Business Process Variability Modeling: A Survey

MARCELLO LA ROSA, Queensland University of Technology, Australia and University of Liechtenstein, Liechtenstein
WIL M.P. VAN DER AALST, Eindhoven University of Technology, The Netherlands and Queensland University of Technology, Australia
MARLON DUMAS, University of Tartu, Estonia
FREDRIK P. MILANI, University of Tartu, Estonia

It is common for organizations to maintain multiple variants of a given business process, such as multiple sales processes for different products or multiple bookkeeping processes for different countries. Conventional business process modeling languages do not explicitly support the representation of such families of process variants. This gap triggered significant research efforts over the past decade leading to an array of approaches to business process variability modeling. This survey examines existing approaches in this field based on a common set of criteria and illustrates their key concepts using a running example. The analysis shows that existing approaches are characterized by the fact that they extend a conventional process modeling language with constructs that make it able to capture customizable process models. A customizable process model represents a family of process variants in a way that each variant can be derived by adding or deleting fragments according to configuration parameters or according to a domain model. The survey puts into evidence an abundance of customizable process modeling languages, embodying a diverse set of constructs. In contrast, there is comparatively little tool support for analyzing and constructing customizable process models, as well as a scarcity of empirical evaluations of languages in the field.

1. INTRODUCTION

The co-existence of multiple variants of the same business process is a widespread phenomenon in contemporary organizations. As a concrete example, The Netherlands has around 430 municipalities, which in principle execute the same or a very similar set of processes. For instance, all municipalities have processes related to building permits, such as the process of handling applications for permits and the process for handling objections against such permits. Due to demographics and political choices though, each municipality executes its processes differently. Some differences are un-

intentional, but others are deliberate. Variations are justified by different priorities and customs, often referred to as the "Couleur Locale". These differences have come to be accepted and there is no willingness to flatten them out. Still, capturing multiple municipality processes in a consolidated manner is desirable in order to design information systems that can support multiple or all municipalities at once, and thus achieve economies of scale.

In a similar vein, *Suncorp Group* – the largest insurance group in Australia – offers a wide portfolio of insurance products, including home, motor, commercial and liability insurance. Each of these products generally exists for different brands of the group (e.g. Suncorp, AAMI, APIA, GIO and Vero). As a result, there are more than 30 variants of the process for handling an insurance claim at Suncorp Group. There is a case for modeling and maintaining these variants in a consolidated manner, not only to avoid redundancy, but also so that improvements and automation efforts made on one variant can also benefit other variants.

Applying conventional business process modeling approaches (see [Mili et al. 2010] for a survey) to families of process variants such as the above ones, requires one of two options to be chosen. Either each variant is modeled separately, resulting in significant duplication as the variants have much in common, or multiple variants are modeled together, but then the complexity of the consolidated model grows rapidly and it becomes difficult to analyze and maintain individual variants.

Motivated by this observation, a number of approaches to modeling families of business process variants have emerged over the past decade. The common aim of these approaches is to provide a means to represent a family of business process variants in a consolidated manner, meaning via a single model. Each variant of the process can be derived from this single model by applying certain allowed transformations. We hereby use the term *customizable process model* to refer to the single unitary model of a family of business process variants.

This survey analyzes the problem space of business process variability modeling and distills a set of evaluation criteria for assessing approaches in this field. Next, the survey identifies and provides a comprehensive comparative overview of the identified approaches, in light of the evaluation criteria. Finally, the survey exposes common limitations of existing approaches and outlines directions for future research.

The remainder of the article is organized as follows. Section 2 analyzes and contextualizes the problem of business process variability management. Section 3 describes the process followed for the selection and review of relevant approaches, while Section 4 defines and justifies the criteria used in the comparative evaluation. Section 5 presents a working example that is used to discuss and illustrate each surveyed approach in Section 6. Section 7 discusses different techniques for ensuring correctness of customized process models, while Section 8 discusses techniques for decision support during process customization. Finally, Section 9 provides a synthesis of the surveyed approaches, leading to insights that are summarized in Section 10.

2. CONTEXT AND SCOPE

Since the 1990s, researchers have acknowledged different types of business processes (or workflows) depending on their level of predictability. For instance, [Georgakopoulos et al. 1995] discusses a spectrum of business processes ranging from fully automated processes to ad-hoc processes with considerable human involvement and high levels of uncertainty. The challenges associated to managing ad hoc (also called *flexible*) business processes attracted the attention of numerous researchers [Reichert and Dadam 1998; Heinl et al. 1999; Rinderle et al. 2004; Weber et al. 2008; van der Aalst et al.

2009]. In fact, flexibility in business processes has been one of the most active areas of research in the field of business process management [Reichert and Weber 2012].

Business process variability is related to flexible process management. To understand this relation, it is useful to distinguish three phases in the lifecycle of customizable process models (see Figure 1):

— **Design-time.** In this phase, the customizable process model is created. The design choices taken in this phase will impact the entire process family captured by the customizable process model.
— **Customization-time.** In this phase, the customizable process model is customized to realize a particular process variant. The *customized process model* describes a single process variant that is ready to be enacted. The customization choices taken in this phase will affect a selected group of process instances, i.e. those that can be generated by the specific process variant obtained via customization.
— **Run-time.** In this phase, the customized process model is enacted for *individual process instances*. For each instance, run-time choices are made, e.g. accepting or rejecting a particular claim based on the evidence gathered. Such choices affect only one specific instance.

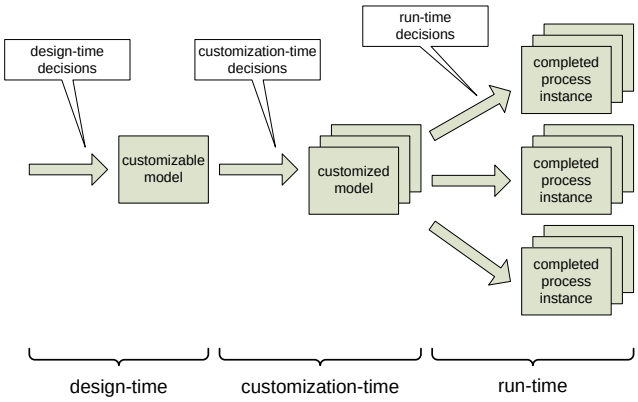

Fig. 1: Three main phases of decision making related to customizable process models.

Design-time and customization-time decisions affect numerous process instances and perhaps even multiple organizational units. The timeframe associated with such decisions is long (e.g., months or years). In contrast, run-time decisions are local and impact only one or a few process instances.

Given the above view of the lifecycle of a customizable process model, we can state that *flexibility* and *variability* are concerned with different stages at which decisions regarding a business process are made. Flexibility is concerned with run-time decisions while variability is concerned with design-time and customization-time decisions.

Zooming further down in this conceptualization, [Schonenberg et al. 2008] distinguishes four types of flexibility as depicted in the left side of Figure 2.

Flexibility by Design is the ability to incorporate alternative execution paths within a process definition at design-time such that the selection of the most appropriate execution path can be done at run-time for each process instance. Consider for example a process model that allows the user to select a route (using for example XOR splits); this is a form of flexibility by design. Also parallel processes are more flexible than sequential ones, e.g., a process that allows activities A and B to be executed in parallel,

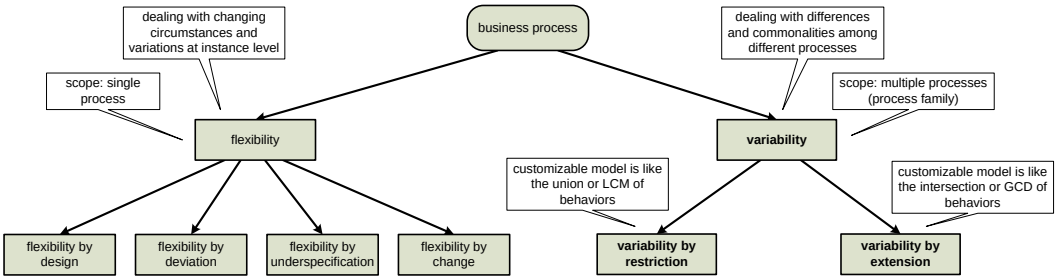

Fig. 2: Taxonomy of flexibility and variability in business processes.

also allows the sequence B followed by A (and vice versa), thus offering some form of flexibility. Declarative approaches such as the constraint-based language Declare provide a completely different starting point, i.e., anything is possible as long as it is not forbidden [van der Aalst et al. 2009].

Flexibility by Deviation is the ability for a process instance to deviate at run-time from the execution path prescribed by the original process model without altering the model. Such a deviation refers to changes to an execution sequence for a specific process instance (e.g. activity a is skipped for a case); it does not refer to changes in the process definition or to the activities that it comprises. Many systems allow such functionality to some degree, i.e., there are mechanisms to deviate without changing the model or requiring any modeling efforts. The concept of case handling allows activities to be skipped and rolled back as long as people have the right authorization [Reijers et al. 2003]. Flexibility by deviation is related to exception handling as exceptions can be seen as a kind of deviation [Casati et al. 1999].

Flexibility by Underspecification is the ability to execute an incomplete process model at run-time, i.e., one which does not contain sufficient information to allow it to be executed to completion. This type of flexibility does not require the model to be changed at run-time, instead the model must be completed by providing a concrete realization for the undefined parts. There are two types of underspecification: *late binding* and *late modeling*. Late binding means that a missing part of the model is linked to some pre-specified functionality (e.g., a sub-process) at run-time. Late modeling means that at run-time new (i.e., not predefined) functionality is modeled, e.g., a sub-process is specified. For example, worklets allow both late modeling and late binding [Adams et al. 2007]. [Sadiq et al. 2001] discuss various approaches to underspecification.

Flexibility by Change is the ability to modify a process model at run-time such that one or all of the currently executing process instances are migrated to a new process model. Unlike the previous three flexibility types, the model constructed at design-time is modified and possibly one or more instances need to be transferred from the old to the new model. There are two types of flexibility by change: (1) *momentary change* (also known as change at the instance level or ad-hoc change): a change affecting the execution of one or more selected process instances (typically only one), and (2) *evolutionary change* (also known as change at the type level): a change caused by modification of the process model, affecting all new process instances. Changing a process model leads to various types of problems as indicated in [Ellis et al. 1995; Reichert and Dadam 1998; Rinderle et al. 2004]. In [Ellis et al. 1995] the concept of so-called *dynamic change bug* was introduced. In the context of ADEPT [Reichert and Dadam 1998] much work has been performed on workflow change. See [Rinderle et al. 2004] for an overview on related work on flexibility by change.

The above four types of flexibility consider a single process and deal with changing circumstances or variations at the instance level. Flexibility is *not* concerned with maintaining *multiple* process models that together form a family of process models. Flexibility approaches do *not* differentiate between models of families of process variants and models of individual variants, as shown in Figure 1. Nevertheless, there are obvious relationships. For example, completing an incomplete specification (i.e., flexibility by underspecification) can be seen as a form of customization. Flexibility realized by ad-hoc changes creates different variants of the same process at the instance level. Flexibility realized by changes at the type level may also result in multiple variants of a process that are temporarily running in parallel. These examples illustrate that flexibility and variability share common concepts. However, while flexibility is mostly *reactive* (i.e., it deals with changes in the environment or particularities related to specific cases), business process variability is more *proactive* in nature. The goal is to create a customizable process model shared among different organizations or units within the same organization. This customizable process model may then act as a reference model for such a consortium of organizational entities and be customized to fit the specificities of a particular scenario such as a new organization or business unit.

A detailed overview of methods for flexible business process management is given by [Reichert and Weber 2012]. Accordingly, the present survey does not deal with flexibility but instead it focuses on the related topic of business process variability modeling (right side of Figure 2). On this side of the figure, we distinguish two forms of variability,: i) variability by restriction and ii) variability by extension.

Variability by restriction starts with a customizable process model that contains all behavior of all process variants. Customization is achieved by restricting the behavior of the customizable process model. For example, activities may be skipped or blocked during customization. In this setting, one can think of the customizable process model as the union or Least Common Multiple (LCM) of all process variants. Customizable process models of this type are sometimes called *configurable process models*.

Variability by extension takes the opposite starting point. The customizable process model does not contain all possible behavior, but instead it represents the most common behavior or the behavior that is shared by most process variants. At customization time, the model's behavior needs to be extended to serve a particular situation. For example, one may need to insert new activities in order to create a dedicated variant. In this setting, one can think of a customizable process model as the intersection or Greatest Common Denominator (GCD) of all process variants under consideration.

The survey covers both variability by restriction and by extension. In fact, as discussed later in the survey, it is possible for one same approach to combine both types of variability.

Customizable process models ought to be distinguished from so-called *reference process models* [Fettke and Loos 2003; Rosemann 2003]. Some vendors and consultancy firms provide reference process models that are intended to capture common knowledge or best practices in a given field (e.g. the ARIS-based models for ITIL or SCOR). While a reference process model can be very useful in its own way, it should be understood that it is in essence a concrete process model meant to be used as an example. Reference process models do not support customization in a structured manner. In this survey, we focus on approaches that provide support for customization rather than pure reference process models.

Having discussed the scope and context of the survey, we next describe the search process used to retrieve relevant publications and we define criteria to characterize the approaches described in these publications.

3. SEARCH PROCESS

The literature search process was conducted based on the principles of systematic literature review [Kitchenham 2004]. The process started by submitting queries to a well-known research literature database (Google Scholar) covering the main keywords associated with the scope of the survey. We first determined that the term "customization" is associated with "variation" and "configuration". Accordingly, we constructed queries by combining the keyword "business process" with "customization", "variation", "configuration", "customizability", "variability", "configurability", "customizable" and "configurable". We also noted that the keyword "flexibility" is often associated with customizability, and thus also constructed queries based on "flexible" and "flexibility". Since the term "workflow" is sometimes used as a quasi-synonym of "business process", we also included queries combining "workflow" with the above keywords.

For each query, we gathered the first 100 hits in Google Scholar and did a first filtering based on the title in order to eliminate papers that were clearly not related to the topic. As we conducted the search, we noted additional terms appearing in the titles of relevant papers, namely "business process variant", "configurable reference model", "reference model adaptability", "reference model flexibility" and "configurable EPC", and also used such terms in the search.

We are aware that extensive research pertaining to variability modeling in software systems – most notably using feature models [Schobbens et al. 2006] – has been conducted in the field of Software Product Line Engineering (SPLE) [Czarnecki and Eisenecker 2000]. Some of this research addresses the question of variability in business process models captured by means of UML activity diagrams. Accordingly, we also included queries composed of "UML activity diagram" in conjunction with "software product line" or "feature model". In the case of these additional queries, we inspected the first 50 hits.

The search was performed in March 2013 using the above procedure. This resulted in 2,500 hits. Based on title, we filtered out papers that were clearly out of scope. We also removed duplicates, papers with no citations, as well as short papers (less than 5 pages) as they would not contain enough information for an evaluation. This initial filtering reduced the number of candidate papers to 213.

We then proceeded with an inspection of the abstract of each paper in order to eliminate papers that fell within the following related but clearly distinct areas of study: management of process model variants without the use of customizable process models, run-time variation (or process flexibility), exception handling and automated workflow composition.

During the search process, we identified an existing literature mapping study in the field of business process variability [Valenca et al. 2013]. This mapping study provides an inventory of 80 publications covering both design-time and run-time variability. We extracted from this mapping study publications related to design-time variability management via customizable process models. We verified that all relevant publications were also found during our search. Our search returned several publications not covered in [Valenca et al. 2013]. Moreover, [Valenca et al. 2013] only provides a classified listing of approaches to business process variability management, but does not analyze in detail and compare individual approaches as we do in this survey.

At the end of the search process, we obtained 66 relevant publications. An histogram of these publications per year of publication is shown in Figure 3. The histogram shows an increasing trend of publications on the topic, reaching a pick in 2008.

In general, a given approach is covered by multiple publications. Also, we found that some approaches are subsumed by others, meaning that the features and concepts in one approach are contained by the other approach. In total, we found that the 66 pub-

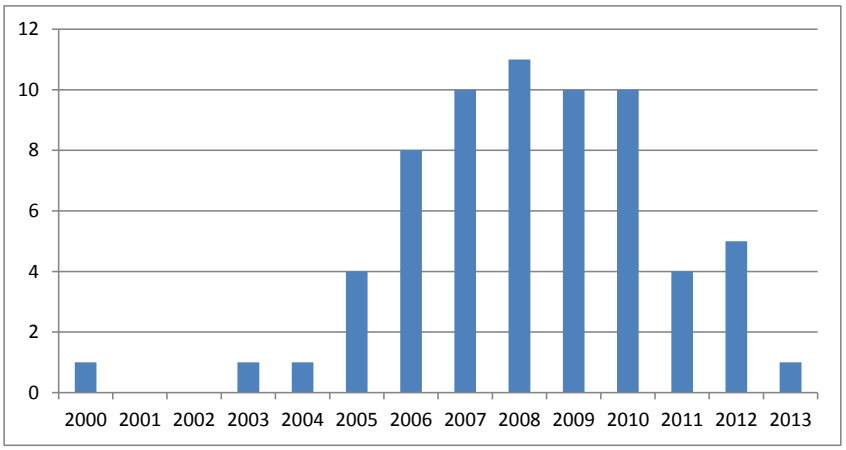

Fig. 3: Number of publications on variability management via customizable process models.

lications cover 19 different approaches, out of which eight approaches are subsumed by the other eleven approaches. Accordingly, the survey distinguishes eleven main approaches and eight subsumed approaches.

The publications covered by the survey are listed in a supplemental table available at: `http://sep.cs.ut.ee/uploads/Main/SupplementaryMaterialListOfStudies.xlsx`. For each approach, the table identifies a primary (earliest) publication describing the approach and, where available, additional publications describing further aspects of the same approach.

4. EVALUATION CRITERIA

The approaches presented in this paper are evaluated against a set of criteria that have been derived from the purpose of customizable process models. We have derived 14 criteria grouped into four main categories. These criteria are mutually orthogonal and have been designed while keeping simplicity in mind. For example, the criteria under "Perspective" could have been decomposed further into "Visual representation of variation points" and "Support for customization". However, such a finer granularity would only add limited value while jeopardizing simplicity.

1 **Inherent type.** This category refers to the mechanism employed to customize a customizable process model. In this respect, we can classify approaches into the two non-disjoint categories already depicted in Figure 2). Hence, we identify the following two criteria:
   - 1.1 **Restriction (IT.Res)**. An approach matches this criterion if variability is captured by restricting the behavior captured in a given process model.
   - 1.2 **Extension (IT.Ext)**. An approach matches this criterion if it supports variability via extension or modification of the behavior captured in a given process model.

2 **Scope.** This category refers to the types of processes and the aspects of a process covered by an approach. This category is decomposed into two sub-categories: *Perspective* (breadth) and *Process Type* (depth). The former refers to the ability of the customizable model to capture variability along the various perspectives of process modeling. The latter refers to variability support during the whole life-cycle of business process management, starting from the inception of customizable models

until the execution of customized process models for concrete process instances (as shown in Figure 1).

2.1 **Perspective.** This category assesses the breadth of an approach in terms of support for customization.

2.1.1 **Control flow (P.Cf).** The ability of a customizable model to capture variability along the control-flow perspective, i.e. variability of activities and routing elements such as gateways or connectors. A language is considered to only partially fulfill this criterion if routing elements or activities are not customizable, or if such elements are customizable but the corresponding customization options are not graphically represented.

2.1.2 **Resources (P.Re).** A customizable model supports the resource perspective if it is able to capture variability in the participating resources of the process (for example managing roles and information systems that may be optional or omitted in specific scenarios). A language is considered to partially fulfill this criterion if resources are customizable but such information cannot be represented graphically.

2.1.3 **Objects (P.Ob).** The last criterion in this category refers to the ability of a customizable model to capture variability in the business objects produced and consumed by a process. In this context, a business object is a physical or a data artifact that is used as input or output by ay least one activity in the process. A language is considered to partially fulfill this criterion if business objects are customizable, but business objects and their customization options are not graphically represented. For convenience, we refer to the set of business objects of a process (and their relations) as the *data perspective* of a process.

2.2 **Process Type.** This category assesses the depth of an approach in terms of usefulness of the customized model.

2.2.1 **Conceptual (PT.Con).** An approach meets this criterion if it is designed to support conceptual process models for the purpose of documentation and analysis, as opposed to executable process models.

2.2.2 **Executable (PT.Exe).** An approach is considered to fulfill this criterion if the customization also removes inconsistencies in resources and data, making the customized models effectively executable on top of a concrete Business Process Management System (BPMS). If the customized models can be executed on a BPMS, but these inconsistencies are not addressed, the approach is considered to only partially fulfill the criterion. Similarly, the criterion is partially fulfilled if there is no BPMS that can effectively support the execution of the customized models, even if inconsistencies on the data/resource perspective are addressed at the algorithmic level.

3 **Supporting Techniques.** This category refers to the supporting techniques on customizable models. This category is decomposed into two sub-categories as follows.

3.1 **Correctness Support.** Ensuring the correctness of a single process model is already challenging. As a high number of customizations can typically be derived from a customizable process model, correctness issues become even more important. For example, it is easy to introduce errors by not anticipating interactions between different configuration decisions. Therefore, we define two criteria related to the ability of an approach to guarantee the correctness of the customized models.

3.1.1 **Syntactical correctness (CS.Syn).** Ability to guarantee the correct structure of the customized models, e.g., by avoiding disconnected nodes.

3.1.2 **Behavioral correctness (CS.Beh).** Ability to guarantee the correct behavior of the customized models, e.g., by avoiding behavioral anomalies such as deadlocks and livelocks. A typical requirement is soundness [van der Aalst et al. 2011], i.e., it should always be possible to complete any process instance properly.

3.2 **Decision support.** This sub-category deals with support for the selection of customization alternatives.

3.2.1 **Abstraction (DS.Abs).** End-users often do not think in terms of process models. Customization is typically based on knowledge and decisions at the business level. An approach supports process model abstraction if end-users can customize a model without directly referring to its model elements, but instead to concepts in their domain of discourse.

3.2.2 **Guidance (DS.Gui).** This criterion is met if there is tool support that: (i) guides users when making customization decisions, for example in the form of recommendations for selecting one option or another; and (ii) prevents users from making inconsistent or irrelevant customization decisions.

4 **Extra-functional.** This category refers to extra-functional aspects, i.e., aspects not directly related to how customizable models are represented or manipulated.

4.1 **Formalization (EF.For).** Some of the approaches described in the literature only present ideas and do not provide concrete algorithms or rigorous definitions. Therefore, we include a criterion indicating whether the approach has been described rigorously in terms of mathematical or logic notations. In order to fulfill this criterion, the approach has to be formally defined, including algorithms used during customization. If such an algorithm is missing, it can at best be considered to partially fulfill the criterion.

4.2 **Implementation (EF.Imp).** Approaches may only exist on paper. However, the usability and maturity of an approach heavily depends on tool support to design and customize customizable process models. If the approach is fully implemented with tool support for enactment of customizable models (including algorithms used during customization), it fulfills this criterion. Approaches with partial implementations, e.g. only offering design-time or customization-time support (see Figure 1), can at best be considered to partially fulfill this criterion.

4.3 **Validation (EF.Val).** More mature approaches have been evaluated based on real-life process variants and through discussions with domain experts. An approach fulfills this criterion if it has been tested on models not created by the authors, and the results verified by domain experts. If one of these two aspects is lacking, e.g., an approach that has been tested without the involvement of domain experts, then it fulfills this criterion only partially.

Using these criteria, in Section 6 we document the results of the evaluation of the approaches retrieved using the literature search process described in Section 3. In particular, each approach was evaluated by two authors and the results were validated through discussions with the other two authors. In the end, all authors were in agreement with the evaluation results.

Before that, in the next section we present an example process family that we will use throughout the paper to illustrate the approaches reviewed.

5. ILLUSTRATIVE SCENARIO

The example process family described in this section is the result of a case study in picture post-production that we conducted with domain experts from the Australian Film, Television and Radio School (AFTRS) in Sydney.[1]

In the film industry, picture post-production (post-production for short), is the process that starts after the shooting has been completed, and deals with the creative editing of the motion picture.

Figure 4 shows several variants of the picture post-production process, represented in the *Event-driven Process Chains (EPCs)* language [IDS Scheer 2002; Davis and Brabander 2007]. An EPC is a directed graph consisting of events, functions, connectors and arcs linking these elements. Each EPC starts and ends with at least one event. Events are triggers for functions and signal their completion, while functions represent activities to be performed. Each function is preceded and followed by an event. Connectors are used to model alternative and parallel branching and merging and are divided into splits (with multiple outgoing arcs and one incoming arc) and joins (with multiple incoming arcs and one outgoing arc). Splits and joins have a logical type. They can be of type OR or XOR (for inclusive, resp., exclusive decision and merging) and AND (for parallelism and synchronization).

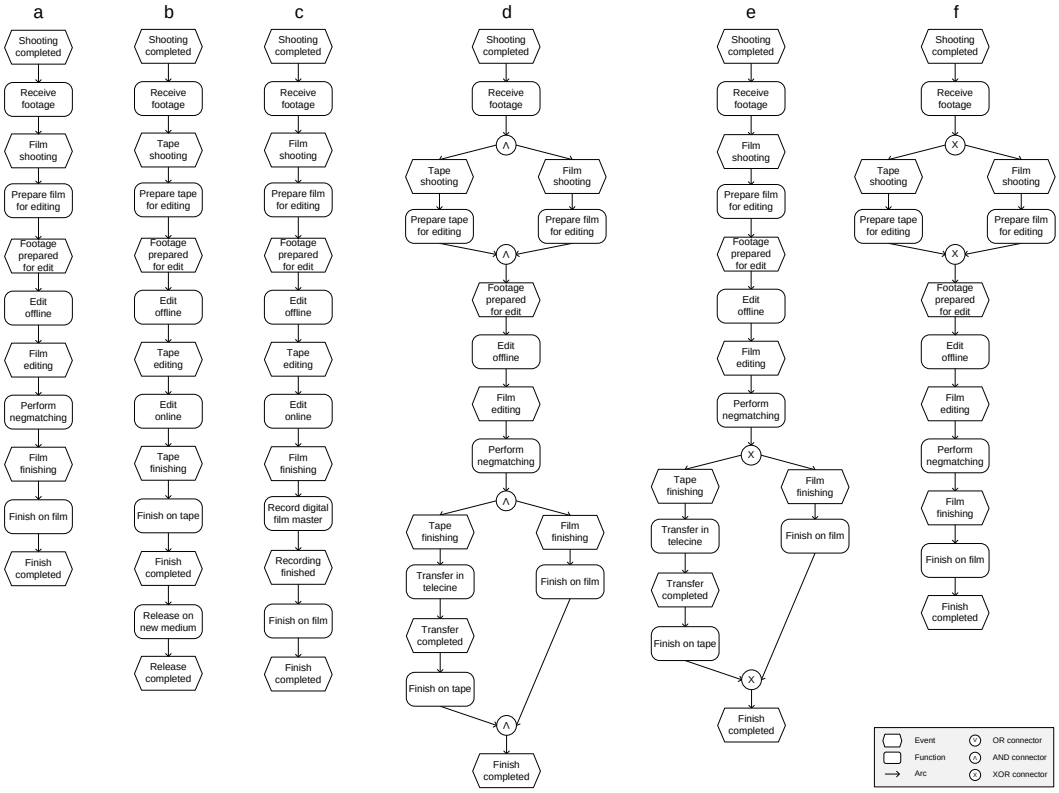

Fig. 4: Different variants of the picture post-production process in the EPC language.

[1] www.aftrs.edu.au

Post-production starts with the receipt, from the shooting, of the footage that needs to be prepared for editing. The footage can either be prepared on film (see e.g. variant a of Figure 4, where function "Prepare film for editing" in performed), on tape (e.g. variant b, where function "Prepare film for editing" is performed) or on both media (variant d) depending on whether the motion picture was shot on a film roll and/or on a tape. Next, the medium is edited offline to achieve the first rough cut (thus function "Edit offline" exists in all variants). However, after this, an online editing is carried out if the footage was shot on tape (variants b and c), while a negmatching is carried out if the footage was shot on film (e.g. variant a). Online editing is a cheap editing procedure that is well suited for low-budget movies typically shot on tape. Negmatching offers better quality results although it requires higher costs; thus it is more suitable for high-budget productions typically shot on film. The choice between online editing and negmatching represents an important decision in post-production: depending on drivers such as budget, creativity and type of project, one option, the other or both need to be taken. Thus, each variant in Figure 4 reflects a common practice in post-production. For example, variant a is a typical low-budget practice (shooting and releasing on tape), whereas variant d illustrates a more expensive procedure (shooting and releasing on both tape and film).

The final step of post-production is the finishing of the edited picture. Again, this can be be done on film (see variant a), on tape (variant b) or on both (variant d). The finishing may involve further activities depending on the combination of type of editing and final medium. For example, if the editing was done online and the final version is on film, a digital film master needs to be recorded from the edited tape (see variant c). Alternatively, if a negmatching was performed and the final version is on tape, the edited film needs to be transferred onto a tape via a telecine machine (see variants d and e).

The project may conclude with an optional release on a new medium (e.g. DVD or digital stream), which follows the finishing on tape or film (for example, in variant b the release on new medium follows a tape finish).

6. PROCESS MODEL CUSTOMIZATION APPROACHES

In this section we evaluate the 19 approaches for representing business process variability that we identified via the literature search described in Section 3. We sort the approaches per year of primary publication. First, we review the eleven main approaches. Next, in a final subsection, we review the eight subsumed approaches.

We use the 14 criteria described in Section 4 to assess each approach. Specifically, we refer to each criterion by its short identifier as introduced in Section 4, and use a "$+$" to denote full support for a criterion, "$\pm$" for partial support and "$-$" for no support. For example "CS.Beh:$\pm$" denotes partial support for the criterion "Behavioral correctness". The results of the assessment are summarized in Table I in Section 9.

The evaluation is complemented by an overview of techniques for correctness support (Section 7) and for decision support (Section 8) during process customization.

6.1. C-EPCs: Configurable Event-driven Process Chains

Configurable EPCs (C-EPCs) [Rosemann and van der Aalst 2003; Dreiling et al. 2005; Dreiling et al. 2006; La Rosa et al. 2011] are an extension of the EPC language. A C-EPC model represents the least common multiple of a family of EPC variants. Differences among the various process variants are indicated by a set of variation points (called *configurable nodes*). Each configurable node can be assigned a set of *configuration alternatives*, each referring to one or more process variant. Configuration is achieved by restricting the behavior of the C-EPC by assigning one alternative to each configurable node. Then the configured C-EPC can be post-processed by removing all

those alternatives that are no longer relevant, in order to obtain a regular EPC.[2] By doing so, one can derive the initial variants of the given process family. Thus, configuration is achieved by restriction (IT.Res:+).

The active nodes of an EPC's control flow, i.e. functions and connectors, can be marked as configurable with a thicker border (P.Cf:±). Figure 5 shows the C-EPC model for the post-production example, which captures all variants of Figure 4. In this model there are four configurable functions and six configurable OR connectors.

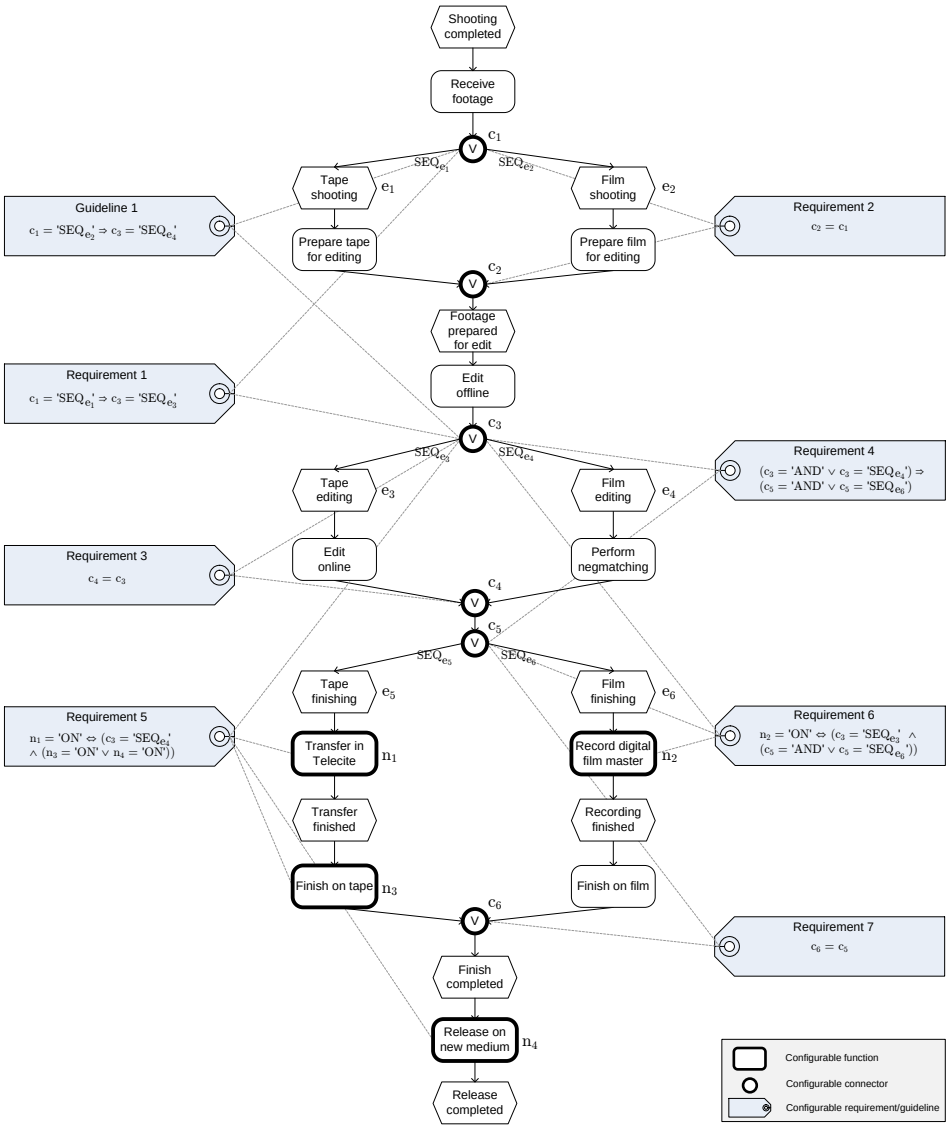

Fig. 5: The C-EPC model representing all post-production variants.

[2]This post-processing step where irrelevant parts are removed is called *individualization*.

Configurable connectors can be configured to an equally or more restrictive connector. In other words, the model resulting from a configuration should have the same or less execution traces than the original model. A configurable OR can be left as a regular OR (no restriction is applied), or restricted to an XOR, to an AND or to one of its outgoing sequences of nodes (if the connector is a split), or to one of its incoming sequences of nodes (if a join). For example, we can capture the choice of the shooting medium by configuring the first OR-split in Figure 5. We can restrict this connector to its left-hand side branch if the choice is tape. As a result the branch starting with event "Film shooting" is removed. Restricting the connector to its right-hand side branch results in the branch starting with "Film shooting" being removed. Restricting the connector to an AND-split ensures that both media are prepared for editing. In the three cases above, we anticipate the decision of the medium at configuration-time. Alternatively, by configuring this connector to an (X)OR-split, we postpone the decision till run-time, when the post-production process is actually enacted (see e.g. variant f in Figure 4). The alternatives for a configurable XOR are a regular XOR or the restriction to one of its outgoing/incoming branches. For symmetry, an AND connector can also be set as configurable, but it can only be mapped to a regular AND, thus no actual configuration is achieved in this case.

Besides configurable connectors, functions can also represent variation points. Configurable functions have three alternatives: ON, OFF or OPT. The first two alternatives allow one to represent variants in which the given function is present, resp., absent. The OPT alternative permits the deferral of this choice until run-time. For example, function "Release on new medium" is configurable in Figure 5, so we can switch it OFF for those projects where this is not required.

Configuration requirements formalize domain constraints over the alternatives of configurable nodes and are enforced during configuration, to rule out unfeasible configurations. Requirements are expressed as logical predicates over configuration alternatives, and depicted as tags attached to the involved nodes. For example, we said that negmatching cannot be performed if the project is not shot on film. So in the C-EPC model function "Perform negmatching" is only relevant if function "Prepare film for editing" is chosen at the first OR-split, otherwise it must not be performed. This is enforced by Requirement 1 in Figure 5. It is also possible to specify *configuration guidelines* in order to aid the configuration process. For example, Guideline 1 suggests to perform negmatching only if the project is shot on film. In fact, although online editing is still possible, in this case negmatching offers the best results as the cut is directly done on the film roll. Only requirements are mandatory and must hold in order for a configuration to be *valid*.

Requirements and guidelines offer rudimentary configuration guidance to the user, since they do not suggest which choices should be taken based on domain conditions, e.g. a high-level budget project. Moreover, they do not offer any form of abstraction for configuring a C-EPC. Rather, they clutter the model with low-level details. An alternative solution is offered by the use of questionnaire models, which can be linked to C-EPCs to provide abstraction and guidance during configuration (DS.Abs:+). Questionnaire models are described separately in Section 8.2.

EPCs are a conceptual process modeling language. Support for the execution of configured process models is thus not provided (PT.Con:+, PT.Exe:−).

The algorithm to individualize a C-EPC into a regular EPC is formalized in [Rosemann and van der Aalst 2007]. If the C-EPC is syntactically correct, this algorithm preserves the syntactic correctness of the individualized models by removing all nodes that are no longer connected to the initial and final events via a path, and by reconnecting the remaining nodes (CS.Syn:+). Behavioral correctness is ensured via the use of a technique based on constraints inference, described in Section 7.1 (CS.Beh:+).

There are several extensions to the original C-EPC specification. In [Dreiling et al. 2005], new language elements are provided. For example, it is possible to represent critical configuration decisions, mutual and default alternatives, and interleaved parallel routing of functions. In [Dreiling et al. 2006] configurable connectors can also be restricted to any subset of their incoming or outgoing sequences of nodes, rather than a single sequence of nodes. For example, in this extension an AND-split can be configured to a regular AND connector but with a restricted set of outgoing arcs.

The most significant extension is represented by the C-iEPC (Configurable integrated EPC) language [La Rosa et al. 2011]. C-iEPCs enable the specification of variation points in the classes of organizational resources (called *roles*) and objects involved in a business process (P.Re:+, P.Ob:+).

Roles can be human (e.g. a financial officer) or non-human (e.g. an information system). Objects capture information artifacts (e.g. a file) or physical artifacts (e.g. a paper document or a production material). Objects can be used as input or output to functions, and can also be consumed, i.e. destroyed, when used as input. Roles and objects can be optional and can be connected to functions via logical connectors, called *range connectors*. Range connectors subsume the three logical types of OR, XOR and AND, but also allow wider ranges (e.g. at least 2 and at most 5 roles). Range connectors can also be optional. In this case they indicate that all connected roles/objects are optional.

Figure 6 depicts the "Edit offline" function in C-iEPCs. This function is performed by at least 2 roles, requires a Temp picture as input and produces an Edited picture. Editing notes are optional, since they may not be produced during the offline editing.

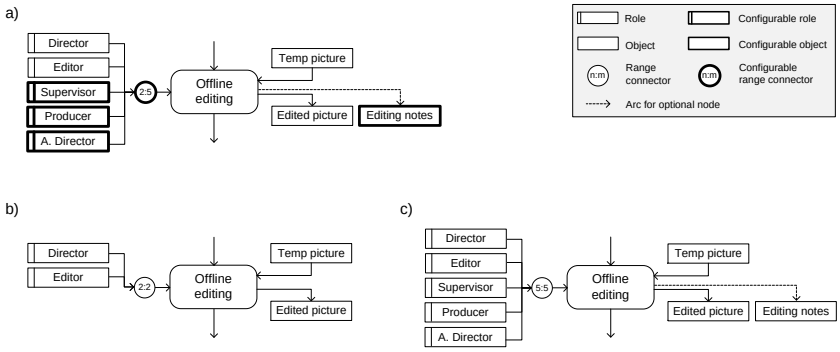

Fig. 6: An example of a C-iEPC with configurable roles and objects (a). Two possible individualizations for low-budget productions (b) and high-budget productions (c).

Roles, objects and range connectors can be made configurable. For example, in Figure 6 three roles, one object and one range connector have been marked as configurable. Let us briefly describe how these elements can be configured. If a role, object or range connector is optional (OPT), it can be configured to mandatory (MND) or switched OFF, while if it is mandatory, it can only be switched OFF. Further, configurable roles and objects can be specialized to a sub-type (e.g. a role Producer can be specialized to an Executive Produced) according to a hierarchy model which complements the C-iEPC model. Configurable input objects that are consumed (CNS) can be restricted to used (USE), in which case they are not destroyed once the respective function completes. The routing behavior of configurable range connectors can be restricted to smaller ranges (e.g. a 2:5 range can be restricted to 3:4).

Figure 6 illustrates two possible configurations for low- and high-budget productions. In the first one, the range connector has been restricted to a fixed range of 2:2 and all configurable roles and objects have been switched OFF. In the second variant, everything has been kept ON and the range connector has been configured to its wider interval.

C-(i)EPCs are formalized [Rosemann and van der Aalst 2003; La Rosa et al. 2011] (EF.For:+), and implemented in the *Synergia* toolset[3] (EF.Imp:+). This toolset provides support for designing C-(i)EPCs, linking these models to questionnaire models, configuring them via questionnaires and individualizing the resulting configured models.

The use of C-(i)EPCs, including decision support via questionnaire models, has been validated in case studies in the film industry involving domain experts [La Rosa et al. 2011] (EF.Val:+).

6.2. Configurative Process Modeling

Configurative process modeling [Becker et al. 2004; Becker et al. 2007a; Delfmann et al. 2006; Delfmann et al. 2007; Becker et al. 2006; Becker et al. 2007c; Becker et al. 2007b] is an approach that relies on the principle of *model projection* (also called *configurative adaptation*) to customize reference process models. The starting point is the assumption that "reference" process models typically contain information for multiple application scenarios.

A projection of a reference process model for a specific scenario is obtained by fading out those model elements that are not relevant to the selected scenario (IT.Res:+). This method is called *element selection*. *Configuration parameters* are used to determine the available application scenarios, and later to drive the customization process. These are divided into *Business characteristics* and *perspectives*.[4] Business characteristics and their *values* identify companies or company-specific aspects, whereas perspectives identify requirements for different classes of users.

For example, in the case of post-production, we can identify the business characteristic "Shooting type" (ST), with values: "Tape shooting" (T) or "Film shooting" (F), and "'Budget Level" (BL), with values: "Low budget" (L), "Medium budget" (M) or "High budget" (H). This is a high-level characteristic since a choice on the budget typically affects multiple decisions in post-production. Configuration parameters like these are linked to the elements of a process model by means of simple attributes or logical terms over parameters. The elements of a process model which are linked to configuration parameters are thus the parts of the model that can vary, although these are not explicitly represented in the process model like in C-EPCs. The language chosen to capture process models is (plain) eEPCs. eEPCs (extended EPCs) [Davis and Brabander 2007] are an extension of EPCs which is supported by the ARIS toolset. Similar to iEPCs, eEPCs enrich the EPC meta-model with further elements to capture organizational resources, input and output objects, products, etc., that are involved in a business process. Specifically, configuration parameters can be linked to functions, events, organizational resources and objects of an eEPC. Connectors cannot be directly configured (P.Cf:±, P.Re:+, P.Ob:+).

Figure 7 shows a reference model for post-production in eEPCs, where some elements are associated with a logical term referring to the project's budget (the control flow of this model is the same as that of the C-EPC model of Figure 5, except for the absence of configurable nodes). For instance, event "Film editing" and function "Perform negmatching" are linked to the term BL(H), meaning that these elements are not suitable for low and medium budget projects, due to the high costs involved in editing

[3] See www.processconfiguration.com

[4] not to be confused with the process perspectives presented in Section 4.

on film. On the other hand, function “Edit online” is not associated with any term, since it is suitable for any type of budget.

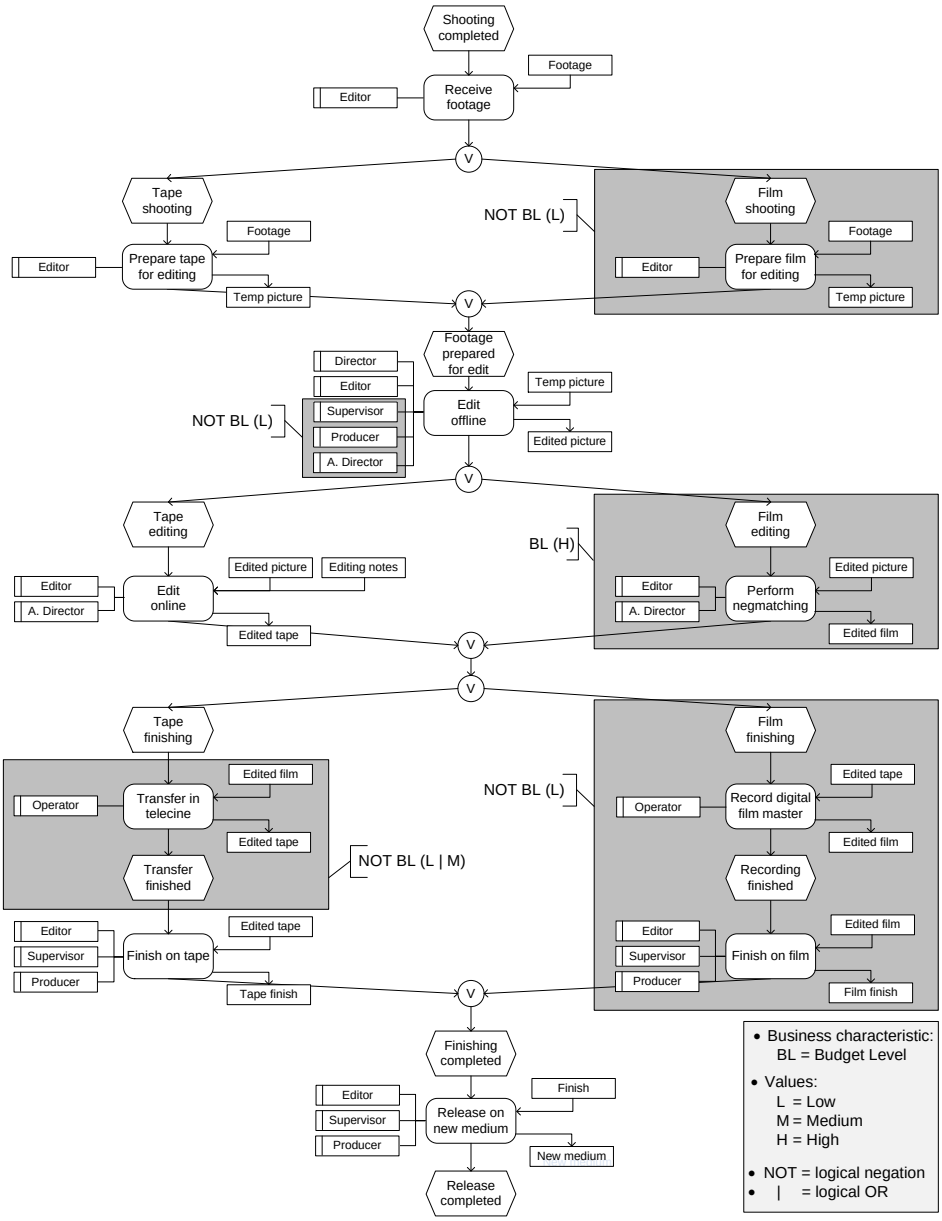

Fig. 7: A reference process model for post-production in eEPCs with logical terms for the budget levels.

The projection of a process model to a specific scenario is done by marking those elements whose parameters evaluate to false as hidden. Then an individualization is

performed to remove the hidden elements and reconnect the remaining nodes. If we individualize the example model in Figure 7 for a low budget project, we obtain variant b in Figure 4. The individualization can fix simple syntactic issues, e.g. the removal of connectors which have one input and one output arc, as in the example, but cannot ensure the syntactical and behavioral correctness of the resulting models (CS.Syn:$\pm$, CS.Beh:$-$). For example, since both events and functions can be removed, the algorithm does not work in the context of cycles, or when a function between two events is removed. Syntactic issues that cannot be fixed like the above ones, are prompted to the modeler who is demanded to fine-tune the configuration.

Configuration parameters can also be applied to the meta-model layer (i.e. to the eEPC meta-model), to remove process modeling perspectives that are not relevant to a specific scenario. This method is called *element type selection*. For example, it is possible to hide the resource perspective in the eEPC meta-model. In this way all the resources associated with functions are removed from the model. Besides configuration parameters, one can change specific terms in the labels of modeling elements, or change the modeling language to adapt it to a specific class of users.

Customization is not carried out at the process model level, but via the evaluation of a set of configuration parameters (DS.Abs:$+$). However the approach does not offer guidance to users when assigning values to the configuration parameters (DS.Gui:$-$).

The authors acknowledge that not every possible configuration requirement can be anticipated by developers of reference models. Thus, they envisage that further (company-specific) customizations may need to be performed once a reference process model has been configured. For this reason, besides customization by restriction, the configurative process modeling approach provides a set of *generic adaptation mechanisms* that can used to refine, and possibly extend, the behavior of a configured reference process model (IT.Ext:$+$). These are:

— *Aggregation*: the configured reference process model can be enriched with new model fragments. These fragments can be made available as generic model components in a repository. Since they are generic, they first need to be adapted to the specific scenario in question, and then combined with the configured reference process model. The possible combinations of components can be restricted by interface definitions.
— *Instantiation*: the abstract parts of a component can be concretized via the insertion of feasible values into placeholders that are provided within the component. Instantiation can be applied at the level of entire model sections, individual model elements or attributes (e.g. a simple numeric value).
— *Specialization*: component adaptation can also be achieved by adding, removing or changing model elements without any semantic restrictions.
— *Conclusion by analogy*: envisions the reuse of model structures found in the reference process model by the user.

Instantiation and specialization can also be applied directly to the configured reference process model, rather than to a generic component. The configurative process modeling approach described in [Becker et al. 2004; Becker et al. 2007a] mainly focuses on model projection, and the generic adaptation mechanisms described above are only sketched. Besides simple controlling, the application of these mechanisms is left to the user without specific support.

The model projection mechanism has been implemented as a prototype tool [Delfmann et al. 2006] that interacts with the ARIS platform (EF.Imp:$+$). Configuration parameters can be defined and linked to elements of an eEPC, which can be designed in ARIS. Then an interface allows users to select the desired parameters and a projection is performed on the initial eEPC to remove irrelevant elements.

The configurative process modeling approach has been applied to the fields of method engineering [Becker et al. 2007c] and change management [Becker et al. 2007b], and validated in the German public administration sector [Becker et al. 2006], though without involving domain experts (EF.Val:±). The approach is not formalized (EF.For:−).

6.3. PESOA: Process Family Engineering in Service-Oriented Applications

The idea of capturing variability in process models has also been explored in the PESOA (Process Family Engineering in Service-Oriented Applications) project [Puhlmann et al. 2005; Schnieders and Puhlmann 2006; Schnieders 2006]. The aim of this project is not to provide a language for representing and customizing process models, rather to improve the customization of process-oriented software systems, i.e. systems that are developed from the specification of process models. If the variability of a software system can be directly represented in a process model that describes the system's behavior, it is then possible to generate code stubs for the system from the customization of the process model itself. Since code generation is outside the scope of this paper, we only focus on the way the authors represent process variability.

According to PESOA, a *variant-rich process model* is a process model extended with stereotype annotations to accommodate variability. Although stereotypes are an extensibility mechanism of UML, in this approach they are applied to both UML ADs and BPMN models. The places of a process model where variability can occur are marked as variation points with the stereotype `<<VarPoint>>`. These can be actions in UML ADs and tasks in BPMN. A variation point represents an abstract action (task), such as "Prepare medium for editing", that needs to be realized with a concrete variant (`<<Variant>>`) among a set of possible ones. For example, "Prepare medium for editing" can be realized with the variant "Prepare tape for editing", or with "Prepare film for editing", or with both of them. One can also annotate the default variant for a variation point with the stereotype `<<Default>>`. Figure 8.a shows the process model for post-production in annotated BPMN, where some variation points have been identified. For example, "Prepare tape for editing" is annotated as the default variant of "Prepare medium for editing", as this is the most common choice in post-production.

If the variants are exclusive, i.e. if only one variant can be assigned to a given variation point, the stereotype `<<Abstract>>` is used instead of `<<VarPoint>>`. In Figure 8.a we assume that the variants "Edit online" and "Perform negmatching" are exclusive, so the relative variation point "Cut picture" is annotated with the tag `<<Abstract>>`. As a shortcut, when the variants are exclusive, the default resolution can be depicted directly on the variation point with the stereotype `<<Alternative>>`.

A variation point annotated with the stereotype `<<Null>>` indicates optional behavior. It can only be associated with one variant and its resolution is not mandatory. This is the case for the variation point "Transfer tape to film" that may be resolved with the variant "Record digital film master", or be completely dropped from the process model. A shortcut for a `<<Null>>` variation point and its variant is to directly depict the variant on the variation point, using the stereotype `<<Optional>>`. This is the case for task "Transfer in telecine", which subsumes the variation point "Transfer film to tape".

Annotations can only be applied to BPMN activities or actions in UML ADs, as well as to the data objects connected to these elements. For example, an input data object "Tape" to task "Prepare tape for editing" can be realized with either "Digital tape" or "Analog tape" (not shown in Figure 8). However, control-flow gateways and process resources cannot be customized (P.Cf:±, P.Re:−, P.Ob:+).

Through a configuration each variation point is realized with one or more variants according to its type. Figure 8.b shows a fragment of the BPMN process model for post-

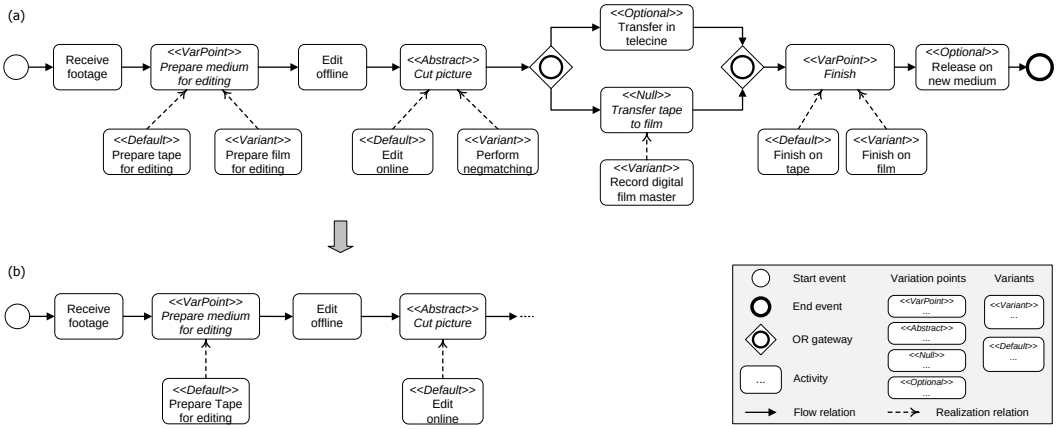

Fig. 8: (a) The picture post-production process model in annotated BPMN. (b) A configured process model.

production configured for a project shot on tape and edited online. In this model the variants that are not required have been removed (IT.Res:+).

Abstraction from the process modeling language is achieved by linking process variants with features from a feature model (see Section 8.1). This is done by tagging each process variant with the name of a feature, such that when a feature is disabled in a feature model configuration, the corresponding variant is removed from the process model (DS.Abs:+).

Constraints over feature values can be used to capture domain requirements, thus restricting the possible combinations of variants in the process model. However, there is no guidance for the selection of a suitable set of features (DS.Gui:−).

Given the objective of producing a software system, an individualization algorithm for transforming variation points to selected variants is out of the scope of this approach, and process models are only considered at a conceptual level (PT.Con:+, PT.Exe:−). Further, the absence of an individualization algorithm leaves room for interpretation. For example, it is not clear how model elements have to be reconnected after removing a `<<Null>>` or `<<Optional>>` variation point, nor is it clear how a variation point should be transformed when multiple variants are selected. These transformations may lead to correctness issues which have not been taken into account. A formalization is provided for selected concepts only [Puhlmann et al. 2005] (EF.For:±).

The approach has been implemented as an Eclipse plugin. This tool provides support for configuring a feature model and for applying the results to the underlying process model. However, since an individualization algorithm has not been defined, the configuration of a process model is limited to the removal of the undesired variants (EF.Imp:±).

PESOA has been validated in the hotel booking domain, in collaboration with ehotel and Delta Software Technology [Schnieders and Puhlmann 2006]. In this study, a set of BPMN process models were configured to drive the generation of web applications in collaboration with domain experts (EF.Val:+).

6.4. Superimposed Variants

The idea of annotating model elements to represent variability has also been investigated in [Czarnecki and Antkiewicz 2005; Czarnecki et al. 2005]. In this approach, any control-flow element of UML ADs can be annotated using *presence conditions* (PCs)

and *meta-expressions* (MEs). A PC indicates if the model element it refers to should be present or be removed (IT.Res:+). MEs are used to compute attributes of model elements relevant to the UML language (e.g. the name of an action). These attribute variations span a finite range of options.

Both PCs and MEs are captured by boolean formulae over the features and feature attributes of a feature model (see Section 8.1), and are evaluated against a feature configuration. These formulae can be represented in disjunctive normal form or as XPath expressions. UML stereotypes are used to create annotations for these formulae to be assigned to model elements. For example, the stereotype <<Tape ∨ Film>> indicates the disjunction between the two features "Tape" and "Film", while the stereotype <<PC>> indicates an XPath expression, where the expression is specified as a String property of the stereotype itself. The assignment of stereotypes to modeling elements is done through rendering mechanisms such as labels, color schemes or icons.

Figure 9.a shows the finishing phase of the picture post-production process as an annotated UML AD; for simplicity, in this example we have only specified PCs. These PC annotations, rendered with a color and a number in the example, have been defined over the features of the feature model of Figure 17 (see Section 8.1). Essentially, this feature model captures the features (i.e. properties) of the post-production domain, such as the type of finish and the type of transfer. For instance, action "Transfer in telecine" is associated with the sub-feature "Telecine" of feature "Transfer" (annotated in blue with label "1"), while the two outgoing flows of the decision point are associated with the two sub-features of "Finish": "Tape", resp., "Film". All the non-labeled elements (in black) are associated with the *always-true* formula. These represent the commonalities of the model and as such cannot be removed, e.g. the decision node and the ActivityFinal element.

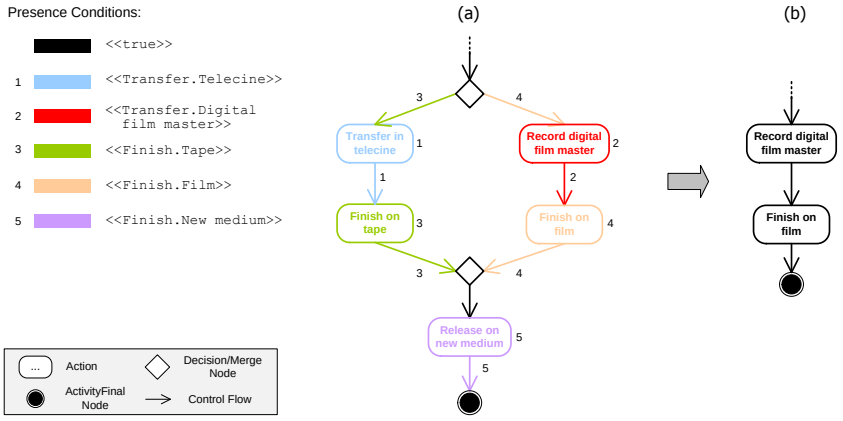

Fig. 9: (a) an annotated UML AD modeling an extract of the post-production process; (b) a possible individualization of this model.

Process configuration is achieved by evaluating PCs and MEs against a feature configuration (DS.Abs:+). Those model fragments whose PCs evaluate to false are removed from the model, while those model attributes that are affected by MEs are changed accordingly (e.g. an action name is changed). No guidance is provided for the selection of the features to be kept (DS.Gui:−).

Figure 9.b shows a possible individualization for the post-production example where only the actions "Record digital film master" and "Finish on film" have been kept. This

model can be obtained through an individualization algorithm which applies *patches* to reconnect modeling elements that have been disconnected during configuration, and *simplifications* to remove splits and joins that have been left with one inflow and one outflow. However patches are only applied to those nodes that have exactly one inflow/outflow, and an annotation error is raised otherwise. Therefore, syntactical correctness is only partly addressed (CS.Syn:±); behavioral correctness is not dealt with (CS.Beh:−).

UML ADs are not an executable language, thus the individualized models cannot be executed (PT.Con:+, PT.Exe:−). The approach only supports configuration of control-flow elements (P.Cf:+). Resources and objects cannot be configured (P.Re:−, P.Ob:−).

The approach has been formalized (EF.For:+) and implemented in an Eclipse plugin, namely *fmp2rsm*[5] (EF.Imp:+). This plugin allows users to configure UML ADs via cardinality-based feature models [Czarnecki et al. 2005] (a special type of feature models). The approach has not been validated in practice (EF.Val:−).

6.5. Configurable Workflows

The Configurable Workflows approach [van der Aalst et al. 2006; Gottschalk et al. 2007; Gottschalk et al. 2008] proposes to customize process models by applying two operators, namely *hiding* and *blocking*, to process model activities, with the purpose of reducing the process behavior (IT.Res:+). Hiding corresponds to *abstraction*, i.e. the execution of an activity becomes unobservable, but the path the activity is in, is still possible. Blocking corresponds to *encapsulation*, i.e. the execution of an activity is disabled. The path from the activity cannot be taken anymore and the process flow will never reach a subsequent state.

This approach was first designed for conceptual models [van der Aalst et al. 2006] (PT.Con:+), and later applied to executable languages, with the aim to guarantee that the individualized process models can be executed (PT.Exe:+). This led to the extension of several executable languages, such as SAP WebFlow [Gottschalk et al. 2007], YAWL and BPEL [Gottschalk et al. 2008]. In this survey we focus on the extension to the YAWL language, namely Configurable YAWL (C-YAWL), since this is the most significant extension. The extensions to the other languages work in a similar way.

In YAWL, splits and joins are attached to tasks: a join precedes a task and models the task's joining behavior; a split follows a task and models its splitting behavior. C-YAWL extends YAWL with *ports* to capture variation points, which are applies to splits and joins. A task's join has an *inflow port* for each combination of arcs through which the task can be triggered, whilst a task's split has an *outflow port* for each combination of subsequent arcs that can be triggered after the task's completion. For example, let us consider the case of a join with two incoming arcs and a split with two outgoing arcs. If the join (split) is of type XOR, it has two ports. This is because an XOR-join can be activated by each incoming branch, while an XOR-split will give control to one of its outgoing branches. If the join (split) is of type AND, it only has one port. In fact, an AND-join is activated when receives control from both the incoming branches, while an AND-split simultaneously givens control to all its outgoing branches. Finally, if the join is of type OR, it has one port, as the OR-join is considered as an AND-join from a configuration perspective, due to its synchronizing merge behavior. On the other hand, an OR-split has one port for each combination of its outgoing arcs, as it can give control to any such combination.

Hiding and blocking are applied to inflow and outflow ports. An inflow port can be configured as *blocked* to prevent the triggering of its task, or as *hidden* to skip the task's execution without blocking the subsequent tasks. An inflow port that is neither

[5]See `http://gp.uwaterloo.ca/fmp2rsm`.

blocked nor hidden, is said to be *allowed*. An outflow port can only be blocked to prevent the triggering of the outgoing branches, or left allowed. In C-YAWL all the ports are configurable. This provides full customization of the control flow (P.Cf:+). On the other hand, resources and objects cannot be customized (P.Re:−, P.Ob:−).

Figure 10.a depicts the post-production process in C-YAWL where for illustration purposes, we modeled the preparation and editing of tape and film as mutually exclusive activities. This figure also shows a sample port configuration for a project shot on tape, edited online and finished on film, overlaid on the C-YAWL model.

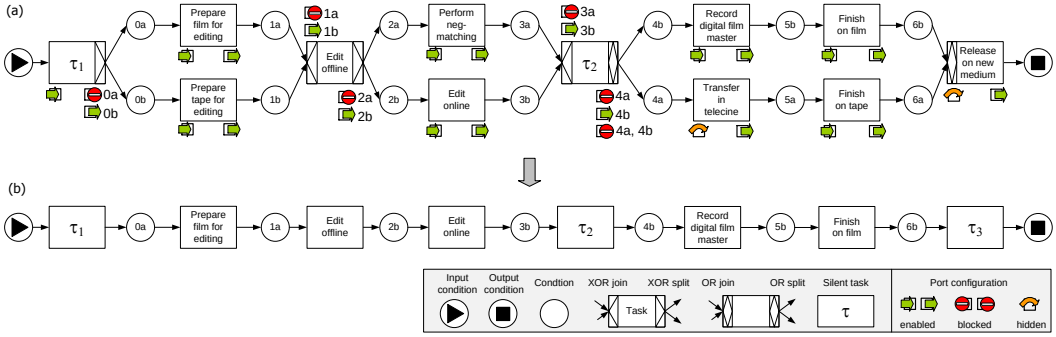

Fig. 10: (a) the post-production process model in C-YAWL with a sample configuration overlaid; (b) the individualized YAWL model.

For example, the first task of the model, τ_1, is used to route the process flow according to the shooting media. This task has only one incoming arc from the input condition. Thus, its join has only one inflow port which always needs to be enabled (in YAWL, a task with no join/split has an XOR behavior by default). The task's XOR-split has two outflow ports: one to trigger the path to condition $0a$ (leading to the preparation of the film), the other to trigger the path to condition $0b$ (leading to the preparation of the tape). Of the two, the only port allowed by the example configuration is the one that leads to task "Prepare tape for editing". The inflow port from condition $1a$ to the XOR-join of task "Edit offline" is configured as blocked whereas the other inflow port for this join is allowed, to match the configuration of the preceding XOR-split. Since the project is edited online, the outflow port of task "Edit offline" triggering condition $2b$ is the only one to be allowed.

The hiding and blocking operators can also be applied to other YAWL elements such as cancelation regions, composite tasks and multiple-instance tasks. Specifically, one can block the cancelation region of a task, restrict the number of implementations (called *worklets*) assigned to a composite task by blocking them, and restrict the values of the parameters of a multiple instance task, e.g. by decreasing the maximum number of allowed instances.

Figure 10.b shows the YAWL model resulting from the example configuration, after applying the individualization algorithm formalized in [Gottschalk et al. 2008]. This algorithm removes all nodes that after configuration are no longer on a path from the input to the output condition. In this way the syntactical correctness of the model is guaranteed. Moreover, potential conflicts in the data conditions of the outgoing arcs of (X)OR-splits are taken care of, in order for the resulting models to be fully executable (CS.Syn:+). This approach benefits from two different techniques for ensuring the behavioral correctness of the resulting models: Constraints Inference and Partner Synthesis, both described in Section 7 (CS.Beh:+).

In C-YAWL, domain requirements can be defined to restrict the values each port can take based on the configuration of other ports. These requirements are expressed as boolean conditions over port configurations, similar to the configuration requirements in C-EPCs. For example, with a requirement one can bind the outgoing ports of tasks τ_1 and "Edit offline", by implying that task "Prepare film for editing" is allowed if task "Perform negmatching" is allowed. Abstraction and guidance are both supported (DS.Abs:+, DS.Gui:+) via the use of questionnaire models (see Section 8.2).

The C-YAWL approach has been formalized [Gottschalk et al. 2008] (EF.For:+). The *YAWL Editor*[6] allows one to create C-YAWL models, and to configure and individualize them, while support for configuration via questionnaire models is provided by the *Synergia* toolset (EF.Imp:+).

The use of C-YAWL models, and their configuration via questionnaire models, have been validated in the municipality domain [Gottschalk et al. 2009; Lönn et al. 2012], involving consultants and subject-matter experts, as well as in software development processes for very small entities [Boucher et al. 2012] (EF.Val:+).

6.6. ADOM: Application-based Domain Modeling

The ADOM (Application-based Domain Modeling) approach [Reinhartz-Berger and Sturm 2007; Reinhartz-Berger et al. 2009; 2010] proposes a three-level architecture to customize process models. The first level (*language*), hosts the meta-models that can be used to describe business process models, e.g. in EPCs. The second level (*domain*), hosts the customizable "reference" process models which serve as templates for a particular domain, e.g. logistics. Finally, the last level (*application*), hosts the customized process models for specific companies, which can be directly derived from the reference process models at the domain level.

In order to capture variability, cardinality attributes of type `<min,max>` are used to annotate the elements of a reference process model, i.e. activities, events, connectors and arcs. These attributes specify the number of instantiations that are available for a given element, i.e. how many times an activity can be used in a customized process model. For example, an activity tagged with `<0,1>` is optional, and as such it can be dropped in the customized model; an activity tagged with `<1,n>` is mandatory and can be instantiated up to n times in the customized model; an activity tagged with `<1,1>` must be instantiated exactly once. The default cardinality `<0, n>` implies no constraints, and as such it is not represented in the reference process model. The cardinality assigned to connectors of type (X)OR indicates when the decision represented by the connector should be made. Specifically, a cardinality of `<0,0>` indicates that the connector must not appear in the customized model and so its decision must be taken at customization time. Moreover, during customization, the type of an OR connector can be restricted to XOR or AND, in the same way as in C-EPCs.

In ADOM, commonalities are thus captured by the mandatory elements while variability is captured by the optional elements and by those which can be instantiated multiple times. Since an ADOM reference process model is meant to be used as a template, it does not need to be a complete specification, i.e. some parts can be intentionally underspecified. During *specialization*, i.e. when a reference process model is customized, each annotated element can be instantiated according to its cardinality constraint. Moreover, *application-specific* elements can be added, e.g. to extend underspecified parts: these elements will only appear in the customized process model without any counterpart in the reference process model. However, there is no control over which parts of the reference process model can be extended and how. In other

[6] See `www.yawlfoundation.org`

words, one could virtually change the whole reference process model, provided that the cardinality constraints are met.

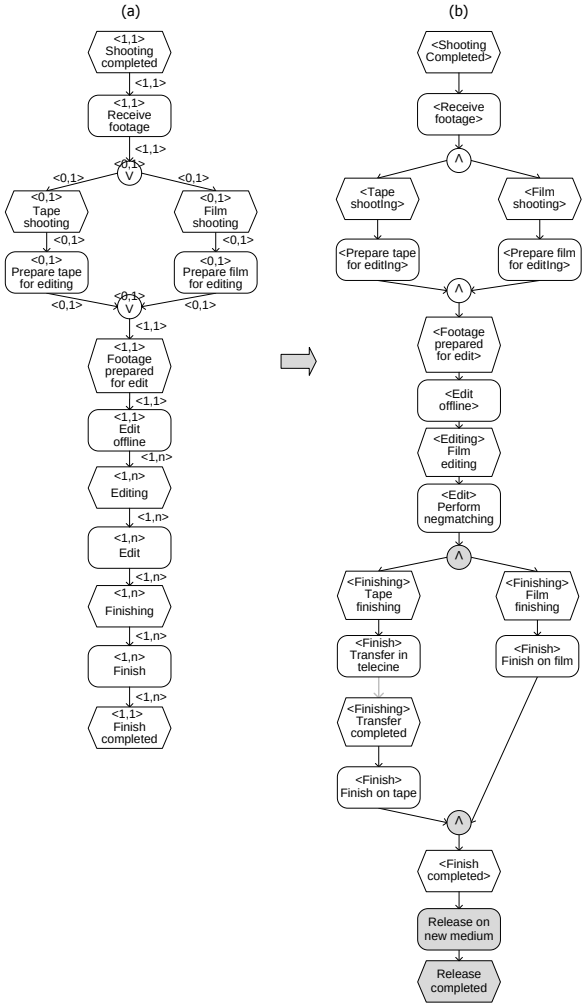

Fig. 11: (a) A reference process model for post-production in ADOM-EPC. (b) A possible specialization of this model.

Figure 11.a shows an EPC reference process model for post-production annotated with ADOM cardinality constraints. For example, event “Shooting completed”, function “Receive footage” and the arc in-between denote commonalities, since they can neither be removed nor instantiated more than once during customization. The OR-split and its matching OR-join are optional, and so are the nodes in-between. This is done to allow a choice between either of the two branches or both. All elements after function “Edit offline” are mandatory but have a maximum cardinality greater than 1. By doing so, each of these elements can be instantiated multiple times to model the various options that exist for editing and finishing in post-production.

In a specialized process model, each element that has been derived from an annotated element in the reference process model bears a *reference model classifier*, i.e. an annotation pointing to the element in the reference process model this element originates from. If the name of the element needs to be changed, i.e. a more specific one must be used, this can be added below the reference model classifier. Figure 11.b shows a possible specialization of the post-production reference process model, where application-specific elements are highlighted in gray. For example, the first AND-split/join has been obtained by restricting the type of the respective OR-split/join to an AND. Event "Film editing" and function "Perform negmatching" are specializations of event "Editing", resp., function "Edit", and have been given each a new name. The second set of AND-split/join and the arc between function "Transfer in telecine" and event "Transfer completed", are application-specific elements which have been added to allow multiple instantiations of event "Finishing" and function "Finish". We observe that we could add any arbitrary control-flow structure, provided that we met the multiplicity constraints for these elements.

ADOM has been applied to reference process models described in UML ADs, BPMN and EPCs. For the latter language, specific rules have been defined to bind the specialization of an event to that of a function, in order to maintain the alternation between events and functions required by EPCs. A validation technique for ADOM-BPMN is described in [Reinhartz-Berger et al. 2009]. This technique can check the compliance of a customized process model with its reference process model. Beyond this, syntactical and behavioral correctness of the customized models is not guaranteed (CS.Syn:−, CS.Beh:−).

In ADOM, process customization is achieved by both behavioral restriction, through removing optional elements (IT.Res:+) and extension, through instantiating elements with multiple cardinality, and adding application-specific elements (IT.Ext:+). Customization is performed directly at the model level. Moreover, there is no means to specify which combinations of instantiations are unfeasible from a domain point of view, and the addition of application-specific elements cannot be constrained. Thus, no decision support is provided (DS.Abs:−, DS.Gui:−).

A formalization of ADOM is provided in [Reinhartz-Berger et al. 2009] for BPMN and in [Reinhartz-Berger et al. 2010] for EPCs (EF.For:+), though the approach has not been implemented in a tool (EF.Imp:−). In both variants of ADOM, process models are only considered at a conceptual level, i.e. the individualized models cannot be executed (PT.Con:+, PT.Exe:−). A subset of ADOM-BPMN has been validated in the development of a process-driven service-oriented system, though without involving domain experts [Reinhartz-Berger et al. 2009] (Ef.Val:±).

6.7. BPFM: Business Process Family Model

The Business Process Family Model (BPFM) [Moon et al. 2008] is a two-level approach to capture customizable business process models using an extended version of UML ADs. The first level deals with basic customization. At this level, an action can be defined as *common* (mandatory) or *optional* (can be omitted during customization). The second level defines a more fine-grained customization by marking an action as a *variation point* and assigning it one or more variants.

Specifically, a variant can be an atomic action or a sub-process (called activity in UML ADs) while a variation point can either be *boolean*, *selection*, or *flow*. A boolean variation point requires exactly one variant to be selected. A selection variation point requires at least one variant to be selected. In this case, the exact number of variants to be selected can be specified with a cardinality. For example, a cardinality of `1..2` indicates that at least one and at most two variants can be selected for the variation point. When selecting more than one variant, one needs to specify the control-flow

relation between the selected variants (called *flow pattern*). This can be a *sequence* (all selected variants are ordered sequentially), *parallel* (the variants are executed in parallel using a fork and a join node) or *decision* (they are made mutually exclusive via the addition of a decision and a merge node). A flow variation point is assigned a *variants region*, i.e. a set of actions whose flow relations may be underspecified. At customization time, one needs to restrict the behavior between these actions, by adding the required flow relations. A flow pattern can also be specified for the flow variation point, in which case the actions inside the variants region are organized according to the pattern, though the precise order needs to be decided by the user at customization time.

Further, the boundary of a variation point can be classified as either *open* or *closed*. A closed boundary type restricts the choice of variants to those already identified whereas an open boundary type allows the introduction of new variants during customization. Thus, in principle this approach supports both customization by restriction and extension (IT.Res:+, IT.Ext:+). However, there is no concrete support for plugging in new variants into a variation point.

An example of the use of these constructs is illustrated in Figure 12.a. Here there are three actions marked with a variation point and one optional action. Actions "Prepare medium for editing" and "Cut picture" are of type selection decision. They have been assigned two variants each. The first action prescribes a parallel pattern while the second a sequence pattern, each with the option of selecting at least one and at most two variants. Accordingly, Figure 12.b shows an example customized model where the first variation point has been customized to the parallel execution of both its variants, whilst the second variation point has been customized to a sequence of its variants. Action "Transfer & finish" is an open variation point of type flow decision which prescribes a decision pattern between the actions in its associated variants region. Accordingly, in Figure 12.b this variation point has been customized to a decision between two branches, each hosting two of the four actions present in the respective variants region. We observe that in case of an open flow variation point, the arrangement of the actions inside the variants region within a given control-flow structure is entirely left to the modeler. Finally, in our example customization action "Release on new medium" has been switched off.

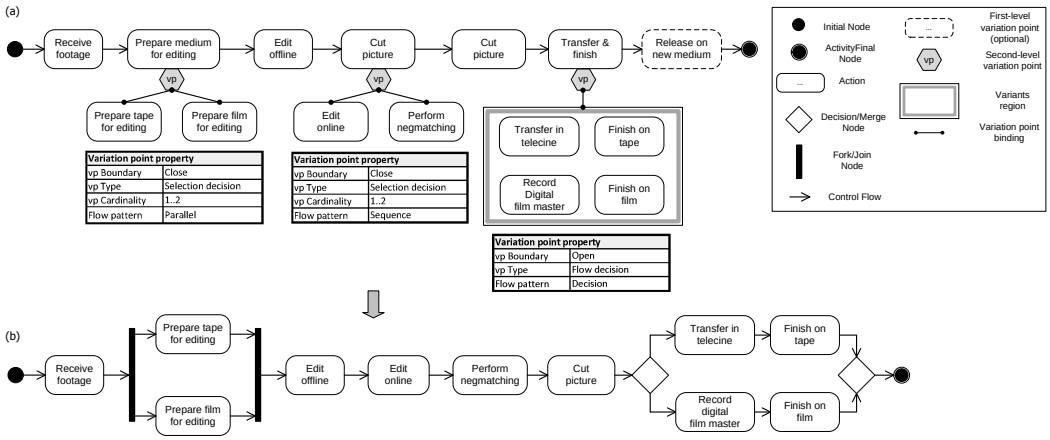

Fig. 12: A customizable process model for the post-production example in BPFM (a); a possible customization (b).

In BPFM it is also possible to define *dependency constraints* between variants. If for example, a variant is chosen for a given variation point, it can restrict the choice of variants for another variation point. Dependencies can be between variation points, between variants or between variation points and variants.

Only control-flow actions can be customized in BPFM (P.Cf:$\pm$). There is no mechanism to customize resources and business objects (P.Re:$-$, P.Ob:$-$).

A tool implementing this approach is available as an Eclipse plugin. The tool can prune a customized process model by removing the unused variants, but does not offer support for embedding the selected variants back into the process model (EF.Imp:$\pm$). The approach has not been formalized (EF.For:$-$) nor validated in practice (EF.Val:$-$) and does not provide any correctness (CS.Syn:$-$, CS.Beh:$-$) nor decision support (DS.Abs:$-$, DS.Gui:$-$).

6.8. Provop: Process variants by options

The Provop approach [Hallerbach et al. 2008; 2009a; 2009b; 2010] proposes to derive process model variants via restriction or extension of a base model (IT.Res:$+$, IT.Ext:$+$). The base model is a process model annotated with *adjustment points* to allow its customization. For example, in Figure 13 we identified variant a from the set of post-production variants in Figure 4 as the base model, since this is one of the simplest variants out of those for post-production, and defined eight adjustment points on top of this model.

An adjustment point is a point where variations to the basic model can be made by applying three change operations: DELETE, INSERT and MOVE. A fourth operation, MODIFY, is applied to single model elements. For convenience, instances of these operations can be organized in sequences, called *options*, such that all operations in one option are applied in the order established by the option. Next, we describe the four operations.

DELETE allows the restriction of the basic model's behavior. This operation requires two adjustment points to delimit the portion of basic model to be removed. For example, the DELETE operation in Option 1 of our example will delete the content between adjustment points "w" and "z". It is also possible to delete a single node by providing its identifier.

INSERT allows the extension of the base model's behavior by inserting a fragment in the model. The fragment to be inserted is delimited by an entry and an exit point that need be matched with two adjustment points of the base model. A fragment is inserted in parallel to the portion of base model delimited by the two adjustment points, if this portion contains some element. For example, in the case of the first INSERT of Option 4 of our example, an AND-split and an AND-join are used to link the fragment to the adjustment points. If the portion delimited by the two adjustment points contains an arc only, or is empty (e.g. as a result of a previous DELETE), the fragment is inserted in place of the arc, respectively, between the two adjustment points. An example of this is the second INSERT of Option 2, where the sequence "Record digital film master"–"Recording completed" is essentially inserted in place of the arc between "y" and "n" in the base model.

With the MOVE one can move a fragment delimited by two adjustment points in the base model, to another part of the model delimited by two different adjustment points. MODIFY allows one to change the attributes of a modeling element, like the role associated with an activity. However, it is not possible to represent variability in other process aspects beyond the control flow (P.Cf:$+$, P.Re:$\pm$, P.Ob:$\pm$). This is because adjustment points can only be defined for entry or exit points of control-flow nodes, like functions and connectors in EPCs.

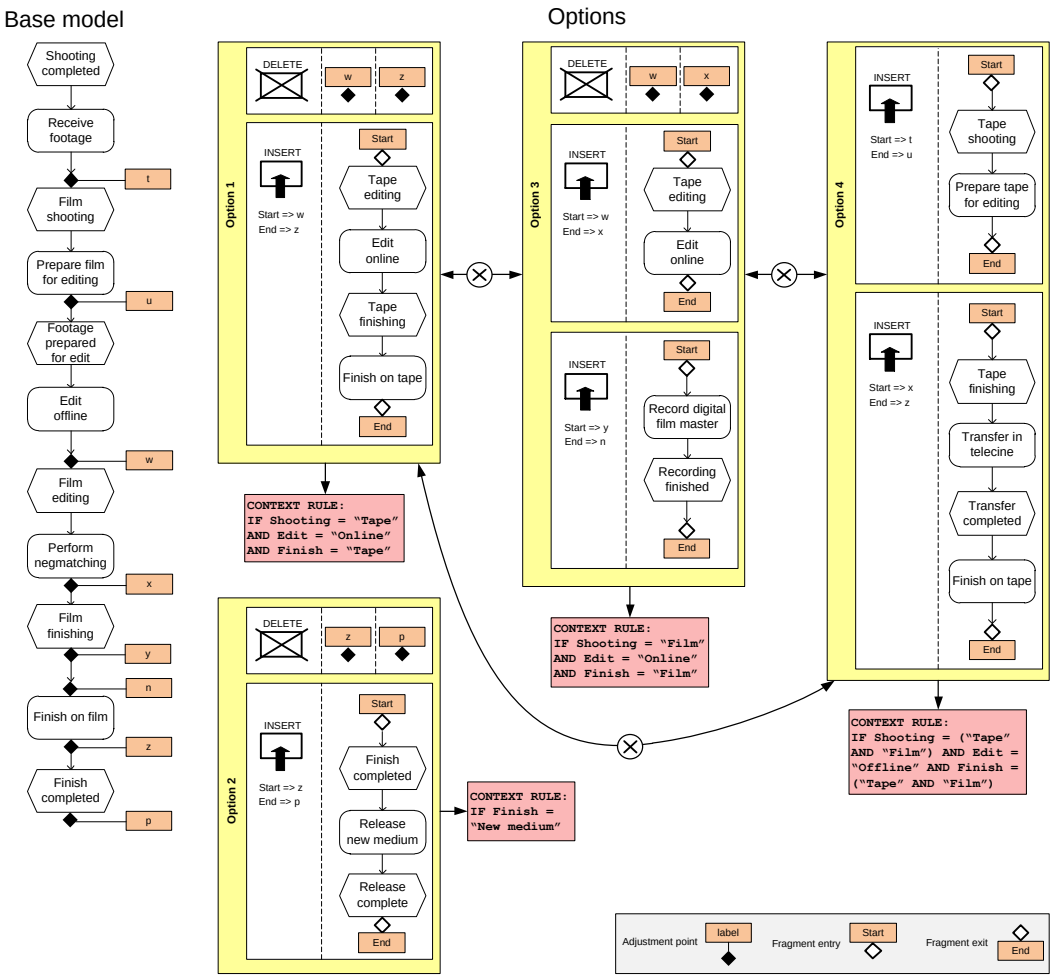

Fig. 13: The application of Provop to the process family of Figure 4.

In [Hallerbach et al. 2010] five different *policies* are described for designing a base model. The base model could be a standard process (e.g. a reference model for a particular domain), the most frequently used process variant, a generic model (like variant a in our example), the superset of all variants or their intersection. The policy chosen determines which change operations can be applied to customize the base model. For example, if the base model is the superset of all variants, customization is achieved by using the DELETE only, whereas if the base model is the intersection of all variants, INSERT, MOVE and MODIFY have to be used.

We organized the change operations in our example in four options. The application of Options 1 and 2 on the base model yields variant b of post-production (cf. Figure 4), Option 3 yields variant c while Option 4 yields variant d. The use of certain combinations of options can be restricted by defining *option constraints* such as mutual exclusion, implication and n-out-of-m, among the different options. For example, Options 1 and 3 of our example are set as mutually exclusive, since Option 1 removes the adjustment point "x" required by Option 3. The rationale behind the use of these con-

straints is to avoid creating situations that may prevent the application of an option or that introduce errors in the resulting variants. In [Hallerbach et al. 2010], each option can also be assigned a *context rule* constraining the applicability of that option to a particular business context. For instance, Option 1 can only be applied if shooting and finish are done on tape, and editing online. These context rules offer abstraction for the customization of the base model, though guidance is not offered (DS.Abs:+, DS.Gui:−).

The authors describe a five-step method to drive the customization of base models via context information [Hallerbach et al. 2009a; 2010]. In Step 1, the user is asked to determine all the possible *contexts* in the application domain. A context is characterized by a set of variable-value assignments. For example, a variable could be "budget" with three possible values: "high", "medium" and "low". One can also specify *context constraints* in the form of boolean expressions to limit the interplay among the various context variables, e.g. "budget = low ⇒ finish = tape". Each option is then assigned a context rule, which is a boolean expression over the values of context variables. In Step 2, for each context the set of relevant options is automatically determined. In Step 3, the consistency of the retrieved options for each context is checked against the option constraints. If inconsistencies are found, these are prompted to the user. For example, an option constraint may contradict a context constraint. Next, in Step 4 all valid sets of options are applied to the base model for each context, and the resulting variant is checked for soundness in Step 5. Those models that are not sound are discarded.

An advantage of this method is that it does not require the base model to be sound. However a disadvantage is that it does not guarantee the syntactical and behavioral correctness of the resulting variants a priori. Instead, soundness is checked a posteriori. Thus, a derived variant that is unsound will be discarded only at the end, with no help as to how this can be avoided. Further, the number of valid combinations of context variables into contexts may still be very large rendering an a-priori check of all derivable variants unfeasible (CS.Syn:−, CS.Beh:−).

Provop does not impose any restriction on the language used to model base models and fragments to be inserted. However, the examples shown seem to suggest a restriction of the approach to block-structured models since DELETE, INSERT and MOVE need to be mapped to a pair of adjustment points, e.g. it is not clear how multi-entry/multi-exit fragments can be inserted between two adjustment points, or how a new start or end node can be created on the base model. The approach only addresses customization of conceptual process models (PT.Con:+, PT.Exe:−). The basic concepts are formalized, though the semantics of the change operations is not specified (EF.For:±).

A partial implementation of Provop on top of the ARIS tool is described in [Hallerbach et al. 2010] (EF.Imp:±). This tool implements Step 4 of the Provop method: It allows users to define change operations and organize them in options, and to apply them on top of BPMN models enhanced with adjustment points, in order to derive a variant thereof. The tool is not publicly available.

Finally, in terms of validation, Provop's requirements for the "definition, adaptation and management of process variants" have been derived from various case studies in the automotive and healthcare industries [Hallerbach et al. 2010], and Provop models have been created in these domains. However these models have not been validated with domain experts (EF.Val:±).

6.9. aEPCs: Aggregated EPCs

The Aggregated EPCs (aEPCs) approach [Reijers et al. 2009] proposes a class of eEPC models, namely aEPCs, that can be used to capture a family of process variants. This class includes functions, events, connectors, arcs and products. The idea is to use *products* to indicate a particular business context in which the associated function

or event exists. For example, function “Transfer in telecine” only occurs in high budget projects, whereas function “Record digital film master” can also occur in medium budget projects. Accordingly, “High budget” and “Medium budget” can be considered as sub-products of a composite product “Budget” in post-production.

Figure 14.a shows an example aEPC where the products associated with the functions and events of this model refer to the various budget levels in post-production. Thus, this model represents how the various budget levels influence the post-production process.

A function or event may be associated with more than one product. In our example, function “Record digital film master” is associated with two products (“High budget” and “Medium budget”), but it could also be associated with the “Film finish” product (not shown). In fact, it is typical for a business process with different variants to feature a large number of products. For example, if we consider the different budget levels (3), shooting formats (2), picture cut methods (2) and finishing formats (3), we obtain a total of 10 different products for post-production. This means that each function or event may easily be associated with two or more products. In order to avoid cluttering the model with many product associations, an aEPC can be accompanied by one or more *product hierarchies* where the various products are organized hierarchically. Specifically, a product hierarchy is a rooted tree where the leaves are products and all other nodes are composite products representing product generalizations. In this way a process model element can be associated with a composite product in place of a set of products. For example, Figure 14.b shows the product hierarchy for the budget, which includes three budget levels and one composite product “Budget”. This latter product can be used in the aEPC when an element is present in all budget levels, like in the case of function “Receive footage”.

The aEPC approach is different than other approaches like C-EPCs or Provop: instead of capturing implications among process model elements via boolean expressions (e.g., function “Edit online” can only be present if function “Prepare tape for editing” is present), all possible variants are resolved beforehand and mapped to a set of products. Therefore, while the use of composite products can in principle reduce the number of products associated with an element, a realistic scenario where many possible variants exist, will still lead to a large number of composite products. As a result, the aEPC may be cluttered anyway [Baier et al. 2010]. That said, the choice of not modeling implications explicitly is motivated by the observation that in practice these logical expressions may be too difficult to conceive and interpret by a domain expert. This was the result of testing C-EPCs with domain experts of ING Investment Europe, with whom the aEPC approach was later validated [Reijers et al. 2009] (EF.Val:+).

An aEPC is customized by choosing one or more products, and removing all process model elements that are not associated with the products chosen (IT.Res:+). Customization is restricted to the control-flow perspective, since the only non control-flow element of the eEPC meta-model that is allowed in an aEPC is a product (P.Re:−, P.Ob:−). Products can only be applied to EPC functions and events; connectors are not customizable (P.Cf:±). The customization operation may require some cleaning up of the resulting model, in order to keep the model syntactically correct. In fact, besides the requirements of an EPC, there are requirements on how products can be associated with elements appearing before or after a sequence of connectors.

aEPCs are fully formalized (EF.For:+) including a customization algorithm (called *extraction algorithm*) that has been designed to preserve the syntactical correctness of an aEPC during customization (CS.Syn:+). Behavioral correctness is not dealt with (CS.Beh:−). The customization algorithm has been implemented in a tool that can import eEPCs from the ARIS toolset (EF.Imp:+).

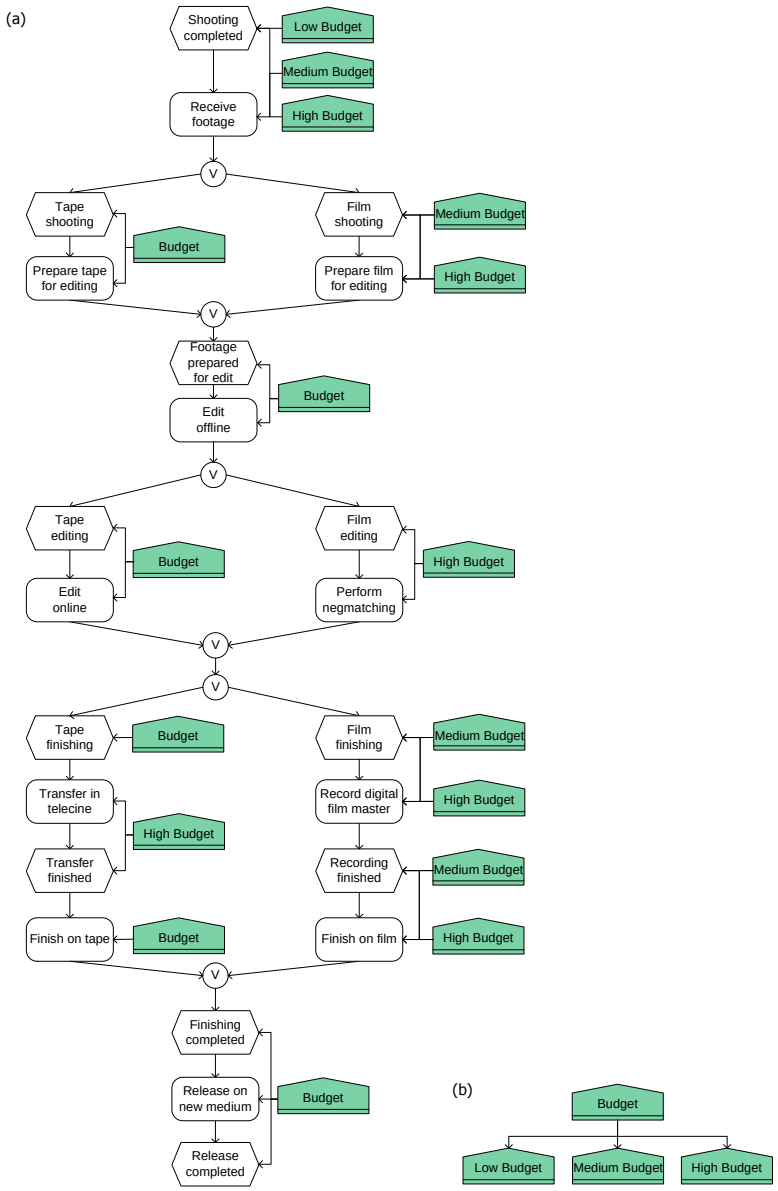

Fig. 14: An aEPC for post-production (a); the associated product hierarchy for the budget (b).

An advantage of organizing products in hierarchies is that an aEPC can be customized by removing products directly from the associated product hierarchy. This way, one does not necessarily need to work on the aEPC. Thus, this approach achieves process abstraction, though guidance is not offered when customizing a product hierarchy (DS.Abs:+, DS.Gui:−).

6.10. Template and Rules

The approach presented in [Kumar and Yao 2009; 2012] proposes to capture the variability of a process family by processing a set of *business rules* associated with a *process template*. The process template is a simple, block-structured process model which should be chosen in order to have the shortest structural distance to all process variants of the family. The rules can be used to configure the template by restricting or extending its behavior via *change operations* (IT.Res:+, IT.Ext:+). Change operations affect the control-flow perspective (by deleting, inserting, replacing or moving a single task or a process fragment), the resource perspective (by assigning a role to a task), and the data perspective (by assigning a value to a data attribute or changing the value of a role's property or of a task's input or output data). However, as far as the control flow is concerned, these change operations can only be applied to tasks and model fragments, not to routing elements (P.Cf:±, P.Re:+, P.Ob:+). Another customization option is offered by the possibility of changing the status of a process among four predefined values: normal, expedite, urgent and OFF.

Rules associate change operations with a boolean condition over so-called *case data*, so that if the condition is satisfied, the corresponding change operation is applied onto the process template. Depending on the type of operation, the approach differentiates between control-flow rules, data rules, resource rules and hybrid rules (the latter incorporating multiple process perspectives). Case data are not necessarily executable data but they rather capture business policies, so they may refer to domain aspects such as budget level and delivery media in post-production. In this respect, the use of business policies provides abstraction from the template since one may configure the template by reasoning at the level of the business domain (DS.Abs:+). There is however no guidance support (DS.Gui:−).

Figure 15 shows the application of this approach to our running scenario, using the BPMN language. Here the template describes a simple variant, namely that for editing and finishing on tape and releasing on new medium. This template is accompanied by three rules (*R1*, *R2*, and *R3*) embracing control-flow and resource aspects. For example, *R1* is used to configure the template for a high budget production process. Accordingly, we need to insert the tasks required for editing and finishing also on film, such as task "Prepare film for editing", to be inserted in parallel to task "Prepare tape for editing", "Transfer in telecine", which goes before "Finish on tape" and so on (where t_1, S_b, t_2 in a rule indicates to insert task t_1 before t_2). *R3* is an example of a rule to configure resource aspects. Accordingly, if the budget is high, multiple roles (e.g., "Director", "Editor", "Supervisor") get associated with the performance of task "Edit offline".

Change operations are applied to a tree representation of the template and only affect single-entry single-exit blocks of the template. Moreover, the application of each change operation triggers some cleaning operations required to avoid disconnected model elements and remove trivial gateways and arcs. For example, after deleting task "Release on new medium" from the template in Figure 15 via the application of rule *R1*, task "Finish on tape" and event "Finish completed" will be reconnected. Similarly, if there remains one branch only between two AND gateways, the two gateways will be removed altogether. Thus, change operations cannot cause any syntactical nor behavioral issues in the process template (CS.Syn:+, CS.Beh:+).

Change operations are described in detail in terms of changes to the tree representation of the template and an algorithm is provided to configure the tree. However, a systematic formalization of all notions is missing. Furthermore, an algorithm to transform a process model into a tree representation and vice versa is missing, while the resolution of rule conflicts is only exemplified by a matrix that disallows certain combinations of rules (EF.For:±).

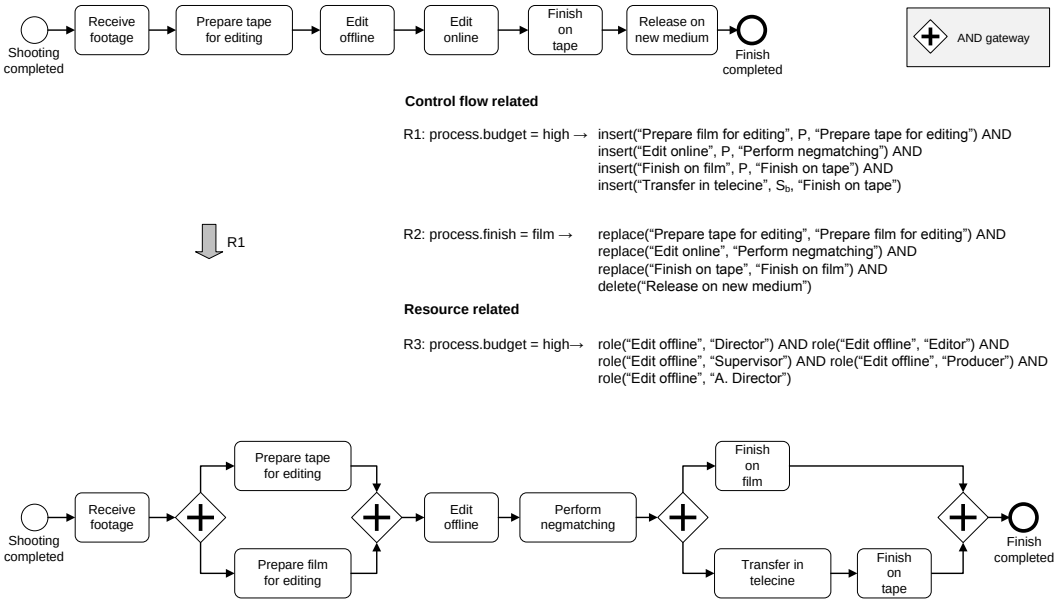

Fig. 15: The application of the Template and Rules approach to the process family of Figure 4.

A prototype tool has been implemented for this approach (EF.Imp:+), though it is not publicly available. Taken a template and a set of rules, the tool relies on the Drools-expert rule engine[7] to check for conflicts between the available rules. If conflicts exist (e.g. one rule deletes a task another rule is trying to insert), the user is notified to either resolve the conflicts or set a priority for the application of the rules. Similarly, the applicability of each rule is checked (e.g. it is not possible to delete an inexistent node) and errors are triggered for those rules that are not applicable. Finally, a configured process model is obtained from the template by using only those rules that are non-conflicting and applicable. After such a model has been obtained, it is checked for data-flow inconsistencies, e.g. a task whose data input is no longer available, in order to guarantee the executability of the customized model. However, the language used is not supported by a BPMS, thus the individualized models cannot effectively be executed (PT.Con:+, PT.Exe:±). However this check is only done a posteriori as a result of which, a customization may be unfeasible altogether. The tool supports templates and variants in the BPEL executable language.

Judging from [Kumar and Yao 2009; 2012], the approach has not yet been validated in practice (EF.Val:−).

6.11. Feature Model Composition

In the Feature Model Composition approach [Acher et al. 2010b], a process (called *workflow*) is defined as a collection of services, where each service corresponds to an activity. Activities are implicitly related via data dependencies. Specifically, each service has a collection of *dataports*. A dataport corresponds either to an input data object or to an output data object. If an input data port of a service refers to the same ob-

[7]See `www.jboss.org/drools/drools-expert`

ject as an output dataport of another service, then there exists an implicit data-flow dependency between these services.

In order to capture variability, a service is allowed to have any number of variation points (called *concerns*). A concern may refer to any service property (called *dimension*). Examples of dimensions are dataports, service interfaces, the service behavior and other low-level implementation aspects. Each concern is modeled as a separate feature model, which captures the various variants that exist for a concern, and their relations. The customization of concerns is achieved by deselecting features from the respective feature models (IT.Res:+).

In this approach, feature models do not really provide abstraction for the customization of concerns, since they refer to low-level aspects such as different dataports related to a workflow service. Moreover, one has to configure one feature model for each concern. In other words, there is no overarching feature model to customize a workflow using domain knowledge, like for example in PESOA. Similarly, no guidance support is provided (DS.Abs:−, DS.Gui:−).

Figure 16 shows a workflow in the Feature Model Composition approach using the running example. For example, activity "Prepare medium for editing" has been modeled as a service with output dataport "Shooting medium". A feature model has been defined for the concern "Shooting medium" and mapped to the output dataport of "Prepare medium for editing", in order to capture the fact that the service can have a film, a tape or both media as input. Similarly, the same feature model has been associated with the input dataport of service "Edit offline" and so on for all other input and output dataports. Further, the concern "Cut" with variants "Online" and "Negmatching" has been associated with the functional interface of service "Perform cut", to indicate that the type of this service can also be configured.

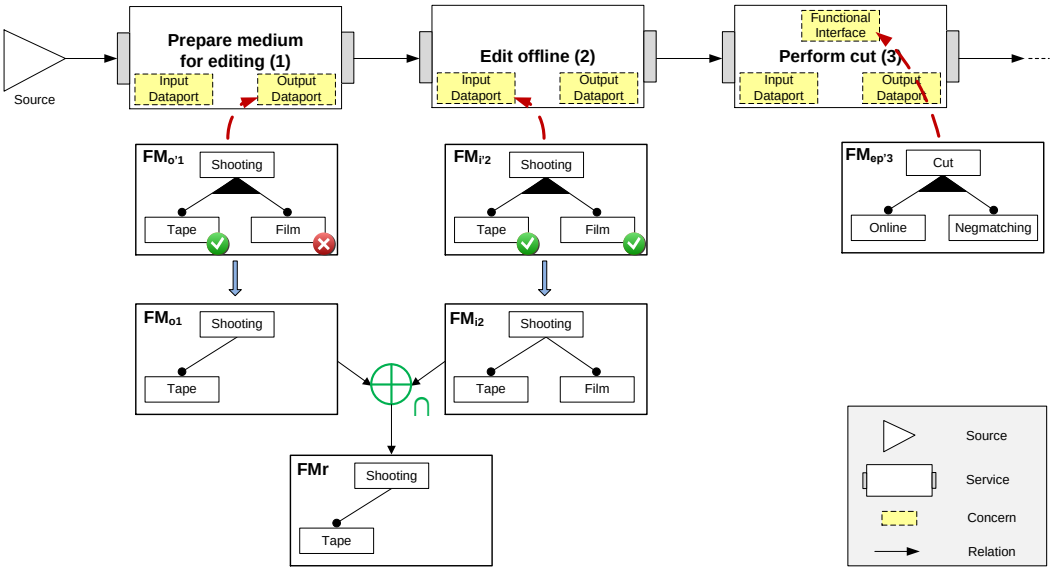

Fig. 16: A configurable workflow capturing the post-production scenario, according to the Feature Model Composition approach.

A concern of one service may be incompatible with a concern of a second service and therefore, a consistency check is needed when customizing a workflow. The consistency

check is performed by analyzing the input and output dataports of services based on dependency rules. Specifically, the feature models of the relevant concerns are checked for mutual consistency and then a merged diagram is created by intersecting the various feature models. In this way, the consistency of two connected services is ensured. The merge operator is used to compose feature models that refer to the same service dimension. Its syntax and semantics are defined in [Acher et al. 2010a], while the syntax of a workflow is defined in [Acher et al. 2010b] (EF.For:+). While inconsistencies between data dependencies that may arise during customization are addressed by this approach, the workflow language adopted is abstract, not actually executable (PT.Con:+, PT.Exe:−).

When producing a configured workflow, it is necessary to add the control-flow dependencies based on the implicit dependencies of the various service dimensions, such as the data-flow dependencies. The authors of this approach identify three types of control-flow dependencies: sequential, concurrent and conditional. The dependency rules for consistency checks between two services, as illustrated in Figure 16, are not sufficient when there is a sequential, concurrent or conditional ordering of more than two services. This is addressed via an extended set of dependency rules that ensures the consistency of services in a workflow.

As we have seen from the example, this approach can be used to configure business objects (P.Ob:+) as well as any other aspect related to a service, such as the associated resources (P.Re:+). However, concerns are only internal to each service. As such, the workflow's control flow cannot actually be configured (P.Cf:−). This is the only approach among those surveyed that suffers from this limitation.

While the syntax of workflows is not explicitly discussed, from the examples described in [Acher et al. 2010b], it seems that a workflow can only be block-structured and since its control flow cannot be configured and data-flow dependencies are preserved by the rules illustrated above, the approach supports both syntactical and behavioral correctness (CS.Syn:+, CS.Beh:+).

An implementation is described on-line[8] (EF.Imp:+), though the tool cannot be downloaded. While this approach is motivated by the need of customizing medical imaging grid services, it has not been validated in practice (EF.Val:−).

6.12. Subsumed Approaches

Various other proposals for customizable process modeling are subsumed by the ones described above. Here, we say that an approach A is subsumed by B if A supports a subset of the variability mechanisms of B. Note that the focus is on the supported variability mechanisms and not on the process modeling language, supporting techniques or extra-functional aspects. Thus, for example, an approach A can be subsumed by B even if A and B are applied to different process modeling languages (e.g. EPC vs. UML ADs). Subsumed approaches are minor approaches compared to the approaches presented so far. They are briefly reviewed below.

6.12.1. Subsumed by C-EPCs. The initial incarnation of the KobrA (Component-based Application Development) method [Atkinson et al. 2000] provides a mechanism to capture a family of process variants via a customizable process model. The purpose is that of customizing component-based software systems. As such, process models are employed for the description of components' behavior. In KobrA, process model customization is done using UML ADs and is driven by a decision table (see Section 8.3). Similar to C-EPCs, decision nodes in UML ADs can be marked as variation points (using a black background and a letter "M") to indicate that subsequent actions are

[8]See `http://modalis.polytech.unice.fr/softwares/manvarwor`.

optional. However, an individualization algorithm to transform the customized process model is not discussed. Further, there is no method to guarantee the correctness of the resulting process model, nor is there a mechanism to exclude unfeasible configurations as a result of wrong combinations of answers. The Kobra method has been implemented in a tool and validated in numerous industrial settings [Atkinson et al. 2008], though there is no information on the involvement of domain experts.

Korherr & List [Korherr and List 2007] present an approach which extends UML ADs with stereotypes to capture variability. In their approach, variability can be defined at the level of an atomic action, a group of actions (called an *activity partition*) or a decision node. An action or activity partition can be defined as being `<<mandatory>>` (the action or partition must be retained during customization) or `<<optional>>` (the action or partition can be excluded during customization). A decision node can be defined as being an `<<alternative choice>>` with a `0..1` range (at most one outgoing flow must be selected during customization) or with a `1..*` range (at least one outgoing flow must be selected). Similar to C-EPC requirements, it is also possible to state that the selection of an element (action, partition or flow) during configuration requires the selection of another element elsewhere (denoted by a dependency arrow marked with the `<<requires>>` stereotype), or that the selection of an element excludes the selection of another one elsewhere in the model (denoted by an arrow with the `<<excludes>>` stereotype). Further constraints between customization elements can be defined using the Object Constraint Language (OCL). Abstraction from the customization of the process model can be achieved via the use of a UML Profile for variability, provided by this approach. This is similar to a feature model in terms of functionality (see Section 8.1).

6.12.2. Subsumed by PESOA. Razavian and Khosravi [Razavian and Khosravi 2008] propose an approach to define customizable process models in the form of UML ADs enhanced with a fixed set of stereotypes. The set of stereotypes is a subset of that in PESOA. Specifically, their approach covers two types of variation points: *optional* and *alternative*. An optional variation point allows the selection of at most one variant among the available ones. An alternative variation point allows the selection of exactly one variant. Variation points can be defined on both control-flow elements and on data objects. Along the control-flow perspective, a decision node can be marked with `<<opt_vp>>` to indicate an optional variation point, in which case the node is customized by choosing at most one of its outgoing paths, and with `<<alt_vp>>` to indicate an alternative variation point, in which case the node is customized by choosing exactly one of its outgoing paths. However, other types of gateways such as forks and joins cannot be customized. Actions can be marked as `<<optional>>` or as `<<vp_al>>`. In the former case, the action can be excluded during customization, while in the latter case the action can be customized to one of its variants. Interdependencies between model variants cannot be defined beyond stating that a variation point is either optional or alternative. As a result, only simple configuration scenarios can be captured. The customization of input and output data objects and data stores (used to persist data beyond a process instance) is achieved in the same way as for actions. It is also possible to mark a composite activity with the stereotype `<<variable>>` to indicate that the underlying subprocess includes some variation points. The authors recognize that if no variant is chosen for an optional variation point in the control-flow, the model may become disconnected. The correctness of customized models is not checked.

A similar approach is provided by Ciuskys and Caplinskas [Ciuksys and Caplinskas 2007] in the context of UML ADs. In this approach the only process model elements that can be made customizable are actions. A configurable action is called a *generic activity*. During configuration, a generic activity can be replaced by one of several possible concrete (non-generic) activities. Alternatively, an activity may be skipped

(removed) during configuration if it is marked as *optional*. The space of customization options is specified using a feature model, where each feature corresponds to a (generic or non-generic) activity and where feature inter-dependencies can be defined directly on the feature model. The features that are inner nodes in the feature model represent generic actions, while the leaf features correspond to non-generic actions. One may configure a process model by selecting features in the feature model. These features then determine how the generic actions in the process model are configured, i.e. which concrete action is selected for a given generic action or which generic actions are removed during configuration. For the purpose of reasoning about the consistency of a given subset of features of a feature model, the feature model is translated into a theory in Description Logic (DL) and a DL reasoner is used for consistency checking.

Kulkarni & Barat [Kulkarni and Barat 2011] put forward a similar approach in the context of BPMN models. Here a generic activity (called *abstract activity*) can be replaced by a single (atomic) *concrete activity* or by an entire subprocess (called *composite activity*). Also abstract events can be replaced by concrete events. Routing elements cannot be customized. Kulkarni and Barat suggest that feature models can be used to guide the customization process, but they do not specify any concrete mechanism for linking a process model with a feature model. Thus, effectively they do not provide any decision support for process model customization. A formalization of the basic notions is provided, though an individualization algorithm has not been formalized.

6.12.3. Subsumed by Configurable Workflows. A CoSeNet (Configurable Service Net) [Schunselaar et al. 2011; 2012] is an alternative representation of a configurable workflow model via a directed acyclic graph. CoSeNets have been designed to fulfill two requirements: i) always yield behaviorally correct (i.e. sound) individualizations and ii) being reversible, i.e. it should be possible to obtain the initial process variants used to create the CoSeNet, through individualization. Each leaf of a CoSeNet represents a process task and each parent node represents an operator. The available operators are sequence, the logical connectors OR, AND, data-driven XOR and event-driven XOR, and the structured REPEAT-UNTIL loop. Connections between nodes are achieved via special nodes, called VOID nodes, linked to parent and child nodes via edges. These nodes do not bear any semantics. For example, an OR between "Prepare film for editing" and "Prepare tape for editing" means that either or both of these tasks can be executed. A CoSeNet thus captures a block-structured process model where each single-entry single-exist block is identified by an operator and its children nodes. While this structure guarantees soundness of the process model by construction, it may be cumbersome to capturing complex business processes, e.g. to model processes with arbitrary cycles one needs to duplicate activities. Configuration is achieved by applying the hiding and blocking operators of Configurable Workflows to the VOID nodes. CoSeNets have been defined formally, though a definition of the individualization algorithm is not available, and have semantics in terms of YAWL. A mapping from CoSeNets to plain YAWL models can be used for executing the configured models. However, the approach abstracts from data and resource aspects, thus effectively offering limited support for execution. Moreover, a mapping in the opposite direction is not described, which may hinder the applicability of this approach in practice. CoSeNets have been used to capture process models from ten different municipalities. The CoSeNet approach has been implemented via various plugins for the ProM environment.[9]

6.12.4. Subsumed by BPFM. Ripon et. al. [Ripon et al. 2010] present an approach similar to BPFM. Here actions in a UML AD can be marked with a stereotype `<<variant>>`. An action tagged with such a stereotype represents a variation point and is linked to

[9]See `http://processmining.org`.

an entry in a *variant model* listing all possible options (i.e. variants) for customizing the action. Such variants are also summarized in a decision table (see Section 8.3) that is presented to the user. Multiple variants can be selected for the same variation point, depending on the constraints specified among the variants of the same variation point, though it is not clear how multiple variants, when selected, will be represented in the individualized process model. By selecting/de-selecting variants from the decision table, one can determine which variant(s) of an action will be picked during configuration time. While multiple variants can be selected for a variation point, different than BPFM, a cardinality cannot be specified. Further, the approach only works by restriction of variants.

Nguyen et. al. [Nguyen et al. 2011] operate in the context of BPMN models. In this approach, variation points can be defined in BPMN activities, data objects, as well as message flows connecting activities in different pools. Each variation point is assigned one or more variants and a minimum and maximum cardinality is attached to define the number of variants that can be selected. Dependencies between variants, within and across variation points, can also be defined. The approach works on BPMN process models at the conceptual level. Abstraction is achieved via the use of feature models (see Section 8.1). An implementation of the approach as an Eclipse plugin is available.

7. TECHNIQUES FOR CORRECTNESS SUPPORT

The customization of variation points attached to parallel splits, decision points (XOR- and OR-splits) and synchronization nodes of a customizable process model may lead to the introduction of model errors. On the one hand, syntactical errors such as disconnected model elements are easy to detect and fix. On the other hand, behavioral anomalies such as deadlocks and livelocks pose challenges, as they can typically only be identified via a state-space analysis of the model, which is exponential in complexity. A customizable process model capturing a realistic scenario can easily induce a large number of possible customizations. For example, if we assume 50 variation points each with three alternatives, we get $3^{50} \approx 7.18e + 23$ possible customizations. Checking the behavioral correctness of each individual customization *a-posteriori* is thus unfeasible.

In this section we discuss two techniques that can be used to guarantee both syntactical and behavioral correctness of customized process models *a-priori*, i.e. while customizing the process model. Both techniques apply to configurable process models where configuration is achieved using the hiding and blocking operators, such as in the C-EPCs (See Section 6.1) and Configurable Workflows (see Section 6.5) approaches. In particular, the first technique, namely Constraints Inference, has also been applied to a subset of C-EPCs. Moreover, it has been shown that any behavioral restriction of a process model's control flow can be explained via a combination of the hiding and blocking operators, applied to the model's tasks [van der Aalst and Basten 2002]. Thus, the correctness techniques presented in this section can in principle be adapted to offer correctness support to all approaches modeling variability by restriction.

7.1. Constraints Inference

The work in [van der Aalst et al. 2010] proposes a formal framework for individualizing configurable process models incrementally, while preserving both syntactical and behavioral correctness. The framework is based on a technique to automatically infer propositional logic constraints from the control-flow dependencies of a process model (i.e. from its syntax), that, if satisfied by a configuration step, guarantee the syntactic correctness of the individualized model.

The theory was first developed in the context of Workflow nets (WF-Nets) and then extended to a subset of C-EPCs. WF-Nets are a class of Petri nets specifically designed to model business processes. They come with a definition of *soundness* which ensures a

process model to be free of deadlocks and to be able to complete properly. Each WF-Net transition captures a process task and can serve as a variation point: it is allowed by default and can be hidden or blocked during configuration.

Whenever a transition is hidden or blocked, the current set of constraints is evaluated. If the constraints are satisfied, the configuration step is applied. If the constraints are violated, a reduced propositional logic formula is computed, from which additional transitions are identified that need to be configured simultaneously in order to preserve the syntactic correctness. For example, if a transition in the process model is blocked, all nodes in a path starting with that transition need also to be removed to avoid disconnected model elements. The set of constraints is incrementally updated after each step of the configuration procedure.

A core result of this technique is that, for WF-Nets satisfying the *free-choice* property [Desel and Esparza 1995], if the outcome of a configuration step starting from a sound WF-Net is a WF-Net (i.e. it is syntactically correct), then this latter WF-Net is also sound. This means that for this class of nets, configuration steps that preserve syntactical correctness also preserve behavioral correctness.

This technique provides an efficient way of ensuring behavioral correctness during configuration. However, it assumes that the configurable process model is already sound and free-choice. The latter does not represent a significant limitation since the large majority of constructs of languages such as EPCs, BPMN or BPEL can be mapped to WF-Nets in this class [Lohmann et al. 2009]. On the other hand, imposing the configurable process model to be sound may hinder the applicability of this approach in practice, e.g., abstracting from data and resources may generate false positives (e.g., models that have behavioral problems due to data dependencies) and false negatives (e.g., the reported error is circumvented using data conditions).

7.2. Partner Synthesis

With the aim to address the shortcomings of [van der Aalst et al. 2010], the work in [van der Aalst et al. 2012] proposes a new technique for ensuring behavioral correctness during process configuration. This technique relies on the application of hiding and blocking to a wider Petri net class than free-choice WF-Nets, namely Open Nets. Open Nets can have multiple end states (whereas WF-Nets only have one) and can have complex non-free choice dependencies. Moreover, this technique relies on *weak termination* as a notion of behavioral correctness. Weak termination is a weaker correctness notion than soundness, because it only ensures that a process instance can always reach an end state from any state that can be reached from the start state. This means that even if some transitions are unreachable (i.e. “dead”), the model will still be considered behaviorally correct. The authors argue that this correctness notion is more suitable for configuration as parts of a configured net may be left intentionally dead.

The technique is based upon the notion of *configuration guideline*, which is inspired by the notion of operating guidelines used for partner synthesis [Wolf 2009]. A configuration guideline is a complete characterization of all *feasible configurations*, i.e. those configurations yielding a weakly terminating configured model. These configurations are represented as possible combinations of blocked transitions assuming that in the initial Open Net all transitions are allowed by default. Alternatively, it is possible to start from an Open net where all transitions are blocked by default, and configure the net by allowing transitions. In this case, the configuration guideline will store all possible combinations of allowed transitions. Thus, one can check the configuration guideline at each configuration step, and enforce further transitions to be blocked or allowed, in order to keep the current configuration feasible, in a way similar to the staged configuration in [van der Aalst et al. 2010]. However, the initial configurable

Open Net does not need to be sound. If the net has no feasible configurations, this is reported and the user will not be able to hide or block any transition.

The technique automatically generates the configuration guideline from an Open Net. This is done by first building a *configuration interface* which can communicate with services that configure the original model. The configuration guideline thus represents the most permissible service that is able to interact with the Open Net by configuring it.

The Partner Synthesis technique has been applied to the C-YAWL language, and implemented as a component of the YAWL Editor. Once the tool has computed the configuration interface and its guideline for a C-YAWL model, the user can interactively configure the model. At each configuration step, the system analyzes the configuration guideline and automatically blocks further YAWL ports to keep the current configuration feasible. The user can also roll back a previously taken decision, e.g. by allowing a port that was blocked. In this case, the tool may impose that further ports have to be allowed in order to keep the configuration feasible. The tool's response time is instantaneous, because the traversal of the configuration guideline has a linear complexity. The rationale of this approach is thus to move the computation time from customization-time to design-time, i.e. when the configuration guideline is built.

8. TECHNIQUES FOR DECISION SUPPORT

In this section we report on three techniques that can be used to provide decision support during process model customization. In particular, two techniques, namely Feature Models and Decision Tables, offer *abstraction* from the customizable process model and its variation points when performing a customization. This is achieved by capturing variability at the level of the domain in which the process model has been constructed, in order to allow users to reason in terms of domain concepts rather process modeling elements. This is especially useful when the customizable process model features many interdependent variation points, as one would expect is a realistic scenario. A third technique, namely Questionnaire Models, also offers *guidance* for the customization of process models. Guidance entails the provision of a mechanism to guide users in making the right decisions, for example in the form of recommendations.

The three techniques reported in this section apply to customizable process models that capture variability by behavioral restriction.

8.1. Feature Models

Feature models are a family of techniques originally conceived to describe the variability in software product lines in terms of their features, and later applied to different domains. Various feature modeling languages have been proposed [Schobbens et al. 2006] since feature models were first introduced as part of the FODA (Feature Oriented Domain Analysis) method [Kang et al. 1990].

A feature model is represented graphically by one or more feature diagrams. A feature diagram is a rooted tree with high-level features being decomposed into sub-features. A *feature* captures a property of the domain under analysis, that is relevant to a stakeholder.

Figure 17 shows a possible feature diagram for the picture post-production domain, using the notation proposed in [Batory 2005]. There are features related to the options available for shooting, type of editing, transfer and finish.

A feature can be mandatory or optional (in which case it can be deselected), and can be bound to other features via constraints. Constraints between features can be expressed as arbitrary propositional logic expressions over the values of features [Batory

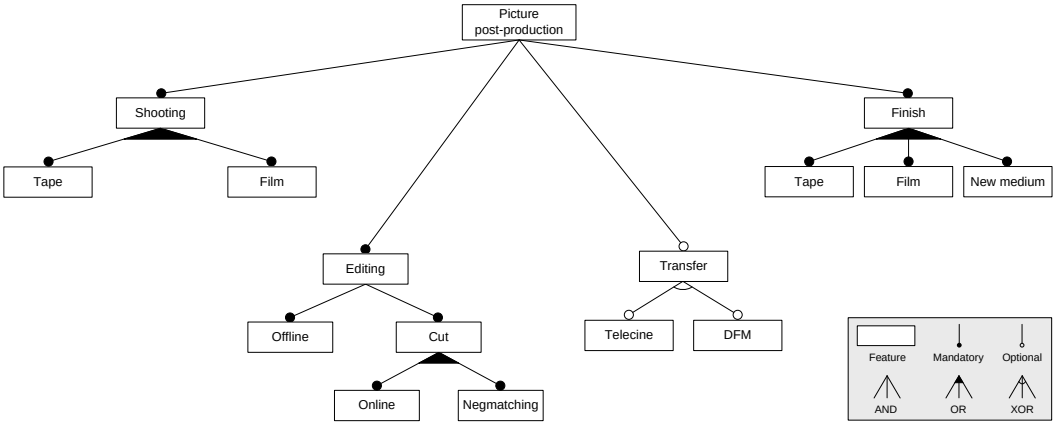

Fig. 17: A feature diagram for post-production.

2005]. For example, the sub-feature "Negmatching" of "Cut" must be deselected if the sub-feature "Film" of "Shooting" is not selected.

Constraints between the sub-features of a same feature can also be represented graphically. This way restrictions with respect to the number of sub-features a feature can have can be modeled. These relations can be: AND (all the sub-features must be selected), XOR (only one sub-feature can be selected) and OR (one or more can be selected). OR relationships can be further specified with an $n : m$ cardinality [Czarnecki et al. 2005], where n indicates the minimum and m indicates the maximum number of allowed sub-features. For example, in Figure 17 the sub-features of "Cut" are bound by an OR relation as it is possible to have more than one type of cut in post-production.

Observe that while an optional feature always represents a variability, a mandatory feature does not necessarily represent a commonality in the domain under analysis. In fact, a mandatory feature can still be excluded if it has an XOR/OR relation with its sibling features. This is the case of the sub-features of "Finish", which are all mandatory (a choice on the finish is required), though it is possible to choose any subset of them due to the OR relation.

A *feature configuration* specifies a valid scenario in terms of features selected/deselected, i.e. a scenario that complies with the constraints. Figure 18 depicts the feature diagram for post-production configured for a project shot on tape, edited online and delivered on film.

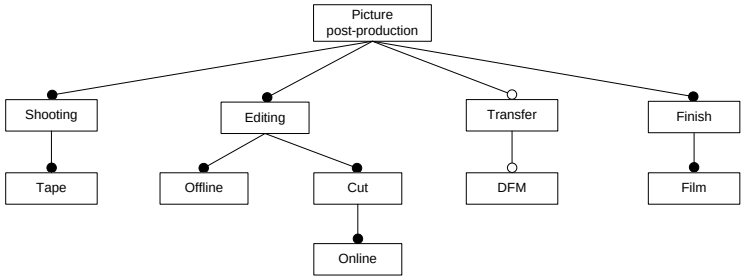

Fig. 18: A possible configuration for the post-production feature diagram.

Various tools supporting the definition and configuration of feature models are available. Some examples are the *AHEAD Tool Suite*,[10] the Eclipse plugin *FeatureIDE*,[11] and the online toolset *SPLOT*.[12]

Although the initial aim of feature models was to facilitate the configuration of software product families, this technique has also been used to provide abstraction for the customization of process models in various approaches. Feature models are used by PESOA (see Section 6.3), Superimposed Variants (see Section 6.4), Feature Model Composition (see Section 6.11), and the approaches by Ciuskys and Caplinskas (see Section 6.12.2) and by Nguyen et. al. (see Section 6.12.4). Korherr & List (see Section 6.12.1) use UML Profiles for variability which are similar to FDs.

In these approaches, one can customize a process model by selecting/deselecting features from a feature model. In order to do so, one has to first establish a mapping between features on the one hand, and variants of variation points in the process model on the other hand. Once a feature configuration has been completed, an algorithm exploits this mapping to select the right variant(s) for each variation point of the process model. Then an individualization algorithm, if available, is triggered to individualize the customized process model.

8.2. Questionnaire Models

Questionnaire models [La Rosa et al. 2009] are another technique for representing the variability of a domain. The idea behind this technique is to organize a set of features, called *domain facts*, into *questions* that can be posed to end users in order to configure the domain under analysis.

Figure 19 shows a possible questionnaire model for post-production, where all questions and facts have been assigned a unique identifier. Questions group domain facts according to their content, so that all the domain facts of a same question can be set at once by answering the question. For example, the question "What type of shooting has been used?" groups the domain facts "Tape shooting" and "Film shooting". Each domain fact is a boolean variable and has a default value, which can be used to identify the most common choice for that fact. For example, since the majority of production projects are shot on tape because it is less expensive than film, we can assign a default value of true to "Tape shooting", and of false to "Film shooting". Moreover, a domain fact can be marked as mandatory if it needs to be explicitly set when answering the questionnaire. If a non-mandatory fact is left unset, i.e. if the corresponding question is left unanswered, its default value can be used to answer that question. In this way, each domain fact will always be set, either explicitly by an answer or by using its default value. Observe that the mandatory attribute of a domain fact has different semantics than the mandatory attribute of a feature in a feature model.

In Figure 19, there are questions that gather information about the type of shooting media (q_3), picture cut (q_4) and deliverables (q_5). These questions capture domain facts similar to the features in the feature diagram of Figure 17. Next to these questions, however, we have defined two high-level questions: question q_1, which enquires about the estimated budget for a post-production project (low, medium or high), and question q_2, which inquires about the distribution channel (e.g. cinema, TV, home). Different than feature diagrams, where each feature is mapped to a single alternative of a variation point in the customizable process model, "high-level" questions are defined with the intention of configuring multiple variation points at once, as we will show later.

[10] See `http://www.cs.utexas.edu/users/schwartz/ATS.html`.
[11] See `http://fosd.de/fide`.
[12] See `http://www.splot-research.org`.

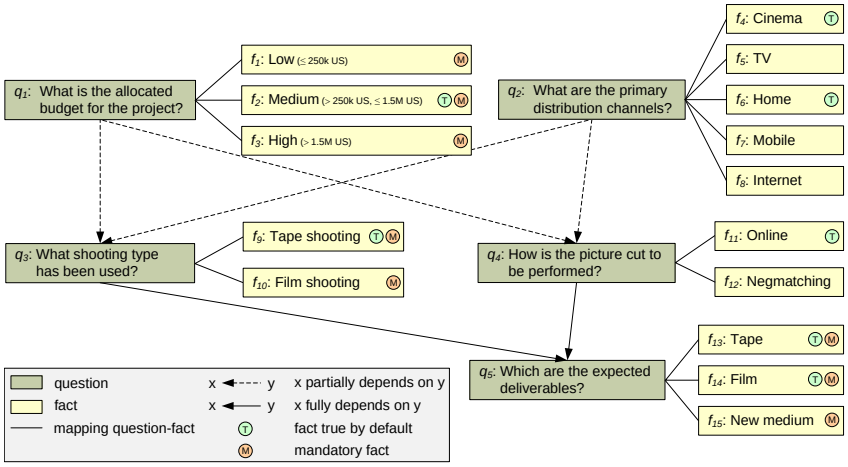

Fig. 19: A questionnaire model for post-production.

In general, one cannot freely answer the questions because of interdependencies. This is also the case in our example questionnaire model. Here, the answers to the questions are interrelated as there is interplay among their facts due to the constraints imposed by post-production. For example, negmatching (f_{12} in q_4) is a costly operation that can only be chosen if the project is shot on film (i.e. if f_{10} is true in q_3). However, the choice of which shooting medium to use is influenced by the project budget, and by the distribution channel. For low budget productions (f_1 set to true in q_1), shooting and finishing on film are not allowed (hence the corresponding domain facts f_{10} and f_{14} must be set to false). In turn, if shooting on film is not allowed (f_{10} = false), negmatching must also be denied (f_{12} = false), and so on. This interplay among domain facts can be encoded by a set of *domain constraints* expressed as boolean formulae over the values of the domain facts, similar to the constraints defined among the features of a feature model. A *domain configuration* is thus a valuation of domain facts, resulting from answering a questionnaire, which does not violate the constraints.

A further difference between feature models and questionnaire models is the ability in a questionnaire model to establish an order for posing questions to users. This is achieved via *order dependencies*. A *partial* dependency captures an optional precedence between two questions: e.g. q_3 in Figure 19 can be posed after q_1 OR q_2 have been answered. A full dependency captures a mandatory precedence: e.g. q_5 is posed after q_3 AND q_4. Dependencies can be set in a way to give priority to the most discriminating questions, i.e. the high-level questions q_1 and q_2 in our example, so that subsequent questions can be (partly) answered, automatically, by enforcing the domain constraints. If, for example, we pick "Low" budget in q_1, the questions about the shooting and cut type (q_3 and q_4) become irrelevant, because one can only choose facts "Tape shooting", respectively, "Online" editing, and will thus be skipped. These order dependencies can be arbitrary so long as cycles are avoided.

Questionnaire models offer both abstraction and guidance for the configuration of process models. Users can answer a questionnaire using an interactive questionnaire tool called *Quaestio*,[13] that poses questions in an order consistent with the order dependencies, and prevents users from entering conflicting answers to subsequent questions

[13] *Quaestio* is part of the *Synergia* toolset, see `http://processconfiguration.com`.

by dynamically enforcing the domain constraints. The tool also takes care of skipping questions that have become irrelevant while answering the questionnaire. Further guidance is provided in the form of recommendations attached to single questions and domain facts, providing contextual information on how to answer the questionnaire.

Questionnaire models have been applied to the configuration of C-(i)EPCs (see Section 6.1) and Configurable Workflows (see Section 6.5). By assigning a *process fact* to each alternative of a variation point in the configurable process model the questionnaire gets linked to the customizable process model [La Rosa et al. 2008]. A process fact is a boolean variable that captures the selection of a specific alternative of a variation point in the customizable process model: A process fact set to true means that the corresponding alternative in the process model has been selected; vice versa, setting the fact to false means that the alternative is not selected. Different than feature models, there is not necessarily a one-on-one mapping between process facts and domain facts. Rather, a boolean expression over the domain facts of the questionnaire model is assigned to each process fact so that the latter is set to true when the corresponding expression evaluates to true. Thus, depending on how this mapping is defined, the configuration of a single variation point can be affected by multiple domain facts, as well as a single domain fact can affect the configuration of multiple variation points. For example, we can map the questionnaire model of Figure 19 to the C-EPC example of Figure 5 in such a way that when q_1 is answered with a "Low" budget level, all the configurable OR connectors in the process model get configured, at once, to their left-hand side branches. This is because a low budget production imposes that shooting, editing and release are all done on tape.

Similar to feature models, this technique is generic and can be applied to other approaches that achieve customization by behavioral restriction.

8.3. Decision Tables

Decision tables are an alternative, tabular representation of questionnaire models. A decision table is composed of *decisions* (also called *conditions*). A decision, which can be expressed in the form of a question, is associated with an enumerated set of possible *resolutions* (also called *alternatives*). Each resolution can be linked to one or many variation points in a process model via an *effect* (also called *action*). The effect explains how the variation point needs to be customized when a particular resolution is taken.

Decision tables have been suggested as an abstraction mechanism in the KoBrA method (see Section 6.12.1) and in the approach by Ripon et al. (see Section 6.12.4). However, while decisions can be ordered in a decision table, the approaches that resort to this technique do not provide any mechanism to skip irrelevant decisions nor recommendations for taking the decisions.

9. DISCUSSION

The results of the comparative analysis are summarized in Table I. The first column lists the 19 approaches considered. Each approach is identified by a reference to the primary publication. The second and the third column indicate the year of the primary publication and the modeling language employed by the approach. The remaining columns indicate to what extent the approach in question covers each criterion.

From this table it is evident that while the surveyed approaches cover a heterogeneous set of methods for capturing variability in process models, none of them addresses all the identified criteria. This observation however should not be construed as a fundamental limitation of existing approaches. It is justifiable for example that a given approach focuses on conceptual rather than executable models, or that an approach supports customization by restriction rather than extension.

Approach	Year of primary publication	Process modeling language	Inherent Type		Scope					Supporting Techniques				Extra-Functional		
					Perspective			Process Type		Correctness Support		Decision Support				
			Restriction [IT.Res]	Extension [IT.Ext]	Control-flow [P.Cf]	Resources [P.Re]	Objects [P.Ob]	Conceptual [PT.Con]	Executable [PT.Exe]	Syntactical [CS.Syn]	Behavioral [CS.Beh]	Abstraction [DS.Abs]	Guidance [DS.Gui]	Formalization [EF.For]	Implementation [EF.Imp]	Validation [EF.Val]
KobrA	2000	UML ADs	+	-	+	-	-	+	-	-	-	+	-	-	+	±
C-EPCs	2003	C-iEPCs	+	-	+	+	+	+	-	+	+	+	+	+	+	+
Configurative Process Modeling	2004	EPCs	+	+	±	+	+	+	-	±	-	+	-	-	+	±
PESOA	2005	BPMN, UML ADs	+	-	±	-	+	+	-	-	-	+	-	±	±	+
Superimposed Variants	2005	UML ADs	+	-	+	-	-	+	-	±	-	+	-	+	+	-
Configurable Workflows	2006	C-YAWL, C-SAP, C-BPEL	+	-	+	-	-	+	+	+	+	+	+	+	+	+
Ciuksys & Caplinskas	2006	UML ADs	+	-	±	-	-	+	-	-	-	+	-	-	-	-
ADOM	2007	UML ADs, EPCs, BPMN	+	+	+	-	-	+	-	-	-	-	-	+	-	±
Korherr & List	2007	UML ADs	+	-	+	-	-	+	-	-	-	+	-	-	-	-
BPFM	2008	UML ADs	+	+	±	-	-	+	-	-	-	-	-	-	±	-
Razavian & Khosravi	2008	UML ADs	+	-	±	-	+	+	-	-	-	-	-	-	-	-
Provop	2008	Block-structured	+	+	+	±	±	+	-	-	-	+	-	±	±	±
aEPCs	2009	aEPCs	+	-	±	-	-	+	-	+	-	+	-	+	+	+
Template and Rules	2009	Block-structured	+	+	±	+	+	+	±	+	+	+	-	±	+	-
Feature Model Composition	2010	Block-structured	+	-	-	+	+	+	-	+	+	-	-	+	+	-
Kulkarni & Barat	2010	BPMN	+	-	±	-	-	+	-	-	-	-	-	±	-	-
Ripon et al.	2010	UML ADs	+	-	±	-	-	+	-	-	-	+	-	-	-	-
Nguyen et al.	2011	BPMN	+	-	±	-	+	+	-	-	-	+	-	-	+	-
CoSeNets	2011	CoSeNets (DAGs)	+	-	+	-	-	+	±	+	+	-	-	±	+	±

Table I: Evaluation results at a glance.

Zooming column-by-column, we observe that all of the considered approaches support customization by restriction (column IT.Res), but only few support customization by extension (column IT.Ext). This can be explained by the fact that there is a tradeoff between supporting customization by extension and other criteria. In particular, when extension is supported, the modeler is given substantial freedom to add or modify parts of the model, to the extent that it may become impossible to guarantee correctness of the customized models. Not surprisingly, approaches that support extension do not support correctness, except for Template and Rules, which supports correctness but at the expense of restrictions on the structure of the customizable model and allowed extensions – namely that these must be block-structured — and Configurative Process Modeling, which only partly supports structural correctness and does not support behavioral correctness. Similarly, if an approach supports extension, it cannot provide full guidance during customization, since this would require that the customizable process model contains information to provide guidance for every possible extension that can be made. But if every possible extension could be foreseen in the customizable process model, then the customization would be made by restriction (removing foreseen options) rather than by extension.

Regarding the modeling scope, we note that all approaches provide at least some customization mechanism along the control-flow perspective, but only a handful support the customization of resources and objects. Approaches based on UML ADs and BPMN do not support customization of resources. This can be explained by the fact that these two languages provide limited support for capturing resources — basically the ability to associate a lane or a pool (representing a resource or organizational unit) to each activity in the process. Accordingly, it is mainly in the context of EPCs or ad-hoc languages (block-structured ones) that the question of customization of resources (but also business objects) is posed.

In a similar vein, most approaches are based on conceptual modeling languages (UML ADs, EPCs, BPMN), and are hence focused on the customization of conceptual rather than executable process models. BPMN version 2.0 supports the specification of executable processes, but no customization approach so far covers the executable fea-

tures of BPMN. The Configurable Workflows approach is in essence the only approach to support customization of executable models (in YAWL, BPEL and SAP Workflow), down to the level of producing models that can be deployed in a BPMS.

Correctness of individualized models is only enforced by a handful of approaches. Among them, Templates and Rules and Feature Model Composition achieve correctness largely because of the syntactic restrictions (block-structuredness) they impose on customizable models and allowed extensions. Indeed, block-structured process models are guaranteed to be semantically correct (e.g. deadlock free). In a similar vein, CoSeNets achieve correctness support largely because of their syntactic restrictions (no cycles). C-EPCs and Configurable Workflows are the only approaches that achieve both syntactical and behavioral correctness support without such restrictions (EPCs). This is achieved via incremental checks that detect combinations of configuration options leading to incorrect models.

A clear majority of approaches support customization based on domain models, which may take the form of rules operating on customization parameters (as in Configurative Process Models and Provop), or feature models, questionnaire models or decision tables as discussed in Chapter 8. On the other hand, only two approaches (C-EPCs and Configurable Workflows) provide step-by-step guidance to users in making customization decisions.

It is reassuring to observe that the majority of approaches have corresponding tool implementations, at least to some extent, and about half of the approaches have been formalized syntactically and/or semantically. However, validation of the approaches in practice is in most cases either partially or totally missing.

10. CONCLUSION

Table I and the accompanying discussion put into evidence a wide heterogeneity of features and levels of sophistication across the surveyed approaches. At the same time, a number of core elements are present across all approaches. In essence, the surveyed approaches take as starting point a host process modeling language — usually a conceptual one rather than an executable one — on top of which a notion of *variations point* is added. Variations points are usually associated with specific model elements, which usually are control-flow elements (activities or routing nodes) but in some approaches can also be resources and objects. In most approaches, variation points in a customizable process model can be linked to elements in a domain model so as to facilitate model customization. At customization time, some approaches would offer guidance and iterative feedback to the user to ensure that the correct customization options are selected and that the resulting model is syntactically and behaviorally correct. However, support for incremental correctness checking is not a widespread feature, and in all but two cases, such support comes at the expense of restrictions on the structure of the models (block-structuredness or acyclicity).

One the most striking observations stemming from Table I is that the vast majority of approaches published in 2007 or later have not been validated (or have been, but only partially). One of the bottlenecks for validation — and more broadly a bottleneck for the adoption of customizable process models — is the effort involved in constructing customizable process models beyond trivial examples. Indeed, a customizable process model represents an entire family of processes. The amount of information required to construct such a model is usually an order of magnitude larger than that required to capture a model of one singular process.

This observation suggests that the question of how to construct customizable process models requires further attention. Recent research in this direction has led to algorithms for constructing a customizable process model from a collection on separate models of process variants [La Rosa et al. 2013], as well as algorithms for con-

structing a customizable process model from execution logs extracted from enterprise systems [Buijs et al. 2013]. One conclusion stemming from this research is that fully-automated approaches to customizable process model discovery are not likely to work in practice. Further, the effort involved in constructing and maintaining customizable models is considerable. Hence, there is a case for developing hybrid methods that combine user input with automated steps based on different information sources (e.g. logs, existing process models or other structured documentation). In any case, more effort should be put into creating customizable process models so as to support empirical studies in the field and to foster industrial uptake.

Beyond the fact that approaches to customizable process modeling have not been individually validated, it is surprising that these approaches have not been comparatively evaluated via case studies or other empirical means. Barring a conceptual comparative evaluation of C-EPCs and Provop using a cognitive psychology framework [Torres et al. 2013], there is virtually no empirical evidence to back any statement that one customizable process modeling approach is more usable than others in a particular context. The lack of comparative evaluations in the field is arguably a crucial gap that deserves in-depth attention.

Looking forward, the widespread adoption of multi-tenant enterprise systems has opened the possibility to use customizable process models as artifacts to drive the configuration of such systems. At present, the configuration of multi-tenant systems is manual and resource-intensive due to the large number of configuration points typically offered by such systems. Initial visions for multi-tenant system configuration based on customizable process models have been put forward [Fehling et al. 2011; van der Aalst 2011]. However, the realization and empirical validation of these visions remains an open avenue for future research.

REFERENCES

ACHER, M., COLLET, P., LAHIRE, P., AND FRANCE, R. B. 2010a. Composing Feature Models. In *Proceedings of SLE 2009*, M. van den Brand, D. Gašević, and J. Gray, Eds. Vol. 5969. Springer, 62–81.

ACHER, M., COLLET, P., LAHIRE, P., AND FRANCE, R. B. 2010b. Managing variability in workflow managing variability in workflow with feature model composition operators. In *9th International Conference on Software Composition (SC), Malaga, Spain*. Springer, 17–33.

ADAMS, M., TER HOFSTEDE, A., VAN DER AALST, W., AND EDMOND, D. 2007. Dynamic, Extensible and Context-Aware Exception Handling for Workflows. In *Proceedings of the OTM Conference on Cooperative information Systems (CoopIS 2007)*, F. Curbera, F. Leymann, and M. Weske, Eds. Lecture Notes in Computer Science Series, vol. 4803. Springer-Verlag, Berlin, 95–112.

ATKINSON, C., BAYER, J., AND MUTHIG, D. 2000. Component-Based Product Line Development: The KobrA Approach. In *Proceedings of the 1st Software Product Lines Conference (SPLC'00)*, P. Donohoe, Ed. Kluwer, 289–309.

ATKINSON, C., BOSTAN, P., BRENNER, D., FALCONE, G., GUTHEIL, M., HUMMEL, O., JUHASZ, M., AND STOLL, D. 2008. Modeling components and component-based systems in kobra. In *The Common Component Modeling Example: Comparing Software Component Models*. Lecture Notes in Computer Science Series, vol. 5153. Springer, 54–84.

BAIER, T., PASCALAU, E., AND MENDLING, J. 2010. On the suitability of aggregated and configurable business process models. In *Enterprise, Business-Process and Information Systems Modeling - 11th International Workshop, BPMDS 2010, and 15th International Conference, EMMSAD 2010, held at CAiSE 2010, Hammamet, Tunisia, Proceedings*, T. A. Halpin, J. Krogstie, S. Nurcan, E. Proper, R. Schmidt, and R. Ukor, Eds. Lecture Notes in Business Information Processing Series, vol. 50. 108–119.

BATORY, D. 2005. Feature Models, Grammars, and Propositional Formulas. In *Proceedings of the 9th International Conference on Software Product Lines (SPLC'05)*, J. Obbink and K. Pohl, Eds. Lecture Notes in Computer Science Series, vol. 3714. Springer, 7–20.

BECKER, J., ALGERMISSEN, L., DELFMANN, P., AND NIEHAVES, B. 2006. Configurable reference process models for public administrations. In *Encyclopedia of Digital Government*. 220–223.

BECKER, J., DELFMANN, P., DREILING, A., KNACKSTEDT, R., AND KUROPKA, D. 2004. Configurative Process Modeling – Outlining an Approach to increased Business Process Model Usability. In *Proceedings*

of the 14th Information Resources Management Association International Conference, M. Khosrow-Pour, Ed. IRM Press.

BECKER, J., DELFMANN, P., AND KNACKSTEDT, R. 2007a. Adaptive Reference Modeling: Integrating Configurative and Generic Adaptation Techniques for Information Models. In *Proceedings of the Reference Modeling Conference (RM'06)*, J. Becker and P. Delfmann, Eds. Springer, 27–58.

BECKER, J., JANIESCH, C., KNACKSTEDT, R., AND RIEKE, T. 2007b. Facilitating change management with configurative reference modelling. *International Journal for Information Systems and Change Management 2,* 1, 81–99.

BECKER, J., KNACKSTEDT, R., PFEIFFER, D., AND JANIESCH, C. 2007c. Configurative method engineering - on the applicability of reference modeling mechanisms in method engineering. In *Proc. of the 13th Americas Conference on Information Systems (AMCIS 2007)*. 1–12.

BOUCHER, Q., PERROUIN, G., DEPREZ, J.-C., AND HEYMANS, P. 2012. Towards configurable iso/iec 29110-compliant software development processes for very small entities. In *Proceedings of the 19th European Conference "Systems, Software and Services Process Improvement" (EuroSPI 2012)*, D. Winkler, R. V. O'Connor, and R. Messnarz, Eds. Communications in Computer and Information Science Series, vol. 301. Springer, 169–180.

BUIJS, J. C. A. M., VAN DONGEN, B. F., AND VAN DER AALST, W. M. P. 2013. Mining configurable process models from collections of event logs. In *Proceedings of the 11th International Conference on Business Process Management (BPM)*. Lecture Notes in Computer Science Series, vol. 8094. Springer-Verlag, Berlin, 33–48.

CASATI, F., CERI, S., PARABOSCHI, S., AND POZZI, G. 1999. Specification and Implementation of Exceptions in Workflow Management Systems. *ACM Transations on Database Systems 24,* 3, 405–451.

CIUKSYS, D. AND CAPLINSKAS, A. 2007. Reusing ontological knowledge about business processes in is engineering: Process configuration problem. *Informatica, Lith. Acad. Sci. 18,* 4, 585–602.

CZARNECKI, K. AND ANTKIEWICZ, M. 2005. Mapping Features to Models: A Template Approach Based on Superimposed Variants. In *Proceedings of the 4th International Conference on Generative Programming and Component Engineering*, R. Glück and M. R. Lowry, Eds. Springer, 422–437.

CZARNECKI, K. AND EISENECKER, U. 2000. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

CZARNECKI, K., HELSEN, S., AND EISENECKER, U. 2005. Formalizing Cardinality-Based Feature Models and Their Specialization. *Software Process: Improvement and Practice 10,* 1, 7–29.

DAVIS, R. AND BRABANDER, E. 2007. *ARIS Design Platform: Getting Started with BPM*. Springer.

DELFMANN, P., JANIESCH, C., KNACKSTEDT, R., RIEKE, T., AND SEIDEL, S. 2006. Towards Tool Support for Configurative Reference Modeling – Experiences from a Meta Modeling Teaching Case. In *Proceedings of the 2nd International Workshop on Meta-Modelling (WoMM'06)*, S. Brockmans, J. Jung, and Y. Sure, Eds. LNI Series, vol. 96. GI, 61–83.

DELFMANN, P., RIEKE, T., AND SEEL, C. 2007. Supporting enterprise systems introduction by controlling enabled configurative reference modelling. In *Proceedings of the Reference Modeling Conference (RM'06)*, J. Becker and P. Delfmann, Eds. Springer, 79–102.

DESEL, J. AND ESPARZA, J. 1995. *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science Series, vol. 40. Cambridge University Press.

DREILING, A., ROSEMANN, M., VAN DER AALST, W., HEUSER, L., AND SCHULZ, K. 2006. Model-Based Software Configuration: Patterns and Languages. *European Journal of Information Systems 15,* 6, 583–600.

DREILING, A., ROSEMANN, M., VAN DER AALST, W., SADIQ, W., AND KHAN, S. 2005. Model-Driven Process Configuration of Enterprise Systems. In *Wirtschaftsinformatik 2005: eEconomy, eGovernment, eSociety*, O. K. Ferstl, E. Sinz, S. Eckert, and T. Isselhorst, Eds. Physica–Verlag, 687–706.

ELLIS, C., KEDDARA, K., AND ROZENBERG, G. 1995. Dynamic change within workflow systems. In *Proceedings of the Conference on Organizational Computing Systems*, N. Comstock, C. Ellis, R. Kling, J. Mylopoulos, and S. Kaplan, Eds. ACM SIGOIS, ACM Press, New York, Milpitas, California, 10 – 21.

FEHLING, C., LEYMANN, F., SCHUMM, D., KONRAD, R., MIETZNER, R., AND PAULY, M. 2011. Flexible process-based applications in hybrid clouds. In *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 81–88.

FETTKE, P. AND LOOS, P. 2003. Classification of Reference Models - A Methodology and its Application. *Information Systems and e-Business Management 1,* 1, 35–53.

GEORGAKOPOULOS, D., HORNICK, M., AND SHETH, A. 1995. An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases 3*, 119–153.

GOTTSCHALK, F., VAN DER AALST, W., AND JANSEN-VULLERS, M. 2007. SAP WebFlow Made Configurable: Unifying Workflow Templates into a Configurable Model. In *International Conference on Business Process Management (BPM 2007)*, G. Alonso, P. Dadam, and M. Rosemann, Eds. Lecture Notes in Computer Science Series, vol. 4714. Springer-Verlag, Berlin, 262–270.

GOTTSCHALK, F., VAN DER AALST, W., JANSEN-VULLERS, M., AND LA ROSA, M. 2008. Configurable Workflow Models. *International Journal of Cooperative Information Systems 17,* 2, 177–221.

GOTTSCHALK, F., WAGEMAKERS, T., JANSEN-VULLERS, M., VAN DER AALST, W., AND LA ROSA, M. 2009. Configurable Process Models: Experiences From a Municipality Case Study. In *Proc. of CAiSE*, P. van Eck, J. Gordijn, and R. Wieringa, Eds. Lecture Notes in Computer Science Series, vol. 5565. Springer-Verlag, Berlin, 486–500.

HALLERBACH, A., BAUER, T., AND REICHERT, M. 2008. Managing Process Variants in the Process Life Cycle. In *10th Int'l Conf. on Enterprise Information Systems (ICEIS'08)*. Vol. 22. 154–161.

HALLERBACH, A., BAUER, T., AND REICHERT, M. 2009a. Guaranteeing Soundness of Configurable Process Variants in Provop. In *CEC*. IEEE, 98–105.

HALLERBACH, A., BAUER, T., AND REICHERT, M. 2009b. Issues in Modeling Process Variants with Provop. In *Business Process Management 2008 Workshops*, D. Ardagna, M. Mecella, and J. Yang, Eds. Lecture Notes in Business Information Processing Series, vol. 17. Springer.

HALLERBACH, A., BAUER, T., AND REICHERT, M. 2010. Capturing variability in business process models: The provop approach. *Journal of Software Maintenance and Evolution: Research and Practice 22,* 6-7, 519–546.

HEINL, P., HORN, S., JABLONSKI, S., NEEB, J., STEIN, K., AND TESCHKE, M. 1999. A Comprehensive Approach to Flexibility in Workflow Management Systems. In *Work Activities Coordination and Collaboration (WACC'99)*, G. Georgakopoulos, W. Prinz, and A. Wolf, Eds. ACM Press, San Francisco, 79–88.

IDS SCHEER. 2002. ARIS Process Performance Manager (ARIS PPM): Measure, Analyze and Optimize Your Business Process Performance (whitepaper). IDS Scheer, Saarbruecken, Gemany, http://www.ids-scheer.com.

KANG, K., COHEN, S., HESS, J., NOVAK, W., AND PETERSON, A. 1990. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University.

KITCHENHAM, B. 2004. Procedures for undertaking systematic reviews. Tech. rep., Computer Science Department, Keele University (TR/SE- 0401) and National ICT Australia Ltd. (0400011T.1).

KORHERR, B. AND LIST, B. 2007. A UML 2 profile for variability models and their dependency to business processes. In *Proceedings of the 18th International Workshops on Database and Expert Systems Applications, Regensburg, Germany*. IEEE Computer Society, 829–834.

KULKARNI, V. AND BARAT, S. 2011. Business process families using model-driven techniques. Springer, 314–325.

KUMAR, A. AND YAO, W. 2009. Process materialization using templates and rules to design flexible process models. In *RuleML*, G. Governatori, J. Hall, and A. Paschke, Eds. Lecture Notes in Computer Science Series, vol. 5858. Springer, 122–136.

KUMAR, A. AND YAO, W. 2012. Design and management of flexible process variants using templates and rules. *Computers in Industry 63,* 2, 112–130.

LA ROSA, M., DUMAS, M., TER HOFSTEDE, A., AND MENDLING, J. 2011. Configurable Multi-Perspective Business Process Models. *Information Systems 36,* 2.

LA ROSA, M., DUMAS, M., UBA, R., AND DIJKMAN, R. M. 2013. Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering Methodology 22,* 2, 11.

LA ROSA, M., GOTTSCHALK, F., DUMAS, M., AND VAN DER AALST, W. 2008. Linking Domain Models and Process Models for Reference Model Configuration. In *Business Process Management 2007 Workshops*, A. ter Hofstede, B. Benatallah, and H. Paik, Eds. Lecture Notes in Computer Science Series, vol. 4928. Springer, 417–430.

LA ROSA, M., VAN DER AALST, W., DUMAS, M., AND TER HOFSTEDE, A. 2009. Questionnaire-based Variability Modeling for System Configuration. *Software and Systems Modeling 8,* 2, 251–274.

LOHMANN, N., VERBEEK, E., AND DIJKMAN, R. 2009. Petri net transformations for business processes - a survey. *T. Petri Nets and Other Models of Concurrency 2*, 46–63.

LÖNN, C.-M., UPPSTRÖM, E., WOHED, P., AND JUELL-SKIELSE, G. 2012. Configurable process models for the swedish public sector. In *Proceedings of the 24h International Conference on Advanced Information Systems Engineering (CAiSE 2012)*, J. Ralyté, X. Franch, S. Brinkkemper, and S. Wrycza, Eds. Lecture Notes in Computer Science Series, vol. 7328. Springer, 190–205.

MILI, H., TREMBLAY, G., JAOUDE, G. B., LEFEBVRE, E., ELABED, L., AND EL-BOUSSAIDI, G. 2010. Business process modeling languages: Sorting through the alphabet soup. *ACM Computing Surveys 43,* 1, Article 4.

MOON, M., HONG, M., AND YEOM, K. 2008. Two-level variability analysis for business process with reusability and extensibility. In *Proceedings of the 32nd Annual IEEE International Computer Software and Applications Conference (COMPSAC), Turku, Finland*. IEEE Computer Society, 263–270.

NGUYEN, T., COLMAN, A. W., AND HAN, J. 2011. Modeling and managing variability in process-based service compositions. In *Proceedings of the 9th International Conference on Service-Oriented Computing (ICSOC), Paphos, Cyprus*. Springer, 404–420.

PUHLMANN, F., SCHNIEDERS, A., WEILAND, J., AND WESKE, M. 2005. Variability Mechanisms for Process Models. PESOA-Report TR 17/2005, Process Family Engineering in Service-Oriented Applications (PESOA). BMBF-Project. 30 June.

RAZAVIAN, M. AND KHOSRAVI, R. 2008. Modeling Variability in Business Process Models Using UML. In *Proceedings of the 5th International Conference on Information Technology: New Generations (ITGN'08)*, S. Latifi, Ed. 82–87.

REICHERT, M. AND DADAM, P. 1998. ADEPT$_{flex}$: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems 10,* 2, 93–129.

REICHERT, M. AND WEBER, B. 2012. *Enabling Flexibility in Process-Aware Information Systems: Challenges, Methods, Technologies*. Springer-Verlag, Berlin.

REIJERS, H., MANS, R., AND VAN DER TOORN, R. 2009. Improved Model Management with Aggregated Business Process Models. *Data Knowl. Eng. 68,* 2, 221–243.

REIJERS, H., RIGTER, J., AND VAN DER AALST, W. 2003. The Case Handling Case. *International Journal of Cooperative Information Systems 12,* 3, 365–391.

REINHARTZ-BERGER, I., SOFFER, P., AND STURM, A. 2009. Organizational Reference Models: Supporting an Adequate Design of Local Business Processes. *International Journal of Business Process Integration and Management 4,* 2, 134–149.

REINHARTZ-BERGER, I., SOFFER, P., AND STURM, A. 2010. Extending the Adaptability of Reference Models. *IEEE Transactions on Systems, Man and Cybernetics part A 40,* 5.

REINHARTZ-BERGER, I. AND STURM, A. 2007. Enhancing UML Models: A Domain Analysis Approach. *Journal on Database Management (special issue on UML Topics) 19,* 1, 74–94.

RINDERLE, S., REICHERT, M., AND DADAM, P. 2004. Correctness Criteria For Dynamic Changes in Workflow Systems: A Survey. *Data and Knowledge Engineering 50,* 1, 9–34.

RIPON, S., TALUKDER, K., AND MOLLA, K. 2010. Modelling variability for system families. *Malaysian Journal of Computer Science 16,* 1, 37–46.

ROSEMANN, M. 2003. Application Reference Models and Building Blocks for Management and Control (ERP Systems). In *Handbook on Enterprise Architecture*, P. Bernus, L. Nemes, and G. Schmidt, Eds. Springer, 596–616.

ROSEMANN, M. AND VAN DER AALST, W. 2003. A Configurable Reference Modelling Language. BPM Center Report BPM-03-08, BPMcenter.org. (Later published as ROSEMANN, M. AND AALST, W. 2007).

ROSEMANN, M. AND VAN DER AALST, W. M. P. 2007. A Configurable Reference Modelling Language. *Information Systems 32,* 1, 1–23.

SADIQ, S., SADIQ, W., AND ORLOWSKA, M. 2001. Pockets of Flexibility in Workflow Specification. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER 2001)*. Lecture Notes in Computer Science Series, vol. 2224. Springer-Verlag, Berlin, 513–526.

SCHNIEDERS, A. 2006. Variability Mechanism Centric Process Family Architectures. M. Riebisch, P. Tabeling, and W. Zorn, Eds. IEEE Computer Society, 289–298.

SCHNIEDERS, A. AND PUHLMANN, F. 2006. Variability Mechanisms in E-Business Process Families. In *Proceedings of the 9th International Conference on Business Information Systems (BIS'06)*, W. Abramowicz and H. Mayr, Eds. LNI Series, vol. 85. GI, 583–601.

SCHOBBENS, P.-Y., HEYMANS, P., AND TRIGAUX, J.-C. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *Proceedings of the 14th IEEE International Conference on Requirements Engineering (RE'06)*, M. Glinz and R. Lutz, Eds. IEEE Computer Society, 136–145.

SCHONENBERG, H., MANS, R., RUSSELL, N., MULYAR, N., AND VAN DER AALST, W. 2008. Process Flexibility: A Survey of Contemporary Approaches. In *Advances in Enterprise Engineering I*, J. Dietz, A. Albani, and J. Barjis, Eds. Lecture Notes in Business Information Processing Series, vol. 10. Springer-Verlag, Berlin, 16–30.

SCHUNSELAAR, D., VERBEEK, E., VAN DER AALST, W., AND REIJERS, H. 2011. Creating sound and reversible configurable process models using CoSeNets. Tech. Rep. BPM-11-21, BPM Center Report.

SCHUNSELAAR, D., VERBEEK, E., VAN DER AALST, W., AND REIJERS, H. 2012. Creating sound and reversible configurable process models using CoSeNets. In *Proc. of Business Information Systems*. Lecture Notes in Business Information Processing Series, vol. 117. Springer, 24–35.

TORRES, V., ZUGAL, S., WEBER, B., REICHERT, M., AYORA, C., AND PELECHANO, V. 2013. A qualitative comparison of approaches supporting business process variability. In *Business Process Management Workshops*. Lecture Notes in Business Information Processing Series, vol. 132. Springer, 560–572.

VALENCA, G., ALVES, C., ALVES, V., AND NIU, N. 2013. A systematic mapping study on business process variability. *Int. Journal of Computer Science & Information Technology 5,* 1.

VAN DER AALST, W. AND BASTEN, T. 2002. Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science 270,* 1-2, 125–203.

VAN DER AALST, W., DREILING, A., GOTTSCHALK, F., ROSEMANN, M., AND JANSEN-VULLERS, M. 2006. Configurable Process Models as a Basis for Reference Modeling. In *Proceedings of the Business Process Management 2005 Workshops*, E. Kindler and M. Nüttgens, Eds. Springer, 76–82.

VAN DER AALST, W., DUMAS, M., GOTTSCHALK, F., TER HOFSTEDE, A., LA ROSA, M., AND MENDLING, J. 2010. Preserving Correctness During Business Process Model Configuration. *Formal Aspects of Computing 22,* 3, 459–482.

VAN DER AALST, W., HEE, K., TER HOFSTEDE, A., SIDOROVA, N., VERBEEK, H., VOORHOEVE, M., AND WYNN, M. 2011. Soundness of Workflow Nets: Classification, Decidability, and Analysis. *Formal Aspects of Computing 23,* 3, 333–363.

VAN DER AALST, W., LOHMANN, N., AND LA ROSA, M. 2012. Correctness Ensuring Process Configuration: An Approach Based on Partner Synthesis. *Information Systems 37,* 6, 574–592.

VAN DER AALST, W., PESIC, M., AND SCHONENBERG, H. 2009. Declarative Workflows: Balancing Between Flexibility and Support. *Computer Science - Research and Development 23,* 2, 99–113.

VAN DER AALST, W. M. P. 2011. Business process configuration in the cloud: How to support and analyze multi-tenant processes? In *Proceedings of the 9th IEEE European Conference on Web Services (ECOWS)*. IEEE, 3–10.

WEBER, B., REICHERT, M., AND RINDERLE-MA, S. 2008. Change Patterns and Change Support Features: Enhancing Flexibility in Process-Aware Information Systems. *Data and Knowledge Engineering 66,* 3, 438–466.

WOLF, K. 2009. Does my Service Have Partners? In *Transactions on Petri Nets and Other Models of Concurrency II*, K. Jensen and W. van der Aalst, Eds. Lecture Notes in Computer Science Series, vol. 5460. Springer-Verlag, Berlin, 152–171.