Journal Title
XX(X):1–17
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Enabling Smart Behaviour through Automatic Service Composition for IoT-based in Smart Homes

Maria J. Santofimia[1], David Villa[1], Oscar Aceña[1], Xavier del Toro[2], Cristian Trapero[1], Felix J. Villanueva[1] and Juan Carlos Lopez[1]

Abstract

This paper presents an IoT architecture for Smart Homes that specifically targets service composition and reconfiguration as enablers for the actuation and smart behavior capabilities. To this end, the main challenge that has to be addressed is the support to a seamless integration, composition, and reconfiguration of IoT objects. Two enabling technologies are proposed here: a planning strategy based on a common-sense reasoning approach for service composition and a virtual-network protocol for Inter-Domain Messaging (IDM). The planner will identify the services that, properly connected, will cater for arisen, and therefore, unexpected needs. The virtual-network protocol will provide the support for this interconnection to take place in a transparent and orthogonal manner. This is particularly important to enable autonomous systems to instantiate composite services. To demonstrate the capabilities of the resulting framework two use cases are presented, which under real circumstances demonstrate the potential of the proposed approach.

Keywords

IoT, Automatic Service Composition, Smart Homes, Service semantics, Virtual Networking

Introduction

Traditionally, research efforts in IoT have been addressed to support data-collection applications, overlooking the need for actuation and smart behavior. The main challenge these data-collection applications has to face is how to deal with data-source heterogeneity. Two main approaches can be identified in the literature and the market, namely: the use of the Cloud and the use of gateways. Cloud-based applications delegate the management of IoT networks and data to remote servers. Under this perspective, the Cloud typically assembles a set of repositories in which sensors have published their values. These data are, afterwards, retrieved and processed to take a centralized decision. Such architecture enables the interoperability between different IoT networks only at the Cloud level without any direct communication between the information source and those that use it. The use of gateways, as technology translators, is also a common solution, although as stated in Tao et al. (2018), the conversion among protocols also entails an important overload that affects performance.

The acting side of IoT systems has been traditionally overlooked, mainly due to the inherent complexity of supporting interoperability among IoT objects. A review of the state of the art for IoT interoperability support brings about solutions classified into the following five groups Tayur and Suchithra (2017): standards Alliance (2016), reference architectures and frameworks Shelby and Chauvenet (2012); Datta and Bonnet (2015); Datta et al. (2015); Mulligan (2007), protocols and media-type standards Shelby et al. (2014); Hunkeler et al. (2008); Fielding et al. (1999), and definition languages and ontologies Compton et al. (2012); Alaya et al. (2015). The work of Tayur and Suchithra (2017) concludes that interoperability has to be addressed at the application layer by combining the aforementioned techniques.

From our point of view, acting in IoT does not solely depend on enabling interoperability at the application layer but, additionally, on the capability to understand the context and the objects that populate it. To this end, we point out the need for two enabling elements: 1) a model that captures the semantics of the system and human behavior and 2) the capability to automatically compose or reconfigure service functionality.

Regarding the need for semantic models, the most complex ones are those involving humans. We cannot obviate that modeling human behavior is far from being a trivial task due to the inherent complexity of human aspects such as cognition, psychological preferences, or emotions Nunes et al. (2015). The main reason behind this complexity lays in the unstructured and large-scale characteristic of the knowledge that rules human behaviour. Our approach to tackle this complexity consists in using the *mental state* abstraction. All that information regarding humans that is relevant for the system performance is modelled in terms of concepts such as *beliefs*, *knowledge*, *free will*, *intentions*, *consciousness*, *ability*, or *wants* Mccarthy (1973). Then, the

[1]Computer Architecture and Networks Group, University of Castilla-La Mancha, Paseo de la Universidad, 4, Ciudad Real, Spain
[2]Institute of Energy Research and Industrial Applications, University of Castilla-La Mancha, Ciudad Real, Spain

Corresponding author:
Mara J. Santofimia,
Email: mariajose.santofimia@uclm.es

use of a common-sense knowledge base as it is Scone Chen (2009); Fahlman (2011); Chen and Fahlman (2008) will support inference, search, and reasoning processes over that information. **The Scone project[*], led by Scott E. Fahlman at the Carnegie Mellon University, represents an opensource knowledge-based approach in which the focus is not on collecting common-sense knowledge but rather at providing the means for supporting common-sense reasoning mechanisms. The Scone system therefore pays special attention to providing an expressive, easy to use, scalable and efficient approach for accomplishing search and inference operations.**

Regarding the capability to automatically compose or reconfigure services, it is our believe that the service semantics has to be decoupled from the service itself and its programming interfaces in order to be modelled and stored as knowledge about the service. By decoupling services from their semantics, service composition and reconfiguration can be automated and therefore delegated to a standardized computation system. In contrast to traditional approaches used for service composition, such as web services, in our approach the programmer does not have to explicitly state how services can be composed. On the contrary, this information is implicit in its semantic description, from which an advanced inference mechanism can derive how services can be composed to offer a composite functionality. This is a disruptive approach to service elicitation since the information about how services can be composed is not handcrafted by the programmer or service designer. On the contrary, it is infered from the service description. Moreover, the composition or reconfiguration process is normally motivated by an arisen need. In this sense, our approach is also disruptive because this *need* is not foreseen in advanced. In other words, there is no need for a previous declaration in terms of tuples $\{need, services\}$. In contrast, our approach consists in describing the service, from a semantic point of view, and then look for the service configuration whose semantics matches the arisen need. For example, think of a situation in which a need for illumination arises (during a blackout for instance). Since no power is available none of the services specifically designed for that purpose are available. Our architecture provides the mean for the system to conclude that switching on the cell phone screen will satisfy that need. This scenario illustrates how an understanding of what a cell phone is, and how it works, leverages more flexible capabilities for dealing with unexpected needs.

The main contribution of this paper is the proposal of an architecture for IoT that explicitly targets acting and smart behavior. The proposed architecture provides support for seamlessly integrating heterogeneous devices and technologies and for understanding and reacting to ongoing context situations. These capabilities will be grounded in a novel modeling approach that combines human behavior and a novel IoT information model. Additionally, the proposed architecture faces the technology heterogeneity issue proposing a virtual protocol that supports communication among different technology domains. To demonstrate the acting and smart behavior capabilities of the proposed architecture we have evaluated the platform performance in a testbed scenario in which different use cases have been carried out. The testbed has considered the deployment of different sensors and actuators and the retrofitting of already-deployed appliances to incorporate communication and acting capabilities. The IoT system considered in this test-bed scenario is aimed at improving the well-being conditions of its inhabitants and the energy efficiency of the building, all that following a low-cost approach.

This paper will be organized as follows. First, the Previous Work section reviews previous solutions for Smart Homes, specially those provided by the market, paying special attention at how those solutions target the problem of understanding and acting in IoT. The next section describes the lexical approach we propose for service composition and reconfiguration. The Proposed Architecture section describes the insights of our proposal, describing the different layers comprising the architecture. The Experimental Evaluation section describes the hardware prototypes that have been designed and deployed in the testbed scenario. Finally, the Conclusions section summarizes the main ideas withdrawn from this work.

Previous works

The introduction of Internet of Things (IoT) paradigm and the proliferation of ubiquitous Wireless Sensor Network (WSN) have enabled machine to machine (M2M) connectivity in the Smart Home. In the work of Hui et al. (2017) the authors identify the major requirements for building Smart Home systems in which multiple people interact with the environment. The device heterogeneity is, once again, pointed out as the main challenge to be addressed. The long list of network protocols and data structures currently in use makes it difficult to integrate different solutions.

The analysis of the state-of-the-art solutions for device heterogeneity brings about the following approaches. A Server Centralised Architecture (SCA) is proposed by Xuemei and Gang (2008) as a solution to connect devices in the home space, using to this end a home gateway. SCA addresses the incompatibility issues that arise when trying to communicate protocols like 6LoWPAN, Bluetooth LE, ZigBee or Z-Wave. The proposed solution consists in a hub server in which these protocols have been implemented. That same approach has been followed by some companies like Wink[†] or Samsung Smartthings Hub[‡]. The use of gateways, despite being a widely employed solution, leads to incompatibility issues when trying to integrate devices that implement protocols different from the ones initially considered by the manufacturer. On the contrary, open-source automation platforms like OpenHab[§], Home Assistant[¶] or Domoticz[‖] are specifically aimed at

[*]http://www.cs.cmu.edu/~sef/scone/
[†]https://www.wink.com/
[‡]https://www.samsung.com/us/smart-home/how-it-works/
[§]https://www.openhab.org/
[¶]https://home-assistant.io/
[‖]http://www.domoticz.com/

facing this problem. These platforms are supported in open-hardware devices such as Raspberry Pi, which enables the control and automation of different technologies in Smart Homes.

The increasing complexity of services and devices has called for new paradigms that assure aspects such as scalability, interoperability and reliability. The Service Oriented Architecture (SOA) is one of these approaches intended to provide a number of software services re-mapped in a typical Software-as-a-Service (SaaS) cloud architecture for reshaping home services and applications. Moreover, these architectures have, traditionally, paid great attention at the automation of the composition task.

The Cloud Computing paradigm has gained attention as a solution to address interoperability among different vendor devices, easing the process of interconnecting different services and therefore leveraging the expansion of smart environments. The use of the Cloud also provides a new solution to integrate on-board network modules in home devices which connect the devices to Internet, simplifying the process of mapping, encapsulating and composing the services that provide. There are some solutions such as Apple Homekit[**], Samsung Smartthings[††] or Google Home[‡‡], which provide a platform to integrate different vendors devices to control them through cloud infrastructures. Some platforms such as Ambient OS[*] or Amazon Alexa[†] provide a framework for developers, in order to integrate speech recognition and natural language understanding (NLU) capabilities to simplify the user interactions with the devices.

The device integration capabilities are restricted by the cloud platforms because they have a limited number of calls defined through an API which communicate the user operations with the devices. In spite of the SOA approach, the service composability does not work properly in cloud platforms due to the lack of uniform treatment of services.

Interconnecting heterogeneous devices and services provided by different vendors and providing seamless interoperations across the available platforms still remains a big challenge. Tao et al. (2018) describe a multi-layer cloud architecture model and an ontology-based security framework to integrate the different common cloud-based platforms using ontologies to address data, knowledge, and application heterogeneity in the available devices.

The IoT paradigm is generating an unprecedented volume and variety of data and although the Cloud computing has many advantages, it requires a high bandwidth to exchange the vast amount of data between the devices and the Cloud infrastructure. This leads to increase the latency in the time response of services and spreading the data to other locations different from the source. It is in this point where the Edge computing paradigm has the aim to push computation on acquired data away from the core of data centers and get closer to the data sources.

Projects such as iSapiens Cicirelli et al. (2017) propose a platform which implements the Edge computing paradigm through both the exploitation of the agent metaphor and a distributed network of computing nodes directly scattered in the smart environment. The current vision of Smart Homes is focused on the management and control of devices. Nevertheless, the raw data produced by those devices does not provide any meaningful value. It is in this situation where the context-aware computing brings value by deducing knowledge and providing better understanding of raw data. The work of Lalanda et al. (2017) and Rahman et al. (2017) takes advantage of the Edge computing and pervasive applications to propose context-aware platforms to provide a reasoner in charge of deducing knowledge and dealing with the environment by using context management. The design of these platforms is based on a service component model or OSGi specification that describes a modular system and a service platform for the Java programming language. Frameworks such as Apache Felix or Eclipse SmartHome provide the necessary tools for building Smart Home solutions based in the principles of modularity, component-orientation, and service-orientation. Lukas Smirek et al. Smirek et al. (2016) evaluate the Eclipse SmartHome framework to address backend technologies and personalized user interfaces in a Smart Home. These solutions, despite being very convenient for basic service composition lacks the flexibility demanded by the IoT vision.

A three-stage process for service composition and reconfiguration

The architecture proposed in this paper is mainly intended to support actuation and smart behaviour in IoT. To this endeavour, it should be equipped with the appropriate mechanisms for enabling service composition and reconfiguration, while assuring high-flexibility and low coupling in the process. These requirements are imposed because the composition or reconfiguration process has to be automatically driven and accomplished and, therefore, no human intervention has to be required before, during, or after the composition or reconfiguration process.

The proposed approach consists in a three-stage process in which the semantic, syntactic, and lexical compatibility is verified before accomplishing the composition or reconfiguration process. Whereas the lexical compatibility concerns about providing the programmatic support for service composition and reconfiguration the syntactic compatibility concerns about the assurance that the data provided and consumed by the bound services match in the expected format and content. Finally, the semantic compatibility concerns about the functionality provided by the bound services.

One of the main characteristics of the spaces envisioned by the IoT paradigm is that they tend to be driven by events. In this sense, most of the IoT objects populating these spaces are intended to capture the reality into messages or events. IoT objects are eventually intended to perceive the activities that are taking place and, consequently, react to them. The mechanisms therefore provided for supporting composition and reconfiguration capabilities count on the premise that services implement an event-driven approach. This is totally compliant with what sensor services are expected to do, as

[**] https://www.apple.com/es/ios/home/
[††] https://www.samsung.com/us/smart-home/smartthings/
[‡‡] https://developers.google.com/actions/smarthome/
[*] ttps://www.essential.com/home
[†] https://developer.amazon.com/alexa/smart-home

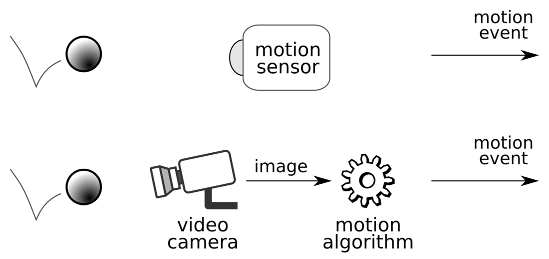

Figure 1. Physical and virtual sensor services

known: wait for an event to take place and then, notify its occurrence in a reactive way.

It is important to highlight that most of the services considered in our architecture follow this reactive-system approach. Nonetheless, other approaches can be considered and supported by the architecture with the only difference that those that do not comply with the reactive approach are not considered for automatic composition or reconfiguration.

The mechanism proposed here for service composition and reconfiguration is eventually based on the idea that events can be *virtually* propagated, in a transitive manner, without having to propagate the external stimulus that caused it. Services can be connected in a short of pipeline with these transitive events working as the linking element. Figure 1 depicts this idea. The first case scenario describes the situation in which a bouncing ball is the stimulus captured by a motion-sensor service, which turns it into a motion event. This event, for example, can be the trigger for a lighting service to illuminate the room. The second scenario in the same figure replicates the first one but rather than using a single service it uses a composite one. This composit service provides the same functionality by binding a video-camera service and a motion-algorithm service, that implement video content analysis, capable of detecting motion. The bouncing ball stimulus is captured by the video recording service. The stimulus is therefore captured in a video that is transmitted to the motion algorithm, which will, as result, notify of a motion event.

Service composition or reconfiguration is based in the capability to build this service pipeline, whose triggering is determined by the occurrence of a given event. As it can be observed from the example outlined in Figure 1 it is essential that effects of events can be propagated in a transitive manner without having to propagate the stimulus itself. Only by supporting this transitiveness, services can be automatically composed or reconfigure without affecting its normal behaviour. This approach for composition and reconfiguration also assures that there is no difference, not even in the way the are used, between services provided by physical sensors and those pipelines, resembling virtual ones.

There are, however, some important requirements that need to be satisfied in order to enable the automatic generation of service pipelines, as known:

1. A mechanism to support the pipeline links, in terms of programming interfaces and event propagation support.
2. A mechanism to verify the syntactic compatibility of services to be linked.
3. A mechanism to determine the semantic compatibility of services.

The following subsections address each of these requirements.

Support for service binding: lexical compatibility

Figure 1 depicts the way how service composition and reconfiguration is carried out by establishing a pipeline of services linked by unidirectional messages, propagated all along the pipeline. The service pipeline is supported on the fact that every service points to its next element in the pipeline using, to this end, the service-reference address. Additional support is required for the binding process so that the following requirements can be met:

- No human intervention should be required in the process of configuring a service to send (in a one-way fashion) the results it produces to the next service in the pipeline.
- It should support on-runtime configuration, meaning that there is no need for a predefined list of possible connections.
- Bindings can be modified any time.
- One service can be bound to more than one consumer: One-to-n relations are supported and made transparent for the pipeline builder, known here as the *scheduler*.

To address these requirements, we propose that all services that can, eventually, be part of a reconfiguration or composition work, implement a common interface. This interface, named `Linkable`, provides a method for establishing the reference to the next service. The method is named `linkTo`, as described in the following code listing:

```
interface Linkable {
    void linkTo(string next);
};
```

Despite its simplicity, the implementation of this interface assures that any service can be bound, automatically (without human intervention), to other services. It is important to recall that the binding process $S_1 \to A$ implies the creation of a unidirectional channel from S_1 to A through which service S_1 communicates a message to service A.

It is the role of the *scheduler* to determine which services are going to be part of the pipeline, and the order in which they will be bound. It is therefore its responsibility to invoke the `linkTo` method of the first service with the reference of the next service in the pipeline, as argument. By doing so, the *scheduler* is stating that the first service is being *observed* by the one whose reference has been provided with. Figure 2 depicts the linking process. First, the scheduler notifies service `S1` the reference of the service to link to, in this case service `A`. This is accomplished through the invokation of the `linkTo` method, reciving as argument the reference to service `A`, depicted in Figure 2 using the `&` symbol.

This mechanism assures that any service can be part, eventually, of a service pipeline as long as it implements the

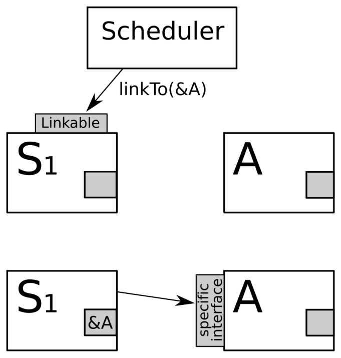

Figure 2. Process of linking two services

`Linkable` interface. Additionally, the pipeline construction can be delegated to a computation entity (no human intervention required) that can, on runtime, decide the services that will comprise the pipeline. The fact that the binding process is orthogonal assures that it can be accomplished automatically, without requiring any type of presetting.

There will be situations in which the information provided by one service can be of interest to more than one services. For example, more than one services can be interested in the information provided by temperature sensor. The mechanism described in Figure 2 cannot be used, as it is, because it is limited to having just one consumer. Binding service $\mathtt{S}_1$ to a hypothetical service B would imply the replacement of the previous reference A.

An additional mechanisms for supporting this one-to-n connection is therefore required. To this end, we propose the implemetation of the traditional solution to this problem, as it is an event-propagation mechanism based on communication channels. Figure 3 describes this process. As it can be observed, there is an intermediary service, referred here as the *wiring service*, in charge of decoupling the sheduler (or other services interested in the linking process) and the services involved in the pipeline.

The *wiring service* takes care of the channel creation, management, and elimination (once they are not needed any more). This service also tracks the list of subscribers to each channel. This knowledge enables the wiring service to create a channel only when more than one service is consuming events, or it can delete the channel when less than two consumers are subscribed to the channel.

Compatibility at the syntax level

The second stage of the composition or reconfiguration process verifies that the service pipeline is valid in terms of the sintactic correctness of the unidirectional messages used as linking elements. The *validity* concept refers here to the expected data type or format and content. Due to the fact that service pipeline has to be dynamically constructed (without requiring human intervention), an appropriate mechanism has to be provided to automatically verify this compatibility. In other words, referring to Figure 2, it is necessary to provide a way to assure that the method $\mathtt{S}_1$ invoked on `A` is, indeed, provided by `A`.

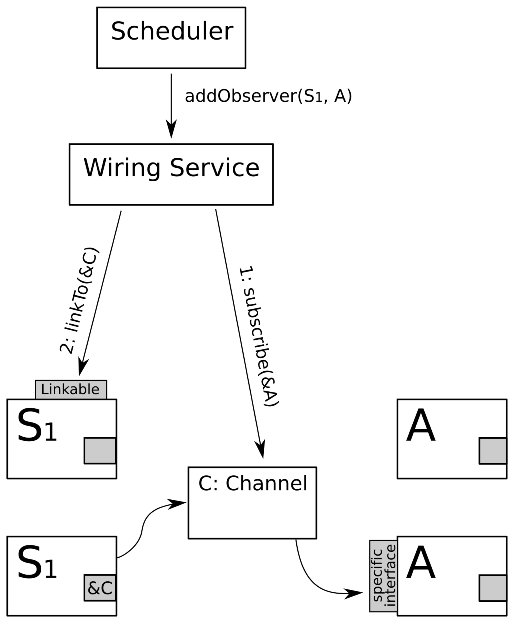

Figure 3. Process of linking one service to several ones

Despite the fact that at first, this can be considered a simple type-checking problem, the need for checking other aspects rather than just types calls for more advanced mechanisms. For example, in the same case depicted by Figure 1, consider that the motion algorithm consumes images in `jpg` format. However, in terms of data type, the service is described as consuming `Byteseq` (a sequence of bytes). A basic type-checking compatibility will permit the binding of a service providing a `png` image with the aforementioned motion algorithm that consumes `jpg` images, since both are `Byteseq`.

The paradigm of design by contract (DbC) or contract programming provides the means to state and verify these type of constraint checking. Languages like Ada provide native support for DbC. However, these are programming languages and not service or interface description languages so this would not be totally appropriate. In fact, the interface description language employed, at the middleware level, is not expressive enough to support the statement of these types of constraints. This is not, however, a particular problem of the employed language, on the contrary, the most commonly-used description languages (**WSDL**Christensen et al. (2001), **IDL**Cleary et al. (1998), **Protocol buffers** Buffers (2011), etc.) suffer from this limitation. This calls for a way to enhance service descriptions so that the correctness, at the syntax level, of the service binding can be automatically assured.

Therefore, following a DbC approach, and due to the limited expressiveness of the service description languages, we propose to model these type of restrictions at the knowledge-base level, in which the flexibility provided by the description language is enough to capture the clauses of contract.

According to the semantic model described in more detail in the next section the *capability* concept is the one employed for modeling these type of constraints. Then, at the knowledge-base level, Scone provides a close-to-natural-language description language in which the constraints employed in the example of Figure 1 can be captured as follows:

```
(new-type-role {required-cap} {service} {thing})
(new-type-role {provided-cap} {service} {thing})
(new-indv {jpg} {image format})
(x-is-a-y-of-z {jpg} {provided-cap} {PullSnapshotService})
(x-is-a-y-of-z {jpg} {required-cap} {MotionAlgorithmService})
```

Following the Scone semantics, both `required-cap` and `provided-cap`, have been defined as *roles* or properties of service. Then, the function `x-is-a-y-of-z` is used in Scone to assign a particular value to a given role. In this case, the `jpg image format` is the `provided-cap` of the `PullSnapshotService` service, as well as being the `required-cap` of the `MotionAlgorithmService`.

The scheduler, in charge of deciding the services comprising the pipeline, will just have to verify that provided and required capabilities, if any, match for the bound services.

Apart from the capability-matching checking, it is also necessary to verify that the services to be bound are compliant in terms of provided and required interfaces. From the middleware perspective, since all services implement the `Linkable` interface the programmatic compatibility is assured. The `next` parameter of the `linkTo` method, intended to hold the reference of the next service in the pipeline, does not force any specific type, for two reasons:

- This is the most flexible way to support service binding, given that the service semantics is moved to the knowledge base where more advanced verification works can be carried out.
- This is an addressing-agnostic approach that decouples the type of address from the referred service. In this work, indeed, two addressing schemes are employed: one based on the middleware built-in proxy representation and other based on the addresses used by our protocol for Inter-Domain Messaging or IDM.

It is important to highlight that the alternative to the use of a common and *empty* interface, as the one we propose, is to have an interface for any type of connection we would like to support. For example, to support a hypothetical reference to a `B`-type service or a `C`-type service the following interfaces would be required: `linkTo(B* next)` and `linkTo(C* next)`.

Compatibility at the semantic level

The simplification of the service-binding and type-verification process is a requirement for the service composition and reconfiguration task to be automatically carried out in an unsupervised manner. This simplification has been achieved, as described in this section, by migrating all the service semantics to a level in which higher expressive power and advanced reasoning mechanisms can be supported.

We propose the use of Scone[‡], an open-source knowledge-based system written in Common Lisp. It implements efficient search algorithms, based in the marker-passing algorithm proposed by Fahlman (2006), mainly intended to provide answers in reasonable time, even when the answers are not optimal. Scone supports a higher-order logic language very convinient for describing the domain-specific and context knowledge as well as the insights of IoT services, in terms of the actions and events they are related to and the interfaces they implement, require, or use.

The following code listing shows the flexibility and expresiviness of the considered description language. The *motion sensor service* is described as follows:

```
(new-indv {MotionSensor-service-instance} {MotionService})
(x-is-the-y-of-z {EventSinkInterface} {used-interface} {MotionService})

(new-statement {MotionSensor-service-instance} {causes event through}
{motion detected} :c {EventSinkInterface})

(new-indv {motion_sensor -t -e 1.1:tcp -h localhost -p 9000 -t 60000} {proxy})
(x-is-the-y-of-z {motion_sensor -t -e 1.1:tcp -h localhost -p 9000 -t
  60000} {service-proxy} {MotionSensor-service-instance})
```

The `x-is-the-y-of-z` function is used in Scone to state a certain role of a given concept. In this case, it states that the `used-interface` of the `MotionService` is the `EventSinkInterface`, meaning that the *motion sensor service* can be bound to a service that provides that same interface. For example, the *motion sensor service* could be bound to a *snapshot service*, described underneath, because one uses the same interface implemented by the other:

```
;; SnapshotInterface
(new-type {SnapshotInterface} {interface})
(new-is-a {SnapshotInterface} {LinkableInterface})
(new-is-a {SnapshotInterface} {EventSinkInterface})

;; SnapshotService
(new-type {SnapshotService} {service})
(x-is-the-y-of-z {SnapshotInterface} {implemented-interface} {SnapshotService})
```

Actions and events are also two essential concepts for a complete description of a service. In this case, a tertiary relationship is employed to describe that the *motion sensor service* causes a *motion detected* event through the *EventSinkInterface*. This means that when this type of event is required the *motion sensor service* can be bound to a service implementing the *EventSinkInterface* to cause that event to take place.

Finally, certain services also provide or require a capability, or a set of them. For example, in the following code listing, the `SnapshotService` is described as capable of providing a *jpg* file format.

```
(x-is-a-y-of-z {jpg} {provided-cap} {SnapshotService})
```

This means that a service, as the following one, requiring a `jpg` file could resort to it to *semantically* satisfy that requirement:

```
(x-is-a-y-of-z {jpg} {required-cap} {PersonRecognizerService})
```

The semantic compatibility of two services to be bound is determined considering the following aspects:

- The interfaces that one of them provides and that the other uses have to be compatible.
- The capabilities that one of them requires and that the other one provides should also be compatible.

[‡]https://github.com/sfahlman/scone

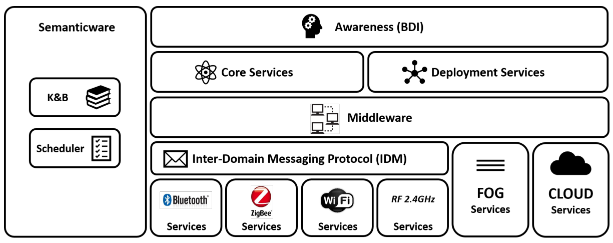

Figure 4. Proposed architecture

The compatibility of interfaces and capabilities is something more ellaborated than a simple verification of the equality of source and target. Scone employs a semantic-network approach, which means that properties inherits from general types to specific ones. This is very powerful because semantic compatibility offers more posibilities for service composition and reconfiguration than forcing strict semantic equality. Moreover, Scone provides native support for semantic compatibility checking offering functions such as `can-x-be-a-y` or `is-x-a-y` among some.

Proposed architecture

This paper proposes an achitecture for IoT-based Smart Homes that specifically targets support for actuation and smart behaviour. Two capabilities are identified as enablers for this endevaour, namely:

1. The capability to automatically understand ongoing situations and the available means for undertaking responses. This capability is mainly based on information about the services deployed in the context, and the knowledge about *how the world works*, also known as *common sense*.
2. The capability to seamlessly integrate, compose, and reconfigure IoT objects. This capability relies on a mechanism for managing the underlying heterogeneity, by providing an abstraction layer upon which objects can communicate to each other in a symmetrical way.

Figure 4 outlines the different elements that comprise the proposed architecture, organized in layers. At the bottom layer we can identify the service oriented architecture (SOA). Current systems for IoT involve services implementing different computational models, like the Edge, Fog, or Cloud Computing. It is important to highlight that all these services have to be equally treated, independently of their implementation details. An abstraction layer is provided by IDM to homogenize the underlying heterogeneity. On top of this there is the middleware layer, built upon a general-purpose object-oriented middleware, whose core functionalities have been extended with advanced capabilities. The most important aspect of this layer is the implementation it does of the semantic model proposed at the *semanticware* layer. This semantic model is common to all the layers of the proposed architecture and it is what supports service interoperability in a transparent manner. The knowledge base is also part of the semanticware layer which, along with the scheduler, is where composition and reconfiguration capabilities reside in. Finally, the awareness layer holds the behavioral models that determine how IoT systems have to behave as result of their understanding of what situation is taking place. The following subsections describe the details of the modules comprising the proposed architecture.

Service layer

Services are located at the bottom layer of the proposed architecture, as the most basic building blocks. As it can be observed from Figure 4 all services are equally considered independently on the underlying communication technology they employ or the computational model they implement. This represents a disruptive approach in the sense that the proposed architecture assures a symmetrical treatment to services independently on how or where they are being deployed or implemented. To be more precise, in any IoT system we can identify the following service types:

- Those provided at the Edge-computing level, generally implemented in IoT devices. This implies that the device and the service it provides are totally coupled. Among some of the most important features of these services, one can highlight their low latency, since the data source and target are directly connected, or the high privacy level they provide since there is no need for data transportation.
- Those provided at the Fog-computing level, generally implemented as application gateways or hubs. We refer here to services that perform some data aggregation, manipulation, or any type of interaction that cannot be carried out at the Edge level. For example, the Fog approach is suitable for some video-processing applications, highly demanding in terms of required resources.
- Those provided at the Cloud-computing level which, by the way, are the most widely spread. Some applications are offered at the Cloud level for cost-efficiency purposes (do it once and provide it to many users) or for resource optimization (specially for machine learning applications which demand large datasets for training purposes, for example), exploiting the most relevant advantages of this computing model.

Whereas traditional SOA have considered services at the same level (either the Edge, Fog, or Cloud) our proposal does not make that distinction and pursues service homogenization, independently on the computational approach they follow. Leveraging automatic service composition or reconfiguration is the motivation behind this homogenization effort. The fact that all services can be equally treated enables a computation entity, as it is the scheduler in our case, to automatically compose or reconfigure services, in an unsupervised manner.

Providing for this transparency is specially important when we are dealing with Edge services because, as it has already been mentioned, they are usually highly coupled to the underlying communication technology. Nonetheless, IoT devices tend to be constrained in terms of memory or processing capabilities which typically lead to the unfeasibility to implement the full TCP/IP stack. In fact, sometimes it is not desirable, nor even possible, to replace specific-purposes protocols with a standard one. Therefore, relying on TCP/IP for communication purposes is not always a choice.

We propose a novel approach for homogenization purposes consisting in a virtual network protocol for inter-domain messaging, that enables every IoT object, or *thing*, to be univocally addressed and accessed, independently of its inherent technology and location. This protocol, known as IDM, is in essence, similar to IP, but it focuses in addressing the IoT issues for which the use of IP is not directly an option Villa et al. (2017). IDM is pursuing a twofold aim: 1) providing an abstraction layer around these technologies; and 2) supporting a seamless interaction among these different technologies.

The importance of catering for these two aims can be more easily understood from the following example. Imagine one has a WiFi appliance (i.e a lamp) in the living room, and this person is also wearing a BlueTooth SmartBand. It would therefore be very handy if the lamp functions (switch on and off) could be controlled by different tapping patterns in the SmartBand. Despite being a very basic problem, it perfectly illustrates the type of challenges arisen due to technology heterogeneity. Different solutions can be proposed to articulate this application:

1. One may implement a WiFi/BL bridge, and translate messages between both devices. This is a fast solution but not the best one because it is highly coupled to the specific devices and therefore very difficult to port elsewhere.
2. One could also make both devices to use TCP/IP, the de-facto standard. It cannot be obviated that we are dealing with constrained devices in which a full implementation of the stack (usually big) is not feasible.
3. One could use a Cloud server, so that the SmartBand may send messages to the Cloud whereas the lamp may be a listener for incoming messages.

Option number 3 is probably the most accepted solution for the considered scenario. However, it cannot be overlooked that the use of the Cloud is very well suited when data aggregation is required to obtain the *big picture*. This is not, however, the need posed by the considered scenario. What is demanded here is a *network* of IoT objects, and a cloud is hardly a network. On the contrary, the Cloud offers a repository of data, which is not bad by itself, but not what is demanded here. Moreover, the use of the Cloud also involves high latency, low fault tolerance due to temporal network failure and a poor efficiency, since it ends up sending a value, through the Internet, to a nearby object.

The most sensible solution therefore involves the SmartBand *directly* talking to the light appliance. IDM offers a technological solution to support this direct communication only using a device to change the communications technology (WiFi/BlueTooth). IDM creates a real object network in which there is no need to use IP everywhere and in which the Cloud is only used when it is really useful (not as a way to work around the technology heterogeneity). IDM provides a mechanism to interconnect objects, so that they are able to communicate to each other in a symmetrical way, without having to know the details of the underlying communication technology. More importantly, with IDM there is no need to transform messages or to set up handlers to process the message between them.

A very important aspect of IDM is that every resource on the IoT network (every sensor or actuator), is an object. Its main implication is that if a single device (hardware) holds more than one resource an individual object will be allocated for each resource. This approach is suitable for constrained nodes as objects can be simple.

It can be assumed that any end device (either sensor or actuator) provides an interface (understood as a set of well-known operations). The role of the IDM protocol is to transport the invocation messages from the client to the end device. These end devices can be referred as objects. The term is inherent to the current implementation of IDM which is based on an object-oriented middleware (ZeroC Ice[§]). However, the use of this term is not totally accurate since they are closer to a Service-Oriented Architecture (SOA) than to an object-oriented one.

The main objective of IDM, as it has been already stated, is to support the intercommunication between network technologies that are, a priori, incompatibles. The use of the name domain stems from the fact that in the IDM infrastructure all the devices that share a technology and addressing scheme are seen as a single entity (a domain). Thus, the entire public Internet is a single IDM domain.

The router does not change the messages it forwards at all. For example, you can receive a message from an RS485 device on one of its interfaces and forward it to a device on a Bluetooth network. That is a key point, the IDM router has no status, does not create device delegates or proxies, does not transform addresses, only forwards complete messages between its interfaces.

This is possible because the IDM message remains unaltered from its creation at the client to its arrival at the target object. IDM routers only change their encapsulation, in a similar way to an IP router. Obviously the router needs to have an interface in every domain it interconnects, but the specific details of that network's technology are hidden from the rest. Unlike a conventional network protocol such as IPv6, IDM addresses refer to objects rather than nodes (a node can hold several objects). These two features allow the IDM message to be encapsulated even on the LAN's link protocol, regardless of the "local" network protocol.

Regarding the services offered at the Fog and Cloud level, it is necessary to provide an adapter that enables the communication between the middleware layer and the service itself. For example, a Cloud service that provides a speech-recognition system, like IBM Watson, offering a REST interface (HTTP), has to be adapted to provide the middleware protocol. Regarding the Fog services, it has to be taken into account that most of them are not third-party services and they are therefore provided by the platform itself. This means that there is no need for adapters since they will normally use the underlying middleware technology.

[§]https://zeroc.com/

The information model for the middleware

The proposed architecture is using, at the middleware layer, a general-purpose object-oriented middleware as it is ZeroC Ice. ZeroC Ice is a remote-procedure-call-based middleware developed by the USA company ZeroC[¶]. The interfaces of any service developed in ZeroC Ice have to be defined in the interface definition language, known as *slice*. After the slice definition, the developer can generate bindings for different languages. Inter-operation among clients and servers is supported independently of the underlying language or the platform, thanks to the Ice protocol (IceP).

ZeroC Ice also comes with a complete set of tools and services to deal with recurrent issues in distributed systems (IceStorm, IceBox, IceGrid, etc.). These core services provide support for event propagation, deployment, platform or node management, etc. We have extended these core services with some capabilities that support the service deployment process and, eventually, the tasks involved in composing services. The middleware has been enhanced with the following services:

- A property service: This service is intended to hold static information about the different elements comprising our system (services, devices, locations, etc.). This service is implemented as a key-value database in which for every property (or key), like *location* there is a value associated to it, *room-1* for example.
- A discovery service: This service provides a list of services complying with a list of requirements, in terms of locations, capabilities, properties, etc.
- A context service: It is like a directory of devices and services. These objects are organized based on a hierarchy, and these hierarchies can be defined by the service users. As a user, for example, you might want to have all services providing temperature measures organized under the same directory.
- An advanced event service: This is an improvement over the event service provided by the middleware. This service supports content filtering enabling, for example, an advanced selection of the events that one might want be notified of, based in the area where these take place. Additionally, this service provides persistence capabilities.

These services cater for service composition and reconfiguration, at a very basic level, by providing efficient mechanisms for managing the interconnection of services and their information exchange. It cannot be obviated that the process of automatic service composition and reconfiguration has to deal with more complex challenges, for which the three-stage process has been designed to. Recall that this process deals with compatibility issues at the lexical, syntactic, and semantic level. The role played by the `Linkable` interface is essential in supporting this three-stage process. However, additional interfaces are required for the normal functioning of services. For this reason and for interoperability purposes, we propose a set of interfaces compliant with the standard proposed by the IPSO (IP Smart Objects) alliance which, at the same time, is based in the object model specified in the OMA LightWeight M2M Alliance (2015).

We can categorize the provided interfaces into two sets: the data-centric interfaces and the message-centric interfaces. The `Linkable` interface, described in detail in the previous section, represents the message-centric interfaces. The data-centric interfaces are intended to communicate data, generally the data associated to sensor events. We have considered the following data-centric interfaces:

```
interface EventSink {
    void notify(string source, Metadata data);
};

interface DigitalSink {
    void notify(bool value, string source, Metadata data);
};

interface AnalogSink {
    void notify(float value, string source, Metadata data);
};

interface DataSink {
    void notify(ByteSeq data, string source, Metadata meta);
};
```

`EventSink`: This interface is intended to notify the occurrence of an specific event. The sensed event is determined by the device that uses the interface. For example, if we are dealing with a glass-break detector sensor that is supervising the state of a window. This is a one-time event that is not revocable, which means that when a windows glass is broken, the sensor emits an event. The event message does not need to contain any data or measure because it is implicit by the event itself.

`DigitalSink`: This interface is intended to deal with events that have two possible states. For example, this is the interface used by a motion sensor which is capable of detecting two states: whether there is movement or not. An event is notified whenever a change in the state is detected. It means that when motion was detected an event is notified containing a True value, and when no motion is detect an event is notified containing a False value.

`AnalogSink`: This interface is intended to notify events related to scalar measure. For example, a temperature measure. The event contains a single measure.

`DataSink`: This interface is intended to notify events containing a stream of data. Generally, it will be used for services providing video streaming or sound streaming. It is important to highlight that the codification used by the streaming is not included in the event. These properties are static and are located in the property service or in the knowledge base.

All these interfaces have two common parameters:

`Source`: The source argument is a string containing the identity of the source that generated the event. The detailed information about the source is held by the Property Service.

`Meta`: This argument contains dynamic information about the data itself. For example, the timestamp of the event.

Semanticware layer

The term *semanticware* has been coined to refer to the layer that holds semantic information. Similar in functionality to a middleware, whose main purpose is to offer an abstraction layer to support information exchange, the semanticware guarantees that all these exchanges work upon the same

[¶]https://zeroc.com/products/ice

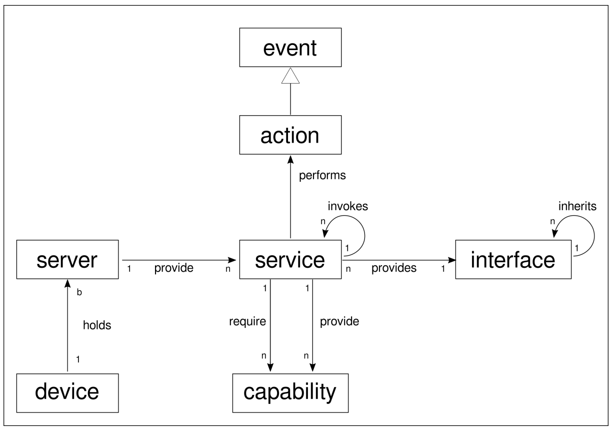

Figure 5. A semantic model for service composition and reconfiguration

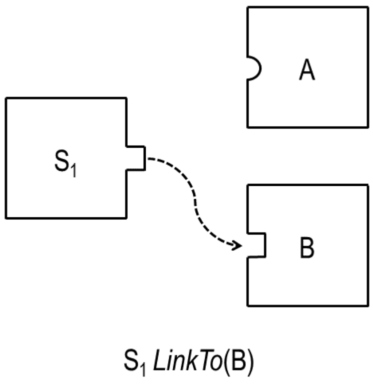

Figure 6. Syntactic matching in service composition

semantics and therefore, share the same understanding of the information or functionality employed.

Continuing with the middleware analogy and similarly to the role played by the programming interfaces, the semanticware is supported on a semantic model that identifies the concept and relationships that are relevant for the system. This model is depicted in Figure 5.

The ultimate goal of the semanticware is to support automatic service composition and reconfiguration. For this reason, a mechanism for service description has to be devised so that not only their functionality is captured, but also the way how they should be used. Figure 5 illustrates the concepts and relationships that have been identified to this end.

The *action* concept refers to those events with a known agent and, more importantly, that are motivated by a *primary reason* that eventually rationalizes it. This approach is, in essence, following the theory of *primary reasons* of Davidson (1963) that advocates the existence of a primary reason or a cause that motivates the realization of an action. The *event* concept is similar to the action one, with the sole difference that nothing is known about the agent that causes it. A *service* is described in terms of its used and provided interfaces. Traditionally, services have been described just in terms of the interface they provide without paying attention at the one they use to accomplish their task. A *capability* can be required or provided by a service. For example, a service that performs facial recognition on a png file, requires that *capability* to offer its service. On the other hand, a *snapshot* service that captures images in that file format will be described as a provider for that capability. The *interface* concept refers to the programming interface the service is either providing or using. This information will determine whether it is syntactically possible to compose two services. The composition is first guided by the events or actions they are capable of generating or demanding as a previous requirement, but then it is necessary to check out whether two services fulfilling a required enhanced functionality have interfaces that match, as depicted in Figure 6.

It is the role of the *scheduler* to orchestrate the different stages involved in the process of service composition or reconfiguration. First, once that an arisen need has been identified it is necessary to translate that into a pipeline of services, whose configuration and functionality, will eventually cater for that need. Next, it is necessary to automatically connect the services involved in the pipeline with no intervention of a programmer or a service designer.

Algorithm 1 describes the process we have devised to select a sequence of services whose combined functionality cater for a given need. The algorithm is intended to validate the syntactic and semantic compatibility of the selected services as well as, eventually, undertake the binding process, in an automatic manner.

Algorithm 1 plan-for (*event*)

```
 1: schedule = Schedule()
 2: action = get-action-that-causes(event)
 3: pre-events = get-preceding-events-for(action)
 4: for e in pre-events do
 5:     schedule += plan-for(e)
 6: end for
 7: service = get-service-for-action(action)
 8: capabilities = get-required-cap-for-service(service)
 9: pre-service = get-service-that-provides-cap(capabilities)
10: schedule += pre-service, service
11: return schedule
12: return schedule
```

The algorithm is provided with an event the system is interested in causing. The algorithm returns an ordered list of services to be sequentially bound. Figure 7 summarizes the proposed algorithm. As it can be observed, the scheduler is built using the semantic model concepts as they are the *actions*, *events*, *services*, *capabilities*, and *interfaces*. It is important to note that we use *actions* and *events* as two different concepts despite referring to the same reality.

According to the theory of actions proposed in Kent Bach (1980):

> "Actions are not events but instances of a certain relation, the relation of bringing about (or making happen), whose terms are agents and events."

As for events, the same auhtor states the following assumption:

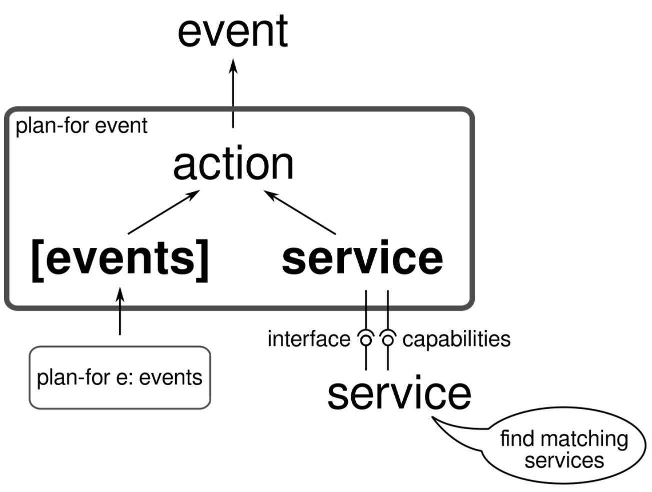

Figure 7. Scheduler service pipeline construction process

"I assume that events exist in space and time and that they enter into causal relations as causes and as effects."

The semantic model we propose complies with this theory and makes this distinction by considering that *actions* are (in a *is-a* relation) *events* with an agent (in a *has-a* relation) and *events* can be related using a special type of inverse relationship that connect one event with another one that *causes* it or vice verse.

The semantic model is captured in the knowledge base where the different actions and events are described in terms of the context *before* and *after* the action takes place. This approach lead us to relate the execution of an action with an event (or set of them) as prerequisites that should have occurred before the action can be accomplished. Furthermore, the execution of an action will lead to a new context in which several events might have been caused as result of the action execution.

The following code listing shows how the action of *executing a command by a person authorized to do so* is described in the knowledge base following this before-after context approach:

```
(new-action-type {execute_authorized_command}
                 :agent-type {person}
                 :object-type {command})
(new-context {e_a_c bc} {general})
(new-is-a {e_a_c bc} {before context})
(x-is-the-y-of-z {e_a_c bc} {before context} {execute_authorised_command})
(new-context {e_a_c ac} {e_a_c bc})
(new-is-a {e_a_c ac} {after context})
(x-is-the-y-of-z {e_a_c ac} {after context} {execute_authorised_command})

(in-context {e_a_c bc})
(new-indv {command is recognized in the scene} {command recognition})
(new-indv {person is recognized in the scene} {person recognition})

(in-context {e_a_c ac})
(new-indv {command executed by authorized person}
          {command execution by authorized person})
```

The before and after context are identified by the `e_a_c bc` y `e_a_c ac` respectively. In each of these context, we have described how is the world before and after the action `execute_authorised_command` takes place. As it can be observed, the before context requires the occurrence of two different events: one regarding the recognition of a command and the other one regarding the recognition of a person (including his/her identity). The execution of the action leads to the occurrence (as a direct cause) of an event that describe the execution of a command by a person authorized to do so.

Awareness layer

Mental states or mental qualities, as referred by Mccarthy (1973), deals with how to represent information regarding beliefs, knowledge, free will, intentions, consciousness, ability, or wants, which represent essential aspects of the human rationality. In his work Bratman (1987) proposed the BDI (Belief, Desire, and Intentions) model for human practical reasoning, as an explanation for the human rationality exhibiting goal-driven behaviors.

Unsatisfied goals is what motivates people to devise plans that lead to goal satisfaction, achievement or maintenance, and therefore, the emulation of intelligent behavior should inexorably be linked to an appropriate representation of the mental events involved in emulating goal-driven behaviors.

Moreover, context-awareness is one of the main requirements for enabling Smart Spaces, since the only way of wisely and proactively or actively reacting to context events is by understanding what is going on in the environment. The events or actions that take place in the context are noticed by means of the sensing devices and services deployed in the environment. Therefore, the only trace evidencing the occurrence of an event is the sensing values captured by any of these IoT objects. Ascribing those values to the effects of an event or an action is the only possible way of interpretation. The accuracy in understanding context situations depends on how extensively and thoroughly preconditions and effects of events and actions have been described.

The Smart Home paradigm relies on its ability to notice the situations that are taking place as well as its ability to generate appropriate responses to undergoing scenarios. However, on what basis does an environment conclude that a certain action is the most appropriate one? Smart Home systems, as humans, count on a set of goals to drive their behavior towards the achievement, maintenance, or desired performance of such environmental goals. In this respect, unsatisfied or deviated goals is what encourages Smart Home systems to devise the most appropriate way to return to or to achieve the desired state. Rather than using hard-coded responses to whatever circumstances that might arise in the environment, it is more feasible to simply dictate the environmental goals that the system is engaged in maintaining or achieving, and try to discern among the available actions, which of them seem more suitable in reducing the distance to the unsatisfied goals.

The way to reduce the distance between the current situation and a targeted one is by devising a plan, here understood as a course of actions. Action planning is intended to consecutively apply changes to an initial state so as to transform it into the goal state. The world states notion of the action planning is very similar to that of situation proposed by McCarthy, and therefore, can be modeled by means of the possible-worlds theory. The occurrence of a given event or action produces changes in the current state of the world. Under incomplete information and reasoning by default, we can expect the world to be in a finite set of states. Action planning, therefore, consists in successively applying changes to the world state to get a glimpse of the future world state.

Plans can be therefore understood as the behavioral responses generated by the Smart Home system whenever unsatisfied goals arise. The device dynamism and heterogeneity that characterize these environments makes it unfeasible to statically determine how those plans should be undertaken. On the contrary, plans should be automatically devised grounded on the knowledge of the devices and services, or IoT objects, available at a given moment.

The BDI (Belief-Desire-Intention) model of agency proposed by Bratman (1987) seems to be a compelling approach to cope with the demands involved in dealing with the identification and management of ongoing situations. To this end, the proposed architecture resorts to a set of software agents in charge of supervising the events to detect unsatisfied or deviated goals. As a result of this detection, plans will be launched to restore the desired state. The goal-driven agents have been built upon the semantic and the middleware layers, meaning that the communication aspects are totally transparent to the agents whereas the knowledge is available at the Scone knowledge base. These BDI agents, understand "beliefs" as the properties that an agent considers to be true, "goals" as the properties that an agent desires to be true, and finally "plans" as the actions that lead an agent to a desired goal. These basic instances define what is known as the agent's mental state.

The agent's beliefs in combination with contextual information (held in the Scone knowledge base) are what lead the agent's behavior towards the goals that the agent desires to achieve or maintain. Interaction between agents, the knowledge base, services and devices is based on the fact that all of them share the same semantic model.

Consider, for instance, the situation in which a person is in front of an office, looking for his/her key or access card, realizing that he/she has forgotten it at home. The system should devise a way to grant access to this person, knowing that he/she is authorised to access that space. The following mental state is held by the agent supervising the context:

- *Belief(a, b)*: Agent *a* believes that an authorised-access attempt event *b* has taken place.
- *Goal(a, g)*: Agent *a* desires to achieve the goal *g* that grants access to that person.
- *Plan(a, p)*; Agent *a* resorts to a set of actions or plan *p* to enable the person to access the room.

Evaluation

The proposed IoT platform for Smart Homes has been deployed for evaluation purposes in the Institute of Information Technologies and Systems (ITSI). This building belongs to the University of Castilla-La Mancha and hosts around 50 people working in different research groups.

Two scenarios are considered for evaluation purposes. One scenario is devoted to demonstrate the service composition capabilities, in the context of access control, whereas the other one is intended to demonstrate the reconfiguration capabilities, in the context of room temperature control. The scenario has been equipped with low-cost devices that avoid expensive and closed commercial solutions.

Like every morning, Bob gets to his office at 8:00 a.m.:

Scenario 1: Standing in front of the door he checks out his pockets looking for his badge. After a few seconds he realizes he left it in the car. By then, the smart environment supervisor detects the unusual circumstance of Bob standing at the door and ask him whether he needs something. Bob asks the system to *open the door* and since he is authorized to do so, the door opens and he can enter his office.

This scenario is intended to demonstrate the system capabilities to understand ongoing context situations and react to them. Despite the fact that this is a predefined scenario, no *recipe-like* instructions have been provided to the system in terms of service or list of services that can grant an access control functionality. On the contrary, based on the knowledge about the functionality of the services available at that location, the scheduler has to work out the service pipeline whose composition can eventually cater for the arisen need.

Scenario 2: As he enters his office his smartwatch *logically wires* to the temperature sensor in the office to accommodate the room to his comfort temperature. Bob's smartwatch holds the user preference profile including, among some other details, the comfort temperature he has defined or the one that has been empirically learned from his previous actions when he has used the smartwatch to directly operate the heating, ventilation and air conditioning (HVAC) system console, as though it was a remote control.

This scenario shows the system capability to reconfigure, on runtime, the behavior of IoT objects, represented here by the temperature sensor. The HVAC system is normally operated on the basis of a set-point manually introduced by the user, to which end the console buttons are employed. Our scenario demonstrates that this control can be extended to be manually or automatically controlled in remote. The user can resort to a *smartwatch* App to control the temperature manually, by tapping the smartwatch, or automatically, by *logically wiring* a temperature sensor to the smartwatch and have this operating over the HVAC system console.

Testbed description and hardware prototypes

Temperature control is an important aspect to be considered when attempting to improve the well-being of users and energy efficiency in Smart Buildings. Nowadays the market of HVAC systems is dominated by a number of traditional manufacturers offering proprietary hardware and software solutions. For such systems, the integration with open IoT platforms is still far from being a reality.

Buildings, such as the ITSI, typically include a centralized HVAC system with a proprietary configuration software. From the end user point of view, the only interaction is with a simple console in each room, with buttons that allows switching on and off the heating and changing the temperature set-point. In the hardware prototype developed in the Smart Office, the console has been retrofitted adding an IoT node with WiFi connectivity. This node is able to interact with the console by means of optocouplers acting as solid-state relays that allow bypassing the button contact. In this way, the console can be operated both manually (i.e. pressing the buttons) and by means of the IoT node outputs, that electrically emulate the action of pressing the buttons. Retrofitting allows the integration of existing products with IoT platforms Medina and Manera (2017). Nevertheless, this solutions should also pursue the compliance with existing safety and quality regulations.

The WiFi node, implemented for temperature control of the Smart Office environment uses a low-cost NodeMCU

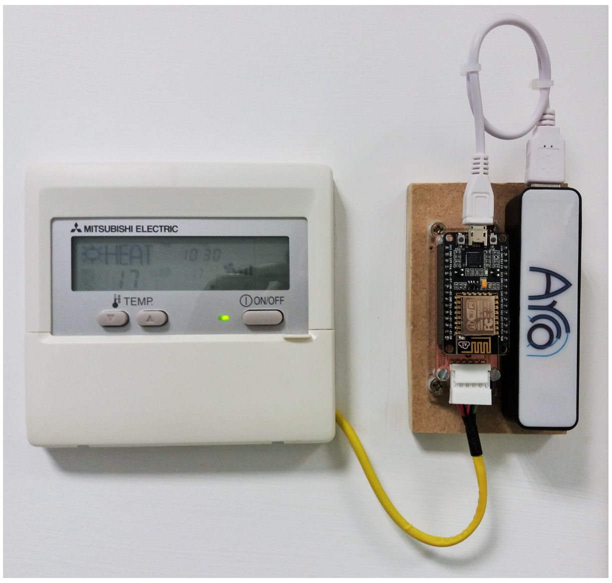

Figure 8. IoT node for temperature control

device[‖], which is based on the compact and low-power Espressif ESP8266 WiFi and MCU chipset [**]. To interface the node with the HVAC console a circuit has been designed to integrate the required optocouplers. An image of the temperature control node connected the HVAC console is shown in Figure 8. A detail of the NodeMCU and the designed shield that incorporates the optocouplers can be seen in Figure 9.

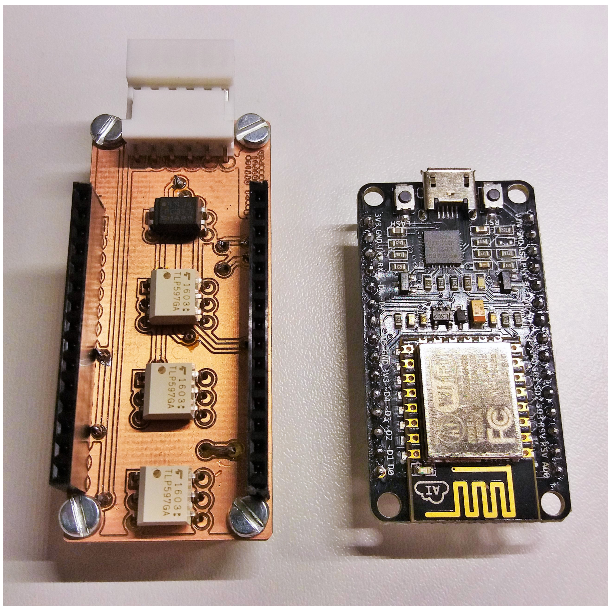

Figure 9. Detail of the NodeMCU device and the designed shield to interface the HVAC console

The integration of the temperature control system in the proposed IoT platform for Smart Homes enhances the possibilities in terms of comfort and efficiency management. The designed node is able to control the on-off, temperature increase and temperature decrease buttons. Moreover, it is able to monitor the state of the heating system (on-off) that is indicated with a green LED in the console. One of the advantages of the developed temperature control node is that, thanks to the integration with the IoT platform proposed in this paper, it can be linked to one or various temperature sensors distributed in the space. Furthermore, the temperature set-point can be established by several users. These two features enable the possibility of advanced control strategies, in which the feedback temperature signal and the set-point values can be obtained as by performing some processing and calculations considering the measurements of several sensors and the comfort of several users. An example to illustrate this could be a scenario in which:

- The feedback temperature is obtained by calculating the mean temperature of the sensors located in areas where the presence of user is detected.
- The set-point is obtained calculating the mode (i.e. the value that appears more often) among the values configured by the users.

Another valuable advantage of the proposed temperature control system, is the ease to perform data logging for pattern extraction and energy estimation.

Additionally, to talk about smart building it is essential to automatically control and manage the people that access or leave the premises. In fact, many of the services provided in a smart building depend on knowing how many people are there inside the building and where are they located in (i.e.: smart evacuation protocols, smart lighting services, smart meeting management, etc.). One of the most common solution is based on the use of RFID readers, deployed at the entrance of the different rooms to be controlled. Every user has a badge that grants or denies access to every room. Figure 10 shows a RFID reader at the entrance of the testbed office.

Similarly, the use of CCTV (close-circuit television) cameras is commonly extended for surveillance purposes. We have also equipped our testbed with a low-cost camera FOSCAM C1.

To validate our capability to seamlessly communicate objects employing different communication technologies several prototypes have been specifically designed to this end. These prototypes, as the one shown in Figure 12, therefore demonstrate the IDM capability to homogenize IoT objects. Despite their heterogeneous communication and architecture details, these objects are considered virtually equal inside an IDM infrastructure. Moreover, the door lock has been retrofitted with an electric door lock, as it can be observed in Figure 11 labeled as *door actuator*. To turn the door actuator into an IoT node we have employed a commercial product known as Sonoff[††]. These devices are basically WiFi switches controlling electric loads (with a 10 A capability according to the specifications) that internally has an ESP8266EX chip, all for a price of approximately 5 dollars.

Finally, the testbed area has also been equipped with some additional sensors (presence, temperature, microphone, light, etc.). Whereas the HVAC console, camera, microphone, and

[‖] http://www.nodemcu.com/index_en.html
[**] http://espressif.com/en/products/hardware/esp8266ex/overview
[††] https://www.itead.cc/smart-home/sonoff-wifi-wireless-switch.html

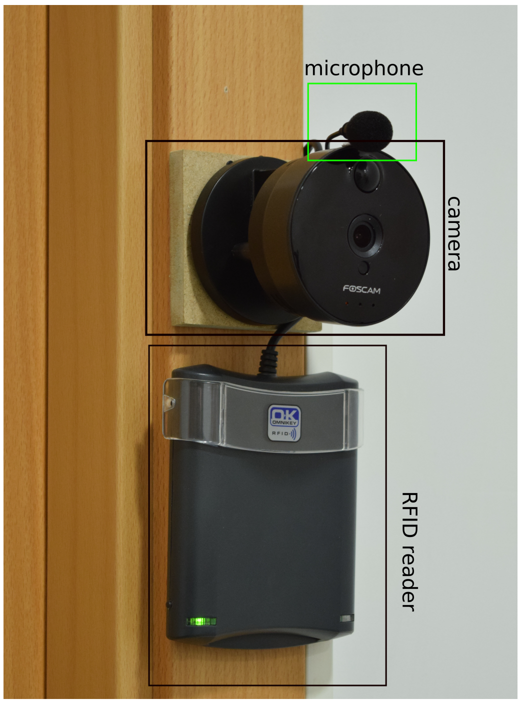

Figure 10. Camera, microphone, and RFID card reader deployed at the door frame

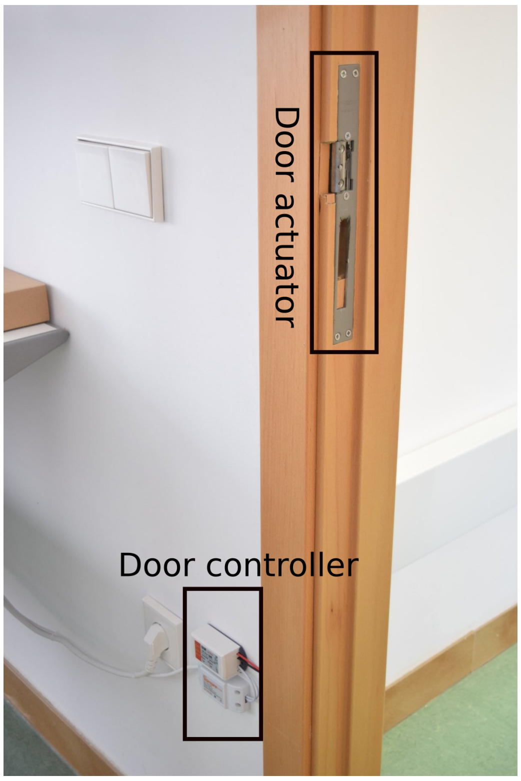

Figure 11. Door configuration

door actuator employ WiFi, we have built a sensor board prototype, as shown in Figure 12 that includes, among some, a presence and a temperature sensor over a ZigBee (Arduino FIO + XBee) and RS-485 domains. Every domain has its own IDM router. The IDM routers for the ZigBee and RS-485 domains run on a Raspberry Pi whereas the router for the WiFi domain runs on a conventional PC. Figure 13 outlines the considered topology.

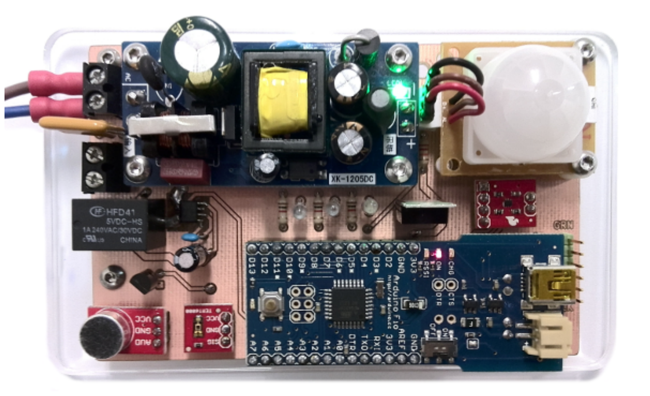
Figure 12. Moth XBee including a presence and a temperature sensor

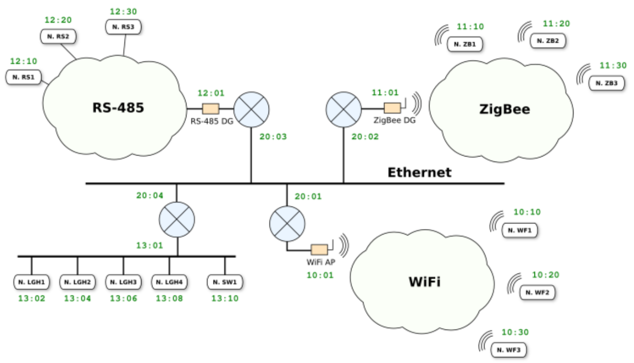

Figure 13. Logic topology of the considered scenario

Experimental results

Both case scenarios described at the beginning of this section have been reproduced in the testbed environment (the ITSI building), using the hardware and network topology previously described. Recall that each of the considered scenarios has targeted a different goal: case scenario 1 pursues the validation of the composition capabilities, whereas case scenario 2 is intended to demonstrate its capability to manually and automatically reconfigure services.

Figure 14 summarizes the scheduler trace yielded in the process of building the service pipeline to cater for the user's issued command. The event `e: open door`, at the top of the hierarchy, is provided by the awareness layer (the BDI agent) to the scheduler. This event therefore determines the start of the scheduling process. The knowledge base is queried about the action or actions capable of causing that event. It answers that the `a: actuate bolt` action is capable of doing so. Then, the knowledge base is also queried about the service or services (if any) capable of performing that action. In this case, the `s: door actuator` service has that capability. However, that action has in his before context the *prerequisite* of having authenticated the identity of the command issuer. This prerequisite is stated as an event in the before context of the action description.

Figure 14 encloses in frames each of the scheduler iterations. For this particular case, the scheduler is launched five times, one for each event that has to be caused. The result

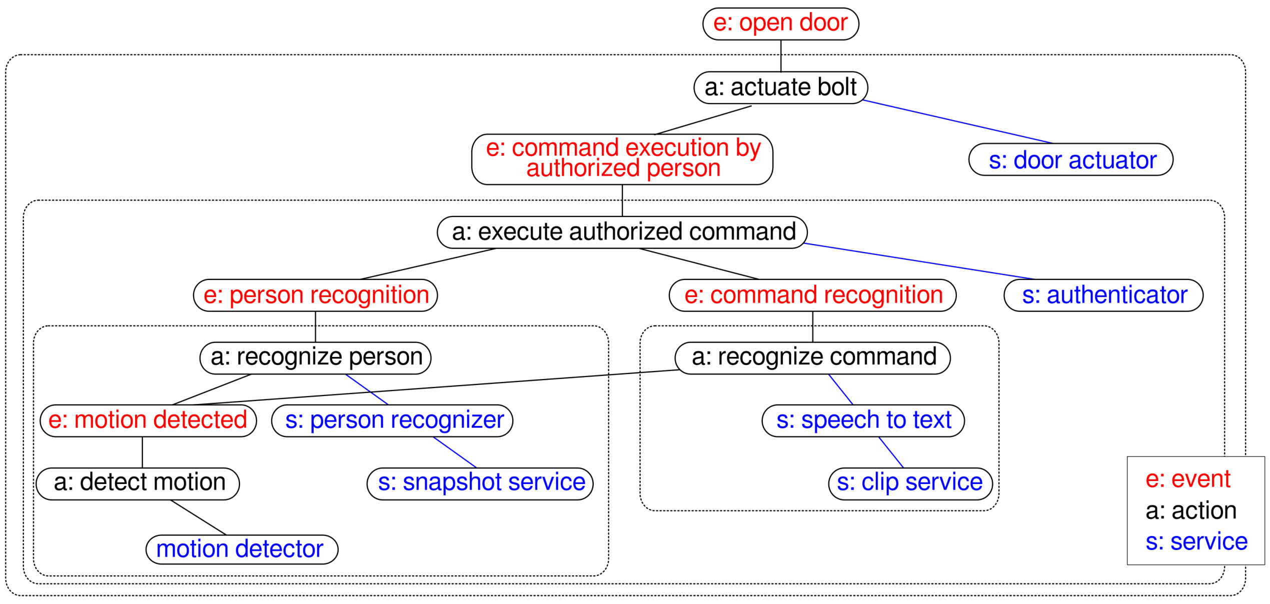

Figure 14. Scheduler result for scenario 1

of the scheduling process is the list (or pipeline) of services that has to be bound.

Figure 15 depicts the sequence diagram that results from binding the service pipeline generated by the scheduler.

The service reconfiguration process is simpler than the composition one mainly due to the fact that the scheduler is provided with events for which there exits a single service that causes it. There is no need to recursively look for alternatives, as in the composition case. Figure 16 summarizes the scheduler trace. As it can be observed, it is simpler than the one for case scenario 1.

The sequence diagram from case scenario 2 is depicted in Figure 17. As the user enters the room, the system looks for services that can provide a measure of the room temperature. Based on that temperature and knowing the comfort temperature established for that user, the *thermostat* service calculates the increase or decrease (in degrees) that should be issued to the HVAC actuator system.

The scheduler is queried about how to cause the event of `e: set to comfort temperature`. This event is caused through the action `a: operate console to comfort temperature` that is carried out by the `s: HVAC system actuator` which, at the same time requires from the service `s: thermostat`.

Conclusions

This paper proposes an architecture for IoT-based Smart Home that focuses in enabling capabilities for automatic service composition and reconfiguration. This architecture is novel in the three-stage process it proposes for the service composition and reconfiguration. An additional contribution of this work consists in how semantic is treated. A common-sense reasoning approach is proposed to capture the semantics of IoT objects and services. This semantic knowledge has demonstrated more flexible and advanced capabilities for the composition and reconfiguration process, to eventually cater for unsatisfied goals or arisen needs. This capability is what turns a normal environment into a *smart* one. The three-stage process for service composition and reconfiguration assures that the process can be carried out in an unsupervised manner. This aspects in essential if actuation and smart behavior can be demonstrated in this type of environments.

To demonstrate the performance of the proposed architecture a testbed scenario has been set up. Different rooms of a working building have been retrofitted with low-cost devices to turn them into IoT objects such as doors or HVAC systems. Two case scenarios have been devised to evaluate the response capabilities of a smart system. Results yield that this approach supposes a low-cost and flexible mechanism for turning homes into smart homes.

Acknowledgements

This work has been partly funded by the Spanish Ministry of Economy and Competitiveness under project PLATINO (TEC2017-86722-C04-04-R), the Univesity of Castilla-La Mancha R&D Plan under the access contracts to the Spanish system of science, technology and innovation call, which is partially funded by the European Social Fund (31/07/2014 Resolution, published in the DOCM on the 25 of August 2014) and by the Spanish Ministry of Economy and Competitiveness under project CitiSim Itea3 (TSI-102107-2016-4).

References

Alaya MB, Medjiah S, Monteil T and Drira K (2015) Toward semantic interoperability in onem2m architecture. *IEEE Communications Magazine* 53(12): 35–41.

Alliance A (2016) Alljoyn framework. *Linux Foundation Collaborative Projects. URl: https://allseenalliance. org/framework (visited on 09/14/2016)* .

Alliance OM (2015) Lightweight machine to machine technical specification. *Candidate Version* 1: 14.

Bratman M (1987) Intention, plans, and practical reason .

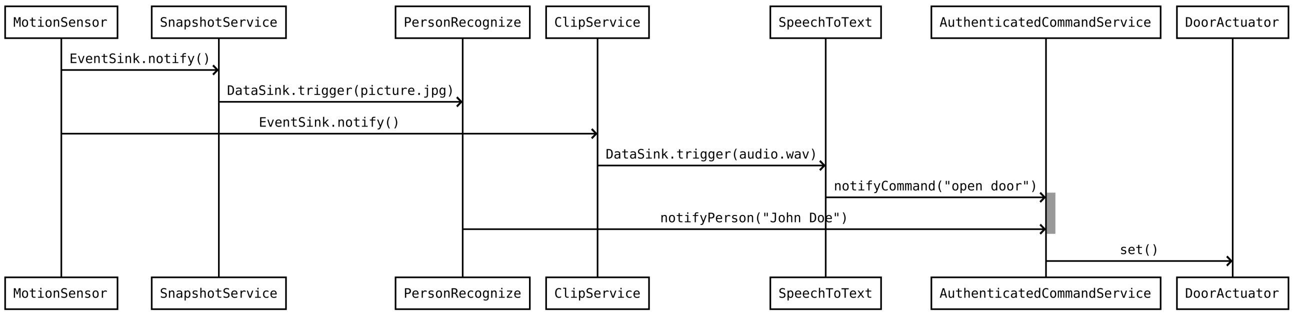

Figure 15. Sequence diagram of the service pipeline for scenario 1

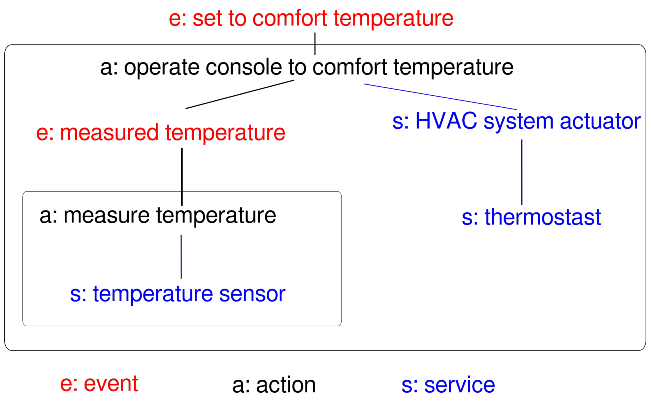

Figure 16. Scheduler result for scenario 2

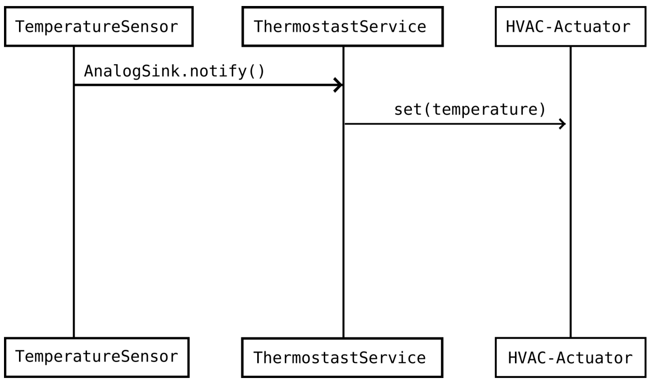

Figure 17. Sequence diagram of the service pipeline for scenario 2

Buffers P (2011) Googles data interchange format.

Chen W (2009) Understanding mental states in natural language. In: *Proceedings of the Eighth International Conference on Computational Semantics*. Association for Computational Linguistics, pp. 61–72.

Chen W and Fahlman SE (2008) Modeling mental contexts and their interactions. In: *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, volume 8. p. 04.

Christensen E, Curbera F, Meredith G and Weerawarana S (2001) Web services description language (wsdl) 1.1 .

Cicirelli F, Guerrieri A, Spezzano G and Vinci A (2017) An edge-based platform for dynamic smart city applications. *Future Generation Computer Systems* 76(Supplement C): 106 – 118. DOI:https://doi.org/10.1016/j.future.2017.05.034. URL http://www.sciencedirect.com/science/article/pii/S0167739X16308342.

Cleary A, Kohn S, Smith SG and Smolinski B (1998) Language interoperability mechanisms for high-performance scientific applications. Technical report, Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

Compton M, Barnaghi P, Bermudez L, Garca-Castro R, Corcho O, Cox S, Graybeal J, Hauswirth M, Henson C, Herzog A, Huang V, Janowicz K, Kelsey WD, Phuoc DL, Lefort L, Leggieri M, Neuhaus H, Nikolov A, Page K, Passant A, Sheth A and Taylor K (2012) The ssn ontology of the w3c semantic sensor network incubator group. *Web Semantics: Science, Services and Agents on the World Wide Web* 17(Supplement C): 25 – 32. DOI:https://doi.org/10.1016/j.websem.2012.05.003. URL http://www.sciencedirect.com/science/article/pii/S1570826812000571.

Datta SK and Bonnet C (2015) A lightweight framework for efficient m2m device management in onem2m architecture. In: *Recent Advances in Internet of Things (RIoT), 2015 International Conference on*. IEEE, pp. 1–6.

Datta SK, Gyrard A, Bonnet C and Boudaoud K (2015) onem2m architecture based user centric iot application development. In: *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*. IEEE, pp. 100–107.

Davidson D (1963) Actions, reasons, and causes. *The journal of philosophy* 60(23): 685–700.

Fahlman SE (2006) Marker-passing inference in the scone knowledge-base system. *KSEM* 4092: 114–126.

Fahlman SE (2011) Using scone's multiple-context mechanism to emulate human-like reasoning. In: *AAAI Fall Symposium: Advances in Cognitive Systems*, volume 11. p. 01.

Fielding R, Gettys J, Mogul J, Frystyk H, Masinter L, Leach P and Berners-Lee T (1999) Hypertext transfer protocol–http/1.1. Technical report.

Hui TK, Sherratt RS and Snchez DD (2017) Major requirements for building smart homes in smart cities based on internet of things technologies. *Future Generation Computer Systems* 76(Supplement C): 358 – 369. DOI:https://doi.org/10.1016/j.future.2016.10.026. URL http://www.sciencedirect.com/science/article/pii/S0167739X16304721.

Hunkeler U, Truong HL and Stanford-Clark A (2008) Mqtt-sa publish/subscribe protocol for wireless sensor networks. In: *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*. IEEE, pp. 791–798.

Kent Bach (1980) Actions are not Events. *Mind* 89(353): 114–120. URL http://online.sfsu.edu/kbach/oldies/actionsnotevents.pdf.

Lalanda P, Gerber-Gaillard E and Chollet S (2017) Self-aware context in smart home pervasive platforms. In: *2017 IEEE International Conference on Autonomic Computing (ICAC)*. pp. 119–124. DOI:10.1109/ICAC.2017.1.

Mccarthy J (1973) Ascribing mental qualities to machines. pp. 161–195.

Medina BE and Manera LT (2017) Retrofit of air conditioning systems through an Wireless Sensor and Actuator Network: An IoT-based application for smart buildings. In: *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*. ISBN 978-1-5090-4429-0, pp. 49–53. DOI:10.1109/ICNSC.2017.8000066. URL http://ieeexplore.ieee.org/document/8000066/.

Mulligan G (2007) The 6lowpan architecture. In: *Proceedings of the 4th workshop on Embedded networked sensors*. ACM, pp. 78–82.

Nunes DS, Zhang P and Silva JS (2015) A survey on human-in-the-loop applications towards an internet of all. *IEEE Communications Surveys & Tutorials* 17(2): 944–965.

Rahman H, Rahmani R and Kanter T (2017) Multi-modal context-aware reasoner (can) at the edge of iot. *Procedia Computer Science* 109(Supplement C): 335 – 342. DOI:https://doi.org/10.1016/j.procs.2017.05.360. URL http://www.sciencedirect.com/science/article/pii/S1877050917310293. 8th International Conference on Ambient Systems, Networks and Technologies, ANT-2017 and the 7th International Conference on Sustainable Energy Information Technology, SEIT 2017, 16-19 May 2017, Madeira, Portugal.

Shelby Z and Chauvenet C (2012) The ipso application framework draft-ipso-app-framework-04. *Avaiable online: http://www. ipso-alliance. org/wp-content/media/draft-ipso-app-framework-04. pdf (accessed on 3 June 2014)* .

Shelby Z, Hartke K and Bormann C (2014) The constrained application protocol (coap) .

Smirek L, Zimmermann G and Beigl M (2016) Just a smart home or your smart home   a framework for personalized user interfaces based on eclipse smart home and universal remote console. *Procedia Computer Science* 98(Supplement C): 107 – 116. DOI:https://doi.org/10.1016/j.procs.2016.09.018. URL http://www.sciencedirect.com/science/article/pii/S1877050916321391. The 7th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN 2016)/The 6th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2016)/Affiliated Workshops.

Tao M, Zuo J, Liu Z, Castiglione A and Palmieri F (2018) Multi-layer cloud architectural model and ontology-based security service framework for iot-based smart homes. *Future Generation Computer Systems* 78(Part 3): 1040 – 1051. DOI:https://doi.org/10.1016/j.future.2016.11.011. URL http://www.sciencedirect.com/science/article/pii/S0167739X16305775.

Tayur VM and Suchithra R (2017) Review of interoperability approaches in application layer of Internet of Things. In: *2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*. IEEE. ISBN 978-1-5090-5960-7, pp. 322–326. DOI:10.1109/ICIMIA.2017.7975628. URL http://ieeexplore.ieee.org/document/7975628/.

Villa D, Aceña O, Villanueva F, Santofimia MJ, Escolar S, del Toro Garcia X and Lopez JC (2017) Idm: An inter-domain messaging protocol for iot. In: *Industrial Electronics Society, IECON 2017-43rd Annual Conference of the IEEE*. IEEE, pp. – .

Xuemei L and Gang X (2008) Service oriented framework for modern home appliances. In: *2008 ISECS International Colloquium on Computing, Communication, Control, and Management*, volume 1. pp. 700–703. DOI:10.1109/CCCM.2008.386.