A Semi-Automatic Approach for Semantic IoT Service Composition

Grigorios Tzortzis
Institute of Informatics and Telecommunications
NCSR Demokritos
Athens, Greece
gtzortzi@iit.demokritos.gr

Evaggelos Spyrou
Institute of Informatics and Telecommunications
NCSR Demokritos
Athens, Greece
espyrou@iit.demokritos.gr

ABSTRACT

The Internet of Things (IoT) and its service-oriented architecture (SoA) has increased the interest on the concept of service composition. Simpler (modular) web services are interconnected to realize more complex ones. To this end, we propose a semi-automatic approach, tailored to suit the needs of developers, allowing them to easily discover, consume and interconnect services so as to create more complex ones by exploiting their semantics. We build upon our previous work on the SYNAISTHISI platform and an ontology for smart meeting rooms. To demonstrate the workflow of our approach, we present a real-world case; we create a complex application for counting the persons within a smart meeting room, by interconnecting simpler, IoT-enabled services.

1. INTRODUCTION

During the last few years, everyday physical objects have been modified with the embodiment of short-range and energy efficient mobile transceivers and enhanced with unique identifiers. The extensive networking of heterogeneous devices has led to the emergence of the Internet of Things (IoT), which is considered as the next industrial revolution [5] and is expected to find numerous applications in diverse areas. IoT adopts a service-oriented architecture (SoA), where all "things" are exposed as web services that can be used and reused [2]. These IoT-related services can be categorized into three distinctive, yet interdependent, types: *(i) sensing* services that capture properties of the physical world and provide raw or slightly processed measurements, *(ii) processing* services that process the acquired measurements and provide the inferred results and *(iii) actuating* services that enable certain actions based on these results.

Available services in an IoT ecosystem can be combined to construct complex applications that fulfill e.g., some new desired functionality which none of the existing services is able to provide. This procedure is known as "service composition" [15]. Key steps in composing a service are the *discovery* of suitable services and their appropriate *interconnection* using an IoT-ready platform, such that a functional composition is ensured where every service can be readily invoked (i.e., each service's inputs and preconditions are satisfied from the outputs and the effects of other services participating in the composition).

In our previous work [1], we described a methodology of developing complex applications using the SYNAISTHISI platform [12]. The core of this work was a smart meeting room ontology, providing semantic annotations for the available services in the room. We showed that a developer with knowledge of *(i)* what kind of services are required for the complex application and *(ii)* how they should be interconnected, is able to discover the appropriate services by utilizing the semantic information (via SPARQL[1] queries) and define an interconnection scheme, thus manually compose a service that is subsequently realized by the platform. In this work we extend the aforementioned ontology and present a *semantically-aware* procedure that allows developers to compose services in *semi-automatic* fashion, alleviating the assumptions made in [1].

As a test case to illustrate our approach, we revisit the composed "person counting" service of [1], using the proposed semi-automatic methodology. We show that the whole procedure becomes now more intuitive. The developer still does not need to have prior knowledge regarding the available services' implementation details (programming languages, technologies, etc.) and is also safeguarded against taking wrong decisions, i.e. interconnecting services whose inputs–outputs do not match.

The rest of this paper is organized as follows: In Section 2 we provide an overview of related work in service composition and web service ontologies. Then, in Section 3 we briefly present the SYNAISTHISI platform. In Section 4 we present the smart meeting room ontology, while in Section 5 we describe the steps that must be performed to compose services and build a complex application. In Section 6 we present a use case of our approach. Discussion follows in Section 7, along with possible directions for future work.

2. RELATED WORK

Semantics provide a means to enhance the descriptions of web services and facilitate various IoT-related tasks, such as service discovery and composition. Several ontologies exist for attaching semantic content to web services, converting them into *semantic web services* [9]. OWL-S [8] and WSMO [14] are two renowned efforts, which provide highly expressive models for annotating web services. The Minimal Service Model (MSM) [11] focuses only on the core semantics of web services, trading the expressivity of OWL-S and WSMO to improve usability. The aforementioned ontologies constitute general purpose upper ontologies that include high-level concepts which can be applied across heterogeneous domains. However, specializing these ontologies to also model the low-level concepts specific to a particular domain demands considerable effort. To overcome this difficulty, web service ontologies that are domain-specific have

[1]http://www.w3.org/TR/sparql11-query

been also developed. The smart building ontology in [17], as well as the smart meeting room ontology we presented in [1] and further extend herein, are two such examples.

Web services may be seen as self-contained units of functionality. Service-oriented architectures (SoA) allow them to be published, discovered and consumed [15]. Apart from using the available services, users are also able to compose them, so as to realize more powerful services that fulfill their needs more effectively. Typically, service composition is accomplished using one of the following approaches [15]:

i. *Manual composition*: the user has total freedom at every step of the process. However, this advantage is compensated by the need of too much knowledge and effort on the part of the user.
ii. *Semi-automatic (interactive) composition* [3, 7]: at every step of the process the system assists the user by presenting him/her with possible choices, however decisions are ultimately taken by the user.
iii. *Automatic composition* [6, 13]: the user typically sets constraints and/or preferences, but is not allowed to intervene at the process. Such methodologies are not demanding nor require knowledge on the part of the user, but may often lead to results that do not match the user's original intentions.

A survey on recent works regarding web service composition may be found in [15].

3. THE SYNAISTHISI PLATFORM

The work presented herein has been developed within the context of the SYNAISTHISI project[2], whose goal was to deliver energy efficient, secure and effective applications, as well as services, to end users, aiming to minimize the environmental impact, monetary costs, user discomfort, delays and utilization of resources. In this section we shall provide a brief description of the SYNAISTHISI platform [12], an IoT-ready platform, and outline how it enables the integration, interconnection and coordination of a large number of heterogeneous devices and algorithms which are exposed as semantic web services. The platform architecture is discussed in detail in our previous work [12]. In brief, the platform supports communication channels which realize service interconnection, controls the available resources, permits the deployment of custom applications and provides a set of tools to accommodate system administrators.

The available services are registered into the service registry, implemented by an RDF triplestore equipped with a semantic reasoner, and following the IoT paradigm discussed in Section 1, they are divided into three distinct categories: *(i)* S-type services corresponding to sensors that sense the physical world, *(ii)* P-type services corresponding to processors (i.e. algorithms) that process the measurements of the S-type services and/or the processed results of other P-type services and *(iii)* A-type services corresponding to actuators that are used for the actuation of devices/signals based on the acquired results. Seen from the services' IOPEs (Inputs, Outputs, Preconditions, Effects) perspective, S-type services produce outputs, but do not have inputs (strictly speaking, they may only receive trigger signals), P-type services have both inputs and outputs, while A-type services have inputs and their output (actuation), is actually an *effect*. We will refer to the set of these services as "SPA services" hereafter.

[2]http://iot.synaisthisi.iit.demokritos.gr

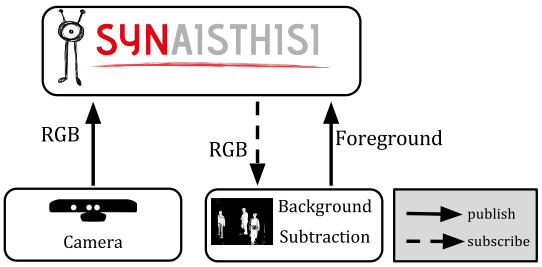

Figure 1: Two services interconnected with the SYNAISTHISI platform exchange messages. Solid arrows correspond to messages published to topics, while dashed arrows depict the messages arriving to the subscribers of those topics.

Within the platform, services communicate by publishing messages and/or subscribing to topics managed by a Message-oriented Middleware (MoM), which is a message broker. Thus, all sensor measurements, processor results, and actuations are encapsulated into messages and communicated via these topics. Once a message is published to a topic, the MoM informs and delivers it to all clients that are subscribed to that topic. An example is depicted in Fig. 1. A camera is an S-type service which captures RGB video data from the physical world and publishes them to a topic. A Background Subtraction module is a P-type service, which receives these data by subscribing at the same topic. Upon processing, it publishes its output at another topic.

4. SMART MEETING ROOM ONTOLOGY

To enhance the efficacy of service discovery and composition towards realizing complex applications, the SPA services of the SYNAISTHISI platform are *semantically enriched* using ontologies. Specifically, we have developed a *domain-specific* ontology that models the concepts related to smart meeting rooms, where all sensors, processors and actuators are exposed as web services.

Our semantic model imports existing ontologies and reuses their knowledge. The Internet of Things Architecture (IoT-A) ontology [4], includes high-level concepts for describing key aspects of the IoT domain and forms its basis. Also, the popular Semantic Sensor Network (SSN) ontology[3] is utilized to represent various features of sensors and actuators (e.g. the measurement capabilities of a sensor), while the QU[4] and QUDT[5] ontologies provide descriptions for physical quantities (e.g. temperature) and their measuring units.

On the following, we outline the main classes and properties that make up the proposed ontology, omitting secondary elements to avoid cluttering the presentation. A preliminary version of the ontology can be found in our previous work [1], however it is here refined and extended with additional concepts, to obtain a comprehensive description of smart meeting room entities and facilitate the execution of service composition tasks.

The *resource* and *service model* of the IoT-A ontology are specialized by introducing classes, subclasses and properties related to the smart meeting room domain. In general, the resource model is responsible for *describing the char-*

[3]http://purl.oclc.org/NET/ssnx/ssn
[4]http://purl.oclc.org/NET/ssnx/qu/qu
[5]http://qudt.org/

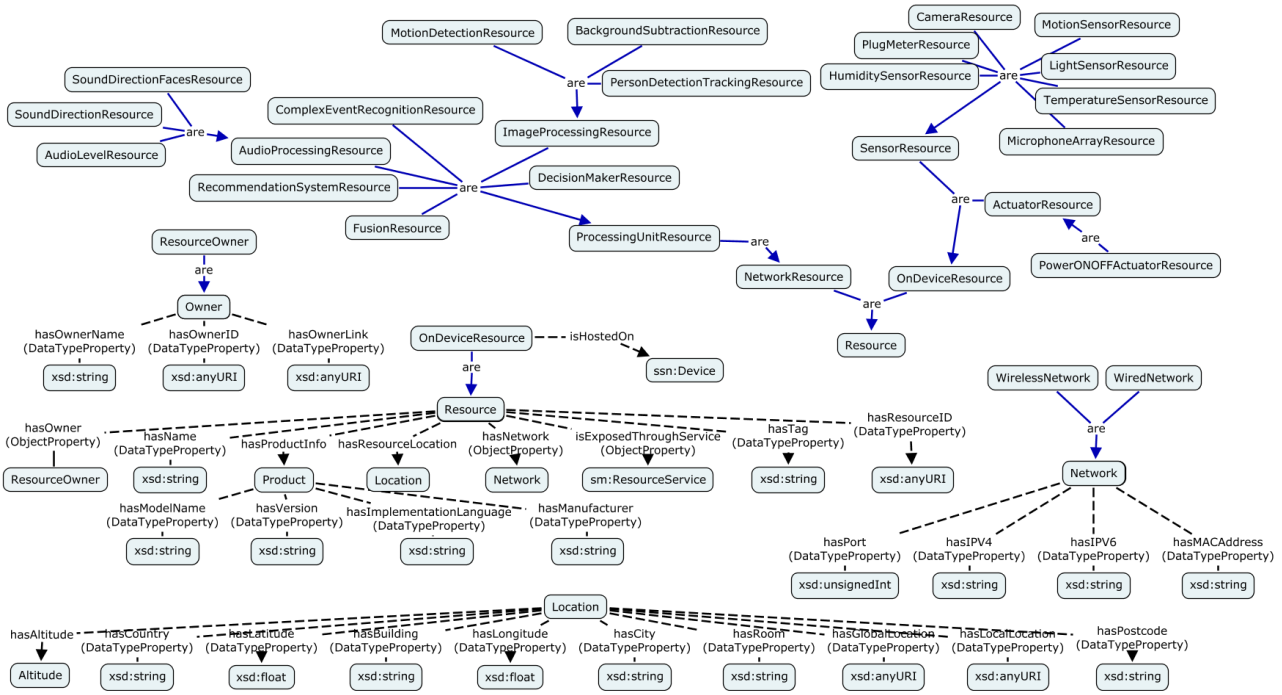

Figure 2: Core classes and properties of the resource model. The `sm` and `ssn` namespaces refer to the service model and the SSN ontology, respectively.

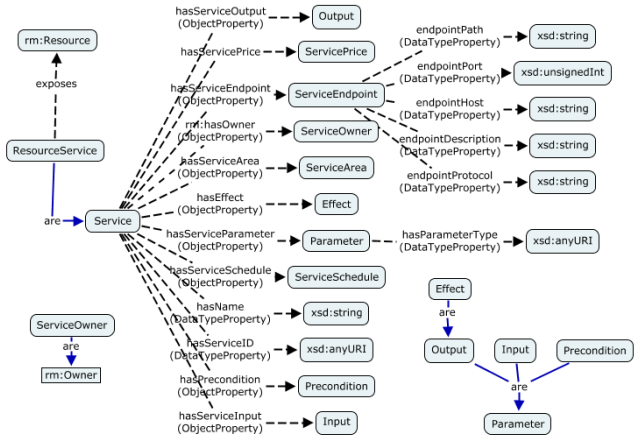

Figure 3: Core classes and properties of the service model. The `rm` namespace refers to the resource model. The hierarchy of service types is not shown, as it has a one-to-one correspondence to the resource type hierarchy.

acteristics of the actual device (i.e., sensor or actuator) *or processor*, referred collectively as "resources" hereafter, that is hidden behind a web service, while the service model is responsible for *describing the characteristics of the web service* (e.g. service input/output, service endpoint etc.) that is used to access a resource and expose its functionality to the outer world. Note that in the context of our work, the notion of SPA services encapsulates both resource and service model concepts.

4.1 Resource Model

The resource model, shown in Fig. 2, consists of a core class, namely `Resource`, that captures the notion of a resource. A hierarchy is defined to represent various types of resources that may exist in a smart meeting room, many of which were not part of the ontology of [1]. The hierarchy includes specific types of sensors, e.g., temperature sensors, cameras and microphones, processors, e.g. modules for subtracting the background of a scene, and actuators, e.g. plug switches. Class names are quite indicative of the resources' functionality. A resource is equipped with properties specifying its name (`hasName`), an ID (`hasResourceID`), some keywords describing the resource (`hasTag`) and, importantly, the web service that exposes its functionality to the outer world (`isExposedThroughService`). The `Location` class defines the location of a resource, while the `Network` class is used to describe the network interface which makes the resource accessible through the web. Product-related information, such as the manufacturer name, can be provided using the `Product` concept. The ontology also allows to associate a resource with its owner (`ResourceOwner`). Finally, for sensing and actuating devices the SSN ontology concepts can be exploited using the link to the SSN `Device` class.

4.2 Service Model

The service model, depicted in Fig. 3, consists of a core class, namely `Service`, that captures the notion of a web service and contains the necessary information for discovering and invoking the service. Various service types are defined in a hierarchy, corresponding to the aforementioned resource types. A service has a name (`hasName`), an ID (`hasServiceID`), an owner (`ServiceOwner`), a link to the resource that is accessed using the service (`exposes`) and an endpoint from where client applications can access the service (`ServiceEndpoint`). The (physical) area that is affected when the service is invoked can be determined via the `hasServiceArea` property. This property is particularly useful for S-type services, declaring the area that is observed by the sensor, and for A-type services, declaring the area that

is affected by an actuation. Possible time constraints on the availability of a service can be defined using the `ServiceSchedule` class. A very important element in the semantic description of a service is to model its IOPEs. In our case this is accomplished through the subclasses of the `Parameter` class and by using the `hasParameterType` property to annotate the IOPEs by providing the URI of the concepts, defined in some appropriate ontology, that capture their meaning. Note that effects are considered as a special type of output intended to describe the actuation of A-type services. Finally, it is possible to declare a price for trading a service in a service marketplace (`ServicePrice`).

5. SERVICE COMPOSITION

The manual service composition approach originally adopted in the SYNAISTHISI platform [1] imposes some rather stringent assumptions that require a considerable amount of effort on the developer's side, who must himself decide what SPA services are needed, write SPARQL queries to discover them and define their interconnection. Hence, high-level skills and a priori knowledge of the IoT ecosystem he operates in are necessary. Here, we transfer part of this effort to the platform that will guide the developer in building a complex application, by proposing a *semi-automatic service composition* method which makes extensive use of IOPE-related semantics. While presenting the method we shall resort to the smart meeting room ontology (Section 4) to obtain the semantic descriptions of SPA services, but as it will become evident our approach is general, in the sense that it can be applied over any SPA services ontology, if SPARQL queries are appropriately adapted. Preconditions of services are not considered by our framework and effects are treated as a special case (i.e. subclass) of output produced by A-type services, therefore the discussion below is oriented around the inputs and outputs of services.

5.1 Service Composition Algorithm

Initially, the developer must specify a *service request* describing the desired functionality of the composite service in terms of outputs that should be generated when executed. Each output is declared using a suitable concept from an ontology that captures its meaning. For each concept in the service request the platform employs a distinct service discovery procedure that searches over the SPA services available in the platform to locate those that produce an output (output concept) that "matches", hence satisfies, the particular service request concept and organizes them in a *matching services' list* (the details of when two concepts match are explained in Section 5.2). Outputs (and inputs) of SPA services are retrieved by issuing platform-generated SPARQL queries conforming to the service model of the smart meeting room ontology, to extract the value of the `hasParameterType` property associated with the `Input/Output` class. Services included in a matching list represent alternative choices for satisfying a concept of the service request and are presented to the developer who must select one to be incorporated into the composite service. Apparently, a selection must be made for each service request concept.

After selecting a service, we must ensure that the service can be invoked, i.e. all the inputs of the service can be supplied with appropriate data, otherwise the composite service will not execute. This is accomplished by retrieving the inputs (input concepts) of the service (via SPARQL queries generated by the platform) and launching a separate service discovery procedure for each input to recover those SPA services available in the platform that produce an output (output concept) that matches the input, hence they provide suitable data for the input. Subsequently, for each input, the developer is prompted to choose a service from the corresponding matching list that will be added to the composite service and a new round of service discovery and selection is initialized. This process is repeated until there are no more SPA services participating in the composite service which cannot yet be invoked. Note that S-type services can be readily invoked since they do not have inputs, thus there is no need to carry out service discovery if such a service is chosen. By progressively selecting S-type services it is possible to reach a state, where all services are invokable and service composition terminates. After termination, the platform proceeds with realizing and executing the composite service. Note that if, at any point, the service discovery procedure for an input returns an empty matching list, the composition process fails and no composite service is created, since all available SPA services are deemed unsuitable.

Algorithm 1 Service Composition

Input: Dummy service SR, representing the service request
Output: Composite service CS, or fail

```
1: Set Q = ∅ // Queue structure with services as elements.
2: ENQUEUE(Q, SR)
3: repeat
4:     S = DEQUEUE(Q)
5:     for each input I_S of service S do
6:         L = SERVICE_DISCOVERY(I_S) // Matching services list.
7:         if L == ∅ then // If the list is empty.
8:             return fail
9:         end if
10:        Prompt the developer to select a service S_L from L
11:        Add S_L to CS to supply the data for input I_S
12:        if S_L not an S-type service then
13:            ENQUEUE(Q, S_L)
14:        end if
15:    end for
16: until Q == ∅ // Until all services can be invoked.
17: return CS
```

The service composition process is summarized in Algorithm 1, where the service request is represented as a dummy service without outputs, having the service request concepts as inputs. A graphic example is depicted in Fig. 4.

5.2 Semantic Relaxation

The service discovery procedure utilizes semantically relevant concepts to decide whether the output of a service matches a particular input of another service, thus include it in the list of matching services (presented to the developer) that can provide this input. An output concept `A` is considered to be semantically relevant (i.e. matched) to an input concept `B` when one of the following *hierarchical relationships* exists between them: *(i)* `exact(A,B)` when the two concepts have the same URI or are equivalent in terms of OWL equivalence, *(ii)* `plugin(A,B)` when `A` is subsumed by (is a subclass of) `B`, and *(iii)* `subsume(A,B)` when `A` subsumes (is a superclass of) `B`. These hierarchical relationships represent different degrees of *semantic relaxation* (in increasing order) and are typical in the service composition literature [10]. The `exact` relationship implies that two concepts are the same and can be used interchangeably during composition,

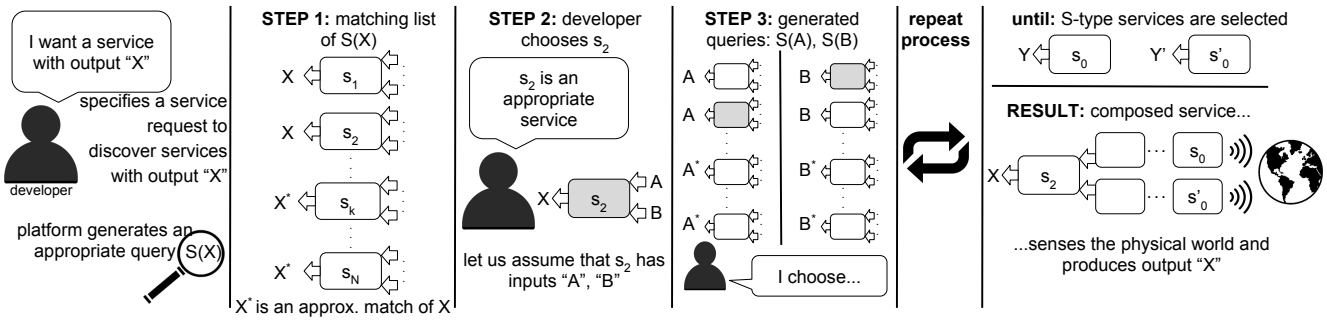

Figure 4: The process of semi-automatic service composition.

thus avoiding syntactic limitations. `Plugin` and `subsume` imply an approximate relationship between two concepts and can be utilized to approximately match inputs to outputs, a useful feature, especially, when exact matches do not exist in all steps of the composition process that, otherwise, would lead to a failure in producing the requested service. In general, we anticipate exact matches to be preferred over plugin matches and plugin matches over subsume[6] matches, hence the *services in the matching list are ordered accordingly.* Hierarchical relationships are calculated by applying a semantic reasoner over the ontologies defining the concepts used to annotate the IOPEs of the SPA services (not the smart meeting room ontology).

6. EXAMPLE USE CASE

In this section we present a use case of the aforementioned service composition methodology, by revisiting the example of [1], where a people counting service was manually composed using simpler ones. We assume that the goal of the developer is to compose a "Person Counting" service, i.e. a service able to provide an estimation of the number of people present within a smart meeting room by utilizing the two cameras overlooking the room. Moreover, we assume that the developer is familiar with basic computer vision tasks, thus is able to understand the flow of services suggested by the system at each step and proceed to meaningful choices.

In Fig. 5 we illustrate the steps of the service composition process. The process begins with the developer's intention to create a service with a specific functionality, i.e. he knows the service's output. He expresses this output via a service request and two services are returned in the matching list. Even though the output of the second service, namely `person_fusion`, is not an exact match, but a plugin match, his intention to interconnect more than one cameras, along with his domain knowledge, constitute the second service a preferable option over the first. The platform then auto-queries the ontology and a single matching service, namely `person_tracking`, that matches both queries is returned. A further auto-query allows the developer to select the next services, namely `dynamic_bg` and `stereo_camera`. The latter is an S–type service, thus does not trigger another auto-query. Finally the process is terminated when the user selects another S–type service, namely a `static_camera`. The composed service is depicted in Fig. 6.

[6]A subsume match does not guarantee that the composite service will execute smoothly during runtime, since the output providing service produces a more general type of data than the one the input receiving service consumes.

7. DISCUSSION AND FUTURE WORK

We next discuss various interesting features of the smart meeting room ontology and the semi-automatic service composition method. Regarding our ontology, its most prominent characteristics are: *(i)* the utilization of existing and well-established semantic models, *(ii)* the ease of use in instantiating a smart meeting room since the domain-specific nature of the ontology means that not only high-level, but also low-level concepts are provided and *(iii)* the ability to readily extend the ontology to include a new type of SPA service, by adding a new class in the resource-service type hierarchy. On the downside, since the ontology is oriented towards modelling SPA services of a smart meeting room, adaptation is necessary so as to apply it on another domain.

Our service composition method guarantees that if a composite service is created, all concepts included in the service request will be satisfied and all SPA services combined in the composite service can be invoked. Moreover, the exploitation of semantics allows us to overcome syntactic barriers and generate solutions that approximate the service request when an exact solution does not exist. The involvement of the developer is limited to the basic tasks of defining a service request and selecting services from platform-generated matching lists presented to him, hence the whole process evolves in a semi-automatic manner with minimum human intervention. It is the responsibility of the platform to discover appropriate SPA services and interconnect them, alleviating the need to know in advance what services are available and manually define their interconnection. It is now evident that service composition becomes a straightforward task, which can be even accomplished by an experienced user who is not a developer, but skillful enough to comprehend the semantics of service descriptions. A drawback of the algorithm described in Section 5.1 is the possibility of failing to produce a composite service, although the necessary services are available. This is attributed to the fact that the composition process immediately stops when an empty matching list is encountered after running the service discovery procedure for one of the inputs of an already selected service. However, this limitation is easily circumvented if, instead of stopping, the composition process prompts the developer to replace the service previously selected with an alternative one contained in the corresponding matching list. If no alternatives are available, it is possible to revert to an earlier step of the composition process, replace the service at this step and resume the composition from this step.

Future work will focus on fully automatic AI planning-based [6] or graph-based [13] service discovery and composition, exploiting semantic information from ontologies. This

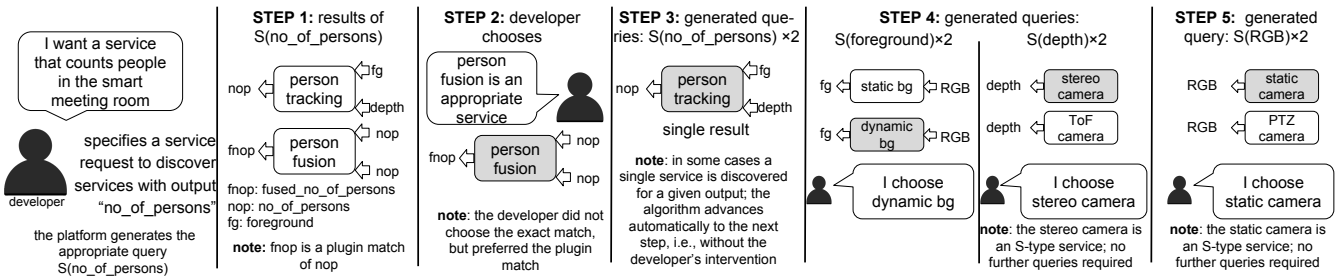

Figure 5: Semi-automatic service composition use case: creating a person counting service. The depicted services' IOPE concepts are assumed to be defined in an appropriate external ontology.

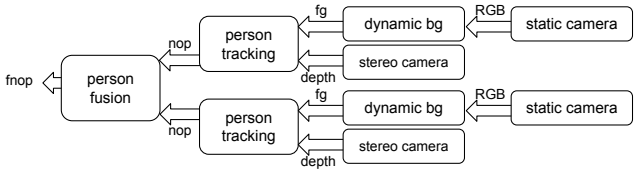

Figure 6: The composed person counting service.

way we aim to assist non-developers (i.e. end-users) to easily develop complex applications within an IoT ecosystem using its available services. Moreover, we shall consider adopting lightweight standards that are native to services, such as SAWSDL[7], to semantically annotate the services and their IOPEs in the SYNAISTHISI platform, as in [16].

Acknowledgments

This work was developed in the context of the "SYNAISTHISI" project, co-financed by the Greek GSRT, Ministry of Culture, Education & RA and the European RDF of the EC, under the OPCE II operational program, action KRIPIS.

8. REFERENCES

[1] C. Akasiadis, G. Tzortzis, E. Spyrou, and C. Spyropoulos. Developing complex services in an IoT ecosystem. In *IEEE World Forum on Internet of Things (WF-IoT)*, 2015.

[2] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. Internet of Things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys and Tutorials*, 17(4):2347–2376, 2015.

[3] A. Albreshne and J. Pasquier. Semantic-based semi-automatic web service composition. *Computer Department, Switzerland*, 2010.

[4] S. De, T. Elsaleh, P. M. Barnaghi, and S. Meissner. An Internet of Things platform for real-world and digital objects. *Scalable Computing: Practice and Experience*, 13(1), 2012.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[6] O. Hatzi, D. Vrakas, M. Nikolaidou, N. Bassiliades, D. Anagnostopoulos, and I. Vlahavas. An integrated approach to automated semantic web service composition through planning. *IEEE Transactions on Services Computing*, 5(3):319–332, 2012.

[7] Q. Liang, L. N. Chakarapani, S. Y. Su, R. N. Chikkamagalur, and H. Lam. A semi-automatic approach to composite web services discovery, description and invocation. *International Journal of Web Services Research (IJWSR)*, 1(4), 2004.

[8] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services, 2004.

[9] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[10] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Int'l Semantic Web Conf. (ISWC)*, 2002.

[11] C. Pedrinaci, D. Liu, M. Maleshkova, D. Lambert, J. Kopecky, and J. Domingue. iServe: a linked services publishing platform. In *Ontology Repositories and Editors for the Semantic Web Workshop at the Extended Semantic Web Conf. (ESWC)*, 2010.

[12] G. Pierris, D. Kothris, E. Spyrou, and C. Spyropoulos. SYNAISTHISI: An enabling platform for the current internet of things ecosystem. In *Panhellenic Conference on Informatics (PCI)*, 2015.

[13] P. Rodriguez-Mier, C. Pedrinaci, M. Lama, and M. Mucientes. An integrated semantic web service discovery and composition framework. *IEEE Transactions on Services Computing*, PrePrints, 2015.

[14] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[15] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu. Web services composition: A decade's overview. *Inf. Sciences*, 280:218–238, 2014.

[16] T. G. Stavropoulos, E. Kontopoulos, N. Bassiliades, J. Argyriou, A. Bikakis, D. Vrakas, and I. P. Vlahavas. Rule-based approaches for energy savings in an ambient intelligence environment. *Pervasive and Mobile Computing*, 19:1–23, 2015.

[17] T. G. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades. BOnSAI: A smart building ontology for ambient intelligence. In *Int'l Conf. on Web Intelligence, Mining and Semantics (WIMS)*, 2012.

[7] https://www.w3.org/TR/sawsdl