International Journal of Software Engineering and Its Applications
Vol. 10, No. 6 (2016), pp. 53-68
http://dx.doi.org/10.14257/ijseia.2016.10.6.05

A Hypermedia Middleware for Process Enactment and Adaptation

Roy Oberhauser

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

Abstract

Dynamic business process management incorporates the capabilities needed to deal with the fast-paced change inherent in today's business processes. While process-aware information systems (PAIS) face an increasing challenge to dynamically adapt running processes to context changes, support for such dynamic adaptation during process enactment remains limited. Furthermore, while the use of web service APIs for enterprise application integration (EAI) and cloud accessibility has garnered broad community support, current PAIS lack a standardized API and often require use of their proprietary clients or APIs. In this paper, we explore the use of hypermedia as the engine of application state (HATEOAS) for process enactment and process adaptation. Our approach Adapting Processes via Processes using hypermedia (AProProh) situates a HATEOAS-based middleware between RESTful clients and heterogeneous PAIS that can dynamically guide a PAIS-agnostic process client in the navigation, enactment, and adaptation of process instances. Dynamically generated hypermedia enables clients to dynamically apply valid adaptations and adjust their process navigation dynamically to a changed process model while supporting long-running operational requests. A case study based on a prototype realization shows the viability of the approach for supporting dynamic process navigation and process adaptation and characterizes its performance.

Keywords*: process-aware information systems; PAIS; business process management systems; BPMS; workflow management systems; WfMS; dynamic business process management; RESTful web services; hypermedia; HATEOAS; enterprise application integration; EAI*

1. Introduction

With the increasing automation in business as seen in trends such as Cyber-physical systems, Internet-of-things [1], and Industry 4.0 [2], the corresponding business processes are being increasingly modeled and enacted in process-aware information systems (PAIS) [3]. As our focus here is only on the executable process, which we shall refer to as the process or workflow, we will not differentiate here between a PAIS, business process management system (BPMS), and a workflow management system (WfMS).

Any business process is typically unique for the organization in which it is utilized, and as such the process modeling and process integration with various information systems typically requires a substantial investment in modeling and administrative effort, costs, and ongoing management and maintenance. With the increasing popularity and criticality of BPMS for businesses, as well as the significant investment in process modeling and integration, a corresponding desire for vendor independence and standard enterprise application integration (EAI) mechanisms is evident. While integration of software systems in general is increasingly cloud-based utilizing web service APIs (application programming interfaces), heterogeneous PAIS integration has hitherto been impeded by a

ISSN: 1738-9984 IJSEIA

lack of standardization and accessibility. Although apparent standards such as BPMN [4] for process modeling notation and BPEL [5] as a process execution language seem common, nevertheless interchange issues exist [6]. Actual internal process model formats and execution languages, which often preceded the standards, are often PAIS-specific and thus exhibit some differences and further possibilities. Moreover, a standardized API to simplify the integration of PAIS from various vendors is not evident or widely supported. As an alternative for such integration, hypermedia as the engine of application state (HATEOAS), a constraint of the Representational State Transfer (REST) application architecture [7], supports dynamic navigation of REST APIs by a REST client based on hypermedia knowledge. REST can be seen in some ways as a type of runtime workflow, yet the application of HATEOAS for EAI in currently available adaptive PAIS for process navigation and adaptation has not been adequately explored.

Since the business environment has become much more dynamic, there is increasing pressure for the capability to dynamically and to some extent automatically adapt processes for changing contextual conditions, also known as dynamic business process management (dBPM). However, currently available PAIS rarely support correctness and soundness guarantees for runtime process adaptation [8], and when then the focus is typically on supporting manual process changes by a human actor [9]. Any recurring process modifications are known as workflow control-flow patterns, change patterns, or adaptation patterns [9]. dBPM will necessitate automated process adaptation, and these will need a viable process schema and a process enactment adaptation capability as well as a way to model the adaptations.

To address both the dBPM and the integration challenge, this paper, a revised and expanded version of [10], elaborates an approach and hypermedia middleware for dynamic process enactment and adaptation called Adapting Processes via Processes using hypermedia (AProProh). It is a hypermedia extension of the original AProPro (Adapting Processes via Processes) [11], which models and enacts adaptations imperatively, practically expressing and maintaining adaptations in an intuitive and sustainable manner for the dBPM lifecycle. Towards a standardized web API mechanism for PAIS integration, this HATEOAS-PAIS middleware includes support for long-running activities, processes, or process adaptations. The evaluation is based on a case study that shows its viability with a prototype and assesses the middleware's performance characteristics with different REST implementations.

The paper is organized as follows: Section 2 discusses related work, which is followed by a description of the solution approach. In Section 4, details of the prototype realization are provided. An evaluation based on a case study is given in Section 5, followed by the conclusion.

2. Related Work

As to process adaptation, various approaches support the manual or automated adaptation of processes. Case handling approaches [12] utilize a case metaphor, deemphasize activities, and are data-driven [9]. Declarative approaches, such as DECLARE [13] support the constraint-based composition, execution, and adaptation of workflows. Agent-based approaches include Agentwork [14], which applies predefined change operations using rules, and [15], where a belief-desire-intention agent utilizes a goal-oriented BPMN modeling language extension. Aspect-oriented approaches include AO4BPMN [16], which requires a language extension. Variant approaches include: Provop [17], which supports schema variants with pre-configured adaptations to a base process schema; and vBPMN [18], which extends BPMN with fragment-based adaptations via the R2ML rule language. rBPMN [19] also interweaves BPMN and R2ML. Automated planning and exception-driven adaptation approaches include SmartPM [20], which utilizes artificial intelligence, procedural, and declarative elements.

In contrast, the AProPro adaptation approach supports runtime adaptation and is imperative without utilizing a case metaphor. It does not require language extensions or paradigms such as rules, declarative elements, or intelligent agents.

Work related to hypermedia in combination with processes includes HATEOAS in combination with BPMS or PAIS. [21] 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 [22], which focused primarily on the composition and integration of services. Similarly, BPMashup [23] focuses on process-centric compositions of RESTful web services without addressing hypermedia. RESTfulBP [24] and CPEE [25] do not mention HATEOAS or hypermedia. RESTful with BPMN [26] extends BPMN notation with graphical syntax and semantics for REST support, but without directly addressing hypermedia or HATEOAS. Various commercial BPMS vendors support REST interfaces, yet to our knowledge HATEOAS is not explicitly mentioned.

3. Solution Concept

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

3.1. Background on the AProPro Solution Approach

As described in [11], AProPro uses an imperative process style, and process models are kept as simple and modular as reasonable in alignment with the orthogonal modularity pattern [27]. 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.

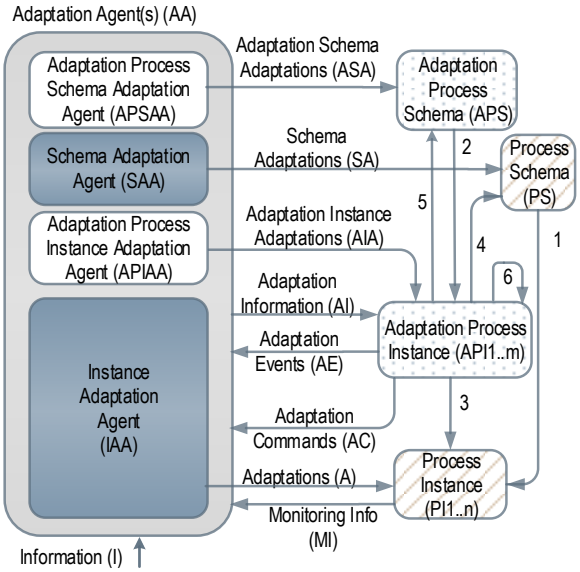

Figure 1. Conceptual Solution Architecture of AProPro [11]

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), 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 BPM Suite [9].

The 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. The following capabilities are elaborated:

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 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 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.

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.

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.

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).

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 processes 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 process is triggered first and applies its changes to the target; for pull, the target process triggers the adaptive process to initiate its adaptations.

Reusability: shared modeled/tested adaptation processes support the wider reuse of adaptation patterns in the community, e.g., via repositories like APROMORE [28].

Composability: more complex adaptations can be addressed by composing multiple adaptation processes, 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 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.

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 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 that can be interpreted to support an approach such as ours include: La Rosa et al. [29], 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. [30], imperative models had better understandability and comprehensibility than declarative ones; Reijers et al. [31] suggests that process modularity via information hiding enhances understandability; and Döhring et al. in [32] found that process complexity affected maintenance task efficiency for process variant construction - subjects hereby preferred high-level change patterns to process configuration.

3.2. The AProProh HATEOAS-PAIS Integration Middleware

The AProProh solution concept provides a HATEOAS-PAIS integration middleware to extend the AProPro approach described above. Initially described in [10], AProProh now includes a task queueing layer to support long-running task requests.

The middleware solution concept is shown in Figure 2. The *dbpm.pais_supplier* package serves as a PAIS abstraction layer containing PAIS-specific supplier plugins. It abstracts the use PAIS-specific APIs in the *connection*, *enactment*, *adaptation*, and *actions* subpackages. In this way, the REST-based middleware client can be decoupled from the actual PAIS supplier.

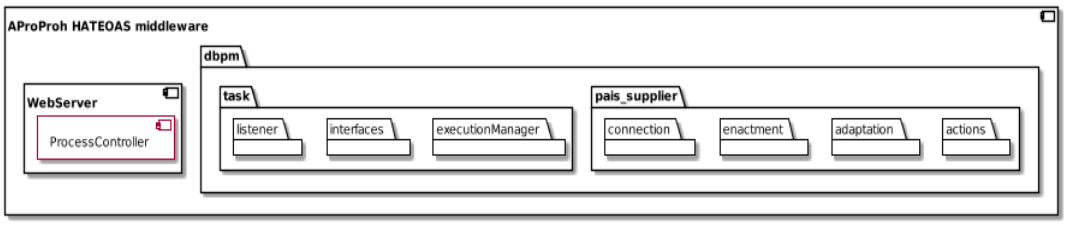

Figure 2. The AProProh HATEOAS-PAIS middleware solution concept

To allow client requests to be independent of the backend invocation durations, the *dbpm.task* package includes the *listener*, *interfaces*, and the *executionManager* packages for queueing and monitoring process task requests made by a client. REST requests to the web server are forwarded to a *ProcessController*, which invokes the task request functionality, which in turn invokes the *pais_supplier* functionality as a plugin.

4. Realization

To determine the viability of our approach, we realized a prototype of the middleware. In this section, the more general realization aspects are discussed. Since specific responses depend on the actual state of a process, these details are shown in the following Section 5.

4.1. Process-aware information system integration

The AristaFlow BPM Suite was used as an adaptive PAIS due to its support for dynamic adaptations with structural correctness checks. 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 in the PAIS supplier plugin 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 a REST server (which could be remote or local), which in turn invokes the PAIS supplier plugin that utilizes PAIS-specific APIs.

For simplicity, the prototype implementation used synchronous communication with the supplier PAIS, although the PAIS-specific plugin can be implemented to handle asynchronous interaction if supported by the PAIS.

4.2. REST stack

Multipart/form-data was in the JSON format. To show that the middleware solution concept can utilize different REST stacks, we chose two variants:

- Variant A: Spring 1.2.7, Spring HATEOAS 0.16, Apache Tomcat 8.0.28, and Jackson 2.4.6 for JSON.
- Variant B: Dropwizard 0.9.1, Jersey 2.22.1, Jetty 9.2.13, and Jackson 2.6.3.

4.1. Process adaptation patterns

As described in [11], 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}

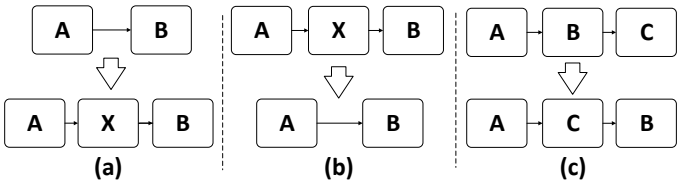

Figure 3. (a) insert, (b) delete, and (c) move process fragment patterns

The *insert* process fragment pattern, shown in Figure 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 Figure 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 Figure 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.

4.2. Task Queueing

To support long duration requests, any adaptation or execution POST request is queued by the ProcessController for the ExecutionManager with some default expiration time and assigned a task ID using a generated UUID that is passed back in the response to the client as a token. This allows a client to check on the status of its request at any time independent of when it is executed. A GET /task/{taskID} can be used to check on the state of a given operational request. DELETE /task/{taskID} removes the task from the execution queue if the task has not been executed yet or if it has completed and the client no longer needs it. Otherwise, the task status and its ID are automatically removed at its

expiration timepoint. A GET /procID/{procID} can also be used by the client to check on the state of any given process instance and to see all the undeleted and unexpired tasks related to that process instance.

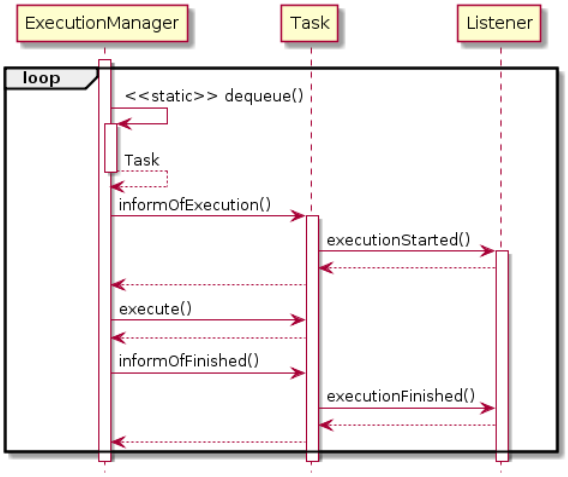

Figure 4. Middleware Internal Interaction

If the *ProcessController* is invoked with a new task request, it creates a new *Task* and invokes the static method enqueue() on the *ExecutionManager*. As shown in Figure 4, an *ExecutionManager*, running in one or more separate threads, determines in a loop if there are any *Task*s to be dequeued. For a given *Task* that is dequeued, a *Listener* is registered to monitor the *Task*. The ExecutionManager then invokes execute() on the *Task*, which then utilizes the PAIS supplier plugin implementation to perform an operation on the process via enactment, adaptation, or actions and thus affect its state.

4.3. Determining Possible Next Actions for the Client

To support HATEOAS, client should be informed of the valid possible next actions. Our algorithm dynamically determines the possible next actions for a request based on the current process state in the PAIS and programmed rules. The process state for a specific process instance in a PAIS is determined by the currently active node id, the state of the currently active node, and the output parameters of the currently active node. This typically includes the directly subsequent node(s), allowed 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, when an adaptation affects 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 as possible next actions in that interaction. Future work may incorporate a more generalized rule engine.

As an example of the concrete steps involved to determine the possible next actions for a given PAIS, our PAIS supplier implementation for AristaFlow connects to the AristaFlow server's AdministrationService, retrieves an InstanceManager, gets and locks the target process instance to determine what states the nodes are currently in, and then gets the id, state, and parameters of the node and returns the process state.

5. 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.

5.1. 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. While software engineering (SE) processes were used to concretely represent and convey the concepts and validate its practicality, the approach is domain independent.

5.1.1. Waterfall process (WP): We loosely follow this simple software engineering (SE) process consisting of common SE activities, making minor modifications (modeled in AristaFlow in Figure 5). A branch was inserted to demonstrate branch navigability, and start and end nodes were cropped from the screen shot due to space constraints.

Figure 5. WP with Branch

5.1.2. 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 Figure 6).

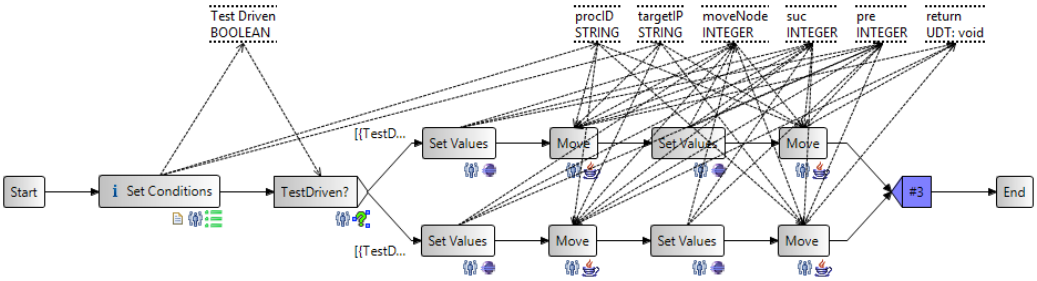

Figure 6. TDDAP

5.1.3. Quality assurance adaptation process (QAAP): This process (see Figure 7) demonstrates process 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.

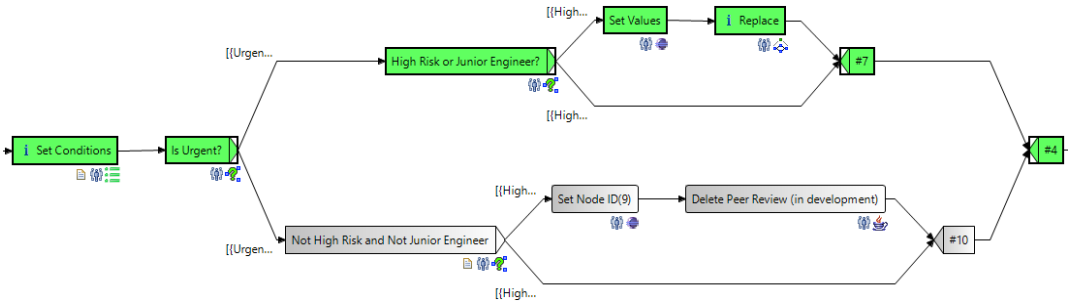

Figure 7. QAAP

5.1.4. REST interaction: An example of the contents of the response body for a typical GET /procID/{procID} where {procID} was the process ID and {ip} was the IP address is shown here:


```
{
 "content": "PossibleActions",
 "procID": "1e936b99-3392-48ec-9ad0-8aede188180b",
 "active_node": 11,
 "node_state": "NS_RUNNING",
 "parameters": {
  "Junior": "BOOLEAN",
  "High Risk": "BOOLEAN",
  "Urgent": "BOOLEAN"
 },
 "_links": {
  "QA": {
   "href":"http://{ip}/procID/1e936b99-3392-48ec-9ad0-8aede188180b/adaptation/qa?
urgent=false&high%20risk=false&junior%20engineer=true&targetIP={ip}" },
  "TDD": {
   "href":"http://{ip}/procID/1e936b99-3392-48ec-9ad0-8aede188180b/adaptation/tdd?
testDriven=false&targetIP={ip}" },
  "execute": {
   "href": "http://{ip}/procID/1e936b99-3392-48ec-9ad0-8aede188180b/execute" },
  "current-tasks": [
   { "href": "http://{ip}/task/06521ed8-8146-4037-91af-070fb572f309" },
   { "href": "http://{ip}/task/45894ecf-bf55-4d47-96b2-4fa13e6029d3" },
   { "href": "http://{ip}/task/a7996c6f-5c02-4766-8742-d01cd96d54bf" }
  ],
  "self": { "href": "http://{ip}/procID/1e936b99-3392-48ec-9ad0-8aede188180b" }
 }
}
```


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" include the next possible actions. The response to a POST provides a link to the task-status for the task just requested as well as a link to the process to provide an overview of the process state.

In the case of a POST, only the taskID and procID are provided as links to determine the possible next actions. An example of the response of a POST /procID/{procID}/adaptation/tdd is shown here:


```
{
```



```
 "content": "TDD",
 "procID": "1e936b99-3392-48ec-9ad0-8aede188180b",
 "active_node": 11,
 "node_state": "NS_RUNNING",
 "parameters": {
  "High Risk": "BOOLEAN",
  "Junior": "BOOLEAN",
  "Urgent": "BOOLEAN"
 },
 "_links": {
  "overview": { "href": "http://{ip}/procID/1e936b99-3392-48ec-9ad0-8aede188180b" },
  "task-status": { "href": "http://{ip}/task/6156a3b2-6086-4e9e-a9e5-c7d40eff87d5" }
 }
}
```


As to parameters, 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.

Figure 8 shows an example interaction. After each POST response, a GET can be invoked to determine the new process state (not shown). After the TDDAP POST and the following execute POST, the GET no longer lists "TDD" as a possible next action, since the currently active node is then "Unit Test". In addition, once "Code Review" is entered, "QA" is no longer listed since it is also no longer valid.

Figure 8. REST Client Server Interaction

5.1.5. Resulting process adaptations: The resulting process state can be seen in Figure 9. The numbers above the activities in Figure 8 indicate the number of visits to that activity node. 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.

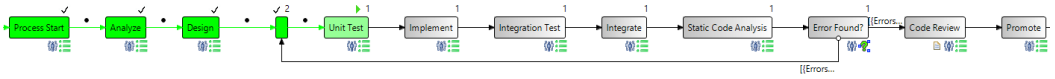

Figure 9. Resulting WP after Adaptations were Applied and Looped Once

The result shows that it is possible for a RESTful client, utilizing HATEOAS principles, to use the AProProh middleware to enact and adapt a process, with available next actions dynamically and contextually provided without a direct coupling to the PAIS.

4.2. Measurements

To characterize the performance implications of the HATEOAS-PAIS middleware, the collocated server-side overhead (without involving remote network latencies) was measured on a PC with the following configuration: i5-4460 CPU@3.2GHz, 8GB RAM, Win10 Pro, Java JRE 1.8, and AristaFlow 1.0.92. The Spring REST stack (variant A) consisted of Spring HATEOAS 0.16, Spring plugin core 1.1.0, Jackson 2.4.6, and Apache Tomcat 8. The Dropwizard REST stack (variant B) consisted of Dropwizard 0.9.1, Jackson 2.6.3, and Jetty 9.2.13. The REST client consisted of the Postman extension for the Chrome browser. For averages, the succeeding four operations after an uninitialized measurement were used. Note that no optimization or tuning was performed with our configuration, including the REST stacks or the PAIS, as these measurements are intended to evaluate support of our concept and not for benchmarking.

Table 1 shows that the initial GET /procID/{procID} request has a significant overhead, presumably related to initialization overhead, which dropped considerably for the subsequent requests, also possibly due to caching effects. The "REST overhead" related to the REST stack (including JSON) contributed on average approximately 3% of the overall latency for both variant A and B. The "Other overhead" can be attributed to the determination of the current process instance state and the possible next steps, as described in Section 4.3 on the realization.

Table 1. GET Latencies (in millisecs)

GET	Initial		Average	
	Spring	Dropwizard	Spring	Dropwizard
Total latency	2632	2345	312	259
Other overhead	2428	2195	256	251
REST overhead	204	150	9.0	8.0
REST overhead	7.8%	6.4%	2.9%	3.1%

Table 2 shows measurement results for POST /procID/{procID}/execute. Since a GET had already been invoked by this point, the initial POST latency is not so dramatically different from the average latencies. Likely, the difference is related to caching effects. For the average case, the REST overhead contributed 5% latency overhead. Compare to GET, for POST we observe on average a reduced latency under "Other overhead" of between 125-147 ms. This can be attributed to the decision not to have the POST response include the current process state nor calculate the possible next actions, as previously described in Section 5.1.4.

Table 2. POST Execute Latencies (in millisecs)

POST	Initial		Average	
	Spring	Dropwizard	Spring	Dropwizard
Total latency	188	145	137	110
Other overhead	175	132	131	104
REST overhead	13	13	6.5	5.8
REST overhead	6.9%	9.0%	4.7%	5.3%

In our configuration with these processes, the average latency for the PAIS operations measured at the PAIS supplier plugin was 166 ms for execute, 207 ms for TDD adaptation, and 205 ms for QA adaptation. When these durations are included in the overall response time without the queueing layer, for the Spring variant the average total latency for a POST execute was then 371 ms. With the queueing layer, the response latency was 137 ms. Thus, task queuing improved the POST response time towards the clients and kept them from waiting for the task completion. However, since the GET adds additional overhead to determine the current state and next possible actions, for short duration tasks the total delay can be longer with a POST followed by a GET combination than with a pure POST and wait, if the client were only interested in determining when the task completed.

In summary, both REST stacks did not significantly affect the latency overhead for the middleware approach. Rather, the performance characterization shows that supporting HATEOAS adds latency overhead to determine the current process state and the possible next actions with PAIS-specific calls. As the number of possible actions increases, each interaction is affected. While direct access of PAIS APIs may be a consideration when performance is critical, the tradeoff is losing a generic HATEOAS interaction and adaptation middleware to heterogeneous PAIS. Future work can investigate optimizations in this area.

6. Conclusion

A HATEOAS-PAIS middleware approach for enacting and adapting processes in a process-aware information system was described. 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 extends this to support PAIS-agnostic hypermedia-driven process enactment, adaptation, and navigation.

The evaluation consisted of a case study and measurements 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.

A deeper integration of REST middleware into the PAIS could reduce the overheads involved in determining the current state and possible next actions. If this were pursued, then the PAIS vendors should agree on a common standard for the web API interface. Otherwise, an approach similar to that advocated in this paper could provide a common web API interface to various PAIS implementations while encapsulating their PAIS APIs and differences. The shared PAIS supplier plugin concept can reduce the integration investment cost and PAIS vendor lock-in, allowing greater freedom to utilize different PAIS implementations for different purposes while unified behind a common interface.

Future work includes the integration of various other PAIS into the middleware, incorporation of a generalized rule engine to assist in determining the possible next actions for a client, and optimizations to reduce latencies.

Acknowledgments

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

References

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," Computer networks, 54(15), **(2010)**, pp. 2787-2805.
[2] M. Herrman, T. Pentek, and B. Otto, "Design Principles for Industrie 4.0 Scenarios: A Literature Review," Working Paper 01/2015, Technishe Universität Dortmund, **(2015)**.
[3] W. Aalst, "Process-Aware Information Systems: Design, Enactment, and Analysis," Wiley Encyclopedia of Computer Science and Engineering, **(2009)**.
[4] Object Management Group, "Business process model and notation (BPMN 2.0)," Object Management Group, **(2011)**.
[5] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guzar, N. Kartha, and C.K. Liu, "Web services business process execution language version 2.0 (OASIS standard)," WS-BPEL TC OASIS **(2007)**.
[6] Unknown, "BPMN Model Interchange: The Quest for Interoperability," available at http://www.omgwiki.org/bpmn-miwg/lib/exe/fetch.php?media=20150611_submission.pdf **(accessed 12/29/2015)**.
[7] R. T. Fielding, Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, **(2000)**.
[8] 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.
[9] M. Reichert and B. Weber, Enabling Flexibility in Process-aware Information Systems: Challenges, Methods, Technologies. Springer Science & Business Media, **(2012)**.
[10] R. Oberhauser, "A Hypermedia-Driven Approach for Adapting Processes via Adaptation Processes," in Proceedings of the 8th International Conference on Advanced Software Engineering and Its Applications (ASEA 2015), IEEE CPS, **(2015)**.
[11] 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
[12] H. de Man, "Case management: A review of modeling approaches," BPTrends, January **(2009)**.
[13] 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.

[22] C. Pautasso, "RESTful Web service composition with BPEL for REST," Data & Knowledge Engineering, 68(9), **(2009)**, pp. 851-866.
[23] 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.
[24] 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.
[25] 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 .
[26] C. Pautasso, "BPMN for REST," Business Process Model and Notation, Springer Berlin Heidelberg, **(2011)**, pp. 74-87.
[27] 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.
[28] 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.
[29] 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.
[30] 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.
[31] 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.
[32] 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.

Author

Roy Oberhauser, has 14 years of software industry experience in the Silicon Valley and in Germany. Since 2004 he has been a Professor of Computer Science at Aalen University in Germany, teaching in the areas of software and systems engineering. His research interest is to leverage technologies and techniques to innovate, automate, support, and improve the production and quality of software for society.