IoT Component Design and Implementation using Discrete Event Specification Simulations

Souhila Sehili
University of Corsica
SPE UMR CNRS 6134
Corte, France
Sehili@univ-corse.fr

Laurent Capocchi
University of Corsica
SPE UMR CNRS 6134
Corte, France
capocchi@univ-corse.fr

Jean-François Santucci
University of Corsica
SPE UMR CNRS 6134
Corte, France
santucci@univ-corse.fr

Abstract— **The Internet of Things (IoT) approach enables rapid innovation in the area of internet connected devices and associated cloud services. An IoT node can be defined as a flexible platform for interacting with real world objects and making data about those objects accessible through the internet. Communication between nodes is discrete event-oriented and the simulation process play an important role in defining assembly of nodes. In this paper, we propose the definition of a modeling and simulation scheme based on a discrete-event formalism in order to specify at the very early phase of the design of an ambient system: (i) the behavior of the components involved in the ambient system to be implemented; (ii) the possibility to define a set of strategies which can be implemented in the execution machine. The DEVSimPy environment is then used to implement the example of a switchable on/off lamp.**

Keywords-DEVS; IoT; formalism; assembly; strategies.

I. INTRODUCTION

Technological advances in recent years around mobile communication and miniaturization of computer hardware have led to the emergence of ubiquitous computing. Computing tools are embedded in objects of everyday life. The user has at its disposal a range of small computing devices, such as Smartphone or PDA (Personal Digital Assistant), and their use is part of ordinary daily life. The definition of such complex systems involving sensors, smartphone, interconnected objects, computers, etc., results in what is called ambient systems. One of todays challenges in the framework of ubiquitous computing concerns the design ambient complex systems. One of the main problems is to propose a management adapted to the composition of applications in ubiquitous computing. The difficulty is to propose a compositional adaptation which aims to integrate new features that were not foreseen in the design, remove or exchange entities that are no longer available in a given context. Mechanisms to address this concern must then be proposed by middleware for ubiquitous computing. Several kinds of middleware tools have been proposed in the recent years [2]. We have being focused on the WComp environment. WComp is a prototyping and dynamic execution environment for Ambient Intelligence applications. WComp [2] is created by the Rainbow research team of the I3S laboratory, hosted by University of Nice - Sophia Antipolis and CNRS. It uses lightweight components to manage dynamic orchestrations of Web service for device, like UPnP (Universal Plug and Play), discovered in the software infrastructure. In the framework of the WComp, it has been defined a management mechanism allowing extensible interference between devices. In order to deal with the asynchronous nature of the real world, WComp has defined an execution machine for complex connections.

In this paper, we propose the definition a modeling and simulation scheme based on the DEVS formalism in order to specify at the very early phase of the design of an ambient system: (i) the behavior of the components involved in the ambient system to be implemented; (ii) the possibility to define a set of strategies which can be implemented in the execution machine. The interest of such an approach is twofold: (i) the behavior will be used to write the methods required in order to code the components using WComp environment; (ii) to check the different strategies (to be implemented in the execution machine) before implementation.

The rest of the paper is as follows: Section II concerns the background of the study by presenting the traditional approach for the design IoT systems. It briefly introduces a set of middleware framework before focusing on the WComp Framework. The DEVS formalism and the DEVSimPy environment are also presented. In Section III, the proposed approach based on the DEVS formalism is given. An overview of the approach as well as the interest in using DEVS simulation is detailed. Section IV deals with the validation of the approach through a case study The conclusion and future work are given in Section V.

II. BACKGROUND

A. IoT Design and WComp.

The ubiquitous computing is a new form of computing that has inspired many works in various fields such as the embedded system, wireless communication, etc. Embedded systems offer computerized systems having sizes smaller and smaller and integrated into objects everyday life. An ambient system is a set of physical devices that interact with

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-371-1

each other (e.g., a temperature sensor, a connecting lamp, etc.). The design of an ambient system should be based on a software infrastructure and any application to be executed in such an ambient environment must respect the constraints imposed by this software infrastructure.

Devices and software entities provided by the manufacturers are not provided to be changed: they are black boxes. This concept can limit the interactions to use the services they provide and prevents direct access to their implementation. The creation of an ambient system can not under any circumstances pass by a modification of the internal behavior of these entities but simply facilitate the principle reusability, since an entity chooses for its functionality and not its implementation. In the vision of ubiquitous computing, users and devices operate in an environment variable and potentially unpredictable in which the entities involved and appear conveniently disappear (a consequence of mobility, disconnections, breakdowns, etc.). It is not possible to anticipate what the design will all devices that will be available and when. As a result a set of tools have been interested in developing software infrastructure allowing the design of applications with the constraint unpredictability availability of component entities [2].

In this paper, we deal with the WComp framework, which is used in order to design ambient systems. The WComp architecture is organized around containers and designers. The purpose of containers is to take over the management of the dynamic structure such that instantiation, destruction of components and connections. An application is created by a WComp component assembly in a container, according to LCA (Lightweight Component Architecture) [2]. WComp allows to implement an application from an orchestration of services available in the platform and/or other off-the-shelves components.

Whatever the tool which may be used, the design of a IoT component leans on the definition of:

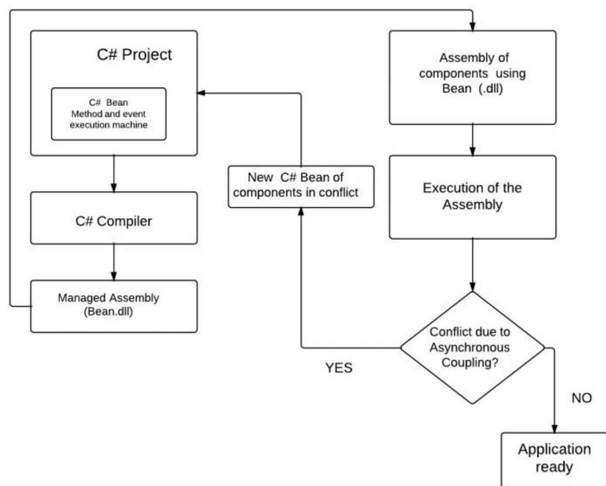

Figure 1. Traditional IoT component design.

- A set of methods allowing to describe the behavior of the component;
- The execution machine associated with the considered component.

The design of ambient computing systems involves a technique different from those used in conventional computing. Applications are designed dynamically by ”smart” devices (assembly components) of different nature. The construction of an ambient IoT system requires the definition Figure 1 of Methods and an execution engine.

The Designer runs the Container for instantiation and for the removal of components or connections between components in the Assembly which has to be created. A component belonging to the WComp platform is an instance of the Bean class implemented in the object language (C#). The description of a given execution machine has to be defined manually using methods for the management of events usually based on automata theory. Figure 1describes the traditional way to design an ambient system using WComp. The behavior and the components involved in the ambient system as well as the Bean classes describing the execution machine are coded using the C# language (C# rectangle in Figure 1).

The compilation allows to derive the corresponding binary files (dll) of the Bean classes involved in the resulting Assembly. The Assembly can then be executed. Conflicts are checked: if conflicts (generally due to asynchronous couplings) are detected the designer has to write a new behavior of the execution machine by recoding Bean classes in order to solve the coupling conflicts while if no conflict are detected the application is ready.
In this paper, we choose the proposed a new approach for a computer aided design of ambient systems using the DEVS formalism by developing DEVS simulation concepts and tools for the WComp platform. The goal is to use the DEVS formalism and the DEVSimPy framework in order to perform DEVS modeling and simulations: (i) to detect the potential conflicts without waiting to implementation and execution phases as in the traditional approach of Figure 1; (ii) to offer the designer to choose between different executions strategies and to test them using DEVs simulations; (iii) to propose a way to automatically generate the coded of the methods involved in the execution machine strategies. The DEVS formalism and the DEVSimPy environment are briefly introduced in the next two sub-sections while the proposed approach is introduced in Section III.

B. The DEVS formalism

Since the seventies, some formal works have been directed in order to develop the theoretical basements for the modeling and simulation of dynamical discrete event systems [8]. DEVS [9] has been introduced as an abstract formalism for the modeling of discrete event systems, and

allows a complete independence from the simulator using the notion of abstract simulator.

DEVS defines two kinds of models: atomic models and coupled models. An atomic model is a basic model with specifications for the dynamics of the model. It describes the behavior of a component, which is indivisible, in a timed state transition level. Coupled models tell how to couple several component models together to form a new model. This kind of model can be employed as a component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion. As in general systems theory, a DEVS model contains a set of states and transition functions that are triggered by the simulator.

A DEVS atomic model AM with the behavior is represented by the following structure:

$$AM = < X, Y, S, \delta int, \delta ext, \lambda, t_a > \quad (1)$$

- X is the set of input values,
- Y is the set of output values,
- S is the set of sequential states,
- δint is the internal transition function dictating state transitions due to internal events,
- δext is the external transition function dictating state transitions due to external input events,
- λ is the output function generating external events at the output,
- t_a is the time-advance function which allows to associate a life time to a given state.

Connections between different atomic models can be performed by a coupled model. A coupled model, tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to hierarchical construction.

C. The DEVSimPy environment

DEVSimPy [1] is an open Source project (under GPL V.3 license) supported by the SPE team of the University of Corsica Pasquale Paoli. This aim is to provide a GUI for the modeling and simulation of PyDEVS [4] models. PyDEVS is an Application Programming Interface (API) allowing the implementation of the DEVS formalism in Python language. Python is known as an interpreted, very high-level, object-oriented programming language widely used to quickly implement algorithms without focusing on the code debugging [6]. The DEVSimPy environment has been developed in Python with the wxPython [7] graphical library without strong dependences other than the Scipy [3] and the Numpy [5] scientific python libraries. The basic idea behind DEVSimPy is to wrap the PyDEVS API with a GUI allowing significant simplification of handling PyDEVS models (like the coupling between models or their storage).

DEVSimPy capitalizes on the intrinsic qualities of DEVS formalism to simulate automatically the models. Simulation is carried out in pressing a simple button which invokes an error checker before the building of the simulation tree. The simulation algorithm can be selected among hierarchical simulator (default with the DEVS formalism) or direct coupling simulator (most efficient when the model is composed with DEVS coupled models).

III. PROPOSED APPROACH

As pointed in sub-Section II-A, the traditional way to design ambient systems described in Figure 1 has the following drawback: the creation of Bean class components using the WComp platform is performed by the definition of methods (both implementing the behavior of a device and its execution machine) in the object oriented language C#. The compilation allows to obtain a set of library components which are used in a given Assembly (which corresponds to the designed ambient system). However, eventual conflicts due to the connections involved by the Assembly can be detected only after execution. This means that the Designer has to modify the execution machine of some components and restart the design at the beginning. We propose a quite different way to proceed, which is described in Figure 3. The idea is to use the DEVS formalism in order to help the Designer to:

- Validate different strategies for execution machines involved in an Assembly.
- Write the methods corresponding to the strategy of the execution machine he wants to implement.

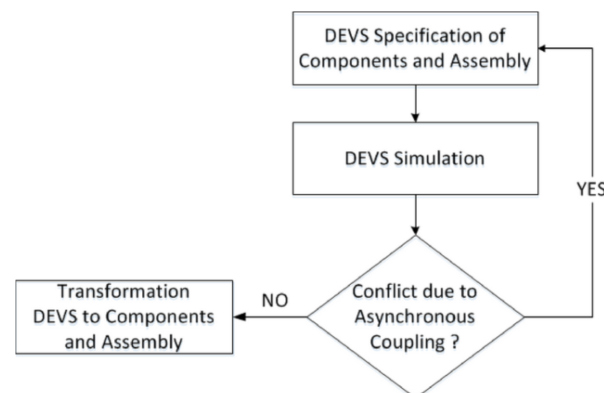

Figure 2. IoT component design using DEVS.

For that, the Designer has first to write the specifications the components as well as the coupling involved in an Assembly (corresponding to an ambient system to implement) then simulations can be performed. According to the results of the simulation, conflicts can be highlighted: if some conflicts exists the DEVs specifications have to be modified if not the design process goes on with C# implementation, as in Figure 1. The DEVS specifications can be used to help the Designer to write the methods of the Bean classes in the C# language Figure 2 and then compile

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-371-1

them and execute the resulting Assembly being assured that there will be no coupling conflict.

Section IV detailed the proposed approach using a pedagogical example. Two different execution machine strategies will be implemented using WComp and using the DEVS formalism. We will point out how DEVS can be used to simulate execution machines strategies before compilation and execution of the C# Bean classes. Furthermore, we also point out how the designer can use the DEVS specifications in order to write the methods involved in an execution machine strategy.

IV. CASE STUDY :SWITCHABLE ON/OFF LAMP

A. *Description*

We choose to validate the proposed approach on a pedagogical case study: realization of an application to control the lighting in a room. The case study involved three components to be assembled: a light component with an input (ON / OFF) and two switches components with an output (ON / OFF), as shown in Figure 3. Two different behaviors concerning the connections between the switch and the light component are envisioned (corresponding to the implementation of two different execution machines):

- First behavior: the light is controlled by toggles switches which rest in any of their positions.
- Second behavior: the light is controlled by pushbutton switches which have two-position devices actuated with a button that is pressed and released.

In this part, we will present first how we have implemented these two previous behaviors using the WComp platform. Then we will give the DEVS approach involving the DEVS specifications of the two behaviors of the case study and the way DEVS can be used for WComp design of the ambient components.

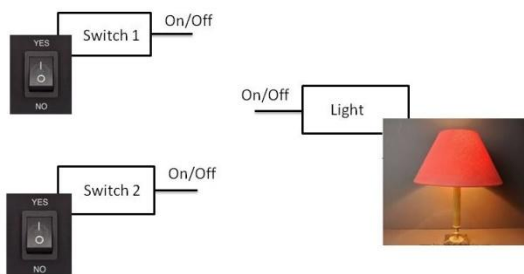

Figure 3. Assembly light and switches

B. *WComp implementation*

We implemented using the WComp platform the behavior corresponding to the toggle switch and the behavior of the push button switch. The two behaviors have been coded using two different Bean classes associated with the light component.

1) First behavior implementation.

The implementation corresponding to the toggle switch is described in the Figure 4. The line 2 is used to check the position of the toggle switch: if ON is true the line 3 ensures that there are subscribers before calling the event Property-Changed. In the lines 4 and 5, the event is raised and a resulting string is transmitted. The Bean class returns the String once the"ControlMethod" method is invoked.

```
1  public void ControlMethod(bool on) {
2  if(on)
3  {if (PropertyChanged != null)
4  PropertyChanged("Light_On");
5  }else{PropertyChanged("Light_Off");}
6  }
```

Figure 4. First light method implementation in WComp.

2) Second behavior implementation.

The implementation corresponding to the pushbutton switch is described in the Figure 5. The initialization of the "lightstate" variable of component Light is performed through line 1. Line 3 allows to switch the value of the "lightstate" variable while line 4 allows to initialize the message to be returned. Line 5 is dedicated to check the "lightstate" variable and to eventually change to returned message. Lines 6 and 7 allow to ensure that there are subscribers before calling the event Property-Changed and transmit the returned message.

```
1   public bool lightstate = false;
2  public void ControlMethod() {
3     lightstate =! lightstate;
4     string msg = "light_off";
5     if (lightstate) { msg ="light_on";}
6     if (PropertyChanged != null)
7       PropertyChanged(msg);
8     }
```

Figure 5. Second light method implementation in WComp.

After the compilation of the two Bean classes, each bean class will be instantiated and connected with two checkbox representing the respective switches in order to realize the assembly in a WComp container, as shown in Figure 6.

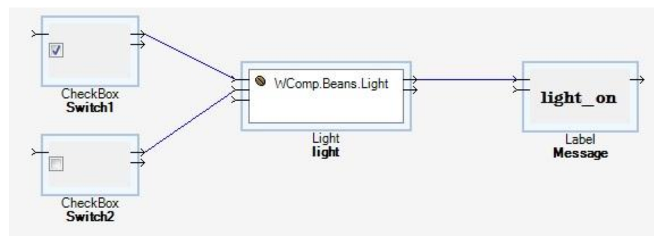

Figure 6. WComp assembly components.

C. *DEVS Specifications*

In order to highlight the interest of the DEVS formalism in the management of conflicts between WComp assembly components, we have written an atomic DEVS model for each kind of component "light" and implement both of them using the DEVSimPy platform.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-371-1

1) **First case:** DEVS implementation corresponding to the toggle switch behavior.

The DEVS specification of the corresponding WComp component of Section IV-B is achieved using the state automaton of Figure 7.

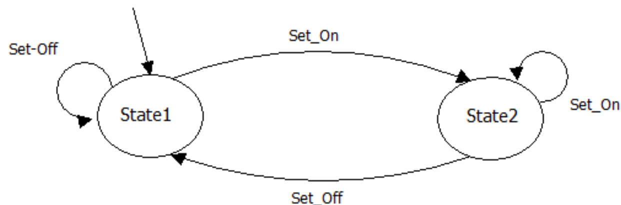

Figure 7. Automaton 1 of the light component.

The corresponding DEVSimPy implementation is given in Figure 8 (the behavior is expressed through the external transition of the Light component atomic model).

```
1    self.intstate= "OFF"
2    def extTransition(self):
3    for i in xrange(len(self.IPorts)):
4      msg=self.peek(self.IPorts[i])
5      if msg :
6        self.result[i]=msg.value[1]
7      if self.result[i]==self.intstate :
8          self.finstate=self.intstate
9          else:
10         self.finstate=self.result[i]
11           self.state['sigma']=0
```

Figure 8. External Transition of the light in DEVSimPy.

The initialization of the state variable “instate” is done in line 1 (initial value is OFF). Line 3 and line 4 allow to assign the variable msg with the value of the events on the input ports. From line 5 to line 10 the code allows to assign the value of the state variable “instate” according to the value of the variable “msg”: if the message on the port is equal to the initial state then the state variable remains on the same state else the value of the “instate” variable is changed. Line 11 by setting the variable sigma to 0 allows to activate the output function.

2) **Second case:** DEVS implementation corresponding to the pushbutton switch behavior.

The DEVS specification of the corresponding WComp component of Section IV-B is achieved using the state automaton of Figure 9.

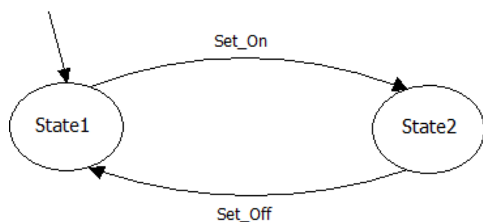

Figure 9. Automaton 2 of the light component.

The corresponding DEVSimPy implementation is given in Figure 10 (the behavior is expressed through the external transition of the Light component atomic model).

```
1  def extTransition(self):
2      for i in xrange(len(self.IPorts)):
3        msg=self.peek(self.IPorts[i])
4  if msg :
5    self.result[i]=msg.value[1]
6    if self.intstate == "ON":
7    self.finstate= "OFF"
8    else:
9    self.finstate="ON"
10   self.intstate=self.finstate
```

Figure 10. External Transition of the light in DEVSimPy.

Figure 10 gives the code of the external transition of the Light component atomic model. One can see from line 5 to 10 that in this second case the output message is switched from ON to OFF or OFF to ON according to the values of the input ports.

D. *Simulation results*

In both two cases, once the modeling scheme has be realized using the DEVSimPy environment we are able to perform simulations which correspond to the behavior of the ambient system under study according to the two different execution machines that have been defined.

The simulation results of the first case express the fact that the execution machine allows the ambient system under study remains in the initial position (ON or OFF) until we will actuate another position using one of the switches.

The simulation results of the second case express the fact that the execution machine allows the ambient system under study to alternately “ON” and “OFF “with every push of one of the switches.

E. *Interest of the presented approach*

As described in Sections IV-C and IV-D, the proposed approach allows to study the behavior of an ambient system using DEVS simulations before any WComp implementation. This will allow a Designer of ambient system to select the desired execution machine without having coding and compiling the C# classes under WComp platform.

Furthermore, in this sub-section, we briefly introduce how the DEVS specifications can be use by an ambient system Designer to write the code of execution machine. From the two previous cases one can note that the WComp method of Bean class of a given ambient component and the external transition of the corresponding DEVS atomic model present some similarities (in the one part, see Figure 4 and Figure 8; on the other part, see Figure 5 and Figure 10).

V. CONCLUSION AND FUTURE WORK

This paper dealt with an approach for the design and the implementation of IoT ambient systems based on Discrete

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-371-1

Event Modeling and Simulation. Instead of waiting the implementation phase to detect eventual conflicts, we propose an initial phase consisting in DEVS modeling and simulation of the behavior of components involved in an ambient system, as well as the behavior of execution machines. Once the DEVS simulations have brought successful results, the Designer can implement the behavior of the given ambient system using an IoT framework such as WComp. The presented approach has been applied on a pedagogical example which is described in detail in the paper: implementation of two different behaviors of a given ambient system, definition of the corresponding DEVS specification, implementation of the DEVS behavior using the DEVSimPy framework analysis of the simulation results. Furthermore, we have also pointed out that the DEVS specifications can be used in order to help the Designer to write the behavior of the IoT components. Our future work will consist in proposing an approach allowing to automatically write the behavior of the execution machines after their validation based on DEVS simulation.

In this paper, we used a pedagogical example to model "execution strategies" to prevent conflict.

REFERENCES

[1] L. Capocchi, J. F. Santucci, B. Poggi, and C. Nicolai, "DEVSimPy: A collaborative python software for modeling and simulation of DEVS systems," Proc. IEEE Conf. WETICE, Eds, IEEE Computer Society, pp. 170–175, 27-29 June 2011, doi: 10.1109/WETICE.2011.31, Available from: http://code.google.com/p/devsimpy. [Retrieved: August, 2014].

[2] D. Cheung-Foo-Wo, "Adaptation dynamique par tissage d'aspects d'assemblage," PhD thesis, University of Nice Sophia Antipolis, Nice (France), 2009.

[3] E. Jones, T. Oliphant, and P. Peterson, "Scipy: Open source scientific tools for python," 2001, Available from: http://www.scipy.org. [Retrieved: February, 2014].

[4] X. Li, H. Vangheluwe, Y. Lei, H. Song, and W. Wang, "A testing framework for DEVS formalism implementations," Proc. Theory of Modeling & Simulation: DEVS Integrative M&S Symposium, Society for Computer Simulation International, San Diego, CA, USA, 2011, pp. 183–188.

[5] T. E. Oliphant. "Python for scientific computing," Computing in Science and Engineering 9, 2007, pp. 10–20.

[6] F. Perez, B.E. Granger, and J. D. Hunter, "Python: An ecosystem for scientific computing," Computing in Science and Engineering 13, 2011, pp. 13–21.

[7] N. Rappin, and R. Dunn, "WxPython in action," Manning, 2006.

[8] B. P. Zeigler, "An introduction to set theory," Tech. rep., ACIMS Laboratory, University of Arizona, 2003. Available from: http://www.acims.arizona.edu/EDUCATION. [Retrieved: April, 2014].

[9] B. P. Zeigler., H. Praehofer, and T. G. Kim, "Theory of Modeling and Simulation," Second Edition. Academic Press, 2000.

Copyright (c) IARIA, 2014. ISBN: 978-1-61208-371-1