Context-Aware Personalization for Smart Mobile Cloud Services

Waldemar Hummer(✉) and Stefan Schulte

Distributed Systems Group, Tu Wien, Wien, Austria
{Hummer,Schulte}@dsg.tuwien.ac.at

Abstract. The advent of the Internet of Things and the increasing sensorization of smart devices that surround us in our everyday lives are spurring the demand for context-aware applications to offer personalized services. With the rapid advances in sensor technology, distributed software architectures and backend infrastructures need to be able to systematically deal with increasing amounts of real-time context data. In this paper, we present an approach for intelligent service clouds to cater for the new challenges associated with complex context-aware applications. Based on an illustrative scenario from the connected car domain, we introduce a detailed system model and approach for context-based personalization of mobile services. Our solution focuses on a three-phase approach with context change analysis, context state management, and context-triggered adaptation actions. We discuss details of our prototype implementation and put the contributions into perspective with the related work. After discussing our preliminary results, we draw a roadmap for future work towards context-aware vehicle information systems.

1 Introduction

Over the past years, we have been witnessing a steep rise in the number of devices and sensors being connected to the so-called Internet of Things (IoT) [1], a phenomenon which spans across a multitude of industry verticals, including connected cars, smart homes, e-health, and more. Our increasingly connected world of cyberphysical systems and smart things opens up a wealth of unprecedented opportunities for applications that are specifically tailored towards the needs of mobile users with the ability to adapt to their current contextual environments.

Both the amount as well as the level of integration of context data in the IoT are rapidly increasing. In the automotive industry, real-time machine data from the vehicle (e.g., fuel level) is matched with driver-related biometrics data (e.g., fatigue sensor) and combined with other contextual information (e.g., time, location) in order to satisfy the driver's preferences (e.g., navigate to a gas station for coffee and fuel). This trend has been referred to as the *quantified car* [24], in a reference to this novel combination of *connected car* and *quantified self*. In light of the leaps in progress being made on the device and data collection layer,

A. Norta et al. (Eds.): ICSOC 2015 Workshops, LNCS 9586, pp. 171–183, 2016.
DOI: 10.1007/978-3-662-50539-7_14

the cloud service and application providers need to catch up with the technical challenges introduced by the increased dynamism and context-awareness of smart applications for predominantly mobile, permanently connected users.

Catering for the personal preferences and individual user contexts is becoming a critical requirement for state-of-the-art cloud applications. The processing logic of applications that deal with user context requires integrated support from intelligent service clouds, which are able to capture and monitor user context, adapt to changing environments, and optimize the application delivery.

In this paper, we tackle this issue and discuss an approach for context-aware personalization of mobile cloud services. We illustrate our solution based on a scenario from the connected car domain, with a multitude of personalized services offered to the drivers in a vehicular information system. In particular, we follow a three-phase approach with context change analysis, context state management, and context-triggered adaptation actions.

The remainder of this paper is structured as follows. In Sect. 2 we introduce an illustrative scenario which serves as the basis for presentation. Section 3 introduces the assumed system model and details our proposed solution. Implementation details are covered in Sect. 4, and our work is evaluated in Sect. 5. In Sect. 6 we discuss previous work related to our approach. Section 7 concludes the paper and highlights topics for active ongoing research efforts.

2 Scenario

We consider a scenario from the *connected car* domain. Figure 1 illustrates a vehicle used by a driver under different user contexts over time ($t \in \{1, \ldots, 10\}$). The car operates a multitude of applications, including navigation, media streaming,

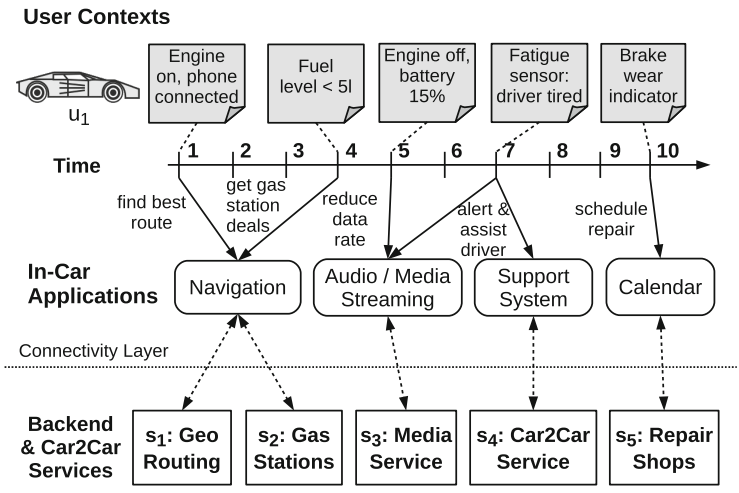


Fig. 1. Scenario: connected car with context-based personalized services

driver support system, and more. Additionally, the car is equipped with a multitude of sensors to regularly monitor the operational status (fuel, battery level, brake wear, etc.) as well as the behavior of the driver (e.g., fatigue sensor).

To enable seamless operation of the in-car applications, the continous context changes are reflected in the applications. At time $t = 1$, the engine is turned on, which triggers the navigation service to determine the target location and find the best route. At time $t = 4$, the fuel drops to a critically low level ($< 5\,\mathrm{L}$), hence the navigation app queries gas stations in the vicinity, and displays commercials for special deals. At $t = 5$, the engine is turned off and battery level is below 15 %, which causes the audio streaming application to reduce the data rate to save power. At $t = 7$, the fatigue sensor of the car alerts that the driver gets tired, which turns on driving assistance in the support system. Finally, at $t = 10$, the brake wear indicator reports that the brakes require maintenance, causing the calendar app to automatically schedule a service with a repair shop.

Table 1. Context-based service personalizations in scenario

Service	Context	Config. effect	Application action
Category: User Preferences			
Streaming	$timeOfDay = morning$ $timeOfDay = afternoon$ $timeOfDay = evening$	$station = news$ $station = pop$ $station = jazz$	subscribe to respective media service
Category: User Experience and Safety			
Navigation	$offTrack = true$	-	re-calculate route
Support System	$fatig = 1$	$assist = yes$	alert cars in vicinity
Category: Service Optimizations			
Streaming	$country = X$	$proxy = X$	use national proxy
Streaming	$battery > 20\,\%$ $battery \leq 20\,\%$	$dataRate = high$ $dataRate = low$	adjust data rate and buffering

Personalized configurations are applied depending on the current context. Table 1 contains an exemplary listing of context settings, plus the configuration effects and associated application actions. We distinguish between *user preferences* (explicitly defined by users), *user experience and safety* (predefined rules encoded within the respective services), as well as *service optimizations* (non-functional aspects). For instance, during morning hours ($timeOfDay = morning$) the driver prefers to listen to news radio, whereas the rest of the day she likes pop or jazz music. If the driver departs from the course ($offTrack = true$), the context triggers a re-calculation of the best route in the navigation service. Or, if the fatigue sensor alerts $fatigue = true$, the in-car driving assistance gets enabled.

Based on this illustrative scenario, we identify the following key challenges:

- **Manage Application-Level Context Changes**: Context data is often imprecise (e.g., precision radius of a GPS sensor) or prone to transient sensor errors. A mechanism is needed to reliably identify relevant context changes.
- **Application Actions and Adaptations**: Context changes result in application actions, which may also involve adaptation of the backend services. A systematic approach for reconfigurations at different levels is needed.
- **End-to-End Cloud Support**: Support for context handling and personalization should be offered on the Cloud layer. Integration with existing Cloud application development paradigms is desirable.

3 Approach

This section presents our approach for context-aware personalization of mobile Cloud services. We introduce the basic system model in Sect. 3.1, outline the approach in Sect. 3.2, and discuss selected details in Sects. 3.3, 3.4, and 3.5.

3.1 System Model

Table 2 lists the elements of the assumed system model with the respective symbols, description, and a brief example with reference to our scenario. In our formalization, $\mathcal{P}(Y)$ denotes the *power set* of a given set Y, and $[f] := \{f' : dom(f) \rightarrow codom(f)\}$ denotes the *function space* of a function f.

The model contains a set of services (S) which are consumed by different mobile service users (U). Each user is associated with a context (X) that changes over time (e.g., vehicle location). The context attributes are captured via a mapping from key (K) to value domain (V). The domain of time (T) is also encoded in the context. The current context of a user is determined by the function c. The function p represents user-specific service personalizations which apply at a certain point in time. Assuming user u_1 is currently driving in Austria, the media service (s_3) utilizes the local proxy ($proxy = AT$) for music streaming.

Function d models derived context attributes, for instance if the fatigue sensor alerts ($fatig = 1$) consecutively at time points $t \in T$ and $t + 1$, a warning event is issued and added to the user's context attributes ($fatigWarn = 1$). This technique can be used to eliminate the impact of *false positives* caused by the fatigue sensor. Finally, function t defines a set of triggers where a certain context configuration leads to a change in a user's service personalization. In our example, if the driving user u_1 crosses the border to Germany ($country = DE$), the local proxy ($proxy = DE$) should be used for service s_3.

3.2 Approach Overview

Figure 2 illustrates an overview of our approach, based on the services and context data in our scenario. The figure depicts a timeline for three mobile service users, with different context changes.

Table 2. System model

Symbol	Description	Example
S	Set of services	$S = \{s_1, \ldots, s_5\}$
U	Set of mobile service users	$U = \{u_1, u_2, u_3\}$
$M : K \rightarrow V$	Domain of key-value maps, mapping a key (K) to a value domain (V)	-
$T \subset V$	Domain of time	$T = \{t = 1, \ldots, t = 10, \ldots\}$
$X \subset M$	Domain of user contexts, mapping context attributes (K) to values (V)	$ctx_1 = \{location \mapsto (47.1, 10.2)\}$
$c : U \rightarrow X$	Current context of a user	$c : u_1 \mapsto ctx_1$
$p : (U \times S) \rightarrow M$	Current service personalizations, mapping users (U) of a service (S) to a configuration map (M)	$(u_1, s_3) \mapsto \{proxy = AT\}$
$d : \mathcal{P}(T \times X) \rightarrow X$	Query rules for derived context attributes	$\{(t, fatig = 1), (t + 1, fatig = 1)\} \mapsto (fatigWarn = 1)$
$t : U \times X \rightarrow [p]$	Context-based configuration triggers	$(u_1, country = DE) \mapsto \{(u_1, s_3) \mapsto \{proxy = DE\}\}$

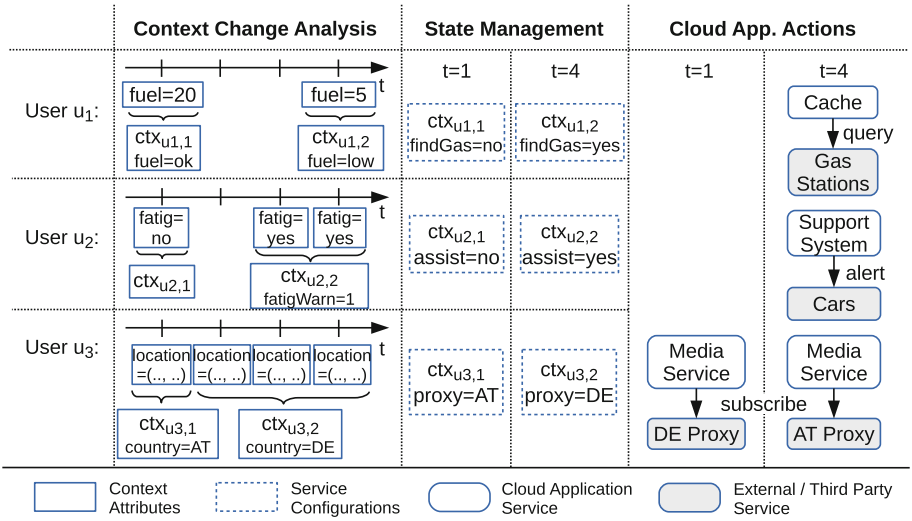


Fig. 2. Illustration of the overall approach

The first part in our three-stage approach is context change analysis, where context events are analyzed over time to derive higher-level context attributes (see Sect. 3.3). In the second stage, the context changes need to be propagated to the state management which tracks the current configurations for all users (see Sect. 3.4). Based on the state management, adaptations are performed to cater for the personalized service configurations (see Sect. 3.5).

3.3 Identification of Context Changes

We distinguish "raw" (or low-level) context changes which are typically measured by sensor devices, and "complex" (or higher-level) context changes which reflect the actual context information that is relevant to the application services.

In Fig. 2, the raw context changes for fuel level in liters ($fuel$), fatigue sensor ($fatig$), and current GPS location ($location$), need to be analyzed and enriched with additional information in order to have a meaningful impact on the service personalization. We distinguish the following mechanisms to derive relevant context changes:

- **Discretization**: Context attributes with a continuous value domain are mapped to a discrete value domain. For instance, a fuel level above 5 L is considered "*ok*", whereas levels below 5 L are considered "*low*". Evidently, the discretization varies from vehicle to vehicle, and also depends on the geographic area in which the vehicle is operated (in a deserted area, 5 L of fuel may be insufficient to reach the next gas station).
- **Sampling**: The raw context values should only be propagated to the application if they are deemed to properly represent the user's environment. The technique of sampling can be used to identify statistically significant context changes, and eliminate the impact of false positives in the sensor measurements. For instance, the fatigue warning in Fig. 2 ($fatigWarn = 1$) is only issued if there are two consecutive fatigue sensor measurements ($fatig = 1$ at time points 3 and 4). Clearly, this simple example can be extended to more meaningful statistical significance levels.
- **Pattern Detection**: For more sophisticated cases, we utilize complex event processing (CEP) [9] to derive high-level context changes from raw events. The context changes are modeled as a stream of events over time, and CEP window queries allow for complex pattern detection. In Fig. 2, if we combine the current GPS location of user u_3 with map data, we can derive the current country (AT) and anticipate the entry into a new country (DE) if the event pattern indicates that the driver approaches the country border.

Currently, rules and CEP queries are defined manually, yet for convenience we offer a set of predefined rule templates which can be parameterized according to individual scenarios. For instance, a *geo fence* template reports whether any tracked objects are within a circular geographical region; parameterizations are (1) the center location, and (2) the radius of the geo fence. In future work, we envision a hybrid approach, mixing manually specified rules for identifying

context changes with automatically learned rules for predicting context changes. The necessary underlying machine learning techniques are readily available, as evidenced, for instance, by the Prediction API[1] in the Google Cloud offering. A specialized module to learn rules for deriving context changes will become an important offering as part of intelligent service clouds.

3.4 Multi-user Context Propagation and State Management

Having a mechanism to identify application-relevant changes (Sect. 3.3), we need to propagate the context information and maintain user-specific configuration states for the Cloud services. The context propagation happens in multiple steps, as demonstrated in the connected car scenario: the values from multiple sensors within the car need to be collected to form the user context (step 1), which is transmitted to the Cloud and transformed into user-specific service personalizations (step 2), and then combined into optimized configurations and variants of the Cloud services shared among the users (step 3).

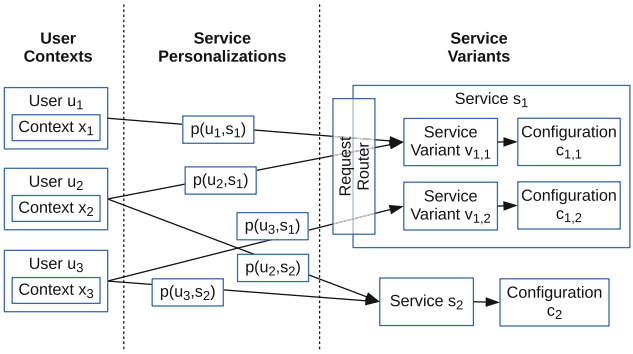


Fig. 3. Propagation of context values and management of configuration state

Figure 3 illustrates the approach based on exemplary context request flows from three service users (u_1, u_2, u_3). The contexts (x_1, x_2, x_3) on the left hand side are transmitted and transformed into service personalizations, via function t in our system model in Sect. 3.1. Any interaction of a user u_m with a backend service s_n carries the service personalization state $p(u_m, s_n)$ that helps the request router to select the respective service variant. A service variant is an instance of the service that has a particular configuration associated, yet this configuration is shared by a multitude of users (as opposed to service personalizations, which are user-specific). For example, in our scenario the media service is configured with two variants for high-quality and low-quality streaming, respectively. Detailed discussion of service variability engineering are out of the scope of this paper, and for details we refer to specialized literature [21].

[1] https://cloud.google.com/prediction/.

Currently, we apply state management for single services and their service variants only, and do not take into account the interactions between groups of services or business processes. In future work, we plan to integrate support for more comprehensive service ontologies [23] allowing to model service workflows.

3.5 Context-Triggered Actions and Adaptations

Based on changes in the service configurations, application actions are triggered which result in adaptation of the Cloud environment. We distinguish three types:

- **Application**: Any actions encoded in the business logic of the application.
- **Infrastructure**: Any adaptations related to current allocation of resources on the infrastructure layer (e.g., virtual machines, disk volumes, databases).
- **Topology**: Any changes in the relationships and interconnections of the components (e.g., master-slave election in a clustered service).

Table 3 contains three examples of context-triggered adaptations, applied to our scenario. If a driver is running out of fuel, a cache infrastructure component is configured before the application starts querying for gas stations. To adjust the *dataRate* for the streaming service, an adaptation is required in the request router to pick the respective service variant. Finally, switching to a different streaming proxy induces a topology change with a new service variant. Our approach also integrates previous work on *service prefetching* [18], an application adaptation that is necessary to handle unreliable connectivity (e.g., in tunnels).

Table 3. Context-triggered adaptations (Examples)

Trigger	Type(s)	Adaptation
$findGas = true$	Infrastr., Application	Instantiate cache, query gas stations
$dataRate = high \vert low$	Topology, Application	Configure request router to pick respective variant of media service
$proxy = X$	Topology	Create a service variant that subscribes to the stream proxy of country X

4 Implementation

We are currently working towards a full-stack implementation of our approach, embedded in Google's scalable services cluster management tool *Kubernetes*[2]. Here we discuss the architectural design and selected details of our prototype.

Figure 4 depicts the architecture of the deployed system (third-party tools are in grey boxes). On the client side (e.g., in-car application platform) each user

[2] http://kubernetes.io/.

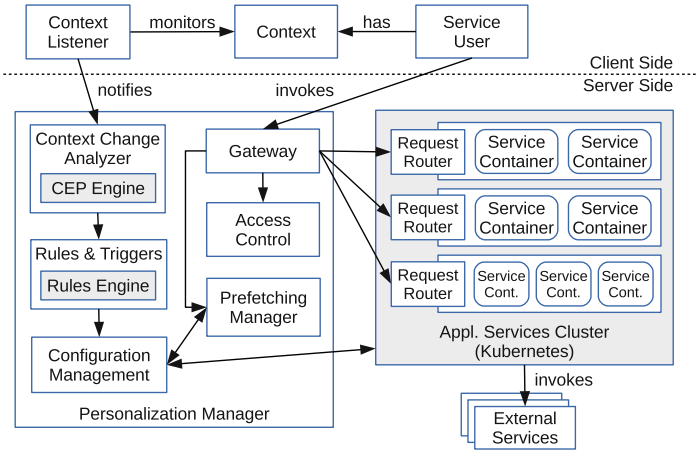


Fig. 4. System architecture

operates under a specific Context, which is monitored by the Context Listener. For the external client-server connectivity, we assume a cellular network connection (e.g., UMTS/HSPA). Upon notification, the Context Change Analyzer uses a CEP engine (*Esper* [5]). We build on our previous work and develop a Domain Specific Language (DSL) for specifying rules to derive application-relevant context changes, based on the MONINA language [19]. The context changes are fed into the Rules & Triggers component, which incorporates a Rules Engine.

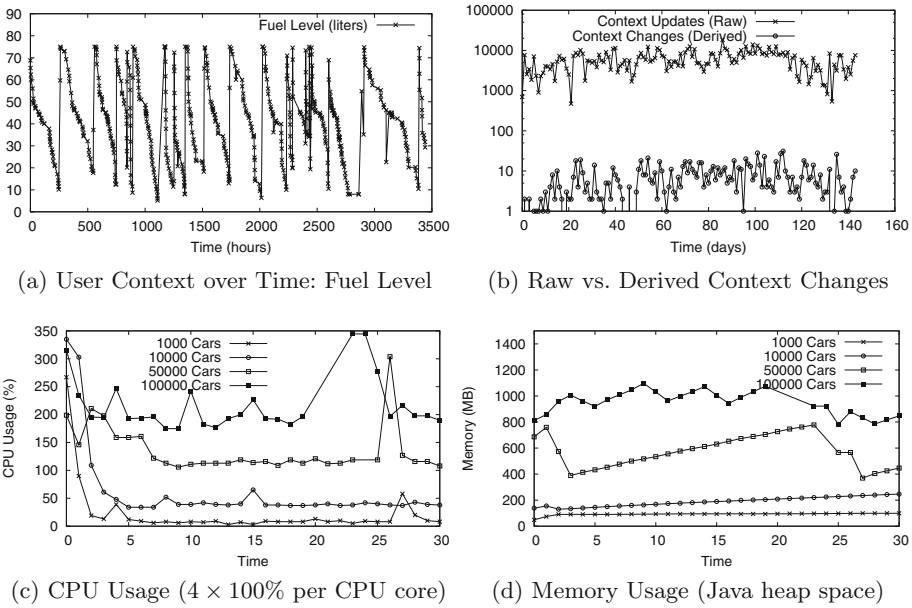
Our implementation caters for the principles of the *12-factor app* [20], which has gained high popularity for scalable Cloud applications. Application services are mostly stateless, the state (e.g., current user context) is maintained in a configuration management tool (we use the distributed key-value store *etcd* provided by *CoreOS*). For deployment, we utilize DevOps automation scripts which deploy the services and reliably bring the infrastructure into the desired state [17].

The user accesses the services via a Gateway which is responsible for performing Access Control, deciding over service prefetching (via the Prefetching Manager [18]), and finally forwarding requests. The Request Routers select among the available service variants (each deployed in a separate service container). Currently, this is tailor-made, because Kubernetes merely performs simplistic round-robin load balancing; we plan to integrate our context-based routing directly into the Kubernetes code base, which we deem a highly useful extension.

5 Evaluation

In this section we evaluate our approach and discuss some of the preliminary results that we have collected. We have set up a test bed on a machine with quad-core 2.5 GHz CPU and 16 GB RAM. For experimentation, we utilize a real-life data set published by Volkswagen as part of a programming contest[3]. The data

[3] Volkswagen CodeFest[8], http://group-it.volkswagenag.com/codefest/codefest.html.



(a) User Context over Time: Fuel Level

(b) Raw vs. Derived Context Changes

(c) CPU Usage ($4 \times 100\%$ per CPU core)

(d) Memory Usage (Java heap space)

Fig. 5. Evaluation results

set contains traces of eight driving cars with a rich set of context attributes (including: speed, battery, fuel, fatigue sensor, brake wear sensor, etc.).

Context snapshots of the vehicles are recorded approximately every 5–30 s, resulting in a total of roughly 800K data points with some 25M context attribute values. Figure 5a plots the fuel level of one of the car traces, recorded over a period of 143 days (apprx. 3500 h). The values are aggregated over periods of one hour; in fact, for our scenario we are not interested in the detailed fuel levels, but only in the critically low levels. The same applies for the fatigue sensor where we only need to monitor the application-relevant transition from $fatig = no$ to $fatig = yes$, or vice versa. This ratio of raw events to derived context changes is illustrated in Fig. 5b (note the logarithmic scale on the y-axis). That is, the majority of context data can be pre-processed inside the car (by the Context Listener) and need not be transmitted to the server, which leaves huge space for optimization of network utilization, as well as energy usage.

Yet, with increasing numbers of cars, the burden on the infrastructure can become significant. In our experiments we have evaluated the end-to-end stress to the backend, for increasing number of cars (1K, 10K, 50K, 100K), assuming each car sends one context update (e.g., location) every 5 s. Figure 5c and d depict the CPU usage and memory usage of the Personalization Manager, which hosts the CEP Engine, Rule Engine, Configuration Management etc. The CPU spikes at the beginning due to high initialization efforts for the CEP engines, and the intermittent CPU spikes (e.g., time points 23/24) are due to Java garbage

collection (see corresponding drops in heap space memory). We observe that for 100K cars the single-node deployment operates almost at its limit; currently, we are extending our implementation with a distributed setup, to evaluate even larger scenarios. In a large-scale setup we further need to evaluate the reliability of the system under different faults during processing of the context events [15].

6 Related Work

In this section, we discuss related work in the areas of context-aware computing, Web service personalization, and adaptive service-oriented systems.

Early work on context-aware systems [2] and pervasive/ubiquitous systems [3] dates back to the early 2000s, with seminal contributions achieved in various areas, including ontologies for context monitoring [8], service-oriented middlewares for context-based computing [12], intelligent context dissemination in vehicular networks [10], as well as context-based human-computer interaction [4]. Recently, topics revolving around context-aware computing have found new application in the area of IoT in general, and the domain of connected vehicles in particular. Gansel et al. [11] introduce a context-aware access control concept for automotive systems, focusing on human interactions with the car computer. Bolchini et al. [6] discuss a taxonomy of context models, along different dimensions like context attributes, representations, and management. The approach by Ouedraogo et al. [22] uses *models@run.time* techniques for contextualized deployment of security policies in intelligent service clouds. Their approach discusses the design of context-based security mediators, while our focus is on end-to-end context propagation and context-triggered adaptations in service clouds.

Previous approaches to Web service personalization have focused on rules modeling, service matching, and personalized adaptation. Yu et al. [26] present a framework for rules-based personalization in Web service workflows, which uses aspect-oriented programming (AOP) with hooks to allow dynamic switching between user contexts. Hella et al. [13] use Semantic Web technologies to perform service matching based on user preferences. Wang et al. [25] propose *rule nets* as a technique to express rules for personalized needs. While their work presents a formal model and language for expressing rules, our focus is on applying CEP and rules for analysis of context changes in adaptive cloud applications.

A large body of research has been done in the area of adaptive service-based systems. Hu et al. [14] discuss a rule-based approach for dynamic adaptation of business processes based on context changes. Their assumed model is a business process definition, whereas we target cloud applications with service variants and context-triggered configuration changes. Inzinger et al. [19] introduce event-based monitoring and adaptation of application configurations, which has influenced our technical solution. Brogi et al. [7] discuss dynamic contextual adaptation for behavioural interfaces, e.g., between a client and server process. Their contextual environments provide interface mappings, but do not take complex patterns of context updates into account, which is at the core of our work.

7 Conclusion

The advance of the Internet of Things poses novel challenges with regards to massive amounts of real-time context data generated by devices and sensors surrounding us in our everyday lives. The ubiquity of contextual information opens up novel opportunities for personalization in modern user-centric applications. Built-in support for personalization is hence becoming a critical requirement for intelligent service clouds. In this paper we introduce and discuss techniques for context-aware personalization of mobile cloud services.

Based on our illustrative scenario from the connected cars domain, we provide a detailed system model and outline our solution, following a three-stage approach. First, we utilize CEP techniques to aggregate raw context events into higher-level information, in order to identify application-relevant context changes. Second, we apply triggers to propagate context changes into user-specific service configurations. Third, we illustrate the integration with cloud services to apply context-triggered adaptation actions. We discuss our prototype implementation which is embedded into Kubernetes, a state-of-the-art cloud technology for cluster management of service containers. In our future work, we extend our approach with various advanced aspects of context processing and service personalization, including high scalability, multi-tenant optimization of the event processing logic [16], as well as privacy and data protection aspects.

Acknowledgements. This work is partially supported by the European Union within the SIMPLI-CITY FP7-ICT project (Grant agreement no. 318201).

References

1. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
2. Baldauf, M., Dustdar, S., Rosenberg, F.: A survey on context-aware systems. Int. J. Ad Hoc Ubiquit. Comput. **2**(4), 263–277 (2007)
3. Bellavista, P., Corradi, A., Fanelli, M., Foschini, L.: A survey of context data distribution for mobile ubiquitous systems. ACM Comput. Surv. **44**(4), 24 (2012)
4. Bellotti, V., Edwards, K.: Intelligibility and accountability: human considerations in context-aware systems. Hum.-Comput. Interact. **16**(2–4), 193–212 (2001)
5. Bernhardt, T., Vasseur, A.: Esper: event stream processing and correlation. ONJava, in OReilly (2007). http://www.onjava.com/lpt/a/6955
6. Bolchini, C., Curino, C.A., Quintarelli, E., Schreiber, F.A., Tanca, L.: A data-oriented survey of context models. ACM Sigmod Rec. **36**(4), 19–26 (2007)
7. Brogi, A., Cámara, J., Canal, C., Cubo, J., Pimentel, E.: Dynamic contextual adaptation. Electron. Notes Theoret. Comput. Sci. **175**(2), 81–95 (2007)
8. Chen, H., Finin, T., Joshi, A.: An ontology for context-aware pervasive computing environments. Knowl. Eng. Rev. **18**(03), 197–207 (2003)
9. Cugola, G., Margara, A.: Processing flows of information: from data stream to complex event processing. ACM Comput. Surv. (CSUR) **44**(3), 15 (2012)
10. Eichler, S., Schroth, C., Kosch, T., Strassberger, M.: Strategies for context-adaptive message dissemination in vehicular ad hoc networks. In: MOBIQUITOUS (2006)

11. Gansel, S., Schnitzer, S., et al.: An access control concept for novel automotive HMI systems. In: ACM SACMAT 2014, pp. 17–28. ACM (2014)
12. Gu, T., Pung, H., Zhang, D.Q.: A service-oriented middleware for building context-aware services. J. Netw. Comput. Appl. **28**(1), 1–18 (2005)
13. Hella, L., Krogstie, J.: Using Semantic Web for Mobile Services Personalization. Int. J. u-and e-Serv. Sci. Technol. **7**(2), 221–238 (2014)
14. Hu, G., Wu, B., Chen, J.: Dynamic adaptation of business process based on context changes: a rule-oriented approach. In: PACEB Workshop @ ICSOC (2014)
15. Hummer, W., Inzinger, C., Leitner, P., Satzger, B., Dustdar, S.: Deriving a unified fault taxonomy for event-based systems. In: 6th ACM DEBS Conference (2012)
16. Hummer, W., Leitner, P., Satzger, B., Dustdar, S.: Dynamic migration of processing elements for optimized query execution in event-based systems. In: DOA (2011)
17. Hummer, W., Rosenberg, F., Oliveira, F., Eilam, T.: Testing idempotence for infrastructure as code. In: Eyers, D., Schwan, K. (eds.) Middleware 2013. LNCS, vol. 8275, pp. 368–388. Springer, Heidelberg (2013)
18. Hummer, W., Schulte, S., Hoenisch, P., Dustdar, S.: Context-aware data prefetching in mobile service environments. In: BDCloud Conference, pp. 214–221. IEEE (2014)
19. Inzinger, C., Hummer, W., et al.: Generic event-based monitoring and adaptation methodology for heterogeneous distributed systems. SPE **44**(7), 805–822 (2014)
20. Kemp, C., Gyger, B.: Professional Heroku Programming. Wiley, Chichester (2013)
21. Kumar, A., Yao, W.: Design and management of flexible process variants using templates and rules. Comput. Ind. **63**(2), 112–130 (2012)
22. Ouedraogo, W., Biennier, F., Merle, P.: Contextualised security operation deployment through mds@run.time architecture. In: ISC Workshop @ ICSOC (2014)
23. Pahl, C., Casey, M.: Ontology support for web service processes. ACM SIGSOFT Softw. Eng. Notes **28**, 208–216 (2003)
24. Swan, M.: Connected car: quantified self becomes quantified car. JSAN **4**(1), 2–29 (2015)
25. Wang, W., Zong, S., Yu, J., Yongchareon, S.: Modelling web service personalization with rule nets. In: Liu, C., He, J., Huang, G., Huang, Z. (eds.) WISE Workshops 2013. LNCS, vol. 8182, pp. 228–238. Springer, Heidelberg (2014)
26. Yu, J., Han, J., Sheng, Q.Z., Gunarso, S.O.: PerCAS: an approach to enabling dynamic and personalized adaptation for context-aware services. In: Liu, C., Ludwig, H., Toumani, F., Yu, Q. (eds.) Service Oriented Computing. LNCS, vol. 7636, pp. 173–190. Springer, Heidelberg (2012)