A Hypermedia-Driven Approach for Adapting Processes via Adaptation Processes

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

***Abstract*—In addition to an increasing need for business process management systems (BPMS) to dynamically adapt to situational change, there is increased interest in web service accessibility of BPMS to broaden their integration in enterprises and the cloud. Towards this end, a RESTful hypermedia-driven enactment and adaptation of processes has not been adequately explored. This paper thus investigates a hypermedia extension of our process adaptation approach AProPro we call AProProh (Adapting Processes via Processes using hypermedia) that dynamically provides hypermedia to guide process clients in the navigation, enactment, and adaptation of process instances. Clients of process-aware information systems (PAIS) have hitherto been tied to vendor-specific APIs and lacked a standard web-based API. The AProProh HATEOAS-PAIS middleware enables process clients to be more generic and PAIS-agnostic via REST APIs. Additionally, process clients can dynamically apply valid adaptations or adjust to process model changes utilizing dynamically generated hypermedia. Based on a case study prototype realization supporting dynamic invocation of process adaptation patterns, the initial evaluation results show its feasibility and gauge its performance overhead.**

Keywords-Adaptive process-aware information systems; PAIS; adaptive workflow management systems; WfMS; RESTful web services; hypermedia; HATEOAS; business process management systems; BPMS; dynamic business process management; aspect-oriented processes; change patterns

I. INTRODUCTION

Software-based automation is affecting all major industries. Correspondingly, business processes are increasingly being modeled and enacted in business process management systems (BPMS), also known as process-aware information systems (PAIS) or workflow management systems (WfMS). These executable processes (workflows) are increasingly required to dynamically adapt, known as dynamic business process management (dBPM). Yet currently available PAIS rarely support correctness and soundness guarantees for such runtime adaptation [1], and that rare subset typically only focuses on manual changes by a human actor [2]. Recurring workflow modifications are known as workflow control-flow patterns, change patterns, or adaptation patterns [2].

With the increasing popularity of BPMS and the criticality of these processes for businesses, there is a corresponding desire for vendor independence. While integration of software systems is increasingly cloud-based utilizing web service APIs, heterogeneous PAIS integration has hitherto been impeded by a lack of standardization and accessibility. Furthermore, Hypermedia as the Engine of Application State (HATEOAS), a constraint of the Representational State Transfer (REST) application architecture [3], supports dynamic navigation of REST APIs by a REST client based on hypermedia knowledge. REST can be seen is in some ways as a type of runtime workflow. Its application for the navigability and adaptation of processes in currently available adaptive PAIS has not been adequately explored.

Our previous work called AProPro (Adapting Processes via Processes) [4] introduced a cloud-based approach for practically expressing and maintaining adaptations in an intuitive and sustainable manner for the dBPM lifecycle. Extending this work, this paper contributes an investigation into supporting hypermedia-driven dynamic resource-oriented navigation and process-based adaptation of process instances. We call this extension and HATEOAS-PAIS middleware AProProh (Adapting Processes via Processes using hypermedia). REST clients can thus be PAIS agnostic and, utilizing the dynamically supplied hypermedia links, automatically navigate process instances, applying adaptations valid to its state. The evaluation based on a case study using a prototype shows its viability and provides performance measurements.

The paper is organized as follows: Section 2 describes related work, which is followed by the solution approach. In Section 4, the realization is described. This is followed by an evaluation and then the conclusion. Since this paper focuses on the technical implementation of a process, the term process is understood to mean a technically enactable process, and the term workflow can be used interchangeably.

II. RELATED WORK

With regard to the basis AProPro, various approaches support the manual or automated adaptation of processes. Declarative approaches, such as DECLARE [5] support the constraint-based composition, execution, and adaptation of workflows. Case handling approaches [6] utilize a case metaphor, deemphasize activities, and are data-driven [2]. Agent-based approaches include Agentwork [7], which applies predefined change operations using rules, and [8], where a belief-desire-intention agent utilizes a goal-oriented BPMN modeling language extension. Aspect-oriented approaches include AO4BPMN

[9], which requires a language extension. Variant approaches include: Provop [10], which supports schema variants with pre-configured adaptations to a base process schema; and vBPMN [11], which extends BPMN with fragment-based adaptations via the R2ML rule language. rBPMN [12] also interweaves BPMN and R2ML. Automated planning and exception-driven adaptation approaches include SmartPM [13], which utilizes artificial intelligence, procedural, and declarative elements. AProPro differs by being an imperative process-based adaptation approach without a case metaphor and being activity-, service-, and process-centric with regard to runtime adaptation. Language extensions or paradigms such as rules, declarative elements, or intelligent agents are not required.

Work related to the hypermedia-driven extension AProProh includes HATEOAS in combination with BPMS or PAIS. [14] involves combining RESTful with BPM, but does not mention of HATEOAS constraints or hypermedia, nor are changes to process models undertaken or performance impacts discussed. RESTful and BPEL work includes [15], which focused primarily on the composition and integration of services. Similarly, BPMashup [16] focuses on process-centric compositions of RESTful web services without addressing hypermedia. RESTfulBP [17] and CPEE [18] do not mention HATEOAS or hypermedia. RESTful with BPMN [19] extends BPMN notation with graphical syntax and semantics for REST support, but without directly addressing hypermedia or HATEOAS. To our knowledge, no commercial BPMS vendors or open source BPMS currently explicitly market or support the capability, although the idea has been pondered in blog and forum posts.

III. SOLUTION

Since the AProProh extension is based on AProPro, we will first briefly describe the AProPro approach.

A. The AProPro solution approach

This description is based on [4]. AProPro uses an imperative process style, and process models are kept as simple and modular as reasonable in alignment with the orthogonal modularity pattern [20]. To reduce model complexity, special cases can either be separated out or handled as adaptations via adaptation processes. A guiding principle is that adaptations to processes should themselves be modeled as processes, remaining consistent with the process paradigm and mindset.

As shown in Fig. 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), utilizing Information (I) (e.g., context or system information such as planning heuristics) or Monitoring Information (MI), trigger modifications to various process structures. A Schema Adaptation Agent (SAA) makes Schema Adaptions (SA) to one or more Process Schema (PS). An Instance Adaptation Agent (IAA) may perform Adaptations (A) on some Process Instance (PI). Support for such manual adaptations has been available in adaptive PAISs, e.g., the ADEPT2-based AristaFlow [2].

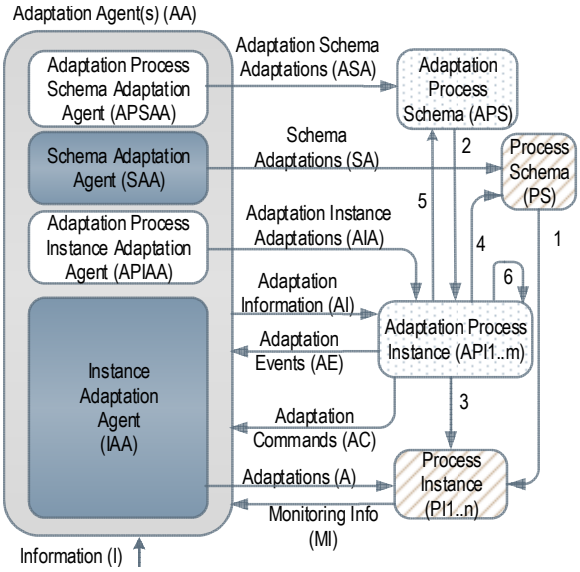


Figure 1. Conceptual solution architecture of AProPro [4].

As previously described in [4], this approach supports process-driven adaptations of processes, adaptation patterns as processes, aspect-oriented adaptations, variation points, adapting adaptation processes, recursive adaptation, exception-based adaptations, reactive and proactive adaptations, push-or-pull adaptations, reusability, composability, process compliance and governance, cloud-based provisioning of adaptation processes, and service-oriented adaptation services.

Process-driven adaptations of processes: adaptations, such as adaptation patterns, are specified in the form of processes that will operate on another process. For this (see Fig. 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).

1) *Adaptation patterns as processes:* Various common adaptation patterns (insert, delete, move, replace, swap, inline, extract, parallelize, etc.) can be modeled a a sequence of actions in an enactable process and applied conditionally based on Adaptation Information (AI), e.g., in the form of process variables or events.

2) *Aspect-oriented adaptations:* Aspect-oriented adaptations are supported by modularizing and constraining an adaptive process to operate on only one aspect (such as authorization). Other adaptation processes can be used to address other aspects. 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.

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

4) *Adapting adaptation processes:* The concept supports a further degree of flexibility by supporting adaptation processes operating on (other) adaptation processes. 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).

5) *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.

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

7) *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.

8) *Push-or-pull adaptations:* for push, the adaptive process is triggered first and applies its changes to the target; for pull, the target process triggers the adaptive process to initiate its adaptations.

9) *Reusability:* shared modeled/tested adaptation processes support the wider reuse of adaptation patterns in the community, e.g., via repositories like APROMORE [21].

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

11) *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.

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

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

In support of dBPM, AProPro enables desired automated or semi-automated adaptations, yet process modelers and users can remain in their current imperative process paradigm and process modeling language without necessitating language extensions. Empirical findings which can be interpreted to support an approach such as ours include: La Rosa et al. [22], an empirical investigation of understandability issues with declarative modeling, finding that subjects tended to model sequentially and had difficulty with combinations of constraints; in Pichler et al. [23], imperative models had better understandability and comprehensibility than declarative ones; Reijers et al. [24] suggests that process modularity via information hiding enhances understandability; and Döhring et al. in [25] found that process complexity affected maintenance task efficiency for process variant construction - subjects hereby preferred high-level change patterns to process configuration.

Further information on the implementation and evaluation of this approach can be found in [4].

B. The AProProh HATEOAS-PAIS integration middleware

AProProh provides a hypermedia extension to the AProPro approach just described.

The AProProh HATEOAS-PAIS integration middleware (shown in Fig. 2) includes a PAIS abstraction layer named pais_supplier. It contains PAIS-specific plugins that abstract the use PAIS-specific APIs in the connection, enactment, adaptation, and possible_actions subpackages.

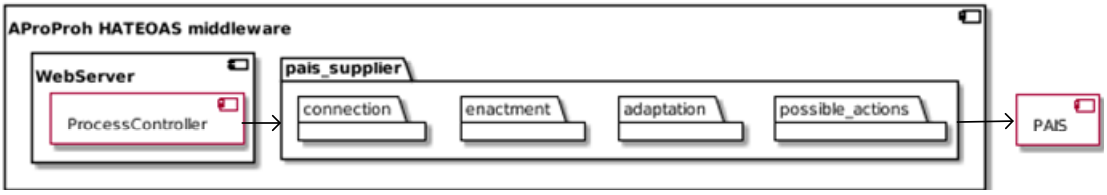


Figure 2. The AProProh HATEOAS-PAIS middleware solution concept.

REST requests to the web server are forwarded to a ProcessController that in turn invokes the pais_supplier functionality.

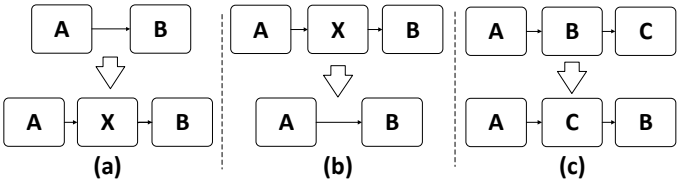
IV. Realization

A prototype was realized based on Spring HATEOAS, Jackson for JSON, Apache Tomcat, and AristaFlow serving as an adaptive PAIS.

The REST requests to the Apache web server are forwarded to a Spring-based ProcessController that invokes the PAIS supplier functionality. For simplicity the implementation provides an immediate response to the POST that only provides a self-reference, rather than creating a separate temporary action task queue for tracking longer-duration invocations. A subsequent GET then determines the new state and possible next actions. Multipart/form-data was in the JSON format.

As previously described in [4], the fundamental insert, delete, and move process fragment patterns were implemented with REST with expected general parameters needed by a PAIS passed as JSON parameters, and the replace pattern was realized as a subprocess that uses the insert and delete patterns. The invocation syntax is shown here, with {command} being one of either "insert", "delete", "move", or "replace".

- `POST /procID/{procID}/{command}`



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

The insert process fragment pattern, shown in Fig. 3a, expects 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.

The delete process fragment pattern, shown in Fig. 3b, 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. 3c, 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 solution approach required no internal changes to this PAIS, relying exclusively on its available extension mechanisms via its generic Java method execution environment. 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 process located anywhere. Only the actual process change operations use a PAIS-specific API. Other PAIS implementations can be integrated relatively easily via plug-in adapters. Adaptation process activity nodes utilize a StaticJavaCall to invoke the AristaFlow extension code contained in a Java ARchive (JAR) file, which sends change requests to our REST server.

Our algorithm to dynamically determine the possible next actions for a GET request is based on the current process state (not applicable to POST) with programmed rules. This typically includes the directly subsequent node(s), adaptations, and a self-reference. However, determining possible adaptations is more involved, since an adaptation process could attempt to change the currently active node. Thus, an adaptation affecting the active node, or if the active node is beyond the nodes adapted by an adaptation process (useless adaptation), then the adaptations are not provided in that interaction. Future work may incorporate a generalized rule engine.

V. EVALUATION

To evaluate the solution concept, this initial case study focused on demonstrating key process navigation and adaptation capabilities and determining if performance degradations are significant, since the server dynamically provides client options. The REST client was Postman extension for the Chrome browser.

A. Case study

The case study explores navigating a running process instance and adapting it via two adaptation processes using only the RESTful interface to our AProProh prototype. Software engineering (SE) processes were concretely used to represent and convey the concepts and validate its practicality, but the approach is domain independent.

Waterfall process (WP). We loosely follow this simple software engineering (SE) process consisting of common SE activities, making minor modifications (modeled in AristaFlow in Fig. 4). A branch was inserted to demonstrate branch navigability.

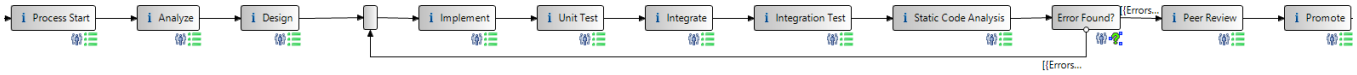


Figure 3. WP with branch. Numbers indicate number of executions (start/end nodes cropped).

Test-driven development adaptation process (TDDAP). TDDAP serves as an example of an aspect-oriented adaptation process. In test-driven development (TDD), test preparation activities precede corresponding software development activities. To support the TDD aspect in the WP, Unit Test is placed before Implement and Integration Test before Integrate utilizing two Moves (see Fig. 5).

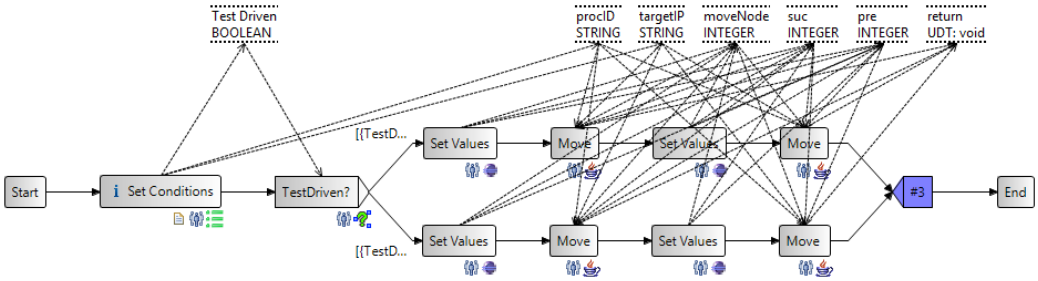


Figure 4. TDDAP (start/end nodes were cropped).

Quality assurance adaptation process (QAAP). This process (see Fig. 6) demonstrates governance and variants, adapting a target process based on situational factors. Since the organizational policy expects a source code peer review before promotion into the source code version control system, WP already includes this activity. However, a "Code Review" is required instead if it is 'NOT urgent AND (high risk OR junior engineer)', hence the "Peer Review" node is deleted and a "Code Review" node is inserted. Only if the situation is 'urgent AND NOT high risk AND NOT junior engineer', then "Peer Review" is deleted. Such contextual parameters could be automatically determined and applied.

If desired, QAAP could also check for compliance and remediate by inserting some missing activity.

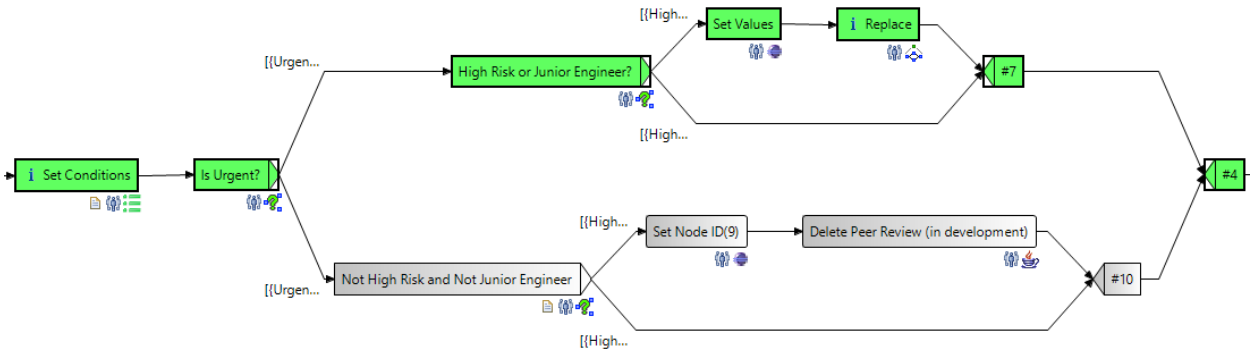


Figure 5. QAAP (start/end nodes were cropped).

Below shows a typical example of the contents of the response body in this case study ("..." indicates additional text not shown):

```
"content": "actions",
"procID": "30442c0f-bf40-4a24-92d5-08f9d1fab508",
"active_node": 3,
"node_state": "NS_ACTIVATED",
"parameters":{
  "High Risk": "BOOLEAN",
  "Junior": "BOOLEAN",
  "Urgent": "BOOLEAN" },
"_links": {...
```

where "content" is the action invoked, "procID" is the process ID, "active_node" is the node number to be executed next, "node_state" is the current state of the node, "parameters" are the output parameters (data elements) the activities depend on, and "_links" are the next possible actions (in a POST response it only contains a self-reference, since a GET is expected thereafter to determine the new state).

For the WP activity "Error Found", the output parameter "Decision" is used to control the return loop to "Unit Test" or continue to "Code Review". The TDDAP POST input parameter included "testDriven" = true. The QAAP POST input parameters included "urgent" and "high risk" as false and "junior engineer" as true.

After instantiating a new process instance that can be done via a POST, Fig. 7 shows the sequence used for navigation and adaptation. Since after each POST response a GET is invoked to determine the new state, this is implied and not shown. After the TDDAP POST and the following execute POST, the GET no longer lists "TDD" as an action, since the currently active node is then "Unit Test". Once "Code Review" is entered, "QA" is no longer listed.

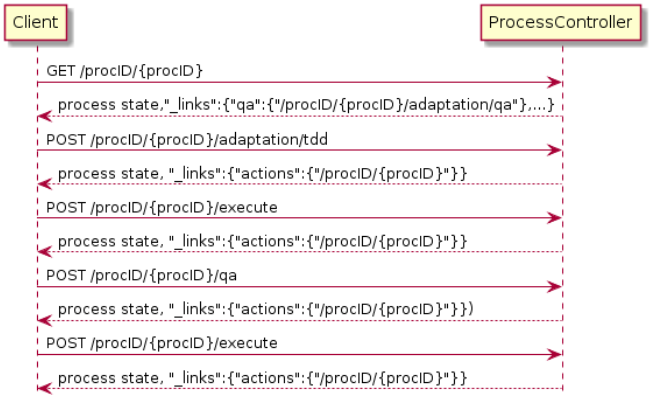


Figure 6. REST client server interaction.

The resulting process state can be seen in Fig. 8, where contextually-driven structural adaptations by the TDDAP invocation caused the unit and integration tests to be swapped with their corresponding activity. The QAAP also demonstrated process governance and compliance, determining that "Peer Review" should be replaced with "Code Review". Moreover, navigation at "Error Found" caused a return loop to "Unit Test" based on the contextual situation transmitted via the "Decision" parameter.

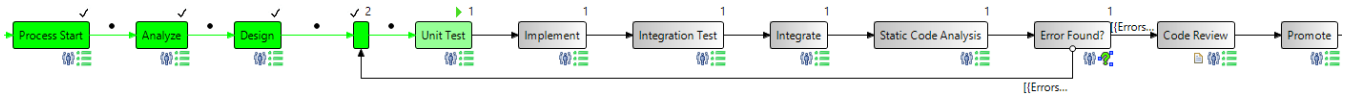


Figure 7. Resulting WP after adaptations applied and looped once (start/end cropped).

The numbers above the activities in Fig. 8 indicate the number of visits to that activity node.

A. *Measurements*

To determine the practical impact of the REST/HATEOAS middleware, the server-side overhead was measured (without network latencies) as shown in Table 1 on a PC with the following configuration without optimization: a i5-4460 CPU at 3.2GHz, 8GB RAM, Win10 Pro, Java JRE 1.8, Apache Tomcat 8, Spring HATEOAS 0.16, Spring plugin core 1.1.0, Jackson 2.4.6, and AristaFlow 1.0.92. For averages, the succeeding four operations after an uninitialized measurement were used.

TABLE I. INITIAL AND AVERAGE LATENCY AND OVERHEAD PER REST COMMAND (IN MILLISECS).

Request	Total latency (ms)		Average overhead	
	Uninitialized	Average	(ms)	%
GET	1656	278	278	100%
POST /execute	1882	371	205	55%
POST /adaptation/tdd	-	318	110	35%
POST /adaptation/qa	-	364	135	37%

The first request in any uninitialized state showed a substantial total latency for both the GET and POST `/execute`. This is primarily due to the need to connect to the PAIS and retrieve an AristaFlow process instance reference. Subsequent operations have access to cached references. Since a GET does not change the state of the PAIS, the latency overhead is primarily for determining the next possible actions. Uninitialized values are not reported for TDDAP and QAAP, as adaptations are typically applied only after initial connection to the target process instance. Average overhead shows the latency resulting from determining the possible actions for a given state and generating the JSON response body. The invocation of an adaptation process involves the complete process enactment time of that process in the PAIS. The average total latency across all four operations was thus 333ms.

While text-based interactions may be easier to integrate and debug, the measurements may be an indication that the verbosity of the text-based REST and JSON and the additional resulting computation of possible actions for the HATEOAS client do add up to a significant performance overhead (greater than 35%). Thus, direct access of PAIS APIs may be a consideration when performance is critical, but at the tradeoff of losing this generic HATEOAS interaction and adaptation middleware to heterogeneous PAIS. As the number of possible actions increases, each interaction is affected.

This overhead could be due to the chosen prototype libraries, the selected web server or its configuration, etc. Due to lack of time, we were unable to probe further into the root causes for the discovered overhead, and plan to do so in future work. Due to the expected GET after a POST to determine the new state, additional overhead is involved.

VI. CONCLUSION

This paper investigated the application of a hypermedia-driven RESTful web service solution approach for enacting and adapting processes in a process-aware information system. While the original AProPro solution approach enables one process to adapt another process dynamically in the cloud utilizing REST services that apply change patterns, AProProh further extends AProPro with a RESTful HATEOAS-PAIS middleware that supports PAIS-agnostic hypermedia-driven process enactment, adaptation, and navigation.

The evaluation consisted of a case study that was based on a sequential process from software engineering domain and two dynamically applied adaptation processes (quality assurance and test-driven development). It demonstrated the viability of enhancing a PAIS with an HATEOAS integration layer without necessitating PAIS modifications. Furthermore, RESTful clients having no prior knowledge of the process model could enact and adapt these models dynamically. In our initial evaluation of the prototype, we determined a significant performance overhead for the HATEOAS-PAIS middleware prototype.

Future work will include a detailed analysis of the performance issues we uncovered and appropriate performance optimization and tuning and providing support for long-duration invocations.

ACKNOWLEDGMENT

The author thanks Florian Sorg for his assistance with the prototype realization, evaluation, and screen shots.

REFERENCES

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

[2] M. Reichert and B. Weber, Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer Science & Business Media, 2012.

[3] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.

[4] R. Oberhauser, "Adapting processes via adaptation processes: a flexible and cloud-capable adaptation approach for dynamic business process management.," in Proceedings of the International Symposium on Business Modeling and Software Design (BMSD 2015), Scitepress, 2015, pp. 9-18, online http://www.is-bmsd.org/Documents/ProceedingsOfFifthBMSD.pdf

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

[6] H. de Man, "Case management: A review of modeling approaches," BPTrends, January 2009.

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

[8] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa, "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, 2008, pp. 37-44.

[9] A. Charfi, H. Müller, and M. Mezini, "Aspect-oriented business process modeling with AO4BPMN," in Modelling Foundations and Applications, Springer Berlin Heidelberg, 2010, pp. 48-61.

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

[11] M. Döhring and B. Zimmermann, "vBPMN: event-aware workflow variants by weaving BPMN2 and business rules," in Enterprise, Business-Process and Information Systems Modeling, Springer Berlin Heidelberg, 2011, pp. 332-341.

[12] M. Milanovic, D. Gasevic, and L. Rocha, "Modeling flexible business processes with business rule patterns," in Proceedings of the 15th Enterprise Distributed Object Computing Conference (EDOC'11), IEEE, 2011, pp. 65–74.

[13] A. Marrella, M. Mecella, and S. Sardina, "SmartPM: an adaptive process management system through situation calculus, IndiGolog, and classical planning," in Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning (KR 2014), AAAI Press, 2014, pp. 1-10.

[14] S. Kumaran, R. Liu, P. Dhoolia, T. Heath, P. Nandi, and F. Pinel, "A restful architecture for service-oriented business process execution," IEEE International Conference on e-Business Engineering, IEEE , 2008, pp. 197-204.

[15] C. Pautasso, "RESTful Web service composition with BPEL for REST," Data & Knowledge Engineering, 68(9), 2009, pp. 851-866.

[16] X. Xu, I. Weber, L. Zhu, Y. Liu, P. Rimba, and Q. Lu, "BPMashup: dynamic execution of RESTful processes," International Conference on Service-Oriented Computing Workshops (ICSOC 2012 Workshops), Springer Berlin Heidelberg, 2013, pp. 447-450.

[17] Q. Lu, X. Xu, W. Zhang, L. Zhu, and S. Li, "Business-driven process fragment selections in RESTful business processes," International Journal of u- and e-Service, Science and Technology, 8(1), 2015, pp. 173-186.

[18] J. Mangler and S. Rinderle-Ma, "CPEE - cloud process execution engine," BPM 2014 Demos, CEUR-WS.org, 2014, online ceur-ws.org/Vol-1295/paper22.pdf .

[19] C. Pautasso, "BPMN for REST," Business Process Model and Notation, Springer Berlin Heidelberg, 2011, pp. 74-87.

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

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

[22] C. Haisjackl, I. Barba, S. Zugal, P. Soffer, I. Hadar, M. Reichert, J. Pinggera, and B. Weber, "Understanding Declare models: strategies, pitfalls, empirical results," Software & Systems Modeling, 2014, pp. 1-28.

[23] P. Pichler, B. Weber, S. Zugal, J. Pinggera, J. Mendling, and H. A. Reijers, "Imperative versus declarative process modeling languages: An empirical investigation," in Business Process Management Workshops, Springer Berlin Heidelberg, 2012, pp. 383-394.

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

[25] M. Döhring, H. A. Reijers, and S. Smirnov, "Configuration vs. adaptation for business process variant maintenance: an empirical study," Information Systems, 39, 2014, pp. 108-133.