Self-Service Ambient Intelligence Using Web of Things Technologies

Nicole Merkle, Benedikt Kämpgen, Stefan Zander

FZI Forschungszentrum Informatik am KIT, Information Process Engineering, Haid-und-Neu-Str. 10-14, D-76131 Karlsruhe, Germany, merkle@fzi.de, kaempgen.fzi.de, zander@fzi.de

Abstract. Context-aware applications play a central role in Ambient Assisted Living (AAL) environments. A study about ethical, legal and social implications of AAL environments revealed that AAL applications need to fulfill two central requirements: (a) they must by tailored towards the specific needs of impaired persons, and (b) they need to be easily configurable by different user groups such as caregivers and relatives since these groups usually do not have profound technical background knowledge. This work discusses how the two main results of a study about ethical, legal and social implications of AAL environments, namely (a) the need-centric orientation of AAL applications and (b) their configurability by non-technical persons, can be addressed by a technical framework that uses semantic Web techologies as interoperabiltiy infrastructure for the integration of IoT devices. The architecture addresses typical aspects of AAL. We show how aspects related to the extensibility of AAL environments and error-checking methods can be applied by the presented approach. More specifically, we present a semantic model to describe user intentions about IoT devices as well as a method to reason about user intentions and to semi-automatically evoke appropriate actions. Our approach is based on the idea of the Web of Things (WoT) and evaluated using a scenario about helping an impaired person with controlling devices.

1 Introduction and Motivation

In 2013, 9.3% of the German population had a severe disability [1]. These people need continuous help by caregivers and relatives to cope with their daily routines. Ambient Assisted Living (AAL) promises to enable impaired people to spend a more self-determined life through the help of context-aware applications. Such applications are capable to interpret the context of the assisted person, i.e. the surrounding things, and take appropriate actions in near-real-time. The pervasiveness of such applications poses a number of severs challenges related to their requirements and development. An analysis of the ethical, legal and social implications (ELSI) of the problem space revealed that AAL applications should be very specific to the assisted person, i.e. either learning its most appropriate utilization automatically or being possible to be configured by the assisted person, the caregivers and relatives. However, there are several challenges. Assisted

persons, caregivers and relatives generally do not have the technical background knowledge to use program code or scripts to configure an application and there is a constant increase of IoT devices in- and outside homes, based on heterogeneous technologies. Moreover, applications that are built using custom program code or scripts have no limits in what they are able to do programmatically. However, they are difficult to extend, refine or check for errors. In this paper, we present an archtectural approach a) to enable an easy programming of context-aware applications and IoT devices and b) to reason about user intentions by IoT device events and to semi-automatically evoke appropriate actions. In Section 2 of this paper, we present a representative scenario and show in Section 3 our approach to address the mentioned challenges and problems. Section 4 runs through the technical setup and execution of the Light Controlling service by means of the Sherlock engine. Section 5 discusses relevant works and Section 6 ends with a conclusion and an outlook about future research work.

2 A Typical AAL Scenario

We represent a fictive but typical scenario in order to discuss the occurring problems and challenges. Erna Huber, a 62 year old retired teacher, is bound to her wheelchair, after a stroke several years ago. She has also an eyesight impairment. Mrs. Huber wants to improve her eyesight by a *Light Controlling* application for increasing and decreasing the brightness in her surroundings. The pre-condition to accomplish this, is to install different kind of sensing and acting devices in her living environment. Mrs. Huber needs an ambient light sensor for measuring the ambient brightness. Furthermore, actors are necessary to control the lamps in her home. The application additionally requires a motion sensor in order to know the current location of Mrs. Huber. In a first step the appropriate devices require to be installed in the living environment. Hence, Mrs. Huber needs somebody who installs the devices and the *Light Controlling* service. She asks her son Peter to help her. However, an AAL environment and the configuration of devices and services is very complex and makes expert knowledge necessary. Thus, Peter has no idea in which way he can setup the devices and the new *Light Controlling Service* of Mrs. Huber. It would be nice, thinks Peter, if he could easily setup and configure the devices and the *Light Controlling Service* by a Web user interface, to enable his mother to use immediately the new installed *Light Controlling Service* and its related devices. Peter and his mother want to save money, so after a long and tiresome installation and a lot of reading of documentation, the assistive environment is almost set up but something is still not yet correctly working, so that Peter at the end requires to call a service provider who fixes the issue and demands a heap of money, before Mrs. Huber finally can use the Light Controlling service.

3 Approach

Our approach enables users to program and integrate IoT devices as well as assistive applications more easily. This is accomplished by adopting the Web of Things (WoT) approach[1] and the modular architecture, published in [2]. In the following we briefly describe all relevant components:

Sherlock engine: The Sherlock engine is a rule-based engine, which aims at deducing the user intention. Hence, the Sherlock engine uses an ontology[2] constituted for describing the environment, users, AAL services and IoT devices. Sherlock requests the RDF(S) instances by a HTTP request and a SPARQL Construct query from the SPARQL endpoint of the triple store. In this way, the *Sherlock Programs* with their contained SWRL rules and linked actions are provided to the Sherlock engine. As an event occurs, the Sherlock engine adds the new device events and states as triples to the ontology and reasons by means of the given SWRL rules and the OWL-API[3] as well as the SWRL-API[4] the user intention and the according actions. If the reasoning provides a result, the Sherlock engine sends the appropriate actions as triples to the WoT server, which controls the devices. After the appropriate actions are executed, the Sherlock engine discards the current ontology and requests and modifies it again, if a new event occurs.

Ontology: The ontology describes the AAL environment and the relevant objects (*Things of Interest*, e.g. IoT devices, Sherlock programs, users, intentions, actions and events). The main value is to map device functionalities to program functionalities and user intentions. A user intention is constituted by rules and linked actions while a service description is related to different user intentions. In this way we can semantically express that every program is interested in and responsible for appropriate user intentions.

Query Generator: The Query Generator component generates SPARQL queries for the given Triple Store. So the Query Generator provides for the Sherlock engine an opportunity to request context information from the Triple Store.

Triple Store: The Triple Store gets its semantical enriched data by a Semantic MediaWiki (SMW). These data is then accessible for the Sherlock engine and other applications.

SMW: The SMW provides for the user (assisted person, caregiver, relative) forms and templates which we implemented to create semantically enriched descriptions about the *Things of Interest* in the environment. These *Things of Interest* can be devices, assisting applications, user profiles as well as abstract ideas such as events and programs. The user inscribes the appropriate

[1] This work considers WoT as an extension of the IoT. The extension comprises the use of standard Web protocols and the semantical description of IoT *Things*.

[2] See the ontology at https://koralle27.fzi.de/aicasys/ontology

[3] http://owlapi.sourceforge.net/

[4] https://github.com/protegeproject/swrlapi

Thing properties into the provided Wiki forms, while the Wiki template annotates the entries in the background to generate a RDF(S) representation of the Thing. The SWRL rules are also annotated as a string property, which can be requested by the Sherlock engine. If the SWRL rules needs to be changed, the user can edit the appropriate rule text field and adjust the rule to his/her needs. The SMW provides for different semantic Things forms and templates. As soon as a Wiki page is created by a form, the RDF(S)[5] representation is exported into a *Sesame* triple store and is available for the Sherlock engine.

Configurator: The Configurator component sends every minute requests to the SMW to check for new device descriptions and to generate by the device representations, IoT system-consistent configuration files.

WoT server: The WoT server is an application layer for the given IoT devices. It provides a semantic representation of the devices for AAL applications. Every device is described by a) events which it triggers, b) actions which are invoked on it, c) properties, which describe it. As soon as device events occur, the WoT server transforms these events into semantic device and event instances. Therefore, it uses the T-Box of the introduced ontology. The Sherlock engine receives these events by WoT server pushed JSON-LD messages, presented in Section 4. Hence, the Sherlock engine is non-stop connected to the WoT server.

IoT Adapter: Every IoT system is tethered by an *IoT Adapter* component for implementing the appropriate IoT system protocol. The appropriate IoT system controls the environmental IoT devices and for- and backwards events to the WoT server. The Sherlock engine as well as the ETG communicates via the *JSON-LD Adapter*.

Eye Tracking Glass (ETG): The ETG is measuring the gaze patterns of the user to recognize which Things in the environment are focussed by the user. Hence, we need a semantic description of all relevant things in the environment.

The message exchange between all components is handled by Websockets, enabling a bi-directional communication. The advantage of Websockets is that no polling—as with REST—is required. Messages can be pushed by means of a full-duplex communication. Furthermore, an event-based communication is no longer a problem. In [2] is presented how user intentions can be linked to appropriate actions. Every user intention consists of rules—expressed as triples—and actions and is linked together in service notations. We extend this approach by SWRL rules. The rule to turn the light on, is expressed in the SWRL syntax in Figure 1. After the reasoning by the SWRL-API is done, the Sherlock engine asks for the intention of the assisted person and the related actions in the SPARQL query in Figure 3. For applying the SWRL rules, we have to assume that all pre-conditions are valid for all instances and that it exists at least one action and one intention which is met by the pre-conditions. Our Sherlock engine uses the programs—created in the SMW—to accomplish appropriate services for the

[5] Resource Description Framework (Schema)

$$\mathsf{AssistedPerson}(?u) \sqcap \mathsf{Lamp}(?l) \sqcap \mathsf{hasState}(?l, \text{“off”}) \sqcap \mathsf{AmbientLight}(?a)$$
$$\sqcap \mathsf{hasState}(?a, ?s) \sqcap \mathsf{swrlb} : \mathsf{lowerThan}(?s, 600) \sqcap \mathsf{wears}(?u, ?e)$$
$$\sqcap \mathsf{EyeTrackingGlass}(?e) \sqcap \mathsf{hasInFocus}(?e, ?l) \sqcap \mathsf{Intention}(\mathsf{SwitchLightOnIntention})$$
$$\sqcap \mathsf{hasAction}(\mathsf{SwitchLightOnIntention}, \mathsf{SwitchOnAction})$$
$$\sqcap \mathsf{hasAction}(?l, \mathsf{SwitchOnAction}) \rightarrow \mathsf{hasIntention}(?u, \mathsf{SwitchLightOnIntention})$$

Fig. 1. SWRL rule to turn lights on.

```
1 select ?intention ?action
2 where {:assistedPerson :hasIntention ?intention.
3         ?intention :hasAction ?action.}
```

Fig. 2. SPARQL query after SWRL reasoning.

user. The advantage of our approach is to allow in this way extendibility of the rule-based Sherlock engine because every program provides new knowledge and functionality, which is programmed by the user and the created SWRL rules.

4 Evaluation Scenario

For evaluation purposes, we have build a prototype consisting of a WoT server, a SMW, a triple store and the adaptive Sherlock engine. The aim of this evaluation is to act out the introduced *Light Controlling* use case and to validate that our approach is feasible, usable and processable in real-time. Furthermore, we show, that the Sherlock engine is programmable on-the-fly by means of the SMW. The following devices are used for the evaluation: An IoT motion detection device, an IoT lamp switch and an IoT ambient light sensor from EnOcean[6]. These are stationary devices for sensing environmental data. Moreover, we use two Android mobile phones with Android version 5.0. On the one mobile phone, an *Eye Tracking Simulator* (ETG Simulator) application is running. On the other phone the Sherlock engine is installed. A Raspberry Pi is in use as hardware platform. On the Pi, we have installed an open-source IoT system called openHAB[7]. openHAB is programmed in Java and based on OSGi[8]. The openHAB server needs to know -before running- the appropriate devices for controlling them. Thus, we have created by means of our installed SMW, a configuration file. The SMW is installed on the Raspberry Pi. Using SMW, we described the existing devices (Lamp1, AmbientLightSensor1, ETG-GG) and the environment with its rooms. Concretely, we linked the AmbientLight1 and Lamp1 to the Living Room. Our

[6] https://www.enocean.com/en/
[7] http://www.openhab.org/
[8] For more information see here: https://www.osgi.org/

created device descriptions about Lamp1 and AmbientLightSensor1 as well as the ETG-GG also contain their device functionality. In this way, we described the *Light Service* and its context. Moreover, we have installed a WoT server on the Raspberry Pi, which is accessible via a Web Socket. In our evaluation, we implemented an openHAB and a JSON-LD Adapter. Our ETG simulator is communicating via the JSON-LD Adapter. As we start the Sherlock engine, it sends a JSON-LD message to request all devices connected to the WoT server. The WoT server returns the URIs and additional properties of the devices, describing the device characteristics. A request for all devices looks as depicted in Figure 3. The method property specifies the request type while the paramType property

```
{
  "@context": "https://koralle27.fzi.de/aicasys/ontology#RemoteMessage",
  "method": "getDevice",
  "paramType": "ALL",
}
```

Fig. 3. A JSON-LD device request.

names the device type. The Sherlock engine gets to its request a response as depicted in Figure 4. The response returns the URI of the devices and its properties with their SMW URI. Figure 4 shows just an excerpt of the message with one device (ETG and its properties). As our *ETG Simulator* detects in our test

```
{
 "sensor": {
   "@context": "https://koralle27.fzi.de/aicasys/onotology#EyeTrackingGlass",
   "@id": "wiki:ETG-GG",
   "attributes": {
     "HasInFocus": "wiki:HasInFocus",
     "HasContext": "wiki:HasContext",
     "HasName": "wiki:HasName",
     "WoT-URI": "wiki:WoT-URI",
     "HasFocusTime": "wiki:HasFocusTime",
     "HasId": "wiki:HasId"
   },
   "type": "rdf:EyeTrackingGlass"
 }
}
```

Fig. 4. A JSON-LD device response with the URI of the device and its properties.

the installed Lamp1, a sensor event is sent and specified by a triple with subject, predicate and object. The id property stands for the subject and references the device which made the observation. The predicate property references the observation type while the object property references the sensed value. Figure 5

depicts an observation of the ETG. In our test setting the triple expresses that the ETG-GG instance has detected a Lamp1 device. As meta information every sensor event consists of a timestamp to evaluate the actuality of the event. Be-

```
{
 "@context": "https://koralle27.fzi.de/aicasys/ontology#RemoteMessage",
 "method": "publishSensorEvent",
 "timestamp": "24.11.2015 13:10:30.128",
 "triples": [
   {
     "@context": "https://koralle27.fzi.de/aicasys/onotology#EyeTrackingGlass",
     "@id": "wiki:ETG-GG",
     "label": "Die Brille hat Lamp1 im Fokus",
     "object": "wiki:Lamp1",
     "predicate": "wiki:HasInFocus"
   }
   ]
 }
```

Fig. 5. A published sensor event. The ETG has detected Lamp1 see also [2].

fore our Sherlock application can receive sensor events, it needs to subscribe at the WoT server. This is done as depicted in Figure 6. A registration message is sent to the WoT server with the appropriate *Light service* program containing the necessary intentions with their rules. We focussed by means of the *ETG Simulator* application the QR Tag of Lamp1 and triggered in this way a sensor event. The WoT server received this event and sent it immediately within 500 milliseconds to the Sherlock application. Afterwards, our Sherlock application requested additional context data as in Figure 7 to evaluate the intention rules. The appropriate SPARQL query is annotated in the SMW, so that the Sherlock engine just need to request it there. The rules matched and Sherlock inferred correctly and in real-time that the user wanted to switch Lamp1 on and so Sherlock sent an appropriate action in order to control Lamp1. Figure 8 shows the matching message which contains the action for switching on Lamp1. With the presented setting, we evaluated the reliability of the user intention recognition by means of our Sherlock engine and the Light service. Moreover, we presented that Sherlock was enabled by the user, to control lamps, user intention driven. In our evaluation, the testpersons had no problem to create by means of the SMW forms new device and program descriptions, so that Sherlock was able to use the new generated knowledge. Our evaluation illustrates that our approach enables users to configure and adapt an AAL environment according to their needs and purposes. During the AICASys project, we plan to test in lab- and field trials the extendibility and usability of the AICA system. Furthermore, we want to review in field trials the ELSI compliance of our proposed system.

```
{
  "program": {
    "@context": "https://koralle27.fzi.de/aicasys/ontology#SherlockProgram",
    "@id": "wiki:Light_Service",
    "serviceName": "Light Service",
    "type": "SUBSCRIBER",
    "userIntentions": {
      "wiki:Switch_Light_On_Intention": {
        "@context": "https://koralle27.fzi.de/aicasys/ontology#Intention",
        "@id": "wiki:Switch_Light_On_Intention",
        "actions": {
          "wiki:Switch_Light_on": {
            "@id": "wiki:Switch_Light_on",
            "@context": "https://koralle27.fzi.de/aicasys/ontology#Action",
            "statement": {
              "predicate": "switchLight",
              "object": "ON",
              "label": "Lamp on",
              "literal": true
            },
            "name": "Lamp shall be switched on.",
            "label": "Lamp on"
          }
        },
        "rules": ""
      }, ...
      }
}
```

Fig. 6. A Sherlock program for registration at the WoT see also [2].

```
{
"method": "queryContextData",
 "paramName": "?room ?sensor",
 "triples": [
   {
     "@id": "http://localhost/aicasys/index.php/Spezial:URI-Aufl%C3%B6ser/Lamp1",
     "object": "?room",
     "predicate": "wiki:Attribut-3AIsInRoom"
   },
   {
     "@id": "?room",
     "object": "http://localhost/aicasys/index.php/Spezial:URI-Aufl%C3%B6ser/Lamp1",
     "predicate": "wiki:Attribut-3AHasInventory"
   },
   {
     "@id": "?room",
     "object": "?sensor",
     "predicate": "wiki:Attribut-3AHasSensor"
   },
   {
     "@id": "?sensor",
     "object": "\"AmbientLightSensor\"",
     "predicate": "wiki:Attribut-3AIsOfType"
   }
 ]
}
```

Fig. 7. Request about the context of Lamp1 see also [2].

```
{
timestamp: 27.02.2016 14:00:56.200,
method:triggerActions,
commands:[
        @context:https://koralle27.fzi.de/aicasys/ontology#Action,
        @id: wiki:Switch_Light_on,
        Action: {
                name: "Lamp shall be turned on",
                label:Lamp on,
                statement: {
                        label: "Switch Lamp on",
                        subject: "wiki:Lamp1",
                        predicate: "switchLight",
                        object : "ON",
                        literal: "true",
                }
        }
        ]
}
```

Fig. 8. A JSON-LD action for switching Lamp1 on.

5 Related Work

This section discusses comparable approaches and distinguishes the presented approach from others. The work in [3] presents an IoT ontology for the integration of heterogeneous IoT devices and applications. The aim at the proposed ontology in [3] is to hide the complexity and heterogeneity of the IoT environment from the enduser and to automate the integration of IoT entities. This objective is also considered in our approach. However, the approach of [3] models only IoT devices but no user intentions and activities. They provide indeed a service description but this service description is just related to IoT devices but not to the user and his/her needs. The approach of Kotis et al. is more focussed on application consumers and providers. However, in our approach the user is independent of the service and application providers, because they are not intended for the integration process of IoT devices and applications. Moreover, our approach enables the user to model and integrate *Things of Interest* without the support of external service providers. Furthermore, we do not envisage to use different applications for assisting the user. We just provide one engine (Sherlock) which can provide assistance by means of various program- and WoT device descriptions. So the user does not need to buy different applications to extend the functionality of the assistive environment. The work in [6] presents in a different domain (surgery) a distributed Web-API and Linked Data based approach for data consolidation and integration. The services in [6] for analysing medical records to derive treatment decisions, are described through Web-APIs. The problem which is not yet solved in [6] and our approach is to handle fuzzy situations. So we want to extend our approach with pattern recognition and machine learning techniques, for covering also fuzzy situations. Furthermore, in [6] is mentioned that expert knowledge is necessary to create the Web service descriptions by means of the presented Surgipedia tool. In contrast, we simplify

the creation of new Things of Interest instances for non-experts. Hence, our approach is extended and more flexible by means of the WoT, which enables the user to describe every relevant object of a use case regardless the application domain.

6 Conclusion and Open Work

In this paper, we proposed a semantic model based on [2] with service notations for linking user intentions to subsequent actions. We extended this semantic model by SWRL rules. Furthermore, we demonstrated an example application (Sherlock Light Control) which is extendible via additional knowledge and functionality by the aforementioned Sherlock programs. The integration of heterogeneous and multi-modal IoT devices is managed by the WoT. We have seen in the Related Work section, that currently there is no attempt to provide user programmable AAL applications and IoT devices, but that expert knowledge is necessary to allow the setup of AAL systems. Additional, we extended the WoT with IoT Adapters for allowing different IoT systems to be integrated into an AAL environment. The advantage is that the user is not bound to specific manufacturers. In future steps, we plan to implement the SWRL rules also for other use cases and to model user profiles, specifying the user characteristics as impairments and diseases so that a better adaption of applications is provided. Further, we want to apply machine learning and pattern recognition to enable the Sherlock application to learn from context data, the resulting actions and to handle also fuzzy situations.

References

1. Statistisches Bundesamt: Pflegestatistik – Pflege im Rahmen der Pflegeversicherung – Deutschlandergebnisse – 2013, Wiesbaden, 2015.
2. N. Merkle, S. Zander. (n.d.). Representing and Reasoning over User Intentions and Actions in Adaptive Ambient Assisted Living Environments, (i), 1124.
3. K. Kotis, A. Katasonov. An ontology for the automated deployment of applications in heterogeneous IoT. Swj, 2012.
4. D. Guinard: A Web of Things Application Architecture – Integrating the Real World into the Web. Switzerland, August 2011.
5. D. Raggett: The Web of Things: Challenges and Opportunities. Computer, vol.48, no. 5, pp. 26-32, May 2015.
6. P. Gemmeke et. al.: Using Linked Data and Web APIs for Automating the Preprocessing of Medical Images. ISWC Workshop, CEUR.