From BPMN to Sequence Diagrams: Transformation and Traceability

Aljia Bouzidi¹, Nahla Zaaboub Haddar¹, Mounira Ben-Abdallah¹ and Kais Haddar²

¹Faculty of Economics and Management, Sfax University, Sfax, Tunisia
²Faculty of Sciences, Sfax University, Sfax, Tunisia

Keywords: Alignment, Traceability, Model Transformation, BPMN Model, UML Sequence Diagram, MVC.

Abstract: A business cannot be competitive unless its business process is aligned with its information system. Indeed, a perfect alignment is key to a coherent management and success of the business. Therefore, it is important to bring closer business process- and IS modeling activities. The current paper presents an approach to derive a dynamic software model from a business process model, including the trace links between source and target elements. Our approach is based on a set of rules that transform a BPMN business process model into a UML sequence diagram structured according to the model view controller design pattern, and a trace model. To show the feasibility of approach in the practice, we developed a tool that implements the transformation rules.

1 INTRODUCTION

A business process model (bpm) represents the way operations are carried out to accomplish business goals. An automated information system (IS) gives important support to the bpm if its capacities are best exploited. Indeed, the IS can offer complete information, allowing to manage one’s business more efficiently, gain a cost advantage over competitors, and make tough decisions. However this can not be reached when the business process is not aligned with its IS. Therefore, it is important to start IS modeling from bpm. In modern software development methods, analysts start the development process with an inception phase to acquire a deep knowledge of the business process model. This phase is crucial since it prepares for requirement discovery and analysis. However, artifacts produced in this phase, such as the bpm, are not exploited in downstream software development phases. A transformation mechanism would produce models that can be used as a starting point for the construction of the structural and behavioural perspectives of the analysis model. The Model Driven Architecture approach (MDA) (OMG-MDA, 2006) recommends the model transformation mechanism between heterogeneous models. Accordingly, the transition from the problem domain expressed with BPMN notations into software models expressed with UML may be resolved basing on the principles of model transformation from the computation independent model (CIM) level that hosts the business model into the platform-independent model (PIM) level that encloses the IS analysis models. The MDA approach is commonly used to get from a PIM to a platform-specific model (PSM) then from a PSM to code. But, very little works have contributed to the CIM to PIM transformation, which in turn commonly focus on the static or/and functional viewpoints.

On the other hand, if a change of the IS model is needed, great efforts, money and time are wasted during the modification process to accommodate the impact of changes, as there is a request to check continuously if the business process model is aware of the changes. Therefore, the need for an approach that deals with the changes of models is of great value to software engineers (Arman and Jabbarin, 2015), (Jabbarin and Arman, 2014). Traceability is an attempt to address this issue. It consists in establishing trace links between overlapping elements. These links may be used then for various practical software diagrams, such as model consistency check, change impact analysis and identification of misalignment sources. However, traceability is not enough addressed in the literature.

These shortcomings form the main motivation factor to propose a new approach that defines model transformations for generating the sequence diagrams from a bpm expressed in BPMN (OMGBPMN, 2013). Besides, it defines a set of trace links that allow relating the overlapping elements to maintain them always aligned. This work is complementary to existing works that deal with static and functional viewpoints.
of the analysis model of the IS so that it can be reused and refined during the overall system design process (Berrocal et al., 2014).

To get to the heart of our approach, the remainder of this paper is structured as follows. The next section 2 discusses related works. In the third section 3, we provide the current proposal contributions. The fourth section 4 denotes the main advantages of our approach. Finally, section 5 concludes and draws some future works.

2 RELATED WORK

We classify related work into three categories according to the method used to obtain the target models: (1) natural language, (2) MDA, and (3) algorithm.

The first category of works includes (Yue et al., 2010) and (Maciaszek and Filipe, 2015) which use the informal description of use cases to generate sds. The approach in (Alami et al., 2017) addresses the problem of generating sds from user requirements expressed in Arabic. It is a semi-automated approach that use a natural language processing tool (NLPT). The aforementioned approaches focused only on the user requirements, which do not guarantee that the system supports the business activities. Further, requirement specifications expressed in natural language may contain semantic ambiguities or implicit information, which may lead to different interpretations and consequently to inappropriate sds.

(Khan and Mahmood, 2016) falls into the second category (MDA) of related works. It proposes to transform a use case map into a sd. Further, (De Souza and de Castro Giorno, 2015) defined a set of rules for marking-up use cases, and developed a transformation process that works according to these rules to generate sds. Moreover, (Kang et al., 2010) propose to transform viewpoints of human type in a scenario into objects to actor messages, while viewpoints of non-human type are transformed into object to object messages. These works addresses only software models (use case and sd), while the bpm’s are out of their scope. A recent approach (Nikiforova et al., 2016) propose a transformation method that generates the sd from a new model called two-hemisphere model. But, the new model is not enough rich to represent a complete bpm. Another recent work is proposed by (Khlif et al., 2018). It uses an annotated BPMN model as a starting point to generate a sd. However, some important BPMN elements, such as exception events, signal events, looping activities, etc., were not addressed in this work. Although it may resolve the misalignment problem, the use of a non-standard model may reduce the usability of the approach as it requires specific but non-standard editors to design these models. Further, the use of annotations expressed informally may generate inconsistent sds.

The third category of related works (algorithm) includes (Suchenia et al., 2017) which defined an algorithm to transform a BPMN model into a sd that may be used by business analysts and software engineers to resolve time issues. In addition, (Salami and Ahmed, 2014) and (Nassar et al., 2017) propose a semi-automated algorithm to generate sds from statements of event flows contained within the use case models. Moreover, (Canal et al., 2018) propose an algorithm that supports the integration of sds by measuring the difference between two sequence messages exchanged by objects. Even if defining an algorithm helps to obtain an accurate result, a formal transformation language may enhance this approach.

Despite the various approaches dealing with sd modeling, there are no previous works which address the traceability between source and target models. Moreover, there is no approach proposed to structure the resulting sd according to the MVC pattern. To our knowledge, there are no approaches which generate sds directly from BPMN standard (OMG BPMN, 2013) or without using other UML diagrams. Moreover, only (Khlif et al., 2018) deal with semantics of source models.

3 BUSINESS PROCESS to-Trace

THE SEQUENCE DIAGRAM

We propose a semi-automatic MDA compliant approach called Business Process to-Trace UML Sequence Diagram (BPto-TraceUSD) that aims to create a dynamic view of software models that supports business expectations, and keeps them always aligned even if they evolve. It defines an automatic model transformation from the CIM to the PIM levels by considering the syntax and the semantic perspectives of the source and the target models. The source model at the CIM level is the bpm expressed with BPMN 2.0. The target model at the PIM is a set of sds structured according to the MVC design pattern. We use the standard notation BPMN 2.0 and UML 2.5.1 without any adaptation. Thus, we assume that the reader is familiar with them. To maintain the alignment of the source and target models, and to guarantee that the IS model meets always the business requirements, we define trace links between source- and target elements throughout the transformation process. Figure 1 outlines our approach.
3.1 Transformation of Fragments

In (Bouzidi et al., 2017), we proved that a use case is generated from a canonical fragment $F$ obtained by the decomposition of the BPMN model into fragments. Therefore, the proposed BPMN-to-sequence diagram transformation rules operate on each element of a canonical fragment $F$. Transformation rule $R1$ is defined as follow:

1. Create an interaction (frame) of sd that has the name of $F$.
2. Create a boundary and a control lifelines, which have the same name as $F$ preceded respectively by $b$ and $c$.
3. Create a collection of objects called allObjects as a lifeline that represents all persistent objects that participate in the execution of the fragment $F$.
4. Traceability: Create a trace link stereotyped Trace from the interaction, the control and the boundary lifelines, and allObjects to $F$.

3.2 Transformation of Empty Lanes

With empty lanes/pools, we mean those lanes/pools that do not incorporate child lane sets. They are commonly used to represent internal roles of organizational units (e.g. Manager, Associate), and systems (e.g. enterprise application). An empty lane/pool is semantically equivalent to the actor definition in UML.

1. Create an object Obj in the interaction $I$ representing $F$ (created by $R1$). The type of obj is Actor and its role is set to the name of EMP-L.
2. Traceability: Create a link stereotyped Trace between obj to EMP-L.

3.3 Transformation of Item Aware Elements

Item Aware Elements (IAEs), which are data objects, data stores, data inputs, and data outputs, are required (read) or produced (written) by BPMN tasks to fulfill their business objectives. From a software development viewpoint, many contributions such as (Brdjanin et al., 2019), (Cruz et al., 2015b) and (Cruz et al., 2015a) confirm that IAEs are semantically close domain classes in UML.

1. Create a lifeline $obj$ that has the name as $IAE$ in the sd interaction that represents $F$.
2. Traceability: Create a link stereotyped Trace between obj to IAE.

3.4 Transformation of Tasks

In the sd, a message specifies an information exchanged between object lifelines. In BPMN, tasks meet the UML information exchanged between participants. Hence, we propose to derive a message $msg$ in sd from each automated task Task of a fragment $F$. Figure 2 depicts an example of applying $R1$ and $R2$ to an extract of a business model that contains a fragment called Manage payment performed in an empty Lane called Agent. By applying $R1$ on this fragment, a sd interaction called Manage payment is generated. In this interaction, $R1$ creates a control and a boundary lifelines named respectively cManagePayment bManagePayment, and an object collection called allObjects that represents all persistent objects having an impact on the execution of this fragment. Moreover, $R2$ is applied on the empty Lane Agent to generate an actor called Agent in the sd interaction.
Moreover, it is important to consider task types to extract the sender and receiver of the generated messages. As a user task ensures that a human performer performs Task, the intervention of the participant P that executes Task should be explicitly stated in sd (cf.3). Further, messages generated from a send task should specify the notification of sending a message because a send task is completed only when a message is sent. Likewise, it is necessary to notify the reception of a message in sd when messages are generated from a receive task. However, the execution of a service, a business rule or a script task may be accomplished only by the system without any human intervention. In this case, only a reflexive message is generated from and to the control lifeline ctr that represents the fragment containing Task (cf.4).

Besides, Task may use resources (IAEs) to be executed, and produce outputs (IAEs) as result of its execution. Hence, the most recent BPMN version, BPMN 2.0, allows bpmns to designate persistent data (OMG-BPMN, 2013) by using a data store to indicate that information remains beyond the process life cycle, or after the process execution ends (OMG-BPMN, 2013). In sd, we denote required data of Task as argument of the msgs. To designate the recuperation of persistent data, messages called search() should be created between ctr and the object lifeline allObjects.

Further, we propose to specify the produced data in the signature of msgs (R4.3). Then, creation messages of the produced data (item aware elements) should be generated. If Task is a writer of data store dstr, a message called store(dstr) is generated at the end to explicitly specify the storage process of persistent data (cf.4).

The overall transformation and trace links are specified by R4. It is noted that this rule is invalid on tasks which appear in multiple fragments of the same business model (cf. rule R10 for more details).

R4. For each automated task Task within a fragment F in the empty lane L:

1. If Task is neither a reader nor a writer of an IAE then:
   1.1. If the Task type is User Task or Send Task (cf. Figure 3.3):
      - Create a message Task() from the actor act that represents L to the boundary lifeline that represents F.
      - Create a message Task() from bdr to the control ctr that represents F
   1.2. If the Task type is a receive atsk:
      - Create a reflexive message in ctr
      - Create a message Task() from the ctr to bdr that represents the fragment F containing...

Figure 3: Example of R4.1.1 application on the user task “process payment”:

- Task(created by R1).
  1.3. If the Task type is service, script or business rule, then create a reflexive message Task() from and to ctr (cf. the example in Figure 4.)

2. If Task is a reader of an IAE iae:
   2.1. If the iae type is data store:
      - Create a message called iae_id:search() from the control ctr to the object allObjects (that is created by R1)
      - Create a message called iae_id:search() from the object allObjects to the control ctr
      - Apply steps 1.1, 1.2 or 1.3 (according to the task type), and add iae_id as an argument to the messages created in these steps.

3. If Task is a writer of an IAE prdD, then:
   3.1. Apply step 1.1, 1.2 or 1.3 (according to the task type).
   3.2. Add a creation message from the control ctr to the object prdD.
   3.3. If prdD is a data store, create a message store(prdD) from ctr to allObjects (cf. the example in Figure 4.).
3.5 Transformation of Signal and Exception Events

In BPMN, a signal event is used to send or receive a signal which may be considered as a warning of the system that triggers the user who might be interested to notice and then react to that signal. In the sd, a message kind called asynchSignal is used to specify signals between objects (actor, control, class, boundary). Accordingly, we define R5 to transform each signal event in BPMN into an asynchSignal in sd as follows.

R5. For each signal event SE in the fragment F:
1. Create an asynchSignal message asynM that has the same name as SE from the control- to the boundary lifeline that represent F.
2. Traceability: Create a link stereotyped Trace from asynM to SE.

BPMN also defines error and cancel events, which interrupt the task to which they are attached. In the sd, a break combined fragment may specify an interruption scenario. To specify exception events in the sd, we propose to transform each exception event into a control and a boundary lifelines, which have the name of the exception event. Moreover, we generate messages for triggering and handling the exception scenario.

R6. For each error and cancel event:
1. Create a boundary bdrExcept and a control ctrExcept lifelines, which have the exception event name repessively preceded by the letters b and c.
2. Add two creation messages from the control lifeline FName (that represents the fragment F) to ctrExcept, and bdrExcept.
3. Create a Break combined fragment.
4. Create a message trigger(), in the Break combined fragment, from the control lifeline FName to the control lifeline ctrExcept.
5. Create a reflexive message treatException() from and to the control lifeline ctrExcept.
6. Create a message called display() from the control lifeline ctrExcept to the boundary lifeline bdrExcept.
7. Traceability: Create a link stereotyped Trace from the Break combined fragment, ctrExcept and bdrExcept to the exception event.

Figure 5: Example of R6 application on the exception event materials unavailable.

3.6 Generation of Combined Fragments

Combined fragments may be generated not only from the exception events, but also from a looping task, a redundant task, or gateways.

3.6.1 Generation of a Loop Fragment from a Loop Task

In BPMN, a task with looping behavior means that the task execution may be iterated multiple times. It is also possible to specify a maximum number of iteration. UML defines a loop combined fragment to designate that the elements which belong to this combined fragment must be iterated \( N \) times. Accord-
ingly, we define \( R_8 \) to transform each loop task into a loop fragment as follow;

\[ R_8 \text{. For each loop task } llt; \]

1. Create a combined fragment \( comf \) in the sd interaction that represents the fragment \( F \) incorporating \( lltl.comf \) encloses all messages created from \( llt \)(cf. \( R_4 \))

2. Create an interaction operator \( loop \) of \( comf \)

3. If the number of iteration \( N \) is indicated, st the loop max boundary to \( N \).
4. Traceability: Create a link stereotyped \( Trace \) from the loop combined fragment to \( llt \).

### 3.6.2 Generation of Combined Fragments from Gateways

Overall the transformation of the gateways into combined fragments is specified by the transformation rule \( R_9 \). In BPMN, a decision gateway that is used to create alternative paths within a process flow is semantically equivalent to the \( Alt \) combined fragment that represents a choice of behavior. At most, one of the operands will be chosen. The chosen operand must have an explicit or implicit guard expression that evaluates to \( true \) at this point in the sd interaction.

Unlike the exclusive gateway that allows only an alternative execution, the inclusive gateway executes also parallel paths within a process flow. Indeed, the evaluation to \( true \) of one condition expression of its outgoing paths does not exclude the evaluation of the other condition expressions. To transform this scenario accurately, we propose an \( Alt \) combined fragment that encloses all alternative paths of the gateway \( gt \) as operands, and we create another operand, in which, we create all paths which get out of \( gt \) to represent the execution of all paths. (cf. Figure 6.). On the other hand, the semantics of a parallel gateway allows a parallel execution of tasks. It is equivalent to the \( par \) combined fragment that represents a parallel merge between the behaviors of the operands (\( R_9.3 \)).

The transformation rule \( R_9 \) is defined as follows:

\[ R_9 \text{. For each gateway } gt \text{ in the fragment } F: \]

1. If \( gt \) is an exclusive gateway : Create an \( Alt \) combined fragment in the sd interaction that represents \( F \).
2. If \( gt \) is an inclusive gateway:
   2.1. Create an \( Alt \) combined fragment that has as many operands as outgoing of \( gt \); the guard expression of each operand in the \( Alt \) combined fragment is the label of the corresponding sequence flow going out of \( gt \).

2.2. Create another operand, and a guard. The expression of this guard represents all the outgoing labels of \( gt \).

![Figure 6: Example of R9.2 application.](image)

3. If \( gt \) is a parallel gateway, then create a \( Par \) combined fragment in \( inter \).
4. In the created combined fragment, then for each sequence flow \( SF \) that goes out of \( gt \) create an operand and in the combined fragment that incorporates messages generated from the target ref (except event, activities) of \( SF \).
5. Traceability: Create a link stereotyped \( Trace \) from the combined fragment to the gateway \( gt \).

### 3.6.3 Generation and of Interaction Uses

In BPMN, many participants may execute the same activity. Hence, the same activity may belong to different fragments of the same business model. In sd, the ref combined fragment(interaction use) is used to reference a part of a sd in another one. It represents relationships between separate sds.

\[ R_{10} \text{. For each task } Task \text{ that appears in multiple fragments, we assume that there exists a sd interaction that has the name of } Task \text{ and represents it.} \]

1. Create an interaction use in the sd interaction that represents the fragment \( F \) (that we are handling and that contains \( Task \)). The interaction use has the same name as \( Task \).
2. Create an operand that has the name \( Ref \) in the interaction use.
3. Traceability: Create a trace link stereotyped \( Trace \) from the interaction use to \( Task \).

In the business model, if there is an inclusive or an exclusive gateway between two different fragments, this means that the fragment that comes into the gateway optionally extends the fragment that goes out of this gateway. In sd, the reuse and extension principle is possible by means of the interaction use element.

\[ R_{11} \text{. Let } F_1 \text{ and } F_2 \text{ be two fragments, and there is an inclusive/exclusive gateway } gt \text{ from } F_1 \text{ and } F_2. \]
Suppose that the sd interaction of F2 is a separate sd from the sd interaction of F1 (that we are handling), and is already created.

1. Create an interaction use of F2 in the sd interaction representing F1 to reference the reuse of sd of F2 in sd of F1. The interaction use is called F2Name.

2. Create an operand called ref in the interaction use.

3. Create an Alt combined fragment that encloses the interaction use.

4. Traceability: Create a link stereotyped Trace from the interaction use and the Alt combined fragment to gt.

In Figure 7 there is an exclusive gateway between two fragments Check order validity and Validate order, which are represented as black boxes in this Figure. The fragment Validate order goes out an exclusive gateway and is specified in the interaction Check order validity representing the fragment Check order validity as an interaction use.

**Figure 7: Example of R11 application.**

### 4 ADVANTAGES OF OUR APPROACH

The strong point of our approach is the complete set of transformation rules in comparison to the existing works. Indeed, we consider many BPMN artifacts in our transformation rules (data store, task type, exception and signal events), which are not considered in previous works. The generated diagram can be used as a starting point for the software development process as it significantly shortens the efforts and the time needed to build sds from scratch. Our approach considers not only sd elements but also relationships between interrelated sds generated from the same business model. Further, structuring the obtained sd according to the MVC design pattern is also a strong point of our approach. Moreover, our approach accounts for traceability. The generated trace links may be used to help designers to reduce the analysis time and cost to identify the misalignment sources, and triggers them if a change is applied on these elements. Furthermore, our approach has the merit of accounting for both the semantic and structural aspects of both the BPMN and the UML elements, which we use without any extension. Accordingly, we have explored existing plugins and tools to implement our approach such the BPMN2modeler, UML designer and the Atlas Transformation Language (ATL)(Jouault and Kurtev, 2005) plugins.

### 5 CONCLUSIONS AND PERSPECTIVES

In the current approach, we base on the MDA approach, and we define BPtraceSD, a semi-automatically transformation approach of a BPMN bpm into sds structured according to the MVC design pattern. This transformation allows obtaining aligned models. Throughout the transformation process, we define traceability links between the source and the target model elements, which permit to maintain mapped elements always aligned even if they evolve, reducing this way the analysis time to recognize sources of the misalignment. An ongoing work is oriented towards broadening the traceability management, attempting to integrate the the IS with the business modeling.

### REFERENCES


