Behaviour-Aware Compositions of Things

Javier Cubo
Department of Computer Science
University of Málaga
Málaga, Spain
cubo@lcc.uma.es

Antonio Brogi
Department of Computer Science
University of Pisa
Pisa, Italy
brogi@di.unipi.it

Ernesto Pimentel
Department of Computer Science
University of Málaga
Málaga, Spain
ernesto@lcc.uma.es

Abstract—The new Internet has led the evolution of the Ubiquitous Web to integrate physical world entities into virtual world things. Thus, Internet is evolving into the vision of the Internet of Things where everyday life things are identifiable, readable, recognizable, addressable, and controllable via the Internet. As mass market penetration of networked embedded devices seems to become a reality, it is necessary to develop platforms considering this kind of devices and their interactions. Future Internet has emerged as a new initiative to pave a novel and dynamic global network infrastructure that handles the changing global needs of business and society. The Internet of Things could benefit from the Web Service architecture like today’s Web does. Then, Future service-oriented Internet things will offer their functionality via service-enabled interfaces. DPWS is a new emergent OASIS standard based on Web Service architectures to support interoperability among heterogeneous things. But there is still a need to represent explicitly the behaviour of things to develop applications in a more rigorous way. We propose to extend DPWS to specify the behaviour of things. We also propose verification techniques to check if a composition of things fulfills or violates the behaviour of those things.

Keywords—Behaviour, mashups, verification, Internet of Things, Web of Things, DPWS, Future Internet.

I. INTRODUCTION

The new Internet has led the evolution of the Ubiquitous Web 2.0 to integrate physical world entities into the virtual world things, as some initiatives are already addressing (e.g., cosm1, formerly pachube). In this pervasive networked environment people can communicate, interact with things, create new contents, and collaborate by taking advantage of the emergence of more dynamic, responsive and user-friendly Web applications, and by considering user experience coming from the increased interest in social networks. Then, today’s Internet is evolving into the vision of the Internet of Things (IoT) where everyday life objects, devices or things such as laptops, smartphones, on-board computers, video systems, household appliances, intelligent buildings, wireless sensor networks, ambient devices, RFID tagged objects, and commodities are identifiable, readable, recognizable, addressable, and even controllable via the Internet. As pointed by the European Research Cluster on the Internet of Things (IERC)2, things are expected to become active participants in business, information and social processes. They are enabled to interact and communicate among themselves and with the environment by exchanging data and sensed information. Furthermore, they can react autonomously to the real or physical world events and create services with or without direct human intervention. Services will be able to interact with these smart objects using standard interfaces that will provide the necessary link via the Internet in order to query and retrieve any information associated with them, taking into account behaviour, semantic or security issues.

Future Internet has emerged as a new initiative to pave a novel and dynamic global network infrastructure, with self-configuring capabilities, to meet the changing global needs of business and society. Future service-oriented Internet devices will offer their functionality via service-enabled interfaces adopting the vision of the Web of Things (WoT) (inspired from the IoT), e.g., via SOAP-based Web Services or RESTful APIs [12], [21]. In this way, other components can dynamically interact with them. Future Internet applications will have to support the interoperability between many diverse stakeholders by governing the convergence between both the physical and the virtual worlds, and by handling dynamic and continuous changes. Furthermore, the global market for networked systems and services is expected to grow exponentially in the next few years. Then, as mass market penetration of networked embedded devices seems to become soon a reality, it is clear that there is a need for both researchers and practitioners to develop platforms for the Future Internet applications taking this kind of devices and their interactions into account.

The IoT, including the mass of resource-constrained devices, could benefit from the Web Service architecture like today’s Web does. Recent work [9], [17] has focused on applying the paradigm of Service-Oriented Architecture (SOA) [10], in particular Web Services standards (SOAP, WSDL, etc.), directly on devices. In general, applying SOA to networked systems is a crucial solution to achieve reusability and interoperability of heterogeneous and distributed things. Specifically, implementing Web Service standards on devices presents several advantages in terms of integration by reducing the needs for gateways and translation between the components. This would enable the direct orchestration of services running on devices with high-level enterprise services. For instance, sensors physically attached to shipments can offer via Web Services their context information. Also these sensors could

1https://cosm.com/
2http://www.internet-of-things-research.eu/
be easily integrated in a process that updates some feature (e.g., temperature) and location of the shipment directly in the involved enterprise systems. Hence, the goal is to provide the functionality of each thing as a Web Service in an interoperable way that can be used by other entities such as enterprise applications or even other devices. However, adapting a given device to SOA is not a trivial problem. Then, it is required to implement efficiently the things, and many efforts are still needed to handle the composition and interaction of things coming from diverse sources, as well as to standardise and manage their data. In this sense, our main contribution is to specify the behaviour of things and to verify the behavioural correctness of their compositions.

This paper is organised as follows. In Section II our proposal is motivated. Section III presents the behavioural description we propose for specifying and composing things. Section IV describes verification techniques to check whether a composition of things fulfills or violates the behaviour of those things. In Section V our proposal is compared with related work. Finally, Section VI details some conclusions and outlines future work.

II. MOTIVATING OUR PROPOSAL

A. Problem Statement

SOA standards were designed primarily for connecting enterprise services. Therefore, the introduction of SOA to specify objects of the Future internet, such as sensor devices, brings new opportunities but also new challenges. Real-world things are deployed on resource constrained devices, e.g., with limited computing, energy, and storage capabilities. Then, we need to study the simplification, optimization, and adaptation of SOA standards to specify data and information coming from this kind of devices. Nevertheless, as even small, resource-constrained networked devices get more and more powerful in peer-to-peer and pervasive computing applications, it is common sense to try to adopt the SOA paradigms to embedded device networks. Hence, several SOA initiatives, such as OSGi\(^3\), UPnP\(^4\), or Jini\(^5\), have evolved to interconnect heterogeneous devices and services. But not all of them can equally adapt to the others using the same hood. Furthermore, the lack of standardization makes programming for devices an arduous task. Then, it is required a standard way for device manufacturers to expose devices to software developers and consumers, while providing developers with a standardized Application Programming Interface (API).

In order to address this issue, the new emergent OASIS standard Devices Profile for Web Services (DPWS)\(^6\) has been designed as a set of guidelines based on WS-* specifications to provide interoperability among different devices and services in a networked environment, e.g., a printer, a smartphone, a sensor or other new devices can detect DPWS-enabled devices on a network [16]. Some convincing points in favor of DPWS are that it is an OASIS standard, it employs a Web Service mode being built on the standard W3C Web Service architecture (SOAP + WSDL + XML-Schema), and it is natively integrated into Windows Vista and Windows 7 (with the WSDAPI implementation of DPWS). In DPWS, every device is abstracted as a service where features of the device are exhibited as hosted services. DPWS is lightweight, supports dynamic discovery in local networks, and can be used by orchestration or choreography standards, such as (Web Services) Business Process Execution Language (WSBPEL) or Web Services Choreography Description Language (WS-CDL). However, the comparison between the important properties of reuse and research challenges of Web Services shows a gap in the use of DPWS in the future focused on reusability [24]. Thus, DPWS shows, for example, those topics like business processes, context dependencies or quality factors have to get more focus in order to increase the reuse of DPWS devices. Hence, some open points of research for the future in this sense have been detected in order for DPWS to be used more easily in the area of software engineering and to become more accepted. Therefore, to develop Future Internet service-oriented applications and exploit correctly the composition among things, it is crucial to define rigorous methodologies. These methodologies should not only consider features as signature, eventing mechanisms, security and discovery as it is currently done by DPWS, but also complex real world integration, such as those involving complicated business processes. To the best of our knowledge, this difficult challenge has not been addressed yet.

In order to fulfill this goal, we have detected the need to explicitly represent the (implicit) behaviours of things in order to develop applications in a more rigorous way [5]. Specifically, we promote the usage of WS-* technologies to specify service interfaces of things by extending the standard DPWS with behavioural descriptions. The main purpose of this is to facilitate to developers the implementation of DPWS-compliant things (or devices) that host services by considering their behaviour in terms of the order in which the actions visible at the interface level are performed. We consider this challenge is crucial to control the behaviour of heterogeneous things during their compositions in highly dynamic environments of the Future Internet. These compositions will allow the creation of new applications generated as mashups of things where some concerns have to be handled, such as that the composition may violate the behaviour of the things (provoking lock situations) and some of their features may change at run-time.

B. Running Example

In this section, we motivate the need of our proposal of specifying the behaviour of things during their compositions. As aforementioned, we consider DPWS as the profile

\(^1\)http://www.osgi.org
\(^2\)http://www.upnp.org
\(^3\)http://java.sun.com/developer/technicalArticles/jini/JiniVision/jiniology.html
\(^4\)http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01
used to specify things or devices as services. Thus, devices may be things connected to the network and they become virtual devices that can be discovered and controlled via dynamic Web Services. With DPWS, devices available on a network can run two types of services: hosting services and hosted services, which are both accessed through messages by the clients. Hosting services are directly associated to a device, and play an important part in the device discovery process. Hosted services, on the other hand, are mostly functional sub-entities that are discovered through their hosting device. Thus, hosted services do not participate in the discovery process, but can be individually addressed through their respective End-Point-References (ERP), once the hosting device has been discovered. Devices may include zero or more hosted services. Unlike UPnP, services are described in WSDL and invoked using the normal Web Service invocation mechanism, which involves an exchange of messages between the service requester and service provider. Then, behaviour of things is required to specify the order of the sequence of exchanged messages. However, only static interfaces are provided by most legacy discovery protocols.

Consider a complex real-world example: an airport surveillance system composed by heterogeneous devices and people (using other devices) interconnected. The devices are: a motion detector and a surveillance camera located in a specific area in the airport, and a video device located in a control center. The motion detector is able to detect non-expected motions, the camera hosts a service to record a video controlling the position of the camera (operations like move, record and zoom), and the video device includes services that support video streaming (executing operations such as play, pause, stop, rewind, or fast-forward, as well as power control operations like on/off, or media ejection). Other DPWS-compliant devices or sensors (e.g., to sense light) could be easily integrated in the system. Figure 1 shows the DPWS-compliant architecture of such surveillance system.

We focus on a scenario where Bob (a security guard moving around the airport) connects, by means of an application installed within a mobile device (with an appropriate user-interface) to a new motion sensor and a new camera (which requires authentication to be accessed), both installed in a specific area of the airport. In our scenario, the behaviour of the system is the following: once Bob finds the new devices, when a non-expected motion is detected, Bob is notified with the exact position of the movement detected. He logs in to access the camera, and after he can perform three actions: (i) move the camera to the desired position, (ii) start to record, and/or (iii) make a zoom. If Bob considers there is an emergency situation, then he sends a command to warn the control center to start the streaming at real-time of the video being recorded by the camera. After this, the control center staff may reproduce and analyze the video, while it is recorded concurrently, and act accordingly. When monitoring and surveillance of the concrete situation is complete, then Bob can finish recording.

In this scenario it is required the handling of the behaviour of the hosted services into the heterogeneous and distributed devices, not only to achieve a correct composition among them, but also to get appropriate specifications of every behaviour-aware service and application. Figure 2 illustrates the sequence of interactions to get a correct execution of both single and composite things in the airport surveillance system.

DPWS devices announce their presence and expose services (if any) using a unique address and a standardized set of XML messages, by means of a message (Hello). DPWS clients can use the discovery process to find a device (Probe and ProbeMatch), enumerate its services (Get and Metadata), and connect to those services to perform specific actions (GetMetadata, Metadata, Subscribe, SubscriptionResponse, or Notification among others). Thus, a client first queries the device for complete descriptions of its services, including the service types. Optionally, the client can monitor state changes in each service by subscribing to events that occur during command execution. The client then controls the device by calling commands or operations defined by a service type, i.e., messages in bold in Figure 2, such as auth, authResponse; move, moveResponse; record, recordResponse; zoom, zoomResponse; and halt for the Service-Record-Contro, or messages like on; play, playResponse, or stop, stopResponse for the Service-Video-Streaming. Note that every pair such as record-recordResponse are two messages that correspond
to an operation *(record* in this case*) with input and output actions defined in its service interface (two-way control messaging). An operation may have only an input or output message (one-way control messaging), e.g., *halt* and *on*. The operations should be invoked in the correct order to avoid violations of the behaviour of things (partial order constraints or ordered sequence executions) that may cause locks.

**Service-Record-Control.** For instance, the Service-Record-Control always waits first for an authentication message *(auth)* before accepting other operations such as moving the camera *(move)*, recording a video *(record)*, or making a zoom *(zoom)*. In addition, the recording only can be finished *(halt)* when the record mode *(record)* is activated. This demonstrates the need to specify a partial order in the execution of the operations implemented in the services of a device. Furthermore, applications from the client-side also have to consider the partial order among the operations defined in their interfaces. Thus, if Bob never sends a login message to the Service-Record-Control to authenticate, then he never could begin to interact with this service, and accordingly the application installed within his mobile device could get locked out due to a deadlock situation during its execution.

**Service-Video-Streaming.** Following Figure 2, it can be observed the necessity of specifying a more complex behaviour of things, such as the one supported by the Service-Video-Streaming. For instance, the service could not accept rewind *(rewind)* or fast-forward *(fast-forward)* operations without receiving first the ordered sequences of operations play-pause or play-stop. Such behaviour needs a more complex representation, since the specification of partial order constraints is not enough to determine the correct full-sequence of operations. Note that some actions can be executed concurrently, for instance, *auth* and *on*, so this should be controlled. Another consideration in our scenario is that after executing *stop*, the *play* will correspond to a *record* of an updated video streaming.

With this example, we have shown that behaviour must be considered in the specification of each thing. Specifically, the airport surveillance system described in this section constitutes a mashup of heterogeneous devices. A mashup of physical things can be abstracted on the Web, dynamically composing and assembling things for a particular application (or even to be reused later by applying a reconfiguration process), where the capabilities of the participating things are utilized to build new more complex and sophisticated services in the Internet. Therefore, mashups should be generated by taking into consideration that heterogeneous behavioural descriptions (also known as interaction protocols) may be willing to cooperate in order to reach some common goal even though they meet dynamically and do not have a priori knowledge of each other. However, in DPWS (or any other SOA initiative to implement devices using the WS-* standards) the interaction protocol of a thing is not specified by the developers of the things or of the hosted services, with the consequent problems that may arise both an incorrect behaviour of a single thing and an incorrect behaviour of mashups of things.

In the next section, we propose a simple notation to specify the behaviour of things by indicating the order on any possible sequence between actions represented in Figure 2.

### III. Behavioural Description of Things

In this section, we introduce our approach of incorporating behaviour to the things in the Future Internet applications.

As is depicted in Figure 3, DPWS uses the primitives of the Web Services Architecture⁷ to create a framework for interoperable and standardised communication between embedded devices. The Web Services specifications in which DPWS is based are the following: (i) **WSDL** for describing the messages each hosted service is capable of sending and receiving, (ii) **SOAP** for transporting all the messages, (iii) **WS-Addressing** for advanced endpoint and message addressing, (iv) **WS-Policy** for policy exchange, (v) **WS-Security** for managing security, (vi) **WS-Discovery** and **SOAP-over-UDP** for device discovery, (vii) **WS-Transfer** / **WS-MetadataExchange** for device and service description, and (viii) **WS-Eventing** for managing subscriptions for event channels.

In DPWS, every device is abstracted as a service where features of the device are exhibited as Web Services. In Section II, we motivated the necessity of extending DPWS to facilitate the implementation of a device (or thing) as a full-service considering that its WSDL description not only specifies signature, but also the behaviour with the order in which input and output actions are performed while the networked system interacts with its environment. Input actions model methods that can be called, or the end of receiving messages from communication channels, as well as the return values from such calls. Output actions model method calls, message transmission via communication channels, or exceptions that may occur during methods execution.

In order to include this extension in the DPWS profile, our proposal aims to maintain a compromise between the DPWS-compliant incorporated expressiveness and the scalability issues in a world composed by billions of resource-constrained devices. In this sense, we apply rigorous and lightweight methodologies to develop things. The proposal consists of promoting WS-* technologies to specify service interfaces of things by adding the behaviour of

---

⁷[http://www.w3.org/TR/ws-arch/](http://www.w3.org/TR/ws-arch/)
things to the DPWS profile. In such a way, this extended DPWS specification will facilitate to the developers the implementation of DPWS-compliant things (or devices) that host services by taking into account their behaviour (by means of constraints or full-sequences) in terms of the order in which the actions visible at the interface level are performed while the things are composed.

**Constraints.** When only a partial order of the behaviour of things is required, we propose to use three types of behavioural “constraints” [6] to be added to the guidelines (statements) exposed by DPWS:

\[
\begin{align*}
&\{b_1, \ldots, b_n\} \text{ afterAll } \{a_1, \ldots, a_m\} \\
&\{b_1, \ldots, b_n\} \text{ afterSome } \{a_1, \ldots, a_m\} \\
\end{align*}
\]

The *afterAll* constraint is used to specify that any left-side action (one or more than one from \{b_1, \ldots, b_n\}, we call them “after” operations, i.e. post-actions) only can be executed iff all the right-side actions (\{a_1, \ldots, a_m\}, called “before” operations, i.e. pre-actions) have been previously executed. The *afterSome* constraint is less restrictive than *afterAll*, since in this case any left-side action can be executed whether some right-side action have been already executed. The *onlyOneOf* constraint means that only one of the set of operations (\{a_1, \ldots, a_m\}) can be executed in an interaction session. We have decided to include these constraints in the <documentation> tag of the WSDL descriptions, by adding a new tag called <behaviour> that specifies the constraints (<constraints> tags) to get a correct partial order in the exchange of messages.

For instance, the behaviour of the Service-Record-Control of the Device-Camera introduced in the previous section can be specified by means of the constraints:

\[
C_{A1}: \{\text{move, record, zoom}\} \text{ afterAll } \{\text{auth}\} \\
C_{A2}: \{\text{halt}\} \text{ afterAll } \{\text{record}\}
\]

Figure 4 depicts an excerpt of the WSDL for the Service-Record-Control hosted in the Device-Camera (which forms part of its DPWS profile).

The corresponding states changes according to the messages execution, we propose to use Finite State Machines (FSMs) [4] as a simple and user-friendly graphic solution to represent the complex relationships between messages.

Coming back to the example (specifically to the **Service-Video-Streaming**), we can observe that a simple operation (e.g., rewind the video streaming (*rewind*)) may require a considerable number of exchanged messages in a concrete order, so the handling of the complex behaviour of these services hosted in a device (and also of applications) is essential. This service may support the actions *on, play, pause, stop, rewind, fast-forward*, and *off*.

In Figure 5, we represent the control of the message full-sequence of the hosted Service-Video-Streaming of the Device-Video using FSM representation, and an excerpt of the corresponding WSDL description. FSM simply represents the states changes according to the messages execution, and it is easy to specify by the developer.

In the next section we will describe verification techniques to check if a composition of things fulfills or violates the behaviour of those things.

**IV. VERIFICATION OF BEHAVIOUR-ARENA MASHUPS OF THINGS**

The explicit specification of the behaviour of things by means of rigorous mechanisms, which we previously presented, is the foundation to develop behaviour-aware compositions of things. These compositions will create applications generated in form of mashups with new functionalities to be remotely accessed (e.g., as Software-as-a-Service, SaaS, or Mashups-as-a-Service). Mashups have surpassed the notions of integration and convergence and have become an important new trend that permeates all of society. Service mashups indicate a way to design and develop novel and modern Web applications by combining existing resources utilising content from diverse sources or devices, and Web APIs. Thus, mashups facilitate the exponential building of new more complex
and sophisticated services in the Internet because of the flexible composition of pre-existing services in order to reuse them not only by selecting and assembling them, but also by combining and consolidating them. Then, from our vision, since mashups of physically separated devices are being attempted through the mutual network connections between heterogeneous objects, these new applications will exploit information coming from diverse sources, for instance, social network services, Web feeds as RSS, or nodes (e.g., sensing temperature or humidity). For example, location-based services, which are a combination of social network and mobile services, are already starting to show numerous new services through mashups. Social network services, such as Twitter, are being combined with things like mobile and local-based services or sensors, while being transformed into a marketing channel, a way for emergency communications, or real-time media services for micro-blogging [22].

Our mashups take into account the behavioural descriptions of things to be connected, thus improving the current technologies to compose heterogeneous things. In such a way, we will compose things represented either by a set of dependencies which act as partial order constraints of their simple behaviour, or by a FSM defining full-sequence behaviour.

To specify composition of things, we propose a process model of a mashup of things simply represented as a graph whose nodes represent invocations (to things), and whose edges represent a precedence relation among those invocations (viz., an edge from A to B states that invocation B can be performed only after performing invocation A).

To illustrate the composition, consider a possible orchestration of the things in the scenario of the running example. This specification is depicted in Figure 6, where the actions of every service (Service-Record-Control and Service-Video-Streaming) are represented by nodes.

According to the example, auth and on can be executed concurrently. To take the behaviour of things into account, we need to verify whether this orchestration fulfills or violates the behaviour of each thing. A possible sequence of invocations of this composition could be auth-record-play-stop-move-play-restart-stop-off, in which case: (1) play will be executed without having executed previously on, which violates the behaviour defined in the automaton of the Service-Video-Streaming (Figure 5); and (2) rewind could be executed without receiving immediately before one of the ordered sequences play-stop or play-pause, which also violates the behaviour of that service, since the first occurrence of play-stop is not considered as play is executed again.

As we specify the behaviour of things using rigorous but lightweight methodologies, we claim our model allows a simple and effective verification. Hence, we have defined the recursive checker function check in order to perform the verification. Function check returns true if the composition respects the behaviour of all things that are composed, and it returns false otherwise. The function has four input parameters: a trace of the orchestration to be analysed, a set E of the actions executed so far, a list of pairs \((M_i, s_i)\) where \(M_i\) is a FSM modelling the behaviour of thing \(t_i\) and where \(s_i\) is the current state of \(M_i\), and a list of sets of behavioural constraints \([C_{n+1}, ..., C_{n+h}]\) where \(C_i\) denotes the set of constraints associated with thing \(t_i\). In order to track the source of every action \(a\) in the trace \(t\) and the set of actions \(E\), actions will be tagged with the thing executing it: \((a : t)\).

For the sake of simplicity we assume that FSMs are deterministic. For FSMs to be in a final state (auxiliary function \(fStates\)), it returns false otherwise. If the trace to be checked is empty, the function returns true if all FSMs are in a final state, otherwise, it returns false. If the trace to be checked is not empty, the function analyses the first invocation \((t_{i_1} : a)\) of the trace still to be analysed. If the behaviour of the invoked thing \(t_{i_1}\) is denoted by a FSM \((i \leq n)\), then the function checks that action \(a\) is executable in the current state \(s_{i_1}\) of the FSM \((T(M_i))\) denotes the set of transitions of FSM \(M_i\), and if so it continues to analyse the rest of the trace. If instead the behaviour of the invoked thing \(t_{i_1}\) is denoted by a set \(C_{i_1}\) of constraints, then the function checks that action \(a\) does not violate such constraints \((C_{i_1} : E \models (t_{i_1} : a))\), and if so it continues to analyse the rest of the trace. Relation \(\models\) is formally defined as follows: if \(C, E \models (t_{i_1} : a)\) and \(C \models (t_{i_1} : a)\) then \(C, E \models (t_{i_1} : a)\) where (B after All A), E \(\models (t_{i_1} : a)\) if (if \(a \in B\) then \(\forall x \in A : (t_{i_1} : x) \in E\)) (B after Some A), E \(\models (t_{i_1} : a)\) if (if \(a \in B\) then \(\exists x \in A : (t_{i_1} : x) \in E\)) (onlyOneOf A), E \(\models (t_{i_1} : a)\) if (if \(a \in B\) then \(\exists x \in A : (t_{i_1} : x) \in E\))
It is easy to see that the check function determines that the orchestration of Figure 6 violates the behaviour of the Service-Video-Streaming\(^9\).

V. Related Work

Recent works are making progress with respect to expose devices as services, and to handle their composition. In [15] are identified both the state of the art and the key research directions related to service-oriented middleware for the Future Internet: service description, discovery, access, and composition. Until recently, these aspects were only considering services, but Future Internet is already a reality, so the necessity of considering contents, devices, sensors, and things is included in the new challenges.

As regards the decision to select the more appropriate technology for specifying devices, some works [12], [21] have compared REST vs WS-* technologies. They claim that RESTful Web Services are easy to learn and suitable for programming IoT applications, and their main advantage is their universality and the uniform service interface. But, they also argue REST addresses only basic distributed interaction and coordination, leaving many open complex issues that have been tackled by WS-* technologies, such as dealing with service behaviour, semantics, or quality of services. In addition, WS-* specifications benefit from a clearer standardization process than REST. The SIRENA\(^10\) European project has played a pioneering role by applying the SOA paradigm to communications and interworking between components at the device level, with the main objective to develop a Service Infrastructure for Real time Embedded Networked Applications. The SIRENA results [3] were used as a foundation for both SODA\(^11\) and SOCRADES\(^12\) projects, with the consideration of selecting DPWS as the best choice to achieve the device integration in heterogeneous domains. Furthermore, also by stemming from the SIRENA project, the initiative “Web Services for Devices” (WS4D)\(^13\) complies with the DPWS. WS4D brings SOA and Web Services technology to the application domains of industrial automation, home entertainment, and automotive and telecommunication systems, by facilitating the setup and management of network-connected devices in distributed embedded systems.

The ongoing FET European project, CONNECT\(^14\), drops interoperability barriers by synthesizing on the fly the connectors via which networked systems communicate [14]. It assumes that a networked system comes together with a Labeled Transition System (LTS) based specification of its interaction protocol by specifying its behaviour. Also, in [2], [7], authors working in CONNECT propose derive from the WSDL of a Web Service a partial ordering relation among the invocations of the different WSDL operations, that they represent as an automaton, which models the interaction protocol that a client has to follow in order to correctly interact with the Web Service. The behaviour protocol is obtained through synthesis (driven by data type analysis, obtaining a preliminary dependencies automaton, and optimized by means of heuristics), and testing stages (to verify conformance). Compared to our approach, the CONNECT project first assumes the behaviour protocol is already specified by means LTSs, while we are proposing to specify service interfaces of things by adding a set of single constraints to the DPWS guidelines in order to determine the links in the interactions. But even later, when they propose to derive a partial order of the message sequence of a service, their approach is too complex and it does not easily maintain the compromise between the expressiveness and the scalability issues in a world composed by billions of resource-constrained devices, since both synthesis and testing processes are required. However, our approach is to apply rigorous and lightweight methodologies to develop things, by facilitating developers the implementation of DPWS-compliant devices.

Some works have proposed service-oriented solutions for Home Network System [19] or Smart Home [20]. The former presents a sensor mashup platform which allows the dynamic composition of the existing sensor services. They mainly focus on helping non-expert developers to create context-aware services within the home network system, but their framework does not offer a guide to control the behaviour of the system, only messages are exchanged by using WSDL and REST/SOAP. The latter is closer to our proposal. Authors propose an application logic distribution where devices in a smart home incorporate a set of rules than can govern their behaviour, following ECA (Event-Condition-Action) rules: they listen to external messages (notifications coming from other services) and, according to some conditions defined in these rules, they decide to perform their own actions. In comparison to our approach, this mechanism is not lightweight and it can introduce too much complexity when we are dealing with resource-constraint devices, being necessary a rule engine to analyze the rules. In addition, a rule is not enough to determine the correct order among operations of a service, since a full-rule requires events coming from other services to be triggered. Therefore, the protocol detailing the partial order or the full-sequence among operations (of a single device) can not be generated in a simple way by means ECA rules.

Other efforts are mainly focused on the WoT vision, both (i) specifying, discovering and integrating things by means of a semantic Web-based architecture [18], [23], and (ii) generating mashups of heterogeneous things [13], [22]. However, the main gap of these proposals is that devices with a more complex behaviour could not be connected by using theirs mechanisms, since implicit behaviours of things are not considered. Then, violations of the (implicit) behaviour of things might happen, and

\(^9\)To avoid violating the behaviour of the Service-Video-Streaming, a precedence edge from on to play must be introduced in the orchestrator, and an invocation to pause may be for instance introduced before the second invocation of play and the invocation of rewind.

\(^10\)http://www.sirena-itea.org/

\(^11\)http://www.soda-itea.org

\(^12\)http://www.socrates.eu/

\(^13\)http://ws4d.e-technik.uni-rostock.de/about/

\(^14\)https://www.connect-forever.eu/
the system could get locked out during its execution, for instance due to a deadlock situation. Our approach proposes to design its foundation on an extension of the DPWS profile by allowing the incorporation of behaviour-aware things in the mashups generated without human intervention as regards to the interaction protocol at run time, and by ensuring the composition work correctly.

ThingML [11] is a domain-specific modeling language to efficiently provide communicating services on resource-constrained devices. The proposed language allows to specify interaction protocols via state machines, and it would be worth investigating whether and how those state machines may be mapped into our FSMs, so as to allow their inclusion in WSDL interfaces as accounted by DPWS.

VI. CONCLUDING REMARKS

In this paper, we have demonstrated the need of considering the behaviour of things, and we have proposed a rigorous and lightweight theoretical foundation for representing the behaviour of heterogeneous things. Our methodology relies on the service-oriented paradigm and extends the DPWS profile with simple constraints and/or FSMs that specify the order with which things can receive messages. To check whether a composition of things fulfills or violates their behaviour, we have proposed a simple and efficient verification technique. We have illustrated our proposal in a real-world scenario.

We are working on the development of a Graphical User Interface (GUI) to ease the generation of behaviour descriptions of things. We plan to try including our verification technique into (some of) the existing things-oriented platforms (like ThingML, WS4D-PipesBox, or Paraimpu).

We also plan to extend the model and the platform to define and exploit a set of recommendations or patterns in order to generate mashups considering semantic description, and context and social information (e.g., based on context or social relationships between objects, that allows, among other features, a certain trust among them).

ACKNOWLEDGMENTS

Work partially supported by projects TIN2008-05932 funded by the Spanish Ministry of Science and Innovation and FEDER, and P11-TIC-7659 funded by the Andalusian Government.

REFERENCES


