Extending Osate 2 for Custom Simulation of Virtual Devices

Faber D. Giraldo[1], Monica M. Villegas[1], Juan E. Medina[2]
1 SINFOCI Research Group University of Quindío, Colombia
Email:fdgiraldo@uniquindio.edu.co, mmvillegasa@uqvirtual.edu.co
[2] Smartenit Inc., San Juan Capistrano, CA, USA
juan@smartenit.com

***Abstract*— When developers test cyber-physical systems, there may be a lack of enough physical devices. In such cases, a tool that allows virtual simulation of the devices is useful. In this work, we propose an extension named *Custom Simulation of Virtual Devices (CSVD)* developed for OSATE[1] 2, an Eclipse-based tool that supports *Architecture and Analysis Design Language (AADL[2])* models. The CSVD extension allows the simulation of virtual serial devices specified in an AADL model inside the QEMU[3] emulator, which is configured using the parameters specified in the AADL model of the designed system. An example of a system in this context is a local network of devices connected by a serial bus to a gateway with the purpose of simulating a data flow between the gateway and a device with serial connection.**

***Keywords*— OSATE 2, AADL, QEMU, Simulation, CPS, Model**

I. Introduction

Nowadays, the design of digital systems that involve the communication between external devices and embedded systems has increased, for instance, in the area of automation, sensors networking or IoT design, where the interaction between devices and embedded platforms is common. When the software for such embedded systems is developed, simulating the devices for validating the software becomes necessary.

The focus of this work is to present an extension of the OSATE 2 tool, an environment that supports AADL models and is based on the Eclipse[4] platform. The extension presented in this work, named *Custom Simulation of Virtual Devices (CSVD)* is an Eclipse plugin that allows the analysis of AADL models for the generation of files that are needed for the execution of the simulation of devices specified in the AADL model. The simulation is executed using the QEMU [5] emulator, which is configured by the parameters defined in the AADL model of the designed system.

The CSVD extension is useful for the OSATE 2 tool because it provides an environment to simulate custom virtual devices, such as sensors connected to a gateway through a device, responsible for collecting information and sending data over a serial bus. This specific scenario emulates a system where a Zigbee[5] dongle is used for controlling a network of sensors reporting temperature and humidity measurements periodically.

The simulation runs in the QEMU emulator, which is launched using the configuration, such as a modeled gateway. The configuration specifies the desired platform (supported by QEMU), the operating system image, the kernel image, RAM size, and storage values.

The usage of the CSVD extension is simple. The user needs to select the AADL model of the system, execute the generation of required files for the simulation/emulation, and if the process finishes successfully, proceed with the execution of the simulation inside the emulator. Debug messages are also provided in the GUI of the CSVD extension in case the AADL model is not valid or if there were problems in the generation.

Section 2 of this paper provides an introduction to the AADL language as an AADL review. Section 3 presents an introduction to the OSATE 2 tool, as an OSATE 2 review. Section 4 includes a description of related tools. Section 5 contains a description of the CSVD extension of OSATE 2. Section 6 describes a case study to show a usage example of the CSVD extension. Section 7 concludes the paper and finally, Section 8 provides some future work proposed for this project.

II. AADL Review

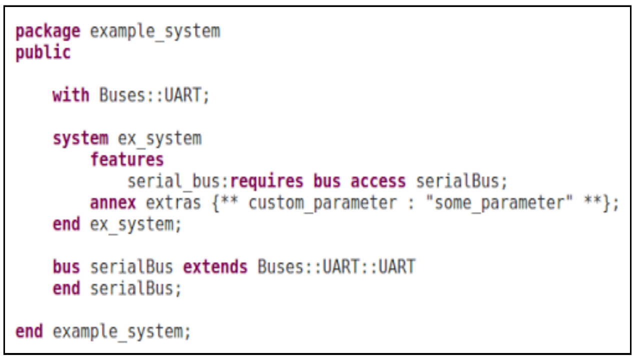
Architecture Analysis & Design Language (AADL) is a modeling language used for modeling embedded systems and representing their architecture as a hierarchy of interacting components that allow the analysis of the modeled system.

AADL is a unifying framework for model-based software systems engineering which can be used to *capture the static modular software architecture*, *the runtime architecture*

[1] http://www.osate.org
[2] http://www.aadl.info
[3] https://www.qemu.org
[4] https://www.eclipse.org
[5] http://www.zigbee.org

regarding communicating tasks, the computer platform architecture on which the software is deployed, and *any physical system or environment that the system interacts with* [1].

AADL models are composed of *components*, *properties*, and *notations*. For components, AADL defines the categories of *data type*, *thread*, *thread group*, *subprogram*, *process*, *memory*, *bus*, *processor*, *device*, *virtual processor*, *virtual bus*, *system* and *abstract*. For properties, *predefined property sets*, such as *deployment properties*, *thread properties*, *timing properties*, *communication properties*, *memory properties*, *programming properties*, *modeling properties*, and *AADL project properties* [2]. As an example, a system can be modeled from its abstraction, where a system is composed of subsystems, or a system can be composed of only devices. As shown in Figure 1, an AADL model of a system can have features and annexes. Features, in this case, is the specification of a serial bus in the system, and annexes can be used for extra details that are not considered in the AADL meta-model. In Figure 2 is shown the diagram of the AADL specified in Figure 1, generated by the OSATE tool.



```
package example_system
public

    with Buses::UART;

    system ex_system
        features
            serial_bus:requires bus access serialBus;
        annex extras {** custom_parameter : "some_parameter" **};
    end ex_system;

    bus serialBus extends Buses::UART::UART
    end serialBus;

end example_system;
```

Fig. 1. AADL model example.

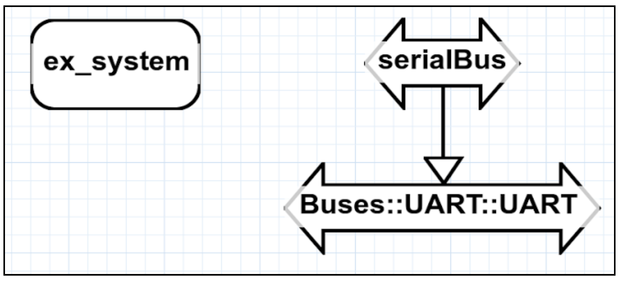
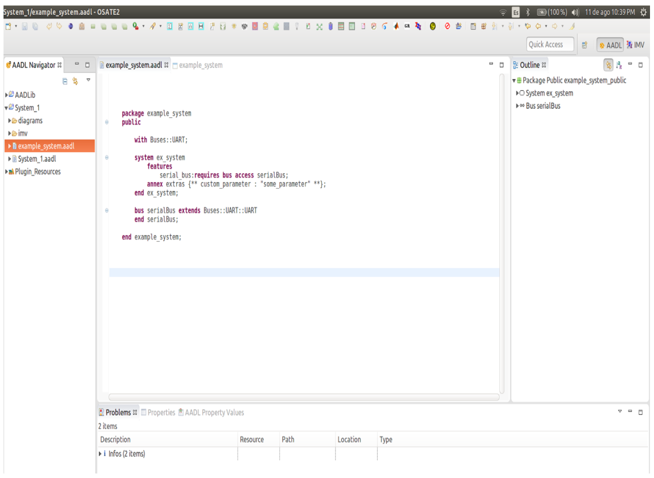


Fig. 2. Diagram of the AADL model in Figure 1.

III. Osate 2 Review

OSATE 2 is an open source platform that fully supports the AADL meta-model and which is based on the Eclipse platform. OSATE 2 provides a complete textual editor for AADL and a set of simple analysis tools [3]. This tool can be extended to allow, for instance, the addition of a simulation/emulation launching method. Figure 3 shows a snapshot of the OSATE 2 environment, containing the example shown in Figures 1 and 2.



```
package example_system
public

    with Buses::UART;

    system ex_system
        features
            serial_bus:requires bus access serialBus;
        annex extras {** custom_parameter : "some_parameter" **};
    end ex_system;

    bus serialBus extends Buses::UART::UART
    end serialBus;

end example_system;
```

2 items

Description	Resource	Path	Location	Type
Infos (2 items)				

Fig. 3. Snapshot of OSATE 2.

IV. Related Tools

After a review of extensions of OSATE 2 that provide analysis of AADL models we found some related works that will be mentioned as follows: Delange et al. [4] described an approach for the modeling, verification, and implementation of ARINC653 systems using AADL. It details a modeling approach exploiting the new features of AADL version 2 for the design of ARINC653 architectures. Delange's work also proposed modeling patterns to represent other safety mechanisms such as the use of Ravenscar for critical applications.

Hamdane et al. [6] proposed an approach for the verification of the AADL architecture, which is assisted by a toolchain. Hamdane's work [6] defined a source meta-model for AADL and a target meta-model for the timed automata formalism. The transformation process works in two steps, a Model2Model transformation, which takes an AADL Model and produces the appropriate timed automata model, and a transformation of a Model2 Text which takes a timed automata model and generates code, which is accepted by the Uppaal toolbox.

The difference between works [4], [6], and the one presented in this paper, is that our proposal is focused on the simulation of serial devices inside an emulated environment with controlled parameters from the architectural model.

Our work allows the interaction with the devices modeled inside the emulator, and is useful for the validation of embedded software, such as software for ARM-based platforms.

V. CSVD Extension for OSATE 2 Tool

The particular process of the CSVD extension consists of (1) loading the AADL model in the GUI of the plugin, (2) executing the generation of the required files and (3) launching the simulation through the QEMU emulator. Figure 4 shows a diagram that represents the structure of the CSVD extension.

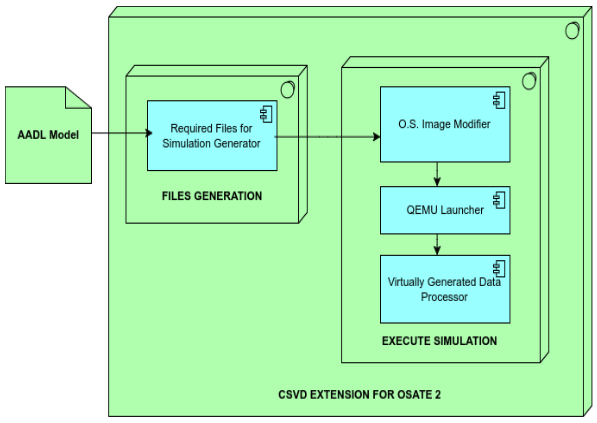

AADL Model
Required Files for Simulation Generator
FILES GENERATION
O.S. Image Modifier
QEMU Launcher
Virtually Generated Data Processor
EXECUTE SIMULATION
CSVD EXTENSION FOR OSATE 2

Fig. 4. Structure of the CSVD Extension.

For using the CSVD extension, it is necessary to click on the CSVD launching icon that is located in the Toolbar of OSATE 2 as shown in Figure 5. After clicking the CSVD launching icon, a graphical user interface (GUI) is displayed as illustrated in Figure 6, which allows the loading of the AADL model, the generation of the required files and the executing of the simulation. The button for the simulation execution is enabled if the generation process is successfully finished. Also, below the 'Execute Generation' button is a label that shows the messages related to the status of the generation of required files, allowing the user to know if the process had an error or if it finished successfully.

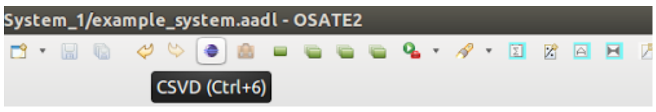
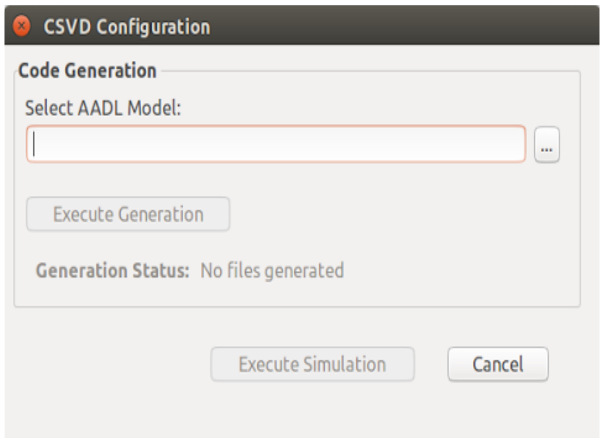


Fig. 5. Icon for launching the CSVD extension in OSATE 2.



Generation Status: No files generated

Fig. 6. CSVD Graphical User Interface.

The generation of the required files is based on a python process that loads specific rules written in JSON format, rules that are specified by the user/developer and which are related to the way specific commands and configurations will be generated from the AADL model of the designed system. After the Python process generates the required files, the Operating System (OS) image that QEMU starts with, must be reconfigured through an OS image modifier. The modification to the OS image is done by a process that unpacks it, adds the generated and required files including the process that simulates the devices and a script to launch the simulation inside the emulator. After the addition of the files is completed, the OS image is re-packed and ready to be started.

The launch of the QEMU emulator is executed through the button in the Plugin GUI. After the window of the QEMU is displayed and the OS starts, the data stream received on the serial port from the simulated devices inside the QEMU emulator can be read and made available to be processed by specific software.

VI. Example of Our Approach

In this section, a system composed of two sensors, one serial device receiving data from the two sensors and one serial bus that connects the serial device with the gateway is modeled as an example (shown in Figure 7). Each sensor has a MAC address which can be used to identify them.

Figure 8 shows the declaration of the systems and devices of the model, where the extra details are specified, for instance, QEMU kernel image path, OS image path and CPU type for the emulation. For the sensors, the MAC address, and the mean and deviation parameters for the random values generated from the sensors are specified.

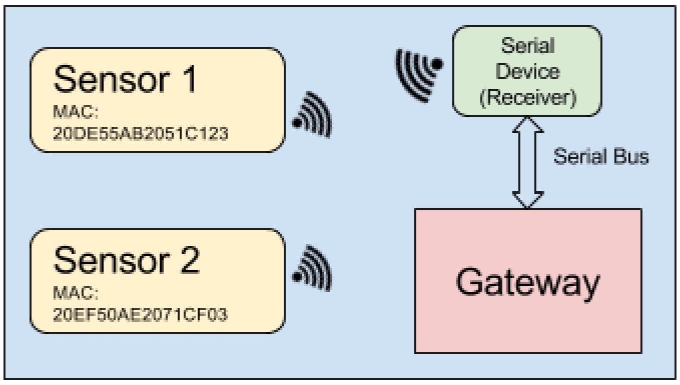
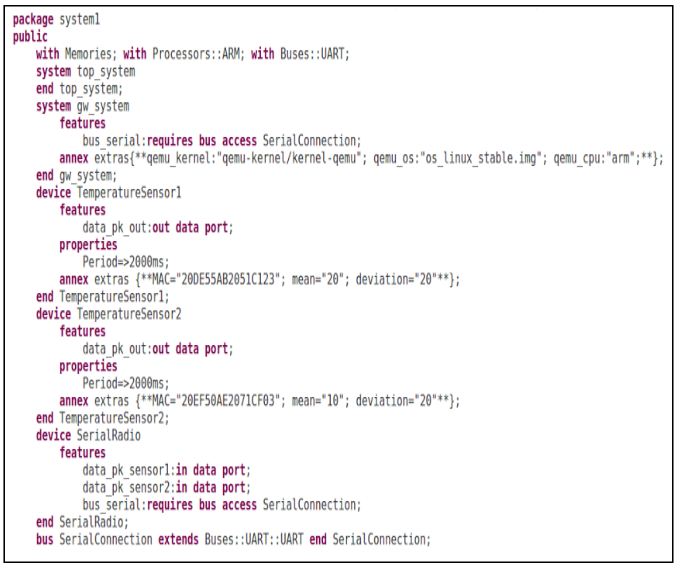
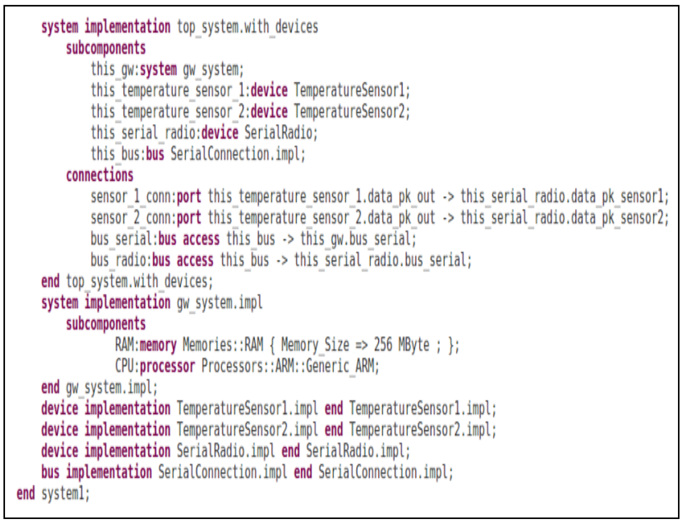


Fig. 7. System Modeled.



```
package system1
public
    with Memories; with Processors::ARM; with Buses::UART;
    system top_system
    end top_system;
    system gw_system
        features
            bus_serial:requires bus access SerialConnection;
        annex extras{**qemu_kernel:"qemu-kernel/kernel-qemu"; qemu_os:"os_linux_stable.img"; qemu_cpu:"arm";**};
    end gw_system;
    device TemperatureSensor1
        features
            data_pk_out:out data port;
        properties
            Period=>2000ms;
        annex extras {**MAC="20DE55AB2051C123"; mean="20"; deviation="20"**};
    end TemperatureSensor1;
    device TemperatureSensor2
        features
            data_pk_out:out data port;
        properties
            Period=>2000ms;
        annex extras {**MAC="20EF50AE2071CF03"; mean="10"; deviation="20"**};
    end TemperatureSensor2;
    device SerialRadio
        features
            data_pk_sensor1:in data port;
            data_pk_sensor2:in data port;
            bus_serial:requires bus access SerialConnection;
    end SerialRadio;
    bus SerialConnection extends Buses::UART::UART end SerialConnection;
```

Fig. 8. Declaration of systems and devices in AADL of the designed system.



```
    system implementation top_system.with_devices
        subcomponents
            this_gw:system gw_system;
            this_temperature_sensor_1:device TemperatureSensor1;
            this_temperature_sensor_2:device TemperatureSensor2;
            this_serial_radio:device SerialRadio;
            this_bus:bus SerialConnection.impl;
        connections
            sensor_1_conn:port this_temperature_sensor_1.data_pk_out -> this_serial_radio.data_pk_sensor1;
            sensor_2_conn:port this_temperature_sensor_2.data_pk_out -> this_serial_radio.data_pk_sensor2;
            bus_serial:bus access this_bus -> this_gw.bus_serial;
            bus_radio:bus access this_bus -> this_serial_radio.bus_serial;
    end top_system.with_devices;
    system implementation gw_system.impl
        subcomponents
                RAM:memory Memories::RAM { Memory_Size => 256 MByte ; };
                CPU:processor Processors::ARM::Generic_ARM;
    end gw_system.impl;
    device implementation TemperatureSensor1.impl end TemperatureSensor1.impl;
    device implementation TemperatureSensor2.impl end TemperatureSensor2.impl;
    device implementation SerialRadio.impl end SerialRadio.impl;
    bus implementation SerialConnection.impl end SerialConnection.impl;
end system1;
```

Fig. 9. Implementation of the systems and devices in AADL of the designed system.

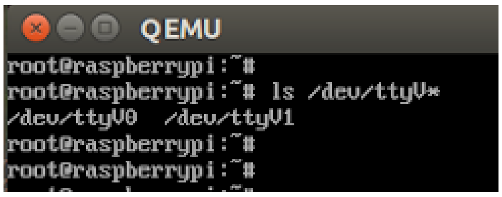
The expected output of this simulation is to view data being generated on the virtual serial port in the QEMU emulator with the MAC address of each sensor modeled. This data could be used by a software process, by reading and processing it according to specific requirements, such as temperature values from a sensor, where a data package could contain start and end bytes, the MAC address and the payload including the temperature value of the sensor.

Figure 9 describes the implementation of the systems and devices. The connections between the devices and systems are specified. The subcomponents area specifies the RAM size and CPU type for the gateway.

Figure 10 shows the interface of the virtual port created from the AADL model inside QEMU. Two virtual ports are created, one for reading and one for writing to the simulated serial device. Currently, the version of this extension supports reading data from the devices but not writing.

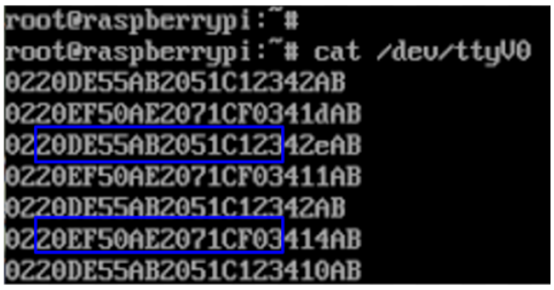
Figure 11 shows the data being received from the serial device that collects the data from the sensors. The simple verification is to check that the data is from the modeled sensors by analyzing the values in the data package and comparing the MAC address with the one specified in the model, this, because the MAC address inside the data package (marked with blue in Figure 11) must match the MAC address of each sensor specified in the model (Figure 8). With this, a software that needs data from several simulated devices can read the port and process the information received.

For this example, only two sensors were modeled, but both the language and the simulation/emulation system supports several more devices.



```
root@raspberrypi:~#
root@raspberrypi:~# ls /dev/ttyV*
/dev/ttyV0  /dev/ttyV1
root@raspberrypi:~#
root@raspberrypi:~#
```

Fig. 10. Interface of the virtual port created from the AADL model inside QEMU.



```
root@raspberrypi:~#
root@raspberrypi:~# cat /dev/ttyV0
0220DE55AB2051C12342AB
0220EF50AE2071CF0341dAB
0220DE55AB2051C12342eAB
0220EF50AE2071CF03411AB
0220DE55AB2051C12342AB
0220EF50AE2071CF03414AB
0220DE55AB2051C123410AB
```

Fig. 11. Serial data received.

VII. Conclusions

The CSVD plugin developed for OSATE 2 is extensible. It allows the addition of new types of models and systems. In its current state, the support is limited regarding specific simulation systems that can be customized. The work described in this paper could be useful for future modifications or specific code generation with simulation/emulation purposes. Modifications to the plugin can be done easily by adding new support, allowing developers to create new simulations of other kinds of devices available in the AADL language and that can be somehow linked to other kinds of simulators and emulators.

The primary idea of the CSVD plugin is to provide serial data packages for them to be used in software processes running in the QEMU emulator, such as processes that need serial data for testing purposes.

VIII. Future Work

We propose the addition of other QEMU like emulators that can be used for simulating the system on other embedded systems architectures. We also propose to continue extending the functionalities of the "files generation" and "execute simulation" components to add support for new devices and protocols.

References

[1] Peter H. Feiler, David P. Gluch, "Model-Based Engineering with AADL", Addison-Wesley 2012

[2] SEI, "AADLv2 Cheat Sheet: Basics", Available: https://wiki.sei.cmu.edu/aadl/images/a/ac/Aadlsheet_letter.pdf, Accessed: July 2017

[3] SEI, "Osate 2", Available: https://wiki.sei.cmu.edu/aadl/index.php/Osate_2, Accessed: July 2017

[4] Julien Delange, Laurent Pautet, Alain Plantec, Mickael Kerboeuf, Frank Singhoff, and Fabrice Kordon. 2009. Validate, simulate, and implement ARINC653 systems using the AADL. In *Proceedings of the ACM SIGAda annual international conference on Ada and related technologies* (SIGAda '09). ACM, New York, NY, USA, 31-44. DOI=http://dx.doi.org/10.1145/1647420.1647435

[5] QEMU, "QEMU", Available: http://www.qemu.org/, Accessed: July 2017

[6] M. Hamdane, A. Chaoui and M. Strecker, "Toolchain Based on MDE for the Transformation of AADL Models to Timed Automata Models," *Journal of Software Engineering and Applications*, Vol. 6 No. 3, 2013, pp. 147-155. doi: 10.4236/jsea.2013.63019.