ContextPoint: An Architecture for Extrinsic Meta-Adaptation in Smart Environments

Christian Piechnick, Sebastian Richly, Thomas Kühn, Sebastian Götz, Georg Püschel and Uwe Aßmann
Software Technology Group, Technische Universität Dresden,
Dresden, Germany
Email: {christian.piechnick, sebastian.richly, thomas.kuehn3, sebastian.goetz1, georg.pueschel, uwe.assmann}@tu-dresden.de

***Abstract*—The establishment of mobile devices had a high impact on the use and development of software systems. It is expected that the ability to automatically adapt to changing environments will be a crucial property for future apps running on mobile devices. The problem with current approaches for self-adaptive systems is that developers must define the adaptive behaviour at design-time. In many cases, however, the developer cannot predict all situations at design-time, which should trigger adaptation at runtime. Furthermore, applications for mobile devices are usually optimized for a small set of use cases and have a narrow, well-defined scope. In order to support more complex tasks, the functionality of several apps has to be composed dynamically. Another problem arises from the ever increasing number of available applications. In this paper, we address these problems by proposing a novel infrastructure for self-adaptive systems in smart environments, namely ContextPoint. Our goal is to describe an architecture which supports unanticipated adaptation for single systems, as well as the automatic integration of actuators and sensors, situated in the environment, with services and data from both, mobile devices and the cloud. Therefore, a distributed adaptation technique is proposed, where adaptation logic and rules are provided by the environment itself. This decentralization simplifies the development of self-adaptive systems with dynamic adaptive adaptation processes (meta-adaptation) and, thus, the design and operation of systems with unanticipated adaptation. Furthermore, our approach provides means for describing context-dependent collaboration between varying systems enabling the design of ad-hoc system-of-systems.**

Keywords*—*Adaptation; Self-Adaptive; Meta-Adaptation; Architecture; Context-aware; Location-aware.

I. Introduction

The wide-spread acceptance of mobile devices (e.g., smartphones, tablets) changed the development as well as the use of software applications radically. Because applications running on mobile devices change their location and, hence, their environmental situations they are used in, they have to adapt their appearance and behaviour accordingly. This kind of flexibility is commonly called context-aware adaptation in self-adaptive systems. Currently, such systems rely on an environmental model (i.e., context model), which describes the entities of the execution context and their relationships. In these models, software engineers predefine statically at design-time which contextual information can be observed at runtime. At runtime a MAPE-K-Loop [1] (a) *monitors* the environment using sensors, (b) *analyses* the gathered data to instantiate the context meta-model, (c) *plans* necessary reconfigurations, and (d) *executes* the chosen plans. The main problem, however, is that software developers usually cannot predefine all environmental entities, which could be important for an adaptation process at runtime, at design-time. Lets consider an application that mutes a smartphone automatically, every time the user must not be disturbed (e.g., the user is participating in a meeting). The fact, however, that a user would be disturbed by a ringing smartphone is highly individual. A developer of such an application could most possibly not foresee all individual cases (e.g., when a baby is sleeping within the same room the user is located). To address this problem, the adaptation process itself should adaptable.

Another consequence arising from the characteristics of mobile devices is the change in size and range of functions. Traditionally, software system for stationary devices increased in their code size and complexity. The goal was to create multi-purpose systems with a huge set of provided functionality. Applications for mobile devices, henceforth denoted as *apps*, reversed that trend. They usually have a narrow scope with a rather small set of offered functionality. Those apps are optimized for a well-defined set of tasks. We call this *Functional Separation of Concerns* (F-SoC). In order to support more complex workflows, several apps have to work together to combine their provided functionality in a seamless way. Currently, there is no mechanism to describe an overall workflow across multiple apps on mobile platforms. On Android, for example, it is only possible to exchange data between applications using intents. Intents restrict an application' access to another application's provided services or data. Those intents are specified using coarse-granular classes of access types defined in the system frameworks. The problem is that developers would need to agree on a shared set of guidelines (like datatypes) in order to establish a seamless integration, which is hard to enforce for domain-specific applications from different domains. Through the establishment and spread of devices for smart environments, this kind of functionality becomes even more important. Since sensors and actuators become cheaper and more standardized, they can be placed easily in any kind of environment. It is likely that, in the future, users want to integrate their apps on multiple mobile devices seamlessly with services provided by both the environment and via Internet, depending on their current situations.

The main problem resulting from the F-SoC expansion is the huge number of similar apps developed for different tasks or usage-scenarios. The Google Play Store contain almost one million apps each. After the user found an appropriate app in this huge variaty of offers, it has to be installed manually and, in many cases, be configured. In order to increase convenience and efficiency, regarding the usage of mobile devices, it should be possible to automatically detect a set of apps that are well-suited to support a user's task in a given situation and automatically deploy, configure and connect those applications.

The mentioned requirements for context-aware, mobile app-infrastructures can be summarized as follows. Apps have to support:

R1 **unanticipated adaptation** by *meta-adaptation*, i.e., the adaptation mechanisms need to be adaptable themselves, because the developer cannot foresee all possible situations the application will operate in.

R2 **runtime composition**. In order to support complex tasks, apps have to be combined to ad-hoc systems of systems (SoS). Those apps can either be executed on the same device or be integrated as services in a distributed system.

R3 **automatic provisioning**. Based on the context and the task of a user, a collection of suitable apps have to be provided.

In this paper, we present an approach which satisfies all three requirements by introducing the concept of extrinsic meta-adaptation in a self-adaptive control loop. Usually, the mechanisms required for self-adaptive systems are implemented statically. Even though, many approaches propose architectures to support adaptive monitoring and analysis by distributing those steps to a varying network of collaborating systems, *plan* and *execute* are usually fixed in their implementation. Even though every self-adaptive systems relies on a component model that can theoretically be extended at runtime, currently there is no process how to dynamically extend the knowledge base used for adaptation as well as how to adapt the adaptation process itself. Furthermore, systems with a flexible monitor and analyze phase are able to dynamically extend the information sources that are used for decision making, while the strategies how those information are processed remain fixed. A self-optimizing system for non-functional properties for example, might at runtime extend the information required to perform optimization and include new components that can be used by the planning component, but will still only optimize non-functional properties. In situations where it is necessary to adapt for self-healing or functional requirements, such a system will fail. Meta-adaptation allows to adapt the adaptation process itself (e.g., introduce new plan and execute logic, etc.) at runtime. We want to propose an adaptation mechanism where apps on mobile devices can automatically be adapted, connected, and provided from an environmental infrastructure. Because the meta-adaptation is provided by the environment itself (i.e., extrinsic), the overall adaptation process can be extended at runtime by changing the location without redeployment of the entire self-adaptive system.

This paper is structured as follows: In Section II, we give a short introduction to the Smart Application Grids (SMAGs) approach, which is used as a basis for the approach presented in this paper and introduce the concept of meta adaptation in Section III. We discuss our approach in Section IV. Section V provides an evaluation of the presented concept using an example. In Section VI, related work is presented. Finally, Section VII presents our conclusion and future work.

II. Smart Application Grids

In order to dynamically adapt an application to varying contexts, the structure and behaviour of an application has to be changed at runtime. Hence, the application architecture has to be *variable* and *extensible*. Variability enables the adaptation of existing behaviour at runtime within a given variability space. In contrast to that, extensibility allows to scale the variability space and build the foundation for Meta Adaptation.

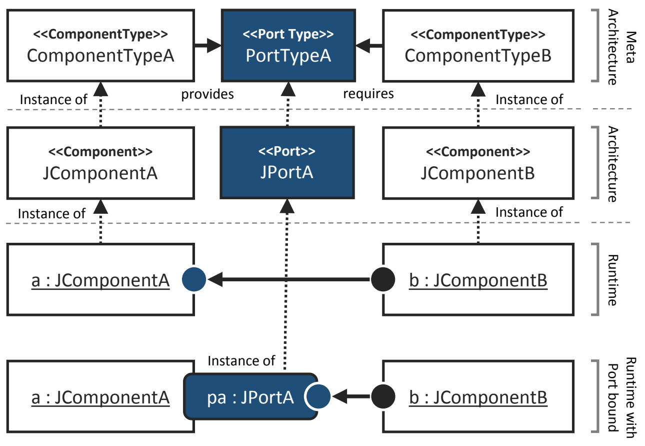

Fig. 1. The Meta-Levels of Smart Application Grids

Traditionally, variability is assured by applying the *Template Hook Meta Pattern* [2]. The application logic is separated in a fixed (template) and variable part (hook). By exchanging the hook at runtime, an application's behavior is adapted dynamically. This very simple procedure introduces three problems. First, the replacement of the complete hook can be expensive concerning resource usage and replacement time. Second, the system might be in an inconsistent state during reconfiguration. Third, if the hook is stateful, the state has to be migrated to the new hook, which can be expensive as well. Because in many designs, the hook can also have external references, both incoming and outgoing, those references have to be migrated, too. To tackle those shortcomings, invasive composition techniques (i.e., Aspect Oriented Programming) for dynamic context-aware adaptation were investigated [3]. With runtime weaving it is possible to exchange program code in a very fine-grained manner during the lifetime of an application. Still, aspects introduce some problems as well. First, it is a code-composition technique on meta-layer M1 [4], i.e., the class-layer. Consequently, it is only possible to change the behaviour of all objects instantiated by a given class [5]. However, for many scenarios it is necessary to have an adaptation technique on meta-layer M0, i.e., the object-layer, which allows to change the behaviour of individual objects. Another major drawback of aspects is that they are a white-box composition technique (i.e., aspects rely on the internals of the application subject to adaptation) which decreases reusability. Consequently, reusing adaptive behaviour across different applications and domains is insufficiently supported. In order to support invasive software composition on an architectural (component) level, we developed our Smart Application Grids (SMAGs) framework. SMAGs is a model-driven, platform independent design and operation principle for fine-grained, dynamic and unanticipated adaptation with a focus on increased reuse. SMAGs consists of many small, distributed applications that are linked dynamically. Role-Based Design and Programming [6] is used to change the structure and behaviour of individual applications as well as to express dynamically varying relationships across several distributed applications. A *role* is a dynamic service of an object in a specified context, offering the possibility to express separation of concerns, interface-structuring, dynamic collaboration description, and access restriction [7]. Roles are played by objects, dynamically altering the structure and behaviour of the player. In other words, roles enable an adaptation technique on meta-layer M0, the object-layer. On the one hand, roles can extend the object's state and functionality (i.e., introduce new methods and/or attributes). On the other hand, the objects existing behaviour can be changed. Furthermore, roles contain references to other roles. Since roles can be played and discarded at runtime, they are capable of expressing dynamically changing relationships across multiple system entities (e.g., objects, components, etc.). The major difference to other invasive composition approaches (e.g., aspects) is that roles are played within a context (e.g., "a person plays the student role within the context of an university"). This tight coupling between behaviour and environmental conditions makes Role-Based Design a powerful approach to model *Self Adaptive Systems* (SAS).

As depicted in Figure 1, at design-time, a platform independent **Meta Architecture** defines *Component Types* which specify provided and required *Port Types*. A Port-Type represents an interface specification. From several Meta Architectures a platform specific architecture can be derived, describing *Components* implementing *Component Types* and *Ports* implementing Port-Types. Ports can be grouped into *Port Models*. Each Port Model is associated with a binding to components and with environmental conditions, stating when it should be integrated into the application. At runtime, Components can be instantiated and connected by their matching required and provided Port Types. With the instantiation of Ports/Port Models and their binding to Components, the behaviour of individual *Component Instances* as well as the structure of the overall application can be adapted according to a given context. For more detailed information we refer to [6].

The SMAGs approach proposes an adaptation architecture. In [6], an overview of this architecture is presented. It represents a *MAPE-K loop* with the Sensor Layer monitoring the environment and transferring the gathered information to a *Context Model*. An *Inference Layer* relates existing and deduces new information based on the data in the Context Model. An *Adaptation Layer* creates reconfiguration plans based on the information from the Context Model and the current configuration of the adaptive application. Those plans are then executed by a Runtime Environment. Because the adaptation architecture itself is a SMAGs-based application, the concrete implementations (e.g., the Context Model representation etc.) can be changed at runtime. This adaptive adaptation architecture enables Meta Adaptation.

Furthermore, the SMAGs approach is based on a distributed repository infrastructure. A repository can be used to store the meta-architecture and architectural information as well as component and port implementations. These artefacts can either be reused at design-time for the design and implementation of new systems or at runtime to extend a running application with new components and ports. Additionally, each repository exposes a *Service Trader*. Applications can register remotely to offered functionality alongside with contextual information at the Service Trader. Other applications can query the published services to autonomously create dynamically varying SoS. The SMAGs approach is used to model a novel adaptation paradigm for unanticipated adaptation in mobile scenarios.

III. Extrinsic Meta-Adaptation

Adaptation mechanisms for context-aware software system can be classified into *parametrised*, *control-flow based* and *compositional* adaptation [6]. For parametrised adaptation the application units expose predefined parameters that can be changed at runtime. Control-flow based adaptation triggers the execution of application-specific behaviour to react on environmental changes. Compositional adaptation allows to change the structure of the application (e.g., create, remove, or reconnect components, etc.). The available components as well as the provided composition operators define which variants of the system are valid configurations, constituting the *variant space*. The other key modelling element, is the adaptation strategy that describes *when* adaptation has to be triggered and *how* the system should be reconfigured in a given situation. Therefore, a context metamodel describes all types of contextual information that may be available at runtime. Whenever the concrete context model changes, the system checks whether or not one of the variants within the variant space is better suited than the current system configuration. When the system detects a better alternative, a reconfiguration plan is generated. However, the main problem for software developers is that they cannot foresee all possible conditions that should trigger adaptation as well as all other systems, with which the application might collaborate. When the variant space as well as the adaptation strategy is fixed, unplanned situations cannot be handled. In order to support adaptation in unanticipated situations the adaptation process itself must be variable and extensible. This concept of adapting the adaptation is called meta-adaptation [8]. Figure 2 shows how meta-adaptation can be achieved by extending (1) either the application's variability space or (2) the adaptation logic itself.

Variability-Space The variability space (Figure 2 top-right) defines which variants of the system exist. In theory, an adaptation process investigates all different alternatives to decide whether or not there is a better configuration of the system w.r.t. the current environmental conditions. The variability space is constructed using all possible component/port combinations and is constrained by architectural templates and rules. Adding new artefacts or changing existing ones changes the variability space at runtime (Extension). This allows to create new variants, not considered at design-time, satisfying dynamic requirements the developers could not foresee.

Adaptation Process When the implementation of a MAPE-K loop itself provides variability and extensibility, the adaptation loop itself can be reconfigured (Figure 2 bottom-right). In the SMAGs approach, the MAPE-K loop is implemented using the same composition system as the system it is adapting. By this design, it is possible to adapt the adaptation architecture, which enables meta-adaptation. Figure 2 shows a second MAPE-K loop that uses the Runtime Environment of the adapted application as an executor for the reconfiguration plan. The second loop can be implemented in any application. This allows, for example, to exchange the context model representation, to introduce parallel representations with different characteristics (e.g., probability) of the context model, to introduce new sensors, extend the inference mechanisms or introduce new planners and executors.

SMAGs supports both dimensions of meta-adaptation. The repository infrastructure enables applications to extend the

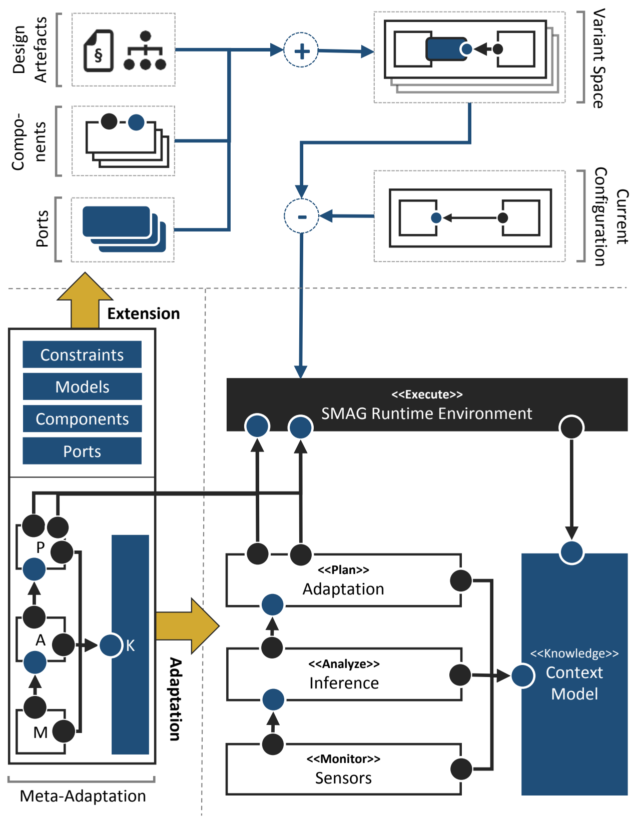

Fig. 2. The SMAGs Meta-Adaptation Process

variability space at runtime, by importing new model artefacts (i.e., (Meta) Architectures and Port Models) as well as implementation artefacts (i.e., Components and Ports). Furthermore, SMAGs supports runtime reconfiguration of the adaptation process because the MAPE-K loop is itself implemented as a SMAGs application. This enables the developer of meta-adaptive applications to change parameters of existing MAPE-K loop components as well as to change the structure of MAPE-K loops. Furthermore, the introduction and binding of new ports can extend or change the behaviour of existing components. This allows to dynamically adapt the context model representation (e.g., add the concept of uncertainty to specific model entries) and to bind according inference strategies.

These two dimensions build a foundation to create location-aware unanticipated adaptation by integrating meta-adaptive systems within the environment. Whenever a mobile application is situated in the same location, the meta-adaptive system can provide location-specific adaptation knowledge.

IV. The ContextPoint

In the research area of context-aware and self-adaptive systems, still no common definition of the term *context* was established. The most accepted and used definition was given by Anind K. Dey in 2001:

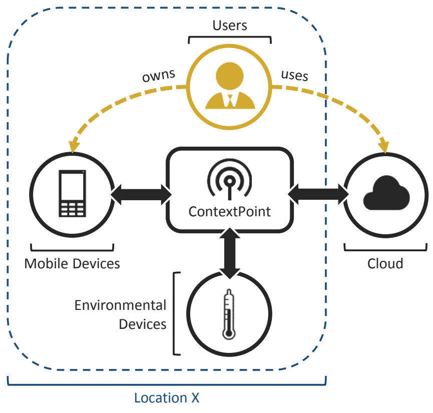

Fig. 3. The Context-Diagram of ContextPoint

"Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves." [9]

Over the last decades many conceptual frameworks for context modelling and interpretation were developed [10] [11] [12] [13]. One commonality is that every definition and conceptualization of the term context treats the location of both the user and the application as a first class citizen. Consequently, the location is a central entity of context-aware adaptation. Nevertheless, already in 1998, Schmidt et al. [14] observed that the exact physical location is only sufficient for a rather limited set of adaptation scenarios. The more important information for adaptation is the semantics of the location and the implications that can be reasoned about the fact that an application is located at a given place at a given time. Especially in indoor scenarios, where it is hard to determine the absolute position of an object, a symbolic representation of the location becomes very important. When, for example, a person enters a meeting room, where a meeting takes place at this time, it can derive that the person is participating in a meeting. Because indoor localization is still difficult and usually requires a special sensing infrastructure involving high costs and high setup efforts, this kind of location aware services could not gained wide spread acceptance.

The ContextPoint is a device that observes the environment and acts as a coordinator for several self-adaptive applications within this environment. The goal of ContextPoint is to provide an easy to install and easy to use infrastructure to enable context-aware adaptation based on the location of the user. Figure 3 shows a context diagram of ContextPoint with three types of interacting services: *Mobile Devices*, *Environmental Devices* and *Cloud Services* mediated by the ContextPoint. The basic principle for interaction is locality. When a user, carrying a mobile device, is close to a ContextPoint, information and services implied by the location and the task of the user are provided automatically.

In this section, we first outline the Top-Level Architecture of the ContextPoint. Afterwards, we explain the main features and their relation to the requirements, stated in Section I. Furthermore, we describe how the ContextPoint architecture relates to the concept of meta-adaptation. Finally, we outline

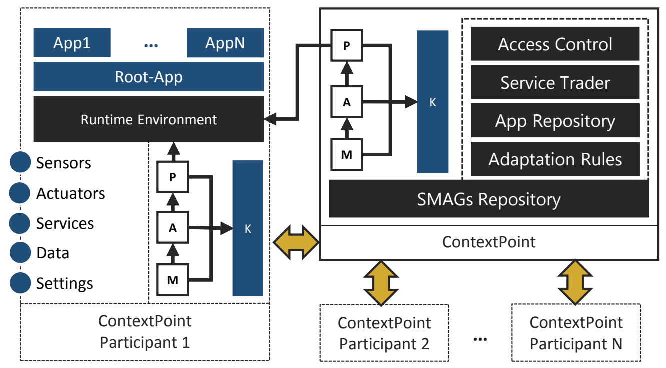

Fig. 4. The ContextPoint Architecture

the crucial aspect of privacy and security.

A. Top-Level-Architecture

ContextPoint is embodied as a device that is integrated in the infrastructure of an arbitrary location, providing contextual services by placing a mobile device at it. The ContextPoint device as well as the mobile devices initiate communication using Near Field Communication (NFC). Therefore, on the mobile device a *ContextPoint Root-App* is running in the background, receiving NFC-Events. When the device is placed near the ContextPoint, it recognizes the ContextPoint's signature and initializes communication. The ContextPoint shares local Wi-Fi credentials to enable the device to automatically connect for a long-term communication. After the device connected to the local Wi-Fi, the Root-App will register itself at the ContextPoint device which contains a SMAGs repository storing implementation artefacts, contextual information, and adaptation rules; offering a Service Trader as well as providing access control.

Figure 4 shows the Top-Level-Architecture of ContextPoint with *ContextPoint Participants* as an abstraction of mobile devices, environmental devices and cloud services which are mediated by the ContextPoint. Each participant runs a ContextPoint Root-App responsible for the communication with the ContextPoint and the provision of base services for Sensors, Actuators, Data, and Settings. This app acts as a runtime environment for all SMAGs-Apps deployed on the device. Each app can publish services and connect to services offered by other participants. The Root-App is responsible for the registration and de-registration at a ContextPoint. After registration, different capabilities can be integrated into the environmental infrastructure. The data delivered by the **Sensors** can be used as additional data sources to construct the ContextPoint context model. Simple services for manipulating its **Actuators** as well as more complex **Services** can be published at the Service Trader. In consequence, they can be used by other applications registered at the same ContextPoint. Furthermore, special services for accessing **Data** (e.g., user profile) or reading and manipulating the devices **Settings** can be published, to externally change the state of a device or service.

B. Location Specific Apps

In the *App Repository* of ContextPoint, location specific apps (e.g., a slide-presentation app for a meeting room) can be stored together with general metadata, contextual information and access rights. When a mobile device registers at a ContextPoint, the Root-App queries the App Repository and presents an overview of all available apps to the user. Based on the user's identity, this list might be filtered according to the access rights of the applications and roles of the user (e.g., speaker) and the current context. Furthermore, the list can be filtered and prioritized using contextual information (e.g., business applications) and application specific metadata using roles. The user can then select apps that are automatically installed within the Root-App. Together with the application, the ContextPoint device provides a context- and user-specific initial configuration (e.g., use the meeting room's projector as a default presentation device). This automatic App-Provisioning covers requirement **R3**.

C. Location Specific Settings

Each participant can expose read and write services for its settings to the environment. Alongside with a capability model of the participant, an access control component is generated by the Root-App. Based on the context, the user and the defined access rights, other devices as well as the ContextPoint itself can evaluate and change the settings of the participant (e.g., mute the mobile device). This kind of parameterized adaptation allows to change the state of a participant in unforeseen ways, addressing requirement **R1**.

D. Service Trader

SMAGs applications can offer a subset of their provided functionality for remote access (e.g., an interface to present and control a presentation on a smart projector). Those dynamically published services can be registered at the ContextPoint. Alongside with the structural description, a subset of the applications' context model, the identity and metadata of the owning user as well as the functionality can be published. Other participants can search for required services that are appropriate with respect to their own context (e.g., search for presentation services). This search is provided by the Service Trader **R2**.

E. Checkout

A participant can be signed off from the ContextPoint by either using a checkout service of the Root-App or when the connection to the ContextPoint is lost. SMAGs Repositories can be interconnected by a Peer-to-Peer network [6]. This infrastructure is used to notify ContextPoint devices when a participant signs in another ContextPoint. In order to avoid concurring adaptations, ContextPoints will close the connection to participants that are still signed in, but have been detected at another ContextPoint devices.

F. Meta Adaptation

In the SMAGs approach, the MAPE-K loop is modelled explicitly using a component architecture, whereby the flow between the components is modelled implictly using events. As depicted in Figure 4, the ContextPoint runs a MAPE-K loop that can adapt participants. Like any other SMAG-based control loop, the loop itself is a SMAGs application.

Consequently different adaptation strategies and context models can be used for different participants or situations. The ContextPoint gathers contextual information using sensors exposed by other participants (e.g., mobile devices, environmental sensors) and stores this information in its local context model. The execution layer of this control loop is the SMAGs runtime environment of the corresponding participant. This allows the ContextPoint to query a participants application runtime model to decide whether or not an adaptation is necessary. Since SMAGs supports parametrized, control-flow based, and compositional adaptation; all three adaptation mechanisms can be used to adapt the MAPE-K loop. This Meta Adaptation triggered by the environment supports adaptation that the developer initially did not foresee, which supports requirement **R1**. Since meta-adaptive SMAGs apps can dynamically connect several applications within one control-flow, also requirement **R2** is tackled.

G. Security and Privacy

We are aware that the proposed architecture creates serious security (e.g., abuse of devices) and *privacy* (e.g., unauthorized access to personal data) threats. On the one hand, the owner of a ContextPoint device must be sure that only approved participants can get access to the provided service- and data-infrastructure. On the other hand, a participant wants to make sure that private data cannot be accessed by other participants and that neither data nor services from potentially compromised sources are used. Because the software running on ContextPoint devices and the participants devices is realized by SMAGs applications, the role-based adaptation mechanism can be used for security and privacy adaptation. Currently, the following mechanisms are included: First, every ContextPoint device has at least one owner that can regulate which participants can sign in. By default two sign-in strategies are supported. Either the owner grants the access for all users, or he has to confirm each user. Second, in order to ensure client-side privacy, *Filter-Ports* [6] can be used to restrict access functionality offered by a component. Special *Access Ports* by default restrict any access to the underlying functionality, only granting access to those participants the user has defined. Hence, within the Root-App the user has the possibility to define which services can be used by which other participants. One serious threat is the possible abuse of the capabilities of Meta Adaptation. One way to address this issue is to use Access-Ports for the MAPE-K loop, too. Because the services of the runtime environment for querying the application model and executing reconfiguration scripts are SMAGs Ports, Filter-Ports can equally be used to restrict the access to the remote services of the runtime environment. When a user does not trust the ContextPoint at a given location he can force the application to not expose any information about the application architectures and forbid any remote access to the reconfiguration system.Security and privacy threats are important topics for adaptive systems in general, especially in extrinsic unanticipated adaptation. We argue that the role-based adaptation mechanism of SMAGs is a well suited mechanism towards safe and secure self-adaptive systems, which is to be investigated in detail in future work.

V. Implementation

To show that the proposed architecture concept for location-based extrinsic Meta Adaptation is feasible, we have implemented ContextPoint as well as several ContextPoint apps using the Java-based implementation of the Smart Application Grids runtime environment. As ContextPoint device, we used a Windows 7 notebook with a Standard JVM. In future, we plan to investigate the use of a Raspberry Pi [15] due to its smaller dimensions and lower energy consumption. On the notebook a ContextPoint application was running on top of the SMAGs runtime environment, supporting the features presented in Section IV. A USB NFC Reader was connected to the notebook for the initial sign-in procedure for NFC-ready mobile devices. As a mobile, device we used a Nexus 7 Android tablet with Android version 4.1.

Our sample scenario is based on a smart meeting room with a built-in, remotely controllable projector, a light system and the ContextPoint device. When a person enters the meeting room he holds his smartphone against the ContextPoint, which exchanges the local Wi-Fi credentials via NFC. The Mother-App running on the smartphone receives the ID of the ContextPoint and the Wi-Fi credentials. Afterwards the user is asked if the smartphone should login into the local Wi-Fi. After the user confirmed to log in, the Mother-App scans the local Wi-Fi for the ContextPoint with the given ID using the Universal Plug and Play (UPNP) protocol. When the Mother-App has found the corresponding ContextPoint it uses the registration API to authenticate and publish a description of available services. In this case the smartphone exposes a brightness sensor that can be used to determine the rooms brightness. The ContextPoint determines via its context model that a meeting is taking place in this room at the time the person enters the room. Based on a rule, the owner of the ContextPoint defined, the smartphone is muted by the ContextPoint (Requirement **R1**). Furthermore, a *"meeting app"* and a *"presentation app"* are offered to the user (Requirement **R3**). The meeting app provides the user with meeting specific information that is preconfigured to show the goal and agenda of the meeting as well as all logged-in participants alongside with their shared profile information of this particular meeting. The presentation app lists all presentation files on the device as well as on the cloud storage associated to the user and offers the capability to start them in a slide show. Based on the information of the context model, the ContextPoint automatically deploys a Filter-Port that orders the presentations by their defined category, so that meeting related presentations are shown first (Requirement **R1**). For the slide show functionality each slide can be shown on a *presentation device* which is by default the screen of the device executing the app. Within the room a projector is installed which can be used remotely as a presentation device. Therefore, another notebook is connected to the projector via cable, running a SMAGs app that remotely offers the *"ISlideShowPresenter"* interface. The ContextPoint device offers to dynamically connect the projector with the presenter app to extend the display (Requirement **R2**). When the user agrees, the slide-show is automatically presented using the rooms projector. Whenever a slide changes, metadata about the slides is transferred along with the original content. The ContextPoint can dynamically include a Port Model within the presentation app that investigates the metadata of each slide when it is shown. When the brightness in the room is high (sensed by the brightness sensor) and media content is shown (e.g., a video within the slide), the Port Model automatically controls the rooms light system to decrease the brightness and increase the visibility of the video (Requirements **R1** and **R2**). Afterwards it will illuminate the room again. This sample application was deployed on exemplary meeting room setup on a local exhibition. With the realization of this example we have shown that the presented approach supports the presented requirements within this scenario.

VI. Related Work

Much research has been done in the field of self-adaptive and context-aware systems. Especially, in the domain of mobile and ubiquitous computing, numerous research projects were conducted. One of the first context-aware systems was *ParcTab*, developed by Schilit in 1993 [16]. ParcTabs are individual mobile devices that are dynamically connected to other devices based on their location. At that time the major problem is to physically connect those devices using a heterogeneous network infrastructure. While those issues were solved over the last decades, current research problems mainly focus to autonomously provide the best suited services on the desired devices based on the users location, time, and surroundings.

Many context-aware and location-based applications have been developed. Bravo et al. presented a self-adaptive, context-aware conference application using RFID tags for localizing people within a conference building and distributed applications sensing a shared context (including the current location of the conference attendees) provided by a central server [17]. There are also other location-based services based on NFC-Tags for advertisement [18] and content delivery [18]. All these examples show that a lot of different use cases for context-aware systems exist, which all treat location as a central aspect. The commonality between these approaches is that they have implemented their own architecture designed for their specific, individual usage scenario. All those architectures support a subset of the adaptation capabilities of the ContextPoint approach. Thus, ContextPoint can be used as a platform for context-aware and location-based applications as it supports all required features of the discussed examples.

Another large research field concentrates on location-based services. Huebscher and McCann, for instance, presented a middleware for location-based, context-aware applications in smart home environments [19]. In their approach, the context (e.g., location, activity, etc.) is provided by *Context Services*, which analyse data delivered by one or more *Context Providers*. Based on the interpretation of the context, services are selected for a current activity of the user. This is similar to the Service Trader architecture of the ContextPoint approach. Nevertheless, they neglect that service selection is only one aspect of context-aware adaptation. Furthermore, traditional service-oriented approaches cannot individualize single services for multiple clients (only for every user or none). Since roles can adapt the behaviour of a player based on relationships, a single service instance can have different behaviour depending on the client using this service.

Other research projects aimed to design variable MAPE-K loops in order to adjust the adaptation process. In most of the cases, only the monitoring and analyse phase can be extended at runtime. The MUSIC project, for example, proposed a self-adaptive architecture for mobile devices with *Context Plug-Ins* [20]. This plug-in infrastructure enables to change the adaptation process. Even though, the MUSIC architecture does not prohibit the introduction and activation of plug-ins at runtime, it is only possible to extend the monitoring and analyse phase with new sensors and reasoners. In contrast, the ContextPoint architecture allows to exchange the operation of the whole MAPE-K loop (e.g., use alternative context model representations, introduce new planners or even change the control flow between the elements of the control loop).

Other approaches in the area of adaptive systems in mobile scenarios with context-aware extensible adaptation focus on the content presented on the device. While those approaches use a similar distributed architecture, they provide user-profile and device-capability adapted content [21]. The fine-grained structure of the applications cannot be adapted and the overall behaviour of an application cannot be changed. Van Sinderen et al. propose an architecture for context-aware adaptation for faster static evolution (i.e., at design-time) of self-adaptive systems [22]. Therefore, ECA-Rules are evaluated against a distributed context-management infrastructure to steer adaptation and context-dependent services. For adaptation they focus on component replacement and reconnection, the drawbacks are discussed in [6]. Like in the proposed ContextPoint architecture the monitoring and analyse phase can be distributed across the environment, while in their concept the plan and execute phase are integrated within the application. This, however, hinders the adaptation to unanticipated scenarios for mobile devices, because concepts that were not considered during design-time of the ECA rules cannot be handled at runtime. The presented Meta Adaptation architecture was first conceptualized by Perrouin et al. [23]. They describe how Meta Adaptation can be used to adapt MAPE-K loops at runtime in order to adjust the adaptation process to the requirements arising from contextual changes. The ContextPoint architecture can be seen as a concrete implementation of this concept. Combined with the reconfiguration capabilities of SMAGs, fine grained and cross-cutting reconfigurations of an applications MAPE-K loop can be modelled and realized. The proposed architecture for location-based external Meta Adaptation aims to dynamically connect local devices to build ad-hoc SoS. Weyns et al. proposed three different architectural styles of self-adaptation for SoS [24]. In his classification, the proposed Meta Adaptation architecture would be categorized as an instance of the *Collaborative Adaptations* architectural style with multiple hierarchical MAPE-K loops. These are able to include the information extracted in the monitoring phase and to reconfigure these loops during execution phase. The presented approach forms a Service-Oriented Architecture (SOA) [25], since all applications expose services that can be integrated into other applications. Traditionally, SOA-based approaches rely on Web Services. As discussed by Piechnick et al. [6], adaptation in classical Web-Service-based solutions use adaptive orchestration or choreography. On the one hand, the selection of services (i.e., which service instances), on the other hand, the process itself (i.e., control- and data-flow between the services) can be varied, to adapt the behaviour of the overall application. Especially the service selection corresponds to adaptation with component replacement (see Section II). In contrast, SMAGs allows for varying the behavior of a single instance of a service based on the environmental situation and the calling instance without the need to replace/create entire service instances, which is important for stateful services, when the state cannot be transferred easily. Web Services can be used as a platform-independent communication infrastructure instead of the current socket-based realization in SMAGs, whereby the implementation of a service is a SMAGs component that can be adapted using roles.

VII. Conclusion and Future Work

Mobile devices changed the use and development of software fundamentally. In the future, users will expect that apps for mobile devices automatically adapt their behaviour based on their physical location, their user profile, and the current task. Furthermore, cheap, standardized, and easy to install sensors and actuators for smart environments offer new possibilities to gather environmental information. This in turn will extend the functionality of a mobile device towards environmental services. Traditionally, adaptive systems are based an a self-adaptive control loop within the application, which senses the environment and coordinates reconfiguration. In this paper, we showed how the adaptation architecture of Smart Application Grids can be used for Meta Adaptation and, in consequence, to support unanticipated scenarios. Because the MAPE-K loop of SMAGs applications is itself designed as SMAGs components, it can be adapted at runtime as well. This allows to create MAPE-K loops in other applications that reconfigure the adaptation process of the original self-adaptive system. Furthermore, we presented an architecture for smart environments, the ContextPoint approach, which aims to provide location-specific unanticipated adaptation. Therefore, the symbolic location of a mobile device is determined by NFC communication with a ContextPoint device. The ContextPoint offers location- and context-specific apps, a Meta Adaptation infrastructure to adapt the participating devices as well as applications running on them in unforeseen ways. Thus, it fully supports unanticipated adaptation, runtime application composition, and automatic application provisioning. For future work, security and privacy issues must be investigated, since those aspects are crucial for a real world application. Furthermore, it must be investigated if low cost computing devices are suitable to handle multiple participants. Additionally, it should be investigated, which of the architectural styles, according to Weyns et al. [24], are suitable for Meta Adaptation.

Acknowledgment

This work is supported by the German Research Foundation (DFG) within the Cluster of Excellence “Center for Advancing Electronics Dresden”, the Collaborative Research Center 912 “Highly Adaptive Energy-Efficient Computing” and the research training group ’Role-Based Software-Infrastructures’.

References

[1] J. O. Kephart and D. M. Chess, “The vision of autonomic computing,” Computer, vol. 36, no. 1, Jan. 2003, pp. 41–50.

[2] W. Pree, “Meta patterns - a means for capturing the essentials of reusable object-oriented design,” in Object-Oriented Programming. Springer, 1994, pp. 150–162.

[3] B. Morin et al., “An aspect-oriented and model-driven approach for managing dynamic variability,” in Model Driven Engineering Languages and Systems, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5301, pp. 782–796.

[4] OMG, Meta Object Facility (MOF) Core Specification Version 2.0, 2006. [Online]. Available: http://www.omg.org/cgi-bin/doc?formal/2006-01-01 [retrieved: April, 2014]

[5] U. Aßmann, Invasive software composition. Springer, 2003.

[6] C. Piechnick, S. Richly, S. Götz, C. Wilke, and U. Aßmann, “Using role-based composition to support unanticipated, dynamic adaptation-smart application grids,” in ADAPTIVE 2012, The Fourth International Conference on Adaptive and Self-Adaptive Systems and Applications, Nice, France, 2012, pp. 93–102.

[7] T. Reenskaug, P. Wold, and O. A. Lehne, Working with objects - the OOram software engineering method. Manning, 1996.

[8] J. Hillman and I. Warren, “Meta-adaptation in autonomic systems,” in Distributed Computing Systems, 2004. FTDCS 2004. Proceedings. 10th IEEE International Workshop on Future Trends of, 2004, pp. 292–298.

[9] A. K. Dey, “Understanding and using context,” Personal Ubiquitous Comput., vol. 5, no. 1, Jan. 2001, pp. 4–7.

[10] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications, ser. WMCSA ’94. Washington, DC, USA: IEEE Computer Society, 1994, pp. 85–90.

[11] S. Greenberg, “Context as a dynamic construct,” Hum.-Comput. Interact., vol. 16, no. 2, 2001, pp. 257–268.

[12] J. Coutaz, J. L. Crowley, S. Dobson, and D. Garlan, “Context is key,” Commun. ACM, vol. 48, no. 3, 2005, pp. 49–53.

[13] A. Zimmermann, A. Lorenz, and R. Oppermann, “An operational definition of context,” in Modeling and Using Context, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, vol. 4635, pp. 558–571.

[14] A. Schmidt, M. Beigl, and H. w. Gellersen, “There is more to context than location,” Computers and Graphics, vol. 23, 1998, pp. 893–901.

[15] “Raspberry Pi,” http://www.raspberrypi.org/, visited 05/05/2014.

[16] B. Schilit, N. Adams, R. Gold, M. Tso, and R. Want, “The parctab mobile computing system,” in Workstation Operating Systems, 1993. Proceedings., Fourth Workshop on, Napa, CA, 1993, pp. 34–39.

[17] J. Bravo, R. Hervs, I. Snchez, G. Chavira, and S. Nava, “Visualization services in a conference context: An approach by rfid technology,” j-jucs, vol. 12, no. 3, 2006, pp. 270–283.

[18] M. Hardt and S. Nath, “Privacy-aware personalization for mobile advertising,” in Proceedings of the 2012 ACM Conference on Computer and Communications Security, ser. CCS ’12. New York, NY, USA: ACM, 2012, pp. 662–673.

[19] M. C. Huebscher and J. A. McCann, “Adaptive middleware for context-aware applications in smart-homes,” in Proceedings of the 2Nd Workshop on Middleware for Pervasive and Ad-hoc Computing, ser. MPAC ’04. New York, NY, USA: ACM, 2004, pp. 111–116.

[20] N. Paspallis et al., “A pluggable and reconfigurable architecture for a context-aware enabling middleware system,” in On the Move to Meaningful Internet Systems, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5331, pp. 553–570.

[21] T. Lemlouma and N. Layaida, “Context-aware adaptation for mobile devices,” in Mobile Data Management, 2004. Proceedings. 2004 IEEE International Conference on, Berkeley, CA, USA, 2004, pp. 106–111.

[22] M. van Sinderen, A. Van Halteren, M. Wegdam, H. Meeuwissen, and E. Eertink, “Supporting context-aware mobile applications: an infrastructure approach,” Communications Magazine, IEEE, vol. 44, no. 9, 2006, pp. 96–104.

[23] G. Perrouin et al., “Towards flexible evolution of dynamically adaptive systems,” in Proceedings of the 34th International Conference on Software Engineering, ser. ICSE ’12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 1353–1356.

[24] D. Weyns and J. Andersson, “On the challenges of self-adaptation in systems of systems,” in Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems, ser. SESoS ’13. Montpellier, France: ACM, 2013, pp. 47–51.

[25] D. Krafzig, K. Banke, and D. Slama, Enterprise SOA: Service Oriented Architecture Best Practices, 8th ed. Prentice Hall Professional Technical Reference, 2005.