Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models

Kálmán Képes, Uwe Breitenbücher, Santiago Gómez Sáez, Jasmin Guth, Frank Leymann, Matthias Wieland

Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models

Kálmán Képes[1], Uwe Breitenbücher[1], Santiago Gómez Sáez[1], Jasmin Guth[1], Frank Leymann[1], and Matthias Wieland[2]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
[2]Institute for Parallel and Distributed Systems, University of Stuttgart, Germany
{lastname}@informatik.uni-stuttgart.de

Abstract The continuous growth of the Internet of Things together with the complexity of modern information systems results in several challenges for modeling, provisioning, executing, and maintaining systems that are capable of adapting themselves to changing situations in dynamic environments. The properties of the workflow technology, such as its recovery features, makes this technology suitable to be leveraged in such environments. However, the realization of situation-aware mechanisms that dynamically adapt process executions to changing situations is not trivial and error prone, since workflow modelers cannot reflect all possibly occurring situations in complex environments in their workflow models. In this paper, we present a method and concepts to enable modelers to create traditional, situation-independent workflow models that are automatically transformed into situation-aware workflow models that cope with dynamic contextual situations. Our work builds upon the usage of workflow fragments, which are dynamically selected during runtime to cope with prevailing situations retrieved from low-level context sensor data. We validate the practical feasibility of our work by a prototypical implementation of a Situation-aware Workflow Management System (SaWMS) that supports the presented concepts.

Keywords: Workflow Technology; Situation-Aware Workflow Execution; Workflow Adaptation; Workflow Transformation; Workflow Fragments

1 Introduction

The significant increase of devices with network capabilities allows the integration of such into large software systems, which enables paradigms such as the Internet of Things [3]. One fundamental aspect of such a paradigm is the existence of multiple sensors that continuously emit data representing the context of physical or virtual entities and running applications, e.g., temperature data of physical machines or the utilization of virtual machines that run software. Dynamic contextual changes have a severe impact on the application behavior, which must be able to cope with and to adapt themselves to different situations, e.g. opening or closing room windows to regulate its temperature. The existence of a wide

spectrum of possibly occurring situations across different application domains, however, arises several challenges to developers regarding the tasks of designing, implementing, and provisioning all necessary software artifacts to realize complex processes that provide the required runtime flexibility.

The workflow technology enables the modeling and executing of process models that describe the desired behavior of information systems [15]. Workflow models typically comprise a set of interconnected activities that are executed by a runtime environment to achieve a business goal. However, these models are not situation-aware by nature. For example, the usage of standardized workflow languages, such as BPEL [18] or BPMN [19], requires the explicit modelling of every individual behavior to cope with each and every possible environmental change. If a workflow model describes the steps of a production process, and one machine of this process breaks during execution, the overall workflow must be adapted to achieve the business goal, e.g. by adding activities that repair the machine or move the process to another machine. Unfortunately, if all possibly occurring situations must be considered, this leads to several issues at both modeling and runtime levels, as (i) modeling all possibly occurring situations leads to extensive and complex workflow models that are hard to create and even harder to maintain. In addition, (ii) process modelers may not have the complete knowledge about all possibly arising situations, and (iii) most of the existing standard-compliant workflow engines are currently not capable of handling the dynamic nature of frequently changing situations. Moreover, using standard-compliant technologies to realize workflows that adapt themselves to changing situations is a non-trivial issue. In contrast, existing situation-aware workflow management systems often employ custom, non-standardized workflow languages, which reduces the portability of workflow models between different runtimes.

In this paper, we tackle these issues. We present the *ProSit Method* that enables creating traditional process models, which are then automatically transformed into situation-aware workflow models, which can be executed by any standard-compliant runtime environment without requiring an extension of the employed workflow system. This transformation is achieved by searching workflow fragments in the original process, replacing each of them by a single activity whose execution is handled by a situation-aware service bus that dynamically selects an appropriate fragment that provides the original functionality for the currently prevailing situation during runtime. This supports the creation of dynamic, self-adaptive processes using the standard-compliant workflow technology and reduces the required expertise regarding possibly occurring situations. Moreover, we present an architecture of a situation-aware workflow management system called *ProSit System*, and validate its practical feasibility by a prototypical implementation. Finally, we conduct a case study to evaluate the approach.

The remainder of this paper is structured as follows. In Sect. 2, we motivate our approach while Sect. 3 presents a life-cycle of situation-aware workflows and the overall method. The architecture and implementation of the *ProSit System* is introduced in Sect. 4, which is subsequently evaluated in Sect. 5 using a case study. Sect. 6 discusses related works, Sect. 7 concludes the paper.

2 Motivation & Background

The workflow technology has considerably influenced the development of software, as it allows the robust and reliable automation of business processes [15]. The foundations of this technology have contributed to the creation of several standards, such as BPEL and BPMN. Due to the standardization, these languages enable creating portable process models that can be executed by different standard-compliant runtimes, therefore avoiding vendor lock-in. However, these languages do not support an efficient means to handle changing situations and to model situation-aware behavior without polluting the respective models with extensive and heterogeneous situation handling logic.

Several works have targeted the workflow adaptation in supply chain and pervasive environments, e.g., [2,8,24]. More specifically, the enhancement of process models with context information and the usage of process fragments as a means to dynamically adapt workflows have been the major research contributions of these works. However, these approaches do not support the development of standard-compliant workflow models that automatically become situation-aware during their execution, which is the major research goal of the ProSit-Method. In addition, due to large amounts of sensors propagating low-level heterogeneous data, there is a need to integrate mechanisms for detecting aggregated high-level situations that provide well-defined semantics.

To overcome these issues, in the SitOPT project[1], we aim at providing a *Situation-aware workflow Management System (SaWMS)* capable of aggregating low level sensor data to high level situations and using these situations for dynamic workflow adaptation [25]. In the following, we describe the necessary concepts of this architecture that are required to understand the contributions of this paper. Fig. 1 depicts the overall SitOPT architecture, consisting of three main layers: *Sensing Layer*, *Situation Recognition Layer*, and *Situation-aware Workflow Layer*. The Sensing Layer comprises the set of domain-specific sensors, which are basically responsible for reading context parameters and propagating data samples to the upper layers. The Situation Recognition layer filters, aggregates, and processes the contextual data retrieved from the different objects. The data aggregation and processing tasks are driven by the Situation Recognition middleware, which is mainly responsible for receiving the low level sensor data and mapping this data to high level situations. The existence of multiple *Sensor Adapters* enable the data processing and aggregation from different domain-specific sensors.

The situation-aware workflow adaptation is handled by the SaWMS and the *Situation Handler*. The SaWMS is responsible for executing workflows and passing all service invocations to the Situation Handler, which mainly acts as a situation-aware service bus regarding the contributions of this paper. If a request is received by the Situation Handler, it selects an appropriate workflow fragment (stored in the *Workflow Fragment Repository*) that is capable of executing the requested operation in awareness of the currently prevailing situation. The *Situation-Aware Workflow Modeling Tool* supports creating workflow models by suggesting

[1] https://www.ipvs.uni-stuttgart.de/abteilungen/as/forschung/projekte/SitOPT

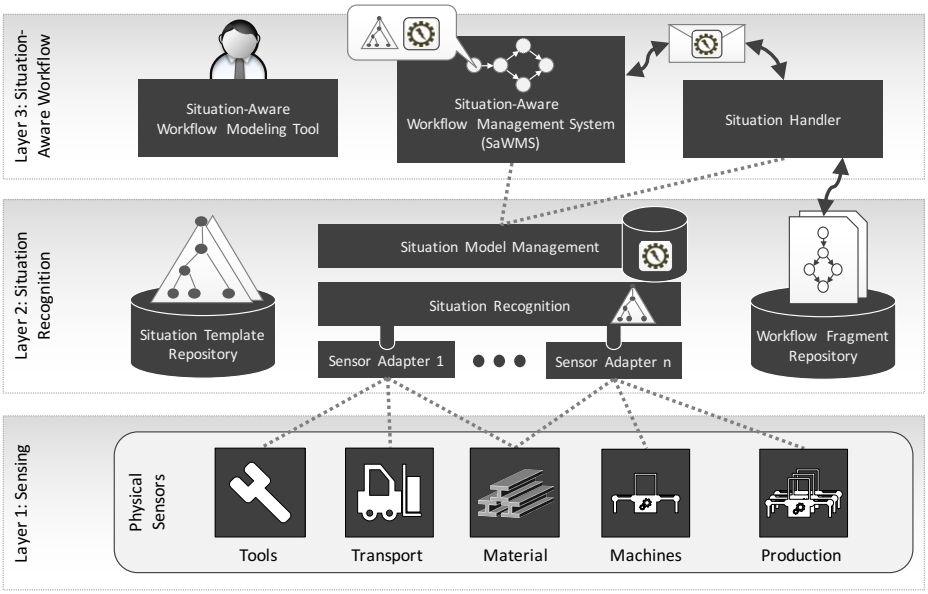

Figure 1. Overview of the SitOPT architecture and its three layers [25]

activities and operations, respectively, which can be adapted dynamically for different situations. In this paper, we extend the SaWMS by a method and concepts to enable automatically transforming traditional, situation-unaware workflow models in situation-aware models that are dynamically adapted using the Situation Handler.

3 Situation-Aware Execution of Workflow Models

As described in the previous sections, the current workflow technology is not situation-aware by nature. In this section, we (i) introduce a life-cycle for situation-aware workflows and (ii) present the ProSit-Method for the transformation of traditional workflow models into situation-aware workflows afterwards.

3.1 Situation-Aware Workflow Model Life-Cycle

In Fig. 2, we introduce a life-cycle encompassing the (i) modeling, (ii) provisioning, and (iii) execution of situation-aware workflows using workflow fragments as the basis for the dynamic runtime adaptation. This life-cycle defines the context and basis for the ProSit-Method that is introduced in the next section.

In a first phase, situation-aware workflow fragment models that implement a certain action for a concrete situation are developed. For example, in the smart home domain, a fragment model is developed that can reduce the temperature in a room if the temperature outside the room is lower. Another fragment may

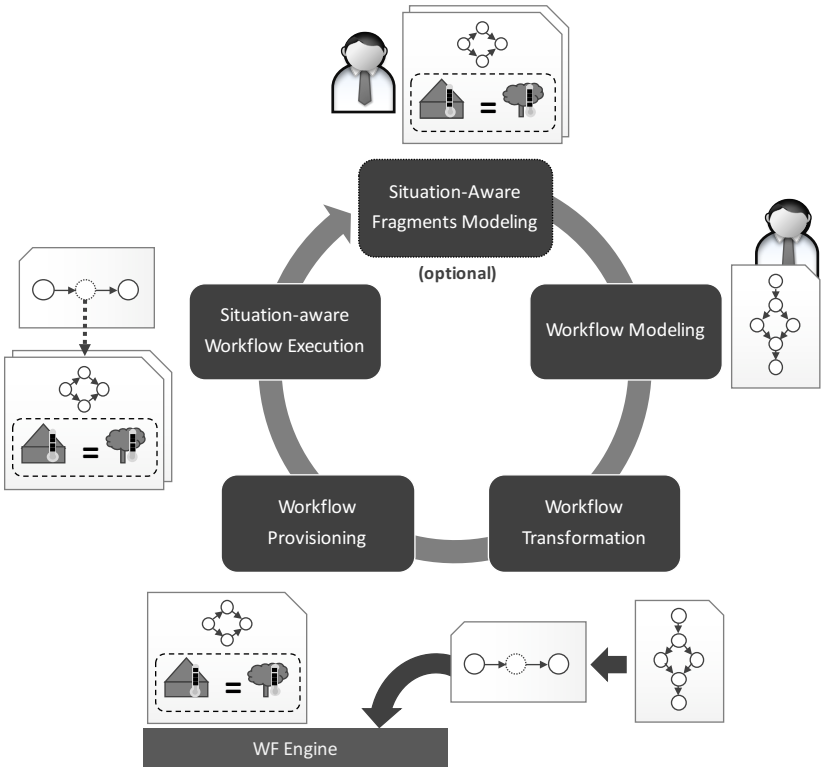

Figure 2. Situation-Aware Workflow Model Life-Cycle

implement the same action for another situation, e.g. for the situation that the temperature outside the room is higher. While the first fragment opens a window to reduce the temperature, the second activates the air conditioner. These fragment models are typically modeled by domain experts and can be used by different processes. However, their suitability differs depending on the prevailing situation. Since the repository of available generic situation-aware workflow fragment models grows over time, this first phase may be skipped if the modeler decides that the available fragments, i.e. the possible adaptations, are sufficient for the workflow to be created. Therefore, this phase is optional.

In a second phase, the workflow model is created, describing the set of activities that must be performed and the data flow between them. To increase the efficiency of modeling, and to avoid errors, available workflow fragments should be used since they have been developed by domain experts.

The created workflow model is then transformed into a situation-aware workflow model in a third phase. This is done by detecting all sets of activities within the modeled workflow that are semantically and structurally equivalent to available situation-aware workflow fragments in the repository. This matching is performed to replace the matching parts in the workflow model by single

placeholder activities that specify the invocation of the respective functionality implemented by the matched fragment, e.g., *Reduce Temperature*. Based on situation-aware workflow fragments, which all specify the action they can execute for a certain situation, this provides the basis to select an appropriate fragment on runtime to execute the functionality specified by a certain placeholder activity. This is detailed in the next subsection that presents the ProSit-Method.

In the next phase, the *Workflow Provisioning* phase takes place: the generated situation-aware workflow model is deployed on a standard-compliant workflow engine that uses the Situation Handler as service bus. Thus, all invocations of placeholder activities are directed to the bus. The provisioning of all workflow fragments, which implement actions that are specified by placeholders, is also done in this phase. To improve this phase, in future work, the provisioning of fragments may be done on-demand, as presented by Vukojevic et al. [23]. In the last phase, the provisioned workflow model is executed in a situation-aware manner, by means of dynamically selecting workflow fragments depending on prevailing situations.

3.2 ProSit-Method: Generating Situation-Aware Workflows

In this section, we present the ProSit-Method, which details the life-cycle presented in the previous subsection. In particular, the method supports transforming traditional, situation-unaware workflow models into situation-aware workflow models that contain placeholder activities for the invocation of actions that shall be dynamically adapted based on the prevailing situations. The method is depicted in Fig. 3 and consists of six steps that are presented in this section.

The first step of the method, *Workflow Modeling*, corresponds to the second phase in the life-cycle and, therefore, consists of modeling the desired workflow model. In this paper, we focus on imperative, graph-based workflow languages as described by Pichler et al. [20], e.g. using BPEL or BPMN. In the second step, the *Fragment Detection*, all available situation-aware workflow fragment models are structurally and semantically matched against the workflow model. More specifically, the main objective of this step is to detect *subworkflow models*, as defined in [14], that are equivalent to a certain workflow fragment model. Since every workflow fragment model describes (i) the action it implements as well as (ii) the situation for which it can be executed, this matching enables detecting the semantics of certain parts of the workflow model. For example, if the fragment matches the model that reduces the temperature of a room by opening the window, the action *Reduce Temperature* has been recognized. It is fundamental to denote that this step does not restrict the techniques to be used to determine neither the semantic nor the structural equivalence. For example, subgraph isomorphism algorithms can be used to match the control flows of the workflow model and workflow fragment model as well the respective data flows [7]. The semantic equivalence of activities mainly depends on the domain and can be realized, for example, by matching the labels of activities. In our prototype, we defined equivalence rules and implemented the described matching for the

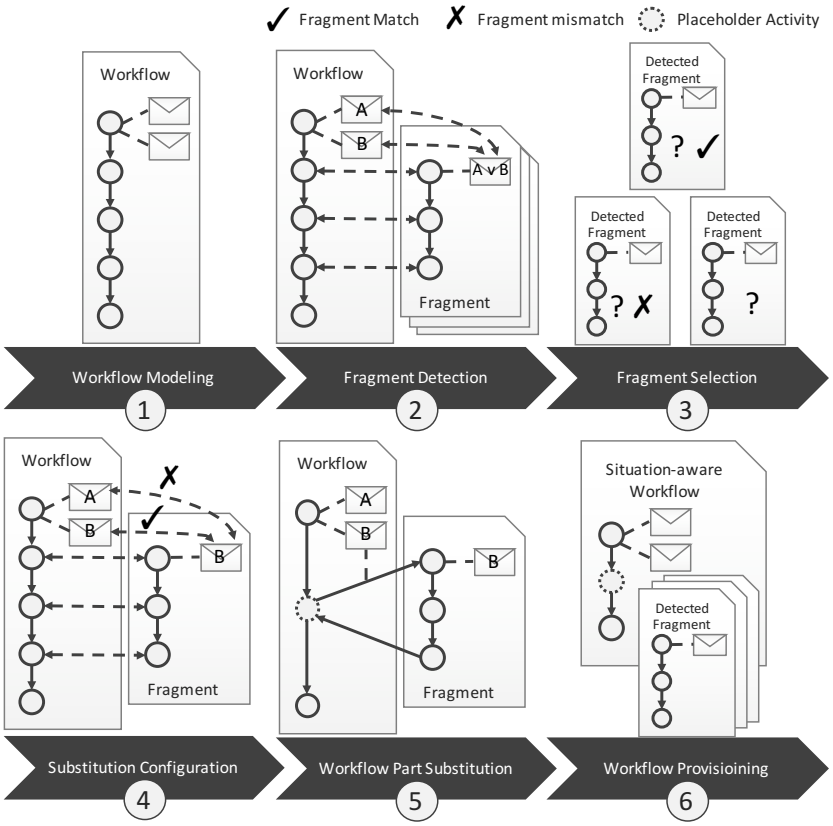

Figure 3. The ProSit Method

language BPEL, (see Sect. 4). If the workflow modeling in the first step uses available fragments, the probability of finding matching fragments increases.

After detecting matching fragments in the workflow model, some of them may be overlapping. For resolving such overlaps, the *Fragment Selection* step enables to manually select the fragment that shall replace the placeholder activities. However, if the matching is unique or the selection shall be done by the system, a fully automated approach can be realized, too.

The *Substitution Configuration* step configures the data flow, if necessary. This step is required, if the data flow of the original workflow and the selected workflow fragment(s) do not uniquely match. More specifically, the matching among the workflow variables and the input and output messages of the workflow fragment is driven. If necessary, this step can also target the semantic checking among existing data variables in the original workflow and their equivalent variables in the workflow fragment model. After this step, it is ensured that the workflow model part to be replaced is matched by at least one proper fragment that is equivalent in control flow, data flow, and semantics.

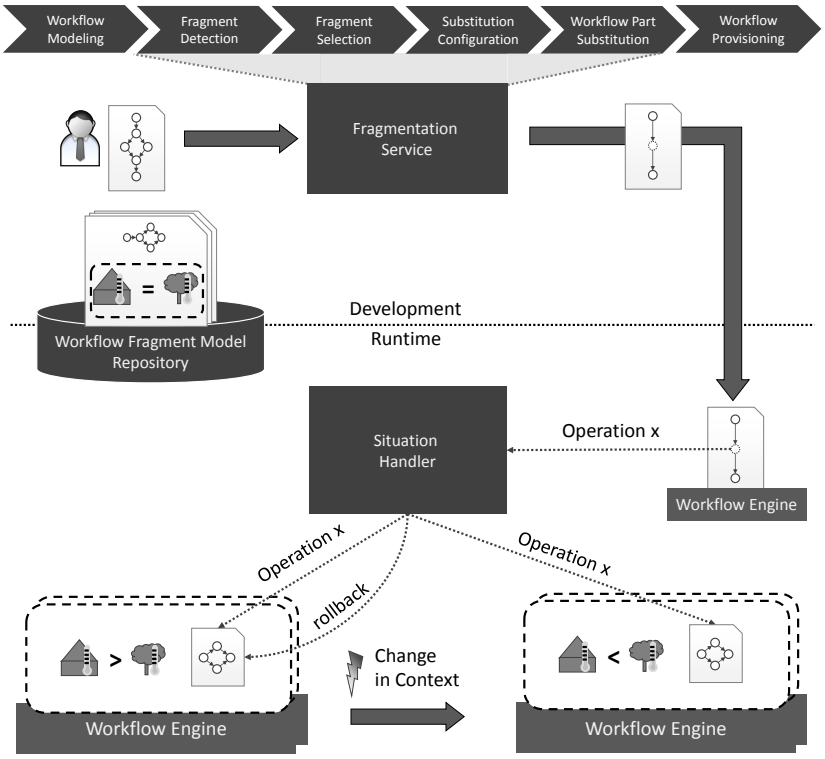

Figure 4. ProSit System - Architectural Overview

Once the data flow compatibilities are resolved, the *Workflow Part Substitution* step replaces each matched part inside the workflow model with a single placeholder activity, therefore transforming the original workflow model into a situation-aware workflow model. The inserted activities are responsible for executing the specified action, e.g., *ReduceTemperature*, depending on the currently prevailing situation. Our architecture presented in the next section supports this by discovering and selecting appropriate workflow fragments for each placeholder activity and invoking them. Thus, the situation-aware placeholder activity prescribes the use of *late binding* [21], since the overall system provides a set of workflow fragments for which it is specified under which situations they are allowed to be executed. The ProSit method does not restrict on the mechanism and technological support for the execution of a placeholder activity, such as inside the running workflow instance itself, as presented in [8], or through an external service bus. However, the latter is realized in our architecture.

Finally, the situation-aware workflow model is provisioned. The provisioning mechanism is not restricted by the method, since it depends on the implementation.

4 Architecture and Realization

In this section, we present the conceptual architecture of the ProSit-System, which realizes the ProSit-Method and the introduced life-cycle. Fig. 4 depicts the architecture. The system is tailored towards two main environments: *Development* and *Runtime*. In the *Development Environment*, the workflow model and workflow model fragment developers create the respective models. Workflow fragment models are persisted in the *Workflow Fragment Model Repository*, which persists a set of tuples, each one containing the (i) fragment model, (ii) deployment artifacts required for deploying and executing the model fragment, and (iii) the situation and goal descriptions that each workflow fragment model is adequate for.

The *Fragmentation Service* enables the transformation of standard-compliant, situation-unaware workflow models into fragmented situation-aware workflow model variants by means of performing matching operations of workflow fragment models in the traditional model, as described in the previous section. The output of the *Fragmentation Service* is a situation-aware workflow model containing concrete executable activities and situation-aware placeholder activities, which are later bound to a certain implementation during the execution phase, such as a workflow fragment model. Situation-aware workflow models are then provisioned on a standard-compliant workflow engine that is capable of executing workflow models specified in a standard workflow language. Since the original, traditional, workflow model has been transformed and changed in terms of replacing activities by placeholder activities, these placeholder activities need to be handled. In particular, this means that for each placeholder activity an invocation of the Situation Handler is defined. Thus, all actions that need to be executed are sent to this handler that is responsible for executing the specified action while monitoring the current prevailing situation. To serve such requests, the handler discovers and invokes an appropriate workflow model fragment, i.e., a fragment that is capable of executing the requested action and that specifies the current prevailing situation for such an action (see Fig. 4 *Operation X*). Thus, the Situation Handler acts as a *Situation-Aware Service Bus* that discovers appropriate services, which are implemented as workflow fragments, and handles the interconnection among them. To enable this, workflow fragments must be complete workflow models that can be executed standalone.

To allow the Situation Handler to find appropriate workflow fragments, these are registered in a repository. This repository contains the workflow fragment models including all meta-data, e.g., which action it implements for which situation, and *situational endpoints*, which are endpoints referencing already deployed fragments that can be invoked directly. Compensation tasks, e.g. due to failures, are also handled by the Situation Handler, by means of rolling back the execution of a workflow model fragment if the originally prevailing situation changes. In this case, a rollback message is sent to the endpoint and a new feasible endpoint is selected. Details about this rollback, as well as about the architecture and prototype of the described Situation Handler can be found in Fürst [6].

We implemented the presented architecture and matchmaking concepts in the scope of the SitOPT Workflow Management Environment [25,4,10]. With respect

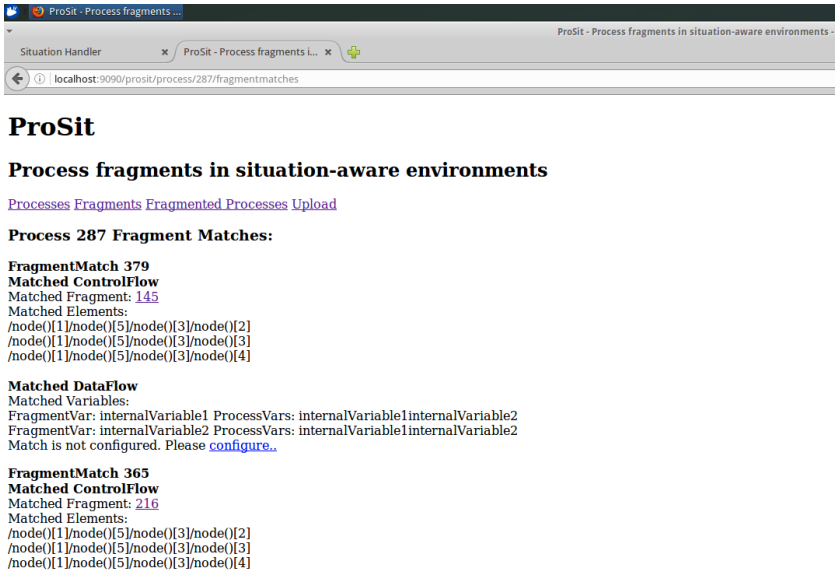

Figure 5. ProSit component showing the matched fragments against a workflow model

to the implementation of our approach, in the modeling environment, workflow developers use the Eclipse based BPEL Designer[2] for the modeling of workflow and workflow fragment models. For the persistence and discovery of workflow fragment models, i.e. for implementing the Workflow Fragment Model Repository shown in Fig. 4, we used the Fragmento repository presented by Schumm et al. [22]. In the scope of this paper, we implemented the ProSit Transformation Service as a RESTful API that allows to process BPEL workflow fragment models and BPEL workflow models. When BPEL workflows are processed, the matching against Single-Entry-Single-Exit-based fragments is started, where we use the library JGraphT [12] to transform these models into graph-representations for solving an subgraph isomorphism between the models. The results of each of the steps are stored as XML data inside the Fragmento repository to be accessable by external clients to additionally configure certain aspects, such as selection of fragments for replacement (see Fig. 5). Details about the mapping of control flow, data flow, and activity semantics regarding BPEL can be found in Képes [13]. The Situation Handler is developed as a RESTful service that allows adding situational endpoints and to register on occurrences of situations (see Fürst [6] for more information). Concrete situational endpoint data is persisted as interface descriptors according to the WSDL standard (see Fig. 6). The routing mechanism in the Situation Handler is handled through the Apache Camel [11].

[2] BPEL Designer: https://eclipse.org/bpel/

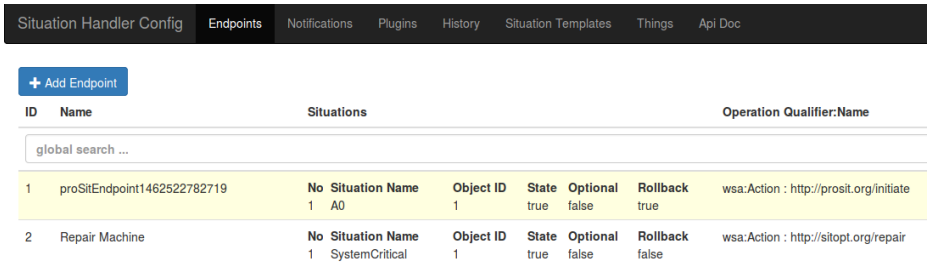

Figure 6. The Situation Handler Web UI to add situational endpoints

5 Case Study

The evaluation of our approach has been performed by means of conducting a case study from the SitOPT project. More specifically, we implemented a room temperature regulation mechanism using situation-aware workflows as the basis, as depicted in Fig. 7. The evaluation presented in this section consists of (i) transforming a *RoomRegulation* workflow into a situation-aware workflow, by means of using the *Fragmentation Service*, and the *OpenWindow* and *RegulateClimate* fragments, which are subsequently (ii) provisioned, executed, and adapted using the standard-compliant workflow engine WSO2 BPS.

A first step in our validation consists of modeling a set of workflow fragment models, i.e. *OpenWindow* and *RegulateClimate*, which are two actions that can be performed to regulate a room's temperature. Subsequently, the modeling of the *RoomRegulation* workflow comprises the set of necessary tasks to regulate the temperature in a smart room. Subsequently to the modeling phase, the *Fragmentation Service* transforms the original *RoomRegulation* into a situation-aware workflow by substituting its logic with situation-aware placeholder activities (see step 2 in Fig. 7). This substitution is performed taking the potential situational workflow fragment models persisted in the Workflow Fragment Model Repository into consideration.

After the provisioning of the *RoomRegulation*, the execution phase takes place. When a variation of temperature in the room occurs, an instance of the *RoomRegulation* workflow invokes the operation *reduceTemperature* in the *Situation Handler* (see step 3 in Fig. 7), which determines the outside temperature by invoking the SitOPT situation recognition system *SitRS* [10]. This allows the Situation Handler to determine which workflow fragment to use for serving the original *reduceTemperature* request (see steps 4 and 5 in Fig. 7). In our scenario, the room temperature is lower than outside, so opening the windows won't suffice to achieve the goal of reducing temperature. Therefore, once a workflow fragment model is discovered, the *Situation Handler* selects the *RegulateClimate* fragment which activates the climate control to serve the *reduceTemperature* request. Once the temperature is regulated, the *RegulateClimate* continues its execution, potentially waiting until the temperature is stabilized.

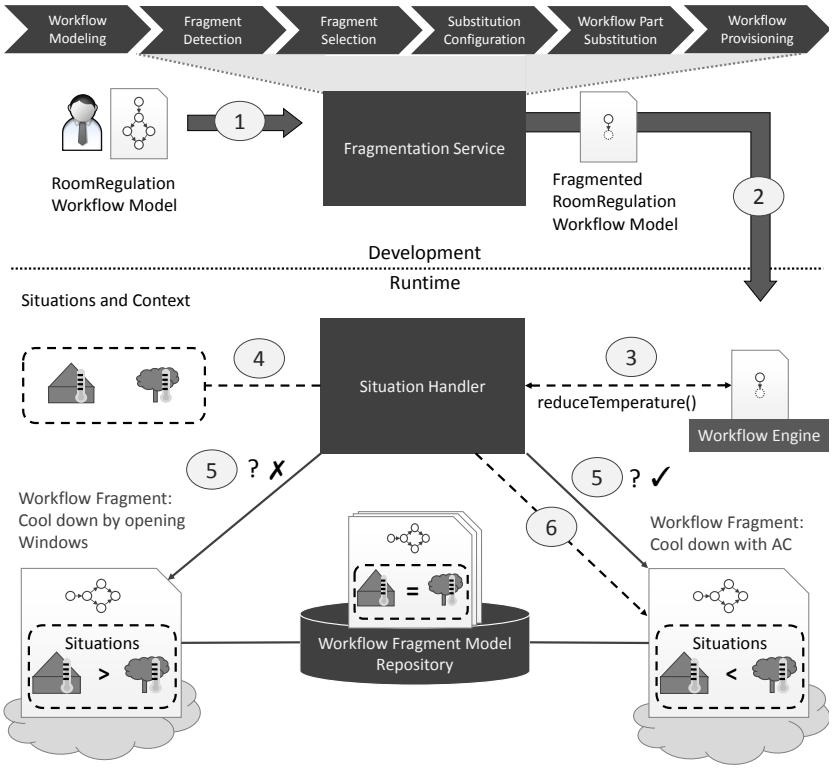

Figure 7. Room Temperature Regulation Case Study

6 Related Work

This section presents related work in the domains of workflow flexibility, and context- and situation aware applications. Refinement of processes has been widely studied in other works. Context-aware process injection (CAPI) is introduced as a concept for the execution of process fragments during runtime [17], by means of enabling the design of processes that adapt themselves into specified process regions based on the actual context. Fragments are executed if a specified region of the process model is reached. Context-aware annotated fragments can be executed sequential or parallel, and once or multiple times. Although this approach represents a language extension, it is not standard compliant. Bucchiarone et al. [5] enable the usage of processes fragments to refine context-aware abstract activities, in order to react to contextual conditions. While this approach is highly flexible, it is neccessary to have detailed knowledge of the domain and processes running in it. Developers have to define entities with their possible states, fragments with annotated goals, preconditions, effects and compensation effects.

The concept of *Worklets* is introduced in [1], which partially or completely implements a context-aware process. A collection of subprocesses are conflated using *Ripple Down* rules, which evaluate activities and discover which *Worklet* matches to a specific context. While the approach is flexible in selecting workflows (Worklets) when activating a task that needs to be substituted, it is missing changes at runtime of such workflows, e.g. the context changes and the activities in the substituting workflow isn't appropriate anymore. Aligned with such an approach, Wieland et al. introduce Context4BPEL, which consists of a language extension for BPEL 2.0 [24]. It focusses on precise context information, which are used for modelling within activities and control flow. Breitenbücher et al. introduce SitME, a concept that enables the modelling of situations on the workflow tier [4]. Within a start activity, a workflow can receive occurring situations to start the execution of a workflow. Additionally, *Situational Scopes* can be defined within a workflow, which can only be executed if specified situations prevail. The approach presented in [24] works on the workflow instance level. Therefore, introducing an huge burden in performance when handling fine-grained sensor data inside the workflow engine. [4] transforms the defined situation-aware constructs to native elements of the target workflow language.

Modafferi et al. [16] introduce a concept for developing context-aware workflows, by selecting alternative subprocesses based on context data. To react on changes within the context, the standard behavior of workflow engines, i.e., the rollback/compensation of activities, is used. Counteracting the expensive rollback, Modafferi et al. define edges between subprocesses, which can be evaluated during runtime. If an edge is existent, the workflow engine can switch to the alternative subprocess without any rollback. Similar to the SitME concept, this approach represents a language extension. In González et al. [9], the usage of a context-aware Enterprise Service Bus (ESB) is proposed to adapt service calls based on complex events. The main idea is that ESBs are responsible for transforming, routing, etc. messages between participants in the system. González et al. define high-level situations based on CEP rules that when are processed by a *Context Reasoner* to adapt specific parts of the message.

7 Conclusion

The emergence of network-capable devices has raised a number of challenges in the last years related to how to aggregate and process massive amounts of data retrieved from multiple interconnected sensors, and react accordingly to environmental changes. Applications utilizing such devices must be context-aware by nature, and must provide agile and flexible mechanisms to react to different situations.

This work focuses on how to build and execute such applications using the well established workflow technology as the basis. As the workflow technology is not situation-aware by nature, we focus in this paper on enhancing such technology to support the situation-aware adaptability features required by context-aware application systems. For such a purpose, a life-cycle for situation-aware workflows

is firstly presented. The life-cycle phases related to the (i) transformation of traditional into situation-aware workflow models, and the (ii) execution of such situation-aware workflows, are covered by the ProSit method and architectural support. Situation-aware workflows can be generated by discovering, matching, and replacing workflow fragments in traditional workflow models. Situation-aware workflows can then be executed and adapted based on retrieved situations. The evaluation of our approach is performed in the scope of the SitOPT project, by means of using a smart home case study as the basis.

Future works are aligned with exploring the usage of workflow fragments of more complex shapes as the Single-Entry-Single-Exit, where only one start and end activity are specified. Moreover, we plan to evaluate our approach using further case studies in the IoT domain, as well as investigating the usage of our approach in Cloud scenarios, i.e. for the situation-aware management of Cloud resources.

Acknowledgments

This work is partially funded by the BMWi German Projects "SePiA.Pro" (01MD16013F) and "SmartOrchestra" (01MD16001F), and the DFG German Project "SitOPT" (610872).

References

1. Adams, M., ter Hofstede, A.H.M., Edmond, D., van der Aalst, W.M.P.: Worklets: A Service-oriented Implementation of Dynamic Flexibility in Workflows. In: On the Move to Meaningful Internet Systems: OTM 2006 (CoopIS 2006). pp. 291–308. Springer (2006)
2. Ardissono, L., Furnari, R., Goy, A., Petrone, G., Segnan, M.: Context-aware Workflow Management. In: Web Engineering, pp. 47–52. Springer (2007)
3. Atzori, L., Iera, A., Morabito, G.: The Internet of Things: A survey. Computer networks 54(15), 2787–2805 (2010)
4. Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Wieland, M.: A Situation-Aware Workflow Modelling Extension. In: Proceedings of iiWAS'15. pp. 478–484. ACM (2015)
5. Bucchiarone, A., Marconi, A., Pistore, M., Raik, H.: Dynamic Adaptation of Fragment-based and Context-aware Business Processes. In: Proceedings of the 19th International Conference on Web Services (ICWS). pp. 33–41. IEEE (2012)
6. Fürst, S.: Konzept und Implementierung eines Situation Handlers. Master Thesis, University of Stuttgart, IAAS (2015)
7. Gallagher, B.: Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. AAAI FS 6, 45–53 (2006)
8. Gómez Sáez, S., Andrikopoulos, V., Hahn, M., Karastoyanova, D., Weiß, A.: Enabling Reusable and Adaptive Modeling, Provisioning & Execution of BPEL Processes. In: Proceedings of SOCA'15. IEEE (2015)
9. González, L., Ortiz, G.: An Event-Driven Integration Platform for Context-Aware Web Services. J. UCS 20(8), 1071–1088 (2014)

10. Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Leymann, F.: SitRS - A Situation Recognition Service based on Modeling and Executing Situation Templates. In: Proceedings of the 9th Symposium and Summer School On Service-Oriented Computing. pp. 113–127. IBM Research Report (2015)
11. Ibsen, C., Anstey, J.: Camel in Action. Manning Publications Co., Greenwich, CT, USA, 1st edn. (2010)
12. JGraphT Team: JGraphT - a free Java Graph Library (2016), http://jgrapht.org/
13. Képes, K.: Erkennung und dynamische Ersetzung von Fragmenten in Workflow-Modellen. Master Thesis, University of Stuttgart, IAAS (2016)
14. Kopp, O., Eberle, H., Leymann, F., Unger, T.: The Subprocess Spectrum. In: Proceedings of the Business Process and Services Computing Conference (BPSC 2010). vol. P-177, pp. 267–279. Gesellschaft für Informatik e.V. (GI) (2010)
15. Leymann, F., Roller, D.: Production Workflow: Concepts and Techniques. Prentice Hall PTR (2000)
16. Modafferi, S., Benatallah, B., Casati, F., Pernici, B.: MOBIS'05, chap. A Methodology for Designing and Managing Context-Aware Workflows, pp. 91–106. Springer (2005)
17. Mundbrod, N., Grambow, G., Kolb, J., Reichert, M.: Context-Aware Process Injection: Enhancing Process Flexibility by Late Extension of Process Instances. In: Proceedings of CoopIS'15. Springer (2015)
18. OASIS: Web Services Business Process Execution Language (WS-BPEL) Version 2.0. Organization for the Advancement of Structured Information Standards (OASIS) (2007)
19. OMG: Business Process Model and Notation (BPMN) Version 2.0. Object Management Group (OMG) (2011)
20. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative versus Declarative Process Modeling Languages: An Empirical Investigation. In: Business Process Management Workshops. pp. 383–394. Springer (2011)
21. Schonenberg, H., Mans, R., Russell, N., Mulyar, N., van der Aalst, W.: Process Flexibility: A Survey of Contemporary Approaches. In: Advances in Enterprise Engineering I, pp. 16–30. Springer (2008)
22. Schumm, D., Karastoyanova, D., Leymann, F., Strauch, S.: Fragmento: Advanced Process Fragment Library. In: Information Systems Development, pp. 659–670. Springer (2011)
23. Vukojevic-Haupt, K., Gómez Sáez, S., Haupt, F., Karastoyanova, D., Leymann, F.: A Middleware-centric Optimization Approach for the Automated Provisioning of Services in the Cloud. In: Proceedings of CloudCom'15. pp. 174–179. IEEE (2015)
24. Wieland, M., Kopp, O., Nicklas, D., Leymann, F.: Towards Context-Aware Workflows. In: CAiSE 2007. pp. 577–591 (2007)
25. Wieland, M., Schwarz, H., Breitenbücher, U., Leymann, F.: Towards Situation-Aware Adaptive Workflows: SitOPT A General Purpose Situation-Aware Workflow Management System. In: Proceedings of PerCom'15. pp. 32–37. IEEE (2015)