Towards a Multi-Model Paradigm for Business Process Management

Anti Alman[1][0000−0002−5647−6249], Fabrizio Maria Maggi[2][0000−0002−9089−6896], Stefanie Rinderle-Ma[3][0000−0001−5656−6108], Andrey Rivkin[4][0000−0001−8425−2309], and Karolin Winter[5][0009−0003−8030−2964]

[1] University of Tartu, Tartu, Estonia
anti.alman@ut.ee
[2] Free University of Bozen-Bolzano, Bolzano, Italy
maggi@inf.unibz.it
[3] Technical University of Munich, Munich, Germany
stefanie.rinderle-ma@tum.de
[4] Technical University of Denmark, Lyngby, Denmark
ariv@dtu.dk
[5] Eindhoven University of Technology, Eindhoven, The Netherlands
k.m.winter@tue.nl

Abstract. The design and development of information systems (IS) often requires not only software development expertise, but also a deep understanding of the multitude of business processes supported by the given IS. Such understanding is usually elicited via business process modeling and numerous, often interrelated, process models can be created even for a single IS. However, commonly used process modeling languages focus on single processes in isolation, while providing, at best, only limited support for modeling process interactions. This does enforce a clear scope on each process model, but also leads to a non-holistic view of the IS behavior. In this exploratory paper, we take the position that, instead of forcing existing "single-process-focused" models to be changed, approaches should be provided for modeling their interactions in a fine-grained and unambiguous manner. To meet this goal, we propose developing a *Multi-Model paradigm* for Business Process Management, where the same, already existing, declarative and procedural modeling languages would be used to represent both the individual processes as well as their interactions.

Keywords: Multi-Model Paradigm · Business Process Management · Business Process Modeling · Hybrid Process Model · Model Interplay.

1 Introduction

The connection between information systems (IS) and business process management (BPM) can range from IS providing simple data storage to fully-fledged Process-Aware Information Systems (PAIS) [1] often directly managing multiple

interconnected business processes. From the IS design and development perspective, this necessitates a deep understanding of such processes, which is commonly achieved by creating multiple, also often interrelated, process models [17].

A wide variety of languages can be used for that purpose, ranging from procedural (e.g., Petri nets [38]) to declarative (e.g., DECLARE [31]), and various combinations in between (cf. [7]). Furthermore, an object-centric view for representing processes has recently emerged in the field of process mining [2]. But despite that variety, existing languages tend to provide significantly more expressive power for modeling single processes, than for modeling their interactions.

This "single-process-focus" helps to considerably simplify modeling by giving a clear scope to each process model. However, it also contributes to the emergence of "process silos", a phenomenon recently recognized as one of the most important BPM problems yet to be solved [10]. In a nutshell, process silos often lead to multiple, potentially heterogeneous and certainly misaligned processes, that nevertheless need to be executed in combination.

In fact, interactions between business processes (and by extension, between the corresponding models) are highly relevant, as demonstrated by ongoing research in areas such as holistic BPM [9], collaborative processes [19], and process-spanning constraints [41]. Similar interactions can also be observed in hierarchical process models where some activities may refer to other process models (subprocesses) [37], customizable processes where a base model is combined with models capturing its variations [26], and even guided process discovery where (possibly interconnected) model fragments are used as an additional input [29]. While very different, all the above examples can be seen as manifestations of a single underlying research question (**RQ**): *"How to represent and handle the interactions of multiple (separately defined) business process models?"*.

Our earlier work [3, 5] also began by tackling that same RQ in a narrow context (i.e., online process monitoring), but was later extended into a data-aware modeling language/approach [4], where individual processes (procedural or declarative) can still be first modeled in isolation, and then simply combined through the use of exactly the same modeling constructs.

Given the above examples, we propose that scenarios, where one or more processes are best represented as sets of smaller interconnected models, are sufficiently prevalent to warrant a dedicated line of research, which we refer to as a *Multi-Model paradigm* for BPM. Furthermore, given our experience with [3–5], we propose that already existing languages can be extended to meet modeling requirements of these scenarios in a formally well-defined manner. In this paper, we explore these ideas further by a language-agnostic analysis of four example scenarios, based on which we derive a corresponding research agenda for [4].

More specifically, Section 2 analyzes a representative example of each scenario, leading to a set of modeling requirements presented in Section 3. Then, Section 4 discusses the requirements (not) met by [4] and the corresponding future extensions. This is followed by a discussion on the limitations of this paper in Section 5, and an overview of related approaches in Section 6. Finally, Section 7 concludes the paper by outlining directions for future works.

2 Multi-Model Scenarios

Based on numerous discussions among the authors of this paper, we identified four types of business processes in which multi-model scenarios are, in our opinion, likely to emerge. These types were validated by searching for concrete examples from the literature. However, as discussed in Section 5, we did not aim for completeness, instead focusing on broadness and variety. We begin with a relatively simple hybrid model, followed by progressively more complex examples involving multiple interconnected process models.

2.1 Hybrid Processes: Management of Funding Applications

By hybrid processes, we refer to business processes which contain both structured and unstructured parts, and therefore, require both the strictness of procedural, as well as, the flexibility of declarative process modeling languages. One of the most common approaches to hybrid processes is using a hierarchical structure of sub-processes, with each being either declarative or procedural (e.g., [37]).

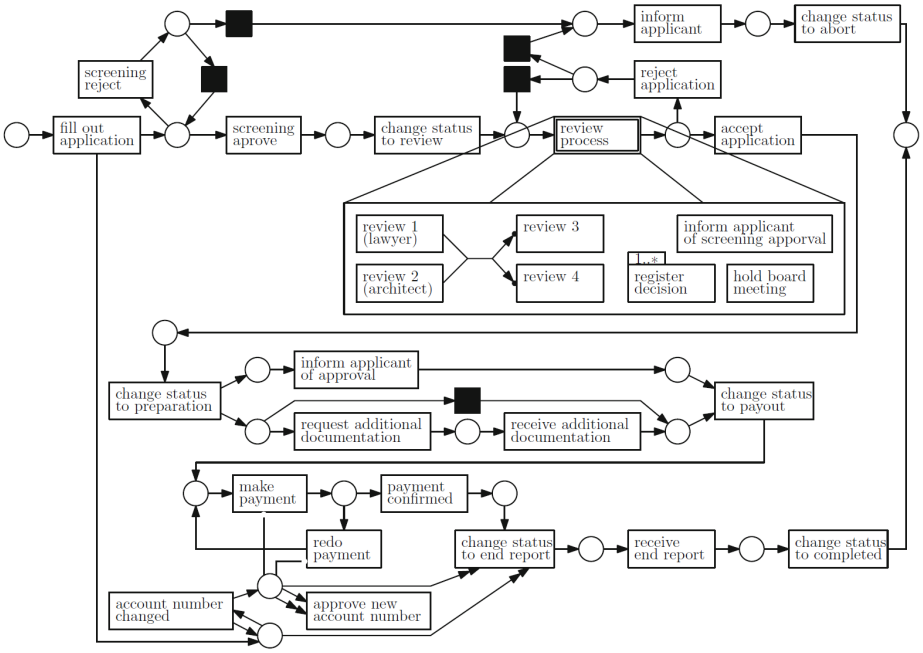

Fig. 1. Hybrid application model for the Dreyers foundation (taken from [37]).

As a concrete example, we use the application process of the Dreyer Foundation from [37]. The same source provides two process models: a fully declarative model using nested DCR Graphs [13]; and a hybrid model using a target-branched DECLARE [14] sub-process within a Petri net. The fully declarative

model is used in practice to drive the electronic case management system of the Dreyer Foundation. However, given the complexity of that model, a hybrid solution was investigated, resulting in the model shown in Fig. 1.

Without describing the full process here (cf. [37]), we highlight the following:

- **Strict control flow** – Based on Fig. 1 it is clear that most of the application process is procedural, which is also the main reason why [37] started exploring alternatives to using a fully declarative model.
- **Flexible control flow** – In contrast, application reviews are highly flexible and therefore modeled declaratively. The corresponding model (`review process`) does not define concrete start and end activities, but does require at least one execution of `register decision`. Authors of [37] also note the flexibility related to account numbers and payments, but state that this part is modeled procedurally due to it having two entry points.
- **Sub-processes** – The declarative model is integrated into the procedural one as a sub-process. More specifically, firing the Petri net transition `review process` means executing the corresponding declarative model. The process itself does not contain a concrete activity `review process`, meaning that `review process` in the hybrid model is effectively a model reference.
- **Data conditions** – While not present in the hybrid model, the fully declarative model contains data conditions defining which reviewer should review which type of application. These conditions are in the form of variable-to-constant comparisons, e.g, $UddelingPulje = 2$.
- **Time perspective** – While not present in the hybrid model, the fully declarative model also specifies a time interval of three days between two reviews.

2.2 Orchestration Processes: Assessment of Loan Applications

By orchestration processes, we refer to scenarios where different roles, systems, artifacts, etc., have their own models to be executed in parallel with others, while following some cross-model dependencies. The most common of these dependencies stem from a shared subject (e.g., a patient) between the models, which may, for example, lead to shared data values (e.g., blood pressure of the patient) and synchronization points (e.g., admission of the patient). This is analogous to orchestration processes in BPMN [30], but with a distinct model for each lane.

As a concrete example, we rely on the loan application event log [16] of BPIC 2017 and our corresponding analysis in [4]. This event log explicitly distinguishes application state changes, offer state changes and workflow events, thus providing a natural basis for three interconnected process models as shown in Fig. 2. Each model is represented procedurally using Data Petri nets, while the interactions of models are defined by milestones, shared variables, and declarative constraints.

Without describing the full process here (cf. [4]), we highlight the following:

- **Strict control flow** – Based on Fig. 2, it is clear that the overall process is largely procedural. However, there is some flexibility in how the activities of the models can interleave, especially after the activity `A_Complete`.

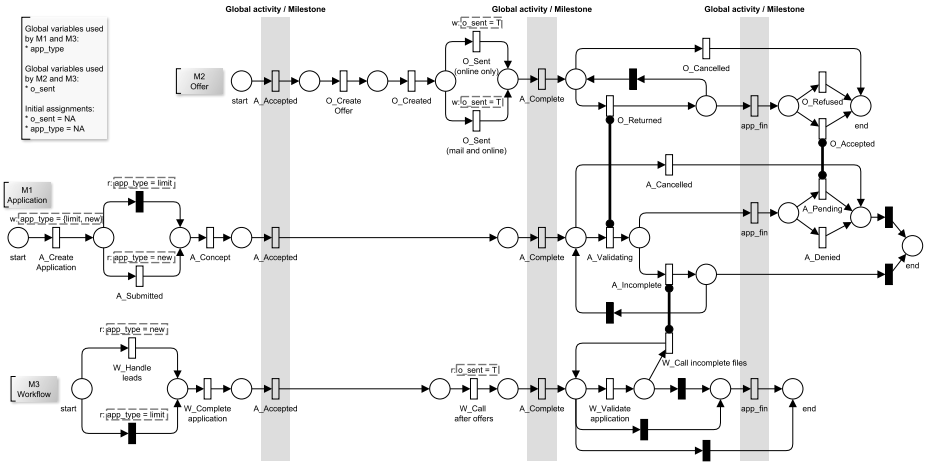

Fig. 2. Orchestration processes for loan application assessment (taken from [4]).

- **Cross-model variables** – Activity `A_Create Application` in M1 stores the loan application type (variable *app_type*). This stored value is then used in M1 to decide if `A_Submitted` will be executed. Furthermore, the same value is also used in M2 to decide if `W_Handle leads` will be executed.
- **System variables** – Model M2 stores a value for the variable (*o_sent*) which is required to execute `W_Call after offers` in M3. However, based on [4], this variable is not part of the original process, but, instead, used specifically to enforce an additional synchronization between M2 and M3.
- **Data conditions** – All three models contain variable-to-constant comparisons, with $r:$ and $w:$ denoting read and write operations respectively, and $\{\ldots\}$ used as a shorthand for possible matching constants.
- **Cross-model activities** – Activities `A_Accepted`, `A_Complete`, and `app_fin` constitute milestones of this process. These activities can only be executed when they are allowed by all models, and a single execution progresses all models concurrently, making them cross-model activities (milestones).
- **Declarative interleavings** – Interleavings of these models are further constrained by three pairs of activities (e.g., `O_Accepted` and `A_Pending`), where executing one requires executing the other at some point in the same trace.
- **Execution cardinalities** – As highlighted in [4], Fig. 2 falls short if multiple loan offers (executions of M2) are made for a single loan application (execution of M1). For example, it would fail to capture that at most one offer can be accepted (`O_Accepted`) per successful application (`A_Pending`).

2.3 Collaborative Processes: Manufacturing of Car Parts

By collaborative processes, we refer to scenarios in which multiple organizations, each having their own business processes, collaborate in a way that creates some dependencies between their processes. Such scenarios are usually tackled by modeling a message flow between process activities, such that each activity having

incoming messages can only be executed after receiving all of them, thus providing a mechanism for modeling synchronizations. A prime example of this approach is the use of pools and message flows in BPMN, along with the related collaboration and choreography diagrams [30].

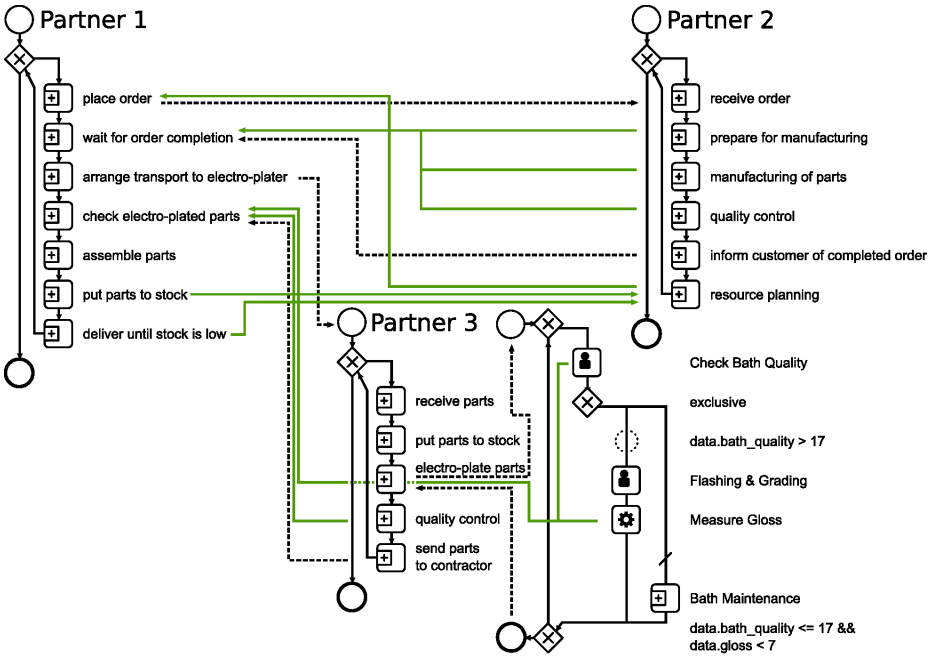

Fig. 3. Collaborative model of processes for manufacturing car parts (taken from [19]).

As a concrete example, we rely on [19], which presents the real-life collaboration of three partners involved in manufacturing car parts, i.e., a car manufacturer, and partners responsible for injection molding and electro plating. As shown in Fig. 3 the process of each partner is modeled using BPMN, while additional black and green lines are used between the models to represent message exchanges and data dependencies respectively.

Without describing the full process here (cf. [19]), we highlight the following:

- **Strict control flow** – The overall process, as shown in Fig. 3, is largely procedural. However, some flexibility in process interleavings is possible.
- **Public representation of private tasks** – Almost all activities in Fig. 3 are modeled as complex tasks to hide internal processes from other partners.
- **Sub-processes** – While all other activities could be treated as atomic, `electro-plate parts` (Partner 3) is explicitly connected to both the start and the end event of the same concrete process model, effectively turning `electro-plate parts` into a sub-process reference.
- **Execution triggers** – The process of Partner 3 is triggered by the execution of `arrange transport to electro-plater` (Partner 1) and needs to be

completed for executing the next activity of Partner 1. A sub-process is not suitable here due to the partners being different and also because the process of Partner 3 does not refer back to the same activity that triggered it.
- **Activity types** – The model for `electro-plate parts` differentiates between user and service activities.
- **Data conditions** – The model for `electro-plate parts` contains variable-to-constant conditions on variables *data.bath_quality* and *data.gloss*.
- **Cross-model variables** – As stated before, each green line represents a data dependency between the processes, i.e., data generated by one process is available for the execution of another process (possibly as shared variables).
- **Message exchange** – The data dependencies represented by the green lines can also be seen as as a message exchange mechanism between the processes.
- **Cross-instance messages** – Each execution of this collaboration produces information for the next execution, as represented by the data dependency from `resource planning` (Partner 2) to `place order` (Partner 1). Here, we consider each iteration of the outer-loops as one execution.

2.4 Instance-Spanning Processes: Operation of a Printing Agency

In addition to model interactions, the examples in Sections 2.2 and 2.3 also contain some instance-spanning behavior, and one can easily argue that the example in Section 2.1 is likely to contain such behavior as well (e.g., the overall funding budget is likely to be limited). In the literature, these types of behaviors/rules are referred to as *instance-spanning constraints* [18], and, in essence, they relate multiple instances of the same process to each other in some way.

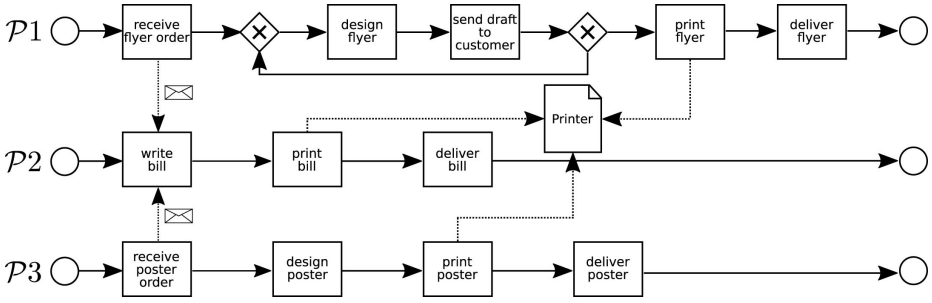

Fig. 4. Instance-spanning processes of a printing agency (taken from [41]).

As a concrete example, we rely on [41], which presents a scenario inspired by an extensive collection of real-life examples of instance-spanning constraints [34]. It consists of three interconnected processes (P1, P2, P3 in Fig. 4), where P1 and P3 describe the design, printing and delivery of fliers and posters respectively, and P2 manages the corresponding bills for both. While not explicitly modeled, [41] also describes six instance- and process-spanning constraints for this scenario, ranging from batching behavior to concurrency constraints.

Without describing the full process here (cf. [41]), we highlight the following:

- **Strict control flow** – Based on Fig. 4 all three processes are modeled procedurally and have largely linear control flows.
- **Execution triggers** – One of the constraints in [41] states that "if a flyer or poster order is received then P2 (billing process) is started", which effectively means that the first activity of either P1 or P3 triggers the execution of P2.
- **Message exchange** – Related to this, P2 receives some billing information from either P1 or P3, which is modeled as an envelope in Fig. 4.
- **Time perspective** – Two constraints in [41] relate to the time perspective. First, finished orders are delivered in the evening. Second, at least 95% printing activities must take 10 minutes or less. The former relates to the time of day, while the latter is basically a key performance indicator (KPI).
- **Batch activities** – Concerning deliveries, [41] also states that all completed deliveries are made simultaneously, which effectively means a single batch activity progressing multiple instances concurrently.
- **Batch conditions** – Printing is also a batch activity in [41], but with an added condition that fliers and posters must be printed separately from bills.
- **Cross-instance cardinalities** – Concerning printing, [41] also states that "printer 1 may only print 10 times per day", which means that a cross-instance cardinality constraint on an activity should hold, with added conditions on data and time.

3 Modeling Requirements

The highlights of the example scenarios in Section 2 directly lead to the modeling requirements for developing a Multi-Model paradigm for BPM. A corresponding overview is provided in Table 1.

Overall, we identified 16 requirements on modeling expressiveness, which we further categorize as requirements on i) individual models, ii) model interactions, and iii) instance interactions. For the individual models, the requirements cover control flow, data flow, and time perspectives. Across the use cases, a mix of strict and flexible control flow as well as support for at least constant-to-variable data conditions is needed. Support of time and system variables is also important, but depends on the use case.

Developing a multi-model paradigm demands for interaction of models. The corresponding requirements (Table 1) cover a range of interactions from nesting of models via sub-processes to interactions based on message exchanges, cross-model variables, and cross model activities. Furthermore, execution of models or activities across models can be further refined through declarative rules for interleavings, execution cardinalities, and execution triggers.

As seen in Table 1, the particular model interactions differ between scenarios. In the hybrid example, the interaction between declarative and procedural models is based on their nesting as sub-processes. In the loan example, models are orchestrated based on shared activities and variables, together with declarative definitions of additional interleaving rules. For the collaboration, a choreography interaction style (based on message exchanges) is used, where the receiving of a

Table 1. Categorized requirements on expressiveness and their applicability to example scenarios. Highlights from Section 2 are marked with "+", while "+/-" marks likely applicability. The last column is used to provide an overview of Section 4.1.

Requirements		Hybrid Funding	Orchest. Loan	Collab. Car	Inst-Span. Printing	Supported by [4]
Individual models	Req1: Strict control flow	+	+	+	+	+
	Req2: Flexible control flow	+	+/-		+/-	+
	Req3: Data conditions	+	+	+	+/-	+
	Req4: Time perspective	+			+	
	Req5: System variables		+			+
Model interactions	Req6: Sub-processes	+		+		
	Req7: Message exchange	+/-		+	+	
	Req8: Cross-model variables		+	+/-		+
	Req9: Cross-model activities		+			+
	Req10: Execution cardinalities		+			
	Req11: Execution triggers		+/-	+	+	
	Req12: Declarative interleavings		+	+/-	+/-	+
Instance interactions	Req13: Batch activities		+/-		+	
	Req14: Batch conditions				+	
	Req15: Cross-instance messages			+		
	Req16: Cross-instance cardinalities		+/-		+	

message can be also seen as an (activity) execution trigger. Meanwhile, the printing example is structured similarly to an orchestration, but the models interact through message exchanges and additional rules across process instances.

Instance level interactions require at least the ability to model batch activities, which progress multiple instances concurrently, along with conditions on which instances belong to the same batch. Furthermore, messages (and possibly data values in general) may cross the instance boundary and the number of activity (and possibly process) executions may be limited across instances.

Finally, we note that two of the highlights from Section 2 are not present in Table 1. First, public representation of private tasks was not added since it can be solved through traditional usage of sub-processes, which is already covered by the requirements. Second, activity types are not added as they are purely informative in the corresponding example. Additionally, KPIs and resource perspective are left out of the scope of this paper (cf. Section 5).

4 Exploring a Potential Baseline Language

Given that, to the best of our knowledge, no existing modeling languages nor frameworks meet all the requirements outlined in Section 3, we could either design a completely new language from ground up or build on already existing works. In this paper, we opt for the latter, which, regardless of the specific works chosen, necessitates an analysis of these works w.r.t. the modeling requirements, and laying out a plan for addressing any potential shortcomings identified during that analysis. In the following, we go through these steps (Section 4.1 and Section 4.2 respectively) on the basis of [4], while noting that a similar exploration could also be undertaken on the basis of other existing works (cf. Section 5).

4.1 Current Capabilities

As stated in Section 1, we choose the formal modeling language of [4] as our baseline. The main reason is that it already provides means to represent multi-model scenarios by using Data Petri nets (DPNs) [12, 20, 28], accounting for procedural components, and DECLARE with "local filters" (LF-DECLARE) [4], accounting for declarative components. The latter is also used in [4] to model complex interactions between DPN components and to further refine their behavior. This fully meets **Req1**, **Req2** and **Req12**.

Activities in [4] are instantaneous and come with attributes in the form of attribute-value pairs (similarly to MP-DECLARE [11]). Both LF-DECLARE constraints and DPNs may contain conditions (in the case of DPNs, such conditions are attached to transitions) over these attributes in the form of variable-to-constant comparisons, using the comparison operators $=, <, >$ and boolean connectives $\neg, \wedge, \vee$. This meets **Req3** and **Req5**, but with some caveats (cf. Section 4.2). Meanwhile, **Req4** is not met, as [4] lacks a notion of time.

Each activity in [4] is enriched with a special provenance identifier referring to a concrete model to which the activity belongs. This allows LF-DECLARE constraints to refer to activities of specific models, even if other models have same-labeled activities, thus further reinforcing **Req12**. However, activity provenance is not required, and in such cases an activity is considered global across all models, which meets **Req9**. Furthermore, variables in [4] can similarly be either local or global, with the latter meeting **Req8**. A combination of global/local variables can also be used to mimic message-passing protocols (**Req7**), but this requires significant modeling effort.

Finally, the LF-DECLARE constraints in [4] capture complex DPN interleavings, but yet fall short of meeting **Req6**, **Req10** and **Req11** as only concrete activities (and not entire models) can be constrained. For example, **Req6** would require defining constraints on each potential start and end activity of a sub-process, and even then, the corresponding "high level" activity would effectively remain instantaneous (i.e., it is executed and the sub-process starts afterwards). Furthermore, LF-DECLARE constraints also hinder from meeting **Req13-16** as instance-spanning behavior was not originally considered in [4].

4.2 Language Extensions

While [4] meets the most crucial modeling requirements (e.g., declarative and procedural control flow, interleavings, etc.) out of the box, we also discovered that it is not sufficiently expressive to fully deal with the scenarios discussed in this paper. We have analyzed these results and identified a set of extensions which would transform the modeling language of [4] into one suitable for a Multi-Model paradigm for BPM. In the following, we discuss each of these extensions, highlighting the requirements it either solves directly or contributes to solving.

Declarative process models. Although LF-DECLARE constraints in [4] already meet **Req2**, there is one caveat that should be addressed. LF-DECLARE constraints use simple filters (i.e., activities are matched based on data payload

values in addition to activity labels), while DPNs use more sophisticated read and write conditions that allow to store and manipulate data within DPN local or global variables. DPN conditions can precisely replicate the behavior of LF-Declare filters, but not the other way around, since LF-Declare lacks a distinction between read and write conditions and the corresponding storage capabilities. Therefore, LF-Declare should be enhanced with precisely the same guard language as used in DPNs, thus enabling **Req5** within declarative models.

Model references. Both DPN transitions and LF-Declare constraints can only refer to concrete activities in [4], which is not sufficient for meeting **Req6** and **Req11**. However, this can be overcome by allowing both LF-Declare constraints and DPNs to refer to entire process specifications, which would meet **Req6**, while **Req11** may also require references specifically to the beginning and end of models. Furthermore, this extension can be designed in such a way that it also addresses **Req10** through declarative cardinality constraints referencing entire models, and contributes to covering **Req7**, as message exchanges can, in some cases, also serve as execution triggers (cf. Section 2.3).

Execution goals. In our previous works [3–5], one of the points of contention was a decision on what should drive the process execution in a multi-model setting, while having a strong requirement that all process components must successfully complete their executions. To accommodate the latter, two options have been explored: one may consider the DPN final markings and/or the satisfaction of LF-Declare constraints as execution goals. However, both options fall short (in the context of [4]) when the execution of a model is required only under specific conditions or constraints (e.g., a model is the sub-process of an optional higher level activity or a model may not be executed more than a certain number of times). To address such issue one may introduce process-spanning constraints, allowing for fine-grained specification of component behaviors, and complement such constraints by requiring each model to be configured as either "mandatory" or "optional". Like that, potentially multiple mandatory models can drive the execution (using the goal conditions from above), leaving the execution of other models optional according to specific execution constraints set among them. This would cover **Req10** and **Req11**, while also reinforcing **Req12**.

Richer comparison operations. The first two requirements on instance interactions (**Req13** and **Req14**) can possibly be solved with LF-Declare constraints by introducing correlation conditions. This would allow each constraint to distinguish and reliably handle activities of different executions through the implicit data condition *same case_id*, while omitting that condition would effectively result in a cross-instance constraint, thus allowing also to meet **Req16**. The solution for DPNs likely involves extending their expressive power towards formalisms like the one of colored Petri nets [23].

Arithmetic operations. An alternative (and possibly complementary) approach to meet **Req16** is adding support for arithmetic operations (e.g., addition, multiplication) and functions (e.g., average, sum), which would also reinforce **Req3** and **Req5**. A way of achieving this is to extend the language of conditions currently used in DPNs (similarly to [20]) allowing for such expres-

sions as $i' = i + 1$, which increments a (possibly global) variable i whenever a corresponding transition is fired. By requiring this increment to happen across multiple instances (thus requiring i to be global), one also contributes to **Req15**.

Time perspective. The most natural way of working towards **Req4** in declarative process models is extending LF-DECLARE with the already existing Timed DECLARE language [39]. However, activities in Timed DECLARE are still instantaneous and modeling truly durative activities (over, e.g., dense time intervals) is out of reach of the language and requires further investigation. As for procedural components, one may think on extending DPNs from [4] towards the support of time, similarly to Timed Petri nets [32]. This extension would also contribute to **Req3** and would be relevant for **Req13** in some cases (cf. Section 2.4).

Message exchanges. Technically, message exchanges (**Req7**) can already be represented using the modeling language from [4]. Specifically, sending messages can be mimicked by updating values of corresponding global variables to the content of the message, while receiving and reading messages can be modeled by copying the values of the same global variables into local variables of a process. Furthermore, the same approach would likely apply to **Req15** after the addition of cross-instance variables (cf. Arithmetic operations). However, modeling these exchanges explicitly would be a significant burden for a type of interaction that is very common in many types of processes (cf. Section 2.3). Instead, a simpler modeling construct should be added, which can then be seamlessly translated into the corresponding interactions through global and local variables.

5 Limitations

In the following, we discuss the main limitations of this paper in more detail.

Selection of multi-model scenarios. We cannot claim any notion of completeness in our selection of example scenarios (Section 2) as it is primarily based on discussions between the authors of this paper. A more systematic review (e.g., based on [25]) was considered, however, we opted against it due to terminological issues. In general, interactions of process models are studied within the context of specific research areas, each using their own terminology. For example, [41] uses the term "process-spanning constraints", while [37] uses terms "hierarchical" and "sub-processes". Meanwhile, more general terms such as "model interaction" or "model interplay" are used in neither. This means that, on the one hand, any systematic review must rely on the specific terminology of the relevant research areas, but, on the other hand, these areas are difficult to systematically identify due to the lack of a common terminology. Given these reasons, we believe that the approach taken in this paper is the most feasible, and also meets the main goal of identifying a broad set of diverse multi-model scenarios.

Completeness of modeling requirements. The modeling requirements presented in Section 3 are likely to be non-exhaustive as we may have missed some research areas and other examples of the areas that we included may lead to additional requirements. Furthermore, there are two broader categories of requirements, which we decided to be out of the scope of this paper. First, the

requirements on resources, partly because it was not discussed in any of the selected examples, but also because resources can be addressed, at least to some extent, through data conditions. However, we acknowledge that a more explicit representation of resources and/or roles (akin to BPMN lane constructs) can be beneficial in process models. Second, incorporating existing probabilistic languages, such as [6], could also be considered. This would allow further refinement of instance-spanning behavior within and between the individual models, and could additionally be used to represent the underlying business rules of KPIs (cf. the scenario in Section 2.4). However, incorporating a probabilistic dimension would increase the scope of this paper considerably and, for this reason, should be tackled separately.

Reliance on a specific baseline language. Section 4 is focused on the analysis and extension of one specific language, namely [4], which is also developed by the authors of this paper. This choice is mainly motivated by our in-depth expertise on [4], based on which we believe it is a natural starting point for developing a multi-model paradigm for BPM. However, we acknowledge that this opinion may be biased by our involvement in [4]. For alternatives, one could consider more well-known formalisms, such as Colored Petri nets [24] and Open Petri Nets [8], or some of the approaches mentioned in Section 6. To accommodate that, we have structured this paper in a way that would allow reusing Sections 2 and 3 as-is for any potential baseline language, while, Section 4 can be leveraged as an example of analyzing a specific language in the context of the given requirements.

6 Related Work

In this section, we highlight a variety of related works ranging from high-level modeling approaches to more detailed approaches specific to Section 3.

Business process architectures. While usually not resulting in multi-model representations akin to the ones analyzed in Section 2, larger collections of processes are commonly managed using business process architecture models (also referred to as enterprise maps, process ecosystems, process landscapes, process maps, etc.) [21]. In these approaches, the focus is on representing the overall structure of business processes and their relationships at a high level of abstraction, while details of the processes (e.g., control flow) are usually omitted. A notable exception, bordering between organizational and process modeling, is the DEMO methodology [15], which incorporates multiple types of models, such as interstriction model, action model, process model, etc., that collectively constitute the essential model of an organization. However, as demonstrated by the Ford case in [15], that essential model may remain exactly the same even in the case of radical reengineering of the underlying business processes.

Process Choreographies and Compositions. Authors in [22] identify problems in the synchronization of independently defined but concurrently executed workflow models. [38] discusses how to compose business processes modeled using Petri nets. In particular, the Petri nets are connected using shared places, thus enabling synchronization. The above approaches do not consider data nor

declarative components. Other formally well-defined alternatives could be Colored Petri nets [24] and Open Petri Nets [8], while a less formal approach could be achieved with collaboration and choreography diagrams of BPMN [30].

Hybrid and Hierarchical Modeling. Authors in [33] highlight the need for hybrid process modeling approaches and propose a research agenda for their development. Later on, other works have defined formal semantics for hybrid processes. For example, [37] presents a formal semantics that uses a hierarchy of models, where each of model may be specified in either an imperative or declarative fashion. A conceptual framework and a common terminology for hybrid models has been proposed in [7] and a number of open research challenges related to hybrid processes have been identified in [36].

Process-Spanning and Instance-Spanning Constraints. Instance and process spanning constraints have been addressed from several perspectives like modeling and enactment [27], supporting runtime and design time verification through formalization with Event Calculus [18] and elicitation of patterns relying on Proclets and timed colored workflow nets [40]. Though these approaches provide means for handling interactions between multiple instances, they mainly require specific formalizations hampering the seamless integration of models expressed in different modeling languages. Therefore, those approaches are only contingently suitable for a multi-model paradigm as envisioned in this paper.

7 Conclusion

As highlighted in the introduction, this paper proposes that scenarios, where one or more processes are best represented as sets of smaller interconnected models, are sufficiently prevalent to warrant a dedicated line of research, which we call a *Multi-Model paradigm* for BPM. In support of this, we derived corresponding modeling requirements from four fundamentally different scenarios and, furthermore, explored a potential baseline language for meeting these requirements. In doing so, we have taken a crucial step in our overarching efforts to provide a fully-fledged framework for addressing a plethora of research areas from hybrid processes to the broader issues related to "process silos".

While the focus of this paper is on the language presented in [4], we described the characteristics of the paradigm in a language-agnostic manner. As a result, both the analysis of the modeling scenarios and also the corresponding requirements can be used as input for developing a similar multi-model solution from scratch or on the basis of any other existing approaches.

For future work, we plan to further investigate the extensions of [4] proposed in this paper with the goal of developing a corresponding complete formalization. This will, in turn, enable us to tackle concrete problems, related to IS development, such as automatic validation and execution support for multi-model scenarios. We also believe that modeling patterns in the style of [35] would need to be developed further down the line. Finally, we reiterate that the resource perspective and potential inclusion of KPIs warrants further investigation.

Acknowledgements. The work of A. Alman was supported by the Estonian Research Council grant PRG1226. F.M. Maggi was supported by the UNIBZ project PRISMA.

References

1. van der Aalst, W.M.P.: Process-aware information systems: Lessons to be learned from process mining. Trans. Petri Nets Other Model. Concurr. **2**, 1–26 (2009)
2. van der Aalst, W.M.P.: Object-centric process mining: Unraveling the fabric of real processes. Mathematics **11**(12) (2023)
3. Alman, A., Maggi, F.M., Montali, M., Patrizi, F., Rivkin, A.: Multi-model monitoring framework for hybrid process specifications. In: CAiSE. vol. 13295, pp. 319–335. Springer (2022)
4. Alman, A., Maggi, F.M., Montali, M., Patrizi, F., Rivkin, A.: A framework for modeling, executing, and monitoring hybrid multi-process specifications with bounded global–local memory. Information Systems **119**, 102271 (2023)
5. Alman, A., Maggi, F.M., Montali, M., Patrizi, F., Rivkin, A.: Monitoring hybrid process specifications with conflict management: An automata-theoretic approach. Artificial Intelligence in Medicine **139**, 102512 (2023)
6. Alman, A., Maggi, F.M., Montali, M., Peñaloza, R.: Probabilistic declarative process mining. Inf. Syst. **109**, 102033 (2022)
7. Andaloussi, A.A., Burattin, A., Slaats, T., Kindler, E., Weber, B.: On the declarative paradigm in hybrid business process representations: A conceptual framework and a systematic literature study. Inf. Syst. **91**, 101505 (2020)
8. Baldan, P., Corradini, A., Ehrig, H., König, B.: Open petri nets: Non-deterministic processes and compositionality. In: ICGT. Lecture Notes in Computer Science, vol. 5214, pp. 257–273. Springer (2008)
9. Bandara, W., Van Looy, A., Rosemann, M., Meyers, L.: A call for 'holistic' business process management. In: Problems@BPM. CEUR Workshop Proceedings, vol. 2938, pp. 6–10. CEUR-WS.org (2021)
10. Beerepoot, I., et al.: The biggest business process management problems to solve before we die. Comput. Ind. **146**, 103837 (2023)
11. Burattin, A., Maggi, F.M., Sperduti, A.: Conformance checking based on multi-perspective declarative process models. Expert Syst. Appl. **65**, 194–211 (2016)
12. de Leoni, M., Felli, P., Montali, M.: A holistic approach for soundness verification of decision-aware process models. In: ER. vol. 11157, pp. 219–235. Springer (2018)
13. Debois, S., Hildebrandt, T.T., Slaats, T.: Hierarchical declarative modelling with refinement and sub-processes. In: BPM. vol. 8659, pp. 18–33. Springer (2014)
14. Di Ciccio, C., Maggi, F.M., Mendling, J.: Discovering target-branched declare constraints. In: BPM. vol. 8659, pp. 34–50. Springer (2014)
15. Dietz, J.L.G.: Understanding and modelling business processes with DEMO. In: ER. Lecture Notes in Computer Science, vol. 1728, pp. 188–202. Springer (1999)
16. van Dongen, B.: Bpi challenge 2017 (2017). https://doi.org/10.4121/uuid:5f3067df-f10b-45da-b98b-86ae4c7a310b
17. Dumas, M., van der Aalst, W.M.P., ter Hofstede, A.H.M. (eds.): Process-Aware Information Systems: Bridging People and Software Through Process Technology. Wiley (2005)
18. Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: BPM. vol. 9850, pp. 348–364. Springer (2016)

19. Fdhila, W., Knuplesch, D., Rinderle-Ma, S., Reichert, M.: Verifying compliance in process choreographies: Foundations, algorithms, and implementation. Inf. Syst. **108**, 101983 (2022)
20. Felli, P., Montali, M., Winkler, S.: Linear-time verification of data-aware dynamic systems with arithmetic. In: In Proc. of AAAI. pp. 5642–5650. AAAI Press (2022)
21. Gonzalez-Lopez, F., Bustos, G.: Business process architecture design methodologies - a literature review. Bus. Process. Manag. J. **25**(6), 1317–1334 (2019)
22. Heinlein, C.: Workflow and process synchronization with interaction expressions and graphs. In: Georgakopoulos, D., Buchmann, A. (eds.) International Conference on Data Engineering. pp. 243–252 (2001)
23. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009). https://doi.org/10.1007/B95112
24. Jensen, K., Kristensen, L.M.: Coloured Petri Nets - Modelling and Validation of Concurrent Systems. Springer (2009)
25. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (2007)
26. La Rosa, M., van der Aalst, W.M.P., Dumas, M., Milani, F.: Business process variability modeling: A survey. ACM Comput. Surv. **50**(1), 2:1–2:45 (2017)
27. Leitner, M., Mangler, J., Rinderle-Ma, S.: Definition and enactment of instance-spanning process constraints. In: WISE. vol. 7651, pp. 652–658. Springer (2012)
28. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P.: Balanced multi-perspective checking of process conformance. Computing **98**(4), 407–437 (2016)
29. Mannhardt, F., de Leoni, M., Reijers, H.A., van der Aalst, W.M.P., Toussaint, P.J.: Guided process discovery - A pattern-based approach. Inf. Syst. **76**, 1–18 (2018)
30. OMG: Business process model and notation (BPMN). Technical report, Object Management Group (2014), `https://www.omg.org/spec/BPMN/2.0.2/PDF`
31. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC. pp. 287–300. IEEE Computer Society (2007)
32. Popova-Zeugmann, L.: On time petri nets. J. Inf. Process. Cybern. **27**(4), 227–244 (1991)
33. Reijers, H.A., Slaats, T., Stahl, C.: Declarative modeling-an academic dream or the future for bpm? In: BPM. vol. 8094, pp. 307–322. Springer (2013)
34. Rinderle-Ma, S., Gall, M., Fdhila, W., Mangler, J., Indiono, C.: Collecting examples for instance-spanning constraints. CoRR **abs/1603.01523** (2016)
35. Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M.: Workflow Patterns: The Definitive Guide. MIT Press (2016)
36. Slaats, T.: Declarative and hybrid process discovery: Recent advances and open challenges. J. Data Semant. **9**(1), 3–20 (2020)
37. Slaats, T., Schunselaar, D.M.M., Maggi, F.M., Reijers, H.A.: The semantics of hybrid process models. In: OTM Conferences. vol. 10033, pp. 531–551 (2016)
38. van Hee, K.M., Sidorova, N., van der Werf, J.M.E.M.: Business process modeling using petri nets. Trans. Petri Nets Other Model. Concurr. **7**, 116–161 (2013)
39. Westergaard, M., Maggi, F.M.: Looking into the future. Using timed automata to provide a priori advice about timed declarative process models. In: OTM Conferences (1). vol. 7565, pp. 250–267. Springer (2012)
40. Winter, K., Rinderle-Ma, S.: Defining instance spanning constraint patterns for business processes based on proclets. In: ER. vol. 12400, pp. 149–163. Springer (2020)
41. Winter, K., Stertz, F., Rinderle-Ma, S.: Discovering instance and process spanning constraints from process execution logs. Inf. Syst. **89**, 101484 (2020)