The semantics of hybrid process models

Slaats, Tijs; Schunselaar, Dennis M. M.; Maggi, Fabrizio M.; Reijers, Hajo A.

Published in:
On the move to meaningful internet systems: OTM 2016 conferences

DOI:
10.1007/978-3-319-48472-3_32

Publication date:
2016

Document Version
Peer reviewed version

The Semantics of Hybrid Process Models

Tijs Slaats[1], Dennis M. M. Schunselaar[2],
Fabrizio M. Maggi[3], and Hajo A. Reijers[2,4⋆]

[1] University of Copenhagen, Denmark
[2] Vrije Universiteit Amsterdam, The Netherlands
[3] University of Tartu, Estonia
[4] Eindhoven University of Technology, The Netherlands
slaats@di.ku.dk, d.m.m.schunselaar@vu.nl, f.m.maggi@ut.ee,
h.a.reijers@vu.nl

Abstract. In the area of business process modelling, declarative notations have been proposed as alternatives to notations that follow the dominant, imperative paradigm. Yet, the choice between an imperative or declarative style of modelling is not always easy to make. Instead, a mixture of these styles is sometimes preferable. This observation has underpinned recent calls for so-called *hybrid* process modelling notations. In this paper, we present a formal semantics for these. In our proposal, a hybrid process model is hierarchical, where each of its sub-processes may be specified in either an imperative or declarative fashion. The semantics we provide will allow modelling communities to build on the benefits of existing imperative and declarative modelling notations, instead of spending their energy on inventing new ones.

Keywords: Hybrid process model, Semantics, Petri net, Declare

1 Introduction

Process modelling languages are used to visually capture how business processes are carried out. The mainstream notations in use are *imperative* in nature, in the sense that their corresponding models describe in full detail the exact behaviour that business processes may exhibit. Recently, as an alternative stream, proposals have been made for process modelling languages that follow a *declarative* style. Declarative models leave implicit in what exact sequence process activities must be carried out. Rather, the emphasis is on the constraints that ought to be respected during the execution of a process. In a declarative model, any behaviour that does not comply with such constraints is forbidden, but anything that is not explicitly ruled out is permissible. Examples of declarative modelling techniques are Declare [18,27], DCR graphs [11], and SCIFF [15].

⋆ This work is supported in part by the Hybrid Business Process Management Technologies project (funded by the Danish Council for Independent Research) and the Computational Artifacts project (VELUX 33295, 2014-2017). The first author would like to acknowledge Søren Debois and Morten Marquard for their valuable feedback.

What motivates our work is the surmise that many real-life processes [7] can be characterized as *combinations* of structured and unstructured parts. Therefore, for such processes, *hybrid* models would actually result in their most compact and simple description. For the parts of a process that are highly flexible, a declarative modelling approach leads to a simple description of such "pockets of flexibility" [22]. Instead of describing all feasible behaviour, the focus is on forbidding what is not allowed. By contrast, for parts of the process that are highly structured, an imperative description is preferable: for these parts, it is simpler to describe the limited behaviour that is allowed than the more numerous situations that are to be excluded. In previous work [21], we presented evidence for the prevalence of business processes that contain both structured and unstructured parts, an insight that followed from a workshop with BPM professionals. The involved professionals indicated that a hybrid process modelling technique would indeed be more attractive than a purely declarative or imperative one.

Previously, a proposal for a hybrid process platform was proposed in [1]. It differs from our approach in that it proposes an entirely new notation, i.e., the task model. Instead, we show how *existing* notations, which may have gained considerable popularity in modelling communities, can be integrated into a hybrid modelling notation. This was attempted to some extent with the modelling approach known as Flexibility-as-a-Service (FAAS) [26], where loosely-structured processes could be modelled in Declare and highly-structured processes in YAWL; these could reference each other as sub-processes. While the authors describe how this idea can be realised as a Service-Oriented Architecture, they left open the *exact formal semantics* of such a hybrid model.

In the current paper, we are inspired by the same considerations as in [1] while extending the contributions of [26] as follows: 1) we widen the *scope* of a hybrid modelling notation through a notation-independent treatment of the topic, 2) we consider in detail the *challenges* that arise when the goal is to arrive at a sensible hybrid model from imperative and declarative parts, and 3) we provide a *formal semantics* for hybrid models based on these considerations. In our treatment of the topic, we will employ Petri nets and Declare as canonical representatives of respectively the imperative and declarative paradigms. Even though we exclusively use these specific notations, we expect that it will become clear to the reader that our approach can entwine most other notations.

Against this background, the paper is structured as follows. In Section 2, we discuss related work. Then, in Section 3, we motivate our work using a case study performed on the funding application process of a Danish foundation, for which we constructed a hybrid process model. Section 4 describes the challenges that motivate our chosen formalisation. In Section 5, we formally define hybrid models and their semantics. We conclude this paper with a reflection on the presented work and future steps in Section 6.

2 Related Work

Two different styles of approaches can be distinguished with respect to the development of a conceptual modelling language. The first of these is the most

straightforward one, namely to design a language from scratch. We will refer to languages that are designed in this way as *autonomous languages*. The second is to combine languages from existing ones, which is the style we will adhere to in this paper. We will refer to these as *hybrid languages*. Both of these streams will be discussed next, where we will pay specific attention to the perspective of process modelling in contrast to other areas of conceptual modelling.

2.1 Autonomous languages

We already referred to imperative process modelling languages, where BPMN is a good example of a widely adopted language that was newly created. More specifically, its first version was proposed by the BPMI consortium in 2004. On the declarative side of the process modelling spectrum, there is Declare [18,27], which has been developed as part of a PhD project at the TU Eindhoven. With this language, a process is specified via a set of constraints between activities, which must be satisfied by every execution of the process. Declare constraints are captured based on templates. Templates are patterns that define parametrised classes of properties, while constraints are their concrete instantiations. Constraints have a graphical representation. The semantics of templates can be formalised using different logics [16], for example LTL for finite traces. The reader can refer to [27] for a full description of the language. Other process modelling approaches that embrace "pockets of flexibility" or unstructured process parts have been proposed as well. We mentioned [1], which proposes an entirely different type of modelling notation. In [22], flexibility pockets can be defined at build-time in a way that is highly similar to a declarative modelling style; at runtime one has to pick a specific procedural instantiation of the workflow that fits the definition. The major drawback of introducing an autonomous language is that it has to acquire a user base from scratch. In practice, it may be difficult to convince active users of a language to leave it behind and adopt an entirely new one, which motivates our choice for a hybrid language. The emphasis is then on motivating modellers to expand rather than to abolish their knowledge of a particular approach.

2.2 Hybrid languages

The combination of conceptual modelling languages to arrive at a hybrid language is mostly pursued when the individual languages cover a *different* perspective on the object that is to be modelled. The idea is that such languages complement each other. A good illustration is the integration of the BPMN process modelling language and the SRML rule modelling language to respectively capture the process and regulatory perspectives on an organisational procedure [32]. A comparable approach, in the sense that it combines existing languages with different focal points, is described in [13]. Here, the modelling of processes is linked to the modelling of business goals.

To combine languages that take the same perspective on an object is relatively rare, although it is pursued in those cases where individual languages

offer distinct advantages. The first case of combining two existing process modelling languages that we are aware of is the Action Port Model, which combines the input/output characteristics of the PPM language with the language-action style of ActionWorkflow and WooRKS [2]. More recently, the combined use of YAWL [28] and Declare was proposed for capturing highly and loosely structured processes respectively [18]. This approach is similar to ours in that it describes a hierarchical ordering of sub-processes where each sub-process can be described in one notation or the other. However, the paper only gives an architectural overview of their proposed solution and does neither provide a formal semantics or a thorough consideration of the challenges that arise from combining the different paradigms. Two other approaches that combine procedural and declarative elements worth noting are Flexibility-as-a-Service (FAAS) [26] and the GSM model [25]. The former expands on the possibilities of combining Declare and YAWL, but also does not provide a formal semantics. The GSM model is a modelling language that is actively being developed, but it has as disadvantages that different groups have developed different variants and that it is to a certain extent proprietary (IBM). With our work, we aim to propose in a consistent and transparent manner a hybrid process modelling language that combines imperative and declarative paradigms.

Recent research has again put into evidence synergies between imperative and declarative approaches [19,21]. Accordingly, hybrid process modelling notations have been proposed. In particular, [3] provides a conservative extension of BPMN for declarative process modelling, namely BPMN-D, and shows that Declare models can be transformed into readable BPMN-D models. [30] proposes to extend Coloured Petri nets with the possibility of linking transitions to Declare constraints directly. The notion of *transition enablement* is extended to handle declarative links between transitions. A recent implementation of this technique is made available in CPN Tools 4.0 [29]. [4,24] extend the work in [30] by defining a semantics based on mapping Declare constraints to R/I-nets and by proposing modelling guidelines for the mixed paradigm.

These approaches differ from our work in that a fully mixed approach is taken, whereas our approach is hierarchical. We have been motivated to clearly separate declarative and imperative model parts, because we expect that mixing two notations within the same sub-process would strongly and negatively influence the perceptual discriminability of the various model elements, cf. [17]. Consider, for example, that many Declare constraints take on arrow-like shapes, similar to how the transition relation is visualized in a Petri net. In work on sense-making of declarative models, one of the major issues identified occurs, indeed, when a user is confronted with a combination of constraint arrows [9]. We also feel that process decomposition is a natural way to separate complexity, allows for the re-use of model fragments, and supports both bottom-up and top-down modelling. Furthermore, existing approaches are only applicable to specific notations, whereas we aim at providing a general semantics for hybrid models using any notation. Note that a hierarchical approach similar to ours has been

used in [12] for the automated discovery of hybrid models. This work, however, does not formally define hybrid models and their semantics.

3 Use Case for Hybrid Process Models

As a practical use case, we consider the application process of the Dreyer Foundation, which provides funding for promoting the development of the lawyers' and architects' professions[1]. In 2014, the largely paper driven processes at the Dreyer Foundation were digitalised in cooperation with the Danish software developer Exformatics and researchers from the IT University of Copenhagen [5]. At the core of the electronic case management (ECM) system provided by Exformatics is a declarative workflow engine [14][23] based on the Dynamic Condition Response (DCR) Graphs [6][10] notation.

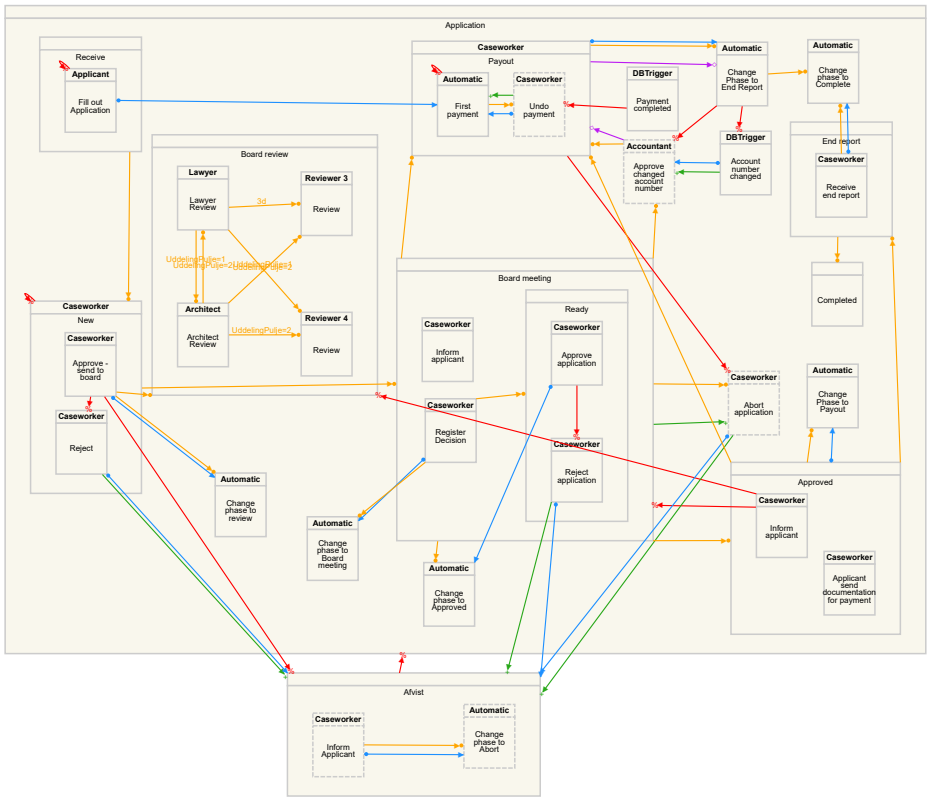

Fig. 1. DCR Graph of the Dreyers Application Process.

[1] `http://dreyersfond.dk/en/`

An elemental part of the Dreyers ECM solution is a declarative DCR Graph model of the main *application* process, which describes what happens from the submission of an application up to the completion of a funded project. As is shown in Fig. 1 the declarative model is fairly complex and therefore one needs to consider if using a declarative notation is truly the most concise way of representing the process. This question was investigated in [7] by using the real-life logs of the process to determine if a simpler flow-based model could be discovered algorithmically. The investigation showed that while parts of the process were quite structured, there were also flexible parts which lead to a high degree of variation in traces and none of the more widely accepted mining algorithms could find a sufficiently simple, precise and fit flow-based model to compete with the declarative original. The authors noted that a hybrid model could potentially make for a good alternative, but left this for future work.

In this paper, we addressed this open question and constructed a hybrid model of the Dreyer application process by hand. As input we used 1) the declarative model, 2) the logs of actual behaviour being exhibited, 3) a meeting with the main case worker at Dreyer Foundation and 4) several discussions with the responsible process modeller at Exformatics. When modelling parts of the process using a procedural notation, we followed the procedural paradigm: we gave priority to the logs as a source of possible behaviour exhibited in practice and limited the procedural model to showing these paths as opposed to showing all possible paths allowed by the DCR Graph. In particular, the DCR Graph often allows for repetition of activities, but unless this repetition was observed in the logs we did not model it in the procedural model. When encountering a high degree of variability in the log, we modelled these parts using a declarative notation and gave greater weight to the original requirement constraints as defined by the DCR Graph. As a result, perhaps not surprisingly, the hybrid model exhibits the most flexibility in declarative sub-processes, but these are also the parts of the process where most variability was observed in the logs.

The resulting model is shown in Fig. 2. The main process is described imperatively as a Petri net; it starts by an applicant filling out an application, followed by a screening of the application by the main case worker. The case worker decides if the application follows the formal requirements and has the opportunity to approve or reject it. A rejection is not immediately final: until the applicant is informed of the rejection it is possible to re-decide and accept the application anyway, modelled through the silent transition looping back to the previous place. If, after a rejection, the applicant is informed of the decision, the process ends by changing the status of the case to *Abort*. If the application is approved for further review, the status of the case is changed to *Review* and a *Review* sub-process is started. The sub-process is highly flexible and therefore modelled declaratively, which we will discuss later. After the review process is completed, a decision is registered: either the application is accepted or rejected. Similar to the screening, a rejection is not immediately final; until the applicant has been informed, it is possible to repeat the review process and arrive at a new decision. After a rejection, the process follows the same steps as a screening

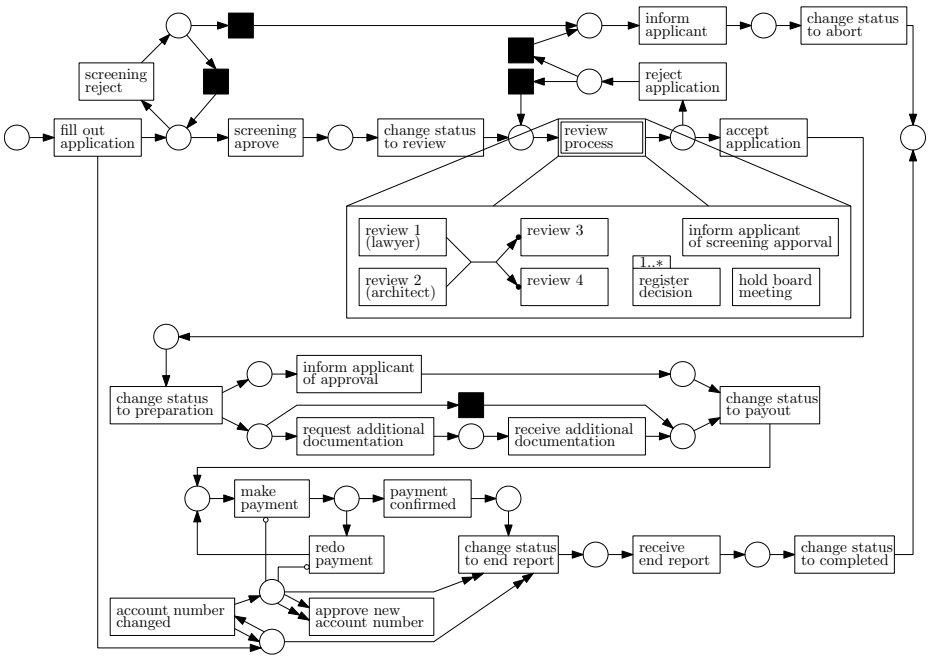

Fig. 2. Hybrid application model for the Dreyers foundation.

rejection: the applicant is informed and the status changed to *abort*. If the board accepts the application, the status of the process is changed to *Preparation*. The case worker will now inform the applicant of the approval and has the option to, in parallel, request additional documentation from the applicant. For example, if the application involved the purchase of some particularly expensive items, a quote from a potential seller may be required. Once the applicant is informed and all the documentation is in order, the status of the process is changed to *Payout*. The following part of the process is fairly flexible, but was not modelled declaratively because it has essentially two separate entry points: first of all, after changing the status to payout, it becomes possible to make a payment. Once the payment is confirmed by the financial system, the process moves on and changes the status to *End Report*. The case worker can also redo the payment, for example if no confirmation is received in due time, this activity removes the current payment and allows a new payment to be made. Secondly, there is the possibility for the case worker to change the account number to which the grant will be transferred. This activity becomes available right after the submission of the application and thereby creates a second entry point for the payment part. Once the account number is changed, no payments should be made or redone before the new number has been approved. This is done by inhibiting the *Make payment* and *Redo payment* activities after the account number has changed. Note that changing the status to *End Report* also removes the ability of changing and approving the account number. Finally, once an end report is received, the status of the process is changed to *Completed*.

Table 1. Overview of the Declare constraints used in this paper.

Graphic	Name	Semantics
[init] A	Init	*A* should be the first action.
[1..n] A	Existence + Absence	Between 1 and n times *A* needs to be executed.
A → B	Response	*A* needs to be eventually followed by *B*.
A → B	Precedence	*B* needs to be preceded by *A*.
A ⇒ B	Chain response	*A* needs to be directly followed by *B*.
A ⇏ B	Not chain succession	*A* cannot be directly followed by *B*.

Prior to presenting the Declarative review process, we summarise the semantics of the Declare constraints used within this example and the remainder of the paper in Table 1. In the review process (Fig. 2), all activities can happen multiple times and a number of them are unconstrained: at any point it is possible to inform the applicant that their application was approved for review, hold a board meeting, and to register a decision. Note that in practice informing the applicant appears to occur at most once for each application, but since it is not constrained to a limited number of executions in the DCR Graph and we employ the declarative paradigm in the creation of the Declare model, we do not add unnecessary constraints. Registering the decision should occur at least once, as modelled by the existence constraint. There are four reviewers, two of which are experts in either the legal or architecture field. Depending on the type of application one of the experts should be the first to submit a review. As we wish to avoid including data in the current model, we abstract this requirement to that at least one of the expert reviews should happen before either of the non-expert reviews can happen. The existence constraint on register decision is the only liveness constraint (placing requirements on future behaviour) in the model and therefore the Declare model is satisfied and the sub-process can end whenever register decision has been done at least once. One may note that this means that doing actual reviews is not required by the model and one can simply register a decision and then accept the application. Closer inspection will show that this is the case for the DCR Graph as well. The reason being that in some cases the reviewers simply want to convey their decisions orally and not be constrained to using the underlying IT system. Therefore, the caseworker is able of registering a decision even if no reviews have been submitted.

The hybrid model was well received by the process modeller at Exformatics, who agreed that it provided an accurate and concise representation of the actual process in use at Dreyers. Discussing the model even led to the discovery of a small error in the original, where the secretary was not able of changing her decision after she had chosen to screening reject an application. The hybrid model helped in the discovery of this mistake because the relevant part of the process could be intuitively represented as a flow, whereas the declarative model required a number of constraints whose interaction was not directly obvious to the modeller.

The model already provided value to the process modeller, but to avoid ambiguity in discussions one needs to make precise *exactly* what behaviour it represents by providing a *formal semantics*. Doing so will also facilitate the creation of modelling, simulation, execution and analysis tools based on the approach. As a formal semantics for such a hierarchical hybrid approach has been lacking in the literature to date, we set out to define it ourselves. While doing so for the current example is fairly straightforward, providing a semantics for hybrid models in general posed several challenges that needed to be considered in detail. We discuss these challenges in the following section.

4 Formalisation Challenges

While formalising the semantics of hybrid models, we encountered various challenges. Within this section, we discuss these challenges in detail and show how they motivate our chosen formalisation.

4.1 Identifying the Challenges

The challenges presented in this section originate from in-depth discussions between the authors on the exact semantics of hybrid models. Next to this, these challenges have been found by considering the relevant aspects of a declarative/imperative process modelling language, i.e., the enablement/execution of an action. As such, these challenges follow naturally from going from a flat model with atomic actions to a hierarchical model where actions are no longer atomic in conjunction with termination not explicitly being encoded (in case of Declare). The following challenges were raised:

(Q1) How can we combine the open and closed world assumptions made by different formalisms?
(Q2) When does a transition representing a declarative sub-process become enabled and/or has fired?
(Q3) When is a constraint on a sub-process activated and/or fulfilled?

For all challenges to be partly met, we have introduced a notion of a *context*. This context ensures that there is a unique mapping between actions and sub-processes. Furthermore, by allowing the context to encompass more actions than the ones modelled in a sub-processes, we close the gap between the open and closed world assumptions. The challenges raised related to enablement/fired and activated/fulfilled of a sub-process were addressed by a thorough analysis of the possible combinations of a declarative and imperative process models, e.g., an imperative process model within a declarative process model or a declarative process model within a declarative process model. This gave raise to two solutions: (1) the termination of a declarative sub-process requires the execution of an action, and (2) an interpretation of the negation of a sub-process in a declarative constraint. One might wonder whether moving from atomic instantaneous actions to sub-processes in a declarative language might give additional

challenges with respect to the concurrent execution of actions. By treating concurrency for declarative languages the same as for imperative languages, we did not encounter any additional challenges.

In the remainder of this section, we elaborate further over the challenges.

4.2 Closing the open/closed world gap by means of a context

The first challenge we encountered was the open world assumption of Declare. In other words, *everything* is allowed unless stated otherwise. This means that actions not explicitly modelled in the Declare model can indeed occur.

Take the example hybrid model in Fig. 3. Within this hybrid model, first the constraint specified by sub-process X is active. When X has been executed, the constraint specified by sub-process Y is active. Due to the open world assumption of Declare, just performing C would be valid within sub-process X. It is, however, not a valid execution of sub-process Y. One can either have a *permissive* approach or a *non-permissive* approach to resolve this ambiguity. Within a permissive approach, one would argue that C was executed within X. In the non-permissive approach, one would argue that C was executed within Y. Since in the permissive approach, any sequence of actions is allowed except for those sequences which adhere to the following general pattern: $\langle \ldots, A, (\neg B)^*, C, (\neg B \wedge \neg D)^* \rangle$, one would probably be inclined, in this example, to follow the non-permissive approach.

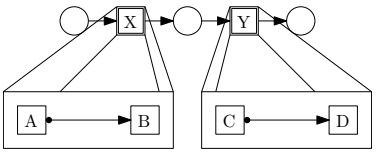

Fig. 3. Hybrid model where one can argue whether just performing C is a valid execution.

If we would take the hybrid model in Fig. 2, then every instance of the process starts with a *fill out application* action and ends with either a *change status to abort* or a *change status to completed* action. In the *review process* action, the execution has to adhere to the Declare model. Within this Declare model, it is possible to execute the *change status to abort* or *change status to completed* actions (open world assumption). Although this does not result in the kind of ambiguity mentioned earlier, one can argue that the modeller did not intend to change the status within the review sub-process. After all, she already modelled it in the Petri net.

Given the above discussion, one may prefer to limit the execution to only those actions modelled within the sub-process. This has as main disadvantage that the execution semantics of a Declare model changes. Take for instance the hybrid model in Fig. 4. We have the parallel execution of two Declare models. In the top Declare model, the execution always starts with A, A cannot be directly

followed by B, and there needs to be between 1 and n Bs. If one would only allow for the execution of the modelled actions, then after A, one can only execute A, and a B can never be executed in the top model. As a result, the top Declare model cannot terminate since it requires that B is executed at least once. If the top Declare model would not be part of a hybrid model, then, thanks to the open world view, there could be an action F after which B could be executed. After this B, the Declare model could terminate. This results in *different* execution semantics for the *same* Declare model.

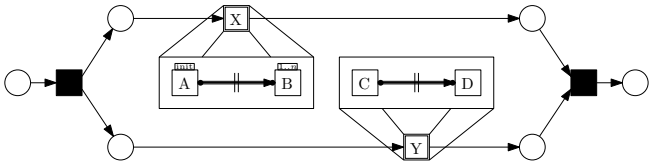

Fig. 4. If we would limit the set of permissible actions to those explicitly modelled, then sub-process X would not be able to terminate.

To resolve the aforementioned issue, we propose the use of a unique *context* for each sub-process. The context specifies which actions can be executed, including those that are not explicitly modelled. To resolve, amongst others, the ambiguity shown in Fig. 3, we require that all contexts of a hybrid model are pair-wise disjoint. Contexts are allowed to contain an infinite number of possible actions (thereby supporting an open world assumption).

The existence of a context fits naturally within the way process models are created. When the same process is modelled in the same way, but executed within a different location/organisation/etc. then the ideas behind the process and process model do not change. Only the location/organisation/etc. and environment do; hence, the actions one expects to happen also change.

Returning to the hybrid model in Fig. 4 this means that the execution sequence $\langle A, C, B, D\rangle$ is not allowed. The actions C and B are only attributed to a single sub-process and as a result, the top Declare model executes $\langle A, B\rangle$ and the bottom model executes $\langle C, D\rangle$, which are both not allowed.

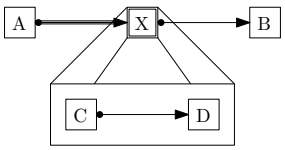

Fig. 5. X should always perform an action.

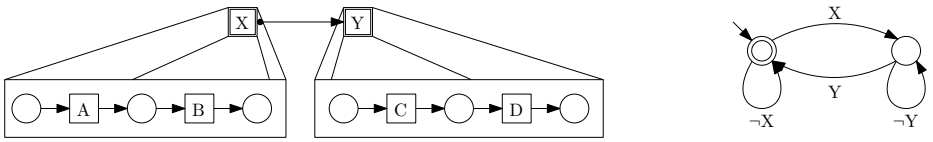

Fig. 6. Within the automaton describing the execution semantics of the Declare model, what does $\neg X$ mean when X is a Petri net?

4.3 Termination of a Declare model requires the execution of an action

In Fig. 5, we have part of a hybrid model where every A is directly followed by X, which is, in turn, eventually followed by B. However, within X, one does not need to execute an action. This means that A can or cannot be directly followed by X dependent on whether an action is executed within X. This allows for the sequence of actions: $\langle A, B\rangle$ whilst if X was not a sub-process but instead an action, then this sequence would not be allowed. We argue that the fact that A needs to be directly succeeded by X implies that *some action* needs to be performed within X directly after A. Note that a similar notion of sub-processes in Declare is proposed in [31]. However, the paper assumes that one can observe an explicit start and completion of the sub process and the exact semantics are left unclear (e.g., it is not made explicit if sub-processes are executed atomically and if they can exhibit empty behaviour).

4.4 Fulfilment of a Declare constraint when a sub-process is negated

Fig. 6 shows a Declare model with two sub-processes modelled as Petri nets. On the right is the automaton encoding the execution semantics of the Declare model. The automaton contains a transition $\neg X$. What does this mean when X is a Petri net, or a sub-process in general? In other words, when is the constraint activated/fulfilled? We see two possible interpretations: 1) $\neg X$ is the complement of the language of X or 2) $\neg X$ is the absence of X. Note that if X was a single action instead of a sub-process, then these two views would largely coincide.

Following the first interpretation of $\neg X$, i.e., it is the complement of the language of X, then the next question becomes whether $\langle B, B, A, B\rangle$ would be a valid execution of the hybrid model? One can argue this sequence is not allowed within X (it only allows for $\langle A, B\rangle$). As a result, it is a valid execution. At the same time, one can argue that $\langle B, B\rangle$ belongs to $\neg X$ and the last two actions ($\langle A, B\rangle$) belong to X. As a result, $\langle B, B, A, B\rangle$ would not be a valid execution.

In the second interpretation (absence of X), we only allow for sequences of actions in the Declare model which are allowed by the sub-processes. Taking our earlier sequence ($\langle B, B, A, B\rangle$), this would not be allowed since X does not allow for this sequence.

We argue that a modeller wants to specify that one can execute $\langle A, B\rangle$, and $\langle C, D\rangle$, and if one would execute $\langle A, B\rangle$, then it needs to be eventually followed by $\langle C, D\rangle$. As a result, we interpret the negation of a sub-process as its absence.

Having the solutions to the challenges for combing different formalisms in a hybrid process model, we can now present the semantics of hybrid process models.

5 Hybrid Models and their Semantics

In this section, we formally define hybrid models and their semantics. We first formally introduce hybrid models and the consistency thereof. Thereafter, for ease of formalisation, we first define the semantics of hybrid models without concurrency. Finally, we generalise the semantics of hybrid models to support models with concurrency.

The Dreyer process does not contain a parallel sub-process and is therefore ill-suited for exemplifying the semantics of models with concurrency. As an alternative, we employ a basic order-fulfilment process for examples within this section, which is shown in Fig. 7.

5.1 Hybrid Models

We distinguish between the universe of actions $\mathbb{A}$ and the universe of labels $\mathbb{L}$. Actions represent the executable elements of a model, e.g., activities, events and transitions. These do not necessarily need to be visual elements: a Declare model may allow for unconstrained activities that are not drawn explicitly. Labels represent the names of these actions and thereby the observed behaviour of the model. This distinction is important because many notations, such as BPMN, labelled Petri nets, and DCR graphs, allow the same label to occur on different elements of the model. Therefore, independence is most generally defined on the actions of the model instead of the labels. Finally, we define $\mathbb{H}$ as the universe of all hybrid models.

Example 1 (Actions and Labels). Consider the hybrid model of Fig 7. The top-level labelled Petri net, contains the following actions: $\{a, b, c, d, e, h, X\}$ and the labels: {*receive order*, *produce goods*, *quality check*, *ship goods*, *receive payment*, *questionnaire*}. Both action c and e map to the label *quality check*, but e is independent from X, while c is not.

A hybrid model consists of a model in some notation, with actions in the model mapping to either an observable label or another hybrid model (which can be defined in another notation) and a context stating which actions and labels are defined by the model, formally:

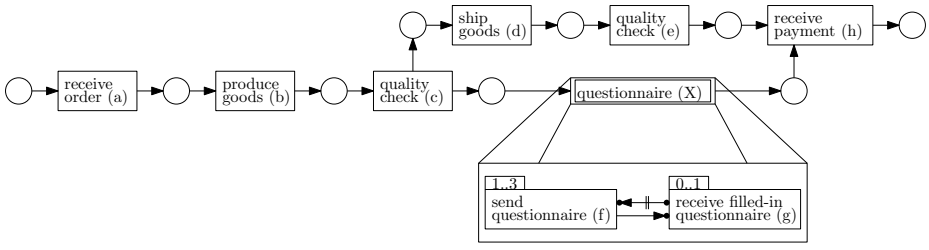

Fig. 7. Hybrid model of a basic order-fulfilment process.

Definition 1 (Hybrid Model). *A* Hybrid Model *is a tuple* $H = (M, C, l, s)$ *comprising a model* M*, a context* $C = (A, L)$*, where* $A \subseteq \mathbb{A}$ *is the set of allowed actions in* M *and* $L \subseteq \mathbb{L}$ *is the set of* observable *labels in* M*, a partial labelling function* $l : A \nrightarrow L$ *mapping actions to labels and a partial sub-process function* $s : A \nrightarrow \mathbb{H}$ *mapping actions to sub-processes.*

As discussed in the previous section, we make a number of assumptions on hybrid models. According to those, we define a consistent hybrid model as:

Definition 2 (Consistent Hybrid Model). *A Hybrid Model* $H = (M, (A, L), l, s)$ *is* consistent *if and only if:*

1. $\mathrm{dom}(l) \cap \mathrm{dom}(s) = \varnothing$ *and* $\mathrm{dom}(l) \cup \mathrm{dom}(s) = A$
2. $\forall\, X, Y \in A_H^+ : X \neq_{id} Y \Rightarrow X \cap Y = \varnothing$ *where* $A_H^+ = \{A\} \bigcup_{(H' \in \mathrm{img}(s))} (A_{H'}^+)$
3. $\forall\, X, Y \in L_H^+ : X \neq_{id} Y \Rightarrow X \cap Y = \varnothing$ *where* $L_H^+ = \{L\} \bigcup_{(H' \in \mathrm{img}(s))} (L_{H'}^+)$
4. $\forall\, H' \in \mathrm{img}(s)$ *: H' is a consistent hybrid model.*

Item 1 ensures that an action is always assigned either a label or a sub-model, but never both. Item 2 ensures that all sets of actions of a hybrid model are pairwise disjoint. Item 3 similarly ensures that all sets of labels of a hybrid model are pairwise disjoint. Items 2 and 3 follow from our requirements on the context. Finally item 4 ensures that any sub-model is also consistent.

5.2 Semantics of Hybrid Models

When defining the semantics of hybrid models, we distinguish between the *action language* $\mathcal{L}_A$, which represents the possible orderings of the actions of the model and the *observable language* $\mathcal{L}_L$ which represents the observable behaviour of the model. The action language allows for a more refined definition of independence.

The action language of a hybrid model will depend on the chosen notations and a formal definition of their semantics. For this paper, we shortly show how one can deduce an action language from Petri nets and Declare. The representation in which both Petri nets as well as Declare models can be expressed are transition systems. Therefore, we provide a mapping from Petri nets and Declare to transition systems.

Action language of Petri nets For Petri nets, transitions are our actions. Given an initial and final marking, we can construct the transition system belonging to the Petri net. Using the transition systems, it is trivial to deduce the action language for the Petri net, i.e., all sequences of actions from the initial state of the transition system to a final state of the transition system. We consider the action language of a Petri net *without* actions originating from silent transitions. Note that it might be that the Petri net contains a deadlock (or cannot terminate in general). This is not an issue for the execution semantics, i.e., the transition system will have a deadlock, but merely a modelling issue.

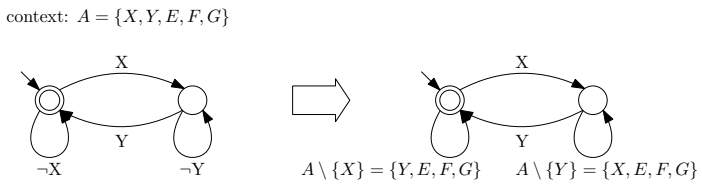

Fig. 8. Transformation from Fig. 6 given a context consisting of the actions: X, Y, E, F, G to a transition system from which we can deduce the action language.

Action language of Declare For a Declare model, we have to pay special attention to the negation, e.g., $\neg X$ in Fig. 6. Similar to a Petri net, we transform the Declare model to a transition system. Using our context, we substitute every negation $\neg X$ by $A \setminus \{X\}$, i.e., we take the complement of $\neg X$ using the allowed actions from the context. After having substituted all negations by their complements, we can, analogous to the Petri net case, deduce the action language. Note that, except for actions mapping to sub-processes, each action in Declare always maps to a unique label. If we take our Declare model from Fig. 6 and assume that the context specifies the following actions: $\{X, Y, E, F, G\}$, then we substitute $\neg X$ and $\neg Y$ as depicted in Fig. 8.

Example 2 (Action Languages for Petri Nets and Declare). The action language of the top-level Petri net in Fig. 7, from here on referred to as $\mathcal{L}_{A(\text{Fig. 7}_{\text{Top}})}$, is $\{abcXdeh, abcdXeh, abcdeXh\}$. The action language of the Declare sub-process in Fig. 7, from here on referred to as $\mathcal{L}_{A(\text{Fig. 7}_{\text{X}})}$, is $\{f, ff, fff, fg, ffg, fffg\}$.

Semantics of Hybrid Models without Concurrency We first consider a special case of hybrid models without parallel constructs. This means that the execution of sub-processes is essentially atomic and will never happen concurrently with other parts of the process. This eases the first definition of the semantics, as concurrency can be disregarded and the action language of a hybrid model becomes the action language of its underlying model where each sub-process is recursively substituted by the action language of that sub-processes. Note that, as discussed in Section 4.3, we always require sub-processes to do at least one action and therefore remove the empty word ϵ from their language.

Definition 3 (Action Language of a Hybrid Model w/o Concurrency). *Let $H = (M, (A, L), l, s)$ be a consistent Hybrid Model and let $\mathcal{L}_A(M)$ be the action language of M, then the action language of H is defined as: $\mathcal{L}_A(H) = \bigcup_{w \in \mathcal{L}_A(M)} \text{Subst}(w)$, where*

1. $\text{Subst}(ab) = \text{Subst}(a)\text{Subst}(b)$
2. $\text{Subst}(a) = \mathcal{L}_A(s(a)) \setminus \{\epsilon\}$ *if* $a \in \text{dom}(s)$
3. $\text{Subst}(a) = a$ *if* $a \in \text{dom}(l)$

Example 3 (Action Language of a Hybrid Model w/o Concurrency). Based on Def. 3 and the action languages defined in Example 2, we can deduce that the action language of the hybrid model in Fig. 7 is: $\mathcal{L}_{A(\text{Fig. 7})} = \{abcf\,deh, abcff\,deh, abcfff\,deh, abcfg\,deh, abcffg\,deh, abcfffg\,deh, abcdf\,eh, abcdff\,eh, abcdfff\,eh, abcdfg\,eh, abcdffg\,eh, abcdfffg\,eh, abcdef\,h, abcdeff\,h, abcdefff\,h, abcdefg\,h, abcdeffg\,h, abcdefffg\,h\}$. As one can see the sub-process X is treated as atomic here and its actions do not interleave individually with the actions d and e.

The observable language for an action language is defined by substitution of actions by their respective labels. We employ a recursive labelling function which combines the labelling functions of all underlying models.

Definition 4 (Observable Language of a Hybrid Model). *Let $H = (M, (A, L), l, s)$ be a consistent Hybrid Model, then the observable language $\mathcal{L}_L$ of H is defined as: $\mathcal{L}_L(H) = \bigcup_{w \in \mathcal{L}_A(H)} \mathrm{LblSubst}(w)$, where*

1. $\mathrm{LblSubst}(ab) = \mathrm{LblSubst}(a)\mathrm{LblSubst}(b)$
2. $\mathrm{LblSubst}(a) = l_H^+(a)$ *where* $l_H^+ = l \bigcup_{H' \in \mathrm{img}(s)} (l_{H'}^+)$

The observable language of Fig. 7 can be obtained by mapping each action to its label.

5.3 Semantics of Hybrid Models with Concurrency

Within Petri nets, having two sub-processes concurrent means that both can execute simultaneously. To extend our semantics to handle concurrency, we first recall the notion of trace languages.

Definition 5 (Trace Languages). *Given an alphabet Σ, the trace equivalence class $[w]_I \subseteq \Sigma^*$ for a word $w \in \Sigma^*$ and an independence relation $I \subseteq \Sigma \times \Sigma$, for which $(a, b) \in I \Leftrightarrow (b, a) \in I$ is a set of words such that:*

1. $w \in [w]_I$
2. $(a, b) \in I \wedge xaby \in [w]_I \Rightarrow xbay \in [w]_I$

We say that a language $\mathcal{L}$ is a trace language consistent *with independence relation I if: $\mathcal{L} = \bigcup_{w \in \mathcal{L}} [w]_I$. For convenience, we write I_M for the independence relation of a model M and say that for a (possibly non-consistent) language $\mathcal{L}$ and an independence relation I, $[\mathcal{L}]_I$ is the minimal language consistent with I such that $[\mathcal{L}]_I \supseteq \mathcal{L}$.*

We now define how to construct the concurrent action language for a hybrid model as a trace language. We reuse the substitution function Subst from Def. 3, but we add an independence relation based on the independence relation of the base models. For any action independent from a sub-process, we add the cross-product of that action with the actions of the sub-process to the independence relation of the parent process.

Definition 6 (Concurrent Action Language of a Hybrid Model). *The concurrent action language $\mathcal{L}_{CA}(H)$ for an independence relation I_H of a consistent Hybrid Model $H = (M, (A, L), l, s)$ and an independence relation I_M for the model M is defined as: $\mathcal{L}_{CA}(H) = [\bigcup_{w \in \mathcal{L}_A(M)} \mathrm{Subst}(w)]_{I_H}$, where* Subst *is defined as in Def. 3 and*

1. $(a, b) \in I_M \Rightarrow (a, b) \in I_H$
2. *For* $s_H^+ = s \bigcup_{H' \in \mathrm{img}(s)} (s_{H'}^+)$*:*
   (a) $(a, b) \in I_M \wedge s_H^+(a) = (M', (A', L'), l', s') \Rightarrow A' \times \{b\} \in I_H$
   (b) $(a, b) \in I_M \wedge s_H^+(b) = (M', (A', L'), l', s') \Rightarrow \{a\} \times A' \in I_H$

(c) $(a, b) \in I_M \wedge s_H^+(a) = (M', (A', L'), l', s') \wedge s_H^+(b) = (M'', (A'', L''), l'', s'') \Rightarrow A' \times A'' \in I_H$

3. $H' \in \text{img}(s) \wedge (a, b) \in I_{H'} \Rightarrow (a, b) \in I_H$

Example 4 (Concurrent Action Language of a Hybrid Model). The independence relation of the top-level Petri net in Fig. 7 is: $I_{\text{Fig. 7}_{\text{Top}}} = \{(d, X), (X, d), (e, X), (X, e)\}$, the independence relation of the Declare model is empty. Following Def. 6, the independence relation of the hybrid model is: $I_{\text{Fig. 7}} = \{(d, f), (f, d), (d, g), (g, d), (e, f), (f, e), (e, g), (g, e)\}$. This means that the concurrent action language of the model is: $\mathcal{L}_{CA}(\text{Fig. 7}) = [\mathcal{L}_A(\text{Fig. 7})]_{I_{\text{Fig. 7}}}$ for the independence relation $I_{\text{Fig. 7}}$. Here $\mathcal{L}_{CA}(\text{Fig. 7})$ is the language of Example 4, but made consistent with the independence relation $I_{\text{Fig. 7}}$: it includes all possible interleavings of the actions d and e with the actions f and g. Note that $\mathcal{L}_{CA}(\text{Fig. 7})$ by itself provides an interleaving semantics for the hybrid model. When combined with the independence relation $I_{\text{Fig. 7}}$, which makes clear which actions can happen independently, it provides a true concurrent semantics.

Applying Def. 4 to the concurrent action language of a hybrid model will yield the observable language respecting all valid interleavings.

6 Conclusion

In this paper, we provided a generic semantics for a hybrid process modelling notation, supported by a use case based on a real-life process in use at a Danish foundation. We discussed various issues that stem from mixing the different paradigms and argued for ways to resolve them in a satisfactory manner.

Our proposal paves the way for the integration of notations that individually are suitable to either model structured or unstructured process parts. While it is to some extent left open how to define the action languages for other formalisms, e.g., BPMN, DCR graphs, as well as the canonical format of `APROMORE`, we believe that the provided semantics will be helpful for this purpose.

Future Work The current paper serves as a stepping stone to a broad range of future work. We intend to investigate each of these opportunities, in part supported by a 3-year project grant from the Danish Council for Independent Research.

Our contribution allows for an empirical evaluation of the expected advantages of hybrid process modelling notations. Building on our proposal, it is possible to design experiments that provide declarative, imperative, and hybrid models of the same process, which can be compared with respect to the relative ease of understanding these. There is a rich stream of experimental research that can inspire such experiments [19,20]. In a similar vein, it is interesting to determine how easy it is for modellers to apply hybrid notations. Finally, through experiments and discussions with practitioners, we can determine for which domains such models are particularly useful.

Providing a formal semantics for hybrid process notations will also support the development of prototype modelling and simulation tools for the paradigm.

These tools, in turn, can be used to run experiments and make the approach accessible to a wider audience.

We see opportunities to alleviate some of the assumptions we have posed on the hybrid notation we proposed. For instance, it is possible to weaken the requirement that no actions at all are shared between contexts. In some cases, for instance, when two contexts can be shown never to run in parallel, there will be no ambiguity. It is also possible to allow for more detailed interactions between sub-processes and their parent by synchronizing on shared actions. While such a framework will be more permissive, it will also allow for more complex models, increasing the need for analysis and verification techniques tailored to hybrid models.

To us, hybrid process models seem well applicable beyond the as-is modelling phase of the BPM lifecycle [8], as has been argued in [1]. Clearly, for the run-time phase, this paper provides an important ingredient, i.e., an execution semantics. It is this semantics that could be taken as a basis for a BPM system to support the execution of a business process that is modelled in a hybrid fashion. For the process discovery phase, we have already presented a mining technique in an earlier paper [12]. It is our plan to extend that work to align it with the newly proposed execution semantics, as well as to study the application of hybrid process models in other phases of the BPM lifecyle.

Finally, for the hybrid paradigm to become useful for practitioners, suitable modelling methodologies will need to be developed. A primary open question is how to identify the right sub-processes and determine the best notation for each. When historical data is available, automated mining approaches can assist in this effort, but for to-be processes, where this is not the case, modelling guidelines are required.

References

1. M.C. Barukh and B. Benatallah. Processbase: A hybrid process management platform. In *ICSOC 2014*, pages 16–31, 2014.
2. Steinar Carlsen. Action port model: A mixed paradigm conceptual workflow modeling language. In *IFCIS*, pages 300–309, 1998.
3. G. De Giacomo, M. Dumas, F. M. Maggi, and M. Montali. Declarative process modeling in BPMN. In *CAiSE 2015*, pages 84–100, 2015.
4. J. De Smedt, J. De Weerdt, and J. Vanthienen. Multi-paradigm process mining: Retrieving better models by combining rules and sequences. In *OTM 2014*, pages 446–453, 2014.
5. S. Debois, T.T. Hildebrandt, M. Marquard, and T. Slaats. A case for declarative process modelling: Agile development of a grant application system. In *EDOCW/AdaptiveCM 2014*, pages 126 – 133, 2014.
6. S. Debois, T.T. Hildebrandt, and T. Slaats. Hierarchical declarative modelling with refinement and sub-processes. In *BPM 2014*, pages 18–33, 2014.
7. S. Debois and T. Slaats. The analysis of a real life declarative process. In *CIDM 2015*, pages 1374–1382, 2015.
8. M. Dumas, M. La Rosa, J. Mendling, and H.A. Reijers. *Fundamentals of business process management*. Springer, 2013.

9. C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber. Understanding declare models: strategies, pitfalls, empirical results. *Software & Systems Modeling*, pages 1–28, 2014.
10. T.T. Hildebrandt and R.R. Mukkamala. Declarative event-based workflow as distributed dynamic condition response graphs. In *PLACES*, pages 59–73, 2010.
11. T.T. Hildebrandt, R.R. Mukkamala, and T. Slaats. Nested dynamic condition response graphs. In *FSEN 2011*, pages 343–350, 2011.
12. F.M. Maggi, T. Slaats, and H.A. Reijers. The automated discovery of hybrid processes. In *BPM 2014*, pages 392–399, 2014.
13. I. Markovic and M. Kowalkiewicz. Linking business goals to process models in semantic business process modeling. In *EDOC*, pages 332–338, 2008.
14. M. Marquard, M. Shahzad, and T. Slaats. Web-based modelling and collaborative simulation of declarative processes. In *BPM 2015*, pages 209–225, 2015.
15. M. Montali. *Specification and Verification of Declarative Open Interaction Models - A Logic-Based Approach*, volume 56. 2010.
16. M. Montali, M. Pesic, W.M.P. van der Aalst, F. Chesani, P. Mello, and S. Storari. Declarative Specification and Verification of Service Choreographies. *TWEB*, 4(1), 2010.
17. D. L. Moody. The physics of notations: toward a scientific basis for constructing visual notations in software engineering. *IEEE TSE*, 35(6):756–779, 2009.
18. M. Pesic, H. Schonenberg, and W.M.P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC 2007*, pages 287–300, 2007.
19. P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H.A. Reijers. Imperative versus declarative process modeling languages: An empirical investigation. In *BPM Workshops*, pages 383–394, 2011.
20. H.A. Reijers, J. Mendling, and R.M. Dijkman. Human and automatic modularizations of process models to enhance their comprehension. *Information Systems*, 36(5):881–897, 2011.
21. H.A. Reijers, T. Slaats, and C. Stahl. Declarative modeling-an academic dream or the future for bpm? In *BPM 2013*, pages 307–322, 2013.
22. S.W. Sadiq, M.E. Orlowska, and W. Sadiq. Specification and validation of process constraints for flexible workflows. *Information Systems*, 30(5):349–378, 2005.
23. T. Slaats, R.R. Mukkamala, T.T. Hildebrandt, and M. Marquard. Exformatics declarative case management workflows as DCR graphs. In *BPM 2013*, 2013.
24. J. De Smedt, J. De Weerdt, J. Vanthienen, and G. Poels. Mixed-paradigm process modeling with intertwined state spaces. *Business & IS Eng.*, 58(1):19–29, 2016.
25. R. Vaculín, R. Hull, T. Heath, C. Cochran, A. Nigam, and P. Sukaviriya. Declarative business artifact centric modeling of decision and knowledge intensive business processes. In *EDOC*, pages 151–160. IEEE, 2011.
26. W.M.P. van der Aalst, M. Adams, A.H.M. ter Hofstede, M. Pesic, and H. Schonenberg. Flexibility as a service. In *Database Systems for Advanced Appl.*, 2009.
27. W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Comp. Sc. - R&D*, 23:99–113, 2009.
28. W.M.P. van der Aalst and A.H.M. ter Hofstede. Yawl: yet another workflow language. *Information Systems*, 30(4):245 – 275, 2005.
29. M. Westergaard and T. Slaats. Cpn tools 4: A process modeling tool combining declarative and imperative paradigms. In *BPM (Demos)*, 2013.
30. M. Westergaard and T. Slaats. Mixing paradigms for more comprehensible models. In *Business Process Management*, pages 283–290. 2013.
31. S. Zugal, P. Soffer, J. Pinggera, and B. Weber. Expressiveness and understandability considerations of hierarchy in declarative business process models. In *BPMDS 2012*, pages 167–181, 2012.
32. Michael Zur Muehlen and Marta Indulska. Modeling languages for business processes and business rules: A representational analysis. *Information systems*, 35(4):379–390, 2010.