2Simulate: A Distributed Real-Time Simulation Framework

Jürgen Gotschlich, Torsten Gerlach, Umut Durak
German Aerospace Center (DLR)
Institute of Flight Systems
{ juergen.gotschlich, torsten.gerlach, umut.durak }@dlr.de

Simulating large scale complex real-time systems requires enabling infrastructure for real-time co-simulation of various subsystems with complex behaviors and interfaces. AVES (Air Vehicle Simulator) is a reconfigurable flight simulator of the Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR) for research into rotorcraft and fixed-wing aircraft behavior. Complex models of aircraft subsystems in AVES required a distributed real-time simulation framework. This paper presents 2Simulate, the enabling simulation infrastructure of the AVES facility that facilitates integrating a wide range of models and simulation hardware and software components. 2 Simulate is a unique simulation infrastructure being domain independent and methodology neutral. Its three components, 2SimCC, 2SimRT and 2SimMC, provide various capabilities including simulation control, task scheduling, model integration and hardware/software interfacing.

1 Introduction

Deutsches Zentrum für Luft- und Raumfahrt e. V. (DLR) Institute of Flight Systems has a reconfigurable flight simulator for research into rotorcraft and fixed-wing aircraft behavior. It is called Air Vehicle Simulator (AVES). AVES features a common motion platform and interchangeable roll-on/roll-off (RoRo) cockpits, enabling rapid turnaround of research activities [1].

Figure 1 DLR AVES

While developing system simulations, composing various models of subsystems and integrating them with the diverse tools and components that are required for the operation of simulation has always been a major technical challenge. The complex nature of the modelled subsystems and evolving requirements of the user community made this challenge heavier for flight simulators.

AVES has been developed based upon the idea to utilize reusable, flexible, standardized and properly validated software modules. It has a distributed architecture that enables each module to run either on a single computer connected via Ethernet or run together on the same hardware as distinct processes. Critical processes, e.g. the flight loop, are run in hard real-time conditions on real-time operating systems.

In this paper, 2Simulate, the enabling simulation infrastructure of the AVES facility is presented. 2Simulate is an overall simulation framework to facilitate integrating a wide range of models and simulation components like data recorders or image generators. The next section will provide a background about simulation frameworks. 2Simulate will then be introduced with a quadrotor simulation example that demonstrates its capabilities.

2 Simulation Frameworks

Simulating large-scale complex real-time systems requires specific attention on the infrastructure that enables the real-time co-simulation of various subsystems with complex behaviors and interfaces. The simulation community has long been working on tools and infrastructures that make reliable, maintainable and extensible complex systems simulations possible.

Huang and Sarjoughian [2] state that a separate effort to develop a methodology for simulation of complex real-time systems is required. They advocate utilizing the system-theoretic modelling approach Discrete Event System Specification (DEVS) [3] for simula-

tion modelling just as the Unified Modeling Language (UML) is used for software design. With RTDEVS/CORBA, Cho et al. introduced a real-time distributed simulation infrastructure [4]. This infrastructure provides services for time synchronization, message delivery, interfacing external systems and implementing real-time computations.

About ten years before the RTDEVS/CORBA effort, Lee and his colleagues had already proposed one of the first simulation frameworks, Ptolemy, for simulating heterogeneous systems [5]. Ptolemy aimed at making use of object-oriented software technology to model subsystems. It was a framework which provides a set of object-oriented class definitions with standard interfaces. Thus, with generic objects more specialized interoperable domain-specific objects can be implemented.

In 1998, NASA published a domain-specific framework for simulation of aircraft [6]. They introduced LaSRS++, an object-oriented framework for developing real-time flight simulators. In this framework, a set of abstract base classes is provided to interface the modelled aircraft with the framework services. These base classes include e.g. *FlightSim* that defines the initialization and execution of the vehicle model, *World* that provides a world to fly around, *HardwareControl* that abstracts the hardware used in the simulation and *Supervisor* that cares about the real-time clock. Via LaSRS++ framework services one can achieve real-time framing, simulation models management like trim, hold, reset, and interfacing with the I/O hardware.

These three important approaches (DEVS, Ptolemy, LaSRS++) each provide an infrastructure for simulation of complex real-time systems. While the first one utilizes a domain-independent simulation formalization approach and expects its user to develop DEVS models, the second one, Ptolemy, provides an object-oriented approach for systems modelling. The third one, LaSRS++, provides a domain-specific solution. However, there are two important issues about these approaches. First, simulation developers require flexible frameworks so that they can utilize various methodologies for systems modeling. There is no single methodology that satisfies all user requirements for modeling large and complex systems. While power system modelers find bond graphs more useful, flight systems modelers may like state flow diagrams better. Second, Simulation developers require frameworks to be flexible also in creating specific architectures for their particular problems. Domain-specific frameworks always possess the developer's abstraction of the domain which may not fit all of their users' needs.

The simulation framework that is presented in this paper, 2Simulate, neither enforces a modelling methodology nor enforces a domain architecture. It provides various real-time simulation services via Application Programming Interfaces (APIs), which do not depend on the domain architecture or the modelling methodology.

3 2Simulate

2Simulate is a C++ real-time distributed simulation framework. It is composed of three components, namely 2Simulate Real-Time Framework (2SimRT), 2Simulate Control Center (2SimCC) and 2Simulate Model Control (2SimMC). Figure 2 presents a simple UML Component Diagram of 2Simulate.

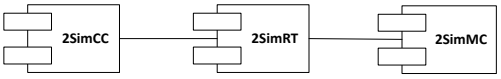

Figure 2 Components of 2Simulate

2SimRT is the core simulation framework of 2Simulate that provides deterministic scheduling and controlling of real-time tasks. It comes as Windows or QNX images (Libraries) and API header files. Any simulation application that is based on 2SimRT is called a Target. Each Target runs various real-time tasks that are implemented utilizing the 2SimRT API. These real-time tasks include model control tasks as well as a wide range of data connections to external devices or components. 2SimRT also provides a Common Database to manage the data that flow through the internal and external interfaces.

2SimMC is the component that abstracts model interfaces for 2SimRT. It works with MATLAB/Simulink [7], Advantage Framework [8] or native C++ models. Targets may have more than one model that they co-simulate over 2SimMC. It supports the real-time operating system QNX and Windows. For native C++ model development, users can employ 2SimMC via developing their models using it API. For MATLAB/Simulink and Advantage Framework, 2SimMC is integrated automatically into the models during the code generation process.

2SimCC is the component to configure the Control Center for specific needs. It is a Windows executable

which can be customized via configuration files called 2SimCC project files. Control Center can run, pause or stop various Targets. Besides, it accesses the Target Data Dictionaries which can be defined as the data access mechanisms and enables presenting or editing Target data at runtime. It can also enable user management to define and enforce user access rights.

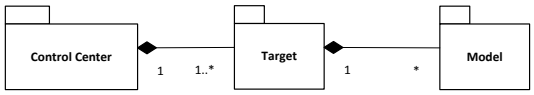

Figure 3 2Simulate Simulation Architecture

As presented in Figure 3, the simulation architecture of 2Simulate has a Control Center that can control a number of Targets which may co-simulate various Models and interacts with various external systems.

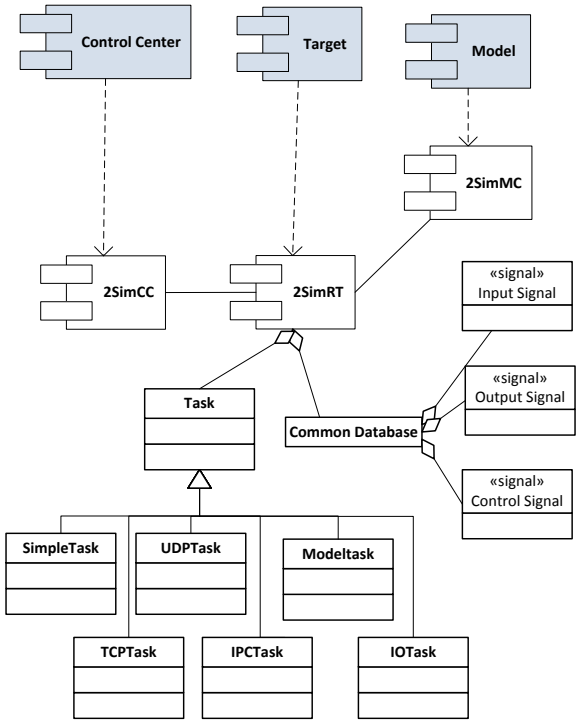

Figure 4 2Simulate Component Architecture

The component architecture of 2Simulate is presented in Figure 4. 2SimRT provides a number of schedulable task templates and a common database. Some major tasks are depicted in the figure. They can be programmed using their pre- and post-initialization and pre- and post-process callbacks with extra functionality depending on their types. SimpleTask is the simplest task type which has no extra functionality. The user can modify it for his/her needs. With a UDPTask, one can schedule a UDP communication and with TCPTask, a TCP communication. IPCTask is used for inter-process communication with other applications on the same machine. With IOTask, a 2SimRT user can connect to I/O interfaces like switches or onboard computers and lastly the ModelTask enables to run the models that are built to be integrated into 2Simulate. There are more task types whose properties and functions are mostly inherited from these major tasks (see Fig. 5). As an example, 2SimRT has an ARINCTask and a CANTask derived from IOTask for these widely used communication protocols. As another example ConTask, that is used to connect the developed 2SimRT application to 2SimCC, is derived from TCPTask. One can integrate Simulink models and C++ models using SimulinkTask and CppModelTask that are derived from ModelTask. Last to mention, users can also extend these tasks to create their own special tasks. A nice example for that is WclsTask that is derived from UDPTask to enable communication with the control loading systems from Wittenstein GmbH. It implements a particular protocol for this subsystem in AVES.

There are also simulation utilities like adding displays or command line monitors and data injectors in 2SimRT.

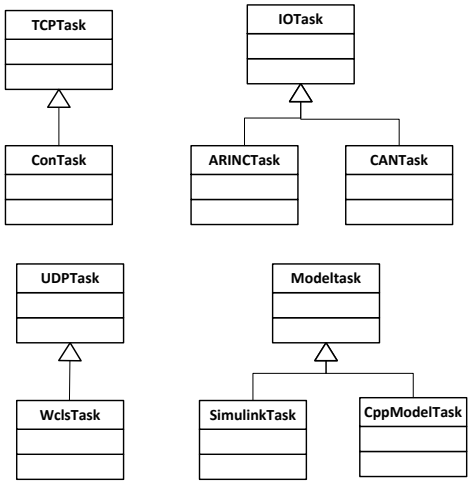

Figure 5 Examples of Task Hierarchies

Common Database is a 2SimRT add-on. It allows its users to define interfaces of the Target by Input and Output Signal specifications. And it provides an internal data interface to Control Signals. These signals are defined in text files, which are then used for automatic code generation that produces a Common Database source code with an API to access and mod-

ify these data items. Users can also design and develop their indigenous data management routines.

4 Simulating a Quadrotor Using 2Simulate

Further details of 2Simulate will be presented by an example implementation that simulates a quadrotor. This Quadrotor Simulator consists of an Operator Node that has a joystick for getting operator inputs, a virtual instrument panel that provides the user with a primary flight display, a Visualisation Node that has an out-of-the window image generator for simulating the camera on the quadrotor. It has a Simulation Node that runs the flight dynamics and control model of the quadrotor and an Instructor Node that controls and monitors the simulator execution. The architecture of this Quadrotor Simulator is depicted below.

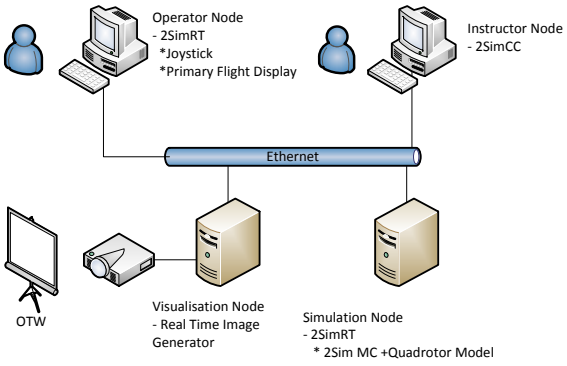

Figure 6 Quadrotor Simulator Architecture

The Simulation Node uses an open source Simulink implementation of a quadrotor model [9] from the Mathworks File Exchange site. The Simulink model implements flight dynamics and control algorithms from Bouabdallah's work [10].

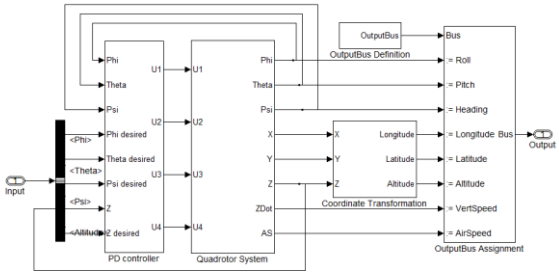

Figure 7 Simulink Model of Quadrotor Flight Dynamics and Control

The Simulink model can be used with 2Simulate after it has been converted into C++ code using Mathworks Simulink Coder [11]. A part of the Simulink Coder is the Target Language Compiler. It is a specification of the code generation [12] utilizing so called system target files, which can be customized for specific needs. 2Simulate has such a set of customized system target files. They embed 2SimMC into the model code during the code generation. Thus, an auto-generated model is readily available for SimulinkTask.

While developing the Simulation Node, a SimulinkModelTask is generated from ModelTask to interface the quadrotor model with 2Simulate. Below is a code extract that adds the SimulinkModelTask to the Simulation Node. The scheduling schema of the task is specified as Round Robin (`TASK_SCHED_RR`), the task priority is set to 30 (0 is the highest and 50 is the lowest) and the frame time is set to 10 milliseconds.

```
quadSimTask *pQuadST = new quadSimTask ( pTSim, "QUAD",
                          TASK_SCHED_RR,30, 10*iMSECtoNSEC);
quadSimTask ->setDesc( "Quadrotor Simulink Task" );
quadSimTask ->setPreProcCB ((void(*)(TSim *,TSimRtTask*))
                          &pQuadrotorTSimSimulinkTask_CB );
```

Code 1. SimulinkModelTask

Using a UDPTask Simulation Node gets the user inputs from Operator Node and sets them in the Common Database. As presented in the third line of Code 1, there is a pre-process callback function pointer specified for the SimulinkModelTask. In this callback function the inputs of the Simulink model are set using the values in the Common Database.

Here is a code extract from Simulation Node, the *com* is an instance of Common Database.

```
void pQuadSimTask_CB( TSim *pAppl, TSimRtTask *pRtTask ) {
   com->o.r.quad.Input.Phi     = com->i.r.acctrl.ksiCmd;
   com->o.r.quad.Input.Theta   = com->i.r.acctrl.etaCmd;
   com->o.r.quad.Input.Psi     = com->i.r.acctrl.zetaCmd;
   com->o.r.quad.Input.Altitude = com->i.r.acctrl.plaLCmd;
}
```

Code 2. SimulinkModelTaskCallback Function

It has input (i) and output (o) signals. As an example, *Phi* is sent to the quadrotor model as an input and it comes from the aircraft command *acctrl ksiCmd.*

The Common Database code is auto-generated using the signal specifications. Signal specifications are well formed text files wherein the user defines the identifier, the type and the length of signals. An extract from the signal specifications of Simulation Node is given below.

```
#//***************************************************************************
#//
#//    aircraft control from joystick
#//    ==============================
#//
#//***************************************************************************
q
| 21000 | 00000 | acctrl | | simModeCmd|    1 | D | D | D | RAW  | [-]        | simModeCmd
| 21001 | 00000 | acctrl | | etaCmd    |    1 | F | F | F | RAW  | [-]        | etaCmd
| 21002 | 00000 | acctrl | | ksiCmd    |    1 | F | F | F | RAW  | [-]        | ksiCmd
| 21003 | 00000 | acctrl | | zetaCmd   |    1 | F | F | F | RAW  | [-]        | zetaCmd
| 21004 | 00000 | acctrl | | plaLCmd   |    1 | F | F | F | RAW  | [-]        | plaLCmd
```

Code 3. Simulation Node Signal Specification

2Simulate provides a Display utility to add a 2Indicate [13] display to its 2SimRT framework presenting the signals that are specified in Common Database. VisualisationTask is a kind of UDPTask to picture the state of the simulated entity in a virtual environment. It requires the spatial state of the entity, i.e. latitude, longitude and altitude and sends this state to an OpenSceneGraph-based RealTimeImageGenerator [14] over UDP.

In Operator Node SimpleTask is used to collect joystick inputs from the user and Display is used to present a Primary Flight Display. Simulation Node employs a VisualizationTask to drive the Visualization Node.

The last component to mention is the Instructor Node which utilizes 2SimCC to control the whole simulation (Fig. 8).

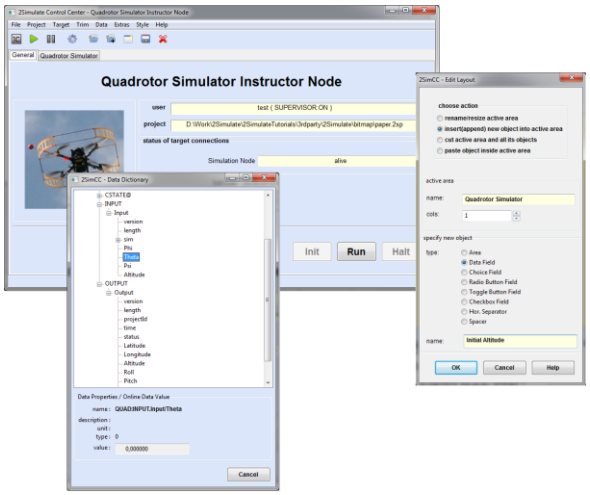

Figure 8 Instructor Node with 2SimCC

It can be presented as a reconfigurable front end for 2SimRT. In its General tab, one can track the status of 2SimRT Targets and Init, Run or Halt the simulations of these targets. It also provides utilities to add new tabs to visualize and edit values that are already defined in the signal specifications of the connected targets. It is also possible to access the values in the Common Database of the connected target and modify it during runtime via the Target Data Dictionary functionality.

5 Conclusion

This paper presents 2Simulate, a distributed real-time simulation framework. With its components 2SimCC, 2SimRT and 2SimMC it furnishes its users with tools and services to simulate complex real-time systems in a distributed fashion. As we identified that the basic pitfall of various other simulation frameworks has been their dependencies either on a modeling methodology or a domain architecture, our objective while developing 2Simulate has been to create a simulation framework that is independent of the domain architecture and the modeling methodology.

2Simulate is being employed as the underlying simulation framework of AVES rotorcraft and fixed-wing simulators with great success. The authors plan to extend the services and facilities of this infrastructure by supporting the commonly used distributed simulation standard IEEE 1516 High Level Architecture [15,16,17] and the emerging independent model interfacing standard Functional Mockup Interface [18].

6 References

[1] H. Duda, T. Gerlach, S. Advani and M. Potter, *Design of the DLR AVES Research Flight Simulator*, in AIAA Modeling and Simulation Technologies (MST) Conference, Boston, MA, 2013.

[2] D. Huang and H. Sarjoughian, *Software and Simulation Modeling for Real-Time Software-Intensive Systems*, in Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04), 2004.

[3] B. Zeigler, T. Kim and H. Praehofer, *Theory of Modeling and Simulation,* New York: Academic Press, 2000.

[4] Y. K. Cho, X. Hu and B. Zeigler, *The RTDEVS/CORBA Environment for Simulation-Based Design of Distributed Real-Time Systems,* Simulation: Transactions of The Society for Modeling and Simulation, Vol. 79, Nr. 4, pp. 197-210, 2003.

[5] J. Buck, S. Ha, E. Lee and D. Messerschmitt, *Ptolemy: A Framework for Simulating and Prototyping Heterogeneous Systems,* Int. Journal of Computer Simulation, Vol. 4, pp. 155-182, April 1994.

[6] R. Leslie, D. Geyer, K. Cunningham, P. Glaab, P. Kenney and M. Madden, *LaSRS++ An Object-Oriented Framework for Real-Time Simulation of Aircraft*, in AIAA Modeling and Simulation Technologies Conference, 1998.

[7] Mathworks, *Simulink: Simulation and Model-Based Design,* 05 Dec 2013. [Online]. Available: http://www.mathworks.com/products/simulink

[8] Applied Dynamics International, *ADvantage Framework,* 05 Dec 2013. [Online]. Available: http://www.adi.com/products/advantage/.

[9] Mathworks File Exchange, *PD Control Quadrotor - Simulink*, 05 Dec 2013. [Online]. Available: http://www.mathworks.com/matlabcentral/fileexchange/41149-pd-control-quadrotor-simulink.

[10] A. Samir, *Design and Control of Quadrotors with Application to Autonomous Flying*, Ph.D. Thesis, École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 2007.

[11] Mathworks, *Simulink® Coder™ Target Language Compiler,* 05 Dec 2013. [Online]. Available: http://www.mathworks.com/help/pdf_doc/rtw/rtw_tlc.pdf.

[12] Mathworks, *Simulink Coder: Generate C and C++ code from Simulink and Stateflow Models,* 5 Dec 2013. [Online]. Available: http://www.mathworks.com/products/datasheets/pdf/simulink-coder.pdf.

[13] DLR, *2Indicate - Flexible Visualisierung technischer Prozesse*, 05 Dec 2013. [Online]. Available: http://www.dlr.de/tm/desktopdefault.aspx/tabid-3015/7941_read-6822.

[14] V. Kuehne and P. Nartz, *OpenSceneGraph Reference Manual v2.2*, Ann Arbor, MI: Blue Newt Software, 2007.

[15] IEEE, *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)– Object Model Template (OMT) Specification,* 2010.

[16] IEEE, *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)– Framework and Rules,* 2010.

[17] IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification,* 2010.

[18] MODELISAR Consortium, *Functional Mock-up Interface for Co-Simulation Version 1.0*, 2010, [Online] Available https://fmi-standard.org/.