Towards Dynamic Business Process Management: Clouding Adaptation Processes that Adapt Processes

Roy Oberhauser

Computer Science Department, Aalen University, Aalen, Germany
roy.oberhauser@hs-aalen.de

Abstract. Dynamic business process management (dBPM) is contingent on the practical viability of automated process adaptation techniques. Various approaches to support process adaptation have been investigated. Yet they typically expect some level of manual interaction or involve some amalgamation of additional modeling paradigms or language extensions. Additionally, cross-cutting process adaptation concerns and a distributed and cloud-based process adaptation capability have not been adequately addressed. AProPro (Adapting Processes via Processes), a flexible and cloud-capable approach towards dBPM, supports adapting target processes using adaptation processes while retaining an intuitive and consistent imperative process paradigm. The evaluation consists of case studies in both a business and an engineering domain and demonstrates the approach in a distributed Adaptation-as-a-Service cloud setting. The results show the viability of the approach across various domains.

Keywords: Dynamic Business Process Management, Dynamic BPM, Adaptive Process-Aware Information Systems, PAIS, Adaptive Workflow Management Systems, Process Change Patterns, Aspect-Oriented Processes, Cloud-based BPM, Adaptation-as-a-Service, Web Services.

1 Introduction

A trend toward increasing automation is rapidly affecting all areas of business. Correspondingly, business processes are being increasingly modeled and enacted in process-aware information systems (PAIS). Dynamic business process management (dBPM) seeks to support the reactive or evolutionary modification or transformation of business processes based on environmental conditions or changes. The technical realization of business processes, known as executable processes or workflows, are implemented in what is known as either a business process management system (BPMS) [1], workflow management system (WfMS) [2], or PAIS [3].

In addressing process adaptation, many PAISs today lack dynamic runtime adaptation with correctness and soundness guarantees [4]. If such dynamic adaptation is supported, it is typically limited to support for manual change interaction by a process actor [3]. Typical types of recurring modifications to workflows are known as workflow control-flow patterns [5], change patterns [6], or adaptation patterns [3].

In light of the dBPM vision, the increasing degree of process automation will necessitate a corresponding need to support adaptation by both human and software agents. Our previous work on an adaptable context-aware and semantically-enhanced PAIS in the software engineering domain [7][8][9] included automated adaptation work and work on supporting the user-centric intentional adaptation of workflows [10]. However, an open challenge remains in practically expressing and maintaining adaptations in an intuitive manner for process modelers and process actors for a sustainable dBPM lifecycle.

Adapting Processes via adaptation Processes (AProPro) contributes a practical and flexible cloud-capable approach for supporting dBPM in a generalized way that can be readily implemented and integrated with current adaptive PAIS technology. This chapter extends [11] and features case studies in multiple domains (business and engineering) to demonstrate its applicability and domain-independence. The approach maintains the ease and accessibility of the more intuitive imperative paradigm for process adaptations by modelers and process actors or users. It can thus further the reusability, portability, maintainability, and sharing of adaptations, including the cloud-based provisioning of adaptation processes within the community, thus supporting sustainable adaptability by extending an adaptation process's lifecycle. The evaluation demonstrates its viability and its performance in the cloud.

The chapter is organized as follows: section 2 describes related work, which is followed by a description of the solution approach. A technical realization is described in section 4, which is followed by an evaluation. Section 6 then concludes the chapter. Since this chapter focuses on the technical implementation of a process, the terms workflow and process are used interchangeably.

2 Related Work

Various approaches support the manual or automated adaptation of workflows. The survey by [12] provides an overview from the perspective of support for correctness criteria, while [6] provides an overview based on the perspective of change patterns and support features.

Declarative approaches, such as DECLARE [13] support the constraint-based composition, execution, and adaptation of workflows. *Case handling approaches*, such as FLOWer [14], typically attempt to anticipate change. They utilize a case metaphor rather than require process changes, deemphasize activities, and are data-driven [3][1]. [15] provides a review of case modeling approaches. Case-based approaches towards adapting workflows include [16]. *Agent-based approaches* support automated process adaptations applied by autonomous software agents. Agentwork [17] applies predefined change operations to process instances using rules. [18] applies a belief-desire-intention (BDI) agent using a goal-oriented BPMN (Business Process Modeling Notation) [19] language extension. *Aspect-oriented approaches* include AO4BPEL [20] and AO4BPMN [21], both of which require language extensions. *Variant approaches* include: Provop [22], which supports schema variants with pre-configured adaptations to a base process schema; and vBPMN [23] that extends BPMN with fragment-based adaptations via the R2ML rule

language. rBPMN [24] also interweaves BPMN and R2ML. *Automated planning* and *exception-driven adaptation approaches* include SmartPM [25], which utilizes artificial intelligence, procedural, and declarative elements.

AProPro differs in that it is an imperative workflow-based adaptation approach that does not require a case metaphor and is activity-, service-, and process-centric with regard to runtime adaptation. Additionally, neither language extensions nor other paradigms such as rules, declarative elements, or intelligent agents are needed. Furthermore, distributed and cloud-based adaptations to either instances or schemas are supported.

3 Solution Approach

The AProPro approach follows an imperative process style. A guiding principle of the AProPro approach is that adaptations to processes should themselves be modeled as processes, remaining consistent with the process paradigm and mindset. Process models are kept as simple and modular as reasonable for typical usage scenarios, in alignment with the orthogonal modularity pattern [26]. Special cases can either be separated out or handled as adaptations via adaptation processes, thus simplifying the target process. The solution scope focuses primarily on adapting process control structures, and not necessarily all adaptations to processes can be accomplished with this approach. In particular, internal activity changes, non-control and (internal) data structure changes, and implicit dependencies are beyond the scope of this chapter.

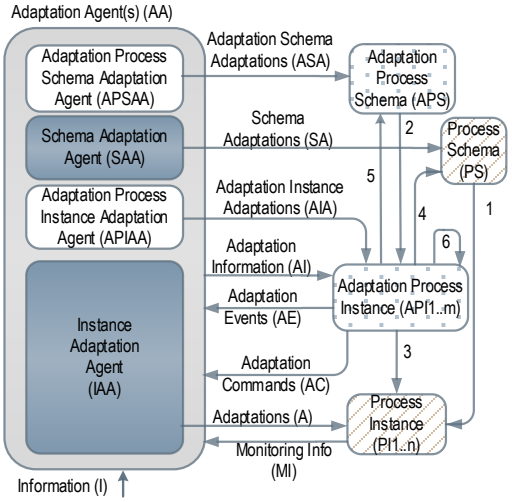

Fig. 1. Conceptual solution architecture.

The following description of the solution approach will highlight certain perspectives. As shown in , *Process Instances (PI1..n)* are typically instantiated *(1)* based on some *Process Schema (S)* within a given PAIS (filled with diagonal

hatching). In adaptive PAISs, *Adaptation Agents (AA)* (shown on the left with a solid fill), be they human or software agents, utilizing or reacting to *Information (I)* (e.g., external information such as context or other internal system information such as planning heuristics) or *Monitoring Information (MI)*, trigger modifications to various process structures. A *Schema Adaptation Agent (SAA)*, such as a process designer or modeler, makes *Schema Adaptions (SA)* to one or more *Process Schema (PS)*. An *Instance Adaptation Agent (IAA)*, such as a process actor or user, may perform *Adaptations (A)* on some *Process Instance (PI)*. Support for such adaptation has been available in adaptive PAISs, e.g., the ADEPT2-based AristaFlow [3].

Workflow-driven adaptations of workflows: adaptations, for instance in the form of adaptation patterns, are specified as workflows that operate on another workflow. For this (see dotted fill), an *Adaptation Process Schema (AS)* is created or modified from which one or more *Adaptation Process Instances (API1..m)* are instantiated *(2)* in the same or a different PAIS. The *(API)* to target *(PI)* relation may be one-to-one, one-to-many, many-to-one, or many-to-many. Utilizing *Adaptation Information (AI)* such as events, triggers, or state, automated instructions denoted as *Adaptation Commands (AC)* can be sent to an *Instance Adaptation Agent (IAA)* that executes *Adaptations (A)* on one or more *(PI)*. Note that in certain PAIS architectures, a direct adaptation mechanism *(3)* that avoids the *(IAA)* intermediary may exist, with *(API)* acting as an *(IAA)*. *(API)* may provide *Adaptation Events (AE)*, e.g., so that an *(AA)* can be aware of the current state of an *(API)*.

Adaptation patterns as workflows: adaptation patterns (insert, delete, move, replace, swap, inline, extract, parallelize, etc.) can be integrated in workflows and applied conditionally based on *Adaptation Information (AI)*, e.g., in the form of process variables or events.

Aspect-oriented adaptations: this is supported by modularizing and constraining an adaptive workflow to operate on one aspect (such as authorization), while having other adaptation workflows address others. The many-to-many relations between *(API)* and *(PI)* or *Schemas (PS)* was previously mentioned. Congruent with the chain-of-responsibility design pattern, adaptations can be modularized and chained.

Variation points: these can be intentionally incorporated via markers for explicit adaptation support during process modeling, e.g., given insufficient modeling information. Adaptation workflows can then dynamically "fill in" these areas during process configuration or enactment.

Adapting adaptation workflows: This concept supports a further degree of flexibility by supporting adaptation workflows operating on (other) adaptation workflows. *Adaptation Process Instances (API)* send *Adaptation Commands (AC)* resulting in *Schema Adaptations (SA)* to a *Process Schema (PS)*, either via a *Schema Adaptation Agent (SAA)* or directly via *(4)*. In a similar fashion, *Adaptation Schema Adaptations (ASA)* can be applied to an *Adaptation Process Schema (APS)* via an *Adaptation Process Schema Adaptation Agent (APSAA)* or directly via *(5)*. Note that in this case, an *(API)* can change its own schema *(APS)* or those of others, and potentially change itself *(API)* or other instances *(API)*, possibly even directly via *(6)*.

Recursive adaptation: instead of separating the *(API)* from its target *(PI)*, if preferable (for instance, to access contextual data), a *(PI)* can include its own *(APS)* fragments and thus become self-modifying.

Exception-based adaptations: (un)anticipated exceptions can be used to trigger the enactment of adaptation workflows within exception handlers.

Reactive and proactive adaptations: in support of dBPM, event- and context-driven changes can automatically trigger and cause automated predictive or reactive runtime adaptations to be incorporated on an as-needed basis, rather than taking all possibilities into process models a priori.

Push-or-pull adaptations: in the push case, the adaptive workflow is triggered outside of the workflow and then applies its changes to the target; in the pull case, the target workflow triggers the adaptive workflow to initiate its adaptations.

Reusability: shared modeled/tested adaptation workflows support the wider reuse of adaptation patterns in the community, e.g., via repositories like APROMORE [27].

Composability: more complex adaptations can be addressed by composing multiple adaptation workflows, e.g., via sub-processes into larger ones.

Process Compliance and Governance: the (AP) can be used to verify expected structural and state conditions (no changes applied), or to additionally apply adaptations when these are not in compliance.

Cloud-based provisioning of adaptation workflows: the concept supports operating in a distributed and PAIS-independent (heterogeneous) manner on other workflows in other clouds, making these adaptation workflows readily available to operate on others as needed. Shared tenancy and pay for use could reduce infrastructural costs.

Service-oriented adaptation services: the approach supports the ability to provision and support adaptations-as-a-service (AaaS) in the cloud.

AProPro supports the goals of dBPM by enabling desired automated or semi-automated adaptations, while allowing process modelers and users to remain in their current imperative process paradigm and modeling language without requiring language extensions. Empirical findings that support such an approach includes: [28] who empirically investigated understandability issues with declarative modeling, and found that subjects tended to model sequentially and had difficulty with combinations of constraints; [29] determined that imperative models have better understandability and comprehensibility than declarative ones; [30] suggests that process modularity via information hiding enhances understandability; and [31], which showed that process complexity affected maintenance task efficiency for process variant construction - here subjects preferred high-level change patterns to process configuration.

4 Realization

To verify the feasibility of the AProPro approach, key aspects of the solution concept were implemented utilizing the adaptive PAIS AristaFlow. The solution approach required no internal changes to this PAIS, relying exclusively on its available extension mechanisms via its generic Java method execution environment. RESTful web services were used for cloud interaction. Adaptation workflow activities utilize a `StaticJavaCall` to invoke the extension code contained in a Java ARchive (JAR) file, which sends change requests to a REST server in the same or another PAIS.

To support heterogeneity, both the communication and the change requests are PAIS agnostic. They could thus be invoked and sent by any PAIS activity in any adaptation workflow located anywhere. Only the actual workflow change operations require a PAIS-specific API. Other PAIS implementations can be relatively easily integrated via plug-in adapters.

4.1 Adaptation Patterns and AaaS

The initial realization focused primarily on demonstrating key AProPro and AaaS capabilities. One such capability is realizing *adaptation patterns as workflows*, specifically those basic to typical adaptations: inserting, deleting, and moving process fragments. On this basis, more complex adaptation workflows can be readily built. For instance, the replace change pattern was realized as a subprocess consisting of an insert and a delete operation, demonstrating the *composability* capability.

The AristaFlow application programming interface (API) expects various method input parameters in order to modify a workflow. Thus, pattern implementations were designed to include these expected values even if some are optional.

The *insert process fragment pattern*, shown in Fig. 2(a), takes the following input parameters:

- *procID*: ID of the target process instance;
- *pre*: ID of the predecessor node;
- *suc*: ID of the successor node;
- *activityID*: ID of activity assigned to node;
- *newNodeName*: name of the new node;
- *staffAssignmentRule*: of this node;
- *description*: of this node;
- *readParameter*: input parameters for the new node;
- *writeParameter*: output parameters of the new node.

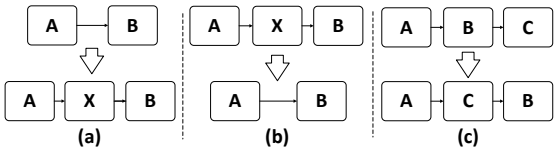

Fig. 2. (a) Insert, (b) delete, and (c) move process fragment patterns.

The *delete process fragment pattern*, shown in Fig. 2(b), takes the following input parameters:

- *procID*: ID of the target process instance;
- *nodeID*: ID of the node to be deleted.

The *move process fragment pattern*, shown in Fig. 2(c), takes the following input parameters:

- *procID*: ID of the target process instance;
- *pre*: ID of the new predecessor node;
- *suc*: ID of the new successor node;
- *nodeID*: ID of the node to be moved.

The *replace* pattern was realized as a subprocess that uses the insert and delete patterns as shown in Fig. 3.

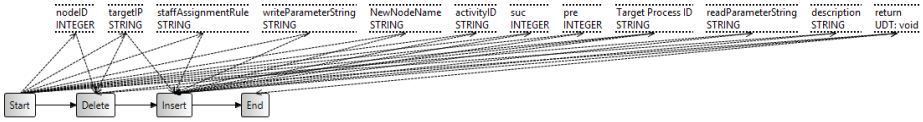

Fig. 3. The Replace process fragment sub-process.

RESTful web services were created in Java according to JAX-RS using Apache CXF 2.7.7, with Java clients using Unirest 1.4.5. For basic pattern AaaS services, the following corresponding REST operations were provided at the target PAIS containing the target workflows to be modified, with procID passed in the URI and the rest of the inputs described above passed as parameters:

- `PUT /procID/{procID}/insert`
- `PUT /procID/{procID}/delete`
- `PUT /procID/{procID}/move`

The following REST operations provide examples of the interfaces for more advanced domains-specific AaaS services that invoke specific adaptation workflows that modify a separate target workflow instance. Such processes are explained later in the evaluation section, while example technical interface parameters of certain adaptation service implementations are given here:

- For quality assurance:
  `PUT /procID/{procID}/adapt/qa` with the string parameters urgent, high risk, junior engineer, and targetIP;
- For test-driven development:
  `PUT /procID/{procID}/adapt/tdd` with the string parameters testDriven and targetIP.

4.2 Implementation Details

The AaaS client-side adaptation process nodes internally invoke static methods in the `ChangeOperations` class for that change pattern (`doInsertProcessFragment` or equivalent). This method invokes a REST client that sends the corresponding request to a REST server. A class diagram of the server-side implementation is shown in Fig. 4. The service determines the type of *Request*, on which basis a corresponding (e.g., *InsertProcessFragment*-) *Command* object is instantiated and passed to a *Controller*, which determines when and in what sequence to execute a given command. To

support heterogeneity, the commands utilize the corresponding *Supplier* classes which utilize PAIS-specific APIs for the operations.

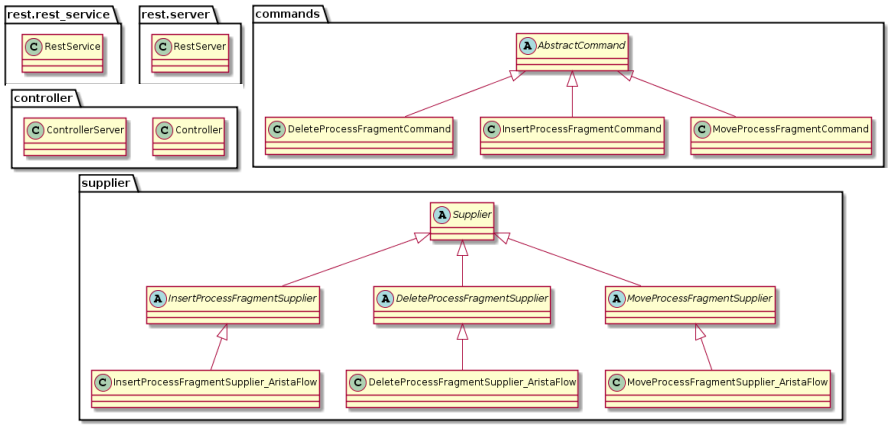

Fig. 4. Class diagram showing key implementation packages and classes.

A lock is acquired for the AristaFlow target process instance and a *ChangeableInstance* object is generated. All changes are first applied to this *ChangeableInstance* object. When the changes are committed, the entire instance is checked by AristaFlow for correctness. If the changes are correct, the actual process instance is modified accordingly. If errors were found, the changes are rejected and the actual process instance remains unchanged.

5 Evaluation

The evaluation focused on the solution concept and a prototype realization thereof. The case studies focused on demonstrating key process adaptation capabilities supported by the concept and its domain independence. The solution concept envisions provisioned adaptation workflows in the cloud that are available to operate on other workflows. Since certain reactive dBPM scenarios may be sensitive to delays, the technical evaluation encompassed cloud performance measurements. In this regard, workflows operating across geographically separate PAISs utilizing a basic cloud configuration would represent a worst case area of the performance spectrum.

As the solution concept is domain independent, a case study in the business domain and the software engineering (SE) domain are used to illustrate the capabilities and adaptation effects. For brevity and readability, certain branches and loops that would likely be involved in realistic models are omitted. Moreover, to maintain readability some screenshots are possibly cropped to remove obvious start and end nodes, while wide horizontal screenshots were cropped and stacked with a curved connector included to indicate the continuation of the workflow.

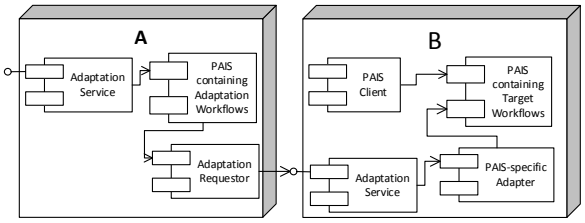

Fig. 5. The evaluation setup.

Fig. 5 shows the evaluation setup. System A, which ran the adaptation workflows, was an Amazon AWS EC2 t2.micro instance eu-central-1b in Frankfurt, Germany consisting of an Intel Xeon E5-2670 v2@2.50GHz, 1 GB RAM, 1 Gbps network, AristaFlow PAIS 1.0.92 - r19, Windows 2012 R2 Standard x64, and Java 1.8.0_45-b15. System B was an equivalent Amazon AWS EC2 t2.micro instance on the US West Coast (Northern California) us-west-1b containing the target workflows. A remote configuration means A is active and communicates with its target on B. A collocated AWS configuration implies that the Adaptation Process is collocated with the Target Process within the same PAIS in the cloud. In a local PC configuration testing adaptations in the first case study, both A and B were collocated on a single PC but commands are still sent via REST.

5.1 Case Study in the Business Domain: Travel Booking

The common conventional approach towards process modeling is restricted in its ability to consider the complexity and reality of real-life workflows, both in the model and its execution. In our view, many of the business process models are artificially simplified in order to be viable.

Therefore, to demonstrate the adaptation capability of the AProPro approach, we chose to model an adaptive variant based loosely on the BPMN 2 Travel Booking Example [32] executed in a local PC configuration. To simplify the diagrams and description for this case study, a limited set of factors, variants, constraints, and assumptions affecting the adaptations are shown, while other basic data such as name, travel dates, etc. are ignored. The following six transportation options exist: rental car, taxi, train, plane, company plane, and helicopter. We also assume that the different transportation options will require different booking steps, and thus we differentiate these workflows rather than assuming that one generic one will do.

5.1.1 Initial travel booking workflow. The initial basis for the *travel booking* workflow is shown in Fig. 6. At the point in time when the process is modeled, the preferences and contextual information are unknown. Conventional process models must consider all possible combinations. Additional (contextual or preferential) factors that could still be integrated include severe weather risk, maximum layover duration preference on connecting flights, etc. The workflow will be described below.

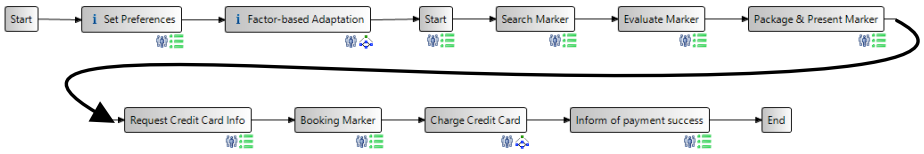

Fig. 6. *Travel booking* initial workflow basis.

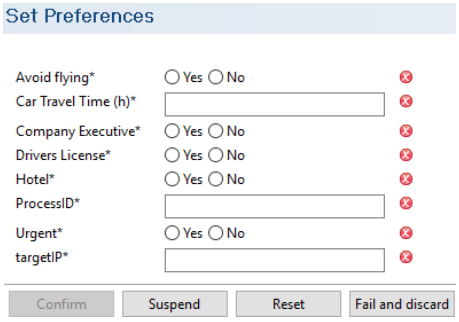

Fig. 7. Screenshot of the *Set Preferences* user interface.

1. The *Set Preferences* node in Fig. 6 takes the preference values shown in Fig. 7. The fields targetIP and ProcessID can be used for remote adaptations. The following preferences can be set:
   - Company executive (yes/no): yes expands the available transportation options with company plane and helicopter.
   - Driver's license (yes/no): if the person has a valid and unsuspended license then rental car is enabled.
   - Urgent (yes/no): if yes, then avoid train. If executive and less than 2 hours and not overseas, then enable helicopter option.
   - No fly list (yes/no): if someone has a flying phobia or is not allowed to fly then air transportation options are disabled.
   - Duration of travel (hours): If car travel duration time is greater than some threshold (e.g., >=1 hour, then disable taxi option; >=2 hours car travel, then enable train option; if >=4 hours, then enable airplane)

   The following environmental context is determined automatically:
   - Unavailability of certain travel options (here due to airline/airport strike or train strike): for simplicity, we assume that a strike affects all transportation of that type (this could of course be adjusted to affect only a single airline or single region).
   - HotelMoreDifficult (yes/no): if due to high expected occupancy rates, e.g., due to a convention or event in the city (e.g., Oktoberfest) reserving a hotel is likely more difficult than reserving a flight, then attempt to book a room first, and then base the flight dates on the available.

2. The *Factor-based Adaptation* node in Fig. 6 triggers the enactment of the adaptation workflows as a subprocess and is described further in Section 5.1.2.
3. The second *Start* node in Fig. 6 waits for a data variable to determine that all adaptations have completed before proceeding.
4. The *Search Marker* node in Fig. 6 is a variation point that is replaced with search nodes applicable to the search criteria for this process instance (e.g., train and rental car only)
5. The *Evaluate Marker* node in Fig. 6 is a variation point to filter and calculate the best offers.
6. The *Package and Present Marker* node in Fig. 6 is a variation point to present the offers and allow an automated or human agent to make its selection.
7. The *Request Credit Card Info* node in Fig. 6 takes in the information required for payment.
8. The *Booking* marker node in Fig. 6 is a variation point to book the reservations.
9. The *Charge Credit Card* in Fig. 6 invokes a subprocess to charge the credit card with the actual amount. This area could also have been adapted based on a payment type preference (which we left out for simplicity).
10. The *Inform of Payment success* in Fig. 6 node notifies the appropriate parties that payment occurred. To demonstrate error-triggered adaptation, if payment was unsuccessful, an error-triggered adaptation replaces the node with a *Inform of Payment failure* node, but could just as well insert a retry loop or offer alternative payment types. Moreover, instead of explicitly invoking the adaptation subprocess as it is here, this adaptation workflow could be placed in an error handler and triggered via a process exception.

5.1.2 Travel booking adaptation workflows. The *Factor-based Adaptation* workflow in Fig. 8 was implemented as a subprocess and triggered by the *travel booking* workflow as described above. As such, it demonstrates the pull adaptation capability, since the adaptations are pulled-in by the target workflow itself. A triggering and "push" of these adaptations on the target workflow, for instance via some external event such as the end of a train strike, would also be feasible, but would require further checks on the process instance state and its progress relative to the adaptation locations to ensure that it has not progressed beyond the adaptation locations. The contextual factors were simulated using a random number generator. The nodes of the workflow are described below:

1. The *Check if hotel more difficult* node returned with 70% likelihood that the hotel was more difficult to book than the flight.
2. The *Check for Airline/Airport strike* and *Check for Train Strike* nodes returned with 50% likelihood that a strike occurred.
3. The *Check for hotel first* node determines if the *Adapt for Hotel* is invoked first, if the *Adapt for Travel* first, or if only *Adapt for Travel* is invoked (assumes some travel is necessarily involved).

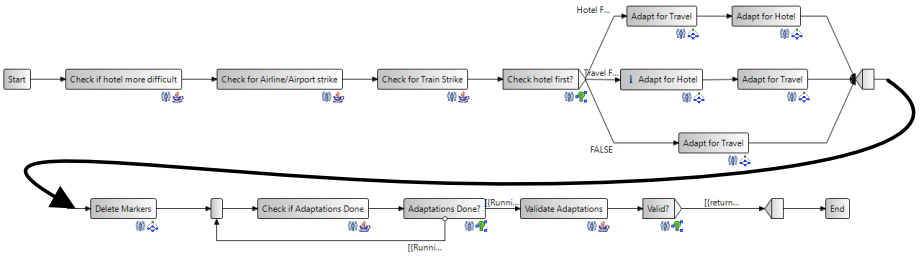

Fig. 8. *Factor-based Adaptation* workflow.

4. The *Adapt for Hotel* node in Fig. 8 invokes the subprocess shown in Fig. 9 and involves the steps of searching, evaluating, presenting, and booking a hotel room. While the workflow was executed, the case study invoked no actual actions.

Fig. 9. *Adapt for Hotel* adaptation workflow.

5. The *Adapt for Travel* node in Fig. 8 invokes the subprocess shown in Fig. 10. It determines the appropriate type of travel medium that, in turn, invokes one of the adaptation subprocesses shown in Fig. 11. While these workflows were executed, for the case study no actual actions were invoked. The reason for assuming custom workflows rather than one generic workflow is that realistically each type of transportation may require different types of data and different service interfaces and interactions.

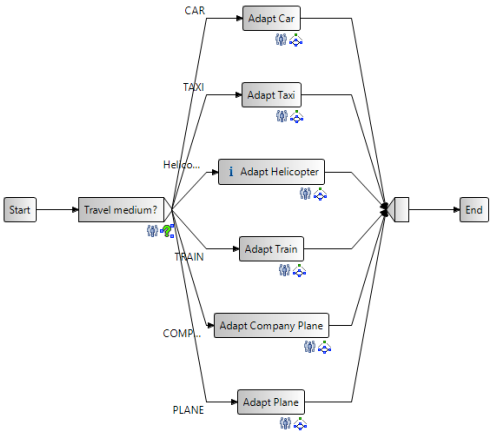

Fig. 10. The *Adapt for Travel* adaptation workflow.

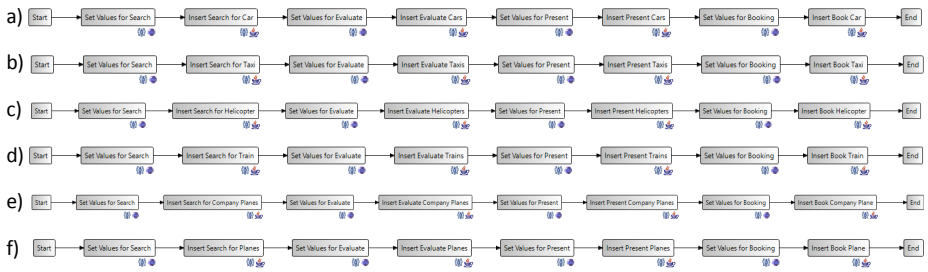

Fig. 11. The a) *Adapt Car* b) *Adapt Taxi* c) *Adapt Helicopter* d) *Adapt Train* e) *Adapt Company Plane* and f) *Adapt Plane* adaptation workflows.

6. After the adaptations are invoked, the *Delete Markers* node in Fig. 8 invokes the *Delete Markers* adaptation workflow shown in Fig. 12 as a subprocess to delete any variation point markers remaining in the target workflow.

Fig. 12. The *Delete Markers* adaptation workflow.

7. The *Check if Adaptations Done* and *Adaptations Done* in Fig. 8 is a loop that waits for the requested adaptations to complete before continuing.
8. As an example for a governance and compliance checking capability, the *Validate Adaptations* and *Valid* nodes in Fig. 8 verify that at least one transport type, at least payment, at least one booking exist in the target workflow after all adaptations were applied. While this compliance checking is "pulled-in" here, it could also be placed in a separate workflow, triggered based on some event occurring, e.g., that an adaptation occurred, and thus reactively pushed externally without the target workflow being aware of or including or excluding the compliance check.

Going back to the initial workflow in Fig. 6, the *Charge Credit Card* node is a subprocess (shown in Fig. 13) that attempts to charge the credit card and, if unsuccessful, the *Payment failed* node invokes the *Payment failed* adaptation subprocess which uses the replace process fragment pattern that was shown in Fig. 3.

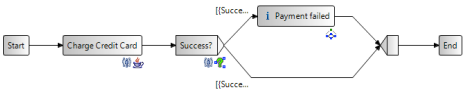

Fig. 13. The *Charge Credit Card* adaptation workflow.

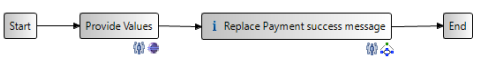

Fig. 14. The *Payment failed* adaptation workflow.

5.1.1 Resulting travel booking workflow. The resulting *travel booking* process instance is shown in Fig. 15 and, because it has already been adapted to its preferences and appropriate context, it exhibits a fairly simplistic workflow that does not show all of the possibilities that do not apply to its current context.

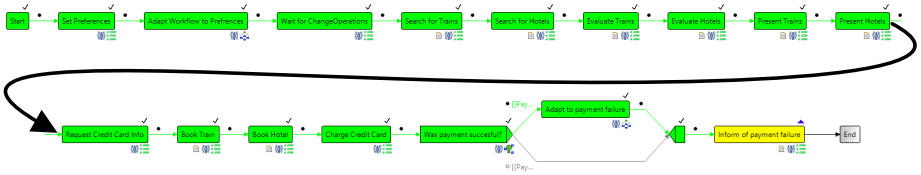

Fig. 15. Resulting travel booking workflow after the adaptations were applied.

This case study demonstrated workflow-driven adaptations of workflows, utilizing adaptation patterns as workflows, aspect-oriented adaptations (transportation vs. payment), variation points were demonstrated with markers being replaced with the actually required workflow activities (based on preferences and environmental context) during enactment, support for proactive (e.g., preferences) and reactive (e.g., payment error) adaptations, pull adaptations, composability in that larger adaptations were composed of smaller adaptation subprocesses, process compliance and governance, and service-oriented adaptation services, in that the adaptations were invoked via REST web services.

5.1.4 Conventional travel booking workflow. We modeled the *travel booking* workflow conventionally without taking any adaptations into account as shown in Fig. 16. Due to space constraints, it is not intended to be readable, but to show a possible process model structure and to determine an estimate of a possible number of nodes required. This workflow consisted of 153 nodes.

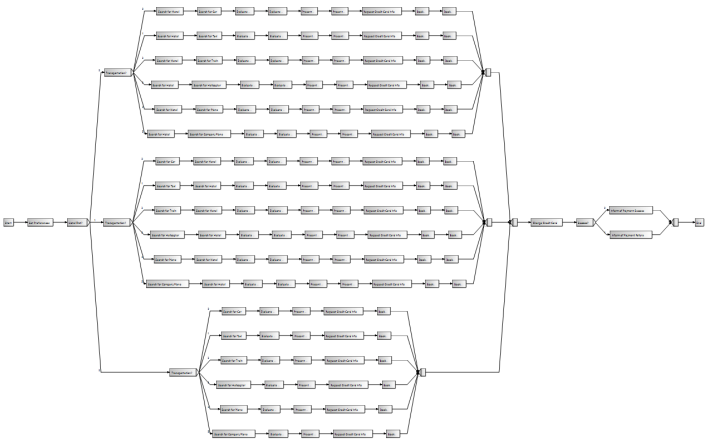

Fig. 16. Conventionally modeled *Travel booking* workflow (assuming no adaptation).

In summary, from this business domain case study we see that adaptations can be readily modeled and modularized in workflows, variation points can be incorporated, compliance checks included, and that both proactive and reactive adaptations are feasible. Since during process modelling the actual preferences and contextual information are often unclear, conventional modeling cannot yet utilize these and thus the conventional model typically must consider all possible combinations, resulting in unwieldy models. In contrast, with the AProPro approach the adaptations are specified as workflows, which can be modularized, easily maintained, and flexibly triggered and applied, resulting in a relatively simple adapted and thus tailored target workflow.

5.2 Case Study in Software Engineering

In addition to demonstrating the AProPro approach in a different domain, the following case study also involved a remote cloud deployment configuration to demonstrate the capability for cloud-based provisioning of adaptation workflows.

5.2.1 Sequential Waterfall Process. For a representative target for the application of adaptation processes, a sequential workflow was chosen, loosely following a waterfall process (WP) consisting of common SE activities for an approved software change request. It represents any standard process in a fictitious organization. The activity sequence is shown in Fig. 17.

Fig. 17. The initial (unadapted) *Waterfall Process* (WP).

5.2.2 Quality Assurance Adaptation Process. To demonstrate process governance and an Adaptation Process producing process variants, a Quality Assurance Adaptation Process (QAAP) variously adapts a target process based on situational factors.

Assume the SE process for a software change varies depending on its urgency, risk, and the worker's experience. The SE organization's policy normally expects at least a peer review before code is committed. The WP already includes this activity, although the adaptation workflow could also check policy compliance and insert such a missing activity. Three configurable boolean parameters were utilized for this process in Fig. 18: Urgent, High Risk, and Junior Engineer (denoting the worker experience level, with false implying a more senior worker). The 'SetConditions' task allows a user to set workflow values, which is skipped when invoked as a service. The following cases besides the default *Peer Review* (no change) were supported:

Code Review Case: A Code Review is required if the circumstances are 'NOT urgent AND (high risk OR junior engineer).' In this case:

- The node Peer Review is deleted via the Delete Process Fragment
- A node Code Review is inserted via the Insert Process Fragment Pattern

No Review Case: Foregoing a review is only tolerated when the situation is 'urgent AND NOT high risk AND NOT junior engineer.' In this case:

- The Peer Review node is removed via the Delete Process Fragment Pattern.

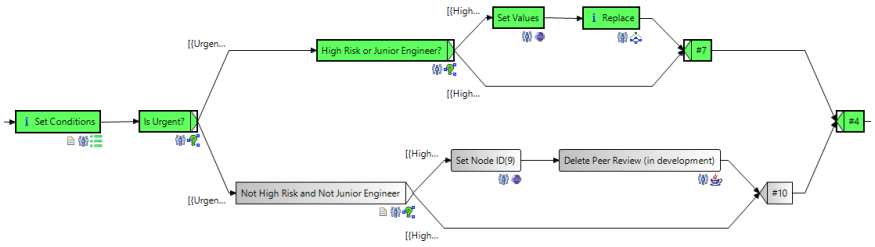

Fig. 18. Quality Assurance Adaptation Process (QAAP).

Fig. 19 shows the result of the application of QAAP to WP for 'not urgent and high risk', resulting in activity *Code Review* replacing *Peer Review*. In a context-aware dBPM environment, such input values could also be automatically determined.

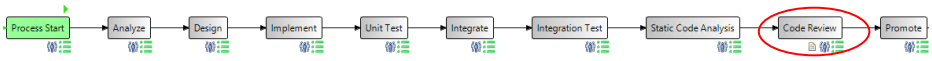

Fig. 19. WP after application of the QAAP.

5.2.3 TDD Adaptation Process. In test-driven development (TDD) [34], test preparation activities precede corresponding development activities. To support the TDD aspect in the WP, the *Unit Test* is executed before *Implement* and *Integration Test* executed before *Integrate*. Thus, the TDD Adaptation Process (TDDAP) shown in Fig. 20 utilizes the Move Process Fragment Pattern twice.

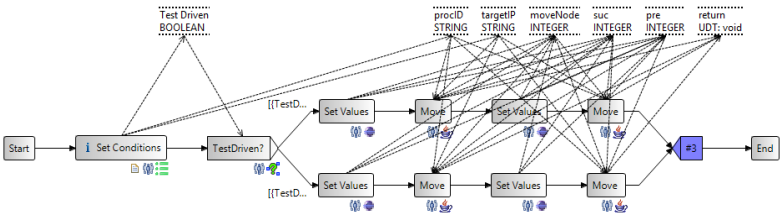

Fig. 20. Test-Driven Development Adaptation Process (TDDAP).

The resulting adaptations are shown in Fig. 21.

Fig. 21. WP after application of the TDDAP.

5.2.4 Aspect-oriented Adaptations. Multiple separate Adaptation processes can be advantageous for modularity and maintainability. Analogous to aspect-orientation, each aspect and its associated adaptations can be modeled in separate conditionally dependent adaptation processes. In Fig. 22, both QAAP and TDDAP were applied to the target WP, with each adaptation process representing a different aspect (reviews or testing).

Fig. 22. Waterfall Process after application of both Adaptation Processes.

5.2.5 Self-adaptive Processes. Self-adaptive processes support the integrative modeling of possible adaptations into the target processes themselves. As shown in Fig. 23, both the QAAP and TDDAP adaptations were modeled before the WP, with the target process for the adaptations being the enacting process instance itself. This demonstrates the feasibility of adapting adaptation workflows and of recursive adaptations, and in a similar way adaptations could be integrated into process exception handlers.

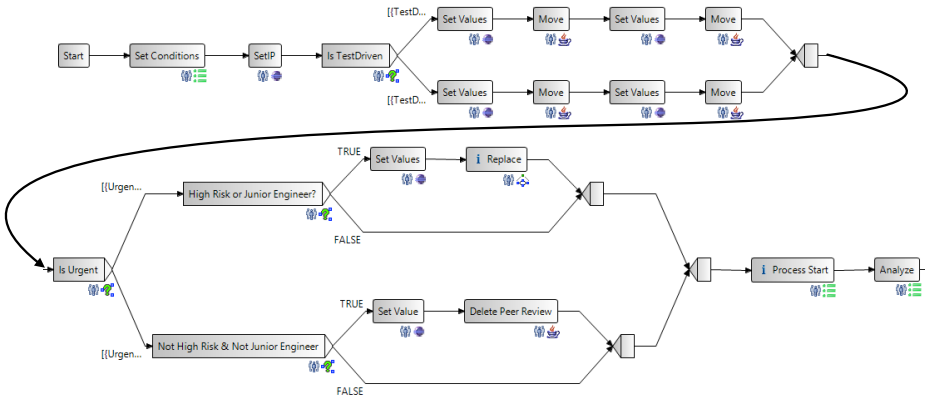

Fig. 23. Self-adaptive Waterfall Process partial screenshot (continues to right as in Fig. 17).

5.3 Measurements

To determine the performance of dBPM adaptation operations by an adaptation process in a geographically distributed cloud scenario, the durations for various basic operations (insert, delete, move) and adaptation processes (QAAP and TDDAP) were measured in a remote cloud configuration.

The test procedure was as follows: On system B, the Apache CXF REST server was started, and then the target workflow was manually started via the AristaFlow client to bring it into a started initialized state. On system A, an adaptation workflow was triggered by a REST client using Postman 2.0 in a Chrome web browser with which the necessary adaptation parameters were entered (e.g., procID, target workflow IP address, etc.). Activities in the adaptation workflow send adaptation requests via REST to system B. All latency and processing times were measured within systems A and B.

For the case that an initial measurement was significantly longer than the ones following (e.g., due to initialization and caching effects), this value was noted separately and not included in the average, since a dormant adaptation process might exhibit such an effect, whereas an active adaptation process would not. Each measurement was repeated in accordance with setup and test procedure described previously. To gather upper bounds, neither optimizations nor performance tuning were attempted.

Table 1 through Table 3 show the results for the execution of the basic adaptation operations insert, delete, and move respectively in a local and a remote configuration. The average was calculated from the four measurements that followed the initial measurement. To see if cloud network delays play a significant role, the network latencies and the adaptation times are differentiated, which is also depicted in Fig. 24.

Table 1: Insert operation duration (in seconds).

	Local (B to B)		Remote (A to B)	
	Initial	Average	Initial	Average
Adaptation	4.033	3.468	3.588	3.203
Latency	0.418	0.373	0.686	0.675
Total	4.451	3.842	4.275	3.878

Table 2: Delete operation duration (in seconds).

	Local (B to B)		Remote (A to B)	
	Initial	Average	Initial	Average
Adaptation	3.251	3.295	2.880	3.749
Latency	0.444	0.448	1.013	0.674
Total	3.695	3.743	3.893	4.423

Table 3: Move operation duration (in seconds).

	Local (B to B)		Remote (A to B)	
	Initial	Average	Initial	Average
Adaptation	6.796	3.311	6.105	4.005
Latency	0.577	0.347	0.772	0.692
Total	7.374	3.658	6.877	4.697

Table 4: Average Adaptation Process duration (seconds).

	Local (B to B)	Remote (A to B)
QAAP (1 replace)	15.748	16.285
TDDAP (2 swaps)	14.971	14.226

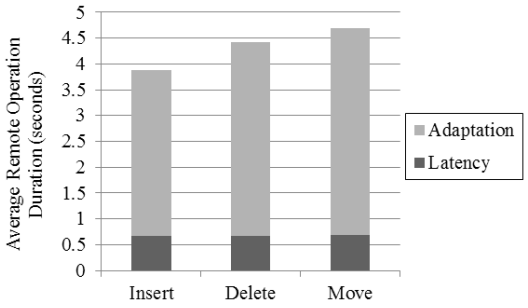

Fig. 24. Average duration (in seconds) for various remote basic adaptation operations.

Table 4 shows average of five repeated execution durations for the QAAP and separately for the TDDAP. For a self-adaptive process, Fig. 23 combines the QAAP and TDDAP workflow fragments before the WP fragment. When executed three times in a local configuration, the average duration was 34.321 seconds. This corresponds closely with the sum of the separate QAAP and TDDAP measured times. Thus, there appears to be no significant performance benefit to integrating adaptation logic in the target process when using a communication interface. Hence, for the aforementioned benefits of process modularization, separating the adaptation logic from target processes and supporting aspect-oriented processes appears practical.

Performance results show that the adaptation delays are potentially tolerable for non-time-critical situations in dBPM, such as proactive or predictive adaptations. When reactive adaptations to executing processes in the cloud are involved, or when human actors cause adaptations and await responses, the delays may be unsatisfactory. Networking had a relatively minor effect on the overall operation duration. Available RAM limitations likely affected PAIS performance, and different configurations and profiling could provide further performance insights.

In summary, the evaluation demonstrated that the solution concept is technically viable for non-time-critical cloud scenarios and can be practically realized by extending a currently available adaptive PAIS. Furthermore, adaptive process modularization and cloud distribution appears to currently have relatively little performance impact versus the cost of the adaptive workflow operations themselves.

6 Conclusions

AProPro provides a flexible cloud-capable approach for process adaptation. Its feasibility was shown with a realization and case studies in the business domain and engineering domain, involving cloud-based adaptation workflows and measurements. Key adaptation capabilities towards dBPM were shown, including workflow-driven adaptations of workflows, variation points, aspect-oriented adaptations, self-adapting workflows, composability, process governance, and the cloud-based provisioning of adaptation processes with an Adaptations-as-a-Service (AaaS) paradigm. Proactive adaptations were applied in push fashion and pulled via self-adaptation. Measurements show that pursuing cloud-based process distribution and adaptation modularization is likely not detrimental to performance, but that the actual application of process adaptations involves certain latencies.

The advantages of the AProPro adaptations for dBPM could be readily realized and benefit various domains such as healthcare, automotive, etc. For instance, a healthcare process could view allergies as an aspect and utilize an allergy adaptation workflow.

The solution faces issues analogous to those of aspect-oriented approaches, in that it may not be readily clear to process modelers which adaptations or effects may be applied in what order at any given workflow point. Thus, additional PAIS tooling and process simulation should support adaptation management, version and variant management, compatibility checking, and make adaptation effects or conflicts visible to process modelers.

Future work will investigate these issues, and also involves comprehensive adaptation pattern coverage, empirical studies, optimizations, and heterogeneous PAIS testing. To achieve the dBPM vision, further work in the process community includes standardization work on interchangeable concrete process templates, repositories, and AaaS cloud APIs, which could further the provisioning, exchange, and reuse of workflows, especially adaptive workflows such as those of the AProPro approach, thus mitigating hindrances for widely modeling and supporting dBPM adaptation.

Acknowledgments. The author thanks Florian Sorg for his assistance with the implementation and evaluation and Gregor Grambow for his assistance with the concept. This work was supported by AristaFlow and an AWS in Education Grant award.

References

1. Weske, M.: Business process management: concepts, languages, architectures. Springer Science & Business Media (2012)
2. Van Der Aalst, W., Van Hee, K. M.: Workflow management: models, methods, and systems. MIT press (2004)
3. Reichert, M., Weber, B.: Enabling flexibility in process-aware information systems: challenges, methods, technologies. Springer Science & Business Media (2012)
4. Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U., Göser, K.: Architecural principles and components of adaptive process management technology. In: PRIMIUM -

Process Innovation for Enterprise Software, pp. 81-97. Lecture Notes in Informatics (LNI) (P-151). Koellen-Verlag (2009)
5. Russell, N., ter Hofstede, A.H.M., van der Aalst,W.M.P., Mulyar, N.: Workflow Control-Flow Patterns: A Revised View. BPM Center Report BPM-06-22 (2006)
6. Weber, B., Reichert, M., and Rinderle-Ma, S.: Change patterns and change support features – Enhancing flexibility in process-aware information systems. In: Data and Knowledge Engineering, 66, 438–466 (2008)
7. Grambow, G., Oberhauser, R., Reichert, M.: Employing Semantically Driven Adaptation for Amalgamating Software Quality Assurance with Process Management. In Proceedings of the 2nd International Conference on Adaptive and Self-adaptive Systems and Applications, pp. 58-67. IARIA XPS Press (2010)
8. Grambow, G., Oberhauser, R., Reichert, M.: Contextual Injection of Quality Measures into Software Engineering Processes. International Journal on Advances in Software, 4(1 & 2), 76-99 (2011)
9. Grambow, G., Oberhauser, R., Reichert, M.: Event-driven Exception Handling for Software Engineering Processes. Proc. 5th International Workshop on event-driven Business Process Management, pp. 414-426. LNBIP 99, Springer-Verlag Berlin Heidelberg (2011)
10. Grambow, G., Oberhauser, R., Reichert, M.: User-centric Abstraction of Workflow Logic Applied to Software Engineering Processes. Proceedings of the 1st Workshop on Human-Centric Process-Aware Information Systems, pp. 307-321. LNBIP 112, Springer-Verlag Berlin Heidelberg (2012)
11. Oberhauser, R.: Adapting Processes via Adaptation Processes - A Flexible and Cloud-Capable Adaptation Approach for Dynamic Business Process Management. In Proceedings of the Fifth International Symposium on Business Modeling and Software Design, ISBN 978-989-758-111-3, pp. 9-18. SCITEPRESS. DOI: 10.5220/0005885000090018 (2015)
12. Rinderle, S., Reichert, M., Dadam, P.: Correctness criteria for dynamic changes in workflow systems--a survey. Data & Knowledge Engineering. 50(1), 9-34 (2004)
13. Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: Declare: Full support for loosely-structured processes. Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07), pp. 287-298. IEEE CPS (2007)
14. Van der Aalst, W. M., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. Data & Knowledge Engineering, 53(2), 129-162 (2005)
15. de Man, H.: Case management: A review of modeling approaches. BPTrends (January 2009)
16. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In: Case-based reasoning: Research and development, pp. 421-435. Springer Berlin Heidelberg (2010).
17. Müller, R., Greiner, U., Rahm, E.: Agentwork - a workflow system supporting rule-based workflow adaptation. Data & Knowledge Engineering, 51(2), 223-256 (2004)
18. Burmeister, B., Arnold, M., Copaciu, F., Rimassa, G.: BDI-agents for agile goal-oriented business processes. In Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track, pp. 37-44. International Foundation for Autonomous Agents and Multiagent Systems (2008)
19. Object Management Group: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (2011)
20. Charfi, A., Mezini, M.: Ao4bpel: An aspect-oriented extension to bpel. World Wide Web, 10(3), 309-344 (2007)
21. Charfi, A., Müller, H., Mezini, M.: Aspect-oriented business process modeling with AO4BPMN. In: Modelling Foundations and Applications, pp. 48-61. Springer-Verlag Berlin Heidelberg (2010)

22. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the Provop approach. Journal of Software Maintenance and Evolution: Research and Practice, 22(6‐7), 519-546 (2010)
23. Döhring, M., Zimmermann, B.: vBPMN: event-aware workflow variants by weaving BPMN2 and business rules. In Enterprise, Business-Process and Information Systems Modeling, pp. 332-341. Springer Berlin Heidelberg (2011)
24. Milanovic, M., Gasevic, D., Rocha, L.: Modeling flexible business processes with business rule patterns. In: Proceedings of the 15th Enterprise Distributed Object Computing Conference (EDOC'11), pp. 65–74. IEEE (2011)
25. Marrella, A., Mecella, M., Sardina, S.: SmartPM: An Adaptive Process Management System through Situation Calculus, IndiGolog, and Classical Planning. Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2014). AAAI Press (2014)
26. La Rosa, M., Wohed, P., Mendling, J., Ter Hofstede, A. H., Reijers, H. A., van der Aalst, W. M.: Managing process model complexity via abstract syntax modifications. IEEE Transactions on Industrial Informatics, 7(4), 614-629 (2011)
27. La Rosa, M., Reijers, H. A., Van Der Aalst, W. M., Dijkman, R. M., Mendling, J., Dumas, M., Garcia-Banuelos, L.: APROMORE: An advanced process model repository. Expert Systems with Applications, 38(6), 7029-7040 (2011)
28. Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J., Weber, B.: Understanding Declare models: strategies, pitfalls, empirical results. Software & Systems Modeling, 1-28 (2014)
29. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H. A.: Imperative versus declarative process modeling languages: An empirical investigation. In: Business Process Management Workshops, pp. 383-394. Springer Berlin Heidelberg (2012)
30. Reijers, H. A., Mendling, J., Dijkman, R. M.: Human and automatic modularizations of process models to enhance their comprehension. Information Systems, 36(5), 881-897 (2011)
31. Döhring, M., Reijers, H. A., Smirnov, S.: Configuration vs. adaptation for business process variant maintenance: an empirical study. Information Systems, 39, 108-133 (2014)
32. Object Management Group: BPMN 2.0 by Example. Object Management Group (2010)
33. Royce, W.: Managing the Development of Large Software Systems. In: Proceedings of IEEE WESCON, vol. 26, No. 8, pp. 328-388. (1970)
34. Beck, K.: Test-Driven Development by Example. Addison-Wesley Professional (2003)