A Dynamic Service Module Oriented Framework for Real-World Situation Representation

Peter Halbmayer and Gerold Hoelzl and Alois Ferscha
Institute for Pervasive Computing
Johannes Kepler University Linz
Linz, Austria
{halbmayer, hoelzl, ferscha}@pervasive.jku.at

***Abstract*—The maintenance of context information for real-world environments contains several challenges when it has to be computationally observed. Any event that is observable in the real-world has to be registered and may lead to transitions in the digital system. Therefore, a representation of the environment, the affected users, their whereabouts and their interactions with the system is required. A software framework is presented that provides a generic set of methods to collect information from multiple, heterogeneous sensors deployed within the environment. A non-deterministic communication topology is established, which handles a distributed version of a system state on a best effort basis. The system is designed to be potentially applied within various environments and the primary application scenario is represented by the implicit energy management in single-family homes. There the practical deployment of the system proves the usability and sustainability of the presented approach.**

***Keywords*—Dynamic Device and Service Discovery; Dynamic Context Recognition; Adaptive System Behavior; Flexible Power Management; Opportunistic Sensing.**

I. Introduction

The PowerIT system [1][2] is a development that provides implicit energy management in households on the behalf of users living in the environment. For this purpose, different aspects need to be handled in real-time like the activities users perform, the situations they are in, the energy that is drawn by household gadgets and also the interaction with notification and services is an important aspect.

The main contribution of the system is the processing of dense context information for practical and ubiquitous digital system operation without explicit user interaction. An important aspect in this sense is to provide unobtrusive operation in the background for optimal and convenient system utilization. Thus the digital system moves to the background and the users focus is directed to actual every day tasks at hand instead of being concerned with system maintenance.

The deployment diagram depicted in Figure 1 presents a schematic overview of how the system is generally setup. A base infrastructure that is comprised of a Home Server (HS) that is connected to a set of energy meter and control devices, build the core of the system. The energy meter and control devices are implemented as wall plug outlets that are used to connect any household gadgets, measure their energy consumption and switch their state on or off. The user is integrated into the system by collecting and evaluating data from the mobile devices that are carried along. A typical system installation consists of 20 wall plug outlets, a single HS, and a smart watch - smartphone/tablet combination for every participant which lies around four.

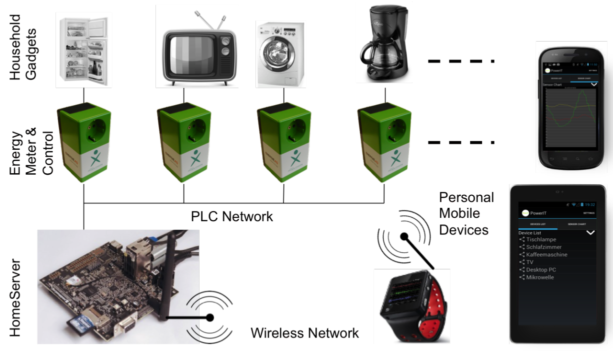


Figure 1. Deployment Diagram

The connection of mobile devices (e.g., smartphone or smart watch) to the base infrastructure is of a dynamic nature as users enter and leave the environment in an unpredictable way. This is also true for interconnections between mobile devices themselves (e.g., user take smartphone away and grabs it later). Therefore, such cases have to be handled with special precautions. The corresponding underlying network has to adapt itself automatically for seamless service provision.

From this description, the main aspects the system has to handle in software can be drawn. They are:

(i) Activity and context recognition
(ii) Opportunistic sensor and actuator management
(iii) A rule engine for implicit control.

Dynamic and adaptive sensor and actuator management builds the base for context recognition and implicit control. When an event is sensed (e.g., a user leaving the house), control mechanisms have to trigger to switch the state of devices (e.g., turn off all lights). To achieve the transport of corresponding messages and commands, the communication topology must be available to guarantee the successful delivery of messages. The dynamics of users have to be maintained in the system by unregistering users when leaving and (re-) registering them when they return.

As the practical usability of the system was one of the main interests in our project, it was necessary to determine the most important aspects to obtain an implementation that can withstand real-world conditions. In a previous project [3], where similar studies were conducted on a smaller scale than in PowerIT, it has been seen that certain characteristics need to be regarded to ensure the success of such a system. This was learnt from user questionnaires as well as practical system application in real-world sites.

Qualitative requirements that are necessary in such a system are

(i) The recoverability and fault tolerance of the system,
(ii) The unobtrusive operation and minimization of configuration and maintenance steps and
(iii) The organized reporting of failure in case of unrecoverable states entered.

A software model that addresses these aspects is required for practical realization of the system. Functionalities to build up basic communication streams, analyze collected data in real-time and forward control commands to dedicated endpoints are required. Therefore, in the following sections, first, related work is reviewed which is followed by a detailed system architecture description comprised of a system overview, system dynamics regarding the roles of devices and temporary registration, as well as activity recognition for system control. The final conclusion evaluates the results found in this work.

II. Related Work

The requirements mentioned in the last section are of vital importance for the realized system. Existing work already covers the core technology that is necessary to let devices exchange information between each other on an ad-hoc basis. Here, part of this technology is used in a practical context and concerned with the application of the implemented methods in real-world deployments. Building up on this base, a dynamic middleware was developed that (i) is platform independent and (ii) autonomously reconfigures itself dependent on actual states of subsystems and events in the current environment.

The base infrastructure in a household provides the backbone regarding the energy management concerned for the particular installation site. System dynamics are mainly represented by inhabitants appearing and disappearing throughout the day and the sensor traces they leave. These events are registered, and system components will get adapted accordingly (e.g., switching off unneeded devices currently in standby). It is common nowadays that a user is equipped with more than one personal device (e.g., smartphone and smart watch) where device interactions have to be defined. The influence of the developed system goes further in the way that part of these dynamics will also get reflected on the mobile devices the user carries around whether at home, at work or in times when any leisure activities are performed.

Therefore, the essential aspects the system is concerned with, are multi platform capability (to enable widespread deployment), different interconnection constellations (given by dynamic system behaviour over time), as well as general adaptation over time by recognizing and mapping behaviour and preferences of users.

A. Dynamic Module Systems and Service Platforms

At the core of the system it was required to enable system execution on different end devices. This ranges from conventional desktop systems to mobile smartphone and tablet devices. A modular component setup was specified for which the OSGi framework reference implementation Felix [4] was used. Modular system specifications have been addressed in various literature where multi-layered, service oriented software has been developed for application in different domains like telematics [5], web technology [6], cloud services [7], health and elderly care [8], vehicular network management [9] or context computing [10], in general.

B. Device and Service Discovery

Device and service discovery is of major interest in all cases where distributed entities have to exchange information on behalf of an unreliable and dynamic connection infrastructure. In the PowerIT system, the dynamic part mainly consists of the interactions of the mobile devices (e.g., smartphone and smart watches of a user) amongst each other, and with the infrastructure. Also, the change of a user between multiple infrastructures (e.g., from home to work) has been considered as an extension.

Depending on the connectivity to the system, the state of users and present and upcoming interactions, the system needs to adapt to these conditions. Although the main infrastructure stations might be known in advance, for mobile entities, this is not the case throughout the whole lifetime of the system. Therefore, device and service discovery has been utilized much in the manner of [11][12] where standard internet protocol (IP) services are applied for this purpose.

C. Opportunistic sensor configuration, activity recognition and decision making

For automatic control, the activities of users get observed and are distributed accordingly in the system to enable the switching of devices or groups of gadgets. For this purpose results found in [13][14][15] can be utilized by establishing recognition chains for every sensor data input stream that is of interest and importance for the system functionality. This issue is addressed in more detail in the architecture technology section. After activities, or a change of activities has been observed from sensor data, this information is forwarded to the system where it gets decided if it will eventually lead to any state change or not. This is achieved by putting the results of Kurz et al. [16] into practical application. Within our first test setups it has been shown that this approach already shows positive impact although it was deployed only for a controlled part of the system to prevent a degradation of overall system usability. The decision module that is responsible to determine if any input data will lead to an actual actuator control command is reused from the work in [17]. A generic set of rules that is capable of integrating arbitrary sensor inputs dynamically to build up statements about the system state was realized and successfully brought to practical application.

III. System Architecture

The intention for the system was to have an unobtrusive setup of personal information and communication technologies (ICT) services that allow the recognition of user centered activities of daily living to control the electrical system in its surroundings. It has soon been detected by the authors that for a system that has to provide its functionality in a 24/7 fashion, special precautions needed to be taken to serve the

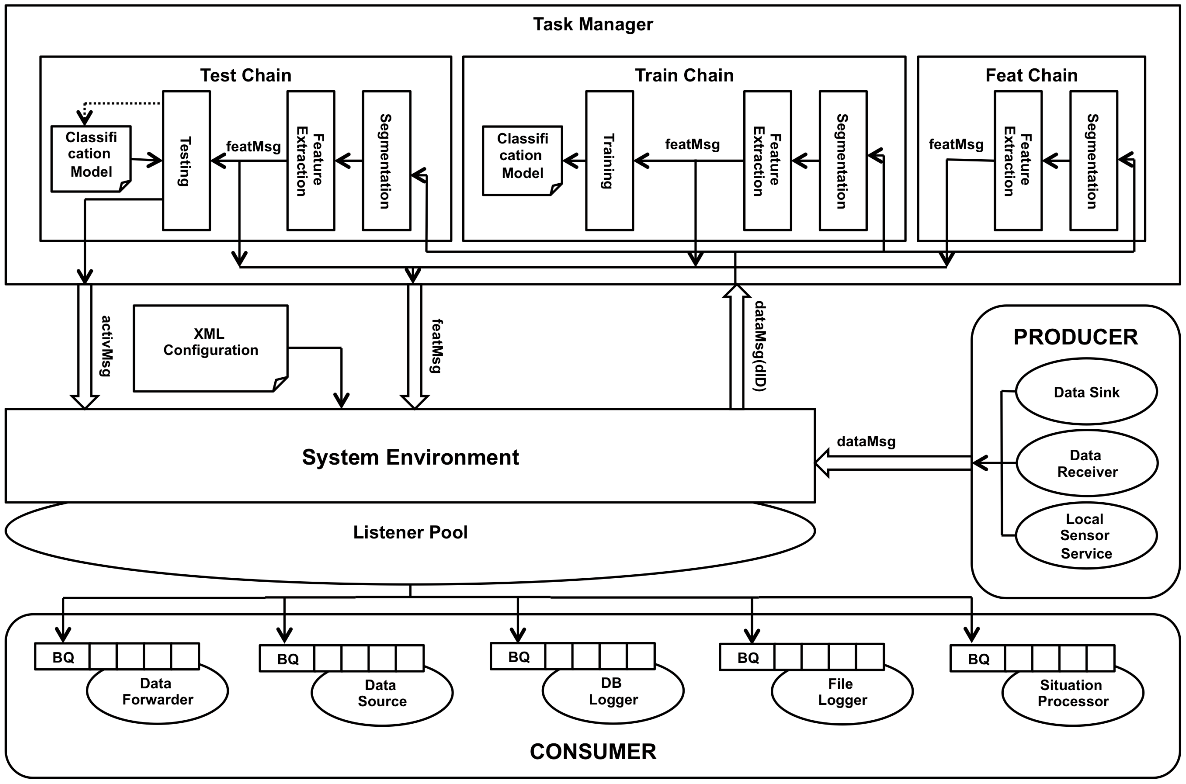


Figure 2. System Architecture Diagram

given requirements. The system needed to be implemented in a way, so that failures on a long term basis (e.g., unforeseen bugs in rarely called functions or accumulated and never freed memory) will be recovered automatically, unrecognized by the user in the optimal case.

From this description it is clear that the system depends on several parameters that can be divided into options that are static throughout the lifetime of the system, and others that change over time dependent on various parameters like the time of day, location and activity of a user, but also the intentions of a user with respect to electric device usage, convenience functions and real-world situation context (e.g., times when system automatisms are unwanted).

The specified system architecture results in a generic framework that can be executed on various devices (like smart watches, smartphones, tablet computers or desktop systems) running different operating systems. It is implemented in the Java programming language, and is, therefore available for all environments capable to execute a Java Runtime Environment (JRE) [18]. By relying on the OSGi middleware framework, a better separation and modularization of system components is reached. This supports the execution of the framework under different role settings that are addressed below in Section III-B.

A. System Overview

```
<config>
(a)     <system>
                <devicename>MotoACTV</devicename>
                <is-log>true</is-log>
                <is-act-rec>true</is-act-rec>
                <local-sensor>true</local-sensor>
        </system>
(b)     <fwds>
                <fwd>
                        <name>GGPH</name>
                        <type>feat</type>
                        <conn>bt</conn>
                        <mac>AC:22:0B:A4:22:93</mac>
                        <uuid>00001101-0000-1000-8000-00805F916001</uuid>
                </fwd>
        </fwds>
(c)     <rcvs>
                <rcv>
                        <name>HS</name>
                        <type>feat</type>
                        <conn>ip</conn>
                        <ip>10.0.0.1</ip>
                        <port>16001</port>
                </rcv>
        </rcvs>
(d)     <snks>
                <snk>
                        <name>GGPH-ctrl</name>
                        <type>ctrl</type>
                        <conn>bt</conn>
                        <uuid>00001101-0000-1000-8000-00805F918001</uuid>
                </snk>
        </snks>
(e)     <srcs>
                <src>
                        <name>GGPH-cnt</name>
                        <type>cnt</type>
                        <conn>ip</conn>
                        <port>22001</port>
                </src>
        </srcs>
(f)     <loggers>
                <logger>
                        <name>wws</name>
                        <type>file</type>
                </logger>
                <logger>
                        <name>feat</name>
                        <type>file</type>
                </logger>
        </loggers>
</config>
```

Figure 3. Main configuration sections

The system architecture is divided into three main categories that are enlisted below. A graphical representation of the system architecture is shown in Figure 2. There, the system components available locally on a host node are depicted.

1) The base functions like configuration handling, message passing and thread pools.
2) Communications including device and service discovery and transmission of different message types.
3) Activity recognition, locally and remotely.

The system core consists of a structure for managing the runtime environment of the framework which also includes a listener pool to which elements interested in any specific messages register. To forward and distribute messages within the system a producer - consumer pattern is specified where producers offer data to the system from local or remote sensor sources and consumers forward data to remote hosts or are responsible for local persistence. Activity recognition is defined in terms of a task chain that contains data segmentation, feature extraction, training and testing of classification models. Multiple task chains can be handled concurrently on a single host.

Depending on the configuration of every host node, an end device can take a different role, which is explained in the following subsection. A simple example configuration is presented in the listing in Figure 3.

Initially, a communication system that allows the propagation of sensor data from low power embedded devices up to full-blown server installations to access the data at different device instances with varying computation performance was designed and developed. The system implements general patterns that allow the execution of the same software on different host nodes with varying configurations.

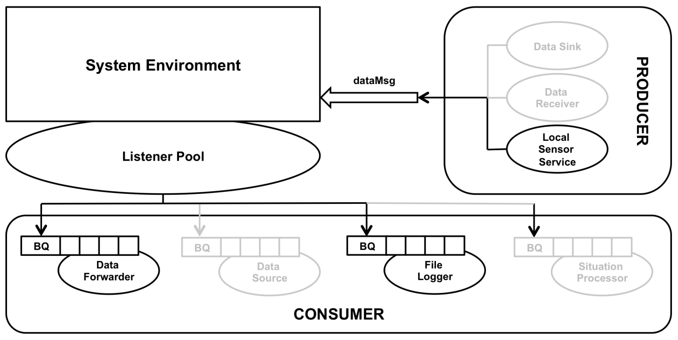
Starting with an extendable configuration, devices have the capability to re-configure themselves dependent on other devices found in the environment and the role the local device has to full fill dependent on the global system state.

The configuration depicted in Figure 3 represents a default configuration that illustrates possible settings. It is used as example to enlist the main configuration items that will get deployed across the nodes included in the system. The first section of the listing, named (a) *system*, defines general system properties which inform if local sensors (if available), file logging and activity recognition are enabled. This already partially defines the role (cf. Section III-B) of the host, on which this configuration item is deployed. Without any connection configurations only local operation would be possible but for full adaptive participation in the system the corresponding communication channels need to be stated. The relevant items for this purpose in the configuration file are found under the sections (b) *fwds*, (c) *revs*, (d) *snks* and (e) *srcs*, respectively. Although it is not shown in the listing, it is possible to setup several connections of every type together to form the different roles required in the system. Additionally, as depicted in Figure 3, it is possible to drive different connection types simultaneously which even extends the number of possibilities how system parts can be connected together.

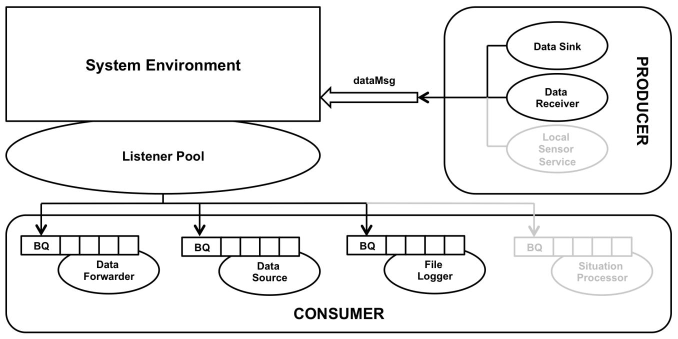
Section (f) *loggers* enables the corresponding logging capabilities where the subsections determine which types of messages are logged. In the above example, raw data (for later offline analysis) as well as feature data derived from raw data are logged. It is possible to log other data types like preprocessed raw data or activities retrieved from the recognition functionality.

By using this definition for system entities, a message passing system in the style of a Model-View-Controller (MVC) [19] design pattern has been specified. To evaluate this design, but also to serve as a workhorse for data acquisition, the implemented system has been deployed and put in operation in three installation sites. From the configuration items presented in the listing, it can be seen that IP, as well as bluetooth based connections are possible (as depicted in configuration sections (b) and (c)).

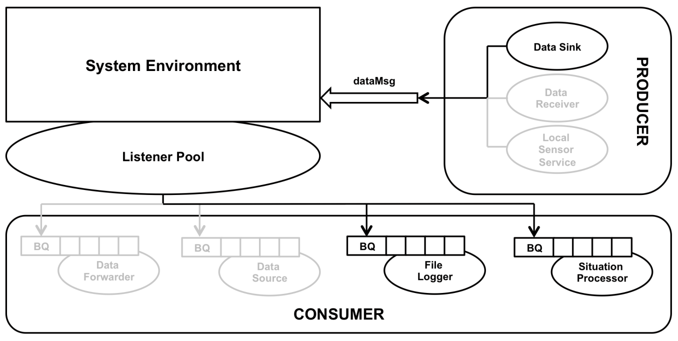
B. Dynamic Device Roles



a) Sensor Endpoint



b) Proxy Node



c) Home Server
Figure 4. Dynamic Device Roles

The system provides multiple sub services on different end devices. With this approach, services can be offered that optimizes system utilization regarding the resources of the runtime device. This way it is possible to keep the data source on the smart watch device while interaction and control can be handled over to a remote device like a smartphone. An accompanying advantage of this design is that the same software packages can be reused on different end devices to provide different services corresponding to the used configuration.

Device roles are tightly dependent on the functionality that an entity in the system will perform and the type of connection required for data exchange. Communication is set up in a server - client fashion and dual roles for servers as well as clients exist to form proxy devices. One type is the *sink* server to which a corresponding *forward* client pushes data. The other server type is called *source* that that provides data to be fetched by *receiver* clients. By this approach, any device in the system can flexibly be configured to play any role, which is autonomously reconfigured during runtime if the purposes of a device changes (i.e., switching from an end device configuration to a proxy configuration as another end device requests this feature temporarily).

The general idea is that temporal unreachability or failure of one device will get detected by the system and another device in the system formation will be adapted to take over the role of the failed device. This was one reason for the unified software implementation as it can be guaranteed that the replacement device is able to handle the same functionality. The roles that are relevant in our setting are:

1) Sensor endpoint
2) Proxy
3) Local (installation site) server, Home Server
4) Global server

Corresponding schematics are depicted in Figure 4. In our system, sensor endpoints (Figure 4a)) are the energy meter devices and smart watches. Smartphones are utilized as proxy devices (Figure 4b)) and the central access point is represented by the Home Server (Figure 4c)). The functions that are necessary on the corresponding type of device are depicted in bold in the respective subfigure, the illustration concerned with activity recognition is omitted there.

C. Activity Recognition

The dynamic nature of system connections turned on the requirement that the system component performing activity recognition and tracking has to be capable of these dynamics as it was necessary to rapidly switch devices that are executing recognition chains. For example, when a smartphone performs activity recognition for the local sensor, as well as for data from a remote smart watch sensor and the user leaves the smartphone back while keeping the watch on, then, globally this task has to be split to both devices performing its own activity recognition each. If afterwards the user returns to the smartphone this needs to be detected by the system and recognition chains can be processed on the smartphone again.

Also, for practical reasons, the resources on a smart watch are much more limited than on a smartphone or on the Home Server especially regarding battery lifetime. As the activity recognition is a computationally expensive task, it might be convenient to outsource parts of this task from the device where the sensor data acquisition is performed. These cases are depicted in the upper third of Figure 2.

Activity recognition is defined in terms of task chains, where a single task chain is set up for every type of sensor and device. A task chain performs raw data segmentation, feature extraction, the classification of featured training data, and the evaluation of new data according to a set up classification model. Besides classification of raw sensor data, corresponding statistics are collected for every sensor of how long and how much a device is used for which purpose. This information is used to continuously update the corresponding background context to offer system services according to their utilization within the actual application scenario.

D. Registration, Un-Registration and Re-Registration

In a smartphone/smart watch or smartphone/smart watch/HS constellation, communication is always preceded with a discovery stage dependent on the actual states of devices. If, for example, active communication is ongoing, no discovery is necessary and can therefore be disabled at this time. If connections break suddenly, or the activity state of a user indicates a change within the network topology (e.g., because of change in location), a re-scan of the environment will become necessary. Therefore, occasion based device and service discovery is implemented to execute the process only when the system is in a certain state and disabled in all other cases. The main intention for this approach is to save battery lifetime of mobile devices.

When device and service discovery is processed, potential remote nodes get detected and their supported services registered. On initial system startup a device and service discovery is performed where found nodes and their capabilities are put into a local cache. At subsequent connection attempts, this cache is first iterated over to reestablish well-known connections. Only if this is not possible new device and service scans are started. This process is depicted in the diagram in Figure 5. For this purpose, JMDns [20] is utilized as it has proven to be a simple, efficient and reliable solution that is able to provide this functionality at a sufficient degree.

After the temporal topology is determined the self descriptions of newly found nodes are exchanged. The configuration exchange function is a service every device supports to be able to participate in system communication. The configuration items can also be cached to avoid the explicit exchange for this item. After remote configurations are known, any required reconfiguration is handled.

To determine which device is responsible for which task, preferences are defined. If two devices share the same preference for a certain function, a default preference is given by the device roles in the order of their enlisting. Depending on the specific task the preference for a role is higher as for another. Practically, it is better to perform calculations locally and only transmit results than forwarding raw data in a stream like fashion which has severe implications on the battery life times in case of mobile devices.

IV. Conclusion

In this work, it has been shown that the existing problem of integrating heterogeneous technical entities in digital environments can be replied by (i) the establishement of conventions regarding the definition of concerned items, (ii) specification of communications, and (iii) information representation within its actual context. A generic architecture approach has been presented that allows the execution on a variety of platforms and maps platform features (i.e., present sensors, actuators or processing units) onto corresponding input and output channels. This way, device interaction can be achieved to perform information exchange for a variety of purposes in an adaptive manner.

The necessity of scaling up the number of items and involved people will appear when the system is going to be installed in broader context environments like workplace (e.g., office or factory) or leisure sites. These cases are considered in the framework implementation and need deeper research under practical real-life conditions.

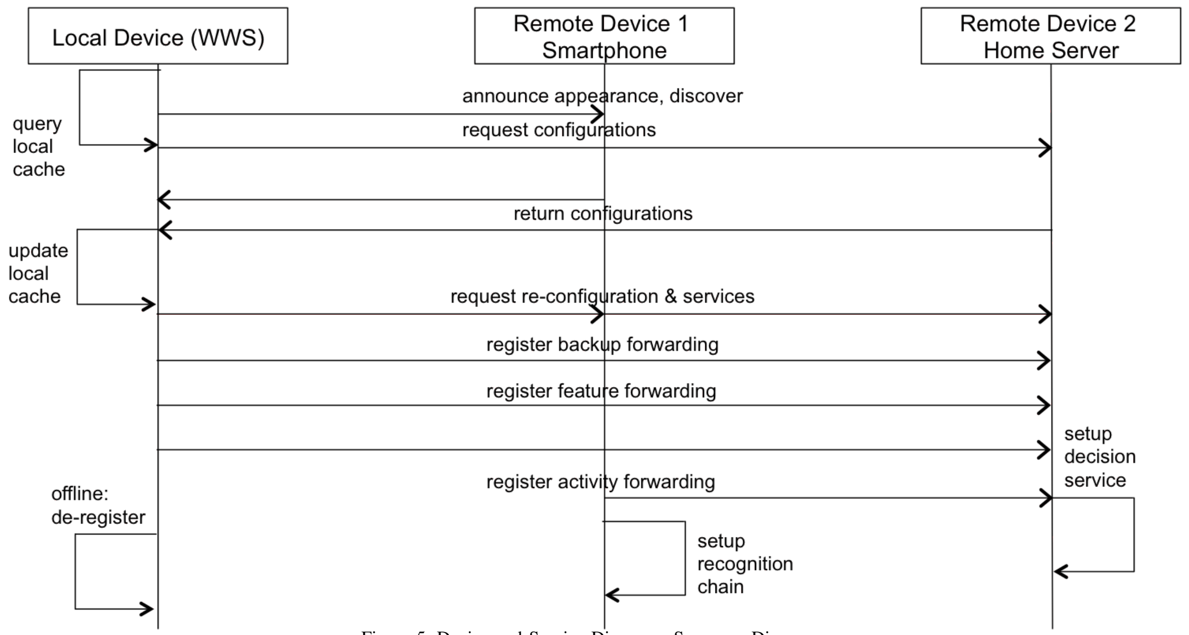

Local Device (WWS)
Remote Device 1 Smartphone
Remote Device 2 Home Server
announce appearance, discover
query local cache
request configurations
return configurations
update local cache
request re-configuration & services
register backup forwarding
register feature forwarding
setup decision service
register activity forwarding
offline: de-register
setup recognition chain

Figure 5. Device and Service Discovery Sequence Diagram

The requirements identified in the introduction section were shown to be solved by the implemented methods. The developed approach is empirically tested to gain real-world evidence within the PowerIT project at the moment. Utilizing the test installations, it turned out that the implemented services and modules are performing as expected. The real world installations are an ongoing and evolving process yielding to new findings for future enhancements of the developed framework. Consequently, it was shown that the application of our dynamic system for context and situation representation heads in the right direction.

Acknowledgment

The project PowerIT acknowledges the financial support of the FFG FIT-IT under grant number: 830.605.

References

[1] G. Hoelzl, P. Halbmayer, H. Rogner, C. Xue, and A. Ferscha, “On the utilization of smart gadgets for energy aware sensitive behavior,” in The 8th International Conference on Digital Society, ICDS 2014, March 23 - 27, Barcelona, Spain, March 2014, pp. 192–198.

[2] G. Hoelzl et al., “Locomotion@location: When the rubber hits the road,” in The 9th International Conference on Autonomic Computing (ICAC2012), San Jose, California, USA, September 2012, pp. 73–78.

[3] A. Ferscha, J. Erhart, P. Halbmayer, M. Matscheko, and M. Wirthig, “Powersaver - activity-based implicit energy management,” in 15th International Symposium on Wearable Computers (ISWC2011), June 2011.

[4] “Apache Felix,” http://felix.apache.org/, [retrieved: 05, 2014].

[5] Y.-L. Chu et al., “An integrated java platform for telematic services,” in Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on, 2010, pp. 590–593.

[6] D. Carlson, B. Altakrouri, and A. Schrader, “Ambientweb: Bridging the web’s cyber-physical gap,” in Internet of Things (IOT), 2012 3rd International Conference on the, 2012, pp. 1–8.

[7] F. Houacine, S. Bouzefrane, L. Li, and D. Huang, “Mcc-osgi: An osgi-based mobile cloud service model,” in Autonomous Decentralized Systems (ISADS), 2013 IEEE Eleventh International Symposium on, 2013, pp. 1–8.

[8] K. C. Kang, S. U. Heo, and C. S. Bae, “Android/osgi-based mobile healthcare platform,” in Advanced Information Management and Service (ICIPM), 2011 7th International Conference on, 2011, pp. 125–126.

[9] T.-W. Chang, “Android/osgi-based vehicular network management system,” in Advanced Communication Technology (ICACT), 2010 The 12th International Conference on, vol. 2, 2010, pp. 1644–1649.

[10] D. Carlson and A. Schrader, “Dynamix: An open plug-and-play context framework for android,” in Internet of Things (IOT), 2012 3rd International Conference on the, 2012, pp. 151–158.

[11] R. Klauck and M. Kirsche, “Bonjour contiki: A case study of a dns-based discovery service for the internet of things,” in Ad-hoc, Mobile, and Wireless Networks, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7363, pp. 316–329.

[12] R. N. Lass, J. Macker, D. Millar, and I. J. Taylor, “Gump: Adapting client/server messaging protocols into peer-to-peer serverless environments,” in Proceedings of the 2Nd Workshop on Bio-inspired Algorithms for Distributed Systems, ser. BADS ’10. New York, NY, USA: ACM, 2010, pp. 39–46.

[13] D. Roggen, K. Förster, A. Calatroni, and G. Tröster, “The adarc pattern analysis architecture for adaptive human activity recognition systems,” Journal of Ambient Intelligence and Humanized Computing, 2011, pp. 1–18.

[14] G. Hoelzl, M. Kurz, and A. Ferscha, “Goal processing and semantic matchmaking in opportunistic activity and context recognition systems,” in The 9th International Conference on Autonomic and Autonomous Systems (ICAS2013), March 2013, pp. 33–39.

[15] ——, “Goal oriented recognition of composed activities for reliable and adaptable intelligence systems,” Journal of Ambient Intelligence and Humanized Computing (JAIHC), July 2013, p. in Press.

[16] M. Kurz, G. Hoelzl, and A. Ferscha, “On the utilization of heterogeneous sensors and system adaptability for opportunistic activity and context recognition,” in Fifth International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE 2013), May 27 - June 1, 2013, Valencia, Spain, May 2013, pp. 1–7.

[17] “JRuleEngine - OpenSource Java Rule Engine,” http://jruleengine.sourceforge.net/, [retrieved: 05, 2014].

[18] “Java Standard Edition,” http://java.oracle.com/, [retrieved: 05, 2014].

[19] G. E. Krasner and S. T. Pope, “A cookbook for using the model-view controller user interface paradigm in smalltalk-80,” J. Object Oriented Program., vol. 1, no. 3, Aug. 1988, pp. 26–49.

[20] “Java Multicast DNS,” http://jmdns.sourceforge.net/, [retrieved: 05, 2014].