Adapting Processes via Adaptation Processes:
A Flexible and Cloud-Capable Adaptation Approach for Dynamic Business Process Management

Roy Oberhauser
Computer Science Department, Aalen University, Aalen, Germany
roy.oberhauser@htw-aalen.de

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.

Abstract: Dynamic business process management (dBPM) is dependent on automated adaptation techniques. While various approaches to support process adaptation have been explored, they typically involve another or some combination of modeling paradigms or language extensions. Moreover, cross-cutting concerns and a distributed and cloud-based process adaptation capability have not been adequately addressed. This paper introduces AProPro (Adapting Processes via Processes), a flexible and cloud-capable approach towards dBPM that supports adapting target processes using adaptation processes while retaining an intuitive and consistent imperative process paradigm. Based on a case study using a REST-based Web Service prototype realization invoking process adaptation patterns in a distributed of Adaptation-as-a-Service cloud setting, the initial evaluation results show the feasibility of the approach and gauge its performance in the cloud.

1 INTRODUCTION

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), workflow management system (WfMS), or process-aware information system (PAIS).

However, many PAISs today lack dynamic runtime adaptation with correctness and soundness guarantees (Reichert et al., 2009). And when such adaptation is supported, it is typically limited to support for manual change interaction by a process actor (Reichert & Weber, 2012). Typical types of recurring modifications to workflows are known as workflow control-flow patterns (Russell, van der Aalst & ter Hofstede, 2006), change patterns (Weber, Reichert & Rinderle-Ma, 2008) or adaptation patterns (Reichert & Weber, 2012).

In light of the dBPM vision, as the degree of automation and workflow usage increases, there is a corresponding need to support adaptation by agents, be they human or software. Our previous work on an adaptable context-aware and semantically-enhanced PAIS in the software engineering domain (Grambow, Oberhauser & Reichert, 2010, 2011a, 2011b) included automated adaptation work and work on supporting the user-centric intentional adaptation of workflows (Grambow et al., 2012). However, an open challenge remains towards practically expressing and maintaining adaptations in an intuitive manner for process modelers and process actors for a sustainable dBPM lifecycle.

This paper introduces and contributes a practical and flexible cloud-capable approach called Adapting Processes via adaptation Processes (AProPro) for supporting dBPM in a generalized way that can be readily implemented and integrated with current adaptive PAIS technology. It supports the ease and accessibility of process adaptations in an intuitive imperative PAIS paradigm for process modelers and process actors or users. It can further the maintenance, reuse, portability, and sharing of adaptations, including cloud-based provisioning of adaptation processes within the community, thus supporting sustainable adaptability by extending an adaptation process's lifecycle. A case-study demonstrates its feasibility and its cloud-based performance.

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

2 RELATED WORK

Various approaches exist that can support the manual or automated adaption of workflows. The survey by (Rinderle, Reichert & Dadam, 2004) provides an overview from the perspective of support for correctness criteria. (Weber, Reichert & Rinderle-Ma, 2008) provide an overview based on the perspective of change patterns and support features.

Declarative approaches, such as DECLARE (Pesic, Schonenberg & van der Aalst, 2007) support the constraint-based composition, execution, and adaptation of workflows. *Case handling approaches*, such as FLOWer (Van der Aalst, Weske & Grünbauer, 2005), typically attempt to anticipate change. They utilize a case metaphor rather than require process changes, deemphasize activities, and are data-driven (Reichert & Weber, 2012) (Weske, 2012). (de Man, 2009) provides a review of case modeling approaches. Case-based approaches towards adapting workflows include (Minor, Bergmann, Görg & Walter, 2010). *Agent-based approaches* support automated process adaptations applied by autonomous software agents. Agentwork (Müller, Greiner & Rahm, 2004) applies predefined change operations to process instances using rules. (Burmeister, Arnold, Copaciu & Rimassa, 2008) applies a belief-desire-intention (BDI) agent using a goal-oriented BPMN modeling language extension. *Aspect-oriented approaches* include AO4BPEL (Charfi & Mezini, 2007) and AO4BPMN (Charfi, Müller & Mezini, 2010), both of which require language extensions. *Variant approaches* include: Provop (Hallerbach, Bauer & Reichert, 2010), which supports schema variants with pre-configured adaptations to a base process schema; and vBPMN (Döhring & Zimmermann, 2011) that extends BPMN with fragment-based adaptations via the R2ML rule language. rBPMN (Milanovic, Gasevic & Rocha, 2011) also interweaves BPMN and R2ML. *Automated planning* and *exception-driven adaptation approaches* include SmartPM (Marrella, Mecella & Sardina, 2014), 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, no language extensions or other paradigms such as rules, declarative elements, or intelligent agents are required. Further, distributed and cloud-based adaptations to either instances or schemas are supported.

3 SOLUTION APPROACH

The AProPro approach follows an imperative style, and process models are kept as simple and modular as reasonable for typical usage scenarios. This is in alignment with the orthogonal modularity pattern (La Rosa, Wohed, et al., 2011). Special cases can either be separated out or handled as adaptations via adaptation processes. 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. 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 paper.

The following description of the solution approach will highlight certain perspectives. As shown in Figure 1, *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 (Reichert & Weber, 2012, Ch. 15).

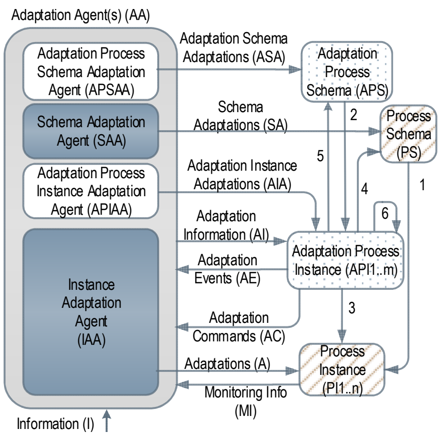
Adaptation Agent(s) (AA)
Adaptation Process Schema Adaptation Agent (APSAA)
Schema Adaptation Agent (SAA)
Adaptation Process Instance Adaptation Agent (APIAA)
Instance Adaptation Agent (IAA)
Adaptation Schema Adaptations (ASA)
Adaptation Process Schema (APS)
Schema Adaptations (SA)
Process Schema (PS)
Adaptation Instance Adaptations (AIA)
Adaptation Information (AI)
Adaptation Events (AE)
Adaptation Process Instance (API1..m)
Adaptation Commands (AC)
Adaptations (A)
Process Instance (PI1..n)
Monitoring Info (MI)
Information (I)
1
2
3
4
5
6

Figure 1: Conceptual solution architecture.

Workflow-driven adaptations of workflows: adaptations, such as adaptation patterns, are specified in the form of workflows that will operate on another workflow. For this (see Figure 1 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: for push, the adaptive workflow is triggered first and applies its changes to the target; for pull, 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 (La Rosa, Reijers, et al., 2011).

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.

Thus, 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 process paradigm and modeling language without requiring language extensions. Empirical findings that support such an approach includes: (Haisjackl et al., 2014) who empirically investigated understandability issues with declarative modeling, and found that subjects tended to model sequentially and had difficulty with combinations of constraints; (Pichler et al., 2012) determined that imperative models have better understandability and comprehensibility than declarative ones; (Reijers, Mendling, & Dijkman, 2011) suggests that process modularity via information hiding enhances understandability; and (Döhring, Reijers & Smirnov, 2014), 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 activity nodes 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 on demonstrating key AProPro and AaaS capabilities 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.

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 Figure 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.

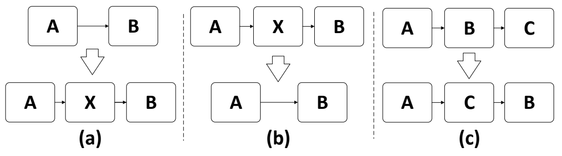

Figure 2: (a) insert, (b) delete, and (c) move process fragment patterns.

The *delete process fragment pattern*, shown in Figure 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 Figure 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 (see Figure 6).

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 the interface for more advanced AaaS services that invoke adaptation workflows which modify a separate target workflow instance. These processes are explained later in the evaluation section, the technical interface parameters of the adaptation service implementations are given here:

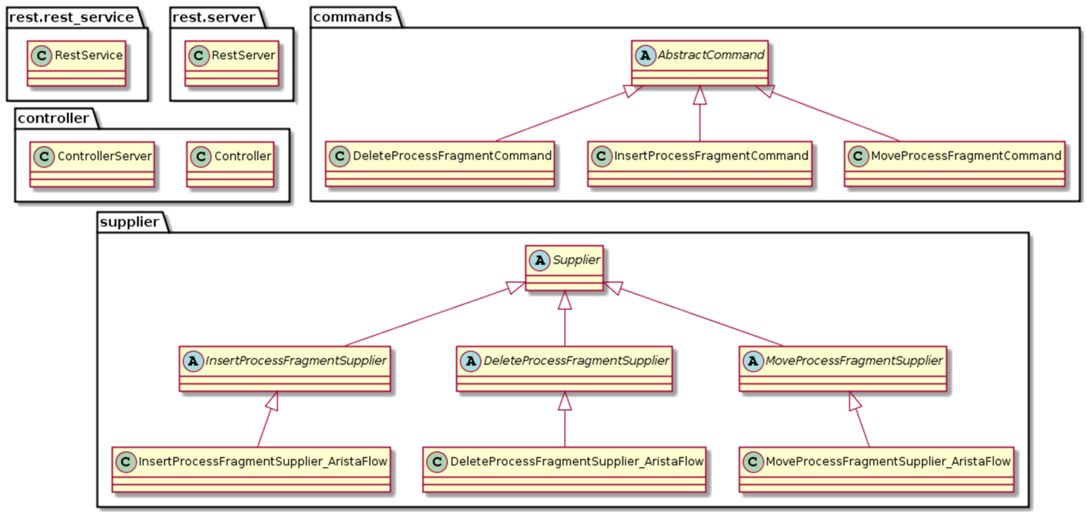
rest.rest_service
RestService
rest.server
RestServer
commands
AbstractCommand
DeleteProcessFragmentCommand
InsertProcessFragmentCommand
MoveProcessFragmentCommand
controller
ControllerServer
Controller
supplier
Supplier
InsertProcessFragmentSupplier
DeleteProcessFragmentSupplier
MoveProcessFragmentSupplier
InsertProcessFragmentSupplier_AristaFlow
DeleteProcessFragmentSupplier_AristaFlow
MoveProcessFragmentSupplier_AristaFlow

Figure 3: UML2 class diagram showing key implementation packages and classes.

- 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. 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. Refer to Figure 3. To support heterogeneity, the commands utilize the corresponding *Supplier* classes which utilize PAIS-specific APIs for the operations.

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

To evaluate the solution concept and one realization thereof, this initial case study focused on demonstrating key process adaptation capabilities of the concept and assessing its viability with respect to performance, especially for a distributed cloud scenario. 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. Workflows operating across geographically separate PAISs utilizing a basic cloud configuration would represent a worst case area of the performance spectrum.

Figure 4 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 local configuration implies that the Adaptation Process is collocated with the Target Process within the same PAIS, but commands are still sent via REST.

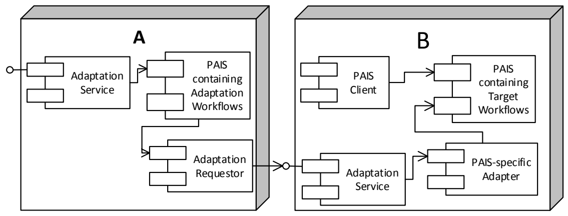

Figure 4: AWS cloud evaluation setup.

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.

5.1 Case Study

While the solution concept is domain independent, this case study uses software engineering (SE) processes to illustrate the capabilities and adaptation effects. The branches and loops involved in realistic models are omitted for simplification and space.

5.1.1 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 Figure 7.

5.1.2 Quality Assurance Adaptation Process

To demonstrate process governance and an Adaptation Process producing process variants, the 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 Figure 8: 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.

Figure 10 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.

5.1.3 TDD Adaptation Process

In software test-driven development (TDD), test preparation activities precede corresponding development activities. To support the TDD aspect in the WP, Unit Test is placed before Implement and Integration Test before Integrate. Thus, the TDD Adaptation Process (TDDAP) shown in Figure 9 utilizes the Move Process Fragment Pattern twice. The resulting adaptations are shown in Figure 11.

5.1.4 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 Figure 12 both QAAP and TDDAP were applied to the target WP, with each Adaptation Process representing a different aspect (reviews or testing).

5.1.5 Self-adaptive Processes

Self-adaptive processes support the integrative modeling of possible adaptations into the target processes themselves. As shown in Figure 13, 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.

5.2 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 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, no optimizations or 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 4 repeated 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 Figure 5.

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

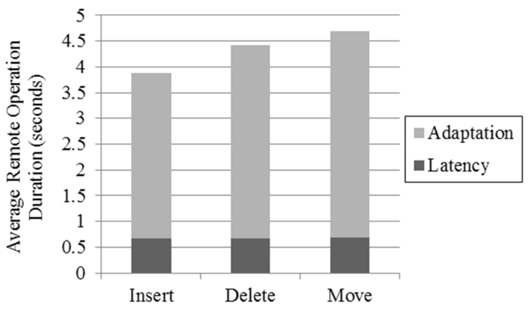

Figure 5: Average duration (in seconds) for various remote basic adaptation operations.

Table 4 shows average of 5 repeated execution durations for the QAAP and separately for the TDDAP.

For a self-adaptive process, Figure 13 combines the QAAP and TDDAP workflow fragments before the WP fragment. When executed 3 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. Thus, 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 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 may have limited PAIS performance, and different configurations and profiling could provide further 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. Further, adaptive process modularization and cloud distribution appears to currently have relatively little performance impact versus the cost of adaptive workflow operations. This could however change if optimization and performance issues are addressed.

6 CONCLUSIONS

A flexible cloud-capable approach for process adaptation called AProPro was introduced. Its feasibility was shown with a realization and a case study involving cloud-based adaptation workflows and measurements. Key adaptation capabilities towards dBPM were shown, including workflow-driven adaptations of workflows, 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 distribution and adaptation modularization is likely not detrimental to performance, since adaptations had more impact.

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 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.

ACKNOWLEDGEMENTS

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

Burmeister, B., Arnold, M., Copaciu, F. & Rimassa, G. (2008). BDI-agents for agile goal-oriented business processes. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track* . International Foundation for Autonomous Agents and Multiagent Systems, 37-44.

Charfi, A. & Mezini, M. (2007). Ao4bpel: An aspect-oriented extension to bpel. *World Wide Web*, 10(3), 309-344.

Charfi, A., Müller, H. & Mezini, M. (2010). Aspect-oriented business process modeling with AO4BPMN. In *Modelling Foundations and Applications* (pp. 48-61). Springer Berlin Heidelberg.

de Man, H. (2009, January). Case management: A review of modeling approaches. *BPTrends*.

Döhring, M., Reijers, H. A. & Smirnov, S. (2014). Configuration vs. adaptation for business process variant maintenance: an empirical study. *Information Systems*, 39, 108-133.

Döhring, M. & Zimmermann, B. (2011). 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.

Grambow, G., Oberhauser, R. & Reichert, M. (2010). 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.

Grambow, G., Oberhauser, R. & Reichert, M. (2011a). Contextual Injection of Quality Measures into Software Engineering Processes. *International Journal on Advances in Software*, 4(1 & 2), 76-99.

Grambow, G., Oberhauser, R. & Reichert, M. (2011b). Event-driven Exception Handling for Software Engineering Processes. *Proc. 5th International Workshop on event-driven Business Process Management*, LNBIP 99, 414-426.

Grambow, G., Oberhauser, R. & Reichert, M. (2012). User-centric Abstraction of Workflow Logic Applied to Software Engineering Processes. *Proceedings of the 1st Workshop on Human-Centric Process-Aware Information Systems*, LNBIP112, 307-321.

Haisjackl, C., Barba, I., Zugal, S., Soffer, P., Hadar, I., Reichert, M., Pinggera, J. & Weber, B. (2014). Understanding Declare models: strategies, pitfalls, empirical results. *Software & Systems Modeling*, 1-28.

Hallerbach, A., Bauer, T. & Reichert, M. (2010). Capturing variability in business process models: the Provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(6-7), 519-546.

La Rosa, M., Reijers, H. A., Van Der Aalst, W. M., Dijkman, R. M., Mendling, J., Dumas, M. & Garcia-Banuelos, L. (2011). APROMORE: An advanced process model repository. *Expert Systems with Applications*, 38(6), 7029-7040.

La Rosa, M., Wohed, P., Mendling, J., Ter Hofstede, A. H., Reijers, H. A. & van der Aalst, W. M. (2011). Managing process model complexity via abstract syntax modifications. *IEEE Transactions on Industrial Informatics,* 7(4), 614-629.

Marrella, A., Mecella, M. & Sardina, S. (2014). 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.

Milanovic, M., Gasevic, D. & Rocha, L. (2011). Modeling flexible business processes with business rule patterns. In: *Proceedings of the 15th Enterprise Distributed Object Computing Conference (EDOC'11)* (pp. 65–74). IEEE.

Minor, M., Bergmann, R., Görg, S. & Walter, K. (2010). Towards case-based adaptation of workflows. In *Case-based reasoning. Research and development* (pp. 421-435). Springer Berlin Heidelberg.

Müller, R., Greiner, U. & Rahm, E. (2004). Agentwork: a workflow system supporting rule-based workflow adaptation. *Data & Knowledge Engineering*, 51(2), 223-256.

Pesic, M., Schonenberg, H. & van der Aalst, W.M.P. (2007). Declare: Full support for loosely-structured processes. *Proceedings of the 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC'07)* (pp. 287-298). IEEE.

Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J. & Reijers, H. A. (2012). Imperative versus declarative process modeling languages: An empirical investigation. In *Business Process Management Workshops*. Springer Berlin Heidelberg., 383-394.

Reichert, M., Dadam, P., Rinderle-Ma, S., Jurisch, M., Kreher, U. & Göser, K. (2009). Architecural principles and components of adaptive process management technology. In: *PRIMIUM - Process Innovation for Enterprise Software*. Lecture Notes in Informatics (LNI) (P-151). Koellen-Verlag, 81-97.

Reichert, M. & Weber, B. (2012). *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer Science & Business Media.

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

Rinderle, S., Reichert, M. & Dadam, P. (2004). Correctness criteria for dynamic changes in workflow systems--a survey. *Data & Knowledge Engineering*. 50(1). pp. 9-34.

Russell, N., ter Hofstede, A.H.M., van der Aalst,W.M.P. & Mulyar, N. (2006). Workflow Control-Flow Patterns: A Revised View. *BPM Center Report* BPM-06-22.

Van der Aalst, W. M., Weske, M. & Grünbauer, D. (2005). Case handling: a new paradigm for business process support. *Data & Knowledge Engineering*, 53(2), 129-162.

Weber, B., Reichert, M., and Rinderle-Ma, S. (2008). Change patterns and change support features – Enhancing flexibility in process-aware information systems. In: *Data and Knowledge Engineering,* 66, 438–466.

Weske, M. (2012). Business process management: concepts, languages, architectures. Springer Science & Business Media.

APPENDIX

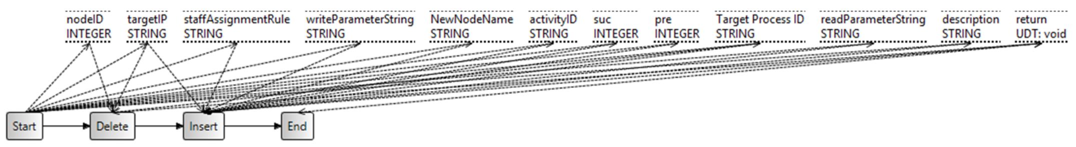

Figure 6: The Replace process fragment sub-process.

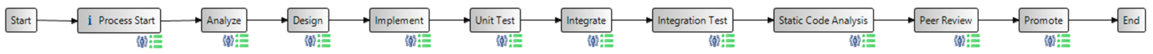

Figure 7: The unchanged Waterfall Process (WP).

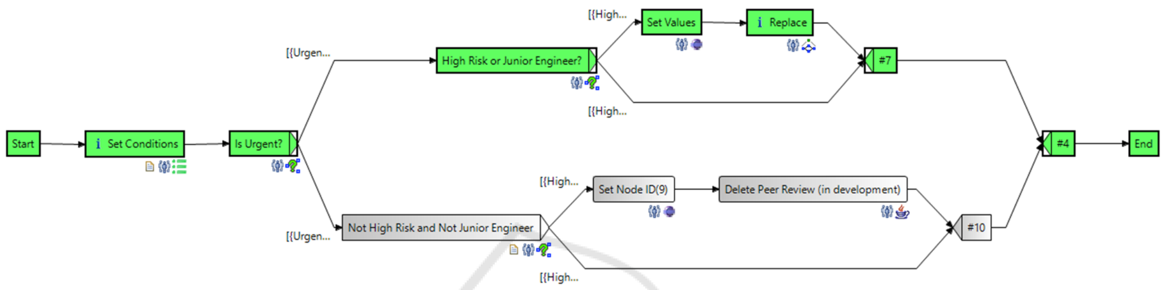

Figure 8: Quality Assurance Adaptation Process (QAAP).

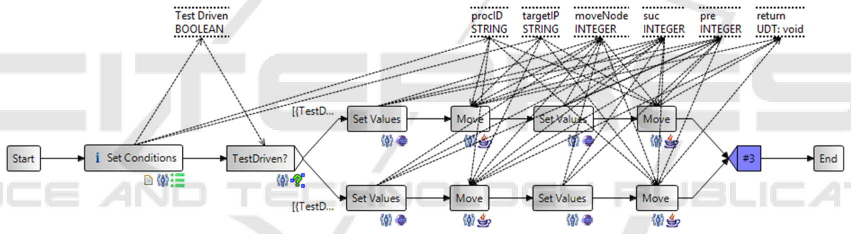

Figure 9: Test-Driven Development Adaptation Process (TDDAP).

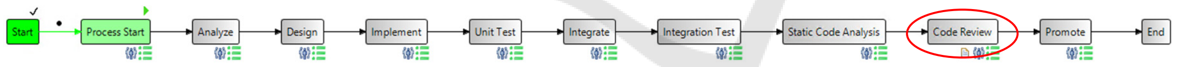

Figure 10: Waterfall Process after application of the Quality Assurance Adaptation Process.

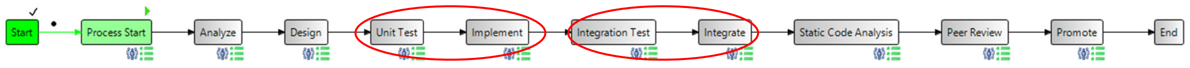

Figure 11: Waterfall Process after application of the Test-Driven Development Adaptation Process.

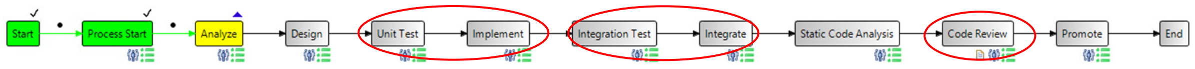

Figure 12: Waterfall Process after application of both Adaptation Processes.

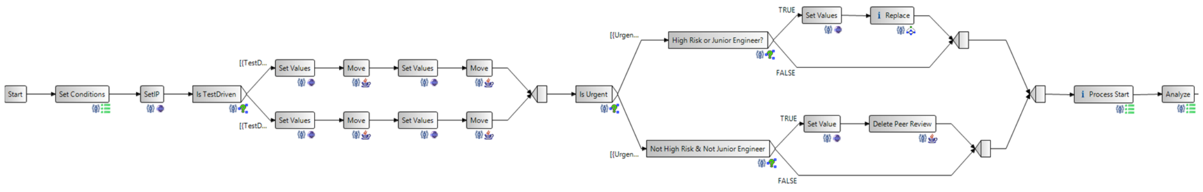

Figure 13: Self-adaptive Waterfall Process screenshot (continues to right as in Figure 7).