Running High Level Architecture in Real-Time for Flight Simulator Integration

Torsten Gerlach[1], Umut Durak[2], Alexander Knüppel[3]
DLR, Institute of Flight Systems, Braunschweig, 38108, Germany

and

Tim Rambau[4]
DLR, Institute of Flight Guidance, Braunschweig, 38108, Germany

DLR's Air Vehicle Simulator (AVES) is designed such that interchangeable cockpits of an EC135 rotorcraft and an A320 airplane can be operated either on a motion or on a fixed-base platform. While the EC135 and the A320 simulators are usually deployed standalone, there are emerging requirements to conduct simulator experiments that both simulators are involved in. Further demands include adding a traffic server for computer generated aircraft and a tower simulator to the simulation facility. To tackle the challenge of developing a complex distributed network simulation, the High Level Architecture (HLA) and Aviation-SimNet are selected as the underlying standards. As the AVES is a flight simulation facility with hard real-time constraints, the chosen HLA implementation, which is CERTI from ONERA, has to fulfill the deterministic processing and data exchange requirements. CERTI had to be ported to the AVES's real-time operating system QNX, and integrated into the core software infrastructure, the AVES Software Development Kit. To prove the usability of the real-time implementation, the worst case transfer times are measured for typical scenarios in order to validate the solution. Finally, a full scale implementation is carried out in AVES.

Nomenclature

ACT/FHS	=	Active Control Technology/Flying Helicopter Simulator
ADS-B	=	Automatic Dependent Surveillance - Broadcast
API	=	Advanced Technology Research Aircraft
ATRA	=	Advanced Technology Research Aircraft
AVES	=	Air Vehicle Simulator
DIS	=	Distributed Interaction Simulation
DLR	=	Deutsches Zentrum für Luft- und Raumfahrt e.V.
DMAN	=	Departure Traffic Manager
HLA	=	High Level Architecture
ICAO	=	International Civil Aviation Organization
IC	=	Interface Computer
FOM	=	Federate Object Model
MOM	=	Management Object Model
NTP	=	Network Time Protocol
ONERA	=	Office National D'Etudes et de Recherches Aerospatiales

[1] Group Leader, Flight Dynamics and Simulation, Lilienthalplatz 7, 38108 Braunschweig, Germany, torsten.gerlach@dlr.de.
[2] Research Scientist, Flight Dynamics and Simulation, Lilienthalplatz 7, 38108 Braunschweig, Germany, umut.durak@dlr.de, AIAA Member.
[3] Scientific Assistant, Flight Dynamics and Simulation, Lilienthalplatz 7, 38108 Braunschweig, Germany, alexander.knueppel@dlr.de.
[4] Research Scientist, ATM-Simulation, Lilienthalplatz 7, 38108 Braunschweig, Germany, tim.rambau@dlr.de.

OS = Operating System
SOM = Simulation Object Model
QNX = Unix based real-time operating system
QoS = Quality of Service
RTIA = Runtime Infrastructure Ambassador
RTIG = Runtime Infrastructure Gateway
RTOS = Real-Time Operating System
SDK = Software Development Kit
TCP = Transmission Control Protocol
UDP = User Datagram Protocol
WCET = Worst Case Execution Time
WCTT = Worst Case Transfer Time
XML = Extended Markup Language

I. Introduction

FOR more than 30 years, the aeronautics research community has been making use of simulators for developing and experimenting with advanced concepts and conducting human factor research. After having operated fixed-base simulators of an airplane and of a rotorcraft for the last two decades,[1,2] the DLR's Institute of Flight Systems is now operating a modern research simulator facility, the Air Vehicle Simulator (AVES) in Braunschweig, Germany.[3] The AVES has interchangeable cockpits of two of DLR's research aircraft, the EC135 ACT/FHS rotorcraft and the A320 ATRA airplane. Both can be operated on a motion- or on a fixed-base platform according to the particular needs. The two simulators of the AVES are usually deployed alone to exercise particular concept studies for the rotorcraft or fixed-wing aircraft. However, in order to run multi-actor scenarios in the AVES that target experiment interactions among the flying agents as well as the ones on ground, there are emerging requirements to conduct simulator experiments that both simulators, a traffic server for computer generated aircraft and a tower simulator are involved in.

Providing such a real-time human-in-the-loop networked simulation environment is challenging. High Level Architecture (HLA)[4,5,6] is selected as the underlying standard of networking heterogeneous simulation participants based on the recommendations of the AviationSimNet.[7] The simulation data exchange model is constructed based on the Federation Object Model (FOM) proposed by AviationSimNet. The open source Run-time Infrastructure (RTI) CERTI[8] from ONERA for real-time distributed simulation is selected as the underlying middleware. Its as-is performance has been investigated against the real-time constraints of a flight simulator integration using the AviationSimNet FOM-based federation. For that, CERTI was ported to the QNX Real-Time Operating System (RTOS), which is being used in the AVES.

CERTI is easily accessible by the AVES users by a wrapper, called SimNet. It provides a simplified programming interface based on the AviationSimNet FOM by utilizing the standard CERTI application programming interface (API). In order to enable the development of real-time federates, SimNet is further adapted to a real-time task, which is now part of the 2Simulate real-time framework. 2Simulate is the underlying real-time infrastructure of the AVES Software Development Kit (AVES SDK).[9]

In order to collect evidence towards the validation of the real-time capabilities of the proposed approach, the Worst Case Transfer Times (WCTT) are measured for typical scenarios with test federates in order to validate the solution against real-time constraints. Finally, a full scale implementation is carried out in AVES which includes both AVES simulators, the traffic server and the tower simulator.

The paper starts with a brief introduction to the HLA. After presenting the basics of real-time, it discusses the requirements for developing an HLA real-time implementation. Then the proposed approach to run the HLA in real-time is explained. Following the section that describes how this approach is incorporated in the AVES software infrastructure, the full scale implementation and validation studies will be presented.

II. Running HLA in Real-Time

A. Introduction to HLA

The HLA is a general purpose architecture which offers data exchange between distributed simulations. The most recent HLA Standard is defined under IEEE Standard 1516-2010 and defines HLA as an architecture developed to address the continuing need for interoperation and composability of new and existing simulations.[4,5,6] This

enables the reusability of a well-defined set of components, that communicate data and synchronize actions regardless of the underlying computing platforms.

The HLA was first initiated by the Defense Modeling and Simulation Office in 1995 and builds upon the results of the Distributed Interactive Simulation effort.[10] HLA is not software but a framework consisting of a set of specifications, respectively basic components:

- *HLA Framework and Rules*: the rules that define the basic principles underlying the HLA
- *Object Model Template (OMT):* the OMT provides a standard format for describing information of common interest to more than one simulator (federate)
- *Interface Specification (IF):* the IF defines the interface to the Runtime Infrastructure (RTI) that provides the means for simulators to coordinate the execution and exchange of information.

Typically, an HLA simulation consists of a collection of simulators interconnected through the RTI. Each simulator is referred to as a *federate*. The set of federates interacting through the RTI is referred to as a *federation*.

1. Federate and Federation

Federates are single applications with a single connection to the RTI. Thus, they can be seen as a unit of reuse and should be designed accordingly. Each federate conforms to the HLA standard via the interfaces that are specified in the HLA Federate Interface Specification. This is documented in each federate's Simulation Object Model (SOM). A federate can be a data consumer, producer or both. For example, a federate can represent an arbitrary aircraft in an aggregate level simulation, which exchanges data through an RTI with other federates. A named set of federate applications is then called a federation, if it shares a common specification of data exchange that is specified in the Federation Object Model (FOM) to achieve some specific objective. The execution of a simulation is also called a *Federation Execution*, i.e. the actual operation, over time, of a set of *joined federates* that are interconnected by an RTI.

2. Runtime Infrastructure

The RTI is middleware that enables and supports the inter-federate communication through a set of services and can be viewed as a special purpose distributed operating system. It supports the HLA rules with the services it provides over the interfaces specified in the IF during a runtime execution, but is not itself part of the specification. However, modern RTI implementations conform to the IEEE 1516 specifications.

3. Object Models

The HLA framework defines three types of object models. First is the Simulation Object Model which is a specification of an individual federate for the types of information it provides to other federates and is able to receive from other (joined) federates.

Second is the Federation Object Model (FOM). The FOM defines all the object classes, object class attributes, interaction classes, interaction parameters and other relevant information that can be shared at runtime among a federation and thus is public to each joined federate. In IEEE 1516 and IEEE 1516-2010 the FOM is passed to the RTI as an XML-file.

Third is the Management Object Model (MOM), which is a collection of predefined constructs enabling support for monitoring and controlling a federation execution. The OMT is a standardized and structured template for specifying both SOMs and FOMs.

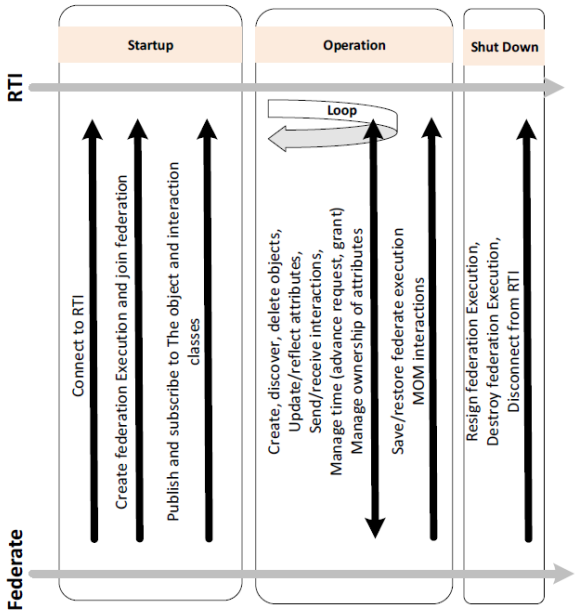

Figure 1. HLA Simulation Flow.[11]

4. Simulation Flow

Figure 1 shows the typical federation execution lifecycle of an individual federate. At startup the federate connects to the RTI, creates a federation execution (if it does not already exist) and joins it. After that, the federate will inform the RTI about its capabilities and interests in the simulation and therefore subscribes and/or publishes to the relevant object and interaction classes. In the operation phase (the main loop), federates may create new objects or discover objects created by other

federates. They also may delete their own created objects, may receive updates for subscribed attributes or may provide updated attributes. The time management of each individual federate is done by using the RTI time management services, if a time management policy is specified. In the shutdown phase, the federate resigns from the federation. The last resigning federate destroys the federation execution eventually. Finally, the federate disconnects from the RTI.

B. Real-Time Scheduling

The objective of scheduling is to provide a policy for ordering the execution of outstanding tasks on the processor. Such a schedulable set of tasks is termed a feasible schedule. In real-time systems, the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced.[12] Each task in a feasible schedule is therefore given a timing constraint within which it needs to respond. The maximum valid response time of a task is termed a deadline. Between the invocation of a task and its deadline, a task needs a certain amount of computation time for its execution. In a hard real-time system, a task needs to meet a deadline deterministically. Hence, the following relationship should be guaranteed for all tasks:

$$C \leq D$$

where C is the computation time and D is the deadline.

Hard real-time systems are used in fields where meeting the deadline of a task is of utter importance and can lead to catastrophic behavior otherwise. For example, an in-flight computer system controlling the landing of an airborne plane may cause a crash if a deadline is missed. In contrast, a loss of a frame in a video game may only lead to a glitch. A statistical distribution of response times in a soft real-time system, i.e. an occasional miss of a deadline, is therefore acceptable.

C. Real-Time Requirements for Simulations

In man-in-the-loop simulations, especially in flight simulation, a hard real-time system is desirable. Any non-flight-system-related delay between input and actuation is perceived as unnatural and distracting. Hence, to satisfy the operator's sensation, tasks in such a simulation must be computed within 20-50 ms.

This is even more critical in flight simulations where the computation frequency of aircraft dynamics, the motion platform and the visual system must be at 50-60 Hz. This directs to a computational period of less than 20 ms. This is a worst-case value and therefore a hard constraint and should never be exceeded in order to ensure a correctly perceived flight simulation.

If the accumulation of the worst-case performance of all tasks lies below the hand-eye coordination response of a human, the simulation meets these real-time requirements.

D. Real-Time HLA

A distributed real-time simulation implies that a real-time execution must meet a deadline as an absolute requirement in order to ensure the correctness or the fidelity of the simulation. However, some earlier reports noted that the HLA standard's performance suffers from a big infrastructure overhead in order to be useful for wide area distributed simulations.[13] Some of the overhead is due to network latency. But a greater part is suspected to be caused by the underlying middleware, namely the RTI. Indeed, HLA does not provide interfaces to specify end-to-end Quality of Service (QoS) requirements for interoperability among federates. Furthermore, the HLA standard only supports two transportation protocols: Best Effort and Reliable, usually implemented through well-established ethernet communication protocols like the User Datagram Protocol (UDP) and the Transport Control Protocol (TCP).

These protocols are normally sufficient for many applications in the simulation community, but in general are not suitable for real-time constraints, which require a higher communication performance.

In order to run a distributed simulation using HLA in real-time, the following requirements are applicable:[14]

- QoS policies shall be able to be specified at the RTI level for the interoperability among federates. This includes a guaranteed latency bound, cycle times, jitter and bandwidth.
- The underlying infrastructure (operating system, hardware) shall also support the real-time performance requirements.
- A federation-wide synchronized clock, which can be observed with a known maximum error by each federate, shall exist. This leads to a common notion of real-time and Worst Case Execution Time (WCET) and ensures that deadlines are meaningful throughout the execution of the simulation.

If these requirements are fulfilled, the federation can be pronounced as real-time. There have been some efforts that aim at integrating these real-time requirements into the middleware (RTI).[15-18] However, these efforts did not lead to standardization and HLA still fails to grant standardized support for real-time applications.

Notwithstanding, in this effort we do not concentrate on enhancing the RTI with these requirements, rather we would like to stay conformant to the standard. Therefore, we aim to run the HLA standard in a real-time environment and evaluate how well it achieves real-time constraints. This will be done by providing a real-time environment, constructing a plausible simulation including several federates and measuring the WCTT.

E. Running HLA in a Real-Time Environment

There are some previous efforts that report utilization of HLA for real-time flight simulators. In the SIMULTAAN project, a component architecture is proposed for simulator development based on HLA.[19-22] The runtime communication infrastructure of SIMULTAAN provides not only communication between simulation components by utilizing RTI, but also extends RTI with a time-triggered (real-time) scheduling mechanism based on a synchronized wall clock.

In the last decade, the French Aerospace Laboratory (ONERA) has been spending some efforts on providing a real-time simulation approach with their open-source RTI, CERTI, which is originally applied to flight simulators.[14,23,24] CERTI is an HLA-compliant RTI with a unique architecture of communication processes: The Run-Time Infrastructure Gateway (RTIG) as a global process, the Run-Time Infrastructure Ambassador (RTIA) as a local process and the libRTI library, which is linked to the federate's process. They reported possible real-time problems with the current CERTI implementation and discussed their real-time RTI vision. They proposed extensions to utilize the processor load management for assigning CPUs to federates, extensions for RTIA and RTIG processes, scheduling algorithms in RTOS and lock mechanisms for disabling memory paging into the address spaces of calling processes. They then provided a WCTT analysis for their implementation in a flight simulator scenario.

Having read ONERA's work, in this study we decided first to evaluate ONERA's open-source RTI, CERTI, as-is without any particular modifications for real-time execution, but rather on an RTOS driven by DLR's real-time simulation framework 2Simulate. While the federate execution is scheduled by 2Simulate as a separate runnable process (task) with its own frequency and priority, a new runnable process class is developed for the tick() mechanism so that the HLA communication can also be scheduled by a respective priority and frequency with the federate execution. The scheduling algorithm of choice then takes care of the control of the execution. While it is possible to assign CPU affinity to processes in QNX, it is not implemented in this study and RTOS is allowed to manage the thread affinity automatically.

III. HLA in AVES Software Infrastructure

A. AVES Software Infrastructure

As AVES is designed as a research flight simulation facility, the software infrastructure is flexible and extendable. All core components are developed in-house and either use a DLR software product or are proprietary standalone runnable modules. AVES's main philosophy is to reduce time-to-simulation in a user friendly manner. All software components shall be platform independent according to their use case. The simulation user, e.g. aerospace engineers, shall be able to develop at the desktop, run and test in software-in-the-loop and hardware-in-the-loop environments, and deploy to the simulator independent from the operating crew. To achieve this goal, DLR designed the AVES Software Development Kit (AVES SDK), a full set of software modules and runnable programs from the real flight simulation facility.[9] The infrastructure contains a set of rules for the user and provides space for runtime usage and different development stages. Together with a source code management system and configuration management, the AVES SDK is an evolving environment with the ability to create reproducible and standardized software versions and to support experimental software setups with the ability to connect the simulation to the development environment during runtime.

A flight simulation appears to be a heterogeneous system with a real-time constraint for most of the components. This distributed character is typical for a large simulation system with involved hardware. Figure 2 shows the current structure of the A320 simulation infrastructure.

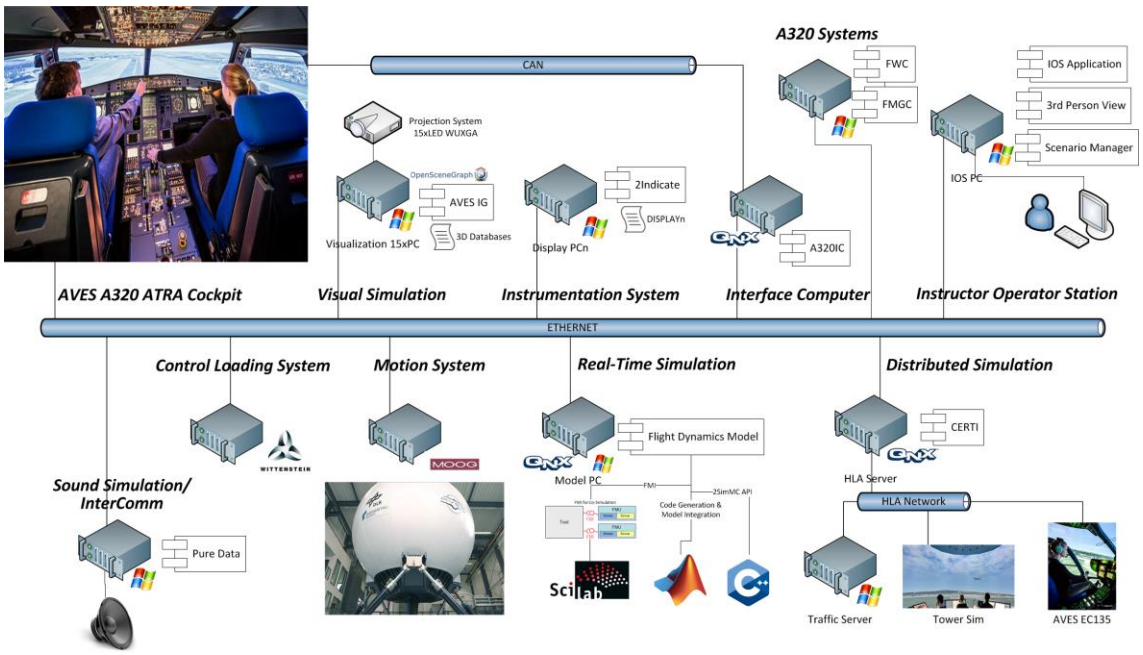

Figure 2. AVES A320 Simulation Infrastructure.

The data exchange is done via Ethernet connections in separated private segments. Because of the small frame rates, UDP was chosen for data distribution. To avoid large overheads, software protocols are proprietary and tailored to the use case. One of the main components is the so called Interface Computer (IC). It collects and distributes most of the data to all peripheral hardware, e.g. the motion system. This module must run in hard real-time on a separate computer. Connecting other simulations via HLA is also done here. For keeping real-time constraints, either hard or soft, a framework is necessary to form the foundation of the AVES flight simulation.

B. 2Simulate

2Simulate is a platform-independent C++ real-time framework to facilitate the integration and connection of heterogeneous simulation modules.[25] The main philosophy is to offer an easy way to develop real-time applications on a non-expert level. It is possible to use a graphical user interface for controlling the simulation and a model integration and control interface. The three components are 2Simulate Real-Time Framework (2SimRT), 2Simulate Model Control (2SimMC) and 2Simulate Control Center (2SimCC).

2SimRT is the core library with a large set of real-time-capable tasks, which are able to process incoming and outgoing data from various hardware interfaces. Most important for AVES applications are the *TSimUdpTask* for Ethernet data exchange and the *TSimSimpleTask* for running any program code in a real-time environment. The library consists of a set of header files, a precompiled library and third party software. Everything can be used either under Microsoft Windows, for soft real-time execution, or QNX, for hard real-time execution. A central task supervises the real-time execution and scheduling. The real-time data are administrated inside a common database, namely the data dictionary. An application that is based on 2SimRT can be connected to a 2SimCC application, which is the graphical user interface.

2SimCC acts as a control center for various targets. The 2Simulate target is a 2SimRT application using the *TSimConTask* for exchanging data with the control center. A control center application has access to the data dictionary to present or manipulate data during runtime; it controls the simulation status and can perform online trimming of connected models using the 2Simulate model control interface.

2SimMC is a generic interface for integrating models into the simulation. It delivers a set of modules to connect a real-time model to a 2SimRT application and enables the control interface from 2SimCC. The model control framework is supporting models in several different modeling languages. AVES widely uses MATLAB/Simulink for modeling flight dynamics systems. Therefore, 2SimMC offers a complete set of Simulink Coder Target Language Compiler files (TLC files) for automatic code generation. Using all these possibilities, a complete automatized model integration process was developed.[26]

As 2Simulate is a flexible object-oriented framework, it is easy to extend the functionalities with minor effort. Implementing the HLA protocol itself was not necessary. Facilitating the AviationSimNet API is a more effective method.

C. Easier API for AviationSimNet

To enrich its simulator connectivity, DLR's Institute of Flight Guidance joined the AviationSimNet community a couple of years ago, bringing HLA to its simulators. AviationSimNet publishes an open specification that defines the rules, standards and guidelines for interoperating air traffic management simulations. Included in this specification is a Federation Object Model which contains object definitions for aircraft movement, flight plans and simulation control. This FOM was chosen as the basis for the first HLA tests in the simulators. As is typical for HLA implementations the RTI is not specified inside AviationSimNet and is up to each party. To get a broad base for application and development, the RTI used in the simulators should support at least C++ and Java. With this in mind, and with a preference for open source and therefore low costs per client, the CERTI RTI from ONERA was chosen and is used as basis for the HLA development.

As AviationSimNet serves as the underlying standard, an object-oriented HLA wrapper library, namely simNet, was developed to hide and standardize specific parts of the HLA and lower the threshold for other developers to use HLA in their projects. All objects from the FOM are mapped to equivalent code objects equipped with the usual setter and getter methods. Inside these methods all HLA communication is encapsulated and selected automatically, in particular for ownership management of the attributes. First speed and reliability tests were conducted, adding simNet to the control of the visual system and connecting the cockpit simulator with the radar simulator. After a final fine tune simNet was introduced to the institute's own MATLAB-based departure manager (DMAN)[27] and the surface manager[28] written in Java, interconnecting air traffic controller tools and air traffic simulation. For easier debugging and testing an additional simNetViewer was written. Via this viewer it is possible to monitor and edit every object attribute of the FOM to test the behavior or simulate unconnected clients. Last but not least, the AviationSimNet FOM was extended with additional objects mainly for the application in the Institute's of Flight Guidance fixed-base simulator, e.g. weather layer and cloud areas. Also some more attributes to the flight plan object to enhance the support for future research applications were added.

D. 2Simulate SimNetTask

After providing the 2Simulate core library and the simNet wrapper library, a federate class for handling aircraft objects was developed. Each federate is associated with an abstract task in 2Simulate called *TSimHLATask*. A *TSimHLATask* extends the abstract task *TSimRtTask*, which handles the real-time scheduling of a periodical executing task. There are three phases, a *TSimRtTask* has to handle: (1) the initialization, (2) the synchronization to insure the correct start-point for all corresponding real-time tasks and (3) the run phase. 2Simulate offers user callback routines for every phase that can be overwritten by inherited classes.

The real implementation of *TSimHLATask*, using SimNet as the communication layer, is the *TSimSimNetTask*. This task needs to participate in the RTIG and will either join a federation or create a new one if the specified one does not exist, see Section I. A. for more information about the simulation workflow. A federation is simply associated with a name (here: *simNet*) besides the network location of the RTIG. Because one federate can publish or subscribe to multiple objects, we normally restrict each federate to exactly one executable/simulation.

The aircraft information that is shared with or read from the federation is stored in a special data structure, namely the

```
TSimHLATask::OBJData *send =  new TSimHLATask::OBJData(),
                     *receive = new TSimHLATask::OBJData();

//################################
// callback
//################################
void doSomethingCB( TSim *pAppl, TSimRtTask *pRtTask ) {
    send->pm_dLatitude += ...;
    if(simNetTask->isUpdated( receive->pm_iAddrModeS )) { ... }
}

[...]

//################################
// simNet task
//################################
TSimSimNetTask* simNetTask = new TSimSimNetTask( pTSim, "simNet",
                                    TASK_SCHED_RR, 30, 20*iMSECtoNSEC, true, true, true );

//################################
// participate
//################################
simNetTask->participate( "127.0.0.1", "simNet" );

send->pm_iAddrModeS = ...;
receive->pm_iAddrModeS = ...;

simNetTask->publish( send );
simNetTask->subscribe( receive );

simNetTask->setPreProcCB( (void(*)(TSim *,TSimRtTask *))&doSomethingCB );

//---- start all tasks ----
pTSim->start( 10000, 0 );
delete pTSim;
```

Figure 3. Code Excerpt Sample of a 2Simulate Federate.

TSimHLATask::OBJData. This consists of aircraft standard parameters like geographic position, attitude or airspeed and uses a unique number for identifying the aircraft. The identification number is chosen according to the mode S transponder ICAO 24-bit address code.

A code excerpt of the instantiation and processing of a sample federate is presented in Figure 3. After creating the task with a 20 ms periodicity and participating in a federation, we can create the aircraft objects. Published aircraft objects are created and handled by the federate itself and as such can be manipulated or deleted (using unpublish). On the other hand, subscribed objects are created and owned by other federates, thus they are only readable. If a subscribed object does not exist or was removed, no further updates will be received on this particular object.

In specified callback functions, each federate can manipulate its published aircraft objects or receive updates of other aircraft objects in the federation.

The task itself recognizes changes of published objects and therefore will only commit these changes at the end of each period. This may lead to a lower network throughput. Other federates may check whether values were changed on particular objects using *isUpdated(…)* and can work with the changed values from there.

IV. Implementation and Validation

A. HLA in AVES

In the past AVES used a proprietary but easy-to-use Ethernet protocol to connect different simulations in DLR's company network. Extending 2Simulate with a standardized interface like HLA is a more powerful approach and opens up a wider range of distributed applications. The starting application is the introduction of the AVES TrafficServer for simulating realistic non-human-controlled air traffic. This feature supports the wake vortex research activities of the Institute of Flight Systems. Additional airplanes act as wake vortex generators. The AVES A320 flight dynamics model is equipped with an aerodynamic interaction model for simulating the impact on those vortices. After the successful setup of the TrafficServer, other federates can easily join. The AVES helicopter simulation is connected and the tower simulation of the Institute of Flight Guidance is also connected to the federation. To identify each simulator and its corresponding aircraft in the virtual airspace, a unique number is assigned.

As the data distribution in the A320 simulation is done by a 2Simulate-based interface computer, a new *TSimSimNetTask* was introduced and connected to the federation. The interface computer is time critical and is therefore an RTOS application. To keep hard real-time constraints, the RTIG is installed on the same hardware. The A320 simulation publishes its own data via the IC. All other federates from the HLA server are requested and updated frequently. The data exchange does not only contain the position and attitude of the aircraft for visualization. It also represents a data exchange via ADS-B, to allow the possibility of simulating realistic system behavior. Figure 4 describes the layout of the current HLA federation and its participants.

The AVES federation forms a heterogeneous network of participants with specific needs. The A320 ATRA and EC135 ACT/FHS simulations need hard real-time and deterministic behavior, whilst the AVES TrafficServer and the tower simulator are only generating other traffic so they do not have to be hard real-time applications. To ensure the deterministic behavior of the simulation and prove that an RTOS is capable of running an HLA server application, the worst case transfer time (WCTT) had to be measured.

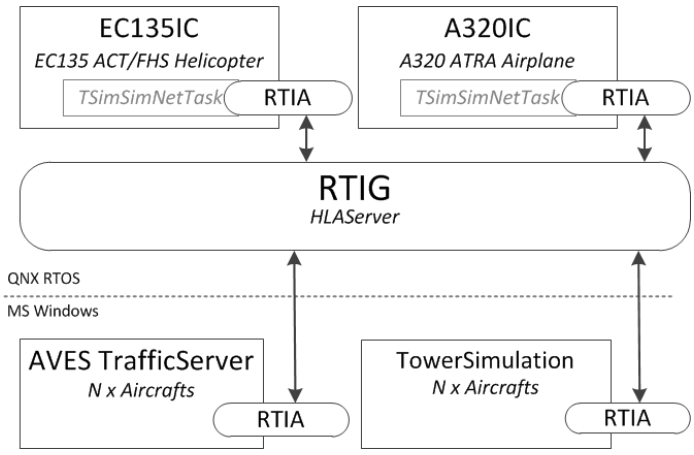

Figure 4. AVES HLA Federation.

B. Worst Case Transfer Time Measurements

The calculation of the WCTT is a benchmark test to determine the maximum throughput on specific configurations and test scenarios. This is an important property to evaluate, as our approach needs to be reliable in terms of real-time capability. In our case, we evaluated a scenario where federation *Fed1* publishes an object, and federate

Fed2 subscribes to it. We then measured the time interval between *Fed1* sending an update on that object and *Fed2* receiving it. *Fed1*, *Fed2* and the RTI are all on different nodes in our network. Therefore, the measurement is divided into three network phases: First, send data from *Fed1* over RTIA to the RTIG. Second, the RTIG receives the data and sends it to the RTIA of *Fed2*. Finally, *Fed2* receives the data from its RTIA. The sum of times is the transfer time of a specific instance *i* and denoted by WCTT(i):

$$WCTT(i) = WCTT(RTIA(Fed1) \rightarrow RTIG) + WCTT(RTIG \rightarrow RTIA(Fed2)) + WCTT(RTIA(Fed2) \rightarrow Fed2)$$

Figure 5 visualizes one iteration where the WCTT is measured. Note that WCET refers to the Worst Case Execution Time. *Fed1* runs with a frequency of 20 Hz whereas *Fed2* runs with a frequency of 1000 Hz.

For the clock synchronization, the Network Time Protocol (NTP) was used. Before each run a synchronization event by an independent NTP server was triggered to ensure the same time on all network nodes.

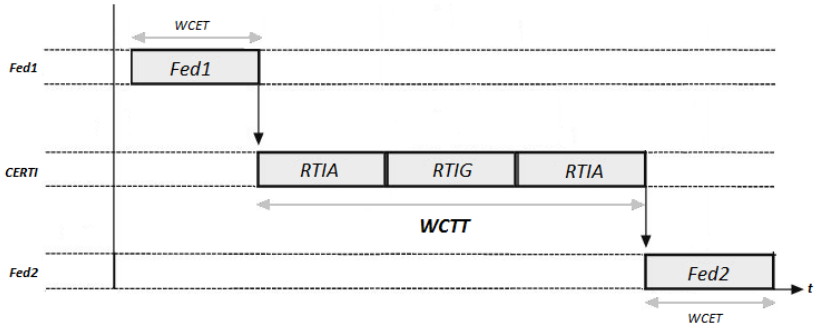

Figure 5. WCTT measurement iteration step.

Every time *Fed2* recognizes an update event, the time between sending and receiving is measured. As *Fed2* runs at 1000 Hz there is a bias of 1 ms in the results. The transferred object data itself have a fixed size of 216 bytes (1728 bits).

We repeated the experiment eight times with 100 update events per repeat and used a distributed network infrastructure. Each federate, as well as the RTIG, was therefore running on a different network node. Each node used QNX as operating system. The minimum, average and worst case transfer times from *Fed1* to *Fed2* were then measured per run. The following table gives an overview of our results:

Table 1. Measured data transfer times.

	Run 1	Run 2	Run 3	Run 4	Run 5	Run 6	Run 7	Run 8	**Ø**
Minimum	0.1 ms	1.2 ms	0.3 ms	0.7 ms	0.2 ms	0.4 ms	0.8 ms	0.1 ms	**0.5 ms**
Average	0.8 ms	2 ms	1 ms	1.5 ms	0.9 ms	1.7 ms	1.7 ms	1.4 ms	**1.5 ms**
WCTT	2 ms	6 ms	2 ms	2 ms	2 ms	4 ms	3 ms	2 ms	**2.9 ms**

As depicted in table 1, the WCTT of all runs occurred in the second run with a measured time of 6 ms. The average WCTT is roughly 3 ms. Both are faster than the simulator's minimum update frequency of 60 Hz (~16 ms per frame).

V. Conclusion

High Level Architecture was successfully introduced to the Air Vehicle Simulator's infrastructure. This powerful standardized concept was implemented and extended to real-time capabilities. As CERTI is open source and simNet is an accessible AviationSimNet implementation, it was possible to port the source and develop a QNX version. The real-time framework 2Simulate was extended with new features to easily create a connection to an RTIG. The current setup of the AVES federation includes the A320 ATRA and EC135 ACT/FHS flight simulations. A traffic server for realistic air traffic was also introduced. The A320 simulation can especially benefit from these new capabilities. By connecting the tower simulator of the Institute of Flight Guidance, an overall distributed simulation can be generated.

To keep real-time constraints, a WCTT measurement with the 2Simulate implementation was conducted. The results show a fast processing and data transfer rate that is above the minimum frequency of the simulation system. This encourages us to setup and use an HLA real-time infrastructure.

Future applications for AVES being part of an HLA federation are studies with DLR-external simulators and the integration of unmanned aircraft for teaming operations. Together with the tower simulation of the Institute of Flight

Guidance, the AVES can now provide an immersive simulation environment with pilot-in-the-loop and controller-in-the-loop capabilities.

References

[1]Saager, P., "Real-Time Hardware-in-the-Loop Simulation for 'ATTAS' and 'ATTHeS' Advanced Technology Flight Test Vehicles," *AGARD Guidance and Control Panel*, 50th Symposium, 22-25 May, Izmir, Turkey, 1990.

[2]Klaes, S., "ATTAS Ground Based System Simulator -An Update-," *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Denver, CO, 2000.

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

[4]IEEE, *IEEE Standard for Modeling and Simulation High Level Architecture (HLA)– Framework and Rules*, IEEE Std. 1516-2010, New York, NY, 2010.

[5]IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)– Federate Interface Specification*, IEEE Std. 1516.1-2010, New York, NY, 2010.

[6]IEEE, *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specification*, IEEE Std. 1516.2-2010, New York, NY, 2010.

[7]Bodoh, D.J., Brown, P.L., Liguori, P.A. and Pollack, M.E., "AviationSimNet™ Specification," 2005.

[8]Noulard, E., Rousselot, J. Y. and Siron, P., "CERTI, an Open Source RTI, why and how," *Spring Simulation Interoperability Workshop*, San Diego, CA, 2009..

[9]Gerlach, T. and Durak, U., "AVES SDK: Bridging the Gap between Simulator and Flight Systems Designer," *AIAA Modeling and Simulation Technologies Conference*, Dallas, USA, 2015.

[10]IEEE, *Protocols for Distributed Interactive Simulation Applications--Entity Information and Interaction*, IEEE Std. 1278-1993, New York, NY, 1993

[11]Topcu, O., Durak, U., Oguztuzun, H. and Yilmaz, L., *Distributed Simulation – A Model Driven Engineering Approach*, Springer International Publishing, Cham, Switzerland, 2016

[12]Stankovic, J. A., "Misconceptions about real-time computing," *IEEE Computer*, Vol.10, 1992, pp.10-19.

[13]Purdy, S. and R. Wuerfel, "A comparison of HLA and DIS Real-Time Performance," *Proceedings of 1998 SPRING SIW Conference*, Orlando, FL, USA. 1998.

[14]Adelantado, M., Siron, P. and Chaudron, J.B., "Towards an HLA run-time infrastructure with hard real-time capabilities," *Spring Simulation Interoperability Workshop*, Ottawa, Canada, 2010

[15] Bachinsky, S, Noseworthy, J. and Hodum, F. "Implementation of the next generation RTI," *Spring Simulation Interoperability Workshop*, Orlando, FL, 1999.

[16]Zao, H. and Georganas, N.D., "Architecture proposal for realtime RTI," *Simulation Interoperability Workshop*, Orlando, FL, 2001.

[17]McLean, T., Fujimoto. R.M. and Fitzgibbons, B., "Middleware for real-time distributed simulations," *Concur Comput Pract Exp* Vol.16, No. 15, 2004, pp. 1483–1501.

[18]Boukerche, A. and Kaiyuan, L., "A novel approach to real-time RTI based distributed simulation system," *38th Annual Symposium on Simulation*, San Diego, CA, 2005.

[19]Kuijpers, N., Van Gool, P. and Jense, H., "A component architecture for simulator development," *Spring Simulation Interoperability Workshop*, Orlando, FL, 1998.

[20]Huiskamp, W., Janssen, H. and Jense, H., "An HLA based flight simulation architecture," *AIAA Modeling and Simulation Technologies Conference*, Denver, CO, 2000.

[21]Lemmers, A., Kuiper, P. and Verhage, R., "Performance of a component-based flight simulator architecture using the HLA paradigm," *AIAA Modeling and Simulation Technologies Conference*, Monterey, CA, 2002

[22]Jansen, R., Huiskamp, W., Boomgaardt, J. and Brassé, M., "Real-time Scheduling of HLA Simulator Components," *Spring Simulation Interoperability Workshop*, Orlando, FL, 2004.

[23]Gervais, C., Chaudron, J.B., Siron, P., Leconte, R., Saussié, D., "Real-time distributed aircraft simulation through HLA," *IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, Dublin, Ireland, 2012

[24]Chaudron, J.B., Saussié, D., Siron, P., Adelantado, M., "Real-time distributed simulations in an HLA framework: Application to aircraft simulation," *Simulation*,Vol.90, No.6, 2014, pp.627-43.

[25]Gotschlich, J., Gerlach, T. and Durak, U., "2Simulate: A Distributed Real-Time Simulation Framework," *ASIM STS/GMMS Workshop 2014*. Reutlingen, Germany, 2014.

[26]Gerlach, T., Durak, U. and Gotschlich, J., "Model Integration Workflow for Keeping Models up to Date in a Research Simulator," *Simultech 2014*, Vienna, Austria, 2014.

[27]Schaper, M., "Operational Improvements in the Context of DMAN, A-SMGCS and A-CDM," *CEAS Conference*, Manchester, UK, 2009.

[28]Gerdes, I. and Temme, A., "Taxi Routing for Aircraft: Creation and Controlling – Ground Movements with Time Constraints," *SESAR Innovation Days*, Braunschweig, Germany, 2012.