Unraveling Unstructured Process Models

Marlon Dumas[1], Luciano García-Bañuelos[1], and Artem Polyvyanyy[2]

[1] Institute of Computer Science, University of Tartu, Estonia
{marlon.dumas,luciano.garcia}@ut.ee
[2] Hasso Plattner Institute at the University of Potsdam, Germany
{Artem.Polyvyanyy@}@hpi.uni-potsdam.de

Abstract. A BPMN model is well-structured if splits and joins are always paired into single-entry-single-exit blocks. Well-structuredness is often a desirable property as it promotes readability and makes models easier to analyze. However, many process models found in practice are not well-structured, and it is not always feasible or even desirable to restrict process modelers to produce only well-structured models. Also, not all processes can be captured as well-structured process models. An alternative to forcing modelers to produce well-structured models, is to automatically transform unstructured models into well-structured ones when needed and possible. This talk reviews existing results on automatic transformation of unstructured process models into structured ones.

1 Introduction

Although BPMN process models may have almost any topology, it is often preferable that they adhere to some structural rules. In this respect, a well-known property of process models is that of *well-structuredness*, meaning that for every node with multiple outgoing arcs (a *split*) there is a corresponding node with multiple incoming arcs (a *join*), such that the set of nodes between the split and the join form a single-entry-single-exit (SESE) region. For example, the process model shown in Fig.1(a) is unstructured because the parallel split gateways do not satisfy the above condition. Fig.1(b) shows an equivalent structured model.

The automatic transformation of unstructured process models into structured ones has been the subject of many R&D efforts. This keynote paper summarizes some of the results of these efforts, including the initial results on an ongoing research effort aiming at developing a complete method for structuring (BPMN) process models. But before discussing how to structure BPMN process models, let us briefly discuss why should we care about doing so.

2 Structured BPMN Models: Why?

There are multiple reasons for wanting to transform unstructured BPMN models into structured ones. Firstly, it has been empirically shown that structured process models are easier to comprehend and less error-prone than unstructured

(a)

(b)
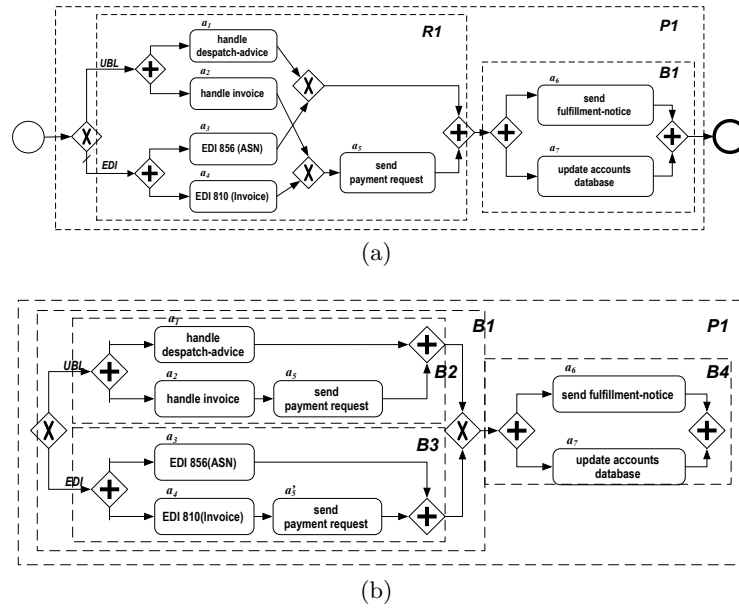
Fig. 1. Unstructured process model and its equivalent structured version

ones [1]. Thus, a transformation from unstructured to structured process model can be used as a refactoring technique to increase understandability. In particular, models generated by process mining techniques are often large, spaghetti-like and difficult to understand and would benefit from being re-structured. Also, mined process models come without layout information, thus requiring automated layout techniques to be applied. Automatic layout of structured process models is easier compared to layout of arbitrarily unstructured models.

Secondly, several existing process model analysis techniques only work for structured models. For example, an efficient method for calculating cycle time and capacity requirements for process models is outlined by Laguna & Marklund [2], but this method is only applicable to well-structured models. Other methods for computing the Quality of Service (QoS) of process models and service orchestrations assume that models are structured [3, 4], and the same applies to methods for analyzing time constraints in process models [5]. By transforming unstructured process models to structured ones, we can extend the applicability of these analysis techniques to cover a larger class of models.

Finally, a transformation from unstructured to structured process models can be used to implement converters from graph-oriented process modeling languages like BPMN to structured process modeling languages such as BPEL [6].

3 A Short History of Structured Process Models

In many ways, *flowcharts* can be seen as predecessors of business process modeling notations such as BPMN. Thus, before discussing how to structure BPMN

models, it is useful to summarize some key results of a large body of research that has tackled the problem of structuring flowcharts. This research, dating mostly from the 70s and 80s, was initially motivated by the debate between proponents of structured programming (based on "while" loops) and those who wanted to stick to programs with GOTO statements. Proponents of structured programming showed that any unstructured flowchart (representing a program with GOTO statements) can be transformed into a structured one. In fact, the title of the present paper is inspired by that of a seminal paper by Oulsnam [7], which presented a classification of unstructured flowchart components and showed how each type of component can be transformed into an equivalent structured one. Oulsnam and others noted that, when structuring loops with multiple exit points as well as overlapping loops, one needs to introduce boolean variables in order to encode parts of the control flow. In any case, we can retain from this work that every unstructured BPMN model composed of tasks, events, exclusive gateways and flows can be transformed into a structured BPMN model.

Another heritage from the research on program structuring is the *Program Structure Tree* (PST). The PST of a program is a tree in which the nodes represent SESE regions in the program's flowchart. The root of the PST represents the entire program. As we go down the PST, we find smaller SESE regions, until we reach individual steps. The SESE region associated to a node contains the SESE regions associated to each of its child nodes, and the SESE regions of these child nodes are disjoint. The concept of PST can be applied to BPMN models because it is unimportant whether the nodes represent tasks, events, exclusive or parallel gateways, or other BPMN nodes. Fig.1(a) shows the SESE regions composing the PST of the BPMN model. The largest region (P1) contains the entire BPMN model. Nested inside it we find two other regions (R1 and B1).

Recent research on structuring business process models has motivated further developments around the concept of PST. Motivated by the problem of transforming unstructured BPMN models into structured ones, Vanhatalo et al. [8] have proposed an improved version of the PST called the RPST (Refined PST). The RPST addresses some technical issues in the PST that we do not discuss because they are irrelevant to this paper. In fact, the RPST of Fig.1(a) and 1(b) are exactly the same as the corresponding PSTs. The difference between RPST and PST is only visible in more specific examples, particularly when some gateways are used both as split and joins.

The RPST also introduces a classification of SESE regions (also called *components*) into four classes: A *trivial* (T) component consists of a single flow arc. A *polygon* (P) represents a sequence of components. A *bond* (B) stands for a set of components that share two common nodes (basically: a split gateway and a join gateway). Finally, any other component that does not fall in these categories is a *rigid* (R) component. In Figures 1(a) and 1(b), the labels of the components reflect their types (e.g. *R1* is a rigid, *P1* is a polygon). For the purposes of this paper, trivial components are unimportant, and therefore we ignore them.

A process model is structured if its RPST does not contain any *rigid* component. For example, Fig.1(b) only contains B and P components. The problem of

structuring BPMN model boils to transforming R components into combinations of P and B components.

The methods for structuring rigids differ depending on the types of gateways in the rigid and whether the rigid contains cycles or not. Accordingly, we classify rigids as follows (cf. Fig.2). A homogeneous rigid contains either only exclusive (*xor*) or only parallel (*and*) gateways. We call these rigids *(homogeneous) and rigids* and *(homogeneous) xor rigids*, respectively. A heterogeneous rigid contains a mixture of *and/xor* gateways. Heterogeneous and homogeneous *xor* rigids are further classified into cyclic or acyclic. We leave *cyclic homogeneous and rigids* out of

Fig. 2. Taxonomy

the discussion, because it can be shown that BPMN models containing such rigids are not *sound* according to the usual definition of soundness [9]. Soundness is a widely-accepted correctness criterion for process models.

One of the earliest studies on the problem of structuring BPMN-like models is that of Kiepuszewski et al. [10]. The authors showed that not all acyclic *and* rigids can be structured by putting forward a counterexample, which essentially boils down to the one in Fig.3. The authors showed that there is no structured (BPMN) model that is equivalent to this one under an equivalence notion known as Fully-Concurrent Bisim-

Fig. 3. Inherently unstructured BPMN model

ulation (FCB). This equivalence notion is arguably the one we seek in the context of transforming unstructured process models into a structured one. We could transform the model in Fig.3 into a trace-equivalent or weakly-bisimilar structured model that only contains exclusive gateways or event-driven decision gateways by enumerating all possible sequential executions of the tasks in the model, but this leads to spaghetti models. If there is parallelism in the original model, we also want the restructured model to have parallelism to the same extent. This is precisely what FCB-equivalence captures.

Liu & Kumar [11] continued this work by outlining a taxonomy of unstructured process model components. Their taxonomy puts in evidence several types of cyclic and acyclic rigids, distinguishing those that are sound and those that are not. The taxonomy includes a class of heterogeneous acyclic rigids called *overlapping structures*, of

Fig. 4. Overlapped structure

which Fig.1(a) is an exemplar. Another example of an overlapped structure is shown in Fig.4. The authors note that such "overlapping structures" are sound and that they have an equivalent structured model, but without defining an automated method for structuring these and other unstructured rigids. Also, the taxonomy is not complete: some unstructured components do not fall into any of
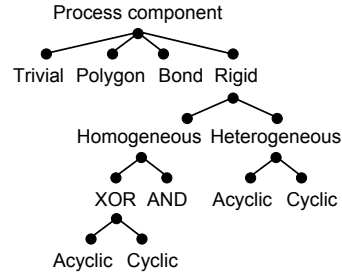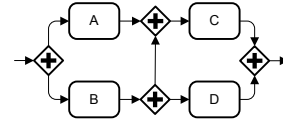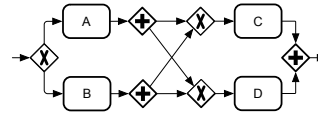
the categories. Later, Hauser et al. [12] outlined another classification of process components using *region trees* – a structure similar to the RPST. The authors showed a method for detecting and refactoring the "overlapped structures" identified by Liu & Kumar. Hauser et al. also observe that all homogeneous rigids are sound. Unsoundness comes from heterogeneous rigids.

We retain from the above that:

– Thanks to the RPST, we can structure a process model if we can structure every rigid component in the process model.
– Any homogeneous *xor* rigid can be structured (cf. GOTO-to-While problem).
– Some homogeneous *and* rigids cannot be transformed into equivalent structured components under FCB-equivalence.
– Heterogeneous rigids are unsound in some cases. When they are sound, it may or may not be possible to transform them into structured components.

4 Towards a Complete Structuring Method

In recent work [13], we presented a method for structuring acyclic BPMN process models. This method is implemented in a tool called BPStruct.

To intuitively explain how BPStruct works, we observe that when transforming an unstructured model into a structured one, we need to duplicate some tasks. For example, in Fig.1(b), task send payment request appears twice, whereas it only appears once in Fig.1(a). If we dig deeper, we observe that this duplication occurs when the unstructured model contains an *xor*-join that is not paired with a unique *xor*-split. This is the case for example in Figure 1(a), which features two *xor*-joins that are not paired with an *xor*-split. In this case, we need to duplicate the tasks that come after such an *xor*-join, and at the same time, push the *xor*-join downstream in the process model, until we get to a point where the *xor*-join is paired with an *xor*-split. In the general case, this "duplicate-and-push" procedure is complicated. But fortunately, this problem has been tackled in the context of Petri nets. Petri net researchers have developed techniques to "unfold" a net so that *xor*-joins are pushed as far as possible downstream. If we push the unfolding to the extreme, we obtain something called an occurrence net, which is basically a net without *xor*-joins.[1] While unfoldings solve the problem of duplicating tasks and getting rid of unstructuredness caused by improperly paired *xor*-splits, they can become quite large if we don't stop unfolding at the right point. Esparza et al. [14] have devised a technique that computes an unfolding that is rather small compared to other possible unfoldings. This is called the *complete prefix unfolding*, and it is the intermediate structure that BPStruct uses for structuring both acyclic and cyclic rigids.

In addition to duplicating tasks when required, an unfolding puts into evidence the fundamental ordering relations between pairs of activities. Specifically, the unfolding allows us to easily determine which pairs of tasks are in a causal

---

[1] For those familiar with Petri nets, an *xor*-join translates into a place with two input arcs. A net in which every place has only one input arc is called an occurrence net.

relation (meaning that the execution of one task causes the execution of another task), which pairs tasks are in a conflict relation (meaning that if one task is executed the other one will not) and which pairs of tasks are in a concurrency relation, meaning that they are both performed, but in any order. In other words, from the unfolding we can directly compute a graph of ordering relations between activities. Each edge in this graph is labelled by one of three types of relations: causality, conflict and concurrency. For example, the ordering relations extracted from the unfolding of the BPMN model in Fig.4 are shown in Fig.5(a). The filled one-way arrow denotes causality, the filled two-way arrow denotes conflict, and the dotted two-way arrow denotes concurrency.
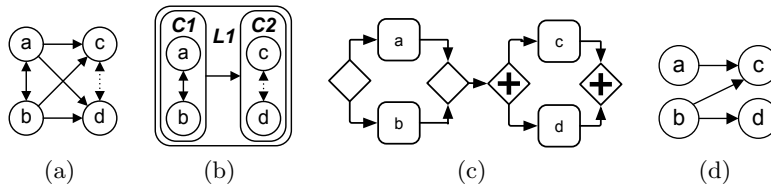
Fig. 5. (a) Ordering relations of Fig.4, (b) modular decomposition, (c) resulting structured component, (d) Ordering relations of Fig.3

The ordering relations capture all the control-flow information in the original model in a compact way. At this stage, all the necessary duplication of tasks has been done, and the model has been cleaned from spurious gateways that sometimes appear in unstructured models. In principle, we should be able to reconstruct a BPMN model by converting these ordering relations into flows and gateways. But how do we ensure that the resulting model is structured? Here is where another theory comes in very handy: that of *modular decomposition* of graphs. The modular decomposition theory enables us to find blocks in an arbitrary graph. BPStruct applies a modular decomposition algorithm on the graph of ordering relations in order to separate it into blocks. For example, the modular decomposition of Fig.5(a) is shown in Fig.5(b). Here we can clearly see that tasks a and b belong to a conditional block, because they are in conflict. Meanwhile, tasks c and d belong to a parallel block because they are in parallel. These two blocks are in a causality relation. This modular decomposition can then be used to synthesize the structured BPMN model shown in Fig.5(c).

We have shown in [13] that an acyclic rigid can be structured if its modular decomposition is composed of linear and complete modules as in the Fig.5(b). These modules basically correspond to P and B components in the RPST. If the ordering relations graph contains a third type of module known as a *primitive*, then the original process model is inherently unstructured. For example, Fig.5(d) shows the ordering relations graph of Fig.3. The modular decomposition of this graph contains only one primitive module. Hence, it cannot be structured.

The method outlined above only works for acyclic rigids because the graph of ordering relation is not a convenient abstraction for models with cycles. In the presence of cycles, we may have causal relations from task A to B and vice-versa,

leading to a chicken-or-egg issue. Still, Petri net unfoldings can also be used to structure cyclic rigids. The idea is to extract acyclic parts from the unfolding, abstract them as black-boxes (that can be structured using the above method), and then construct a cyclic rigid that only contains *xor* gateways. This latter rigid can be structured using GOTO-to-While transformations. A preliminary solution for structuring cyclic rigids is implemented in BPStruct. However, as of the time of writing this paper, the underlying theory is still being worked out.

Also, BPStruct currently does not deal with inclusive and complex gateways, error events, exception flows, attached events and non-interrupting events. As the tool matures, we hope to lift as many of these restrictions as possible.

References

1. Laue, R., Mendling, J.: The Impact of Structuredness on Error Probability of Process Models. In: UNISCON. Volume 5 of LNBIP. (2008) 585–590
2. Laguna, M., Marklund, J.: Business Process Modeling, Simulation, and Design. Prentice Hall (2005)
3. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of service for workflows and web service processes. Web Semantics: Science, Services and Agents on the World Wide Web **1**(3) (2004) 281–308
4. Zeng, L., Benatallah, B., H.H. Ngu, A., Dumas, M., Kalagnanam, J., Chang, H.: Qos-aware middleware for web services composition. IEEE Transactions on Software Engineering **30**(5) (2004) 311–327
5. Combi, C., Posenato, R.: Controllability in Temporal Conceptual Workflow Schemata. In: BPM. Volume 5701 of LNCS. (2009) 64–79
6. Ouyang, C., Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M., Mendling, J.: From business process models to process-oriented software systems. ACM Trans. Softw. Eng. Methodol. **19**(1) (2009)
7. Oulsnam, G.: Unravelling unstructured programs. Comput. J. **25**(3) (1982) 379–387
8. Vanhatalo, J., Völzer, H., Koehler, J.: The Refined Process Structure Tree. Data & Knowledge Engineering **68**(9) (2009) 793–818
9. Kiepuszewski, B., ter Hofstede, A.H.M., van der Aalst, W.M.P.: Fundamentals of Control Flow in Workflows. Acta Inf. **39**(3) (2003) 143–209
10. Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.: On Structured Workflow Modelling. In: CAiSE. Volume 1789 of LNCS. (2000) 431–445
11. Liu, R., Kumar, A.: An Analysis and Taxonomy of Unstructured Workflows. In: BPM. Volume 3649 of LNCS. (2005) 268–284
12. Hauser, R., Friess, M., Küster, J.M., Vanhatalo, J.: An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees. IEEE Transactions on Systems, Man, and Cybernetics, Part C **38**(3) (2008) 347–359
13. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. In: Proc. 8th International Conference on Business Process Management, Hoboken, NJ, USA (September 2010)
14. Esparza, J., Römer, S., Vogler, W.: An Improvement of McMillan's Unfolding Algorithm. FMSD **20**(3) (2002) 285–310