Towards Run-Time Verification of Compositions in the Web of Things using Complex Event Processing

Javier Cubo[1], Laura González[2], Antonio Brogi[3], Ernesto Pimentel[1], Raúl Ruggia[2]

[1]Department of Computer Science, University of Málaga, Spain
{cubo,ernesto}@lcc.uma.es
[2]Instituto de Computación, Facultad de Ingeniería, Universidad de la República, Uruguay
{lauragon,ruggia}@fing.edu.uy
[3]Department of Computer Science, University of Pisa, Italy
brogi@di.unipi.it

Abstract. Following the vision of the Internet of Things, physical world entities are integrated into virtual world things. Things are expected to become active participants in business and social processes. Then, the Internet of Things could benefit from the Web Service architecture like today's Web does, so Future service-oriented Internet things will offer their functionality via service-enabled interfaces. In previous work, we demonstrated the need of considering the behaviour of things to develop applications in a more rigorous way, and we proposed a lightweight model for representing such behaviour. Our methodology relies on the service-oriented paradigm and extends the DPWS profile to specify the order with which things can receive messages. We also proposed a static verification technique to check whether a mashup of things respects the behaviour, specified at design-time, of the composed things. However, a change in the behaviour of a thing may cause that some compositions do not fulfill its behaviour anymore. Moreover, given that a thing can receive requests from instances of different mashups at run-time, these requests could violate the behaviour of that thing, even though each mashup fulfills such behaviour, due to the change of state of the thing. To address these issues, we present a proposal based on mediation techniques and complex event processing to detect and inhibit invalid invocations, so things only receive requests compatible with their behaviour.

Keywords: Composition, Mashup, Verification, Service-Oriented Things, Web of Things, Internet of Things, Mediation Patterns, Complex Event Processing.

1 Introduction

Following the vision of the Internet of Things (IoT), physical world entities are integrated into virtual world things. Things are expected to become active participants in business, information and social processes. 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 [1]. The IoT, including the mass of resource-constrained devices, could benefit from the Web Service architecture like today's Web does. Recent work

[2,3] has focused on applying the paradigm of Service-Oriented Architecture (SOA), in particular Web Services standards, directly on devices. Applying SOA to networked systems is a crucial solution to achieve reusability and interoperability of heterogeneous and distributed things. 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. 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.

To address this issue, the new emergent OASIS standard Devices Profile for Web Services (DPWS)[1] 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. 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 from Windows Vista and 7. 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 WSBPEL or 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 [4]. For example, business processes, context dependencies or quality factors have to get more focus to increase the reuse of DPWS devices. Hence, some points in this sense have been detected so that DPWS become more used and accepted in software engineering.

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 complex business processes. In our previous work [5], we detected the need to explicitly represent the (implicit) behaviours of things to develop applications in a more rigorous way. Specifically, we promoted 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 (partial) 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

[1] http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01

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. We proposed a static verification technique to check whether or not a composition or mashup of things respects the behaviour of the composed things specified at design-time.

However, a change in the behaviour of a thing may cause that various compositions do not fulfill its behaviour anymore. Although compositions could be redesigned to comply with the new behavior, it would be appropriate to design run-time verifications techniques to react when this situation occurs. Moreover, given that a thing can receive at run-time requests from instances of different mashups, these requests could violate the behaviour of that thing, even though each mashup fulfills such behaviour, because of the state's change of the thing. This kind of situations cannot be detected at design-time, so run-time mechanisms are required to become aware of it and act accordingly. To address these issues, as main contribution, in this ongoing work, we present a proposal based on mediation techniques and Complex Event Processing (CEP) [6] to detect and inhibit invalid invocations, so that things only receive requests compatible with their behaviour. Also, as a future plan, issues such as temporal restrictions or quality of services aspects could be managed using this proposal.

This paper is organized as follows. Section 2 introduces the background for our work. In Section 3, we present our proposal in a nutshell to detect invalid invocations at run-time during the composition of things. Section 4 briefly compares our approach to related works. Finally, Section 5 outlines some conclusions and perspectives.

2 Background

This section presents background information on the technologies used in this work.

2.1 Behavioural Description of Things and Static Verification

As aforementioned, in [5] 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 should specify not only signature, but also the behaviour with the order in which input and output actions are performed while the networked system interacts with its environment. To include this extension in the DPWS profile, our previous proposal aimed to maintain a compromise between the DPWS-compliant incorporated expressiveness and the scalability issues in a world composed by billions of resource-constrained devices. We applied lightweight methodologies to develop things. The proposal consists of promoting WS-* technologies to specify service interfaces of things by adding the behaviour of things to the DPWS profile. In such a way, this extended DPWS specification will facilitate to the developers (the user profile using our proposal to specify the behaviour of things) the implementation of DPWS-compliant things (or devices) that host services. The behavior of things will be taken into account in terms of the order in which the actions visible at the interface level are performed while the things are composed, by means of *constraints* or *full-sequences*.

- Constraints. When only a partial order of the behavior of things is required, we propose to use three types of behavioural *"constraints"* to be added to the guidelines (statements) exposed by DPWS: $\{b_i\}$`afterAll`$\{a_i\}$, $\{b_i\}$`afterSome`$\{a_i\}$, `onlyOneOf`$\{a_i\}$, where $\{b_i\}$ and $\{a_i\}$ are actions of a service hosted in a device.

- Finite State Machines. In those cases where it is required to specify not only the partial order, but also the ordered *"full-sequence"* among operations with the corresponding states changes according to the messages execution, we propose to use Finite State Machines (FSMs) to represent the complex relationships between messages.

The explicit specification of the behaviour of things by means of constraints or FSMs, 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, or Mashups-as-a-Service). But it is required to check whether a composition of things fulfills or violates their behaviour, so we proposed a simple and efficient verification technique at design-time. We defined a checker function to perform the static verification, analysing traces and actions executed of the orchestration specified by the user, according to a set of constraints and/or FSMs, both determining the behaviour of the things.

Example. In order to illustrate this model, we considered in [5] a complex real-world example: an airport surveillance system composed by heterogeneous devices (a motion detector and a surveillance camera located in a specific area in the airport, and a video device located in a control center) and people (using other devices) interconnected. Here, for space reason, we only give an example of two possible constraints which may be specified for determining the behavior of a service record control hosted in the device camera, with actions such as *auth*, *move*, *record*, or *halt*, as follows:

C1: {*move*, *record*} `afterAll` {*auth*}; *C2:* {*halt*} `afterSome` {*move*, *record*}

2.2 Complex Event Processing and Mediation Patterns

Complex Event Processing (CEP) refers to methods, techniques, and tools for processing events while they occur. CEP allows deriving relevant higher-level events (i.e. complex events) from a combination of lower-level events, in a timely fashion and permanently [7]. To this end, event queries are continuously monitoring incoming streams of simple events. The use of production rules is one of the approaches to implement event queries [7]. CEP platforms provide support for various types of event patterns, which allow specifying combinations of events.

We also review the integration and mediation solutions relevant for our proposal [8,9,10]. Service virtualization patterns take an existing service and deploy a new virtual service in a mediation platform [8]. The VETO pattern [9], which consists in applying a sequence of mediation mechanisms: validate, enrich, transform and operate, is a frequently applied mediation pattern that can be used in conjunction with the previous one. Event-driven integration patterns deal with distribution of events in real time and integration with CEP engines. The event reactor pattern extends the previous one by supporting a synchronous interaction with a CEP engine to check if the latest event has triggered a complex event, so that a mediation flow can react to it [10].

3 Our Proposal in a Nutshell

This section presents the proposed approach, which has the goal of enabling the runtime verification of behaviour-aware compositions of things.

3.1 General Description

The main idea of our proposal consists in processing invocations of services hosted in devices through a mediation platform, in order to detect and block the invalid ones using CEP techniques. In this way, devices only receive requests which are compatible with their behaviour. Fig. **1** presents the general architecture of the proposal.

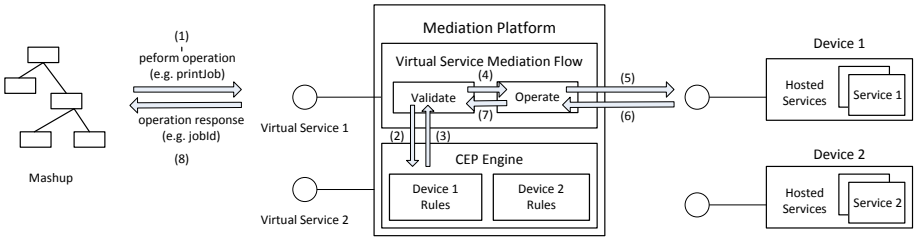

Fig. 1. Overview of the General Architecture

Following the Virtual Services mediation pattern, hosted services are invoked through virtual services deployed in the mediation platform (1). Virtual services consist of a mediation flow, which is a simplification of the VETO pattern, and comprises two mediation mechanisms: validate and operate. The validate mechanism synchronously interacts with a CEP Engine (2), following the event reactor pattern, to check if the invoked operation is invalid. If this is the case, a complex event is triggered (3) and the invocation is blocked. Otherwise, the operate mechanism is executed (4) which, in turn, invokes the target operation in the hosted service (5). Finally, the response is returned to the invoking client (6), (7) and (8).

3.2 Detecting Invalid Invocations at Run-Time

In order to detect invalid invocations, the platform leverages CEP techniques. Concretely, based on the specified behavior, by means of constraints (this work's scope, FSMs will be addressed further), of each device, a set of production rules is deployed on the CEP Engine. In addition, when the platform receives an invocation a new event, or fact, is generated and sent to the engine. These events and the deployed rules constitute the basic elements used to trigger a complex event when an invalid invocation for a hosted service is received, allowing the platform to detect this situation.

Example. Considering the simple example introduced in Section 2.1, Table 1 presents the production rules to be deployed for the hosted service record control. Note that rules can be automatically generated based on the specified constraints.

Table 1. Rules for the Service Record Control with two Behavioural Contraints

Rule Name	Rule Description	Rule Specification Pseudo-code
overlaps	If a mashup instance is running, it detects if	**CONDITION:** an invocation for the service was received **and**

	invocations to the service from other instances were received.	a mashup instance using the service was running **ACTION:** Trigger an "invalid invocation" event with description: "Overlapping instances"
check-c1	It detects when a received invocation violates the constraint *C1:* {*move*, *record*} `afterAll` {*auth*}	**CONDITION:** an invocation for the service was received **and** the invoked operation is "*move*" or "*record*" **and** an "*auth*" operation was not received before for the mashup instance **ACTION:** Trigger an "invalid invocation" event with description: "Constraint C1 violated"
check-c2	It detects when a received invocation violates the constraint *C2:* {*halt*} `afterSome` {*move*, *record*}	**CONDITION:** an invocation for the service was received **and** the invoked operation is "*halt*" **and** a "*move*" operation or a "*record*" operation was not received before for the mashup instance **ACTION:** Trigger an "invalid invocation" event with description: "Constraint C2 violated"

There are several CEP engines in which this kind of rules can be deployed. In particular, Drools Fusion is the module responsible for enabling CEP capabilities within the Drools platform[2]. Fig. **2** presents how a simplified version of the rule "check-c1" can be specified in Drools Fusion, using the Drools Rule Language (DRL).

```
rule "check-c1" // {move, record} afterAll {auth}
        when
                $op: ServiceOperation(operationName=="move" ||
                                      operationName=="record")
                not (ServiceOperation(this before $op,
                                      instanceId==$op.instanceId,
                                      operationName=="auth"))
        then
                insert (new InvalidInvocation("Constraint C1 violated"));

end
```

Fig. 2. Rule "check-c1" specified with the Drools Rule Language

4 Related Work

CEP and mediation techniques are being increasingly used for runtime monitoring, verification and adaptation in service-oriented and event-based solutions.

In [11] the authors describe a general approach to deal with differences between Web Services protocols, by using CEP to adapt their interactions and resolve their conflicts. Compared to our approach, message consumption and transmission are

[2] http://www.jboss.org/drools/

modeled as events, and the adaptation is specified using automata and deployed as CEP adapters. In [12] an event-based approach to verify the compliance of the overall sequence of inter-organizational choreography operations is presented. In this case, each message received or sent by an organization is associated to an event and CEP is used to verify whether the participating parties have performed their tasks according to the control flow constructs of the choreography. And in [13] the authors propose an integrated solution for run-time compliance governance in SOA, focusing on quality of service, security and licensing issues. In a similar way to our proposal, this solution uses CEP to monitor the compliance of business processes during their execution. However, although these proposals leverage CEP and mediation techniques for run-time verification, none of them focus neither in the field of the Web of Things nor in verifying the compliance of invocations according the specified behaviour of services. Nevertheless, recently, CEP techniques and mediation solutions have been applied and considered relevant in the field of the Web of Things in a separate way.

As regards CEP applied to the things world, in [14] the authors propose a solution to deal with imprecise timestamps and events order in this highly distributed context. Also, in [15] a solution to solve the integration of heterogeneous event information resources is proposed. However, none of these solutions uses CEP techniques for the run-time verification of invocations. Mediation solutions have been also proposed in the field of the Web of Things. In [16] a middleware infrastructure focused on enabling an efficient collaboration between device-level services and enterprise applications is presented. In [17] the authors propose the concept of Gateway as a Service: a cloud computing framework for the Web of Things, focused on integrating devices into service compositions and business processes. Also, in [18] an integrated development and runtime environment for the future internet is proposed, which include a Light Service Bus to address the access to things considering their resource constraints and leveraging DPWS. All these proposals focus on using mediation capabilities to enable the connectivity to heterogeneous things; but unlike our proposal, they do not provide mechanisms to detect invalid invocations to things according their behaviour. Therefore, to the best of our knowledge there is not any effort in the field of the Web of Things that uses both CEP and mediation techniques jointly to address the run-time verification of the behaviour of things.

5 Conclusions and Perspectives

We have briefly presented our ongoing work to detect and inhibit invalid invocations at run-time while things are composed, by using mediation techniques and CEP. We complement our previous static verification mechanism, and we check at run-time things only may receive requests compatible with their behavior. We have illustrated our proposal generating production rules for a service with two behavioural constraints, and deploying them in a particular CEP engine. We are currently working on applying, implementing, and deploying this proposal not only to create rules based on constraints but also determined by FSMs. We also plan to extend the general architecture with (i) a Discovery Proxy to discover services in a managed network, (ii) an Integrated Developer Environment (IDE) as Mashup Editor to specify the orchestration corresponding to mashups, and (iii) a Mashups Execution Environment to deploy

the generated mashups. Furthermore, we could control other future issues related to the behaviour including temporal restrictions, or quality of services (e.g., managing the maximun number of invocations that a thing may receive) with this new approach. In addition, as a long-term future work we are considering to study the inclusion of some recovery strategy in case an invalid invocation occurs. We are also planning the evaluation of potential performance issues in scenarios with many things interacting.

Acknowledgements

Work partially supported by projects TIN2008-05932, TIN2012-35669, CSD2007-0004 funded by Spanish Ministry MINECO and FEDER; P11-TIC-7659 funded by Andalusian Government; and Universidad de Málaga, Campus de Excelencia Internacional Andalucía Tech.

References

1. Guinard, D., Ion, I., Mayer, S.: In Search of an Internet of Things Service Architecture: REST or WS-*? A Developers' Perspective. In Proc. of MobiQuitous'11, volume 104 of LNICST, pages 326–337. Springer (2012).
2. de Souza, L.M.S., et al.: Socrades: A Web Service Based Shop Floor Integration Infrastructure. In Proc. of IoT'08, volume 4952 of LNCS, pages 50–67. Springer (2008).
3. F. Jammes and H. Smit. Service-Oriented Paradigms in Industrial Automation. IEEE Trans. Ind. Informatics, 1(1):62–70 (2005).
4. M. Zinn et al.: Device Services as Reusable Units of Modelling in a Service-Oriented Environment-An Analysis Case Study.In Proc. of ISIE'10,pages 1728–1735.IEEE CS (2010).
5. Cubo, J., Brogi, A., Pimentel, E.: Behaviour-Aware Compositions of Things. In Proc. of iThings'12 in conjunction with GreenCom'12, pages 1–8. IEEE CS (2012).
6. Luckham, D.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002).
7. Eckert, M., et al.: A CEP Babelfish: Languages for Complex Event Processing and Querying Surveyed. Studies in Comp Intelligence, volume 347, pages 47–70. Springer (2011).
8. Wylie, H., Lambros, P.: Enterprise Connectivity Patterns: Implementing integration solutions with IBM's Enterprise Service Bus products. Accessible at IBM website.
9. Chappell, D.: Enterprise Service Bus: Theory in Practice. O'Reilly Media (2004).
10. Hohpe, G., Woolf, B.: Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. Addison-Wesley Professional (2003).
11. Taher, Y., et al.: Adaptation of Web Service Interactions using Complex Event Processing patterns. In Proc. of ICSOC'11, volume 7084 of LNCS, pages. 601–609. Springer (2011).
12. Baouab, A., et al.: An Event-Driven Approach for Runtime Verification of Inter-Organizational Choreographies. In Proc. of SCC'11, pages 640–647. IEEE CS (2011).
13. Birukou, A., et al.: An Integrated Solution for Runtime Compliance Governance in SOA. In: Maglio, P.P., et al. (eds.) Service-Oriented Computing,pages 122–136. Springer (2010).
14. Fengjuan, W., et al.: The Research on Complex Event Processing Method of Internet of Things. In Proc. of ICMTMA'13, pages 1219–1222 (2013).
15. Wang, W., Guo, D.: Towards Unified Heterogeneous Event Processing for the Internet of Things. In Proc. of IOT'12, pages 84–91 (2012).
16. De Souza, et al.: SOCRADES: A Web Service based Shop Floor Integration Infrastructure. In Proc. of IOT'08, pages 50–67. Springer (2008).
17. Wu, Z., et al.: Gateway as a Service: A Cloud Computing Framework for Web of Things. In Proc. of ICT'12, pages 1–6 (2012).
18. Hamida, A.B., et al.: An Integrated Development and Runtime Environment for the Future Internet. In: Álvarez, F., et al. (eds.) The Future Internet, pages 81–92. Springer (2012)