A Service Framework for Smart Mobility Scenarios

Stefan Schulte*, Philipp Hoenisch*, Kristof Kipp†, Daniel Burgstahler‡, Sven Abels§, Giuseppe Liguori¶
*Distributed Systems Group, TU Wien, Austria, {s.schulte, p.hoenisch}@infosys.tuwien.ac.at
†University of Bremen, Germany, kkipp@informatik.uni-bremen.de
‡Multimedia Communications Lab, TU Darmstadt, Germany, daniel.burgstahler@kom.tu-darmstadt.de
§Ascora GmbH, Germany, abels@ascora.de
¶SRM - Reti e Mobilita, Italy, giuseppe.liguori@srmbologna.it

***Abstract*—In Smart Mobility scenarios, users depend on reliable and timely provision with travel-related information. The application of service-oriented concepts for bringing this information to the end user is a promising approach. Nevertheless, there has been surprisingly little work on service frameworks which explicitly take into account specific demands of Smart Mobility.**

Within this paper, a software framework aiming at service consumption in Smart Mobility scenarios is presented. The framework features specific functionalities for such scenarios, taking into account both functional and non-functional requirements. This includes the means to push information from services to the end user, data prefetching mechanisms, and support of context-based adaptation. The applicability of the framework is demonstrated through representative use case scenarios.

***Index Terms*—Smart Mobility, Service-oriented Computing, Context-based Service Adaptation**

I. Introduction

Today, a multitude of data sources and services allows the interested user to gather mobility-related information [1]. These data sources and services range from the omnipresent online route guidance systems and online trip planners to sensors offering real-time information about the traffic situation at a location, vehicle telematics, cooperative systems combining information from different entities (like vehicles, infrastructure, drivers) to deliver services, or open (government) data sources [2]. Data from such sources is potentially beneficial for all people involved in some form in traffic and transportation, e.g., motorcyclists, car or truck drivers, bicyclists, pedestrians, and other road users.

Mobility information is very often consumed while the user is en route. Usually, this is done in an ad hoc manner on a mobile device like a smartphone [3]. Today, mobility-related data sources and services do not necessarily aim at interoperability and end-to-end integration [1], [4]. Even in simple use cases such as the integration of routing information needed by a navigation app and information from a travel schedule (e.g., stored in a personal online calendar), users have to cope with different Web content or apps, if not making use of proprietary, closed solutions like Google Now. Information the user receives from one app has to be inserted manually into other apps instead of being automatically transferred, making data handling unnecessarily difficult. Obviously, alone for safety reasons, too much interaction with different apps should be avoided, especially for vehicle drivers. Hence, end-to-end integration of data sources and software services is necessary to support users with a uniform, yet intuitive, interface to access mobility-related functionalities.

The usage of service mashups and compositions, as well as Cloud technologies to integrate data from different sources is a promising approach. While such techniques have been applied in many domains, e.g., manufacturing [5], [6], smart cities [7], or smart grids [8], to the best of our knowledge, there is little work on service frameworks for the Smart Mobility domain. Smart Mobility describes the integration of mobility-related data in order to provide energy-efficient, secure, individual, and comfortable mobility solutions [9], and can be treated standalone or as part of smart cities. Still, there are important domain specifics, which require attention: First, road users are especially affected by volatile network conditions, e.g., because a car is driving through a tunnel or crosses an area with low network coverage [10]. The integration of functionalities to mask insufficient connectivity is therefore an important prerequisite in Smart Mobility. Second, mobility-related information may change at a rapid pace, e.g., data about free parking spaces, traffic conditions, or delays of trains, needs to be communicated to mobile users immediately [11].

To take into account volatile network conditions, we propose the usage of *data prefetching*. Data prefetching allows to retrieve data before the data is actually required [12]. For prompt information updates, we integrate *push* functionalities. While pushing of information is very common in mobile computing [13], this approach is partially contradicting the request-response invocation pattern in Service-oriented Computing and is therefore rarely applied in service frameworks. Both functionalities – data prefetching and pushing of information – depend on the user context. Therefore, it is necessary to exploit data about the user context and integrate according mechanisms into a service framework for Smart Mobility.

To provide these functionalities, we present the SIMPLI-CITY Service Framework. This software framework features different functional and non-functional service adaptation mechanisms for Smart Mobility scenarios. To implement data prefetching, data pushing, and helper functionalities like monitoring and accounting in a transparent way, we extend a state-of-the-art service framework by several software components.

The remainder of this paper is organized as follows: First, we will comment on the related work in the fields of service frameworks, Smart Mobility, and context-based service adaptation (Section II). Afterwards, we define the design of the envisioned software framework (Section III) and present the framework implementation (Section IV). We assess the usability of the proposed framework by discussing exemplary use cases in Section V. Eventually, we identify upcoming research topics and conclude this paper (Section VI).

II. Related Work

Recently, Smart Mobility has gained a lot of attention from both the research community and the software industry. In short, Smart Mobility describes the usage of mobility-related data sources in order to increase the user experience. These mobility-related data sources are heterogeneous with regard to content and used technologies. Smart Mobility can be seen as an advancement of Intelligent Transportation Systems [9], which has also evolved into a data-driven discipline [1], [14].

From the perspective of the work at hand, two particular aspects of Smart Mobility are of importance: First, the trend towards integrating different mobility-related data services and related functional services and their provision to the user on a mobile device. Examples for this are, e.g., a mobility assistant aiming at promoting sustainable travel choices [15], the decentralized traffic information system (TIS) D4V [3], or the work by Gisdakis et al., who provide a privacy-preserving TIS [16]. While these solutions share a common basic idea with the SIMPLI-CITY Service Framework, they are not aiming at an open ecosystem. In contrast, the SIMPLI-CITY Service Framework is not limited to a particular use case, but offers the functionalities to integrate arbitrary services and data sources, and to provide the results to the end user.

Second, data integration is very often done using Internet of Things (IoT) technologies [17]. While transportation and logistics are named as primary application areas for IoT technologies [18], there are still only few service frameworks explicitly aiming at this area [19]. However, there are frameworks for smart cities, e.g., [7], [20], [21], which partially address the field of Smart Mobility. These frameworks provide complimentary functionalities to the SIMPLI-CITY Service Framework with regard to data integration and data processing. However, the named IoT frameworks omit functionalities to provide services to the mobile end user, since the frameworks focus on large-scale applications, e.g., smart city management.

The mobile usage of Web services and Cloud services has gained some momentum in recent years. One particular research focus is on the adaptation of communication between the Web or Cloud and a mobile device. For this, Papageorgiou et al. propose the automated selection of the most appropriate communication protocol in a particular context [22]. The same authors also propose the usage of reactive caching in order to decrease the amount of transmitted data [23]. Apart from reactive caching mechanisms, several researchers have worked on proactive caching, i.e., prefetching. In an early approach, Schreiber et al. propose a piggybacking approach for prefetching in SOAP messages [24], where prefetching is controlled by a server-based middleware. Other prefetching solutions apply a dual prefetching approach for Web services, i.e., on the client and on the server/proxy side [25], [26], [27]. The SIMPLI-CITY Service Framework is able to encapsulate such mechanisms in order to increase the Quality of Service (QoS) or Quality of Experience (QoE) of the end user.

To the best of our knowledge, there has only been little work on pushing technologies for information updates from Web services. The energy efficiency of pushing and polling of data has been investigated by Burgstahler et al. [28], [29]. A comparison of popular push services is presented in [30].

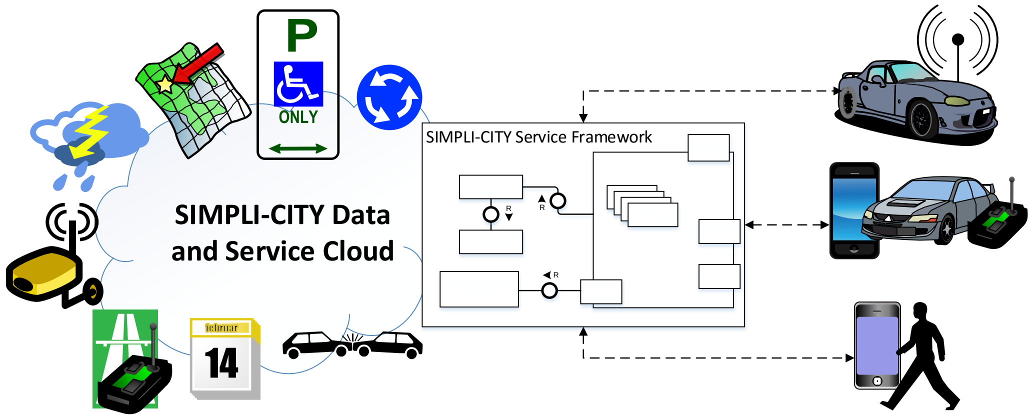
While the aforementioned research has focused on single functionalities to improve the usage of Web and Cloud services on mobile devices, there has also been some work on service frameworks supporting mobile service consumption. An according extension to the OSGi framework called *MA-OSGi* has been proposed by Lee et al. [31]. The main motivation for MA-OSGi is the need to invoke services remotely, which is a prerequisite to use OSGi-based services on a mobile device. MA-OSGi provides four major features, aiming at (i) energy efficiency of service consumption, (ii) the definition of mobile itineraries, (iii) service matchmaking to take into account context changes, and (iv) asynchronous communication for reliable communication. While MA-OSGi requires explicitly defined itineraries, the SIMPLI-CITY Service Framework takes into account implicit and context information to adapt to the user's needs. Despite the fact that MA-OSGi is not explicitly aiming at mobility scenarios, this research comes closest to our own work. Interestingly, MA-OSGi and our own work largely complement each other, i.e., a future integration of the different functionalities is reasonable.

In many cases, mobility demands depend on the views and options of a single person. Therefore, as a third major area of related work for the SIMPLI-CITY Service Framework, context-based service adaptation needs to be discussed. Most adaptation and personalization approaches for services focus on the user's location. The possible applications range from data tailoring [32] to the offloading of computational tasks [33]. We refer for an in-depth discussion on context-based adaptation for mobility scenarios to [34]; a broader overview can be found in [35], [36]. However, it should be noted that the current state-of-the-art in the field of context-based adaptation for Smart Mobility focuses on single applications and use cases, while we aim at providing an extensible framework which supports arbitrary adaptation and personalization mechanisms.

To sum up the discussion of the related work, there are a number of single functionalities related to the SIMPLI-CITY Service Framework, but none of them provides a holistic service framework.

III. Design Overview

In the following subsections, we discuss the basic design considerations to implement the three main functionalities mentioned above, i.e., context-based service adaptation in general, pushing of information, and data prefetching. Details about their integration into the SIMPLI-CITY Service Framework and implementation details are provided in Section IV.



SIMPLI-CITY Data and Service Cloud
SIMPLI-CITY Service Framework
P
ONLY
februar
14

Fig. 1: Scenario Overview

As it can be seen in Figure 1, the SIMPLI-CITY Service Framework aims at exploiting mobility-related online data sources like sensors, cooperative systems, telematics, or open data repositories. Concrete example data sources as depicted in Figure 1 might be (but are not limited to) live traffic information including traffic accidents, data from infrastructure sensors, personal calendar data, map and route data, weather information, and many more. In addition, vehicle data could also be exploited. Data sources may be directly or indirectly accessed via value-adding services, but in any case, through the SIMPLI-CITY Service Framework.

In Smart Mobility scenarios, services and data sources are often consumed on mobile devices like smartphones. The tremendous success of mobile apps has shown that end users are willing to either pay for software functionalities or accept advertisements, which is also an important revenue stream for app developers. Naturally, the commercial exploitation of data sources and Web services is of interest to data and service providers alike. Hence, a Smart Mobility service framework needs to be able to support monitoring of service invocations for accounting purposes. This allows different pricing models like pay-per-use or flatrates and reduces the overhead for software developers. Service invocations need to be monitored with regard to the success of the actual invocation as well as QoS properties in order to make sure that predefined Service Level Agreements (SLAs) between service consumer and service provider are respected.

Since the focus in Smart Mobility scenarios is on mobile consumption of online (Web) services, there is a clear distinction between the server-side and the client-side: While the server-side provides more heavyweight functionalities, the client-side is by design a lightweight thin client. This means that large computational efforts, the actual business logic, etc. are done on the server-side, while the client-side provides the frontend to these functionalities. Integration of non-local data sources is also done on the server-side. However, it should be noted that the integration of sensor data from the client device (e.g., GPS data on a smartphone) could be done on both the server-side and the client-side.

With the Google-lead Open Automotive Alliance[1] or Apple CarPlay[2], direct interaction with apps via a car's dashboard unit is very likely to enter the mass market in the near future. Hence, the smartphone may act as an intermediary between the car and the Internet, or the car may directly communicate with the Internet. As depicted in Figure 1, the SIMPLI-CITY Service Framework can deliver its functionalities independent from the technical nature of the client device, i.e., it does not matter if the client device is a smartphone or an in-vehicle (entertainment) system. In general, the user requires a stable and reliable Internet connection, which may not always be available. In order to mitigate this risk, prefetching mechanisms are integated into the SIMPLI-CITY Service Framework, as discussed in Section III-B.

Last but not least, Smart Mobility scenarios naturally depend on the remote invocation of services. While this appears to be a frequently requested functionality in service frameworks, it should be noted that standard framework specifications like the OSGi framework indeed do not support remote service invocation if the client is not an OSGi container itself. This means that without according extensions, a mobile client cannot directly access services hosted by a OSGi-based service framework. To enable remote invocation of services, a service framework for Smart Mobility scenarios needs to be extended by a proxy component which acts as a gateway for remote service invocations. As positive side effect, a proxy is able to provide additional functionalities, e.g., for retrieving information about the client like the location (see below), network conditions, etc. Also, the monitoring of service invocations as mentioned above can be achieved via proxy functionalities.

Summarized, the basic design considerations of the SIMPLI-CITY Service Framework are as follows:

- The SIMPLI-CITY Service Framework allows the exploitation of arbitrary data sources.
- While the framework may also be accessed from stationary computers, user interaction is primarily done remotely

[1]http://www.openautoalliance.net/
[2]http://www.apple.com/ios/carplay/

via mobile clients.
- Hence, the server-side (i.e., the framework itself) delivers the main functionalities, while the client-side is a lightweight thin client.
- Monitoring and accounting functionalities are integrated in order to allow different business models for service and data providers.

After the basic design considerations of the SIMPLI-CITY Service Framework have been defined, we are now able to discuss the three major functionalities which are necessary to provide mobile users with on-time, personalized data in a Smart Mobility scenario, i.e., context-based service adaptation, data prefetching, and pushing of information to the client.

A. *Context-based Service Adaptation*

In Smart Mobility scenarios, context-based service adaptation and personalization can be done with different goals in mind: Obviously, if the user is mobile, especially the current and future location of the user is of primary importance. The location can be used for context-based adaptations both with regard to non-functional aspects (e.g., adapting the communication between a service framework and a client device [22]), and functional aspects (e.g., to provide the user with location-aware information updates [37]).

According to Baldauf et al. [38], the detection of context changes and the actual usage of this information should be separated in order to allow extensibility of a system. Following a service-based data integration approach, the SIMPLI-CITY Service Framework supports this feature. Also, distributed context-based systems should provide a middleware and extend it by a *context server*, which allows concurrent data access by different parties. In a multi-user Smart Mobility scenario, this can be done, e.g., by providing publish-subscribe functionalities, which decouple sender and receiver. Also, a publish-subscribe middleware can be used to detect context changes, while the information is used on a mobile device.

To recognize context data changes as well as the unavailability of external data sources, a context-based system needs to be able to monitor arbitrary context data sources. This is usually achieved by using *logical sensors* [38]. Logical sensors combine information from physical sensors with business logic, e.g., to prefilter data. This is helpful in order to decrease the amount of data which needs to be handled by the context middleware and therefore increases the scalability of the overall system. Also, one logical sensor may encapsulate a multitude of physical sensors which are placed in different locations. Thus, location-aware sensor selection is enabled – a client interacts with a service wrapping the logical sensor instead of directly interacting with various physical sensors. By supporting late binding and loose coupling, different physical sensors may be used within the same software context, e.g., in the same app. Finally, logical sensors are not limited to solely encapsulate physical sensors. In fact, any context data source, e.g., open (government) mobility data, could be provided via a logical sensor.

B. *Data Prefetching*

Data prefetching allows to retrieve data from online sources before it is actually needed on a client device. This is especially important in Smart Mobility scenarios, where the user usually gets information updates on her mobile device. Prefetching could be done for short-term scenarios, i.e., if prefetching needs to be done for a short time period, or for long-term scenarios, i.e., if prefetching needs to be done for a longer amount of time. Scenarios for short-term prefetching include driving through a tunnel or another clearly identifiable area with limited network coverage, while long-term prefetching could be done, e.g., if the user needs to save data volume and therefore prefers to use WiFi instead of mobile networks, lingers in a large area with limited connectivity, or is abroad and wants to avoid roaming fees.

Especially in long-term prefetching, it is not really helpful to prefetch volatile data, since there is a high chance that this data is invalid once it is actually needed [39]. For instance, the prefetching of traffic data, which is constantly changing, is only of limited value. However, there is more stable data, e.g., map data, media data, or historical traffic data [40], where prefetching is helpful.

It should be noted that the success of data prefetching significantly depends on the effectiveness of the prefetching decision algorithm, i.e., an algorithm to decide which data items should (not) be prefetched at a particular point of time [12]. On the one hand, if data items eligible for prefetching are prefetched too early, the data will not be fresh any longer, i.e., the user will receive expired data. If data is not prefetched at all, there will be information gaps. On the other hand, prefetching of unsuitable data items leads to unnecessary energy and storage consumption at the client-side. Hence, optimization of data prefetching is important in order to avoid negative side-effects. For particular algorithms for data prefetching in mobility scenarios, which are able to take into account the current and future locations of the user, we refer to our former work [39], [41].

C. *Pushing*

Context-enabled pushing of information allows proactive user notifications, i.e., to recognize in which situations a user may be provided with a certain piece of information. In Service-oriented Computing, services usually follow a pull-based request-response invocation, while push-based information updates are common in mobile computing, e.g., as implemented for the Android platform in Google Cloud Messaging[3].

Instead of regularly pulling data from a service, which may not lead to any information gain, information should only be pushed to the client if it provides a significant update. The significance can be determined manually or automatically, based again on the user context. For instance, the user's network conditions could define if information updates should be pushed in any case or only when connectivity is above a predefined level. Another benefit of push-based notifications

[3]https://developers.google.com/cloud-messaging/

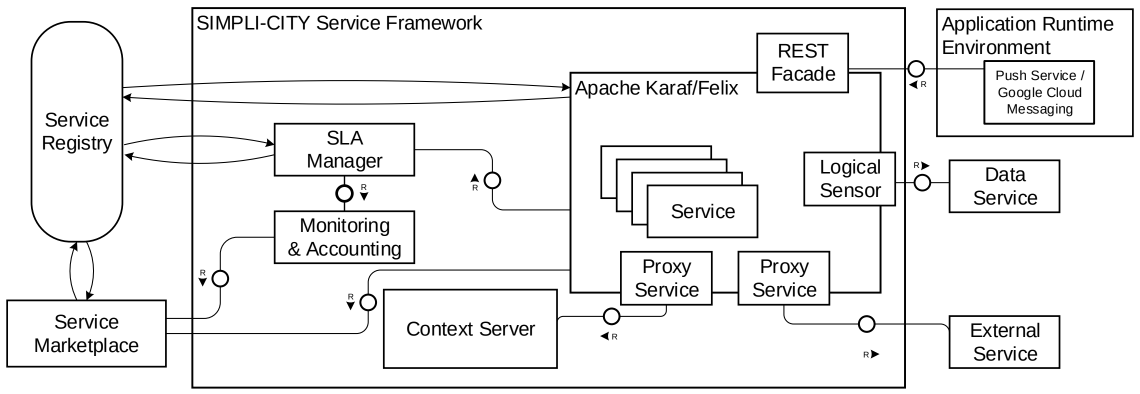


Fig. 2: SIMPLI-CITY Service Framework – Architecture

is the fast availability of information updates. While this could also be achieved via frequent pulling, this is potentially battery-draining on mobile devices. However, it needs to be noted that pushing does not always lead to better energy efficiency than pulling [28]. Therefore, pushing should primarily be done in scenarios where latency is of importance [42].

To push data to a client, at least two software components are needed: First, a *management service* that permits clients (here: the mobile device of a user) to register for or unregister from push notifications, possibly specifying the information which is of interest. This service can follow a traditional request-response approach, i.e., it is only executed when requested. Second, an actual *notification service* establishes the link between an arbitrary data source (e.g., a traffic information-providing logical sensor) and pushes relevant notifications to the registered clients. The latter service has to be a continuous background service of the SIMPLI-CITY Service Framework, thus minimizing latency between data updates and push notifications.

IV. Implementation

In order to realize the functionalities discussed in Section III, the SIMPLI-CITY Service Framework has been implemented as an extension to the OSGi framework (R4)[4]. In general, the OSGi specification is not tailored towards a specific application area, but allows the bundling of arbitrary software functionalities as separate modules. While this makes OSGi-based frameworks very flexible, it also implies that there are no specific functionalities aiming at mobile service consumption or mobility scenarios. Therefore, it was necessary to implement the functionalities discussed in Section III as extensions, i.e., support of pushing, prefetching, context-based service adaptation, the integration of remote service access, and monitoring and accounting capabilities.

It should be noted that we have implemented prototypical apps, which interact with the SIMPLI-CITY Service Framework, for mobile devices running the Android operating system. However, as discussed in Section III, arbitrary client devices could interact with the SIMPLI-CITY Service Framework, as long as the clients are able to invoke the RESTful services hosted by the framework.

Figure 2 shows an overview of the main components of the SIMPLI-CITY Service Framework (in FMC Notation[5]), which will be discussed in more detail in the following paragraphs. Apache Felix[6] is used as the core OSGi framework implementation, while Apache Karaf[7] is used as the runtime container, providing higher level features specifically designed for creating OSGi-based servers.

The most important extension provided by the SIMPLI-CITY Service Framework is the *REST Facade*. This component provides external users with a single point of entry for invoking services hosted in the SIMPLI-CITY Service Framework, and therefore allows the necessary remote service invocations for mobile clients. The REST Facade provides a RESTful interface, and enables proxy functionalities. The REST Facade invokes the *SLA Manager* upon every service invocation. After completion of a service invocation, whether successful or not, the REST Facade will call the SLA Manager again to let it know the invocation has been finished. The SLA Manager in turn calls the *Monitoring & Accounting* subcomponent, which controls that the Service Level Objectives (SLOs) defined for a particular service are met. If there is a SLA violation or the service is not responsive, the SLA Manager can start according countermeasures, e.g., restart a service instance. As another important functionality, the Monitoring & Accounting subcomponent provides billing information to the Service Marketplace (see below), if a pay-per-use pricing model has been defined for a particular service. In any case, the REST Facade checks if there is a valid license for a service invocation, before the service is actually invoked.

Three different types of software services run within the SIMPLI-CITY Service Framework: *(Internal) Services*, *Proxy Services*, and *Logical Sensors*. (Internal) Services are simply

[4]https://www.osgi.org/release-4-version-4-3/

[5]http://www.fmc-modeling.org/

[6]http://felix.apache.org/

[7]http://karaf.apache.org

software services running within the OSGi container, which do provide a specific business logic. Proxy Services are also running within the OSGi container, but delegate their functionality to External Services. Proxy Services can aggregate data from several Internal and External Services, relay service calls to External Services, translate input from a mobile client to the format an External Service requests (and vice versa), and interact with the Context Server (see below) in order to enable different context-based service adaptations, including late binding of Data Services. Logical Sensors are a specific kind of Proxy Services, which provide functionalities to prefilter and fuse data. Also, sensor abstraction and interoperability mechanisms are provided by Logical Sensors in order to wrap different sensor technologies and data formats. In contrast to the three service types running within the SIMPLI-CITY Service Framework, *Data Services* and *External Services* are not hosted by the framework, but might nevertheless be invoked via the SIMPLI-CITY Service Framework, using proxy functionalities. External Services are equivalent to Internal Services, but hosted externally. Data Services are a specific subtype of External Services, and encapsulate a particular, arbitrary data source, e.g., a specific sensor.

The *Context Server* allows concurrent access to different data sources wrapped by Logical Sensors and acts as the basic management service needed for the pushing of information (see Section III-C). To provide only significant information updates to interested users, the core of this component is an Apache Camel-based publish-subscribe middleware[8]. Importantly, Camel allows the definition of rules for subscriptions to particular Logical Sensors, e.g., "all data from Logical Sensor X with a value lower than Y". This is actually the basic functionality for pushing data to a client device. For this, the user (respectively the app running on her device) signs up for particular data updates at the Context Server. Apart from enabling pushing of information to the user, the Context Server also facilitates further context-based service adaptation functionalities, namely location-based data service selection, which allows the adaptation of a service's output based on the location of a user by late binding to the correct data source or service. Also, data prefetching functionalities may be invoked through the Context Server.

The *Application Runtime Environment* (ARE) provides the notification service needed for pushing of information (see Section III-C). The notification service is encapsulated in the *Push Service* subcomponent. Notably, the ARE is not an integral part of the SIMPLI-CITY Service Framework. Instead, it is hosted on a different Cloud server. This has been done in order to provide scalability for the ARE. The Push Service allows services running in the SIMPLI-CITY Service Framework to notify mobile apps in case of significant events, as determined when a client (respectively an app running on a client device) has signed up for receiving push messages. The server-based Push Service has been implemented using Google Cloud Messaging for the delivery and distribution of messages to the different client devices. Google Cloud Messaging has been chosen since it allows an in-depth integration with the mobile operating system, i.e., Android. Furthermore, it provides a messaging infrastructure for the queuing of messages. Thus the integration of push messages into the aforementioned prototypical Android-based client apps can be achieved straightforward.

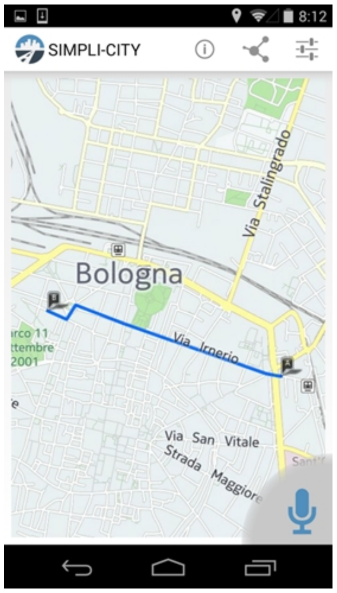
Last but not least, the *Service Registry* and *Service Marketplace* offer helper functionalities to establish a business environment for Smart Mobility services. The Service Registry allows to store software service artifacts along with their description in order to be accessible by services or apps. It provides Java and RESTful interfaces to create, update or delete services including their descriptions. Thus, software developers can deploy their services in the SIMPLI-CITY Service Framework, update the service description of the whole service, or delete everything. The Service Registry is based on the registry which is already part of the OSGi framework, but extends the standard implementation by means for service management and extended service descriptions. These descriptions provide the basic information for advertising services on the Service Marketplace, i.e., the descriptions primarily include advertisement-related information including figures, videos, and payment models.

V. Use Cases

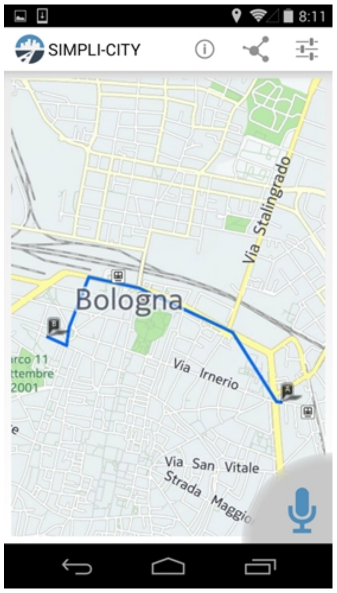
As proof-of-concept, different use case services and mobile apps have been implemented during the course of the implementation of the SIMPLI-CITY Service Framework. In the following paragraphs, two exemplary functionalities are briefly discussed. These have been chosen in order to illustrate the major functionalities of the SIMPLI-CITY Service Framework as discussed in Sections III and IV. In general, arbitrary services can be implemented and hosted within the SIMPLI-CITY Service Framework. For example, apart from the services described below, further use case services covering environmental awareness rising, information pushing about nearby points of interests, cognitive driving, and routing [40] have been implemented. All these use cases take into account the user context and/or user preferences.

a) Use Case I: Location-based Service Personalization: In many Smart Mobility scenarios, the user wants to be provided with data that is related to her (current) location. For example, if a user is looking for a parking space or bike sharing facilities in Barcelona, data sources providing information about parking or bike sharing in Amsterdam will not help. There are two ways to achieve that the user is provided with the right data depending on her current location: Either, there could be a separate software service and end user app for each city, which are tightly coupled to each other and the local data sources. However, this means that the user needs to download and install a new app for each city. The second solution is to provide one general end user app, which invokes Data Services depending on the location of the user. Within the SIMPLI-CITY Service Framework, the second option is directly facilitated: The Context Manager supports the end user

[8]http://camel.apache.org/



(a) With Permission



(b) Without Permission

Fig. 3: Use Case II End User App – Example Screenshots

app by looking up the right External (Data) Service based on the user's location. For this, a Proxy Service or Logical Sensor (as discussed in Section IV) is generated, which reroutes user requests accordingly to the correct Data Service, and maps the input and output data based on the requirements of the Data Service.

Another example for location-based service personalization is data prefetching, since the actual decision when prefetching is meaningful is also based on the user's current and future location. For use cases on data prefetching, refer to [39], [41].

b) Use Case II: Personalized Traffic Restrictions: The second use case deals with personalized traffic restrictions. Such restrictions apply in many European cities. Permissions to enter a particular area may change on a daily basis or several times during the day, or be adapted in real-time based on the amount of traffic in a certain area. Different traffic restrictions may apply to different groups of people, e.g., residents, commuters, tourists, and deliverymen all have different permissions. Also, the type of vehicle may be taken into account, e.g., the emission level or the power source (gasoline, electric vehicle, etc.) of a particular car.

Hence, even residents may not be aware if they are allowed to enter a certain area at a certain point of time. In order to overcome this issue, according use case services and an end user app have been implemented for the SIMPLI-CITY Service Framework, using data sources from the Municipality of Bologna, Italy.

From a technical point of view, different functionalities of the SIMPLI-CITY Service Framework are used in this scenario. First, an Internal Service running in an OSGi container provides the business logic identifying if an individual user is allowed to enter a certain area. The Internal Service makes use of the Context Server in order to gather the necessary user-related information to make this decision, i.e., context data about the user and her car. This includes information if the user owns a permit to enter the area at a certain point of time. The user interacts with the Internal Service via an end user app (see Figure 3). In order to get real-time updates from the Internal Service, the end user app subscribes to information updates via the Push Service and Context Server (see Section IV).

As a proof-of-concept implementation, the Internal Service is used within a routing algorithm. As it can be seen in Figure 3, different users get different routing instructions in the end user app. Figure 3a shows the route for a user who is allowed to cross the city center of Bologna (and especially Via Irnerio), while Figure 3b shows a route for a user who is not allowed to do so. The screenshots have been taken from the end user app mentioned above.

VI. Conclusion

Within this paper, we have discussed the specific needs of service frameworks in Smart Mobility scenarios. A special focus has been on the invocation of mobility services via mobile devices like smartphones. We have presented the SIMPLI-CITY Service Framework, which implements functionalities addressing the needs of Smart Mobility scenarios, namely remote mobile access of services, monitoring and accounting functionalities, context-based service adaptation, pushing of information, and data prefetching.

The SIMPLI-CITY Service Framework has been implemented as an extensible software system with open interfaces. Therefore, the integration of further functionalities, e.g., like the ones offered by MA-OSGi (as discussed in Section II), is a promising starting point for our future work. We are also looking for ways to extend the framework's basic toolset: At the moment, the SIMPLI-CITY Service Framework provides a centralized architecture, i.e., all service requests are routed through the framework. However, if the mobile client directly needs to interact with the environment in order to exploit infrastructure data sources, it might be sensible to deliver some services in a decentralized manner. To substantially reduce latency between data sources and the client's device, it is necessary to distribute heavyweight computing tasks between local devices, e.g., a smartphone, the in-vehicle entertainment system of a car, or sensor motes, instead of invoking services hosted in the Cloud. With the advent of Fog Computing, i.e., the possibility to bring virtualization and resource sharing to the edge of the network, this might be feasible. Hence, we want to investigate the usage of Fog Computing for distributed data integration in Smart Mobility scenarios.

Acknowledgment

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

The SIMPLI-CITY Service Framework and extensive documentation can be downloaded from http://simpli-city.eu/deliverables.html.

References

[1] J. Zhang, F.-Y. Wang, K. Wang, W.-H. Lin, X. Xu, and C. Chen, "Data-Driven Intelligent Transportation Systems: A Survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 12, no. 4, pp. 1624–1639, 2011.

[2] N. Shadbolt, K. O'Hara, T. Berners-Lee, N. Gibbins, H. Glaser, W. Hall, and M. Schraefel, "Linked Open Government Data: Lessons from Data.gov.uk," *IEEE Intelligent Systems*, vol. 27, no. 3, pp. 16–24, 2012.

[3] M. Picone, M. Amoretti, and F. Zanichelli, "A Decentralized Smartphone Based Traffic Information System," in *2012 Intelligent Vehicles Symposium*, 2012, pp. 523–528.

[4] F. Lécué, R. Tucker, V. Bicer, P. Tommasi, S. Tallevi-Diotallevi, and M. Sbodio, "Predicting Severity of Road Traffic Congestions Using Semantic Web Technologies," in *11th Extended Semantic Web Conference (ESWC)*, 2014, pp. 611–627.

[5] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 1, pp. 62–70, 2005.

[6] S. Schulte, P. Hoenisch, C. Hochreiner, S. Dustdar, M. Klusch, and D. Schuller, "Towards Process Support for Cloud Manufacturing," in *18th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2014)*, 2014, pp. 142–149.

[7] F. Li, M. Vögler, S. Sehic, S. Qanbari, S. Nastic, H. L. Truong, and S. Dustdar, "Web-Scale Service Delivery for Smart Cities," *IEEE Internet Computing*, vol. 17, no. 4, pp. 78–83, 2013.

[8] M. Postina, S. Rohjans, U. Steffens, and M. Uslar, "Views on Service Oriented Architectures in the Context of Smart Grids," in *First IEEE International Conference on Smart Grid Communications (SmartGridComm 2010)*, 2010, pp. 25–30.

[9] A. Sassi and F. Zambonelli, "Coordination Infrastructures for Future Smart Social Mobiliy Services," *IEEE Intelligent Systems*, vol. 29, no. 5, pp. 78–82, 2014.

[10] W. L. Tan, F. Lam, and W. C. Lau, "An Empirical Study on 3G Network Capacity and Performance," in *26th IEEE International Conference on Computer Communications (INFOCOM 2007)*, 2007, pp. 1514–1522.

[11] I. Leontiadis and C. Mascolo, "Opportunistic Spatio-Temporal Dissemination System for Vehicular Networks," in *1st International MobiSys Workshop on Mobile Opportunistic Networking*, 2007, pp. 39–46.

[12] B. D. Higgins, J. Flinn, T. J. Giuli, B. Noble, C. Peplin, and D. Watson, "Informed Mobile Prefetching," in *10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 155–168.

[13] I. Podnar, M. Hauswirth, and M. Jazayeri, "Mobile Push: Delivering Content to Mobile Users," in *22nd International Conference on Distributed Computing Systems (ICDCSW '02) – Workshops*, 2002, pp. 563–570.

[14] N.-E. El Faouzi, H. Leung, and A. Kurian, "Data fusion in intelligent transportation systems: Progress and challenges – A survey," *Information Fusion*, vol. 12, pp. 4–10, 2011.

[15] D. Magliocchetti, M. Gielow, F. D. Vigili, G. Conti, and R. D. Amicis, "A Personal Mobility Assistant based on Ambient Intelligence to Promote Sustainable Travel Choices," *Procedia Computer Science*, vol. 5, pp. 892–899, 2011.

[16] S. Gisdakis, V. Manolopoulos, S. Tao, A. Rusu, and P. Papadimitratos, "Secure and Privacy-Preserving Smartphone-Based Traffic Information Systems," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 3, pp. 1428–1438, 2015.

[17] L. D. Xu, W. He, and S. Li, "Internet of Things in Industries: A Survey," *IEEE Transactions on Industrial Informatics*, vol. 10, no. 4, pp. 2233–2243, 2014.

[18] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54, pp. 2787–2805, 2010.

[19] J. A. G. Ibáñez, S. Zeadally, and J. Contreras-Castillo, "Integration Challenges of Intelligent Transportation Systems with Connected Vehicle, Cloud Computing, and Internet of Things Technologies," *IEEE Wireless Communications*, vol. 22, no. 6, pp. 122–128, 2015.

[20] A. Krylovskiy, M. Jahn, and E. Patti, "Designing a Smart City Internet of Things Platform with Microservice Architecture," in *3rd International Conference on Future Internet of Things and Cloud, (FiCloud 2015)*, 2015, pp. 25–30.

[21] M. Vögler, J. M. Schleicher, C. Inzinger, S. Nastic, S. Sehic, and S. Dustdar, "LEONORE – Large-Scale Provisioning of Resource-Constrained IoT Deployments," in *2015 IEEE Symposium on Service-Oriented System Engineering (SOSE 2015)*, 2015, pp. 78–87.

[22] A. Papageorgiou, A. Miede, S. Schulte, D. Schuller, and R. Steinmetz, "Decision Support for Web Service Adaptation," *Pervasive and Mobile Computing*, vol. 12, pp. 197–213, June 2014.

[23] A. Papageorgiou, M. Schatke, S. Schulte, and R. Steinmetz, "Lightweight Wireless Web Service Communication Through Enhanced Caching Mechanisms," *International Journal of Web Services Research*, vol. 9, no. 2, pp. 42–68, 2012.

[24] D. Schreiber, A. Göb, E. Aitenbichler, and M. Mühlhäuser, "Reducing User Perceived Latency with a Proactive Prefetching Middleware for Mobile SOA Access," *International Journal of Web Services Research*, vol. 8, no. 1, pp. 68–85, 2011.

[25] X. Liu and R. Deters, "An Efficient Dual Caching Strategy for Web Service-Enabled PDAs," in *2007 ACM Symposium on Applied Computing (SAC)*, 2007.

[26] M. S. Qaiser, P. Bodorik, and D. N. Jutla, "Differential Caches for Web Services in Mobile Environments." in *9th IEEE International Conference on Web Services (ICWS 2011)*, 2011, pp. 644–651.

[27] N. D. R. Armstrong and P. A. S. Ward, "Just-In-Time Push Prefetching: Accelerating the Mobile Web," in *27th IEEE International Conference on Advanced Information Networking and Applications (AINA 2013)*, 2013, pp. 1064–1071.

[28] D. Burgstahler, U. Lampe, N. Richerzhagen, and R. Steinmetz, "Push vs. Pull: An Energy Perspective," in *6th IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2013)*, 2013, pp. 190–193.

[29] D. Burgstahler, N. Richerzhagen, F. Englert, R. Hans, and R. Steinmetz, "Switching Push and Pull: An Energy Efficient Notification Approach," in *IEEE Third International Conference on Mobile Services (MS 2014)*, 2014, pp. 68–75.

[30] W. Chen, S. Zhou, Y. Tang, and L. Yu, "On Measuring Cloud-based Push Services," *International Journal of Web Services Research*, vol. 13, no. 1, pp. 53–68, 2016.

[31] J. Lee, S.-J. Lee, H.-M. Chen, and K.-H. Hsu, "Itinerary-Based Mobile Agent as a Basis for Distributed OSGi Services," *IEEE Transactions on Computers*, vol. 62, no. 10, pp. 1988–2000, 2013.

[32] C. Bolchini, C. A. Curino, E. Quintarelli, F. A. Schreiber, and L. Tanca, "A Data-oriented Survey of Context Models," *ACM SIGMOD Record*, vol. 36, no. 4, pp. 19–26, 2007.

[33] M. R. Rahimi, N. Venkatasubramanian, and A. V. Vasilakos, "MuSIC: Mobility-Aware Optimal Service Allocation in Mobile Cloud Computing," in *IEEE Sixth International Conference on Cloud Computing (CLOUD 2013)*, 2013, pp. 75–82.

[34] W. Hummer and S. Schulte, "Context-Aware Personalization for Smart Mobile Cloud Services," in *2nd Workshop on Intelligent Service Clouds (ISC) at the 13th International Conference on Service Oriented Computing (ICSOC)*, 2015, pp. 171–183.

[35] H.-L. Truong and S. Dustdar, "A Survey on Context-aware Web Service Systems," *International Journal of Web Information Systems*, vol. 5, no. 1, pp. 5–31, 2009.

[36] P. Bellavista, A. Corradi, M. Fanelli, and L. Foschini, "A Survey of Context Data Distribution for Mobile Ubiquitous Systems," *ACM Computing Surveys*, vol. 44, no. 4, 2012.

[37] M. Kenteris, D. Gavalas, and D. Economou, "An innovative mobile electronic tourist guide application," *Personal and Ubiquitous Computing*, vol. 13, pp. 103–118, 2009.

[38] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, pp. 263–277, 2007.

[39] W. Hummer, S. Schulte, P. Hoenisch, and S. Dustdar, "Context-Aware Data Prefetching in Mobile Service Environments," in *The 4th IEEE International Conference on Big Data and Cloud Computing (BDCloud 2014)*, 2014, pp. 214–221.

[40] F. Lécué, S. Tallevi-Diotallevi, J. Hayes, R. Tucker, V. Bicer, M. L. Sbodio, and P. Tommasi, "Smart Traffic Analytics in the Semantic Web with STAR-CITY: Scenarios, System and Lessons Learned in Dublin City," *Journal of Web Semantics*, vol. 27–28, pp. 26–33, 2014.

[41] M. Borkowski, O. Skarlat, S. Schulte, and S. Dustdar, "Prediction-Based Prefetch Scheduling in Mobile Service Applications (forthcoming)," in *IEEE 5th International Conference on Mobile Services (MS 2016)*, 2016.

[42] D. Hasenfratz, A. Meier, M. Woehrle, M. Zimmerling, and L. Thiele, "If You Have Time, Save Energy with Pull," in *8th ACM Conference on Embedded Networked Sensor Systems (SenSys '10)*, 2010, pp. 423–424.