and output sets are uniquely labelled are important, since the translation depends on the labels of nodes and edges. In short, if a node has the same label as each of its input edges, it is an AND-join, if a node has a label that equals the sum of the labels of all input edges it is an XOR-join and otherwise an OR-join and symmetrically for the split type[3].

In the next section, we consider a realistic example, to show the practical applicability of our approach.

## 4 Bookstore Example

In this section we present an application of our process mining approach to a system modelled by MSCs. The system that we describe is an online bookstore. The system contains the following four MSC processes:

**Customer** The person that wants to buy a book.
**Bookstore** Handles the customer request to buy a book. The virtual bookstore always contacts a publisher to see if it can handle the order. If the bookstore cannot find a suitable publisher, the customer's order is rejected. If the bookstore finds a publisher, a shipper is selected and the book is shipped. During shipping, the bookstore sends a bill to the customer.
**Publisher** May or may not have the book that the customer wants to buy.
**Shipper** May or may not be able to carry out the shipping order.

We start out with some message sequence charts. In Figure 8 we see two examples where the customer orders a book at the bookstore. The bookstore

[3] If one of the two requirements of the input is violated, these translation rules may not valid.

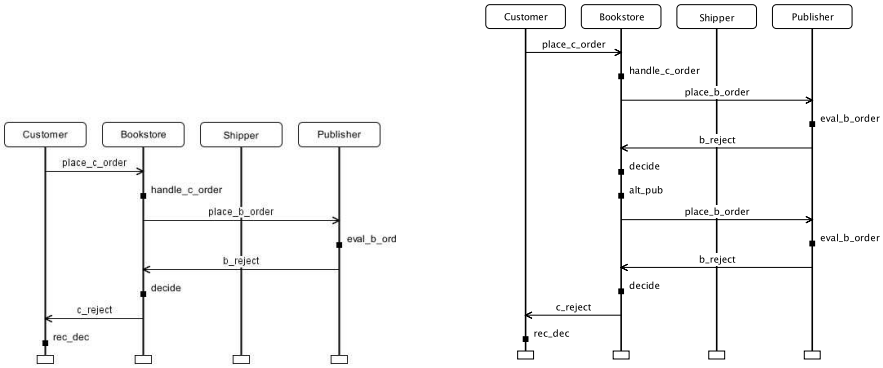


**Fig. 8.** Two MSCs describing scenarios in our bookstore example.

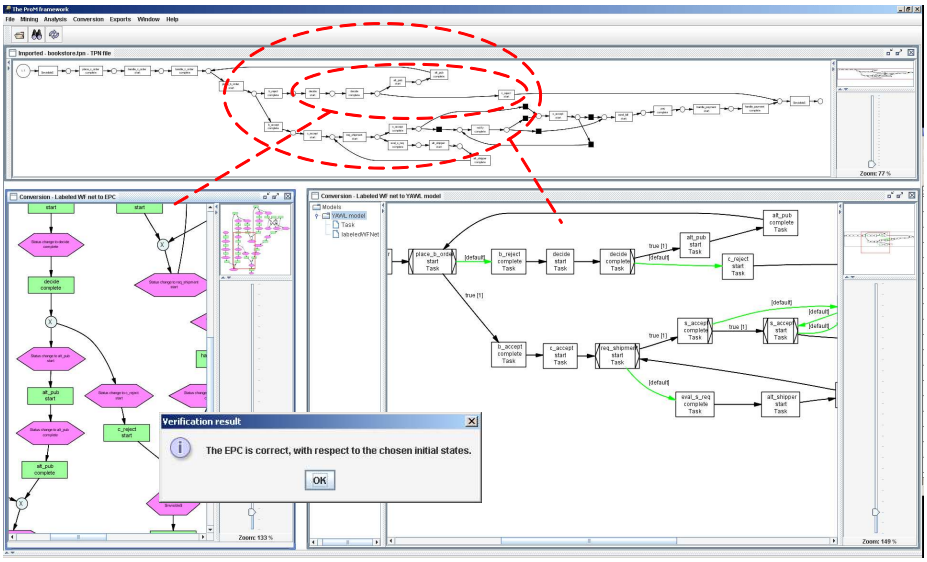


**Fig. 9.** ProM showing the mined bookstore process in different languages.

then tries to get the book from different publishers (1 in the left hand figure, 2 in the right hand figure) until it gives up and tells the customer that the order cannot be fulfilled. Note that in both cases, the shipper does not to anything.

We started from 10 of such MSCs, containing 23 different messages and we translated them into MXML using the plugin for the ProMimport tool we developed. Then, we used the process mining technique described in Section 3 to obtain process models for the bookstore, the publisher, the shipper and the customer, by projecting each process instance on the corresponding audit trail entries.

Due to space limitations, we can not show the resulting models in detail. Hence, we limit ourselves to presenting a screenshot of ProM that shows some models discovered based on the 10 MSCs. The three process models depicted in Figure 9 are the result of projecting the log onto the bookstore, i.e., we only took into account the audit trail entries that had the bookstore as an originator. The Petri net net at the top of the screenshot shows the whole bookstore process discovered using the approach described in this paper. As indicated before, ProM can transform models from one notation to another, e.g., mapping Petri nets onto EPCs, YAWL, BPEL, etc. The EPC in the bottom-right corner and the YAWL model in the bottom-left corner, show only a part of the process. The EPC shows the part of the Petri net in the small oval and the YAWL model the part in the big oval. Furthermore, a dialog is shown resulting from an analysis of the EPC model. ProM EPC analysis plugin reports that the EPC is a correct EPC, i.e., it contains no deadlocks or lifelocks.

## 5 Related Work

We discuss related work in the two research fields our work joins, namely *process mining* and *synthesis of scenario based models*.

*Process mining.* Since the mid-nineties several groups have been working on techniques for process mining, i.e., discovering process models based on observed events. In [2] an overview is given of the early work in this domain. The idea to apply process mining in the context of workflow management systems was introduced in [4]. Cook et al. investigated similar issues in the context of software engineering processes [9]. Herbst [23] was on of the first to tackle more complicated processes, e.g., processes containing duplicate tasks. Most of the approaches have problems dealing with concurrency. The $\alpha$ algorithm [3] is an example of a simple technique that takes concurrency as a staring point. However, this simple algorithm has problems dealing with complicated routing constructs and noise (like most of the other approaches described in literature). Approaches based on heuristics or genetic algorithms can deal with noise [30]. Moreover, advanced techniques, e.g., based on the theory regions [10], can mine processes containing complex mixtures of choice and synchronization. This paper uses a process mining approach based on the technique presented in [12, 14]. As described in this paper, the approach has been adapted to deal with the explicit causal dependencies in scenario-based representations such as MSCs. The results presented in this paper are fully implemented in ProM (www.processmining.org). ProM serves as a testbed for our process mining research. Most of the leading process mining approaches have been implemented in ProM. Through the extensions presented in this paper, these approaches can now be applied to the synthesis of scenario-based models.

*Synthesis of scenario-based models.* This paper considers a specific format for representing scenarios, i.e., Message Sequence Charts (MSCs) [22, 26]. However, many variants and dialects are available, e.g., the ITU standard for MSCs, UML Sequence Diagrams (SDs), Communication Diagrams (CDs), Interaction Overview Diagrams (IODs), and Harel's Live Sequence Charts (LSCs), just to name a few. In their basic form, these notations model individual scenarios, i.e., a particular example behavior of the process/system and not the full behavior. However, many of the approaches have been extended with composition constructs to model a set of example behaviors or even the full process/system behavior. Some authors use the term "high-level MSCs" to refer to MSCs which are composed using operators such as "sequence", "iteration", "parallel composition", "choice", etc. We consider these high-level MSCs as less appropriate, i.e., if one just wants to model example behavior, then the basic MSCs are more suitable. However, if one wants to model the full system behavior, traditional techniques such UML activity diagrams, state charts, BPMN, EPCs, Petri-net based languages, etc. seem more appropriate.

Many researchers have been working on the synthesis of scenario-based models, in particular the generation of process models from different variants of

MSCs. In [24] an excellent overview of 21 approaches is given. This overview shows that existing approaches are very different and typically have problems dealing with concurrency. Other problems are related to performance, implied scenarios (i.e., the model allows for more behavior than what has actually been observed), and consistency (e.g., the synthesized model contains deadlocks). It is impossible to give an overview of all approaches reported in literature. Therefore, we only describe some representative examples. Harel et al. [11, 20, 21] have worked on the notion of Live Sequence Charts (LSCs). The primary goal has been to turn LSCs ("play-in") into an executable system ("play out") without necessarily constructing an explicit process model. However, in [21] the synthesis of LSCs into statecharts is described. Note that this approach is very different from the notion of synthesis used in this paper, i.e., through the so-called prechart of LSCs the links between the various MSCs are made explicit. Hence there is no real synthesis in the sense of deriving a process model from example scenarios. This holds for many other approaches, e.g., several authors assume "state conditions" or similar concepts to make the linking of MSCs explicit [16, 28]. In a way, sets of scenarios are explicitly encoded in high-level MSCs. However, there are also approaches that really derive process models from MSCs without some explicit a-priori encoding. An example is the work by Alur et al. [6, 7]. In [6, 7] two problems are discussed: the inference of additional (possibly undesirable) implied behavior and the construction of incorrect models (e.g., models having potential deadlocks). Using different algorithms implemented in ProM we can vary the degree of implied behavior and balance between under-fitting and over-fitting. For example, the plugins in ProM based on the theory of regions [10] can construct process models without any implied behavior. The plugin introduced in this paper allows for additional implied behavior (this is the effect we aim at). However, the resulting model is always correct, i.e., it does not have deadlocks and it is able to reproduce the observed MSCs. Finally, we would like to mention the approach described in [27]. Like many other authors, these authors provide formal semantics of MSCs in terms of Petri nets. The authors also synthesize MSCs into an overall Petri net. However, they assume that there is a Petri net gluing all MSCs together, i.e., sets of scenarios are explicitly encoded in high-level MSCs.

To conclude this section, we would like to point out that, unlike other methods, we are able to generate different types of process models, e.g., EPCs, Petri nets, BPEL, or YAWL. Most of the approaches described in literature can only generate statecharts or basic transition systems.

## 6 Conclusion

This paper presented a new approach to synthesize a process model from MSCs. The approach uses ideas from the process mining community and adapts these to incorporate the explicit causal dependencies present in MSCs. The approach has been fully implemented in ProM. We showed how an existing process mining algorithm can be adapted to exploit causal dependencies and that the discovered

model can be represented in different notations, e.g., EPCs, Petri nets, BPEL, and YAWL. Moreover, the ideas are not limited to MSCs and can be applied to other event logs containing explicit causal dependencies, e.g., collaboration diagrams, groupware products, document management systems, case handling systems, product data management systems, etc.

The ProM framework can be downloaded from www.processmining.org and can be freely used (it is open source software). The reader is invited to experiment with the plug-in reported in this paper and apply it to MSCs expressed in the OMG's XMI format.

## References


1. W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.
2. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003.
3. W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow Mining: Discovering Process Models from Event Logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.
4. R. Agrawal, D. Gunopulos, and F. Leymann. Mining Process Models from Workflow Logs. In *Sixth International Conference on Extending Database Technology*, pages 469–483, 1998.
5. A. Aho, M. Garey, and J. Ullman. The Transitive Reduction of a Directed Graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.
6. R. Alur, K. Etessami, and M. Yannakakis. Inference of Message Sequence Charts. *IEEE Transactions on Software Engineering*, 29(7):623–633, 2003.
7. R. Alur, K. Etessami, and M. Yannakakis. Realizability and Verification of MSC Graphs. *Theoretical Computer Science*, 331(1):97–114, 2005.
8. T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Standards proposal by BEA Systems, International Business Machines Corporation, and Microsoft Corporation, 2003.
9. J.E. Cook and A.L. Wolf. Discovering Models of Software Processes from Event-Based Data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.
10. J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets from Finite Transition Systems. *IEEE Transactions on Computers*, 47(8):859–882, August 1998.
11. W. Damm and D. Harel. LCSs: Breathing Life Into Message Sequence Charts. *Formal Methods in System Design*, 19(1):45–80, 2001.
12. B.F. van Dongen and W.M.P. van der Aalst. Multi-Phase Process Mining: Building Instance Graphs. In P. Atzeni, W. Chu, H. Lu, S. Zhou, and T.W. Ling, editors, *International Conference on Conceptual Modeling (ER 2004)*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer-Verlag, Berlin, 2004.
13. B.F. van Dongen and W.M.P. van der Aalst. A Meta Model for Process Mining Data. In *Proceedings of the CAiSE WORKSHOPS*, volume 2, pages 309–320. FEUP, 2005.

14. B.F. van Dongen and W.M.P. van der Aalst. Multi-phase Process mining: Aggregating Instance Graphs into EPCs and Petri Nets. In *PNCWB 2005 workshop*, pages 35–58, 2005.
15. B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *Application and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer-Verlag, Berlin, 2005.
16. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Tool Support for Model-Based Engineering of Web Service Compositions. In *Proceedings of 2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 95–102, Orlando, FL, USA, July 2005. IEEE Computer Society.
17. Object Management Group. *OMG Unified Modeling Language 2.0*. OMG, http://www.omg.com/uml/, 2005.
18. Object Management Group. Xml meta interchange. http://www.omg.org/technology/documents/formal/xmi.htm, 2006.
19. Christian W. Günther. ProMimport. http://promimport.sourceforge.net, 2006.
20. D. Harel. From play-in scenarios to code: An achievable dream. *Computer*, 34(1):53–60, 2001.
21. D. Harel, H. Kugler, and A. Pnueli. Synthesis Revisited: Generating Statechart Models from Scenario-Based Requirements. In *Formal Methods in Software and Systems Modeling*, volume 3393 of *Lecture Notes in Computer Science*, pages 309–324. Springer-Verlag, Berlin, 2005.
22. D. Harel and P.S. Thiagarajan. Message Sequence Charts. In *UML for Real: Design of Embedded Real-Time Systems*, pages 77–105, Norwell, MA, USA, 2003. Kluwer Academic Publishers.
23. J. Herbst. A Machine Learning Approach to Workflow Management. In *Proceedings 11th European Conference on Machine Learning*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194. Springer-Verlag, Berlin, 2000.
24. H. Liang, J. Dingel, and Z. Diskin. A Comparative Survey of Scenario-Based to State-Based Model Synthesis Approaches. In *Proceedings of the 2006 International Workshop on Scenarios and State Machines: Models, Algorithms, and Tools (SCESM06)*, pages 5–12, New York, NY, USA, 2006. ACM Press.
25. R.J. Machado, K.B. Lassen, S. Oliveira, M. Couto, and P. Pinto. Execution of UML Models with CPN Tools for Workflow Requirements Validation. In *Proc. of Sixth CPN Workshop*, volume PB-576 of *DAIMI*, pages 231–250, 2005.
26. E. Rudolph, J. Grabowski, and P. Graubmann. Tutorial on Message Sequence Charts. *Computer Networks and ISDN Systems*, 28(12):1629–1641, 1996.
27. M. Sgroi, A. Kondratyev, Y. Watanabe, L. Lavagno, and A. Sangiovanni-Vincentelli. Synthesis of Petri Nets from Message Sequence Charts Specifications for Protocol Design. In *Design, Analysis and Simulation of Distributed Systems Symposium (DASD '04)*, pages 193–199, Washington DC, USA, April 2004.
28. S. Uchitel, J. Kramer, and J. Magee. Synthesis of Behavioral Models from Scenarios. *IEEE Transactions on Software Engineering*, 29(2):99–115, 2003.
29. H.M.W. Verbeek, B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Interoperability in the ProM Framework. In *EMOI workshop*, 2006.
30. A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering Workflow Models from Event-Based Data using Little Thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.