A Dynamic Visual Simulation Environment for Internet of Things

Stéphane Lavirotte, Jean-Yves Tigli, Gérald Rocher, Léa El Beze, Adam Palma

LABORATOIRE

INFORMATIQUE, SIGNAUX ET SYSTÈMES DE SOPHIA ANTIPOLIS
UMR7271

A Dynamic Visual Simulation Environment for Internet of Things

Stéphane Lavirotte, Jean-Yves Tigli, Gérald Rocher, Léa El Beze1, Adam Palma

EQUIPE SPARKS

Rapport de Recherche

08-2015

Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis (I3S) - UMR7271 - UNS CNRS 2000, route des Lucioles - Les Algorithmes - bât. Euclide B 06900 Sophia Antipolis - France http://www.i3s.unice.fr

A Dynamic Visual Simulation Environment for Internet 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

[1] ***Abstract*— The development of living labs or 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 Internet of Things (IoT). 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 which allow the researcher 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 graphical rendering.**

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

Introduction

Many initiatives aim at implementing smart spaces (Living Labs[1] for instance or the Smart Cities initiatives) [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, development and deployment of such smart spaces, which are real physical systems, are complex engineering tasks. The creation of physical spaces equipped with sensors and actuators, like Living Labs or Smart Cities, have drawbacks:

- *Huge investments*: the financial cost of purchase and deployment of such equipment is important.
- *Limited variety of sensors and actuators*: sensors and actuators deployed in these environments are limited in number and type of addressable devices. In any case, they are limited to the products available on the market; so it is difficult to test prospective approaches based on non-available materials.
- *Updates and upgrades are complex and limited*: updating or upgrading hardware and software configurations of the deployed infrastructure is a procedure that is often impossible or, at best, will be expensive and very costly in time; moreover, using equipment outside of their initial purpose is complicated from an engineering standpoint of view.
- *Limited dynamics of the environment*: the dynamicity in these smart spaces comes from people who inhabit them via any wearable device with which they arrive or via interactions they have with the smart space. Meanwhile, the core infrastructure once defined and deployed, do not change at all or with very small variations. But it is still necessary to test different configurations for the infrastructure to be deployed in the smart space.
- *Scalability of deployment*: if limited smart areas can be maintained more easily (such as smart home for instance), the scalability to a building or a city requires to make larger studies before deploying the infrastructure.

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 allow easy experimentation. Therefore, there are numerous advantages at/of having software-based simulators that overcome the limitations of physically deployed smart spaces. Simulation-based approaches are widely used in many fields. For instance, in the field of embedded systems, hardware components are fully simulated. Thus, developed systems can be fully tested before their manufacturing and the simulator facilitates and

[1] http: //openlivinglabs.eu/, http: //livinglabs.mit.edu/

accelerates the development of software applications. 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.
Differences between Simulation, Living Lab and Smart Space approaches are summarized in the next figure along with their main advantages and disadvantages.

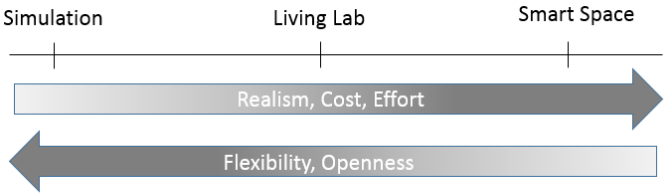

Fig. 1. Simulation, Living Lab and Smart Space comparaison

In this paper, we propose an environment to simulate smart spaces of various kinds (house, building, city, open areas ...). Each device (sensor or actuator) in the smart space 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 this allows to simulate over long periods of time and provides 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 quality.
First, we identify the main features that must have such systems. Then, we study the related works before presenting the proposed simulator and the associated performance assessments results aimed at characterizing the graphic rendering engine fluidity under stress conditions and at highlighting the limitations. We conclude by presenting the perspectives of this work.

Motivations and Criteria

The need to simulate sensors and actuators to evaluate and test mobile and context-aware services [3] first appeared in 2001. *The study of previous works in this domain and the current challenges in simulating environments for IoT lead us to define some criteria. We present them in this section.*

Services in smart spaces are often localized and contextualized. This implies (a) a need to have a 2D or 3D representation of the surrounding environment where the devices are located. The realism and the quality of the simulation are mainly driven by parameters like graphics quality, real time computing, physics engine integration, etc… In the context of IoT, the main requirement of such simulators is their ability to indifferently simulate indoor environments (apartments, offices, museums, hospitals, schools, malls, university campus…) or outdoor areas. It is also crucial that the task of (b) integrating devices and associated services in the simulated scene does not require the IoT developers to learn any 3D engine coding specificities. This is a key point for them to easily develop, use and share a wide variety of simulations. Moreover, when a specific 3D environment is not needed, the use of already developed 3D scenes, for testing purposes, would save time and energy. It means that a (c) loose coupling approach between the 3D representation and the simulated infrastructure will ensure the reusability of 3D environments and developed services. And, by considering the continuous evolution of the 3D engines capabilities, a good simulator architecture should also allow to easily change or upgrade the 3D rendering engine used.
Pervasive systems must offer an open, extensible and evolving set of services corresponding to available devices providing those services in the surrounding environment. It means that (d) a smart spaces simulator must have the capabilities to add, remove and modify the devices deployed in the simulated environment and/or the services attached to the devices. Moreover, (e) some of the services associated to devices must be enabled depending on the localization of the user; this correspond to the discovery of new services as the users get close to the device. And a last functionality would be to (f) allow services to be added, removed or modified in the virtual environment, at runtime. This would allow over long periods of time, to adapt the simulated environment without restarting the simulation or to directly observe the consequences of a modification in the services infrastructure.
We can summarize all the defined criteria as follow:

a. A performant graphic rendering engine with a physics engine to visualize and interact with the simulated environment.
b. Ease of adding new devices in the simulated environment without the need to modify the 3D engine or develop code.
c. Loose coupling between the rendering engine and the simulated services.
d. Capability to add, remove or modify the sensors or actuators deployed in the environment.

e. Activating services based on the user location enabling the discovery of new services as the user moves in the simulated environment (context-awareness applications).

f. Modification of simulated devices and services at runtime while maintaining the rendering process as fluid as possible.

In the next section, previous works on the subject are reviewed and challenged based on the aforementioned criteria.

Related Works

Several projects worked on the simulation of ambient or ubiquitous applications in the context of Internet of Things. In this section, we present the main works and underline their limitations compared to the previously defined criteria. All the presented works in this section address the problem of simulating ubiquitous environments through a rendering engine to visualize it (a) using 3D for most of them and sometimes 2D representations [4].

QuakeSim [3] aimed to evaluate, test and demonstrate context-aware services. It was the first environment for testing mobile services. Few sensors (altitude, temperature and position) were hardcoded in the virtual environment. UbiWize [5] targeted the development of hardware and embedded software in an ubiquitous computing context. UbiWize is the union of UbiSim, which aimed to produce real-time context information in a semi-realistic environment, and Wise, consisting in simulating device with a 2D display. UbiWize thus focused on emulating device being immersed in a 3D environment rather than developing a simulator for context-awareness capabilities (focused on services nearby a character position). Reference [6] presents the design of a generic simulation tool for many scenarios for ubiquitous computing. They provide a layered environment, a kind of middleware between the simulator API and the application, which is less flexible than a service oriented approach. These researches did not focus on the way to publish to third parties the capabilities of the simulated environment as services. This leads to pair the rendering and the behavioral code.

The decoupling between the 3D simulation environment and the behavior of the simulation was introduced by Tatus. Tatus [7] is an ubiquitous computing simulator interfaced with a testbed for wireless communication domain simulation. The authors modified a game engine (Half-Life) to provide a convenient computing environment that researchers can use to test ubiquitous computing applications. But enhancement of the environment is done inside the game engine even if they provide an interface to avoid developing game level code. A system-under-test (SUT) is separated from the game engine and can communicate with it via network communication. But it's not possible to interact with every entities of the simulated environment so it does not consider device simulation. 3DSim [8], UbiReal [9], DiaSim [10] and 3D DEIR [11] proposed to associate services to virtual devices to reach criteria (c). Even if 3DSim, DiaSim and 3D DEIR allow simulated services to be added dynamically, as the simulation of a pervasive computing system runs, there is no evaluation of this criteria to define the number of possible services regarding the continuity of the simulation quality.

However, these approaches are limited because they require significant programming effort to address new pervasive computing challenges. As defined in [12], we can identify three categories of users with different points of view: (1) a developer who extends the simulation, (2) a researcher who configures the simulation for testing purposes and (3) a user who runs the simulation. These persona are not necessarily three different persons but each of them need to exhibit specific skills. Moreover, the developer profile could be divided more precisely: developer coding for extending the virtual environment, and developer dealing with the addition of new protocols or functionalities. To facilitate the development of various simulations, the proposed environment aim at minimizing the needed developments to create a simulation. All the studied solutions (QuakeSim, UbiWize, 3DSim, Tatus, UbiReal, ISS) require to program within the environment of the visual rendering tool used to simulate the ubiquitous environment. This implies a great effort of programming for the developer or the researcher to learn the API and the specificities of a dedicated framework. Only DiaSim proposes a dedicated framework to program specific scenarios, but it requires a specialized developer to create new simulation and it’s an obstacle to create different situations.

With a clear separation between the rendering engine, a way to associate the simulated devices and services and to specify the services and the interaction with the 3D world, it would be possible to specialize the work to dedicated skilled persona. The simulator provides a graphical representation of a virtual environment. As the developers of ambient simulation are not often 3D graphical designers, it is possible to delegate the manufacturing process of the virtual environments to specialists. It’s also possible to find prebuild virtual environments or 3D virtual objects on markets that can be purchased. The developer can then focus on defining

the devices and the provided services to be attached to the virtual devices. Moreover, if it would be possible to limit the programming to create a new simulation, researchers could concentrate on configuring it.
We can summarize the contributions of the studied works against the defined criteria in the following table.

	a	b	c	d	e	f
QuakeSim	Yes	No	No	No	No	No
UbiWize	Yes	No	No	No	No	No
3DSim	Yes	No	Yes	Yes	No	Partial
Tatus	Yes	No	Yes	No	No	No
UbiReal	Yes	No	Yes	No	No	No
ISS	Yes	No	Yes	No	No	No
DiaSim	Yes	Partial	Yes	Yes	No	Partial
3D DEIR	Yes	No	Yes	Yes	No	Partial

This study emphasizes that criteria (b: specifying new service behavior without coding in the virtual environment framework) and (e: activating services depending on the user location in the virtual world for simulating the context-awareness) were not addressed by the previous contributions. Criteria (f: changing dynamically the devices and services in the virtual environment while maintaining the rendering process as fluid as possible) was introduced in 3DSim, DiaSim and 3D DEIR but without any evaluation. In the next section, we will present the proposed framework giving answers to these criteria and all the other defined ones.

Overview of the Proposed Framework

Criteria defined in the previous section allow us to propose the architecture of a simulator for IoT.

Architecture of UbiUnity

The richness of a simulated smart spaces comes from the variety and the number of provided devices. So, to add a device to a virtual object, the quantity of work to be produced by the developer should be as less as possible (criteria b and d). This is, for instance, one of the main drawback of UbiReal[2] [9]. To add a service to a device in the virtual smart environment, the developer must produce a large amount of code. First of all, a new UPnP device has to be developed and imported it in the virtual space of UbiReal. Then the 3D shape has to be designed and the interaction between the UPnP device and the 3D shape has to be specified. Finally, the simulation dependent part, which consists in the implementation of the Importable class, has to be implemented. To include all these developments in the UbiReal environment, the code finally has to be compiled and linked to the simulator. Finally, some manipulations with the graphical user interface are needed to register the classes in the virtual space before using it. All the work needed to add a new virtual smart device in the environment is an obstacle with regards to the number of possible devices.
To tackle the criteria (a), (b), (c) and (d) previously defined, we propose the following architecture for the simulator and the definition of services associated to the virtual devices.

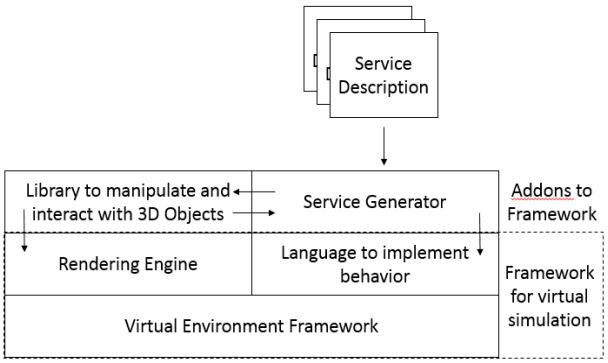

Fig. 2. Architecture of UbiUnity

The use of services associated to virtual devices allows loose coupling between the entities in the simulated world and functionalities provided by any kind of client. This corresponds to the notion of Service on Device or Web Service for Device [13]. A web service for device 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.
With this approach, one or more service descriptions can be associated to a 3D virtual device which becomes a web service for device, attached to the 3D shape, and published outside the simulation environment. Any client can then access the virtual device to interact with it and then access and interact with the simulated world. It permits the loose coupling between the virtual 3D environment and the services associated to the device. The benefit of this approach is the ability to externalize all the functionalities given to the virtual objects. So, the interconnection of services is not done inside the virtual framework, but can be done by, for instance, orchestrating the services externally to the virtual environment. This is helpful to avoid to code any

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

behavior in the virtual environment and allows to use any kind of algorithms or technics to manage the simulated smart space.
But as we mentioned earlier, we do not want to code the web services associated to devices in the framework managing the 3D virtual scene. This correspond to address criteria (d). A researcher could be able to create or configure a simulated environment without having to implement any code inside the simulator. As we mentioned in the related works section, it's really important to separate the expertise of each persona.
To define a new service associated to a device, we propose to create a description of the service using an XML file. This description must define the variables and actions of the service and specify the interconnection of the action or the variable with the virtual device. A library of generic actions on the 3D virtual world has been defined. For instance, basic functionalities are to move, rotate or translate an object or switch on or off a light in the 3D environment. A set of functionalities used to manipulate the 3D objects is provided in the library as well. To be able to create more complex services for devices, it's possible to compose these basic functionalities to create new ones. 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. Thus a more complex device is a composition of basic functionalities proposed by the library.
The description of a service is used to generate the source code corresponding to the functionalities of the device. The behavior of the device must not be included in the simulator. The goal is to manage this behavior outside the 3D simulator engine which is an advantage. For the traffic light example, the behavior of a traffic light is very different depending the countries. If the red light prohibits the traffic, and the green one allows it, the yellow one provides a warning that the signal will 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 scene allows to use the same 3D simulation engine for different kind of simulation for different countries. And it avoids to change the simulation behavior inside the 3D environment which is not suitable. So, the service associated to a traffic light is the definition of actions to turn on each light. These actions must be linked to the 3D shape; turning on a light will change the texture of a part of the mesh and turn on a spot light for better realism. But there is no implementation of any behavior between the lights.

Dynamicity of Simulated Services

To offer the possibility to modify the infrastructure of available services in the virtual environment, each of the web services for device associated to virtual objects can be started, paused or stopped dynamically. Web services for devices can be activated at the startup of the simulation or depending the proximity of the users. It allows to simulate some geo-localized services that are only available in a specific area (due to the limit of a signal strength for instance). So their activation depends on the location of the virtual user; the proximity of two entities activates a new web service for device.
On the other hand, as explained in the previous section, any web service for device is generated by a model transformation from an XML description to the target 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 also allows to dynamically add some new services, and create new web services for device during the execution of the simulation to add new functionalities on the fly.
This implies to generate, activate and deactivate the web services for device during the execution of the simulator in order to adapt the virtual environment to the needs. But this adaptation must be done with a time controlled and constrained adaptation loop in order not to affect the quality of the simulation and, in our case, the fluidity of the graphics rendering (the number of frames per second). Moreover it's not possible to predict the number of the available services (which depends on the user's displacement and on the specified services that can be dynamically added to the virtual environment). There is a need to model the type and the number of modifications that can be supported by the simulator: how many web services for device can be activated concurrently without slowing down or freezing the graphical rendering process.

Implementation of UbiUnity

To implement the web services for device, we used the UPnP protocol which has been also used in some previous works: 3DSim [8], UbiReal [9] and 3D DEIR [11]. Using this protocol is also interesting because UPnP offers a research and discovery mechanism in addition to the request-response and eventing mechanisms. It's also possible to use more recent protocols like DPWS[3] providing the same type of

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

characteristics.

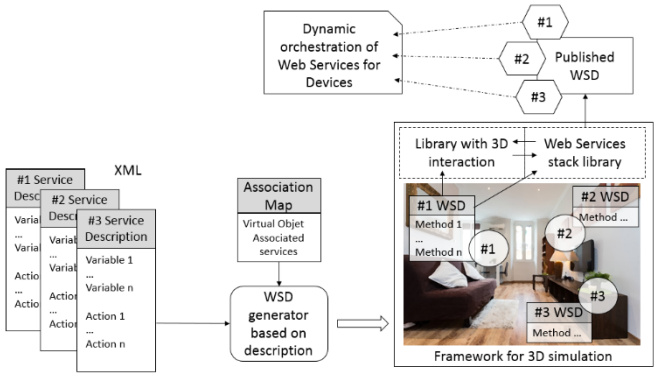

Fig. 3. Implementation of UbiUnity

The service actions defined in the XML file must be bound to the 3D shape in order to trigger the corresponding effect in the virtual environment. To prevent having to code within the virtual environment framework, we developed a library providing a set of basics actions like: moving, scaling any virtual objet (the shape associated to it); modifying its color, texture; playing a media (sound and video along with being able to play, pause, stop and modify the format of loaded media) to offer audio and video services to virtual devices; defining light (type, color, characteristics, …); detect collision between shapes, etc. In the actual release of the library, three main categories are implemented: light, audio, video and object-body. Each category is implemented by a class that implement generic and common interfaces. For instance, it's possible to amplify a sound or the luminosity of a light. This means that the light and audio implement the *IAmplifier* interface.

This also allows to create new kind of devices by composing the available functionalities defined in the library. Moreover, a service is associated to the camera to enable its control via a service and give the possibility to connect to that service any kind of controller (mouse, joystick, trackpad, tablet…). As the services orchestration is done outside the virtual environment, a service can be dynamically bind to the camera to control it. A connection between a web service for a virtual and a physical object also allows the possibility to interact with a physical device from its avatar.

Fig. 4. Coupling virtual and physical world throughout Web Services for Devices dynamic orchestration

This library depends on the used virtual environment framework and must be re-implemented in case of changing it. But, since the services are bound to the 3D environment thanks to a loose coupling mechanism, there is no need to redevelop all the specified services.

Experiments

To evaluate the proposed framework, we measured the graphical rendering fluidity as a function of the amount of concurrently activated and/or deactivated services. To have a reference for this evaluation, we first measured the performances of the library used for implementing web service for device.

Performances Evaluation of the Web Service for Device used library

First of all, and since it is used in our library, we first evaluated the performances of the C# Intel UPnP library. These tests were made outside any 3D rendering environment. The total number of devices that can be instantiated are limited to about 250 devices. This limitation seems to be the consequence of a problem in the Intel UPnP library and was encountered on both .NET frameworks (Mono and Windows) environments. So our tests were limited to 250 UPnP devices instances. For testing purposes, we defined a regular UPnP device with 1 service, 3 variables and 6 methods.

We measured UPnP device class instantiation (Fig. 5) and start method (Fig. 6) execution times on a device. We only present here the evaluations made with mono 2.6 which is the one used in Unity (framework used to implement the proposed solution). For the experiment, devices were created 1 by 1, 2 by 2 and 5 by 5, and the rendering process performances degradations measured for each case.

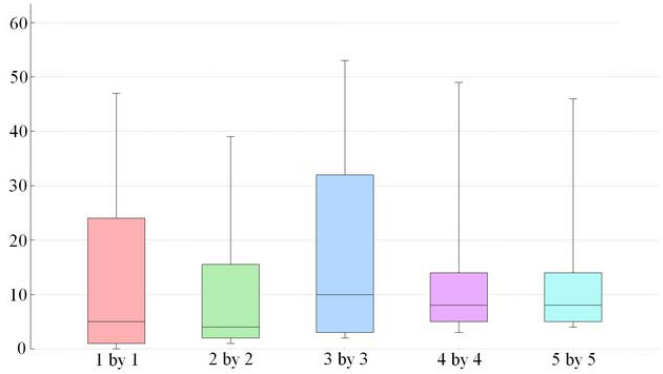

Fig. 5. UPnP Device creation time in milliseconds

The time necessary to create UPnP devices one by one or five by five are equivalent. UPnP device creation time mean value is about 15ms and standard deviation is between 12ms and 17ms. Therefore, there is no significant difference in creating one or more UPnP device at the same time (during the same rendering frame).

We also evaluated the start method execution time. This method is called as soon as a new web service for device bound to a 3D shape is discovered.

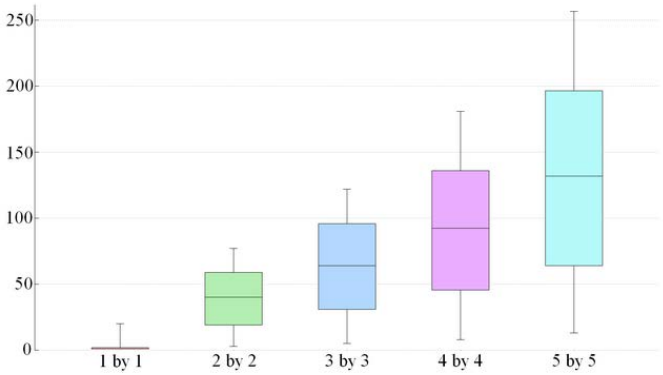

Fig. 6. UPnP device startup time in milliseconds

We measured that starting up a UPnP device takes about 5ms (with a standard deviation of about 10ms) for single device, but if we start up five UPnP devices during the same frame, the time spent grows up to 130ms.

So, we can conclude that with the library used for implementing the UPnP web services for device, it's more efficient to instantiate several UPnP devices at the same time (during 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.

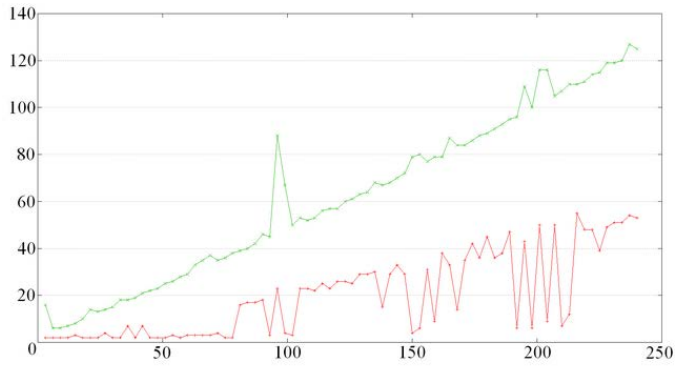

Fig. 7. Total time for creating (red) and starting (green) up to 250 UPnP devices

A deeper analysis shows that the total amount of time spent at creating (red plot) and starting (green plot) a new UPnP device increases with the total number of created devices (Fig. 7). These behavior are inherent to the garbage collector algorithm used in mono 2.6 release (Boehm-Demers-Weiser GC). At each new allocation, the garbage collector processes all the allocated memory to eventually free up space. The more memory will be allocated, the more time will be necessary to instantiate a new object. And all the allocations are not necessarily done during the creation process but also during the start method execution. We could conclude that it would be preferable not to use a managed code language, but this approach allows to introduce the dynamicity wanted for the services (the possibility to create new ones at runtime).

We observed the same results when using the graphical rendering environment with a small overhead due to the framework itself.

Graphical Rendering Evaluation

As the UPnP devices can be started and stopped during the simulation, we measured the impact of these actions on the graphical rendering fluidity by measuring the framerate (number of frames per second). We evaluated it (red plot) as a function of the amount of UPnP devices started (green plot). We started up to 250 UPnP devices created three by three and the framerate never dropped down under 30fps (the default framerate of the scene without any instrumentation was 60fps).

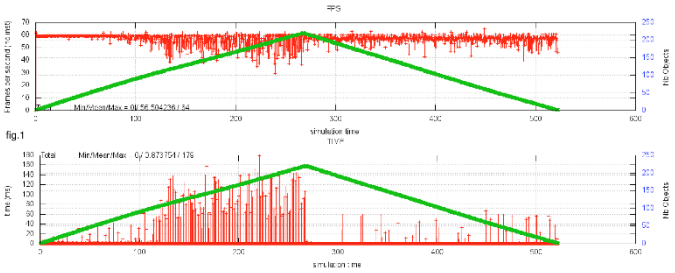

Fig. 8. Evaluation of the FPS when starting and stoping UPnP services

We used Unity[4] 4.5.5 to provide the virtual environment framework [14]. Unity is a cross-platform game engine that offers a variety of tools to create and manage virtual worlds. It proposes to attach "scripts" to virtual objects to create graphical effects, control the physical behavior of objects… These scripts can be specified in JavaScript or C#. We used the Intel C# implementation of UPnP to enable this feature in the Unity environment.

For evaluation purposes, we used a PC with an Intel Pentium B970 2.3GHz with 4Gio of RAM and an AMD Radeon HD 5800 graphical card and one active network interface. The evaluation was made under Windows 7 SP1.

Dynamicity of Web Services for Device Modelisation

The time for creating and starting up UPnP devices is not constant over the time. We measured that, created three by three, the frame rate never dropped down under 30 fps. But as we can see on Fig. 8, we could start more devices during for the first hundred ones without dropping down a give limit. We implemented an algorithm in the library to maximize the number of stated UPnP devices based on the model of reactivity. This allows us to have a minimum framerate for the simulation even activating UPnP Device.

Moreover, for the tests purposes, we also used WComp [15] to dynamically orchestrate the exposed services by the virtual environment. WComp also includes a temporal model at runtime [16] to ensure that the new services can be integrated in an application defining the behavior in a timely fashion. It allowed us to simulate virtual environments like a house or a city district.

Conclusion

In this article, we argued that simulation still have advantages compared to physical testbeds: flexibility and openness. Moreover, we can identify a paradox: living labs or smart spaces are more realist but it's more difficult to equip the real environment with enough sensors to get enough user experimental results. By contrast, it's easier to simulate devices in virtual environments and to monitor the way to use them even if we have less restitution fidelity. We presented UbiUnity, a 3D environment for simulating devices and provided services immersed in virtual situations. Each instrumented virtual device proposes services with the help of web service for devices approach. The main contribution of the present work is the identification and study of two main drawbacks in previous works on that subject: (1) the possibility for the researcher to define new devices and test new behaviors without coding in the graphical engine, (2) to allow the dynamicity of activating web services for devices at runtime with respect to the fluidity of the simulation. We measured the impact of dynamically activating web services in the 3D environment. The use of web services for device attached to virtual devices offer functionalities outside the environment that can be used for dynamic orchestration of these services. It could also allow to couple the interface of physical and virtual device throw the services and offer the possibility to interconnect virtual environment to distant physical site for remote monitoring or remote interaction for instance.

[4] http: //unity3d.com/

ACKNOWLEDGMENT

We would like to thanks Fabrice Agagah, Camille Yacoub, Sergio Baudino and Ely Bezeid Moulaye who are currently working for a public release of the simulator.

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] M. Bylund et F. Espinoza, « Using quake III arena to simulate sensors and actuators when evaluating and testing mobile services », in *CHI'01 Extended Abstracts on Human Factors in Computing Systems*, 2001, p. 241–242.

[4] T. Van Nguyen, J. G. Kim, et D. Choi, « ISS: The Interactive Smart home Simulator », in *11th International Conference on Advanced Communication Technology, 2009. ICACT 2009*, 2009, vol. 03, p. 1828‑1833.

[5] J. J. Barton et V. Vijayaraghavan, « UBIWISE, a simulator for ubiquitous computing systems design », *Hewlett-*

Packard Lab. Palo Alto Â AI HPL-2003-93, 2003.
[6] V. Reynolds, V. Cahill, et A. Senart, « Requirements for an ubiquitous computing simulation and emulation environment », in *Proceedings of the first international conference on Integrated internet ad hoc and sensor networks*, Nice, France, 2006, p. 1.
[7] E. O'Neill, M. Klepal, D. Lewis, T. O'Donnell, D. O'Sullivan, et D. Pesch, « A testbed for evaluating human interaction with ubiquitous computing environments », in *First International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities, 2005. Tridentcom 2005*, 2005, p. 60‑69.
[8] A. A. Nazari Shirehjini et F. Klar, « 3DSim: rapid prototyping ambient intelligence », in *Proceedings of the 2005 joint conference on Smart objects and ambient intelligence: innovative context-aware services: usages and technologies*, 2005, p. 303–307.
[9] 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.
[10] J. Bruneau, W. Jouve, et C. Consel, « DiaSim: A parameterized simulator for pervasive computing applications », in *Mobile and Ubiquitous Systems: Networking & Services, MobiQuitous, 2009. MobiQuitous' 09. 6th Annual International*, 2009, p. 1–10.
[11] K. I.-K. Wang, I. Y.-H. Chen, W. H. Abdulla, Z. Salcic, et B. C. Wunsche, « 3D virtual interface for ubiquitous intelligent environments », 2007, vol. 2007, p. 268‑275.
[12] J. J. Barton et V. Vijayaraghavan, « Ubiwise, a ubiquitous wireless infrastructure simulation environment », *HP Labs*, 2002.
[13] 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.
[14] A. Pattrasitidecha, « Comparison and evaluation of 3D mobile game engines », Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden, Master Thesis, févr. 2014.
[15] N. Ferry, V. Hourdin, S. Lavirotte, G. Rey, et J.-Y. Tigli, « WComp, Middleware for Ubiquitous Computing and System Focused Adaptation », in *Computer Science and Ambient Intelligence*, ISTE Ltd and John Wiley and Sons, 2013, p. 89‑120.
[16] J.-Y. Tigli, S. Lavirotte, G. Rey, N. Ferry, V. Hourdin, S. F. B. Abdenneji, C. Vergoni, et M. Riveill, « Aspect of Assembly: From Theory to Performance », *Trans. Asp.-Oriented Softw. Dev. IX*, p. 53‑91, 2012.