User-friendly Configuration of Smart Environments

Simon Mayer*, Nadine Inhelder*, Ruben Verborgh†, and Rik Van de Walle†
*Institute for Pervasive Computing, ETH Zurich, Switzerland
Email: simon.mayer@inf.ethz.ch
†Multimedia Lab, Ghent University – iMinds, Ghent, Belgium
Email: ruben.verborgh@ugent.be

***Abstract*—The configuration of smart homes represents a difficult task for end-users. We propose a goal-driven approach to this challenge, where users express their needs using a graphical configuration environment. Our system then uses semantic descriptions of devices in the user's surroundings to derive a plan to reach the desired situation. We are able to satisfy complex demands using only first-order logic, which makes this system flexible yet fast. The focus of this paper is to demonstrate how to achieve high usability of the proposed system without burdening users with the underlying semantic technologies. Our initial demo supports setting the ambient temperature, alarms, and media playback, but the use of semantics allows to extend the system with many different kinds of services in a decentralized way.**

I. Introduction

Despite a grand effort by academia and industry to bring the vision of the *smart home* to life, and although the necessary home automation and communication technologies have been available for some time, smart homes have not yet been widely adopted [1], [2]. This adoption failure seems to a large extent due to the inflexibility and poor manageability of current solutions in the home automation domain [1], [3], [4].

In this paper, we discuss a novel mechanism that can compose heterogeneous services in smart homes to yield *physical mashups*. Our main objective is to facilitate the integration of services for end-users by adopting a *goal-driven* approach where users are merely required to state which properties their smart environment should have. From such a statement, a reasoning component in the system determines whether the currently available services can be used to reach the user's goal, and which concrete user actions (in this case, HTTP requests to different services) are required to reach it. By executing these requests, the user then modifies his environment to reflect the conditions specified in his goal.

Figure 1 illustrates a concrete use case of the system: an application that runs on a tablet computer or smartphone can be used to specify the user's preferences – for instance with respect to their comfort temperature – and negotiates with smart devices in the user's surroundings to implement these preferences. In the course of this negotiation, the application is supported by a semantic reasoner that transparently creates service mashups. For instance, to play songs of a specific music genre at the user's current location, multiple services must cooperate: an indoor localization system is used to locate the user and discover a media player in the user's proximity; another service is responsible for finding playable audio files of songs that correspond to the given genre; finally, the discovered media player must be configured to play these files. We want applications such as this to operate successfully

Fig. 1. Users can modify their smart environment using an application that runs on a tablet or smartphone. In this case, the application gives feedback about whether it was successful in matching the user's preferences with respect to the ambient temperature, audio playback, and an ambient alarm (mock-up).

in arbitrary environments including users' private homes, office environments, and even public places. In an industrial context, manufacturing systems could for instance be automatically reconfigured when producing small batch sizes, or machines could adjust to support their current operator [5]. We also imagine that similar systems could be applied in medical environments, for instance to support doctors during clinical diagnosis by automatically adjusting monitor systems.

However, because of the underlying semantics that our system requires, it is potentially hard to use for end-users who are required to be syntactically and semantically precise when formulating goal states of their environment. Therefore, after a brief overview of how we use semantics to configure smart environments in the next section, we discuss two techniques to mitigate this problem by facilitating the formulation of goals for end-users: In Section III, we present an integration of the system with a visual programming language that constrains users to modeling only specific properties that are supported by their current smart environment. The second mechanism, which we propose in Section IV, aims to reduce the burden on the end-user by extracting goal templates from services present in the user's environment and having the user simply select an appropriate goal, rather than writing it himself.

II. Semantics-assisted Service Composition in Smart Environments

To enable the automatic composition of services provided by devices in a smart environment and the execution of mashups

that are derived in this way, we need to communicate in a machine-readable way what *functionality* a service provides and how to *invoke* it. However, because all services we consider feature Web APIs and because their protocol semantics are therefore already specified by HTTP, we only need to embed information about a service's high-level domain semantics and link that data to its Web API. To do this, we use RESTdesc [6]. A RESTdesc description captures the functionality of a service by relating its *preconditions* to its *postconditions* through a first-order logic rule. Services advertise descriptions using a `describedby` relation in the `Link` header[1] as part of their responses to HTTP `GET` and `OPTIONS` requests. Given such descriptions, a semantic reasoner can infer which individual services have to be invoked to achieve higher-level goals within service mashups, and which concrete HTTP requests a client must send to do this.

RESTdesc goals are expressed in Notation3 (N3), a superset of RDF that adds support for quantification and is grounded in first-order logic. RESTdesc therefore works well to describe services that do not induce states or state changes such as, for instance, a converter service. However, to support situations that involve states – which are very common in use cases that involve real-world physical objects (e.g., a room having a specified ambient temperature) – we needed to extend the system by adding an explicit state handling mechanism. In the extended system, services can express that they induce state transitions by using a publicly available states ontology[2].

As a concrete example of how the system works, we assume that a user wishes to set the ambient temperature at a specific location to 23°C. To communicate this desire, the user would need to formulate the following goal:

```
:temp23 a ex:Temperature;
  ex:hasValue "23";
  ex:hasUnit "Celsius".

?state a st:State; log:includes
  { :Office ex:hasTemperature :temp23. }.
```

The first three lines define the `temp23` object, which includes the value of the desired temperature and also specifies that this value is given in degrees Celsius. The rest of the goal description makes use of that entity to describe the desired state of the object `Office`.

To find out how to set the temperature, the client contacts a reasoner[3], which can either run locally on the client or remotely (to additionally support resource-constrained clients). Given the URLs of individual services in a smart environment, this reasoner can find their RESTdesc descriptions by following the links within the `Link` header fields of their responses to HTTP `OPTIONS` requests and will create a local service descriptions catalog. When a client now asks how a given goal can be reached, the reasoner is invoked with all descriptions from the catalog and the user's goal, and will construct a path from the current state to the goal state. If such a path exists, the reasoner returns a description of the necessary HTTP requests, which can then be executed by the client to modify his environment according to the specified goal.

[1]http://tools.ietf.org/html/rfc5988

[2]http://purl.org/restdesc/states

[3]We use the Euler Yap Engine (EYE), available at http://eulersharp.sourceforge.net/

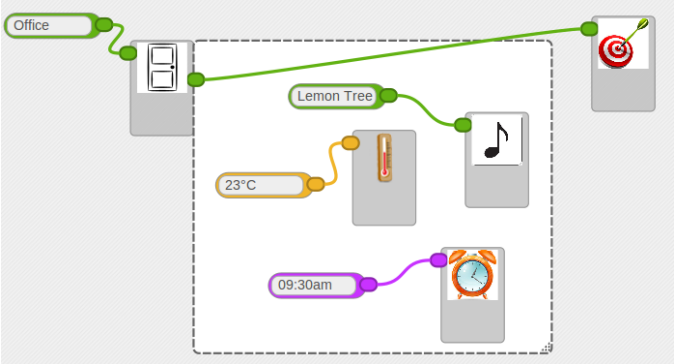

Fig. 2. The *ClickScript* interface allows users to create goals using an intuitive graphical programming abstraction. To do this, a user selects entities that abstract from specific conditions within the desired state of his smart environment and connects them to input data, where colors correspond to different data types.

III. Creation of User Goals by End-Users

To make use of the proposed service composition mechanism, end-users need to be able to formulate semantic goals that describe their preferences and are correct with respect to syntax and semantics. Because we cannot assume that users are familiar with semantic technologies, this step is hard to accomplish for users, so supporting mechanisms are required. Ideally, such a support system would go beyond assisting users with the goal syntax and actively guide the goal formulation process, for instance with respect to the concrete ontologies that are used within goals.

As a first step to facilitate the process of formulating goals for end-users, we integrated the system with ClickScript[4], a visual programming tool that provides an abstraction layer on top of the semantic goal descriptions. Figure 2 shows the editing view of ClickScript where a user sets up the desired configuration of his smart environment. The first step is to create a new room entity and to connect it to the corresponding room identifier (in this case, `Office`). Next, the user models his desired state of this room by adding components which represent different aspects of that state: the note icon represents media playback, the thermometer icon stands for the ambient temperature (i.e., the `ex:hasTemperature` property in the goal shown in the previous section), and the alarm clock icon is used to model ambient alarms. Finally, the user can infer the correct data types of the components' input parameters (for instance, the concrete temperature value) from the colors of the component inputs. A user's task thus only consists of dragging the desired elements to the editing view, connecting the matching input and output types, and entering the parameter values.

When satisfied with the configuration of his smart environment, the user can choose among multiple options of how the created model should be processed by the system by connecting one of multiple components to the output connector of the room entity. The user's first option is to output the goal textually on the screen by connecting the "Target" component (this situation is shown in Figure 2). Alternatively, the system can provide a human-readable description of the HTTP requests that are to

[4]http://clickscript.ch

be executed. Finally, the user can choose to have ClickScript itself execute these requests, and thereby directly modify his smart environment to match the modeled goal state.

ClickScript does not only parse syntactically correct goals from modeled environments, but also constrains the user to specific services that smart environments can provide: the current prototype implementation supports setting the ambient temperature and ambient alarms, as well as configuring media playback. However, the graphical editor can easily be extended with further abstractions, for instance for managing the room ventilation, or for enabling users to configure ambient displays. According to our own and others' experience, the ClickScript tool is intuitive to use even for people without any programming know-how [7].

IV. Automatic Creation of User Goals

In the previous section, we demonstrated that ClickScript allows users to formulate goals from given templates. However, it can also derive components of user goals at runtime in a fully automatic way, based on the available services in the user's surroundings, e.g., the fact that the environment should *have a specific temperature* or that a media player *causes a media file to be played*. This mechanism thus removes the necessity to have an expert create new components for the ClickScript interface. To enable it, the reasoner automatically generates a list of potential goals from the RESTdesc descriptions of services that are present in the environment, and presents this list to the end-user. Specifically, our system derives a potential user goal from each postcondition of discoverable RESTdesc documents.

Because these documents are attached to services that are provided by specific devices, this mechanism allows us to use a smart device itself as a proxy for all mashups that involve a goal state which corresponds to a postcondition of this device's description. In other words, it is possible to use object recognition software (in the way described in [8]) on the user's handheld device to recognize smart devices and immediately display potential actions that they can perform on behalf of the user. Figure 3 illustrates this situation: by recognizing the projector, the system can find out that this device could display an image stored on the user's tablet. Alternatively, the system can unobtrusively propose a mashup with a third-party image editing service to convert the image to grayscale before displaying.

V. Conclusion

In this paper, we presented a novel approach to the composition of services that are offered by physical things in smart environments. In our system, a user formulates a goal to specify the desired state of his environment which is used by a semantic reasoner to deduce the HTTP requests necessary to reach that goal. From our perspective, using semantic technologies to deduce service mashups represents a much more flexible alternative to the process-driven composition of services: because the services are combined at runtime, the system can flexibly react to individual services becoming unavailable by finding alternative paths that also serve to reach the user's goal. Furthermore, the reasoning process could also take into account more information about the user context, or his preferences, to derive mashups that are even better suited for a concrete situation.

Fig. 3. Using image recognition software and automatic deduction of user goals from RESTdesc descriptions, a projector is used as a proxy for service mashups that it can be part of. In this case, the projector offers to display an image stored on the tablet in two different – automatically generated – ways.

One major challenge in the proposed system is that the goal formulation step is hard to accomplish for end-users. For this reason, we extended the system by integrating it with a graphical editor that enables users to easily create a model of the desired state of their environment and translates this model into a goal in the Notation3 format. We also explored a method of deducing potential user goals directly from the service descriptions of smart devices present in the user's environment, thereby transforming the problem of goal formulation into one of merely selecting an appropriate goal.

Acknowledgments

This work was in part supported by the Swiss National Science Foundation under grant number 134631. The authors thank Jos De Roo for his help with the EYE reasoner.

References

[1] A. J. Brush, B. Lee, R. Mahajan, S. Agarwal, S. Saroiu, and C. Dixon, "Home Automation in the Wild: Challenges and Opportunities," in *Proc. CHI*, 2011, pp. 2115–2124.

[2] R. Harper, "From Smart Home to Connected Home," in *The Connected Home - The Future of Domestic Life*, R. Harper, Ed., Springer, 2011, pp. 3–18.

[3] C. Dixon, R. Mahajan, S. Agarwal, A. J. Brush, B. Lee, S. Saroiu, and P. Bahl, "An Operating System for the Home," in *Proc. NSDI*, 2012, pp. 25–41.

[4] W. K. Edwards and R. E. Grinter, "At Home with Ubiquitous Computing: Seven Challenges," in *Proc. UbiComp*, 2001, pp. 256–272.

[5] J. L. M. Lastra and I. M. Delamer, "Semantic Web Services in Factory Automation: Fundamental Insights and Research Roadmap," *IEEE Trans. Ind. Informat.*, vol. 2, no. 1, pp. 1–11, 2006.

[6] R. Verborgh, T. Steiner, D. Van Deursen, S. Coppens, J. Gabarró Vallés, and R. Van de Walle, "Functional Descriptions as the Bridge between Hypermedia APIs and the Semantic Web," in *Proc. WS-REST*, 2012, pp. 33–40.

[7] D. Guinard, C. Floerkemeier, and S. Sarma, "Cloud Computing, REST and Mashups to Simplify RFID Application Development and Deployment," in *Proc. WoT*, 2011.

[8] S. Mayer, M. Schalch, M. George, and G. Sörös, "Device Recognition for Intuitive Interaction with the Web of Things," in *Adj. Proc. UbiComp*, 2013, pp. 239–242.