Situation-Aware Management of Cyber-Physical Systems

Kálmán Képes, Uwe Breitenbücher and Frank Leymann

Institute of Architecture of Application Systems, University of Stuttgart, Germany,
{kepes, breitenbuecher, leymann}@iaas.uni-stuttgart.de

BIBTEX:

```
@inproceedings{Kepes2019_SitAwareMngmntCPS,
  author    = {K\'{a}lm\'{a}n K\'{e}pes and Uwe Breitenb\"{u}cher
               and Frank Leymann},
  title     = {Situation-Aware Management of Cyber-Physical Systems},
  booktitle = {Proceedings of the 9th International Conference on Cloud
               Computing and Services Science (CLOSER 2019)},
  year      = {2019},
  month     = may,
  pages     = {551--560},
  doi       = {10.5220/0007799505510560},
  isbn      = {978-989-758-365-0},
  publisher = {SciTePress}
}
```

These publication and contributions were presented at CLOSER 2019
CLOSER 2019 Web site: http://closer.scitevents.org

Situation-Aware Management of Cyber-Physical Systems

Kálmán Képes, Uwe Breitenbücher and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Universitätsstraße 38, 70569 Stuttgart, Germany
{*kepes, breitenbuecher, leymann*}*@iaas.uni-stuttgart.de*

Keywords: Application Management, Cyber-Physical System, Situation-Aware System, TOSCA

Abstract: The current trend of connecting the physical world with the so-called cyber world resulted in paradigms such as the Internet of Things or the more general paradigm of Cyber-Physical Systems. The wide range of domains applicable results in a heterogeneous landscape of software and hardware solutions. To benefit of the paradigm, developers must be able to integrate different solutions from a range of different domains. However, these systems must therefore be able to change components, configurations and environments, hence, be adaptable at runtime. We present an approach that is based on the combination of Situation-Aware Adaptation concepts and Deployment Models. The general idea is to start processes that can change application structure and configuration when a certain situation in the context of applications occur. We validated the technical feasibility of our approach by a prototypical implementation based on a Smart Home scenario.

1 Introduction

The current trend of connecting the physical world with the so-called cyber world results in applications that depend on integrated software and hardware components that created paradigms such as the Internet of Things (IoT) (Atzori et al., 2010) or the more general paradigm of Cyber-Physical Systems (CPS) (Gunes et al., 2014). There is a wide range of possible scenarios for applying CPS, such as the domains of health care (Haque et al., 2014; YIN et al., 2016), mobility (Guo et al., 2017; Zorzi et al., 2010) and energy (Andr et al., 2011; Shrouf and Miragliotta, 2015), that promise a seamless integration within our everyday life. To benefit of the CPS paradigm, developers must be able to integrate different CPS solutions either by using single or a whole set of software and hardware components. A CPS that controls certain aspects of a Smart Home must be able to adapt to changes at runtime, enabling a home owner to continuously extend the system with new functionality.

However, achieving adaptability, i.e., reconfigurability in CPS is a complex challenge. The changing of CPS has to regard aspects, such as, changes to infrastructures, system configuration, components and users, therefore CPS must be aware of their context. The research field of Context-Aware Systems (CAS) investigates the challenge of making applications aware and adaptable to their context. According to Dey context can be defined as "..any information that can be used to characterize the situation of an entity." (Dey, 2001) which can be categorized into, e.g., spatial-, temporal-, device- and environmental context (Knappmeyer et al., 2013). Therefor applying concepts of CAS to CPS is a promising way to achieve high adaptability, as already identified by Perera et al. (Perera et al., 2014). The execution of management logic at the right time in and state of the context of a CPS enables it to be situation-aware and therefore enable the Situation-Aware Management of CPS.

In this paper our approach is to use Management Process that are able to execute various tasks, such as, storing backups, scale up or update components. These processes are combined with situations (Häussermann et al., 2010; Wieland et al., 2015) that can occur for physical or virtual 'things' and are observed in separated applications with the sole purpose of the observing situations occurring to things. Based on the occurred situations it is possible to execute logic at runtime, therefore, enabling Situation-Aware Management of running CPS applications. We prototypically implemented our approach in the language TOSCA with a Smart Home scenario.

This paper is structured as follows: In Section 2 we describe a motivating scenario and background. We describe our approach on integrating Situations and Management Processes in Section 3. Section 4 describes our prototypical implementation of the approach. Related work is discussed in Section 5. We conclude and outline future work in Section 6.

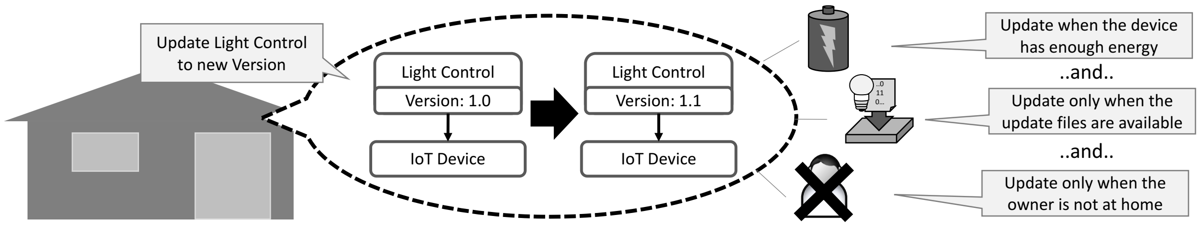
Update Light Control to new Version
Light Control
Version: 1.0
IoT Device
Light Control
Version: 1.1
IoT Device
Update when the device has enough energy
..and..
Update only when the update files are available
..and..
Update only when the owner is not at home

Figure 1: Motivating example scenario, depicting the need for CPS to adapt according to the current context.

2 Motivation & Background

This section describes our motivational scenario for Situation-Aware Management of CPS and a short overview of the SitOPT project that researched the adaptation of Workflows based on situations.

In industry and research the current trend of integrating physical entities that are not in the domain of IT with software components emerged to be paradigms such as IoT (Atzori et al., 2010; Gubbi et al., 2013) or the more general paradigm of CPS (Gunes et al., 2014). These paradigms are at the core a combination of software and hardware systems that monitor their physical and virtual environment based on sensors and act with the help of actuators connected. There is a wide range of scenarios to apply CPS, such as, healthcare (Haque et al., 2014; YIN et al., 2016), mobility (Guo et al., 2017; Zorzi et al., 2010) and energy (Andr et al., 2011; Shrouf and Miragliotta, 2015). Each system operates in a different domain but can overlap, e.g., a healthcare application to monitor high-risk patients can utilize mobility solutions to call an ambulance. Therefore, the integration of different CPS, hence, the combination of domains yields great benefits but is also one of the main challenges of CPS (Gunes et al., 2014), as with each combination these systems get more complex, error-prone and harder to maintain. The Reconfigurability (Gunes et al., 2014) of these systems must ensure that they can adapt themselves to changing requirements, environments and components at runtime. For example, a change on requirements regarding performance may need to scale up certain components, a change in the environment would start reconfiguration of the network, while a change of the components themselves, such as a new version, could change the configuration of the overall system. Therefore designing CPS in a context-aware manner (Perera et al., 2014) to enable such adaptations (Muccini et al., 2016) is a promising direction. According to the widely used definition by Dey context can be defined as "..any information that can be used to characterize the situation of an entity." (Dey, 2001), therefore a plethora of different data sets can be used to describe the situations in the context of a CPS, such, as location, environment, people, devices and activities, all with their describing attributes such as GPS coordinates, the type of environment, the roles of users, the capabilities of devices and running processes. In Figure 1 we depict a Smart Home scenario that needs an approach for Situation-Aware Management. A running CPS application running on a local device monitors movement in rooms and turns on light if movement is detected (See middle in Figure 1), therefore controlling the energy consumption. This system can may need update procedures as the Light Control application must enable the connection to new light systems or patch open security issues. However, the update, i.e., the adaptation of such a system should be achieved preferably in an autonomous manner and not interrupt the functionality when it is needed, e.g., updating when a user wants use functionality will disrupt the availability, thus, reduce customer satisfaction. Therefore, the update should only be started when a system owner is not at home, the device has enough energy to execute the update and still be able to run the application for an appropriate amount of time; and an update is available on the internet (See right hand side in Figure 1). This means a management system must be able to monitor different entities and their situations, such as, the location of people, the battery levels of devices and updates files on servers. To detect situations, context data combined with management operations must be incorporated which is a complex and error-prone task, as developers have to define fine-grained states in context and complex management tasks.

To cope with the challenge of modeling applications to be able to adapt to context the SitOPT (Wieland et al., 2015) project developed the concept of Situation-Aware Workflows. In SitOPT the basic concept for context was enhanced to so-called *Situations* modeled within *Situation Templates*, a modeling concept to enable the modeling of situations that occured to so-called *Things* (Häussermann et al., 2010; Hirmer et al., 2017). This abstraction allows modelers to define arbitrary situations for physical and virtual entities which encapsulate low-level context information and therefore ease the modeling of Context-, i.e., Situation-aware applications or the management of applications, on which our approach is based on.

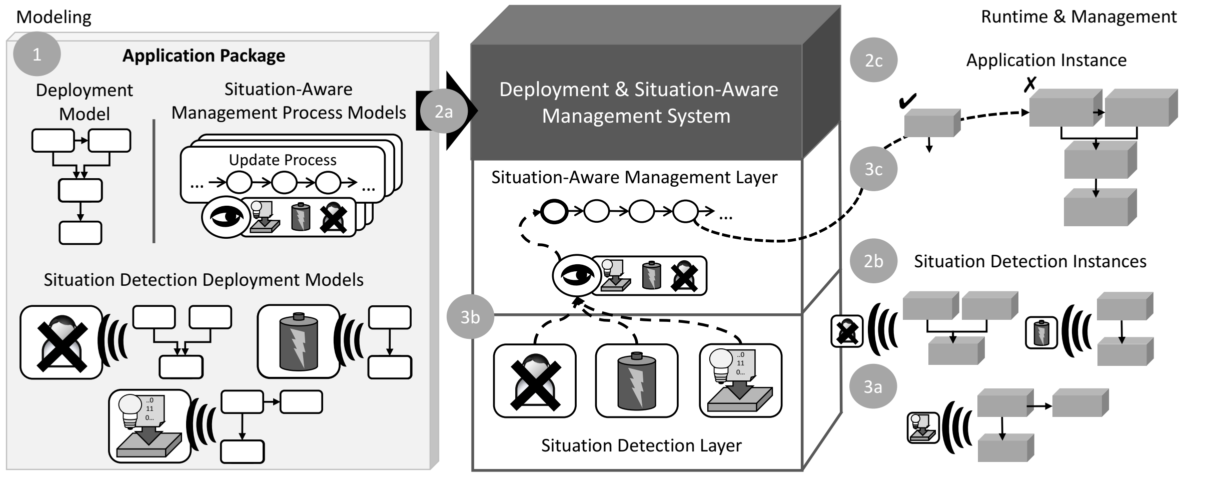
Modeling
Runtime & Management
1
Application Package
Deployment Model
Situation-Aware Management Process Models
Update Process
Situation Detection Deployment Models
2a
Deployment & Situation-Aware Management System
Situation-Aware Management Layer
3b
Situation Detection Layer
2c
Application Instance
3c
2b
Situation Detection Instances
3a

Figure 2: Overview of our Situation-Aware Deployment and Management System enabling Situation-Aware Management of applications.

3 Situation-Aware Management

In this section we describe an overview of our approach to enable Situation-Aware Application Management in Subsection 3.1, a detailed description of the modeling concepts in Subsection 3.2 and runtime aspects in Subsection 3.3.

3.1 Concept Overview

In the following we give an overview of our approach depicted in Figure 2. The basis are Application Packages (See 1 in Figure 2) that contain a Deployment Model, so-called *Situation-Aware Management Process Models* and *Situation Detection Deployment Models*. A Deployment Model specifies the structure of applications, e.g., the software-, hardware components and relations among each other. Additionally, the Situation-Aware Management Process Models define the possible tasks on an application instance, e.g., to backup the contents of state-full components, update components to new versions or scale up parts of the application. These processes are annotated with *Situation-Aware Process Triggers* that specify a certain set of situations that must hold to start a process, e.g., when an user is not at home, a device has enough energy or there is an update available. The second part of an application are Situation Detection Deployment Models that encapsulate the detection of certain situations. Such a model is another Deployment Model, as the applications' Deployment Model itself, but specifies a utility application that is able to detect the specified situations, e.g., an instance of SitOPT Middleware (Wieland et al., 2015) with suitable Situation Templates, a Python application reading the battery levels on a Raspberry Pi or a simple Cron job checking for software updates on the Internet. A Situation Detection Deployment Model therefore specifies the kinds of situations it can detect, these situations are referenced by Situation-Aware Management Process Models within the Situation-Aware Process Triggers.

After having modeled the Deployment Model, Processes and necessary Situation Detection Deployment Models the complete Application Package is uploaded to the Deployment & Situation-Aware Management System (See 2a in Figure 2). The system is responsible for creating instances from the given Deployment Model and Situation Detection Deployment Models by installing, configure and starting the modeled software components according to their specification, e.g., installing necessary packages, uploading and starting Python scripts on a Raspberry Pi. This deployment process starts with the creation Situation Detection Deployment Model instances, which is a mandatory to be started first as the detection of situations must be available (See 2b in Figure 2) before an instance of an application can be created (See Subsection 3.2 for more details). After instances of Situation Detection Deployment Models are available users can create instances of the application itself (See 2c in Figure 2), by requesting the system to install, configure and start software components specified within the applications' Deployment Model. Additionally, to creating such instances, the attached Situation-Aware Management Process Models deployed within the deployment system are bound against the situations detected by the system and therefore creating an instance of a Situation-Aware Process Trigger that will start a process attached to it when the specified situations occur, i.e., are active.

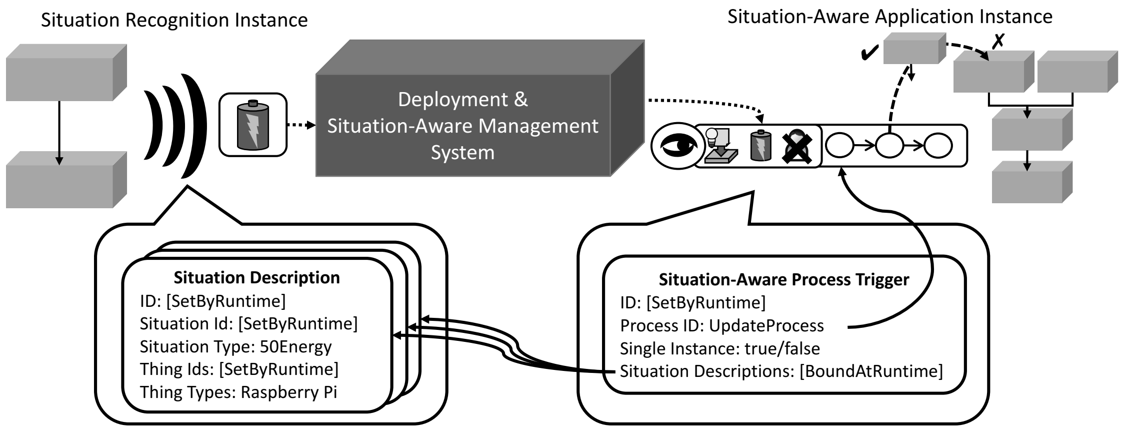
Situation Recognition Instance
Situation-Aware Application Instance
Deployment & Situation-Aware Management System
Situation Description
ID: [SetByRuntime]
Situation Id: [SetByRuntime]
Situation Type: 50Energy
Thing Ids: [SetByRuntime]
Thing Types: Raspberry Pi
Situation-Aware Process Trigger
ID: [SetByRuntime]
Process ID: UpdateProcess
Single Instance: true/false
Situation Descriptions: [BoundAtRuntime]

Figure 3: Modeling details of a Situation Detection Deployment Models and Situation-Aware Management Process Models.

The runtime aspects of the system are in general the following: (i) the running Situation Detection Model Instances are continuously monitoring their respective situations and send the current state to the Situation Detection Layer of the system (See 3a in Figure 2). As the situations are referenced within Situation-Aware Process Triggers a change of a situation state activates such a trigger and therefore starts an instance of the respective Management Process (See 3b in Figure 2). These Management Processes are invoked within the Situation-Aware Management Layer and interact with the running application to achieve the specified management tasks, such as, backing up, scale up and updating software components (See 3c in Figure 2)

3.2 Modeling based on Situations

In the following we describe the modeling of applications and their Situation-Aware Management Processes of our approach in detail (See Figure 3).

A Situation Recognition Deployment Model describes an application that enables to observe situations of relevant entities in the context of a system. The recognized situations are exposed in so-called *Situation Descriptions* (See left-hand side in Figure 3). A Situation Description contains a unique identifier for a single specific Situation Type (e.g. Situation Template in SitOPT) and also a set of unique identifiers for Thing Types. While a Situation Type specifies what kind of situation the Situation Recognition Model is able to observe, the set of Thing Types specifies what kind of things, e.g., a sensor, Raspberry Pi or people, are needed to observe the situation. A Situation Description basically defines what kind of situation can be detected by a Situation Detection Deployment Model. While Situation- and Thing Types are used to define the type of situations and things at runtime a description additionally specifies the runtime identifier of a situation as a *Situation Id* and for things a set of *Thing Ids* (e.g. a MAC address of a Pi)for which a situation detection is executed, i.e., a Situation Description is always bound against a concrete set of physical or virtual objects. When an application is deployed, each Situation Reference is bound against a concrete set of things and therefore a situation detection is started. The binding of things and situation against a reference can be achieved by different methods such as binding by an input value from the runtime at instantiation time of applications (indicated by an *[SetByRuntime]* for SituationId and Thing Ids in Situation Descriptions) or methods that try to automate the binding such as the work of Hirmer et al. (Hirmer et al., 2017).

To enable the triggering of processes we define *Situation-Aware Process Triggers* that model the binding between observed situations and management logic (See right-hand side in Figure 3). This binding is based on the references to Situation Descriptions and a Management Process used by Situation-Aware Process Triggers. A Situation-Aware Process Trigger has at least one reference to a Situation Description and exactly one reference to a Management Process. At runtime a set of referenced situations is evaluated in a conjunction, i.e., all of the observed situations are interpreted as a binary value which are then evaluated with an *AND*-function over the values.

To define on which Management Process is started we define the *Process ID* attribute (See right-hand side in Figure 3) on Situation-Aware Process Triggers. A Process ID references a Management Process by its identifier inside its application package. As there are different management tasks to be executed some may influence the availability of an application, e.g., a backup, update or test of an application component may directly influence the performance of the overall

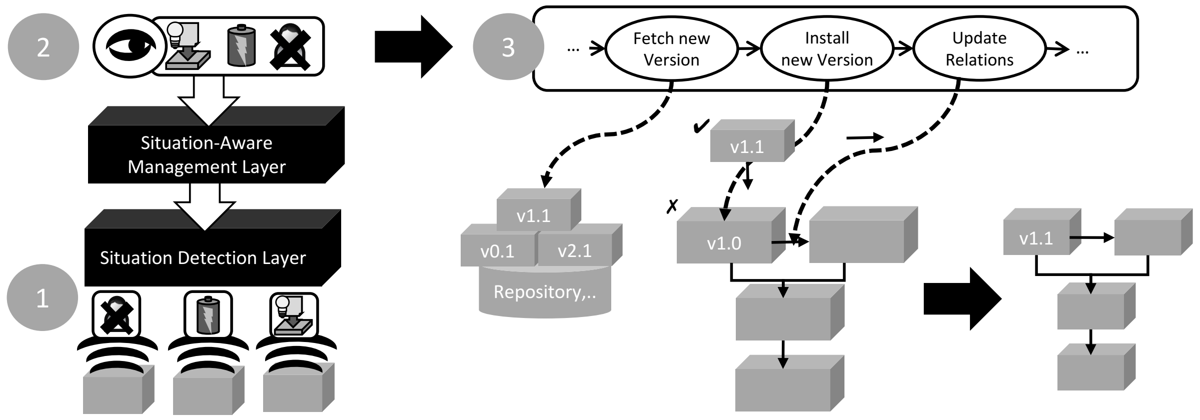

Figure 4: Details of adapting an application instance by Management Process in a Situation-Aware manner.

application. To control the execution of multiple instances of Management Process, e.g., because the situations change often and force the triggering of processes, we define the *Single Instance* attribute for Process References. This forces the runtime to allow the execution of a trigger only when there is no instance of a Management Process already running, enabling to control the number of executions of a trigger.

3.3 Using Situations at Runtime

In this subsection we describe the runtime aspects (See Figure 4), such as, registration of situations at the Situation Detection Layer, the observation by Situation-Aware Process Triggers and how Management Processes are invoked and adapt a running application. Before the Situation-Aware Management of applications is possible the Situation Recognition Deployment Models must be deployed and instantiated (See 1 in Figure 4), i.e., the Deployment and Situation-Aware Management System creates an instance of the models by creating, configuring and starting components defined inside the Situation Detection Deployment Models, e.g., installing packages and applications on a Raspberry Pi. However, the components that are responsible for the detection of the situations are configured to be bound against the Situation Detection Layer of the system to enable the observation of the current state of the defined situations. Note, that the binding between the Situation Detection Components of these Deployment Models must be able to communicate with the Situation Detection Layer, i.e., these detection components must use the available protocols, data formats and a have network connection at runtime, which in different scenarios may be a challenging technical issue, e.g., a pulling protocol from the cloud inside a Smart Home. Another aspect of the instantiation of Situation Detection Deployment Models is the binding of concrete things. The used identifiers must be unique for each thing observed, e.g., a MAC address and credentials identifying a Pi, a URI that specifies a software artifact, a name and birth date for persons. These identifiers can be given as input to the instantiation or use methods such as those by Hirmer et al. and Urbieta et al. (Hirmer et al., 2016; Urbieta et al., 2017).

After the deployment and instantiation, and therefore starting the observation of situations, the Situation-Aware Management Layer can use the observed situations to bind initially and trigger Situation-Aware Process Triggers (See 2 in Figure 4). The binding of a Situation-Aware Process Trigger against concrete situations, i.e., the observation of things and the possible situations their are in is done at instantiation time of the application that should be managed in a situation-aware manner. At instantiation time the concrete identifiers of the needed situations that are defined by the referenced Situation Descriptions of each Situation-Aware Process Trigger can be given, e.g., as input to the instantiation process or derived by more sophisticated methods such as semantically binding the situations to the application. After binding Situation-Aware Process Triggers to concrete situations the Situation-Aware Management Layer is responsible for observing the situations with the help of the Situation Detection Layer an fire triggers as specified, i.e., a Situation-Aware Process Trigger only starts a process if all of its situations are active. This means if a Situation-Aware Process Trigger is specified to observe, as in our motivational scenario, the availability of an update, of enough energy and users are not present, it will only trigger the update process when all of the situations are active.

The last part enabling the Situation-Aware Management of applications are Management Processes. These are triggered by the Situation-Aware Pro-

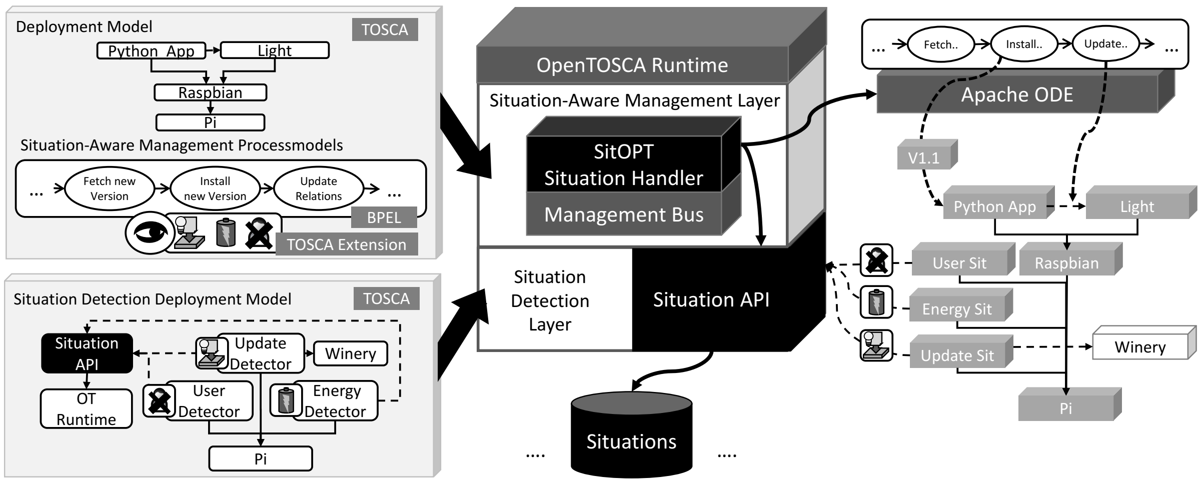

Figure 5: Overview of our prototypical implementation within the OpenTOSCA Ecosystem.

cess Triggers that are actively observed inside the Situation-Aware Management Layer. A Management Process can implement any kind of management task in any kind of language that is able to alter the configuration of applications at runtime. We propose the use of Workflow Technology that is proven in industry enabling properties such as transactional execution and compensation, parallel execution, recoverability, error handling or traceability (Leymann and Roller, 2000). For example, the update process of our motivational example can fetch the newest version of a component, install it and reconfigure needed component relations of application (See 3 in Figure 4).

4 Prototypical Implementation

In this section we describe our prototypical implementation based on the OpenTOSCA Ecosystem(Binz et al., 2013)[1] enabling the modeling and deployment of applications based on the language TOSCA(OASIS, 2013b; OASIS, 2013a; OASIS, 2015), which was originally created for Cloud Applications but also was used for IoT applications (Franco et al., 2017).

The motivating scenario was prototypical implemented based on different Python scripts that are deployed and started on a single Raspberry Pi (See upper and lower left-hand side in Figure 5). For the Light Control application we used a simple script that can periodically switch a power socket *On* and *Off* that was connected to a light. The Energy Monitoring Python application was prototypically implemented by checking whether the Pi is connected to a power outlet which implies that there is enough energy, hence, over 50%. To check the situation that there is an update file available a Python script continuously checked a repository of OpenTOSCA Winery[2] whether an archive with a new version of the Light Control is available. The AtHome check was simulated by altering the situations state periodically every minute, we plan to implement this situation recognition with real GPS values received from a Smartphone. The Update Process was implemented in the workflow language BPEL, which would check for the latest version of the light controlling Python script, fetch it, upload it to the Raspberry Pi and exchange it with the old script.

After implementing the necessary scripts to control the light and observe the necessary situations we modeled a TOSCA application for the situation recognitioning and application itself and extended the Runtime OpenTOSCA Container[3] with the Situation Detection Layer and Situation-Aware Management Layer to be able to hande the applications. The TOSCA applications used newly created generic Situation Detection Component as a TOSCA Node Type (i.e. Component Type) that can register Situations based on Situation Description and modeled properties such Situation Type, Thing Types, Situation Id, Thing Ids and receive an endpoint to the Situation Detection Layer. Each Python Script is able to receive endpoint data from the modeled Situation API component inside their deployment models (See blackbock in the left of Figure 5) and would receive the endpoint to send the current state of the situation they are observing. The Situation Detection Layer was im-

[1]https://www.github.com/OpenTOSCA/

[2]https://projects.eclipse.org/projects/soa.winery

[3]http://opentosca.github.io/container

plemented by the Situation API which allows to store Situation Descriptions and manipulate the state of the registered situations (See middle in Figure 5). To enable the triggering of processes based on situations we implemented the Situation-Aware Management Layer on top of the already available Management Bus of the runtime which already could invoke Management Process implemented in BPEL. The Situation-Aware Management Layer was implemented by integrating the already available Management Bus of the OpenTOSCA Runtime with the Situation Handler component of the SitOPT project (Wieland et al., 2015) (See middle in Figure 5). In SitOPT the Situation Handler was responsible for enabling the registration of notifications when situations changed or invoke services based on situations, i.e., invoking different service implementations when different situations were active. Based on this integration the Situation-Aware Process Triggers, which were mapped as TOSCA application properties, are registered as notifications to the Management Bus of OpenTOSCA which can invoke the processes specified in the TOSCA applications. These processes are implemented in the workflow language BPEL (OASIS, 2007) and are deployed on and executed in the workflow engine Apache ODE (See upper right in Figure 5). The necessary runtime information, such as, Situation Ids are requested by the extended OpenTOSCA Runtime when an application is instantiated by reading it from the input.

5 Related Work

In this section we present related work on the domains of CAS (Baldauf et al., 2007), application provisioning and management, and adaptation. There are multiple surveys on adaptability (Mens et al., 2016; Da et al., 2012; Kakousis et al., 2010; Muccini et al., 2016) and describe different methods for adapting application at runtime, but in the following we focus on (Context-Aware) adaptation of applications.

The domain of Adaptation Systems has broad application areas because of the wide-range of factors that influence a system to adapt itself. According Da et al. (Da et al., 2012) there are three types of adaptations: Reactive Adaptation, Evaluative Adaptation and Adaptation for Integration. Reactive Adaptation is used to react on changes in the environment, such as, network changes or user preferences. Evaluative Adaptation is the category of adaptations that influence the functionality, performance and error handling of a system. The last adaptation type is called Adaptation for Integration that tries to overcome the issue of incompatible components of software and hardware, e.g., by changing implementations or protocols between components. All of the three types can be implemented based on the proposed approach inside Management Processes.

Baldauf et al. (Baldauf et al., 2007) present a survey about Context-Aware Systems focusing on architectures and design principles, followed by an overview of Context-Aware Middlewares. They introduce a layered conceptional architecture that has sensors on the lowest layers, which are read by the next layer of data retrieval. Raw data is then passed to the Preprocessing layer where the results are stored in the Storage/Management Layer, that can be used by the last layer called Application layer. Our proposed approach can be categorized according to Baldauf et al. as our approach has the Situation Detection Layer and the Situation-Aware Management Layer on top.

Muccini et al. (Muccini et al., 2016) present a Systematic Literature Review on Self-Adaptation of CPS. They investigated the use of adaptation techniques in CPS, such as, concerns, on which area these are applied or their domains. One of the insights given by Muccini et al. is that most approaches are used for performance optimization and increased flexibility. Most of the techniques targeted the application layer instead of the infrastructure. In the investigated works the used techniques used different feedback loop mechanisms, with MAPE (Monitor, Analyse, Plan and Execute) functions being the most used ones. From the found works and analysis a layered adaptation model consisting out of an Application layer on top of Context Management Layer that uses the underlying Physical Layer is proposed. Our approach can be categorized into the same layers, although we don't use on the MAPE technique, as used in most approaches according to Muccini et al.

Saller et al. (Saller et al., 2013) present an approach for the context-aware adaptation of dynamic software product lines based on extended feature models. A feature model enables to specify a set of features and relations among each other, e.g., a feature is an alternative to another, a feature requires another feature or is optional. Based on such feature models Saller et al. extend them based on a Context Model where a mapping between the features and context entities is defined to state whether a feature can be used under a certain context. These models are used to adapt applications at runtime by monitoring the context and adapting the currently active features. In contrast to our approach Saller et al. adapt the application model directly while our approach uses Management Process that also allow to execute management logic that is not limited to reconfigurations of the application but also enables tasks, such as, storing backups.

Breitenbücher et al. (Breitenbücher et al., 2015) present an approach to enable the modeling of workflow models in a situation-aware manner. The basic idea is to enable the modeling of events and areas in the control flow of a workflow model that either start the execution of activities or enable the execution of activities only when certain situations occured or are active. This approach is build on so-called Situation Events and Situational Scopes. A Situation Event is an event trigger inside a workflow model that can be connected to other activities inside the control flow and when activated, i.e., a situation is active, triggers the execution of the connected activities. To enable the execution of a group of activities only when a certain situation is active, Breitenbücher et al. introduce Situation Scopes that enable to specify a set of activities inside the workflow models' control flow. These concept enable to model workflows to be situation-aware and therefore enable modelers to add adaptation logic directly into their models. The main difference to our approach is that the Situation-Aware Adaptation is focused on the control-flow of applications, while our approach is focused on adapting application structures. A combination of both approaches could make the Management Processes within our approach adapt at runtime.

The ACCADA framework introduced by Gui et al. (Gui et al., 2009) targets the context-aware deployment and adaptation of software components. These software components are monitored in a continuous loop by an Event Monitor that informs a Structure Modeler responsible for composing the different components in the system on a functional level, such as, binding the components according to their dependencies. As the Structure Modeler is only responsible for composing the components based on this functional level a set of Context-specific Modelers are responsible for adapting the structure of the components and relations based specific context-related constraints. These Context-specific Modelers build therefor Context-Specific Architecture/Deployment Models based on the current context. After determining such a context-specific model these are send to an Adaptation Actuator that is responsible for changing the current system structure according to the determined model. Our approach is similar to the approach of ACCADA in the way that we adapt the application structure based on changes in the environment, but additionally we enable to execute Management Process enabling activities, such as, backing up data in the cloud instead of only enabling structural changes of software components.

Anthony et al. (Anthony et al., 2009) present their Context-Aware Adaptation techniques in their system called DySCAS. The conceptual parts of their approach is based on so-called Decision Points that are embedded in the software components and configured by loading Policies into the components. These policies are responsible for subscribing to context information and adapt the software component they are loaded in by using a sandbox environment inside the components themselves. The supporting DySCAS Middleware is responsible for monitoring and managing the overall system while the software components themselves are responsible for the reconfiguration based on context. The main difference to our approach is the point of adaptation while, DySCAS adapts single components our approach enables to adapt the whole structure of the system, and additionally, doesn't force the software components to be altered to enable the loading of policies, i.e., our approach allows the use of Off-the-Shelve components.

La et al. (La and Kim, 2010) present a framework for the context-aware provisioning of services for mobile devices. The main goal of their framework is to adapt the binding between mobile applications and services on the cloud based on the current context of the user and their mobile device. In their framework a Context Collector running on a mobile device gathers data to determine the current context, such as, device, service or user preference context. When the context changes the framework from La et al. adapts the binding of a mobile application and the used cloud services by either simply rebinding it or using different adapters to adapt the binding to the used services. These adaptation are needed for example when the QoS of the different services change or certain functionalities are not available by a service in the current context, and therefor must be extended by adapters. In contrast to our approach the presented framework is only used to adapt the binding between components based on the current context.

Alcarria et al. (Alcarria et al., 2017) present an approach to enable Context-Aware CPS to enable a seamless service provisioning. This is achieved by modeling an abstract process, e.g., in BPEL (OASIS, 2007), that contain activities invoking various services based on their service description. The selection of a concrete service running on a device is based on situations in the context of the possible devices and therefore enables the Context-Aware Adaptation of the running process. In contrast to our approach Alcarria et al. focus on the adaptation of Management Processes itself, while our approach focuses on executing processes at the right time.

Bucchiarone et al. (Bucchiarone et al., 2012) present an approach to enable the dynamic adaptation of context-aware business processes. The approach uses a high-level business process where its abstract activities are annotated with goals, and based on these goals concrete and finer-grained business process fragments are selected in a context-aware manner to implement the abstract activities. This approach allows even the integration between lower-level fragments as these can also contain abstract activities that can be implemented by a fragment at runtime through a context-aware selection. In contrast to our approach it is possible to generate process from the fragments to implement various management tasks based on defined goals, while we reused already available Management Processes.

6 Conclusion

We presented an approach to enable the Situation-Aware Management of Cyber-Physical Systems. The basic idea of the approach was to reuse concepts of Context-Aware Systems and Management Processes enabling processes to be executed when certain situations occur inside the context of an application. Situations encapsulate low-level context data into higher level knowledge, easing the modeling of Context-Aware, i.e., Situation-Aware Systems as fine-grained states inside the application context don't need to be specified on the application level. The high level situations are observed by instances Situation Detection Deployment Models to recognize situations in the context of applications and making the information available to Situation Detection Layer of the Deployment & Situation-Aware Management System. In the system these situations are observed by Situation-Aware Process Triggers that are observed and fired inside the Situation-Aware Management Layer. A Situation-Aware Process Trigger references a certain set of situations that it reacts to when situations are either active or not. When the set of situations are in a state that fit the specification of a Situation-Aware Process Trigger a referenced Management Process is started. These Mangement Processes are bundled within the Application Packages and are able to change the structure, configuration and execute mangement tasks of the application at runtime, therefore implementing a wide range of management functions. Hence, these processes enable the adaptation of applications at runtime and with the combination of the Situation-Aware Trigger are executed in certain situations, hence, enable the Situation-Aware Management of applications.

In the future we want to extend the presented approach with concepts of timing and compensation, e.g., to predict when a situation will change and c changes of a Management Process. To properly adapt a CPS the role of time is crucial as certain situations can change often and therefore starting Mangement Processes and their compensation can lead to unwanted errors, such as, aborting a flashing process of an embedded system. Therefore incorporating the prediction of situations, i.e., predicting when a situations occurs or not can be beneficial to a more timely execution of Management Processes.

Acknowledgement

This work was funded by the DFG project DiStOPT (252975529).

REFERENCES

Alcarria, R., Bordel, B., and Jara, A. (2017). Flexible service provision in context-aware cyber-physical systems. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 873–883. Springer.

Andr, C., Quijano, N., and Mojica-nava, E. (2011). A Survey on Cyber Physical Energy Systems and their Applications on Smart Grids. *2011 IEEE PES Conference on Innovative Smart Grid Technologies (ISGT Latin America).*

Anthony, R., Chen, D., Pelc, M., Persson, M., and Torngren, M. (2009). Context-aware adaptation in DySCAS. *Communications*, 19(Context-Aware Adaptation Mechanism for Pervasive and Ubiquitous Services (CAMPUS)).

Atzori, L., Iera, A., and Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*.

Baldauf, M., Dustdar, S., and Rosenberg, F. (2007). A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., and Wagner, S. (2013). OpenTOSCA - A Runtime for TOSCA-based Cloud Applications. In *Proceedings of the 11th International Conference on Service-Oriented Computing (ICSOC 2013)*, pages 692–695. Springer.

Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., and Wieland, M. (2015). A situation-aware workflow modelling extension. In *Proceedings of the 17th International Conference on Information Integration and Web-based Applications & Services*, iiWAS '15, pages 64:1–64:7, New York, NY, USA. ACM.

Bucchiarone, A., Marconi, A., Pistore, M., and Raik, H. (2012). Dynamic adaptation of fragment-based and context-aware business processes. *Proceedings - 2012 IEEE 19th International Conference on Web Services, ICWS 2012*, pages 33–41.

Da, K., Dalmau, M., and Roose, P. (2012). A Survey of Adaptation Systems. *International Journal on Internet and Distributed Computing Systems*, 2(1):1–18.

Dey, A. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

Franco, A. C., Breitenbücher, U., Hirmer, P., Képes, K., Kopp, O., Leymann, F., Mitschang, B., and Steinke, R. (2017). Internet of Things Out of the Box : Using TOSCA for Automating the Deployment of IoT Environments. *Proceedings of the 7th International Conference on Cloud Computing and Services Science*.

Gubbi, J., Buyya, R., Marusic, S., and Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*.

Gui, N., De Florio, V., Sun, H., and Blondia, C. (2009). ACCADA: A framework for continuous context-aware deployment and adaptation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 5873 LNCS:325–340.

Gunes, V., Peter, S., Givargis, T., and Vahid, F. (2014). A survey on concepts, applications, and challenges in cyber-physical systems. *KSII Transactions on Internet and Information Systems*.

Guo, Y., Hu, X., Hu, B., Cheng, J., Zhou, M., and Kwok, R. Y. K. (2017). Mobile Cyber Physical Systems: Current Challenges and Future Networking Applications. *IEEE Access*, XX(c):1–1.

Haque, S. A., Aziz, S. M., and Rahman, M. (2014). Review of cyber-physical system in healthcare. *International Journal of Distributed Sensor Networks*, 10(4):217415.

Häussermann, K., Hubig, C., Levi, P., Leymann, F., Siemoneit, O., Wieland, M., and Zweigle, O. (2010). Understanding and Designing Situation-Aware Mobile and Ubiquitous Computing Systems. *World Academy of Science, Engineering & Technology*.

Hirmer, P., Breitenbücher, U., da Silva, A. C. F., Képes, K., Mitschang, B., and Wieland, M. (2016). Automating the Provisioning and Configuration of Devices in the Internet of Things. *CSIMQ*, 9:28–43.

Hirmer, P., Wieland, M., Schwarz, H., Mitschang, B., Breitenbücher, U., Sáez, S. G., and Leymann, F. (2017). Situation recognition and handling based on executing situation templates and situation-aware workflows. *Computing*, 99(2):163–181.

Kakousis, K., Paspallis, N., and Papadopoulos, G. A. (2010). A survey of software adaptation in mobile and ubiquitous computing. *Enterprise Information Systems*, 4(4):355–389.

Knappmeyer, M., Kiani, S. L., Reetz, E. S., Baker, N., and Tonjes, R. (2013). Survey of context provisioning middleware. *IEEE Communications Surveys and Tutorials*, 15(3):1492–1519.

La, H. J. and Kim, S. D. (2010). A Conceptual Framework for Provisioning Context-aware Mobile Cloud Services. In *2010 IEEE 3rd International Conference on Cloud Computing*.

Leymann, F. and Roller, D. (2000). *Production Workflow: Concepts and Techniques*. Prentice Hall PTR.

Mens, K., Capilla, R., Cardozo, N., and Dumas, B. (2016). A taxonomy of context-aware software variability approaches. *Companion Proceedings of the 15th International Conference on Modularity - MODULARITY Companion 2016*, (March):119–124.

Muccini, H., Sharaf, M., and Weyns, D. (2016). Self-adaptation for cyber-physical systems. In *Proceedings of the 11th International Workshop on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '16*.

OASIS (2007). *Web Services Business Process Execution Language (WS-BPEL) Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OASIS (2013a). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Primer Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OASIS (2013b). *Topology and Orchestration Specification for Cloud Applications (TOSCA) Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS).

OASIS (2015). *TOSCA Simple Profile in YAML Version 1.0*. Organization for the Advancement of Structured Information Standards (OASIS).

Perera, C., Zaslavsky, A., Christen, P., and Georgakopoulos, D. (2014). Context Aware Computing for The Internet of Things: A Survey. *Communications Surveys Tutorials, IEEE*, 16(1):414–454.

Saller, K., Lochau, M., and Reimund, I. (2013). Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. *Proceedings of the 17th International Software Product Line Conference co-located workshops*, pages 106–113.

Shrouf, F. and Miragliotta, G. (2015). Energy management based on Internet of Things: Practices and framework for adoption in production management. *Journal of Cleaner Production*.

Urbieta, A., González-Beltrán, A., Ben Mokhtar, S., Anwar Hossain, M., and Capra, L. (2017). Adaptive and context-aware service composition for IoT-based smart cities. *Future Generation Computer Systems*.

Wieland, M., Schwarz, H., Breitenbücher, U., and Leymann, F. (2015). Towards situation-aware adaptive workflows: SitOPT - A general purpose situation-aware workflow management system. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2015*.

YIN, Y., Zeng, Y., Chen, X., and Fan, Y. (2016). The internet of things in healthcare: An overview. *Journal of Industrial Information Integration*, 1:3 – 13.

Zorzi, M., Gluhak, A., Lange, S., and Bassi, A. (2010). From today's INTRAnet of things to a future INTERnet of things: A wireless- and mobility-related view. *IEEE Wireless Communications*.