UbiUnity: a Dynamic Visual Simulation Framework for Web of Things

Stéphane Lavirotte[1,2], Jean-Yves Tigli[1,2], Gérald Rocher[1,2], Léa El Beze[1], Adam Palma[1,2]
(1) Université Nice Sophia Antipolis, Polytech'Nice Sophia
(2) Centre National de la Recherche Scientifique, Laboratoire I3S, CNRS UMR 7271
Sophia Antipolis, France
Stephane.Lavirotte@unice.fr, Jean-Yves.Tigli@unice.fr, Gerald.Rocher@unice.fr

Abstract — **The development of smart spaces is a complex and challenging task. The choice of suitable sensors and actuators to deploy in these physical testbeds is difficult without experimentation. Moreover, several challenges still remain in improving and testing new fields of application based on Web of Things (WoT). In this paper, we present UbiUnity, a dynamic visual simulator environment which can be used during the design phase of smart spaces. Our approach allows to define web services for devices (WSD) associated to 3D virtual shape in the simulated environment. These WSD are defined by combining simple actions in the virtual environment allowing the researchers to focus on the definition of new algorithms or middleware to manage the smart space. Moreover, UbiUnity can dynamically create these WSD during the execution of the simulation with respect to the fluidity of the graphics rendering.**

Keywords—Web of Things, Ubiquitous Computing, Simulation, Virtual Environment, Web Services for Devices

I. Introduction

Many initiatives aim at implementing smart spaces [1]. These smart spaces are composed of Things (entities of interest, like buildings, rooms…) that can be viewed as a set of devices providing services and/or resources [2]. But the design, the development and the deployment of such smart spaces, which are real physical systems, are complex engineering tasks. The creation of physical spaces equipped with sensors and actuators have drawbacks: (a) huge investments, (b) a limited variety of sensors and actuators, (c) updates and upgrades are complex and limited to physical devices constraints, and (d) limited deployment scalability. On the other hand, the definition of new architectures, new middleware or new algorithms for the management of smart spaces required for research purposes need a flexibility of implementation and deployment to ease the experimentations. Therefore, there are numerous advantages of having software-based simulators that overcome the limitations of physically deployed smart spaces. Moreover, simulation enables researchers to evaluate scenarios and applications without the difficulties in dealing with hardware sensors and actuators. It also offers a greater flexibility since it is easy to run a set of simulations with a wide range of parameters.

In this paper, we propose a framework to simulate smart spaces of various kinds (house, building, city, open areas ...). Each device (sensor or actuator) provides services (data like presence, temperature, humidity, location, luminosity… or action: acting on the state of a device, moving objects...). It is possible to add, remove or modify the deployed devices (and therefore the associated services), easily and dynamically even at run-time. So it allows to provide a huge flexibility and openness to test new algorithms or middleware for IoT. The update of the infrastructure of services in the simulation must then verify two points: (a) the ease for developers to create or modify the simulated environment and (b) at run-time, new devices deployment must be done in a timely fashion in order to preserve the smart spaces simulation responsiveness.

II. Overview of the Proposed Framework

A. Architecture

The richness of a simulated smart space comes from the variety and the number of provided devices. So, the amount of work to be produced by the developer to add a device to a virtual object should be as less as possible. This is, for instance, one of the main drawback of UbiReal[1] [3]. The use of services associated to virtual devices allows loose coupling between the entities in the simulated world and the functionalities provided by any kind of client. This corresponds to the notion of Service on Device or Web Service for Device (WSD) [4]. A WSD can be defined as a set properties or variables used to define useful values associated to a device or a specific service of the device (the state of a light for instance) and a set of actions to be performed on the device (switching the light on or off). Events can be generated each time a variable's value changes. There is also the possibility to discover services associated to a device. WSD implementation is based on UPnP protocol. Using this protocol is also interesting because UPnP offers research and discovery mechanisms in addition to the request-response and eventing mechanisms. It's also possible to use more recent protocols like DPWS[2] providing the same type of characteristics.

With this approach, one or more service descriptions can be associated to a 3D virtual device hence becoming a WSD, attached to a 3D shape, and published outside the simulation

[1] http: //ubireal.org/ (release 1.0, published on Sept 28, 2012)

[2] http: //docs.oasis-open.org/ws-dd/ns/dpws/2009/01 (July 1st, 2009)

environment. The benefit of this approach is its ability to externalize the definition and the management of all the virtual objects intrinsic functionalities. Therefore, the interconnection of services is not managed inside the virtual framework, an application, for instance, can take care of it by orchestrating the services externally to the virtual environment. This is helpful to avoid the developer to code the behaviors within the virtual environment and allows the utilization of any kind of algorithms or technics to manage the simulated smart space.

To keep the simulated devices independent of the 3D rendering framework and keep the 3D virtual scene rendering framework reusable for any kind of smart space, we do not want to modify it and code the web services associated to the devices directly in it. Therefore, to define a new service associated to a device, a description of the service is created using an XML file. This description defines the variables and actions of the service and specify their interconnection with the virtual device. A library of generic actions on the 3D virtual world has been defined and includes a set of basic functionalities used to manipulate the 3D objects. More complex services for devices can be created from a composition of these basic functionalities. For instance, a traffic light is composed of a red, a yellow and a green lights. The XML description of the service associated to the traffic light will propose actions like switching on or off each light.

The device behavior must not be managed inside the 3D simulator but instead, managed outside. Back to the traffic light example, its behavior might be very different depending on the countries where it is used. If the red light prohibits the traffic, and the green one allows it, the yellow one provides a warning indicating an imminent state change from green to red but, in some countries, also for a change from red to green. Managing this kind of specific behavior outside the 3D scene allows to use the same 3D simulation engine for different kind of simulations. And it avoids to change the simulation behavior inside the 3D environment which is not suitable for reasons previously detailed. So, the service associated to a traffic light is the definition of actions to switch on/off each of the lights. These actions are linked to the 3D shape; actions can be mapped to 3D visual effects. The XML description of a service is used to *automatically* generate the source code corresponding to the functionalities of the device.

B. *Dynamicity of the Simulated Services*

To offer the possibility to modify the infrastructure of the available services in the 3D virtual environment, each of the WSD associated to 3D virtual objects can be started, paused or stopped dynamically. WSD can be activated at the simulation startup or depending on the proximity of the user avatar relative to the 3D virtual object. It allows the simulation of some geo-localized services that are only available inside a specific area (due to the limit of a signal strength for instance).

On the other hand, the WSD are generated by a model transformation from an XML description to the targeted source code used in the simulation framework. This is possible with the use of a managed language offering the possibility to byte compile the source code on the fly and link this new code to the environment or by the use of scripting interpreted languages. In addition, this approach allows to dynamically add new services, and create new WSD during the execution of the simulation and then add new functionalities on the fly.

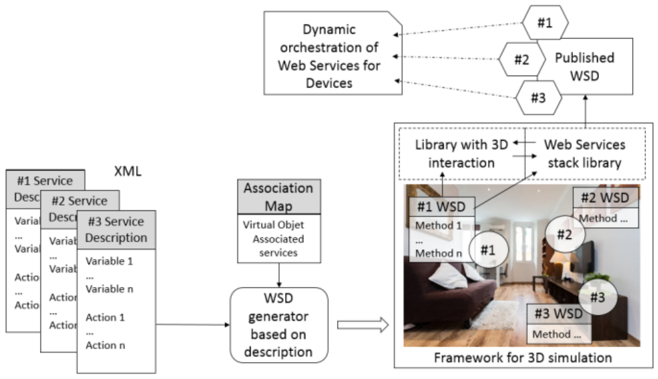
Dynamic orchestration of Web Services for Devices
#1
#2
#3
Published WSD
Library with 3D interaction
Web Services stack library
XML
#1 Service Desc
#2 Service Desc
#3 Service Description
Variable 1
Variable n
Action 1
Action n
Association Map
Virtual Objet Associated services
WSD generator based on description
#1 WSD
Method 1
Method n
#2 WSD
Method ...
#3 WSD
Method ...
Framework for 3D simulation

Fig. 1. UbiUnity Implementation

This implies to generate, activate and deactivate the WSD during the execution of the simulator in order to adapt the virtual environment to the needs. But this adaptation must be done within a time controlled and constrained adaptation loop in order not to affect the responsiveness of the simulation and, in our case, the fluidity of the graphics rendering (the number of frames per second). Moreover it is not possible to predict the number of available services (which depends on the user's displacements and on the specified services that can be dynamically added to the virtual environment). To evaluate the framework, we measured the graphics rendering engine responsiveness as a function of the amount of concurrently activated and/or deactivated services. With the UPnP library used, it is more efficient to instantiate several UPnP devices at the same time (within the same frame) rather than starting them one by one. The average time to create and start a new UPnP device is about 20ms. This time grows up to more than 100ms for the 250th UPnP device. We started up to 250 UPnP devices created three by three and create an algorithm for the framerate to never drop down under 30fps (the default framerate for rendering the scene without any instrumentation was 60fps).

REFERENCES

[1] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, et A. Oliveira, « Smart Cities and the Future Internet: Towards Cooperation Frameworks for Open Innovation », in *The Future Internet*, J. Domingue, A. Galis, A. Gavras, T. Zahariadis, D. Lambert, F. Cleary, P. Daras, S. Krco, H. Müller, M.-S. Li, H. Schaffers, V. Lotz, F. Alvarez, B. Stiller, S. Karnouskos, S. Avessta, et M. Nilsson, Éd. Springer Berlin Heidelberg, 2011, p. 431‑446.

[2] S. Haller, « The things in the internet of things », in *Poster at the (IoT 2010). Tokyo, Japan, November*, Tokyo, Japan, 2010, vol. 5, p. 26.

[3] H. Nishikawa, S. Yamamoto, M. Tamai, K. Nishigaki, T. Kitani, N. Shibata, K. Yasumoto, et M. Ito, « UbiREAL: realistic smartspace simulator for systematic testing », in *UbiComp 2006: Ubiquitous Computing*, 2006, p. 459–476.

[4] F. Jammes, A. Mensch, et H. Smit, « Service-oriented Device Communications Using the Devices Profile for Web Services », in *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing*, New York, NY, USA, 2005, p. 1–8.

[5] A. Pattrasitidecha, « Comparison and evaluation of 3D mobile game engines », Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden, Master Thesis, févr. 2014.