Managing Process Model Complexity via Abstract Syntax Modifications

Marcello La Rosa, Petia Wohed, Jan Mendling, Arthur H.M. ter Hofstede, Hajo A. Reijers and Wil M.P. van der Aalst, *Member, IEEE*

***Abstract*—As a result of the growing adoption of Business Process Management (BPM) technology, different stakeholders need to understand and agree upon the process models that are used to configure BPM systems. However, BPM users have problems dealing with the complexity of such models. Therefore, the challenge is to improve the comprehension of process models. While a substantial amount of literature is devoted to this topic, there is no overview of the various mechanisms that exist to deal with managing complexity in (large) process models. As a result, it is hard to obtain an insight into the degree of support offered for complexity reducing mechanisms by state-of-the-art languages and tools. This paper focuses on complexity reduction mechanisms that affect the abstract syntax of a process model, i.e. the formal structure of process model elements and their interrelationships. These mechanisms are captured as patterns so that they can be described in their most general form, in a language- and tool-independent manner. The paper concludes with a comparative overview of the degree of support for these patterns offered by state-of-the-art languages and tools, and with an evaluation of the patterns from a usability perspective, as perceived by BPM practitioners.**

***Index Terms*—Process model, pattern, complexity, understandability, process metric.**

I. Introduction

Business Process Management (BPM) is increasingly recognized as an overarching approach to improve performance at an operational level. Companies typically utilize BPM technology to reduce costs, save cycle time, and react to changes in a more agile way. While many BPM concepts have been applied to improve business performance in industrial practice, there are still significant challenges, which need to be addressed by BPM research.

One of the challenges in this context relates to complexity management of process models. The capability of a process model to be easily understandable plays an important role for the success of process redesign projects [49]. Business process models in practice often contain dozens of activities and complex behavioral dependencies between them. An increase in size of a business process model beyond certain thresholds, can lead to comprehension problems by its stakeholders. For complex models it becomes difficult to validate them, to maintain them, and to utilize them as means of communication. However, the sheer size of a process model is not the only factor affecting the complexity of a process model. The level of abstraction used to describe the various process tasks, the way the model is laid out, the language used in the labels, etc., are all factors affecting the complexity of a process model. More generally, we relate process model complexity to its understandability, i.e. we refer to the effort required by its users to understand a given process model.

The empirical connection between complexity and process model understanding has been demonstrated in recent publications (e.g. [69], [6], [81]), as much as mechanisms have been proposed to alleviate specific aspects of complexity (e.g. [90], [95], [40]). However, what is lacking is a systematic classification of the various operations that exist for reducing complexity in process models. A comprehensive account of such mechanisms would contribute to improved support for complexity management in process modeling languages, standards and tools. A corresponding classification may be beneficial to research and practice, for instance to initiatives towards process modeling language standardization, to academic and industrial tool evaluation, and to vendors seeking to incorporate innovative features in their tools.

In this paper we address this research gap by compiling a collection of patterns, which define an extensive range of desired capabilities. The approach of capturing design knowledge as patterns has been used in various engineering disciplines including architecture, software engineering, and workflow modeling. The patterns described in this paper capture mechanisms for managing process model complexity. They stem from the literature, process modeling language specifications, and tool implementations.

Essentially, mechanisms for managing complexity of process models can be defined on two different levels [72]: (a) the concrete syntax of a model and (b) the abstract syntax of a model. The *concrete syntax* of a process model deals with its visual appearance, including symbols, colors and position, and is also referred to as *secondary notation* [75]. A collection of patterns for concrete syntax modifications has been presented in [55]. These patterns include mechanisms for arranging the layout, for highlighting parts of the model using enclosure, graphics, or annotations, for representing specific

Manuscript received May X, 2011; revised X X, 2011.

M. La Rosa and A.H.M. ter Hofstede are with the Queensland University of Technology. e-mail: see http://marcellolarosa.com and http://yawlfoundation.org/∼arthur. Ter Hofstede is also with Eindhoven University of Technology.

P. Wohed is with Stockholm University, Sweden. e-mail: see http://people.dsv.su.se/∼petia.

J. Mendling is with Humboldt University of Berlin, Germany. e-mail: see http://mendling.com.

H.A. Reijers and W.M.P. van der Aalst are with Eindhoven University of Technology, The Netherlands. e-mail: see http://www.reijers.com and http://www.vdaalst.com. Van der Aalst is also with Queensland University of Technology.

concepts explicitly or in an alternative way, and for providing naming guidance. The *abstract syntax* of a process model relates to the formal structure of process model elements and their interrelationships. If we draw a parallel with the field of Linguistics [20], the abstract syntax of a process model corresponds to the *deep structure* of a sentence, i.e. its underlying syntactic structure, as opposed to the concrete syntax of a process model, which corresponds to the *surface form* of a sentence, i.e. its presentation. The patterns presented in this paper work on the abstract syntax and complement the patterns collection for concrete syntax modifications presented in [55]. They relate to model operations such as transforming a model into a set of modules or omitting elements to provide a more abstract view on the process. Clearly, a change to the abstract syntax of a process model can be expected to affect the model's visual appearance as well.

In this paper we aim for a language-independent description of abstract syntax related patterns. Each pattern is accompanied with a discussion of its intended effect on model complexity and of different realizations to clarify its scope and to demonstrate its relevance. The pattern description is complemented by an overview of its support in tools and modeling languages, which sheds light on its comparative strengths and weaknesses. Additionally, we evaluate each of the patterns from a usability perspective as perceived by BPM practitioners.

The paper is structured accordingly. Section II describes and justifies the methodology, which we used to identify the patterns. Section III presents the collection of patterns in detail. Section IV evaluates the pattern support of various process modeling languages and tools. Section V presents the results of a usability evaluation with BPM practitioners. Section VI discusses related work while Section VII concludes the paper.

II. Methodology

In this paper we identify *patterns to reduce the model complexity* on the level of its *abstract syntax.* The original idea to organize design knowledge in terms of patterns stems from the architect Christopher Alexander, who collected rules and diagrams describing methods for constructing buildings. In this context, a pattern provides a generic solution for a recurring class of problems. The general idea of design patterns has been introduced to information systems engineering by Gamma, Helm, Johnson and Vlissides, who describe a set of recurring problems and solutions for object-oriented software design. The design patterns by Gamma et al. inspired many patterns initiatives in computer science, including the Workflow Patterns Initiative[1].

The patterns for abstract syntax modifications, which are defined in this paper, have been collected through a series of steps. The starting point was an extensive analysis of the BPM literature. We looked at the last ten years' table of contents of all the major conferences in this field, e.g. BPM, CAiSE, CoopIS, ICSOC, and major journals such as DKE, IEEE TSE, ACM TOSEM, IS, ICJIS. In particular, the selection of forums was determined on the basis of the ERA 2010 classification[2] and the journals' impact factor according to Web of Knowledge.[3] In this step, we also looked at all specifications and standard proposals that are endorsed by the major standardization bodies such as OASIS, OMG, W3C and WfMC. Subsequently, we inspected mainstream commercial BPM tools and the operations they offer for modifying the abstract syntax. The initial set of patterns was first discussed among three of the authors, and then validated with the remaining authors. The patterns derived in this way were then presented to a panel of experts which resulted in a further identification of two additional patterns. This extended set was evaluated with respect to their support by reported research approaches, languages and tools with the goal to distinguish those which are most frequently used. We decided to focus on those patterns that are supported by at least five research approaches/languages/tools. This resulted in a final set of 12 patterns. Finally, we evaluated this final set on its ease of use and usefulness by interviewing a group of 22 BPM professionals.

Each of the twelve patterns in the final set is illustrated in this paper by the use of BPMN (Business Process Model and Notation), an industry standard for modeling business processes. Figure 1 shows the notational elements of BPMN which are used in this paper. The example models are intentionally kept simple such that they can be understood without deep knowledge of this standard.

When referring to a model, we use the term *model element* to indicate any element which has a corresponding *concept* in the language's meta-model. Model elements can be *nodes* (e.g. a task, a gateway or a business object) or *arcs*. We also use the term *fragment* to indicate a set of model elements in a process model that are organized via control-flow relations, and the term *module* to indicate a process model which is part of a larger business process (e.g. a subprocess or the portion of model enclosed by a lane).

We use a fixed format to document the twelve patterns and to discuss their support in languages, tools, and in the literature. This format contains: (a) description, (b) purpose, (c) example, (d) metric, (e) rational and (f) realization of a pattern. The purpose describes the use case in which the pattern is commonly used, while the rationale provides a justification grounded in the literature, as to why a given pattern reduces the complexity of the process model it is applied to. Moreover, we relate each pattern that operates on a process model to a desired improvement of a structural metric. In fact, it has been shown that certain structural metrics can be related to process model understandability [69]. For example, intuitively, the smaller the size of a model, the easier it is to understand it. We discuss the following metrics in this context:

- *module size*, the number of nodes in a module;
- *model size*, the summed size of all modules in a process model [65];
- *repository size*, the summed size of all models in a process model repository;
- *models*, the number of models in a process model repos-

[1]http://www.workflowpatterns.com

[2]http://www.arc.gov.au/era

[3]http://apps.webofknowledge.com

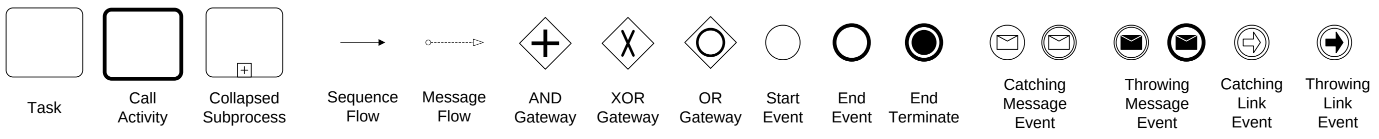


Fig. 1. BPMN 2.0 concepts used in this paper.

itory [95];

- *depth*, the number of modular levels appearing in a process model [95];
- *diameter*, the longest path from a start to an end element in a process model [65];
- *average gateway degree*, the number of nodes a gateway in a specific process model is on average connected to [65];
- *structuredness*, the restructuring ratio of an unstructured model to a block-structured variant of it [58];[4]
- *modules overhead*, the ratio between *modules* and *model size*;
- *fan-in*, the average number of references to a module [57];
- *different modeling concepts*, the number of different modeling concepts used in a process model.

For example, Fig. 5 shows a BPMN model consisting of three levels with one root module and three subprocess modules. This model has the following metrics: depth=3, modules=4, model size=25, average gateway degree=3, different modeling concepts=10 (arc, start event, end event, atomic task, subprocess, AND-split, AND-join, XOR-split, XOR-join). Similarly, we can infer metrics for specific modules. For example the root module has module size=9 and diameter=8.

III. Patterns for Abstract Syntax Modification

From an analysis of relevant BPM languages, tools and approaches, we identified twelve patterns operating on the abstract syntax of a process model and classified them according to the hierarchy in Fig. 2. The patterns are categorized into two main groups: Model Modification (including patterns that directly modify a process model or set thereof) and Meta-model Modification (including patterns that have more profound changes because they affect the underlying process language). Model modification includes *Behavior Abstraction* and *Behavior Neutral* patterns. Behavior Abstraction includes those patterns that operate on a single model and provide a more abstract one as a result. *Omission* simply skips elements of the original model, while *Collapse* aggregates a set of elements into a single, semantically more abstract element. Behavior Neutral patterns preserve the behavior being described in a single model or in a set of models, but organize this behavior in a different representation. *Restructuring* refers to transformations that reorganize the control flow of a process model in a more understandable way, either in terms of *Block-Structuring* or *Compacting* the process model, while *Duplication* introduces model element redundancy in order to simplify its structure. Three *Modularization* patterns, *Vertical*, *Horizontal* and *Orthogonal*, capture different ways in which a process model is decomposed into modules. Two *Integration* patterns, namely *Composition* and *Merging*, refer to features for combining information which is scattered across different modules or models into a single one. While *Composition* uses references among different modules or models to achieve integration, *Merging* relies on an overlap of elements. Finally, *Meta-model Modifications* involve *Restriction* and *Extension*.

Pattern 1 (*Block-Structuring*)

Description This pattern refers to methods to structure a process model in blocks.
Purpose To improve understandability and maintenance through a simpler process model structure.
Example Fig. 3a shows an unstructured model where split gateways are not matched by corresponding join gateways. Fig. 3b shows a behavior-equivalent model which is structured.
Metrics Increases *structuredness* of a process model.
Rationale Structured models are easier to understand [69], [70] and less error-prone [65], [58] than unstructured models.
Realization The problem of structuring process models has been extensively analyzed in the literature both from an empirical and from a theoretical point of view. Laue and Mendling [58] report the results of a study showing that structured models are less error-prone than unstructured equivalent models. Mendling et al. [70] propose seven guidelines to model easily-understandable process models. One of these guidelines is to model processes as structured as possible, which was ranked by a pool of practitioners as the guideline with the highest relative potential for improving process model understandability. Kiepuszewski et al. [48] provide a first attempt to classifying unstructured process models that can be transformed to structured equivalents, and show that structured models are less expressive than unstructured ones. Thus, unstructured model fragments cannot always be replaced with structured fragments that are behavior-equivalent. Liu and Kumar [63] present an alternative taxonomy of unstructured process models which also covers acyclic models, and sketch a method to transform some types of unstructured models into structured trace-equivalent alternatives. Different transformation strategies are also illustrated in [67]. In a similar vein, [43] proposes a classification of (unstructured) process models using region trees, and provides a set of rules to transform certain types of unstructured regions into structured equivalents. An alternative classification of process models based on the RPST decomposition [94] (a refinement of [43]) is proposed in [77], showing how unstructured regions can be generated by composing structured process fragments. A method specifically tailored to untangling unstructured cyclic models and transforming them into structured BPEL models is

[4]a block-structured model is one where each split element has a corresponding join element of the same type, and split-join pairs are properly nested

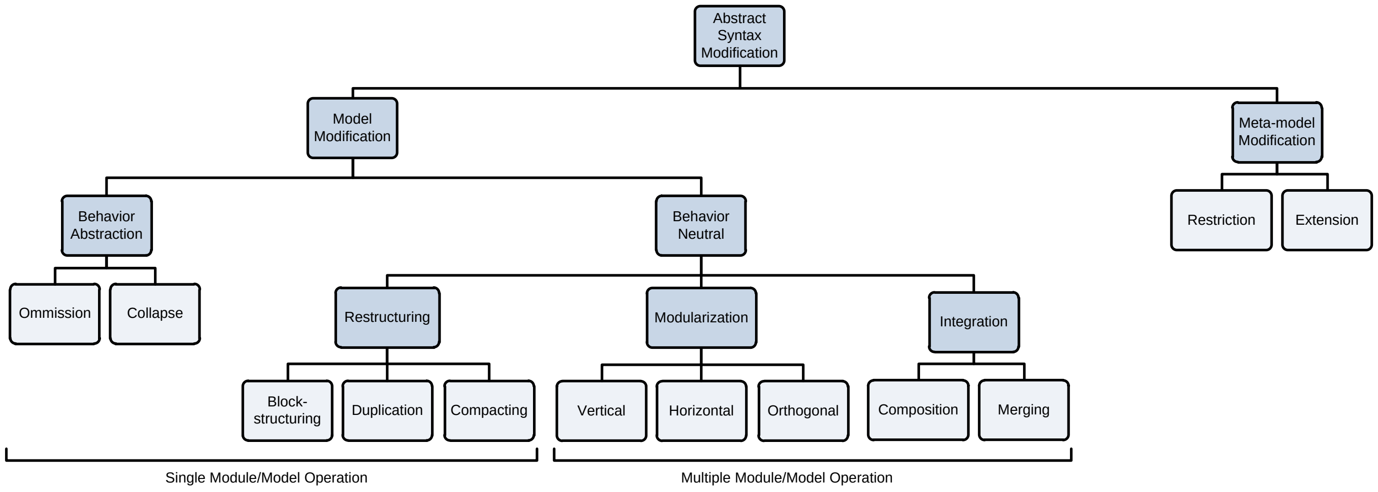


Fig. 2. Patterns for abstract syntax modification.

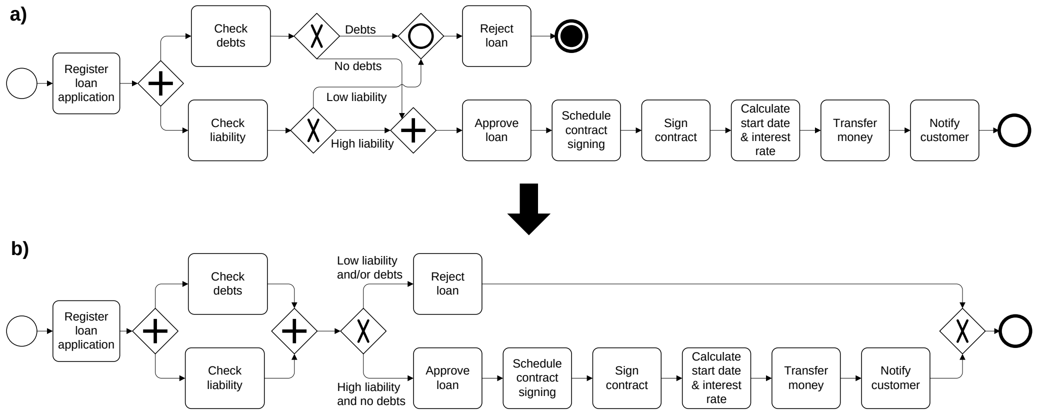


Fig. 3. a) An unstructured BPMN model. b) A behavior-equivalent model which is structured.

presented in [50], [44]. A complete characterization of unstructured acyclic process models that cannot be transformed into block-structured ones is presented in [76], [29]. This method is based on the RPST decomposition and is accompanied by a tool to automatically structure acyclic process models that are not in this class. Finally, Weber et al. [95] propose a set of refactoring mechanisms for process models wherein they devise (but do not operationalize) a mechanism to replace a process fragment with a trace-equivalent fragment having simpler structure.

Pattern 2 (*Duplication*)

Description Duplication (aka Cloning) introduces controlled redundancy in a process model by repeating model elements. Two model elements are duplicated if they point to the same conceptual definition.

Purpose To improve understandability and maintenance through a simpler process model structure. Often required to block-structure an unstructured process model.

Example Fig. 4 shows a behavior-equivalent representation of the model in Fig. 3a after repeating task "Reject loan". This alternative representation of the same model does not have any crossing lines nor OR-join gateways.

Metrics Despite increasing *model size*, this pattern typically also increases *structuredness*.

Rationale Less cluttered and more structured process models are easier to comprehend [69], [70] and less error-prone [65], [58].

Realization Process modeling languages generally provide the possibility of creating duplicate model elements. For example, in YAWL [46] tasks can be duplicated by associating multiple tasks to the same Task Decomposition, so that these tasks have same operationalization. Still, the two duplicates can be allocated to different resources for their execution. In BPMN 2.0 two Service Tasks can be associated with the same Service Reference and Service Endpoint, thus pointing to the same Web service implementation. To duplicate other types of tasks, such as Manual and User Tasks, BPMN 2.0 provides the Global Task construct, which is a reusable atomic Task that can be called from within any process model via an atomic Call Activity. Similar to Service Tasks, a Call Activity can override the Resources attribute of the invoked Global Task. Global Tasks replace Reference Tasks in BPMN 1.2, which supported duplication within the same process model only. In eEPCs, multiple elements (e.g. multiple Events or Functions) can point to the same definition, thus allowing the repeated use of the same concept. Since different Functions may be preceded or succeeded by the same Events, it may be useful to duplicate all preceding/subsequent Events of a given Function so as to line them up close to the corresponding Function's symbol, instead of having crossing arcs. In the Activity Diagrams (ADs) of UML 2.3, duplication is supported by the CallBehaviorAction, which is an Action that points to an Activity Definition. The mechanisms offered by these languages are generally supported by the respective tool implementations. For example, in ARIS one can navigate from the Occurrence of a Definition in a process model back to the Definition Object stored in the ARIS Database, and when creating Occurrence copies of a Definition, all attributes will also be copied. In Tibco Business Studio a BPMN 1.2 Task can be identified as a Reference Task and assigned to another task within the same model, while a function "Reveal" allows highlighting the task being referenced by a Reference Task. In Signavio a task can be set as Call Activity, but it cannot be linked to a global task. In the literature, duplication is used to block-structure process models. For instance, the block-structuring approach in [76] uses unfolding techniques from Petri net theory to construct an occurrence net [31]. In an occurrence net, each XOR-join is unfolded by repeating the subsequent net. The result is a structured, but often much bigger model. The possibility to have multiple tasks carrying identical labels may increase the expressiveness of the language. A classical example can be found in the Theory of Regions for Petri nets. To construct a bisimilar Petri net for a transition system, it may be necessary to do "label splitting" (multiple transitions referring to the same task) [21].

Pattern 3 (*Compacting*)

Description This pattern refers to methods to remove redundant elements in a process model without loss of process behavior. Elements that can be removed include redundant transitive arcs, superfluous gateways or duplicated tasks. This pattern may revert the effects of Duplication.
Example Fig. 3b can also be obtained by compacting the two occurrences of task Reject loan and the two XOR-split gateways from the model in Fig. 4.
Purpose To reduce model size and thus improve the overall model representation, especially in large process models or models that have undergone a number of updates.
Metrics Reduces *model size*.
Rationale Reducing model size positively affects model understanding [69].
Realization Methods for eliminating superfluous elements have been defined for Petri nets. The work by Berthelot defines so-called *implicit places* [13]. An implicit place does not contribute to the overall behavior captured in a process model, since it always has a token when other places have tokens too. Thus, it can be deleted without impacting the model behavior. A related reduction rule is defined by Desel and Esparza [26] with the notion of *transitively dependent arc*. While this concept is defined for verification purposes, it can also be used to eliminate superfluous arcs from a process model without an effect on the behavior. The concept of transitive reduction is also exploited in an approach to synthesize process models from behavioral profiles such that transitive order dependencies are deleted [89]. Furthermore, [27] proposes an approach to merge subsequent connectors of the same type. The same idea is utilized in [66].

Pattern 4 (*Vertical Modularization*)

Description This pattern captures features to decompose a model into vertical modules, i.e. subprocesses, according to a hierarchical structure.
Purpose To increase understandability of large process models by "hiding" process details into sub-levels. To decrease redundancy and foster reuse by referring to a subprocess from several places within the same process model or from different process models in a repository. The maintenance burden of a process model (repository) is also decreased, as a change to a subprocess needs only be performed in one place.
Example Fig. 5 shows the vertical modularization of the model in Fig. 3b in three hierarchical levels.
Metrics Increases *depth* and *models*, and reduces *module size* by replacing a model fragment with a single element referring to that module. It may increase the overall *model size* if the *fan-in* of the module being introduced is one, because for each extracted module a node needs to be inserted in the root model to refer to that module. *Modules overhead* needs to be controlled (i.e. it is pointless to factor out single model elements or very small process fragments to subprocesses).
Rationale Hiding process details into sub-levels increases understanding [83] and fosters reuse [4], [60]. Smaller models are less error-prone [68] and easier to maintain [70] than larger ones. Reducing redundancy helps to avoid inconsistencies, since redundant fragments may be edited by different users simultaneously [93].
Realization Most languages offer modeling constructs to capture vertical modularization. UML ADs provide the notion of an Activity to encapsulate subprocesses, as opposed to atomic Actions. In eEPCs, models of different types can be assigned to different element types. For example, a Function can be decomposed into an eEPC capturing a subprocess, an Event can be decomposed into an Event diagram capturing more detailed sub-events, and an Organizational unit can be decomposed into an Organizational chart. In BPMN 2.0 a Task can be decomposed into a Collapsed Subprocess (the subprocess is available as a separate module) or as an Expanded Subprocesses (the subprocess is represented within the Task box itself). BPMN requires a different construct (the Collapsed Call Activity Task) if the subprocess to be invoked is external (i.e. it is a standalone process model), rather than an internal module. In YAWL, BPEL and Protos subprocesses can only be invoked internally via a Compound Task (YAWL), Scope Activity (BPEL) or Sub-process element (Protos). For correctness purposes, YAWL and BPEL only accept single-entry single-exit subprocesses. Tools offer different degrees of support for this pattern. Some simply allow the specification of pointers to subprocesses (e.g. the YAWL Editor), others allow process hierarchies to be traversed (e.g. Oracle JDeveloper, Signavio), and some also offer a tree-like view of the hierarchy and allow sub-processes to be automatically created from a set of model elements (e.g. Protos and ARIS). General recommendations are also available for modularizing a process model vertically:

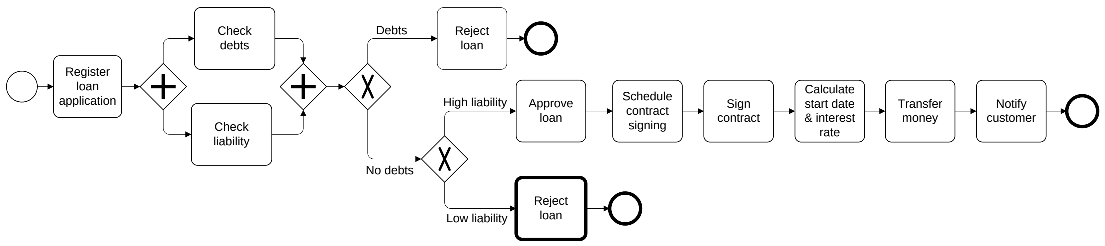


Fig. 4. A behavior-equivalent representation of the model in Fig. 3a after repeating Task "Reject loan" (the top "Reject loan" task is the Global Task while the bottom one is the Call Activity referring to that task).

[59], [10] suggest to encapsulate only single-entry single-exit (SESE) fragments in subprocesses and [94] provides a method for automatically detecting SESE fragments based on the RPST decomposition. Moreover, [47], [86] suggest to introduce subprocesses in models with 5–15, resp., 5–7 nodes, while [70] suggests to decompose models with more than 50 nodes based on empirical evidence. The positive effects of vertical modularization on process model understanding have been studied in [83]. A mechanism to automatically extract cloned fragments in large process model repositories and capture them as globally available subprocesses, is envisaged in [95], while an algorithm to detect such clones is defined and implemented in [93].

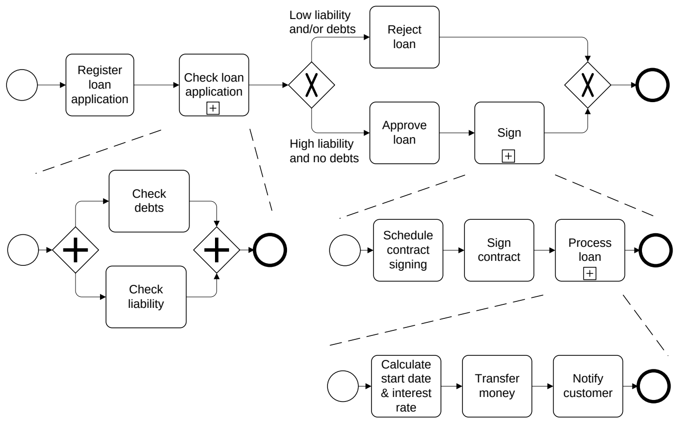

Low liability and/or debts
Register loan application
Check loan application
Reject loan
Approve loan
Sign
High liability and no debts
Check debts
Check liability
Schedule contract signing
Sign contract
Process loan
Calculate start date & interest rate
Transfer money
Notify customer

Fig. 5. Vertical Modularization: the model in Fig. 3b has been decomposed into three levels.

Pattern 5 (*Horizontal Modularization*)

Description This pattern captures features to partition a process model into peer modules.

Purpose To increase maintainability by breaking down a process model into smaller and more easily manageable parts, the ownership of which may be assigned to different users. Hence, to facilitate collaboration. To reduce clutter in those models where long or crossing edges cannot be avoided. To foster reuse of modules within the same process model or across different process models.

Example Fig. 6a shows how the model in Fig. 3b has been decomposed into two *parallel* modules: "Home Loan" and "Sign", which communicate via message exchange. Process model "Sign" can now also be used by a third model, namely "Student Loan", thus increasing reusability. The model in Fig. 3b actually focuses on the loan registration part of a larger, end-to-end process model for handling loans, where "Register Loan" is preceded by "Merchandize" and followed by "Handle Loan". This model can thus also be decomposed into multiple *sequential* modules, e.g. one per logical part. Fig. 6b shows the "Register Loan" module which is connected to the other modules via Link Events. Moreover, the "Sign Loan" part has been extracted from "Register Loan" and modeled separately.

Metrics Reduces *module size* (by replacing a model fragment with a single element referring to that module), and increases *models*. It may also increase *model size* if the *fan-in* of the module being introduced is one.

Rationale Reducing model size positively affects model understanding [69]. Decomposing a process model into modules fosters reuse of shared modules [4], [60].

Realization UML ADs, BPMN and BPEL offer features to represent parallel process modules via message exchange. UML ADs offers the concept of Partitions to divide a process model in peer modules. BPMN provides Pools to create peer modules and Message Flows to connect a source node in one Pool to a target node in another Pool. BPEL supports synchronous and asynchronous inter-process communication via the Invoke, Receive and Reply activities, which link to a BPEL process counterpart via a PartnerLink. BPMN also offers the Signal event to synchronize the control flow in different modules. The difference between these languages is that only in BPEL the modules are completely independent, while in UML ADs and BPMN reuse of a module is limited within the same process model. Other languages such as eEPCs and Protos provide constructs to capture external connections (i.e. Events in eEPCs and Triggers in Protos). However, while in eEPCs two process modules can be formally linked by using two occurrence copies of the same Event definition, this is not possible in Protos. Sequential modules can be obtained in BPMN via the Link Event which interrupts the flow after a source node and before a target node. The corresponding construct in UML ADs is the Activity Edge Connector, in eEPCs it is the ProcessInterface while in Protos it is the Off-Page Connector. Generally, the above concepts are supported by tools. For example, both Signavio and Protos allow users to jump from two corresponding Link Events, resp., Off-Page Connectors. In the literature, [28] defines a mechanism to partition large BPMN models sequentially by cutting those edges with numerous bends and crossings and inserting two pointers at the two ends of each cut edge. The objective is to obtain subgraphs of nearly equal size while keeping the number of cut edges low. Worklets [8] are a mechanism to deal with highly-flexible process models. A Worklet is a process

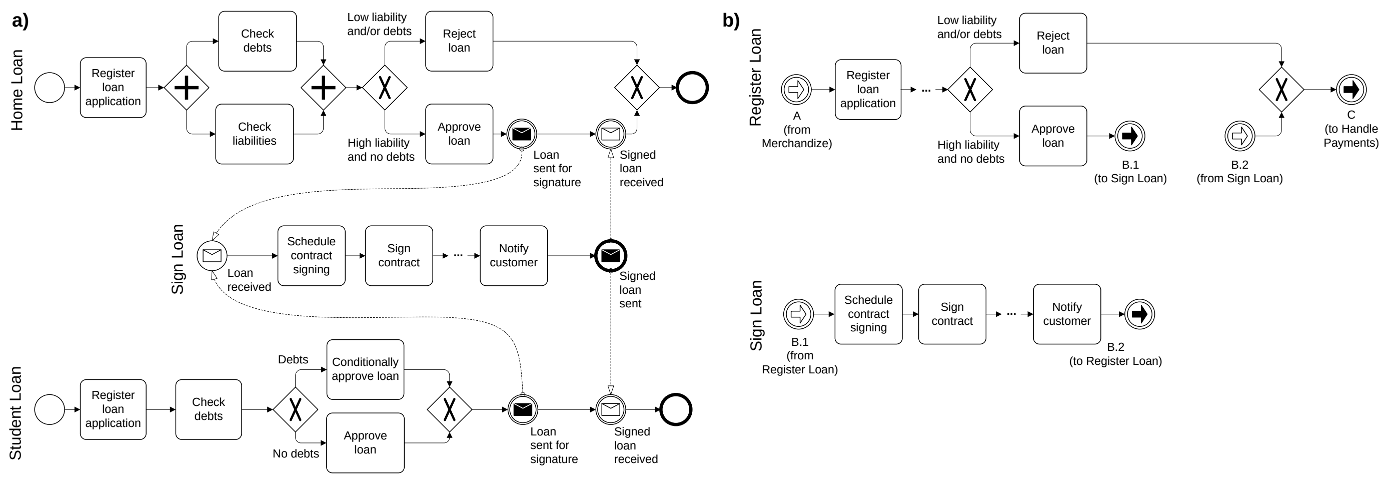


Fig. 6. Horizontal Modularization: (a) the model in Fig. 3b has been partitioned in two parallel modules via message exchange ("Home Loan" and "Sign"), and (b) the end-to-end process model for handling loans has been partitioned in four sequential modules two of which are shown.

fragment capturing a specific process scenario, i.e. variant within a given process model. Various Worklets can exist for a given variation point and one of them will be activated at run-time based on the evaluation of a set of ripple-down rules. This approach has been implemented in the YAWL system where Worklets can be defined to tasks via the Rules Editor, and instantiated via the Worklet Service. A similar mechanism is provided in [32], where a large, unstructured Petri net can be represented as a set of partially-ordered, acyclic process fragments called Oclets. Each Oclet captures a specific scenario of a given process together with information indicating when the scenario occurs. This idea of capturing a process model as a set of partially-ordered runs is also put forward in [25], [12]. Proclets [5] are lightweight Workflow nets which interact via message exchange and have knowledge of previous interactions. A process model is a set of Proclets that are instantiated and coupled according to an underlying business object model. This approach is introduced to deal with highly-interactive business processes.

Pattern 6 (*Orthogonal Modularization*)

Description This pattern captures features to decompose a process model along the crosscutting concerns of the modeling domain, which are scattered across several model elements or modules. Examples of concerns are security, privacy and exception handling.

Purpose To enable a separation of concerns and distribution of responsibilities. To facilitate maintenance of individual, concern-specific process models.

Example Fig. 7a shows a more detailed representation of the "Process loan" subprocess from Fig. 5, which includes further tasks to deal with the security aspect related to loan processing. Fig. 7b shows the orthogonal modularization of this process across its security aspect. The tasks related to this aspect have been extracted and captured in a separate model which is accessible by the first two activities of "Process loan". These activities are instantiated at the *join point* in the "Security aspect" model.

Metrics Increases *models*, as "concern-specific" process models are introduced. Decreases the *model size* when a concern is associated with several process models, as its specification is defined in a single place.

Rationale Reducing model size positively affects model understanding [70], [73]. Decomposing a process model into modules fosters reuse of shared modules [4], [60].

Realization Aspect-oriented modularization [18] is an application of this pattern. This approach is inspired by aspect-oriented programming and defines extensions to workflow languages in general [17], BPEL [18] and BPMN [16], to represent non-functional aspects of a business process such as security and logging, as separate, reusable modules. The notions of *joint point*, *pointcut* and *advice* used in aspect oriented programming, are adopted in this context. For instance, Fig. 7 shows a BPMN model extended with such notions. During process execution, when the task associated with a pointcut is reached, the module (i.e. advice) specified for the corresponding aspect is triggered. Its join point task is then substituted with the task triggering that advice. An advice can be executed before or after a join point, although more advanced mechanisms can be envisaged, e.g. in parallel to the join point. Various aspects can be associated with the same activity and a concern may include more than one module, e.g. different levels of security could be captured via different modules. Orthogonal modularization is also applied to deal with exceptions. The basic idea is to avoid unnecessary cluttering of a process model by extrapolating those procedures to handle possible exceptions from the main process flow. In BPMN 2.0, exceptions can be handled outside the normal process flow via an Event Subprocess. These subprocesses are triggered by the occurrence of a given event, and can either run in parallel to the main process flow (Non-Interrupting Event Subprocess) or in alternative to the main flow (Interrupting Event Subprocess). In UML ADs exception routines are specified in separate Exception Handler activities which are connected to the main activities where exceptions may occur via an Interrupt Flow arc. Similarly, in BPEL Fault Handlers can be attached to a Scope activity or to a global process model. These features are generally provided by the respective tools. [7] proposes to capture exception handling routines via the Exlet notion. An

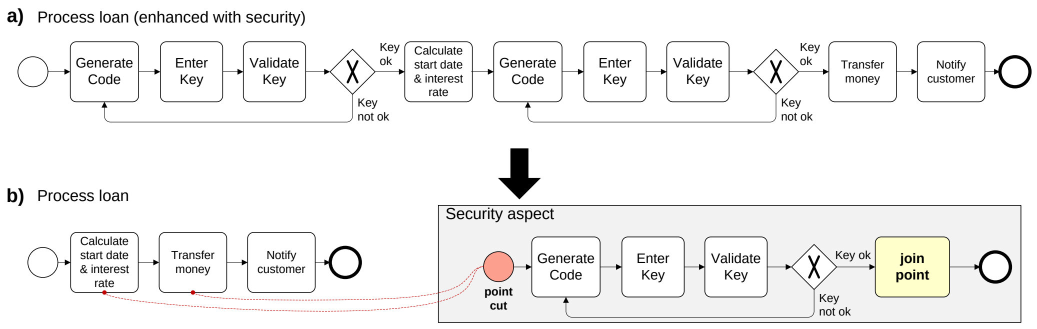


Fig. 7. Orthogonal Modularization: (a) extracting the security aspect from the "Process loan" subprocess in Fig. 5 and (b) capturing it in a separate module.

Exlet specifies how an exception influences a running task, process instance, or all instances of a process model, and can be assigned a compensating routine in the form of a Worklet to be performed when the given exception occurs. This approach is supported in the context of the YAWL language by the Rules Editor and the Exception Service available in the YAWL system.

Pattern 7 (*Composition*)

Description This pattern describes features for constructing a consolidated process model from different disjoint modules. Modules may be organized vertically in a hierarchy (in this case Composition will flatten the hierarchy), or horizontally, or orthogonally (where each module represents a domain-specific concern). As such, this pattern may reverse the effects of Modularization.

Purpose To consolidate a family of interrelated modules into a single process model. The effect may be increased maintainability and understandability when there are too many or too small modules, e.g. due to subsequent modifications. It is useful when the process depth is too high or when a module contains only one or two nodes. The advantages are situation-dependent (e.g. flattening a model may be useful when presenting it to a business user).

Example Fig. 3b can be obtained by composing either the vertical modules in Fig. 5, or the horizontal modules "Home Loan" and "Sign Loan" in Fig. 6a.

Metrics Decreases *modules* and their *fan-in*, but increases *model size*. Reduces *depth* and *modules overhead* when applied to a process hierarchy.

Rationale Extracting information from fewer modules or from a single model may be easier than dealing with many small modules [83].

Realization The concept of this pattern has been defined as a refactoring mechanism in [95]. Here the authors recommend to inline process fragments when the induced navigation overhead of a modularization is not justified. A corresponding change pattern is specified in [96]. Composing vertical modules is also defined as a transformation strategy to convert BPEL models to BPMN [67], and adapted to OWL-S in [35]. [53] proposes a method for composing state machines describing the lifecycle of independent objects involved in a business process, into a single UML AD capturing the overall process. This work assumes that the lifecycles to be composed are disjoint and consistent. The concept of composition is also widely supported by business process modeling tools. For instance, ARIS provides the *model generator* command to construct a flattened model from a subprocess hierarchy, or to compose sequential modules into one model. A similar feature, called *explode* is available in the Protos Editor.

Pattern 8 (*Merging*)

Description This pattern describes features for merging similar process models based on their commonalities, i.e. their identical model elements. The result is a single merged model.

Purpose To consolidate a family of similar process models (i.e. process *variants*) into a single "reference" process model, by removing redundancies.

Example Fig. 8 shows the model resulting from merging the "Home Loan" and "Student Loan" modules in Fig. 6a.

Metrics Decreases *models*. It also decreases *repository size* since redundancies among variants are minimized. However, the resulting reference model may be more complex.

Rationale Consolidating a process family improves its understandability, since a single model is kept instead of many variants. It also improves the maintainability of a repository, given that the total number of models is reduced [95], [54]. Removing redundancies helps decrease the risk of inconsistencies due to independent changes to the repository [95].

Realization [92] proposes, but does not fully automate, an approach for merging block-structured Workflow nets by using a set of merge patterns between mapped tasks. [71] proposes a merging operator that takes two different EPCs each representing a specific process view, as well as a mapping of their correspondences, and produces a merged EPC. Since the models to be merged represent partial views of the same process, the resulting merged model allows the various views to be executed in parallel. [37] merges pairs of EPCs by performing the union of each of the two models. The nodes to be merged must have identical labels and the input EPCs must have single start and end events and no connector chains. This approach is operationalized as a ProM plug-in. [54] presents an algorithm to merge process variants into a configurable process model [84] where divergencies between common and variant-specific fragments are marked via configurable connectors. The mapping between two input variants relies on the similarity of their nodes' labels (including resource and object nodes) and of their graph structures. The algorithm is language-independent (it has been validated on BPMN and eEPC models) and has been implemented on the AProMoRe

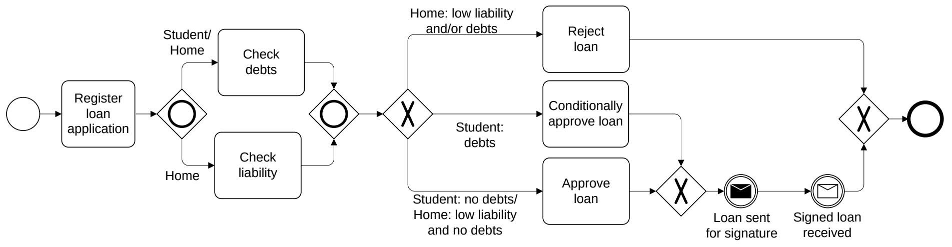

Register loan application
Student/ Home
Check debts
Home
Check liability
Home: low liability and/or debts
Reject loan
Conditionally approve loan
Student: debts
Student: no debts/ Home: low liability and no debts
Approve loan
Loan sent for signature
Signed loan received

Fig. 8. The models “Home Loan” and “Student Loan” from Fig. 6a have been merged in a single *reference* process model for registering loans.

process model repository. Tools deriving from research initiatives such as Greta [33] and VipTool [12] allow the automatic synthesis of a Petri net from a set of partially-ordered Petri nets (called Oclets in Greta). This is closely related to the topic of process mining where example behavior is folded within a single model [1]. [61] proposes to construct a single *generic* model from a set of model variants, such that the sum of the *change distances* between each variant and the generic model is minimal. This approach only works for block-structured process models with AND and XOR blocks and has been implemented on the MiniADEPT tool. The problem of maintaining merged EPCs has been explored in [82]. Here the authors propose an alternative (mostly manual) method which is applicable if the need for maintaining merged models is identified before the actual process modeling effort is started. Finally, [51] describes the requirements of a process merging tool for version conflict resolution. The aim is to assist modelers in resolving differences manually, by identifying changes via the technique in [52].

Pattern 9 (*Omission*)

Description Omission (aka Elimination) captures features to remove one or more elements from a process model and reconnect the remaining ones. It implies loss of process behavior.

Purpose To focus on specific parts of a process model while abstracting from the rest, due to the requirements of a project or specific audience.

Example Fig. 9a shows an example of Omission by displaying only the tasks from the process in Fig. 3b which are performed by a Loan Clerk, and omitting the automated tasks.

Metrics Decreases *model size* and may also decrease *diameter* and *average gateway degree*.

Rationale Simpler process models which focus on specific aspects relevant to a given audience are easier to understand by the latter [88].

Realization Omitting process behavior is related to the notion of *inheritance of workflow behavior* [2]. Accordingly, process model behavior can be restricted by applying two operators to Petri net transitions: *hiding* (an hidden transition remains in the model but its behavior is no longer observable) and *blocking* (a blocked transition is removed from the model with possible consequences to the control flow). [90] proposes a *decimation algorithm* to remove model nodes that are less relevant, and reconstructing the process flow. The relevance of a node is determined according to a *text search* (the nodes whose labels contain a given string are kept), and to *structural importance* (the main model structure is kept). [78] proposes a *slider approach* for building process model abstractions based on thresholds such as execution frequency, time and cost of nodes. [19] applies Omission to create public process views for inter-organizational cooperation, driven by the need to hide business data for confidentiality reasons. Users specify which process tasks represent cooperative tasks that can be safely shared with partners, and then a public process is derived which shows these tasks only. These approaches are close to the separation of concerns between BPEL public processes, containing communication tasks only, and BPEL private processes, also containing internal tasks. Deriving customized process views for specific audiences is also the purpose of [39], where a method for removing EPC functions based on the organizational structure is proposed. Process configuration is another research area where Omission is applied. It consists of removing from a *configurable process model* those fragments that are not relevant to a specific scenario. This is done via the application of an individualization algorithm. Individualization algorithms have been defined for various languages, including UML ADs [22], LTSs [36], EPCs [24] (pp. 111–148) [84], YAWL [38] and Petri Nets [3]. Finally, Omission also occurs in [30] and in [15] for achieving customized process views, and in [41] for simplifying process models that have been mined from logs. These approaches apply Omission in combination with Collapse mechanisms (more details are given in the realization of Collapse).

Pattern 10 (*Collapse*)

Description Collapse (aka Aggregation) describes features to synthesize multiple model elements into a single one of more abstract nature, where the distinction among the constituent elements is no longer relevant. It implies information synthesis.

Purpose To simplify a process model for a specific audience.

Example Fig. 9b shows an example of Collapse between single control-flow elements from the model in Fig. 3b. Here tasks Check debts and Check liability, and all tasks in the lower path of the XOR-split, have been collapsed into a single task for performing the required checks, resp., a single task for approving and processing loans.

Metric Decreases *model size*, and may also decrease *diameter* and *average gateway degree*.

Rationale Simpler process models which focus on specific aspects relevant to a given audience are easier to understand by the latter [88].

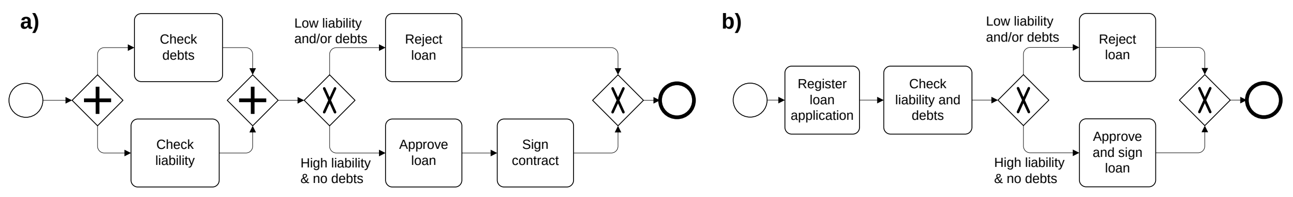

a)
Check debts
Check liability
Low liability and/or debts
Reject loan
High liability & no debts
Approve loan
Sign contract
b)
Register loan application
Check liability and debts
Low liability and/or debts
Reject loan
High liability & no debts
Approve and sign loan

Fig. 9. Two patterns applied to the model in Fig. 3b: (a) the Omission pattern was used to remove the automated tasks, and (b) the Collapse pattern was used to join the tasks in the lower branch following the XOR-split into a single task.

Realization [90] proposes an algorithm to incrementally simplify a process model by applying a set of reduction rules via which nodes with low relevance are gradually collapsed to their predecessor nodes. The order of the collapsed elements can be stored should the user wish to reverse the collapse effects. [30] uses a combination of Omission and Collapse to derive models for cross-organization collaboration. The approach contains two steps: *aggregation* and *customization*. In the first step, the provider selects those internal tasks that they wish to hide to a specific customer, and these tasks are collapsed accordingly. Afterwards, the customer specifies the tasks they are interested in and the remaining tasks are either collapsed with each other, or omitted from the model. Similarly, the Proviado approach [15] applies a combination of graph reduction (Omission) and graph aggregation (Collapse) techniques to obtain customized process views, based on a user query, e.g. "Show the tasks of a particular user only". Attribute values of aggregated tasks, e.g. data information, are also collapsed – a feature that is not available in other approaches, which only focus on control flow. Process views of workflow models are also obtained in [62] by collapsing connected sets of tasks. The approach in [80] proposes an algorithm for abstracting process models. It uses the model's RPST decomposition to identify fragments that are structurally suitable for collapse. Then *transformation rules* are applied to produce an abstracted fragment by collapsing its tasks. This algorithm lifts the limitations of another abstraction algorithm from the same authors [79]. The latter work aimed to collapse those tasks that are observed less often based on the identification of *abstraction patterns* (dead-end, sequential, block, and loop abstractions). As such, it was limited by the occurrence of these patterns in a model. [89] proposes an approach based on the notion of *behavioral profiles* [97]. A behavioral profile describing the behavior of a process model in terms of task order relations if first inferred from a process model. Then this profile is used together with an user-defined grouping of tasks to produce an abstract process model where selected tasks are collapsed based on the defined grouping of tasks. The degree of collapse is determined by the user. Collapse has also been applied in process mining [1]. [41] presents an algorithm in which both Collapse and Omission are used to identify non-essential details when extracting process models from logs. The algorithm applies a number of metrics (derivable from the logs) to establish the significance of and correlation between model elements, according to which process models are either collapsed or omitted.

<u>Pattern</u> 11 (*Restriction*)

Description This pattern captures features to restrict the syntax and semantics of a process modeling language, by removing modeling concepts from the language's meta-model. This pattern impacts all process models described by the restricted meta-model.
Purpose To improve understandability and maintenance through a simplified process model.
Example Fig. 10a shows the restriction of the "Home Loan" model in Fig. 6a after removing the concept of Event from the BPMN meta-model.
Metrics Decreases the number of *different modeling concepts* in a process model. It may decrease *model size* when removing any modeling concepts that are used in the process model.
Rationale A smaller number of element types is easier to learn for modeling novices [42], [74]. For this reason, modelers tend to use only a subset of language elements in practice [74].
Realization Different conformance classes have been introduced in BPMN 2.0 to restrict the meta-model in a controlled way. There are four classes for Process Modeling, Process Execution, BPEL Process Execution, and Choreography Modeling. Another example of meta-model restriction is the proposal by Silver [87]. It introduces three levels of BPMN usage: Level 1 for business people, Level 2 for analysts, and Level 3 for developers. The more business-oriented levels only contain a subset of the overall BPMN notation. A similar distinction is sometimes also made by software vendors. In Tibco's iProcess Suite, for instance, a process model can be classified according to its purpose, i.e. business analysis, deployment or simulation, where the meta-models for deployment and simulation implement a subset of the BPMN notation. Similarly, the BPEL standard defines a subset for abstract processes. Becker et et. [11] outline an approach for creating process model projections based on the restriction of the process' meta-model. These projections are achieved by applying so-called *configuration parameters* to meta-model elements in order to simplify the notation for a specific audience. For example, one can hide the resource perspective in the eEPC meta-model so that all the roles and organizational units that are associated with eEPC Functions are removed. This functionality is also implemented in the form of *filters* by tools such as ARIS and Signavio.

<u>Pattern</u> 12 (*Extension*)

Description This pattern captures features to extend the syntax and semantics of a process modeling language by adding new modeling concepts to the language's meta-model, or refining the existing ones.

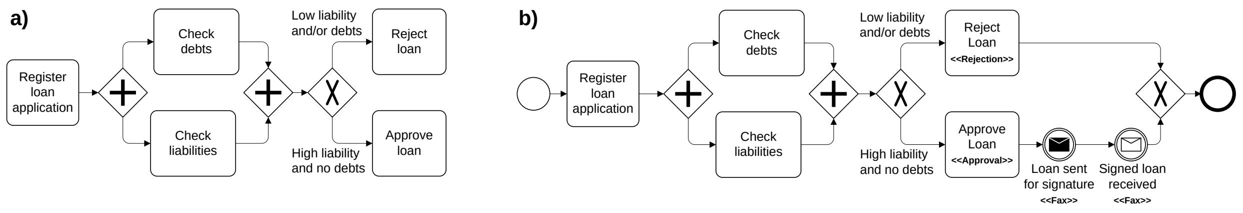

a)
Register loan application
Check debts
Check liabilities
Low liability and/or debts
High liability and no debts
Reject loan
Approve loan
b)
Register loan application
Check debts
Check liabilities
Low liability and/or debts
High liability and no debts
Reject Loan <<Rejection>>
Approve Loan <<Approval>>
Loan sent for signature <<Fax>>
Signed loan received <<Fax>>

Fig. 10. Examples of (a) Restriction and (b) Extension for the "Home Loan" model in Fig. 6a. In (a) the concept of Event has been removed from the BPMN meta-model. In (b) new stereotypes have been added to specialize the concepts of Task and Message Event.

Purpose To obtain either a closer match to the concepts of a particular domain, or a straightforward transformation to executable software.

Example Fig. 10b shows the "Home Loan" model in Fig. 6a after extending the BPMN meta-model with two Task subtypes, namely Approval and Rejection, and with one Message Event subtype, namely Fax (each captured by a stereotype). In Fig. 10b the Approval and Rejection Task stereotypes have been applied to task Reject loan, resp., Approve Loan, while the Fax stereotype has been applied to the two message events.

Metrics Increases the number of *different modeling concepts* in a process model. It may increase *model size* when a new modeling concept is used in the process model.

Rationale Refining a language's meta-model to suit the specific concepts of a domain makes the process models described by this language easier to understand for the audience of that domain [73], [34].

Realization Both UML ADs and BPMN benefit from the extension capabilities of the Meta-Object Facility (MOF) [45], which provides a standardized way for refining existing element types based on stereotyping. For instance, this mechanism has been used in [85] to extend BPMN with the concepts of *variation point* and *variant*, and in [22] to extend UML ADs with similar concepts. Moreover, various approaches exist which extend the meta-models of process modeling languages by adding new elements, rather than using stereotypes. For instance, [98] provides an extension to the BPMN meta-model to explicitly capture access control requirements. [16] extends BPMN to model non-functional aspects while [18] provides a similar extension to BPEL. BPEL4People [9] and BPELJ [14] are two further extensions of the BPEL meta-model to represent human tasks, resp., to incorporate java snippets in BPEL. Further, [91] extends UML ADs with concepts to define role-based access control aspects. EPCs have also been extended in different works with real-time concepts, object orientation, risk, and control-flow constructs of other languages [65]. UML modeling tools such as Enterprise Architect, typically support the standard UML extension mechanism of stereotypes. ARIS provides features to perform limited customizations of an eEPC meta-model (called *ARIS Method*). One can rename meta-model elements and attributes, and regroup attributes. However it is not possible to add new sub-types or attributes.

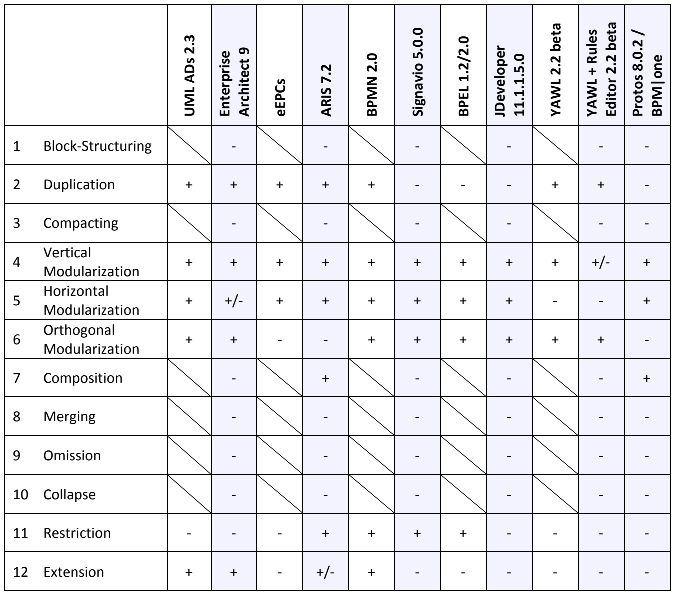
IV. BENCHMARKING

Similar to our previous work in [55], we now report the results of evaluating several languages and tools against their support for the identified patterns. The languages we selected for this evaluation are mainstream process modeling languages stemming from standardization efforts, large-scale adoptions or established research initiatives. Specifically, we chose three languages for conceptual process modeling (UML ADs 2.3, eEPCs and BPMN 2.0) and four languages for executable process modeling (BPMN 2.0, BPEL 1.2/2.0, YAWL 2.2 beta and Protos 8.0.2[5]). For each language, we also evaluated one supporting modeling tool. For UML ADs we tested Sparx's Enterprise Architect 9; for eEPCs we tested ARIS Business Architect 7.2 from Software AG; for BPMN we tested Signavio Editor 5.0.0; for BPEL we tested Oracle's JDeveloper 11.1.1.5.0; for YAWL we tested the YAWL Editor and Rules Editor; and for Protos the Protos Editor 8.0.2/BPM|one from Pallas Athena.

Table 11 shows the results of the analysis, where tool evaluations are shown next to the evaluations of the supported languages, except for Protos, where the language cannot be separated from its tool, because it is vendor specific. For a tool, we measured the extent by which it facilitates the support for a pattern, as it is offered by the corresponding language. We ranked a tool with a '-' if it offers no support for a pattern; with a '+/-' if the support is only partial or if it is the same as that offered by the corresponding language; and with a '+' if the support goes beyond that offered by the language, i.e. if the tool actually facilitates the application of a pattern. Accordingly, for Duplication we ranked Signavio, JDeveloper and Protos with a '-'. In particular, in Signavio call activity tasks cannot be linked with global tasks thus this pattern is not actually supported. For Vertical Modularization, we ranked the YAWL Editor with a '+/-' as it is not possible to navigate from a parent model to a subprocess. Enterprise Architect took a '+/-' for Horizontal Modularization as it supports the concept of UML Partition (to create parallel modules), but not that of Activity Edge Connector (to create sequential modules). For Extension we gave a '+/-' to ARIS as it only allows renaming of elements and attributes of the eEPC meta-model, but not the addition of new concepts.

Six of the twelve patterns that we identified in this paper (i.e. Block-Structuring, Compacting, Composition, Merging, Omission and Collapse) are not applicable to languages since they refer to tool features only (we crossed the corresponding cells for the languages in Table 11). For these patterns, we rated a tool with a '+' if it supported the pattern, and otherwise with a '-'. As per BPEL, even though this language is essentially block-structured, tool features could still be provided to block-structure or compact the content of a Flow

[5]Protos is now part of the BPM|one suite that allows Protos models to be executed. However, in most organizations, Protos is still mainly used for process modeling and analysis rather than for enactment.



		UML ADs 2.3	Enterprise Architect 9	eEPCs	ARIS 7.2	BPMN 2.0	Signavio 5.0.0	BPEL 1.2/2.0	JDeveloper 11.1.1.5.0	YAWL 2.2 beta	YAWL + Rules Editor 2.2 beta	Protos 8.0.2 / BPM|one
1	Block-Structuring	╲	-	╲	-	╲	-	╲	-	╲	-	-
2	Duplication	+	+	+	+	+	-	-	-	+	+	-
3	Compacting	╲	-	╲	-	╲	-	╲	-	╲	-	-
4	Vertical Modularization	+	+	+	+	+	+	+	+	+	+/-	+
5	Horizontal Modularization	+	+/-	+	+	+	+	+	+	-	-	+
6	Orthogonal Modularization	+	+	-	-	+	+	+	+	+	+	-
7	Composition	╲	-	╲	+	╲	-	╲	-	╲	-	+
8	Merging	╲	-	╲	-	╲	-	╲	-	╲	-	-
9	Omission	╲	-	╲	-	╲	-	╲	-	╲	-	-
10	Collapse	╲	-	╲	-	╲	-	╲	-	╲	-	-
11	Restriction	-	-	-	+	+	+	+	-	-	-	-
12	Extension	+	+	-	+/-	+	-	-	-	-	-	-

Fig. 11. Evaluation results.

activity (e.g. by removing redundant Link arcs). However, since JDeveloper does not offer any such feature, we rated this tool with a '-' along these two patterns. In fact, these six patterns are not supported by any of the evaluated tools, except for ARIS and Protos which cater for Composition. This indicates a clear immaturity of process modeling tools for such advanced features, which have only been explored in research.

One would expect that a tool generally offered wider pattern support than the respective language. Indeed, this is what we observed in the benchmark of tools for the concrete syntax patterns [55]. However for the abstract syntax patterns the results are more varied. ARIS supports more patterns than eEPCs, Enterprise Architect supports as many patterns as UML ADs, while Signavio, JDeveloper and the YAWL Editors support less patterns than their respective languages. The reason for such different level of sophistication among the tools evaluated may be twofold. First, these tools have a different maturity (for example, ARIS and Enterprise Architect have been around much longer than the others). Second, the difference in support between BPMN 2.0 and Signavio is likely due to the fact that BPMN 2.0 has only been standardized recently (a few months before the time of writing). Thus, we cannot yet expect a high level of maturity for its supporting tools. Finally, as observed for the concrete syntax patterns, BPMN 2.0 is the language supporting the greatest number of patterns, out of the languages being evaluated. This is clearly a reflection of the evolution of business process modeling languages.

V. Usability Evaluation

For the evaluation of the usability of the patterns we drew inspiration from the technology acceptance model [23] and its adaptation to conceptual modeling [64]. This theory postulates that actual usage of an information technology artifact—patterns in the case of this paper—is mainly influenced by the perceptions of potential users regarding usefulness and ease of use. Accordingly, a potential user who perceives a pattern to be *useful* and *easy to use* is likely to actually *adopt* it.

We conducted a series of focus group sessions with professionals to discuss the patterns, in a similar vein as in our earlier work [55]. Each session started with a presentation of the patterns, which was used to clarify any doubt that may arise in the audience through an open discussion. This was followed by the actual survey, where we inquired the audience about their perceived usefulness and ease of use of the patterns, and gathered their comments. Each session took between one and two hours, depending on the number of attendees and the length of the open discussions that took place. Altogether, 22 process modeling experts participated in these sessions, which took place in Sweden (five participants), in Germany (four participants) and in Australia (13 participants). On average, the participants had close to eight years experience with process modeling. When we asked them, they estimated that in the past twelve months each analyzed approx. 120 models on average, while having created over 45 models in the same period. The average size of such a model would be 30 tasks. Due to this extensive involvement in both process model analysis and development, the participants can be considered as genuine experts in the field.

We used a questionnaire with seven-point scale items adopted from [64] to measure the participants' perceptions on usefulness and ease of use for each of the patterns. Our earlier pattern evaluation using the same set of questions [55] already pointed at the high internal consistency of these questions, which is a measure for the reliability of this evaluation. Indeed, the computation of Cronbach's alpha as *ex post* reliability check for this evaluation provided the values 0.89 for usefulness and 0.82 for ease of use, which confirm our confidence in this instrument. The boxplots in Fig. 12 and 13 display the outcomes of the data analysis for the patterns' usefulness and ease of use respectively (in a boxplot the median value is shown as a horizontal line in a box, which represents the interval between the lower and upper quartiles).

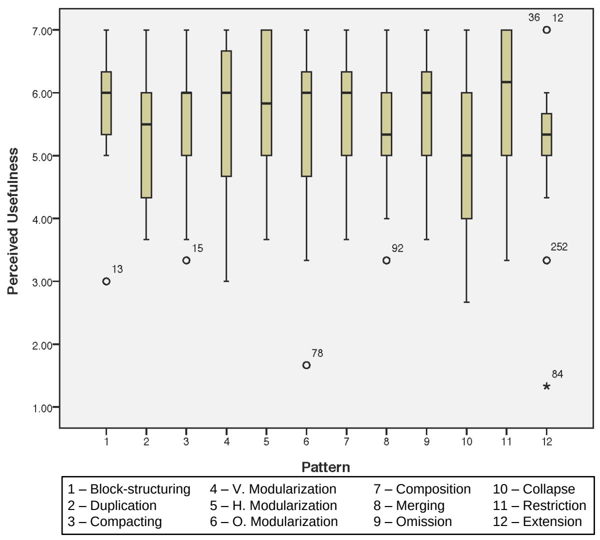


Fig. 12. Perceived usefulness of the patterns.

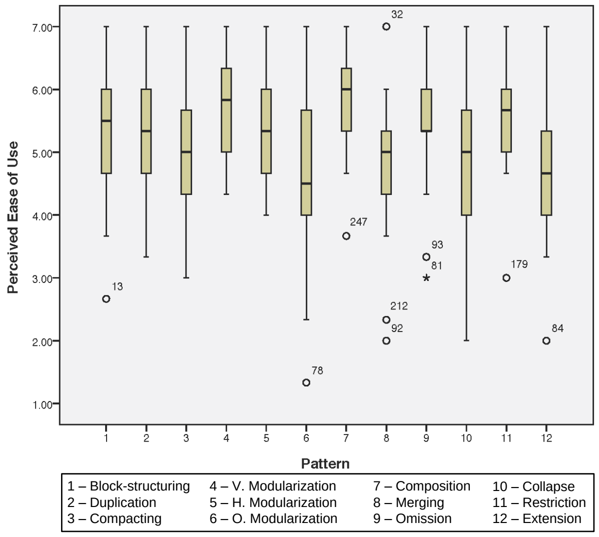


Fig. 13. Perceived ease of use of the patterns.

As displayed by Fig. 12, all patterns are perceived to be useful (each median at least equals 5). The patterns that received the highest scores are patterns 5 (Horizontal Modularization) and 11 (Restriction). Figure 13 shows that ease of use is considered positively too, with median values of 5 or more for all but patterns 6 (Orthogonal Modularization) and 12 (Extension); their median scores still exceeded 4. The patterns that received the highest scores for ease of use are patterns 4 (Vertical Modularization) and 7 (Composition).

During the survey, we invited the participants to explain their scores. One of the Swedish participants singled out Pattern 11 (Restriction), stating that it is "very efficient" and often applied by this person in modeling workshops. A participant from Australia stressed the value of Pattern 2 (Duplication), saying that "We use this a lot in our ARIS modeling, especially in swim lane models to avoid clutter and cross-over". Another pattern that received explicit praise for usefulness in the discussion was Pattern 3 (Compacting), as it was recognized to reduce confusion of name similarities in large models. By contrast, Pattern 6 received critical remarks. One of the German participants noted that it was rather complicated to understand and to apply. As he said, "If at all, this might be relevant for very large repositories of executable process models". However, in follow-up to this remark two other participants in the same session expressed their belief in the value of this pattern, in particular to separate exceptions from the normal flow.

In summary, the focus group sessions support the statement that the patterns can be considered useful and in general easy to use. It is interesting to note that in comparison to our earlier evaluation of patterns for concrete syntax modifications [55], the lack of tool support was not considered an issue. This may very well point at a greater adoption of abstract syntax patterns in praxis.

VI. Related Work

This paper should be seen as a continuation of the work presented in [55] where we described and evaluated eight patterns for concrete syntax modification. The goal of these patterns is to reduce the perceived model complexity without changing the abstract syntax, i.e., the goal is to simplify the representation of the process model without changing its formal structure. An example of such a pattern is Layout Guidance, i.e., the availability of layout conventions or advice to organize the various model elements on a canvas. The other seven patterns described in [55] are Enclosure Highlight, Graphical Highlight, Pictorial Annotation, Textual Annotation, Explicit Representation, Alternative Representation and Naming Guidance.

The twelve patterns presented in this paper complement the patterns of [55] as they operate on the abstract syntax of process models. For example, duplicating a task to make the model structured changes the abstract syntax whereas modifying the layout does not.

Many authors have worked on functionality related to abstract syntax modifications as is illustrated by the many references provided when describing the possible realizations of such patterns. However, we are not aware of other approaches that systematically collect patterns to improve the understandability of process models.

There have been other approaches to analyze the expressiveness or completeness of BPM languages and systems. For a review of these approaches, we refer to the related work section of our previous patterns collection [55].

Our work is also related to [73], which proposes a theory of general principles for designing cognitive-effective visual notations. In particular, our Modularization patterns can be seen as an implementation of the Principle of Complexity Management, which prescribes the provision of modularization and hierarchical abstraction to deal with model complexity. However, our patterns collection also provides other mechanisms related to complexity management besides modularization. Our Restriction pattern is related to the Principle of Graphic Economy, according to which the number of different graphical symbols should be controlled in order to be "cognitively manageable". In fact, as a result of restricting a meta-model, the number of symbols available will also be restricted. This is particularly valid for languages with an extensive number of graphical symbols such as eEPCs and BPMN, where Restriction could be applied to filter-out irrelevant symbols for particular audiences. Further, Extension is related to the Principle of Cognitive Fit, which prescribes the use of different dialects for different audiences. In fact this pattern can be used to create an audience-specific process modeling dialect by extending a process' meta-model.

VII. Conclusion

The main contribution of this paper is a systematic analysis of abstract syntax modifications for reducing the complexity in process models, as they occur in the literature, in process modeling languages, and tool implementations. This analysis took the form of a collection of frequently recurring patterns. These twelve patterns, combined with the eight patterns presented in [55], provide a comprehensive overview of existing mechanisms and language features to improve the understandability of process models by reducing complexity. The patterns in [55] focused on changes to the concrete syntax (e.g., improving the layout) but did not consider changes to the abstract syntax.

After documenting these patterns, we evaluated state-of-the-art languages and language implementations in terms of these patterns, and conducted a usability test with practitioners. The results of the usability test demonstrate that all identified patterns are indeed perceived as relevant and useful.

Although most tools provide some support for modifying models to improve their understandability, there is no real guidance on how to simplify and clarify the representation of process models. For example, many tools allow the duplication of model elements (e.g., two nodes referring to the same activity), but automated support to suggest duplication for increasing the understandability is missing in the current generation of process model editors. Thus, one could argue that today's tools are appropriate for model creation, yet provide little support for model management and maintenance. Since business processes change at an increasing pace and more and more variants of the same process need to be supported, this shortcoming is limiting the applicability of BPM technology. Therefore, we hope that tool vendors will use our patterns as a guide to drive the development of better functionality. We also aim to extend existing research tools such as the YAWL Editor, and the process model repository AProMoRe [56], with innovative features to support the identified patterns.

Acknowledgments We thank the practitioners involved in the empirical evaluation for their valuable time, and the anonymous reviewers for their comments which helped to improve this paper. This research is partly funded by the ARC Linkage Project LP110100252, "Facilitating Business Process Standardisation and Reuse" (La Rosa and Ter Hofstede).

References

[1] W.M.P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] W.M.P. van der Aalst and T. Basten. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[3] W.M.P. van der Aalst, M. Dumas, F. Gottschalk, A.H.M. ter Hofstede, M. La Rosa, and J. Mendling. Preserving Correctness During Business Process Model Configuration. *Formal Aspects of Computing*, 22(3):459–482, 2010.

[4] W.M.P. van der Aalst and K. M. van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, 2002.

[5] W.M.P. van der Aalst, R.S. Mans, and N.C. Russell. Workflow Support Using Proclets: Divide, Interact, and Conquer. *IEEE Data Eng. Bull.*, 32(3), 2009.

[6] A.A. Abdul, G.K.T. Wei, G.M. Muketha, and W.P. Wen. Complexity Metrics for Measuring the Understandability and Maintainability of Business Process Models using Goal-Question-Metric (GQM). *IJCSNS*, 8(5):219–225, 2008.

[7] M. Adams, A.H.M. ter Hofstede, W.M.P. van der Aalst, and D. Edmond. Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In *OTM*, volume 4803 of *LNCS*, pages 95–112. Springer, 2007.

[8] M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A Service-Oriented Implementation of Dynamic Flexibility in Workflows. In *OTM Conferences (1)*, volume 4275 of *LNCS*, pages 291–308. Springer, 2006.

[9] A. Agrawal, M. Amend, M. Das, C. Keller, M. Kloppmann, D. König, F. Leymann, R. Müller, G. Pfau, K. Plosser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. WS-BPEL Extension for People (BPEL4People), Version 1.0. Technical report, Active Endpoints, Adobe, BEA, IBM, Oracle and SAP, 2007.

[10] A. Basu and R.W. Blanning. Synthesis and Decomposition of Processes in Organizations. *Info. Sys. Research*, 14:337–355, 2003.

[11] J. Becker, P. Delfmann, A. Dreiling, R. Knackstedt, and D. Kuropka. Configurative Process Modeling – Outlining an Approach to increased Business Process Model Usability. In M. Khosrow-Pour, editor, *Proceedings of the 14th Information Resources Management Association International Conference*. IRM Press, 2004.

[12] R. Bergenthum, J. Desel, S. Mauser, and R. Lorenz. Construction of process models from example runs. *ToPNoC II*, 2009.

[13] G. Berthelot. Transformations and Decompositions of Nets. In W. Brauer, W. Reisig, and G. Rozenberg, editors, *Advances in Petri Nets 1986 Part I: Petri Nets, central models and their properties*, volume 254 of *Lecture Notes in Computer Science*, pages 360–376. Springer-Verlag, Berlin, 1987.

[14] M. Blow, Y. Goland, M. Kloppmann, F. Leymann, G. Pfau, D. Roller, and M. Rowley. BPELJ: BPEL for Java. Technical report, BEA and IBM, 2004.

[15] R. Bobrik, M. Reichert, and T. Bauer. View-Based Process Visualization. In *Proc of. BPM*, volume 4714 of *LNCS*, pages 88–95. Springer, 2007.

[16] C. Cappelli, J.C.S.P. Leite, T. Batista, and L. Silva. An Aspect-Oriented Approach to Business Process Modeling. In *Early Aspects of Aspect-Oriented Software Development*, pages 7–12, 2009.

[17] A. Charfi and M. Mezini. Aspect-Oriented Workflow Languages. In *Cooperative Information Systems*, volume 4275 of *LNCS*, pages 183–200. Springer, 2006.

[18] A. Charfi and M. Mezini. AO4BPEL: An Aspect-oriented Extension to BPEL. In *Proc. of WWW*, volume 10, pages 309–344. Springer, 2007.

[19] I. Chebbi, S. Dustdar, and S. Tata. The view-based approach to dynamic inter-organizational workflow cooperation. *Data & Knowledge Engineering*, 56(2):139–173, 2006.

[20] N. Chomsky. *Syntactic Structures*. Mouton, 1957.

[21] 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.

[22] K. Czarnecki and M. Antkiewicz. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Proc. of GPCE*, pages 422–437. Springer-Verlag, Berlin, 2005.

[23] F.D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):319–340, 1989.

[24] P. Delfmann. *Adaptive Referenzmodellierung. Methodische Konzepte zur Konstruktion und Anwendung wieder verwendungsorientierter Informationsmodelle*. Logos, 2006. (in German).

[25] J. Desel. Validation of process models by construction of process nets. In *BPM*, 2000.

[26] J. Desel and J. Esparza. *Free Choice Petri Nets*, volume 40 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge Univ. Press, Cambridge, UK, 1995.

[27] B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek. Verification of EPCs: Using Reduction Rules and Petri Nets. In *In Proc. of CAiSE*, volume 3520 of *LNCS*, pages 372–386. Springer, 2005.

[28] P. Effinger, M. Siebenhaller, and M. Kaufmann. An Interactive Layout Tool for BPMN. *E-Commerce Technology*, 0:399–406, 2009.

[29] F. Elliger, A. Polyvyanyy, and M. Weske. On Separation of Concurrency and Conflicts in Acyclic Process Models. In *EMISA*, pages 25–36, 2010.

[30] R. Eshuis and P.W.P.J. Grefen. Constructing customized process views. *Data Knowl. Eng.*, 64(2):419–438, 2008.

[31] J. Esparza and K. Heljanko. *Unfoldings: a partial-order approach to model checking*. Springer-Verlag New York Inc, 2008.

[32] D. Fahland. Oclets – Scenario-Based Modeling with Petri Nets. In *Proc of Petri Nets*, LNCS. Springer, 2009.

[33] D. Fahland and M. Weidlich. Scenario-based process modeling with Greta. In *BPM Demos*, volume 615. CEUR, 2010.

[34] M. Fowler and R. Parson. *Domain-Specific Languages*. Addison-Wesley, 2010.

[35] A. Gater, D. Grigori, and M. Bouzeghoub. Ranked Matching for OWL-S Process Model Discovery. Technical report, PRISM, 2010.

[36] F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Configurable Process Models: A Foundational Approach. In *Reference Modeling*, pages 59–78. Springer, 2007.

[37] F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Merging Event-Driven Process Chains. In *Proc. of CoopIS*, volume 5331 of *LNCS*, pages 418–426. Springer, 2008.

[38] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable Workflow Models. *Int. Journal of Cooperative Information Systems*, 17(2):177–221, 2008.

[39] F. Gottschalk, M. Rosemann, and W.M.P. van der Aalst. My own process: Providing dedicated views on EPCs. In *EPK 2005 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, volume 167 of *CEUR*, pages 156–175, 2005.

[40] V. Gruhn and R. Laue. Reducing the cognitive complexity of business process models. In *IEEE ICCI*, pages 339–345, 2009.

[41] C.W. Günther and W.M.P. van der Aalst. Fuzzy Mining - Adaptive Process Simplification Based on Multi-perspective Metrics. In *Proc. of BPM*, volume 4714 of *LNCS*, pages 328–343. Springer, 2007.

[42] M. H. Halstead. *Elements of Software Science*, volume 7 of *Operating, and Programming Systems Series*. Elsevier, 1977.

[43] R. Hauser, M. Friess, J.M. Kuster, and J. Vanhatalo. An Incremental Approach to the Analysis and Transformation of Workflows Using Region Trees. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 38(3):347–359, 2008.

[44] R. Hauser and J. Koehler. Compiling Process Graphs into Executable Code. In *GPCE*, volume 3286 of *LNCS*, pages 317–336, 2004.

[45] Brian Henderson-Sellers and Cesar Gonzalez-Perez. Uses and Abuses of the Stereotype Mechanism in UML 1.x and 2.0. In *In Proc. of MoDELS*, volume 4199 of *LNCS*, pages 16–26. Springer, 2006.

[46] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors. *Modern Business Process Automation: YAWL and its Support Environment*. Springer, 2010.

[47] N.F. Kock Jr. Product Flow, Breadth and Complexity of Business Processes: An Empirical Study of 15 Business Processes in Three Organizations. *Business Process Re-engineering & Management Journal*, 2:8–22, 1996.

[48] B. Kiepuszewski, A.H.M ter Hofstede, and C. Bussler. On Structured Workflow Modelling. In *Advanced Information Systems Engineering*, volume 1789 of *LNCS*, pages 431–445. Springer, 2000.

[49] N. Kock, J. Verville, A. Danesh-Pajou, and D. DeLuca. Communication flow orientation in business process modeling and its effect on redesign success: results from a field study. *Decision Support Systems*, 46(2):562–575, 2009.

[50] J. Koehler and R. Hauser. Untangling Unstructured Cyclic Flows – A Solution Based on Continuations. In *OTM*, volume 3290 of *LNCS*, pages 121–138, 2004.

[51] J.M. Küster, C. Gerth, A. Förster, and G. Engels. A Tool for Process Merging in Business-Driven Development. In *Proc. of the CAiSE'2008 Forum*, volume 344 of *CEUR Workshop Proceedings*, pages 89–92. CEUR, 2008.

[52] J.M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and Resolving Process Model Differences in the Absence of a Change Log. In *Proc. of BPM*, volume 5240 of *LNCS*, pages 244–260. Springer, 2008.

[53] J.M. Küster, K. Ryndina, and H. Gall. Generation of Business Process Models for Object Life Cycle Compliance. In *Proc. of BPM*, volume 4714 of *LNCS*, pages 165–181. Springer, 2007.

[54] M. La Rosa, M. Dumas, R. Uba, and R. Dijkman. Merging Business Process Models. In *OTM*, LNCS. Springer, 2010.

[55] M. La Rosa, A.H.M. ter Hofstede, P. Wohed, H.A. Reijers, J. Mendling, and W.M.P. van der Aalst. Managing Process Model Complexity via Concrete Syntax Modifications. *IEEE Trans. on Industrial Informatics*, 7(2), 2011.

[56] M. La Rosa, H.A. Reijers, W.M.P. van der Aalst, R.M. Dijkman, J. Mendling, M. Dumas, and L. García-Bañuelos. APROMORE: An Advanced Process Model Repository. *Expert Systems With Applications*, 38(6), 2011.

[57] R. Laue and V. Gruhn. Complexity Metrics for Business Process Models. In *Business Information Systems*, volume 85 of *LNI*, pages 1–12. GI, 2006.

[58] R. Laue and J. Mendling. Structuredness and its significance for correctness of process models. *Inf. Syst. E-Business Management*, 8(3):287–307, 2010.

[59] F. Leymann. Workflows Make Objects Really Useful. In *EMISA Forum*, volume 6, pages 90–99. GI, 1996.

[60] F. Leymann and D. Roller. Workflow-based Applications. *IBM Systems Journal*, 36(1):102–123, 1997.

[61] C. Li, M. Reichert, and A. Wombacher. The Minadept Clustering Approach for Discovering Reference Process Models Out of Process Variants. *Int. J. Cooperative Inf. Syst.*, 19(3-4):159–203, 2010.

[62] D.R. Liu and M. Shen. Workflow modeling for virtual processes: an order-preserving process-view approach. *Information Systems*, 28, 2003.

[63] R. Liu and A. Kumar. An Analysis and Taxonomy of Unstructured Workflows. In *Proc. of BPM*, volume 3649, pages 268–284, 2005.

[64] A. Maes and G. Poels. Evaluating quality of conceptual modelling scripts based on user perceptions. *Data & Knowledge Engineering*, 63(3):701–724, 2007.

[65] J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *LNBIP*. Springer, 2009.

[66] J. Mendling and W.M.P. van der Aalst. Formalization and Verification of EPCs with OR-Joins Based on State and Context. In J. Krogstie, A.L. Opdahl, and G. Sindre, editors, *In Proc. of CAiSE*, volume 4495 of *LNCS*, pages 439–453, 2007.

[67] J. Mendling, K.B. Lassen, and U. Zdun. On the transformation of control flow between block-oriented and graph-oriented process modelling languages. *International Journal of Business Process Integration and Management*, 3(2):96–108, 2008.

[68] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In *OTM*, volume 4803 of *LNCS*, pages 113–130. Springer, 2007.

[69] J. Mendling, H.A. Reijers, and J. Cardoso. What Makes Process Models Understandable? In *Proc. of BPM*, volume 4714 of *LNCS*, pages 48–63. Springer, 2007.

[70] J. Mendling, H.A. Reijers, and W.M.P. van der Aalst. Seven Process Modeling Guidelines (7PMG). *Information and Software Technology*, 52(2):127–136, 2010.

[71] J. Mendling and C. Simon. Business Process Design by View Integration. In *Proc. of BPM Workshops*, volume 4103 of *LNCS*, pages 55–64. Springer, 2006.

[72] B. Meyer. *Introduction to the Theory of Programming Languages*. Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[73] D.L. Moody. The "Physics" of Notations: Toward a Scientific Basis for Constructing Visual Notations in Software Engineering. *IEEE Transactions on Software Engineering*, 35:756–779, 2009.

[74] M. zur Muehlen and J. Recker. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation. In *Proc. of CAiSE*, volume 5074 of *LNCS*, pages 465–479. Springer, 2008.

[75] M. Petre. Cognitive dimensions 'beyond the notation'. *J. Vis. Lang. Comput.*, 17(4):292–301, 2006.

[76] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring Acyclic Process Models. In *BPM*, volume 6336 of *LNCS*, pages 276–293. Springer, 2010.

[77] A. Polyvyanyy, L. García-Bañuelos, and M. Weske. Unveiling Hidden Unstructured Regions in Process Models. In *OTM*, volume 5870 of *LNCS*, pages 340–356, 2009.

[78] A. Polyvyanyy, S. Smirnov, and M. Weske. Process Model Abstraction: A Slider Approach. In *Proc. of EDOC*, pages 325–331. Springer, 2008.

[79] A. Polyvyanyy, S. Smirnov, and M. Weske. Reducing Complexity of Large EPCs. In *Proc of. MobIS*, pages 195–207. GI, 2008.

[80] A. Polyvyanyy, S. Smirnov, and M. Weske. The Triconnected Abstraction of Process Models. In *Proc. of BPM*, volume 5701 of *LNCS*, pages 229–244. Springer, 2009.

[81] J. Recker, M. zur Muehlen, K. Siau, J. Erickson, and M. Indulska. Measuring Method Complexity: UML versus BPMN. In *AMCIS*, pages 1–12. AIS, 2009.

[82] H.A. Reijers, R.S. Mans, and R.A. van der Toorn. Improved model management with aggregated business process models. *Data Knowl. Eng.*, 68(2):221–243, 2009.

[83] H.A. Reijers, J. Mendling, and R.M. Dijkman. Human and automatic modularizations of process models to enhance their comprehension. *Information Systems*, 36(5):881–897, 2011.

[84] M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Information Systems*, 32(1):1–23, 2007.

[85] A. Schnieders and F. Puhlmann. Variability Mechanisms in E-Business Process Families. In *Proc. of BIS*, volume 85 of *LNI*, pages 583–601. GI, 2006.

[86] A. Sharp and P. McDermott. *Workflow Modeling: Tools for Process Improvement and Application Development*. Artech House, 2001.

[87] B. Silver. *BPMN Method & Style*. Cody-Cassidy Press, 2009.

[88] S. Smirnov, H.A. Reijers, T. Nugteren, and M. Weske. Business Process Model Abstraction: Theory and Practice. Technical Report 35, University of Potsdam, 2010.

[89] S. Smirnov, M. Weidlich, and Jan Mendling. Business Process Model Abstraction Based on Behavioral Profiles. In *Proc. of ICSOC*, volume 6470 of *LNCS*, pages 1–16, 2010.

[90] A. Streit, B. Pham, and R. Brown. Visualization Support for Managing Large Business Process Specifications. In *BPM*, volume 3649 of *LNCS*, pages 205–219. Springer, 2005.

[91] M. Strembeck and J. Mendling. Modeling process-related RBAC models with extended UML activity models. *Information & Software Technology*, 53(5):456–483, 2011.

[92] S. Sun, A. Kumar, and J. Yen. Merging workflows: A new perspective on connecting business processes. *Decision Support Systems*, 42(2):844–858, 2006.
[93] R. Uba, M. Dumas, L. García-Bañuelos, and M. La Rosa. Clone Detection in Repositories of Business Process Models. In *Proc. of BPM*, LNCS, 2011.
[94] J. Vanhatalo, J. Volzer, and J. Kohler. The Refined Process Structure Tree. *DKE*, 68(9):793–818, 2009.
[95] B. Weber, M. Reichert, J. Mendling, and H.A. Reijers. Refactoring large process model repositories. *Computers in Industry*, 62(5):467–486, 2011.
[96] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - Enhancing flexibility in process-aware information systems. *Data Knowl. Eng.*, 66(3):438–466, 2008.
[97] M. Weidlich, J. Mendling, and M. Weske. Efficient Consistency Measurement based on Behavioural Profiles of Process Models. *IEEE TSE*, page (to appear), 2011.
[98] C. Wolter and A. Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In *In Proc. of BPM*, volume 4714 of *LNCS*, pages 64–79. Springer, 2007.

Marcello La Rosa is a Senior Lecturer with the BPM research group at the Queensland University of Technology, Brisbane, Australia. He obtained his PhD in Computer Science with the same research group in 2009. His research interests embrace different topics in the BPM area, such as management of large process model collections, process modeling, configuration and automation. Marcello is a senior trainer for professional education courses on BPM and service-oriented architecture topics.

Petia Wohed is an Associate Professor at the Department of Computer and Systems Sciences at Stockholm University, where she is a member of the Information System Laboratory. Since her graduation in 2000, Wohed's main interest is in the area of business process management, in which she has worked on a series of patterns-based analyses of modern process modeling languages and open-source workflow systems.

Jan Mendling is a Junior-Professor with the Institute of Information Systems at Humboldt-Universität zu Berlin, Germany. His research areas include business process management, conceptual modeling and enterprise systems. He has published more than 100 research papers and articles. Jan is member of the editorial board of three international journals. He served as co-chair for the Int. Conference on BPM in 2010 and was the initiator of the BPMN workshop series.

Arthur H.M. ter Hofstede is a Professor at the Faculty of Science & Technology of Queensland University of Technology in Brisbane, Australia. He is co-leader of the BPM research group in this faculty. He is also a Professor in the Information Systems group at Eindhoven University of Technology in Eindhoven, The Netherlands. His main research interests lie in the area of business process automation. He is involved in both the Workflow Patterns Initiative and the YAWL Initiative.

Hajo A. Reijers is an Associate Professor in the Information Systems group at Technische Universiteit Eindhoven and an Affiliated Professor with the TIAS/Nimbas Business School of Tilburg University. His research interests cover business process redesign, business process modeling, workflow management technology, and simulation. He published over 75 refereed papers and served as the co-chair of the Int. Conference on BPM in 2009.

Wil M.P. van der Aalst is Full Professor of Information Systems at the Technische Universiteit Eindhoven and Adjunct Professor at Queensland University of Technology. His research interests include workflow management, process mining, Petri nets, process modeling and analysis. Many of his papers are highly cited (he has an H-index of 80 as per Google Scholar, making him the Dutch computer scientist with the highest H-index) and his ideas have influenced researchers, software developers and standardization committees on process support.