Bridging the Gap between Smart Home and Agents

Jochen Britz, Jochen Frey, Jan Alexandersson
Department of Intelligent User Interfaces (IUI)
German Research Center for Artificial Intelligence GmbH
Campus D3 2, 66123 Saarbrücken, Germany
Email: jochen.britz@dfki.de

Abstract—Intelligent smart home services are usually agent-based distributed approaches, whereas the smart home middleware usually uses centralized plug-in based approaches. Until recently there is no unified communication or cooperation between such services and middleware platforms. Nevertheless, both research branches could gain from each other. To bridge the gap between such agents and middleware platforms we propose a framework that provides a standardized communication based on URC and ACL. It enables agents to easily access functions of target appliances and to create accessible user interfaces.

Index Terms—SmartHome; Agents; UCH; URC

I. INTRODUCTION

In times of the Internet of Things where more and more everyday objects become intelligent, smart homes exist no longer only theoretically, they are available for everyone. In a smart home environment various components have to work together. These components could be grouped in two categories:

1) The controlled components — in the following called targets — are typically sensors, actuators and so forth. Targets can consist of heterogeneous software and hardware components, communication protocols and target appliances.

2) The controlling and/or monitoring components — in the following called controllers — can be all kinds of user interfaces, applications or intelligent services.

However, the research and development in this area seems to drift in two branches: the middleware platforms and the intelligent services. On the one hand, the middleware platforms are concerned with integration and usability of smart home appliances. One of their main tasks is to adapt these targets and provide unified interfaces to users and/or applications. The middleware is usually a plug-in based centralized solution. On the other hand the intelligent services aim to identify behaviors, plans or needs and predict actions of residents of a smart home. Thus they make a comprehensive data analysis of the environmental state. Based on the analysis, they should trigger some actions to support the residents. Such intelligent smart home services are usually agent-based approaches based on a multi-agent system (MAS).

Both of these research branches could obviously gain from each other but until recently there was no noticeable cooperation or unified communication between them. This paper proposes to bridge this gap by introducing a framework which provides a standardized communication between middleware and agent-based services. It also introduces a mechanism to provide accessible interfaces for agents. As shown in Figure 1 the resulting smart home architecture consists of agent based intelligent services for data analysis/reasoning and middleware platforms for target monitoring/control and user interface provision

II. RELATED WORK

A. Middleware Platforms

There are several middleware platforms. Most of them are either focused on home automation or Ambient Assistant Living (AAL). The main objective of the AAL platforms is to provide consistent, adaptive and/or adaptable user interfaces for users with special needs. An example of such AAL platforms are: UCH [4], AppsGate [8], [18], UniversAAL [22], and AALuis [15], [16].

In simple terms, this is done by an abstract description of targets based on some consistent/standardized scheme. With this abstract interface as a basis, the user interfaces...
can either be automatically generated or specially developed for individual users. Most of them provide abilities to add intelligent behaviors to the platform, but these are more or less neglected and not focus of the research. The home automation platforms’ main focus is on providing an extensive list of compatible hardware and creating a control interface for them. Their automation mechanism is usually rule based. An example of such home automation platforms are: openHAB [14], OpenRemote [17] and LinuxMCE [24].

The advantage of middleware platforms in general is the high compatibility to available targets. For each of them there are a couple of plugins available, that allow to connect with at least most common hardware. With their centralized design, the setup is not very complex. Another big advantage is that all of these platforms provide at least one user interface.

The disadvantage of the middleware approach is the centralized design and that both of the above mentioned types are not intended for intensive data analysis based on sensor data. Because of the centralized design, the middleware approach is badly scalable and less fail-safe [8], [10], [14], [22].

B. Agent-based Smart Home Environments

Agent-based smart home environments are famous in research. Their main focus is on data analysis for plan or activity recognition [2], [5], [7], [9], [19], creation of user/behavior models [5], [21], task prediction [6], or energy harvesting/consumption prediction [3].

The advantages of the agent based approach are that they can communicate with each other to avoid conflicting objectives, they can share information and join collaborative groups. A further advantage is the decentralized or peer-to-peer design, which allows mobile agents to move to the platform/host with the most processing power [] and avoids Single Point of Failure (SPOF) [5]. As such, the system is more robust.

The disadvantages are that there is little, not standardized hardware compatibility for agent-based frameworks and decentralized systems are typically hard to configure/setup.

III. RELATED TECHNOLOGY

One of the core concepts of our approach is the usage of standards-based open architectures. For the agent communication we have chosen the FIPA specifications. As a compliant implementation we use the Jade framework as described in section III-B2. As middleware we exemplarily use the industrial Universal Remote Console (URC) standard (ISO 24752) [13] and the OSGi-based UCH implementation as described in section III-A1 and section III-A2.

A. The URC framework

The URC framework provides an adaptive architecture that supports the flexible integration and reuse of heterogeneous software and hardware components, communication protocols, and target appliances. The focus of URC is the notion of Pluggable User Interfaces that allows for interfacing arbitrary targets. The advantage of this approach is the possibility to use consistent, secure, personalized and, perhaps most important, accessible user interfaces [11]. Users can select a user interface that fits their needs and preferences, using input and output modalities, and interaction mechanisms that they are familiar with and work well with them [25].

1) The URC standard — Standardized Contract between User Interface and Target: The standardized definition of the User Interface Socket (UIS) describes the input and output behavior of a specific target device on an abstract level and contains a flat set of socket elements that provide a synchronized communication channel to the controlled device and its current state. The example shows an example socket for a thermostat target device depicting a variable for the current temperature, a notification for requesting a user acknowledge for resetting the device and two commands for confirming and canceling.

Fig. 2. The User Interface Socket (UIS) describes the input and output behavior of a specific target device on an abstract level and contains a flat set of socket elements that provide a synchronized communication channel to the controlled device and its current state. The example shows an example socket for a thermostat target device depicting a variable for the current temperature, a notification for requesting a user acknowledge for resetting the device and two commands for confirming and canceling.
A Universal Control Hub (UCH) is a gateway-based architecture implementing the URC standard and managing the bidirectional communication between controller and targets. It uses a session-based authentication to separate requests from command invocations. On the backend side each target appliance is represented by a dedicated target adapter that is responsible for the grounding of abstract socket elements with any specific network protocol.

2) Architecture and Implementation of the URC platform:
A Universal Control Hub (UCH) is a gateway-based architecture implementing the URC standard and managing the bidirectional communication between controller and targets. It uses a session-based authentication to separate requests from different users and to manage access to targets. Further the UCH guarantees that a direct internal communication between different target adapters is not possible.

Figure 3 shows the modular architecture of a UCH [26] and the four main components of the URC platform: the UCH, the Target Adapters (TA), the User Interface Protocol Modules (UIPM) and the resource server. Each of the components will be described in the following. The UCH architecture mainly consists of three layers:

1) The Target Adapter Layer — It manages the communication with actual targets. It is responsible for loading and managing target adapters. A target adapter consists of software and possibly hardware components. In most cases a network adapter is sufficient thus only the software component is called Target Adapter (TA) in the following.

2) The User Interface Protocol Module Layer — It is responsible for loading and managing User Interface Protocol Modules (UIPM) which specify the communication protocols for the controllers, e.g., HTTP, SOAP, SVG, UPnP or proprietary solutions. A UIPM usually only consist of a software component. Its main task is to translate any protocol from and to the URC standard. Thereby it can do a simple translation between equivalent protocols as in the case of URC-HTTP or it can be more complex by containing logic and/or combining functionalities of different targets.

3) The User Interface Socket Layer — It defines the standardized User Interface Socket (UIS) that serves as a common contract between controllers and targets (see Section III-A1). Thereby targets can consist of several (different) sockets. This layer is the main part of the UCH and is responsible for the communication between UIPMs and TAs, for session handling and controls the standard compliance.

The Resource Server is the only component, that is not directly related to a layer of the UCH. It is a global service for sharing UIPMs, TAs, Pluggable User Interfaces [23], and various resources necessary for interacting with the targets, e.g., language resources, icons or presentation templates defining the arrangement of interface elements. The UCH can download and install the Pluggable User Interfaces, UIPMs, and TAs automatically if required.

As an implementation we use the UCH, which is available as OSGi, standalone Java or C implementation. All of them are compliant to the URC standard. We have to use one of the Java implementations to be able to use the Jade framework. Since a lot of the middleware platforms use OSGi, we selected the OSGi variant.

B. The Agent Framework

A Multi Agent System (MAS) is an environment for multiple agents to ‘live’ in and communicate/interact with each other. Thereby the agent platform provides the infrastructure for agents. It has to manage the life cycles of agents, their addressing, and the communication between the agents and between different platforms. It also has to care about management and discovery of the services provided by agents. An agent framework or MAS can consist of several connected agent platforms, thus it is highly scalable.

An agent is specified as a program with the following characteristics [1], [20]:

- autonomous: It has control about its actions and internal state and operates without the direct intervention of humans.
- social: It cooperates with humans or other agents in order to achieve its tasks.
- reactive: It perceives its environment and responds in a timely fashion to changes that occur in the environment.
- proactive: It does not simply act in response to its environment but is able to exhibit goal-directed behavior by taking initiative.
- goal-based: It has a goal, a finite state and all actions should lead to this particular state.
- adaptive: It can adapt its state to new situations and thus is able to learn.
- mobile: It has the ability to travel between different nodes in a computer network. This is an optional characteristic.
With respect to an MAS an agent has to be uniquely identifiable and it should promote its services to the platform/other agents.

1) The FIPA Specifications: The IEEE Foundation for Intelligent Physical Agents (FIPA) promotes agent-based technology and the interoperability of its standards with other technologies. It has been the standards organization for agents and multi-agent systems since 2005.

The FIPA specifications is a collection of standards to facilitate the interoperation of heterogeneous agents and the services that they can represent [12]. For example they include:

1) The Agent Communication Language (ACL) — It specifies message parameters. For our usage, only a small subset is important: sender, receiver, performative, content, language, ontology. Thereby the language parameter specifies the language that encodes the content, e.g., FIPA Semantic Language (FIPA SL).

2) The Agent Communication Channel (ACC) — It specifies the message transport envelope and the transport protocol. Usable protocols are HTTP or IIOP.

Agents can create message filters based on the ACL parameters. With these filters they can express what kind of messages they can handle and they can subscribe to specific messages at the agent platform.

2) The Architecture of the Jade Platform: A Jade platform can consist of multiple connected agent containers. Agents can travel between these containers. An agent container is a Java process that provides management services for agents. The first container in the network serves as the main container and has a special role. All other containers have to register to this main container, which coordinates the addressing of containers and agents [1]. It has the following responsibilities:

- Administration of the Container Table (CT), a table with addresses of all registered containers.
- Administration of the Global Agent Description Table (GADT), which saves all agent ids with their corresponding address, container and state.
- Hosting of the two specialized agents AMS and DF defined by the FIPA specifications. These agents have to be started at the initialization phase, prior to all other agents.

1) The Agent Management System (AMS) serves as White Pages and has to administer the life cycles of the other agents.

2) The Directory Facilitator (DF) serves as Yellow Pages and manages subscriptions. Agents can register their services and subscribe to services of other agents. The DF can also work distributed over several agent platforms.

When the platform is initialized, agents should ‘live’ on it and interact with each other. To tell agents how they should behave, Jade provides the abstract Java class Behaviour. You can inherit from this class and create behaviors that can be added to agents. An agent processes all its behaviors sequentially until all of them are finished and goes to standby. Sleeping agents can be waked up by ACL messages. Another advantage of Jade is the support for ontologies. It provides three main building blocks to create an ontology:

- Concept: Represents an entity with a structure defined in terms of slots.
- Predicate: Represents an expression saying something about the world, typically relating concepts.
- Agent action: Represents an action that can be performed by an agent, can inherit concepts.

IV. THE PROPOSED BRIDGE BETWEEN AGENTS AND MIDDLEWARE

The proposed framework works in two different directions. First, it enables agents to function as a controller to monitor and control targets. Second, it provides agents the possibility to function as a target, so that it can provide an arbitrary abstract interface which then can be rendered automatically to provide a user interface, e.g., to monitor the state of an agent.

A. Agent as a Controller

To provide a gateway between agents and targets a special Jade UIPM is used. This Jade UIPM creates its own agent container — connected to the main container — which runs a UCH agent that represents the state of the UCH. The user interface sockets of all targets get reported to the DF agent, the Yellow Pages of the main container. Other agents can communicate with the UCH via the UCH agent.

1) UCH Agent: The UCH agent connects the agent framework with the UCH as shown in Figure 5. Other agents can subscribe to different sockets, to get informed about value changes, or they can manually ask for specific values. The UCH agent also listens for discovered or discarded targets in the UCH and directly updates the Yellow Pages service of the DF agent. Other agents can ask the DF agent what
Fig. 5. The UCH and the agent framework are connected through the UCH agent. For communication between the UCH agent and other agents in the URC ontology is used. Other agents can subscribe to different sockets, get value changes, or ask for specific values. The UCH agent also listens for discovered or discarded targets in the UCH and advises the DF agent. Other agents can ask the DF agent about available UCH services. UCH services are available and how they could be addressed. Messages to the UCH agent must have the following format:

<table>
<thead>
<tr>
<th>performative</th>
<th>subscribe</th>
<th>request</th>
<th>protocol</th>
<th>fipa-subscribe</th>
<th>fipa-request</th>
</tr>
</thead>
<tbody>
<tr>
<td>language</td>
<td>fipa-st</td>
<td></td>
<td>ontology</td>
<td>Uch-Ontology</td>
<td>Uch-Ontology</td>
</tr>
</tbody>
</table>

**TABLE I**

Message format of the communication with the UCH agent

When an agent sends a request to the UCH, an open session request is automatically made in the UCH and the resulting session is associated with the AID of the agent. Already opened sessions are reused thus agents do not have to care about session handling.

2) **URC Ontology**: To have a well-defined communication between the UCH agent and all other agents, the URC ontology is used. The URC ontology is a simple ontology that reflects the functionalities of the URC-HTTP protocol as shown in Figure 6. The URC-HTTP protocol provides the following methods for interaction:

- **Get** — request states of specific socket.
- **Add, Set, and Remove** — request state changes of specific socket.
- **Acknowledge** — acknowledge and quit notifications.
- **InvokeCommand** — invoke a command with parameters provided as a specific socket.

The update channel provided by the URC-HTTP protocol is realized by the subscription mechanism. In addition to the functionalities of the URC-HTTP protocol the URC ontology provides the attribute `TimePeriod` which allows to request the history of specific states. This parameter is optional since it is no standard feature of the UCH. It also requires an additional logging UIPM and a database/persistance TA.

3) **Predefined Behaviors for UCH Communication**: To simplify the programming of agents, we developed some behaviors that can be used out of the box:

- Request a list of AIDs of UCH agents at the DF agent.
- Request all sockets provided by a specified UCH agent.
- Request states of a specified socket, given the AID of the UCH agent.
- Change states of a specified socket, given the AID of the UCH agent.
- Invoke commands of a specified socket, given the AID of the UCH agent.
- Subscribe to specific fields of a socket, given the AID of the UCH agent.

With the help of these behaviors it is easy for agents to discover UCH platforms in the framework and to obtain target states or control targets.

**B. User Interfaces for Agents**

Creating user interfaces for agents, especially for mobile agents, is a difficult task. In contrast, the URC platform provides an easy way to create accessible user interfaces based on abstract user interface definitions. To provide a gateway between agents and interfaces, a special TA is used as shown in Figure 7. This TA creates its own agent container, similar to the previous described Jade UIPM, but running a UIS agent instead. Other agents can request the UIS agent to create a user interface socket according to a given instance of the UIS ontology. When the UIS agent processes such a request, it generates a standard compliant UIS description, as described in section II-A1, and notifies the UCH that a new target is discovered. If the socket has changed by a UIPM, the corresponding agent is informed. If the socket has to be changed by its corresponding agent to reflect state changes, this agent can send an update message to the UIS agent. To avoid processing loops/cyclic service dependencies, it is important that the previous described Jade UIPM does not forward the function of this Agent Interface TA to the agent platform.
Fig. 7. To provide a gateway between agents and Interfaces, a special TA is used. This TA creates its own agent container running an UIS agent. Other agents can request the UIS agent to create an user interface according to the UIS ontology. When the UIS agent processes the request, it generates a standard compliant UIS description and notifies the UCH, that a new target is discovered. If the socket is changed by a UIPM, the corresponding agent is informed. If the state of agent has changed, it can send an update message to the UIS agent.

1) UIS Agent: The UIS agent connects the agent framework with the UCH as shown in Figure 7. All other agents can send requests to this agent to create interfaces. The UIS agent creates a target for each agent that requests an interface. The requesting agent automatically subscribes to its corresponding target thus it gets informed about all changes of the target. It can send update message to update the values of the target. Messages to the UIS agent must have the following format:

<table>
<thead>
<tr>
<th>performative</th>
<th>subscribe</th>
<th>request</th>
</tr>
</thead>
<tbody>
<tr>
<td>protocol</td>
<td>lipa-subscribe</td>
<td>lipa-request</td>
</tr>
<tr>
<td>language</td>
<td>lipa-xl</td>
<td>lipa-xl</td>
</tr>
<tr>
<td>ontology</td>
<td>UIS-Ontology</td>
<td>UIS-Ontology</td>
</tr>
</tbody>
</table>

Table II: Message format of the communication with the UCH agent

In more detail, the UIS agent creates a socket description for each requested interface and a corresponding socket model according to the UIS ontology. This interface socket description is a UIS as described in section 3 II-A1 an XML file compliant to the URC standard. Since single sockets cannot be loaded dynamically in the UCH the UIS agent has to discard the target with the old socket descriptions and discover a new target with the updates descriptions, if a new interface is requested. If an agent unsubscribes to its corresponding target, the target will be discarded.

2) UIS Ontology: To have an easy way to create abstract interfaces compliant with the URC standard we created an ontology that reflects a part of the standard. Thus it is possible to create user interface socket descriptions without knowledge of the URC standard. The UIS description contains a set of building elements. Each element can has access rights — e.g., read only — which can be static or dependent on the state of an other element. The following element types are supported:

- **Set** — An element that has an index. It can contain all elements including other sets. The type of the index can be specified thus it can be used as a list or a map.
- **Variable** — It represents a variable with any type that is available as an XML Schema Definition (XSD).
- **Command** — It represents an executable command. It has a state that should reflect the executional state (initial, in progress, success, failed, . . . ) and it can have parameters.
- **Parameter** — It represents a parameter of a command. It can only be used inside a command definition.
- **Notification** — It can be used to notify a user. An activated notification stays active until it is acknowledged.

Our framework currently only supports Java primitive data types and strings and does not support dependencies.

3) Predefined Behaviors for Interface creation: To simplify the programming of agents, we developed some behaviors that can be used for interface generation:

- Request a list of AIDs of UIS agents at the DF agent.
- Request to create a user interface.
- Request to update a specific field of a specific user interface.
- React on value changes, command initiation and notification acknowledgements.

With the help of these behaviors it is easy for agents to discover UCH platforms in the framework and to create accessible UIs, e.g., to reflect internal states.

C. Application scenarios

There are several ways to set up and utilize the smart home system described in this paper. In this section we will describe some possibilities.

1) Fully centralized setup — It could be set up highly centralized with only one single computing component, the UCH. In this case, the UCH’s agent container is the main container. Both, the Jade UIPM and the Agent Interface TA will be installed in the UCH. To avoid processing loops/cyclic service references, it is important that the Jade UIPM does not forward the function of the Agent Interface TA to the agent platform. All targets are connected to this single UCH thus all targets could be
controlled by agents or directly by other UIPMs. This setup approach has a Single Point of Failure (SPOF) which could be problematic especially when the smart home has no fallback controls.

2) **Fully decentralized setup** — It could be set up in a distributed way, where several UCHs are connected to different targets. In this setup all UCHs have the Jade UIPM and the Agent Interface TA installed. The main agent container can run somewhere else in the same network. The UIPMs of each UCH and the user interfaces connected to it can only access the local available targets. The agents can choose on which UCH they request a user interface. When a UCH crashes, only the connected targets become unavailable thus the system has no SPOF. The user can directly control the different devices by accessing the UIPMs of the corresponding UCH. In this setup there is no central user interface that has control over the complete smart home.

3) **Hybrid setup** — It could be set up in a distributed way, where several UCHs are connected to different targets but on UCH servers as the main user interface. In this setup all UCHs except the main one have the Jade UIPM installed. The main UCH runs the Agent Interface TA. Since the UCH provides cascading with the use of a cascading TA and also discovery through Bonjour and UPnP with the according TAs and UIPMs, all UCHs can be discovered and remote controlled by the main UCH. Thus the user has only to connect to the main UCH to have full control of all connected devices and agent interfaces. This setup approach has no SPOF. When the main UCH crashes, an other UCH can become the main UCH. The agents can control and observe the environment without the main UCH and also all UCHs can be controlled by accessing their UIPMs directly. When another UCH crashes, only the devices connected to this UCH are not controllable anymore. This setup is our proposed one and it will be used in the AdAPT project [9]. In this setup the main UCH can be seen as a gateway between the smart home and the outer world. Thus it has a special role for security, this is a good example for the use of a secure UCH (SUCH), which has a role and policy based security mechanisms and access restrictions. Such a secure implementation based on Common Criteria for Information Technology Security Evaluation (CC) facing Evaluation Assurance Level 4 (EAL 4) is currently under development [1].

In the second or third setup scenario it is possible that mobile agents move to different containers, e.g., the platform with the most processing power or the best energy efficiency. Also in the same setup scenarios, it is possible to create an agent for automatic failure detection and possibly failure correction. Failure correction could be done for example by the creation of a new main agent container or by transferring the role of the main UCH. The transfer of the main role of a UCH is possible since TAs and UIPMs can be dynamically loaded and installed from a resource server. In a setup with analysis agents as plan recognition, it is also possible to load a personal agent, e.g., from a mobile device, that instructs the plan recognition agent with the user’s habits to improve the prediction results.

V. CONCLUSION

In this paper, we presented an initial overview of modern smart home systems. Thereby we detail on the two main research branches: the middleware platforms and the agent based approaches. Based on the standardized URC middleware platform and the Jade framework we proposed a system that merges these two research branches. Within this framework, agents can access functionalities of various devices and services connected to different URC platform. They can easily provide accessible user interfaces and can travel between agent containers to maximize their computational power. When using a decentralized setup, a Single Point of Failure is avoided and the framework is capable to fix certain failures. To face other problems with smart home frameworks, the proposed framework is flexible and easy to set up, scalable, and highly extendable. Furthermore we are planning to integrate the secure UCH platform (SUCH), which allows to create a system that strongly faces security problems of state-of-the-art smart home systems.

ACKNOWLEDGMENT

This research is partly funded by the German Federal Ministry of Education and Research (BMBF) under grant number 01IS12050 (Project AdAPT) and under grant number XXX (SUCH). This work is partially supported by EIT ICT Labs. The responsibility for this publication lies with the authors.
REFERENCES


