DDS Based High Fidelity Flight Simulator

Shupeng Zheng, Jingfeng He, Jun Jin, Junwei Han
School of Mechatronics Engineering
Harbin Institute of Technology
Harbin, China
e-mail: hitzsp@163.com

***Abstract*—The real-time distributed computing environment and reusable software architecture are important factors that affect the fidelity of flight simulation. We accomplished a flight simulator based on DDS (Data Distribution Service for Real-time Systems) middleware and structural software architecture and proved its high fidelity as a flight training device. According to the analysis of flight simulator's functional blocks, we developed the real-time distributed computing environment which utilized DDS middleware through Ethernet and decreased the communication latency among functional blocks. Furthermore, we proposed the structural software architecture on the basis of layered and component-based design pattern to facilitate the higher simulation components' reuse and replacement. The real-time communication procedures among simulation components are also described in this paper. The validation method and the contrasting simulation results are presented finally to show the feasible design based on DDS to carry out flight simulation with low communication latency and high quality.**

Keywords-DDS; distributed computing; software architecture; flight simulation

I. INTRODUCTION

The aim of flight simulation is to produce and control animated images, sound reproduction, and device feedback in a manner as realistic and responsive as the real flight, and chasing this ideal has constantly pushed the flight simulator study forward in many different ways. Individual simulators have adopted techniques such as multi-processor systems, high performance graphics cards and distributed sensors and actuators to approach the desired objective.

Increasingly, flight simulator consists of many different high performance processing sub-system units that need to communicate in real time. And as flight simulator gets larger and more distributed the performance issues of latency and system bottlenecks are becoming ever more important in maintaining the simulation experience. A further issue is the critical need to ensure better simulation models in application software can be reused effectively across various types of flight simulator. The combination of these vital but difficult issues is driving the need for more formalized software architecture and a growing move towards commercial off the shelf (COTS) middleware adoption [1].

High level architecture (HLA) [2] has emerged as a widely adopted middleware standard for simulator-to-simulator connectivity and data sharing. But, unfortunately, the HLA wasn't designed to provide the speed and detailed control of real-time performance required by flight simulator that need consistently low latencies [3]. However, there is another middleware standard that provides a much better fit for the real-time requirements of flight simulator. The OMG Data Distribution Service (DDS) [4] is a data oriented middleware optimized for hard real time systems with the low latency and quality of service capabilities to provide the required speed and level of control of real-time performance built in.

Considering advantages of developing flight simulator based on DDS, we built an experimental Boeing 737-800 flight simulator. The simulation models and executable code in the simulator are designed as components with well-defined, source-level language independent interfaces which communicate through the DDS middleware on a PC cluster, thus composed a man-in-the-loop real-time distributed flight simulation system. By this means, the flight simulator becomes ease of maintenance, ease of development, and ease of reusability while the fidelity of the simulator is improved. This paper describes our work and shows the character of DDS based flight simulator.

II. CORE FEATURES OF DDS

At the core of DDS is the DCPS (Data Centric Publish/Subscribe) model, whose specification defines standard interfaces that enable applications running on heterogeneous platforms to write/read data to/from a global data space in a distributed real-time system. Applications that want to share information with others can use this global data space to declare their intent to publish data that is categorized into one or more topics of interest to participants. Similarly, applications that want to access topics of interest can also use this data space to declare their intent to become subscribers. The underlying DDS middleware propagates data samples written by publishers into the global data space, where it is disseminated to interested subscribers [5]. The DCPS model decouples the declaration of information access intent from the information access itself, thereby enabling the DDS middleware to support and optimize QoS enabled communication. As shown in Fig. 1, a canonical DCPS model is comprised of the following elements that provide functionalities for a DDS application to publish/subscribe to data samples of interest.

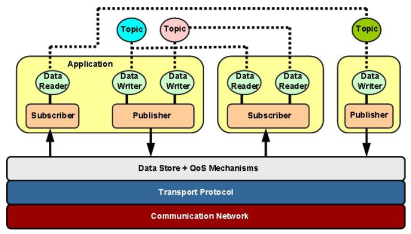


Figure 1. Architecture of DDS

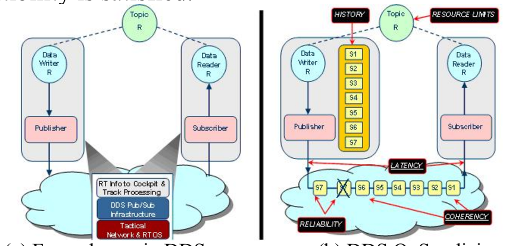
DataWriter and Publisher: A DataWriter is the actual object used to send data samples, and is always associated with a particular Topic. A publisher is used as a factory to create and manage a group of DataWriters with similar behavior or QoS policies.

DataReader and Publisher: A DataReader is the actual object used to receive data samples, and is always associated with a particular Topic. A subscriber is used as a factory to create and manage data readers.

Topic: A Topic consists of a data type and a name, and it connects a DataWriter with a DataReader. Data samples start flowing only when the Topic associated with a DataWriter matches the Topic associated with a DataReader.

Fig. 2 shows DDS capabilities that make it better suited than other standard middleware platforms. Fig. 2 (A) shows that DDS has fewer layers in its architecture which significantly reduces latency and jitter. The ability to specify quality of service parameters for each individual node or data flow is an important feature of DDS. There are Over 20 QoS categories which are supported by DDS, Fig. 2 (B) shows some DDS supplied QoS properties, such as the depth of the 'history' included in updates, the maximum latency of data delivery, the degree and scope of coherency for information updates, the reliability of data delivery, etc.

These properties can be configured at various levels of granularity (i.e., Topic, Publishers, DataWriter, Subscribers, and DataReader), thereby allowing application developers to construct customized contracts based on the specific QoS requirements of individual entities. The DDS middleware is responsible for determining whether QoS parameters offered by a publisher are compatible with those required by a subscriber, allowing data distribution only when compatibility is satisfied.



(a) Fewer layers in DDS (b) DDS QoS policies

Figure 2. Optimizations and QoS Capabilities of DDS

III. DDS Based Flight Simulator Set-up

Based on the DDS middleware used through Ethernet, we developed the flight simulator with six relevant sub-systems which run on the separate host computers. As indicated in Fig. 3, control loading system acquires pilot inputs and gives the force feedback to the pilot, flight system calculates the aircraft movements, avionics system incorporates the instruments and display logic, visual cueing system drives the image generator to reproduce the flight environment, and audio cueing system reproduces sound information. The instructor station is in charge of the pedagogical aspects of flight simulation. By using DDS middleware, each component of the sub-system publishes and subscribes data as indicated by arrows. The representative data elements distributed among components are elevator and rudder position (1a), aileron displacement (1b), flight training instruction (2) and aircraft state (3), etc.

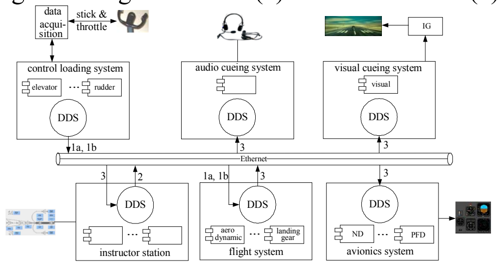


Figure 3. DDS based flight simulator set-up

IV. Simulator Construction

According to the set-up of flight simulator, we utilize the layered and component-based design patterns to construct a component based software architecture for flight simulator. For the sake of convenience to reveal the architecture and prepare for the next section to implement subsystem, we choose the flight system on this typical route to describe the building process.

Fig. 4 describes the function blocks of the flight system which are enclosed in the dash frame. The flight system is mainly composed of equation of motion, aerodynamic block, landing gear block, weight block and atmosphere block, etc. After received the rudder, flap position, landing gear position, and C.G. position from environment models, it will execute simulation computing following the instruction from instructor station and output the computing results to cueing systems. Wang Xinren [6] described the detailed mathematical model for each function block. In the following sections, the software architecture, data model and DDS based communication method for the flight system will be described in detail.

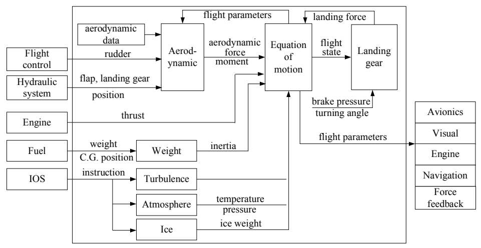


Figure 4. Function blocks of the flight system

A. Software Architecture

Flight simulator typically comprises tens of thousands of lines of code for the simplest training simulation, even over millions of lines of code for complex, high fidelity training simulation. In order to decrease the software development difficulty and achieve function block re-use, we proposed the structural software architecture on the basis of component technology. Fig. 5 shows the four main elements within the software architecture on the flight system simulation node: flight simulation components, the container, DDS middleware services, and the real-time scheduler.

Components manage the simulation algorithms and data flows that control the process. Components are strongly encapsulated objects that provide language-independent, opaque interfaces. Each component has three interfaces: input, update and output, separately carries out parameter input, executes simulation algorithm at a particular rate and parameter inputs.

The container is the real-time running environment for components. It provides PortConnector object to facilitate communication among components or outer simulation nodes. Besides that, a real-time scheduler is included in the container which controlles the application level threads, the container level thread and middleware level threads execute with appropriate priorities. The desired real-time behavior of simulation components can be achieved by static priority scheduling algorithm.

The DDS middleware will provide class libraries from which the container communication objects can be created. Publisher, DataWriter, Subscriber and DataReader are DDS service brokers in the container which publish or subscribe data for components.

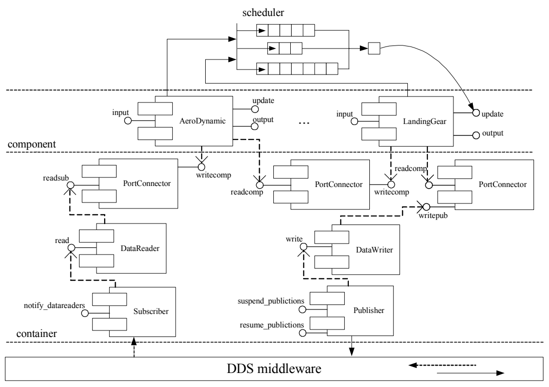


Figure 5. Flight system simulation software architecture

B. Data Models

According to the software architecture for the flight system simulation, we should at first model the publishing/subscribing data among simulation components. Data models are described by OMG IDL to achieve the good platform independence. For example, aerodynamic component in the flight system needs to input rudder angle, aileron angle, elevator angle and angle of stabilizer settings from control loading system, so the designed Aero_Con topic is as follows:

```
module FlightSys{
typedef double Aileron_Angle[2];
typedef double Elevator_Angle[2];
struct Aero_Con{
        long nodeID;
double dRudder_Angle;
        Aileron_Angle dAileron_Angle;
        Elevator_Angle dElevator_Angle;
        double dStabilizer_Angle;
};
#pragma keylist AeroRudAngl nodeID
};
```

C. Implementation of Real-time Communication among Components

After finished the definition of system's data model, the control loading system follows the next five steps to publish Aero_Con topic, the sequence diagram for topic publication is illustrated by Fig. 6:

1) Register data types to the DDS middle:

FltSys_Aero_ConTypeSupport_register_type(Domain-Participant domain, string type_name);

2) Create publisher:

DomainParticipant_create_publisher(PublisherQos qos, PublisherListener a_listener);

3) Create topic:

DomainParticipant_create_topic(string topic_name, string type_name, TopicQos qos, TopicListener a_listener);

4) Create DataWriter:

Publisher_create_datawriter(Topic a_topic, DataWriterQos qos, DataWriterListener a_listener);

5) Write data samples to the topic:

DataWriter_write(Data instance_data);

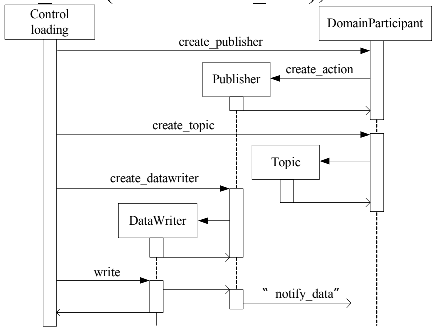


Figure 6. Sequence diagram for topic publication

Accordingly, the aerodynamic component follows the next three steps to subscribe Aero_Con topic, the sequence diagram for topic subscription is illustrated by Fig. 7:

1) Create subscriber:

DomainParticipant_create_subscriber(SubscriberQos qos, SubscriberListener a_listener);

2) Create DataReader:

Subscriber_create_datareader(TopicDescription a_topic, DataReaderQos qos, DataReaderListener a_listener);

3) Read data samples from the topic:

DataReader_take(DataSeq data_values);

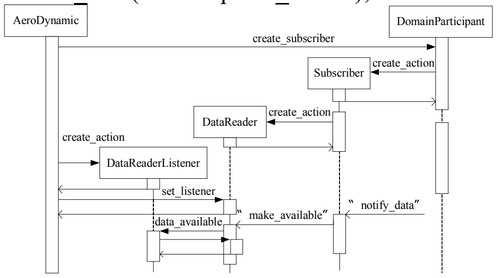


Figure 7. Sequence diagram for topic subscription

V. Experiments and Performance Verification

The communication latency from control loading system to flight system is measured by sending a timestamp together with the data flow and calculating the difference in time between this time stamp and the time the topic sample is received. We choosed thirteen topics which are subscribed from the control loading system as test objects and employed OpenSplice Ver 3.4 [7] as the DDS middle. The comparing communication latency test result with HLA middle-DMSO RTI1.3 NG [8] is showed in Fig. 8. It can figure out from the result that the DDS based communication latency between above-mentioned sub-systems is stabilized around 100µs which is much lower than HLA.

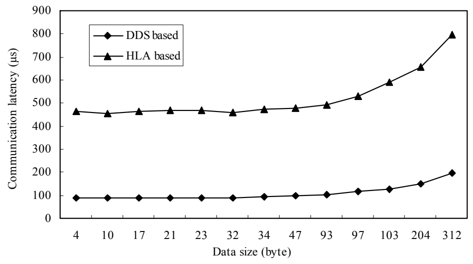


Figure 8. Communication latency between control loading system and flight system

Fig. 9 is a typical simulation about a normal taking off under some special conditions. The input commands are generated by a qualification test program which complies with Airplane Simulator Qualification [9].

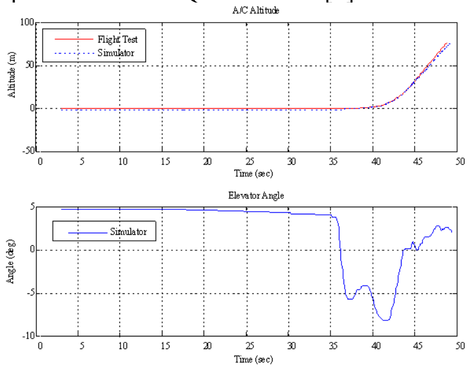


Figure 9. Elevator angle vs. A/C altitude during taking off

This figure describes a pilot pulled the wheel at 35th second to make the elevator to a negative degree, and then the pitch angle increasing, the aircraft began climbing. In this process, the simulation conformed to the flight test strictly. It indicated that the states of the A/C are in the tolerance defined by the Airplane Simulator Qualification. Other simulations such as cruise, landing and so on all had the same good performance and all conformed to the flight tests. Hence, the flight simulator had enough fidelity as a training device.

VI. Conclusion

In this paper, the real-time distributed computing environment and reusable software architecture for the flight simulator based on DDS are presented. The communication latency and typical flight's taking off simulation results under some special conditions are testified to prove the design methods improve the fidelity of flight simulator. The real-time distributed computing environment design based on DDS decreases the communication latency between flight simulator's sub-systems and speeds up the construction of flight simulator. And the reusable simulation architecture is intended to ease the modification of the simulator as better models become available or additional elements are included in the future, such as adding various wind models, more malfunction models and logic functional models. Meanwhile, the flexibility of this simulation architecture may afford help for flight dynamic and control analysis task. The methods simultaneously provide an efficient construction route for other real-time distributed simulation systems.

References

[1] Morris, A. Terry, "COTS Score: An Acceptance Methodology for COTS Software", Proc. Digital Avionics Systems Conf, IEEE Press, 2000, pp. 4B2/1-4B2/8 vol.1, doi:10.1109/DASC.2000.886948.

[2] Simulation Interoperability Standards Committee (SISC) of the IEEE Computer Society, IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-IEEE Std 1516-2000,1516.1-2000,1516.2-2000, Institute of Electrical and Electronics Engineers, Inc., New York, 2000.

[3] A.J.J. Lemmers, P.J. Kuiper and F.R. Verhage, "Performance of a Component-based Flight Simulator Architecture Using the HLA Paradigm", AIAA Modeling and Simulation Technologies Conference and Exhibit, Monterey, California, USA, AIAA 2002-4476, 2002.

[4] OMG, "Data Distribution Service for Real-time Systems Specification", OMG, www.omg.org/docs/formal/07-01-01, 2007.

[5] Gerardo Pardo-Castellote, "OMG Data Distribution Service: Architectural Overview", Proc. Distributed Computing Systems Workshops, IEEE Press, 2003, pp. 200-206, doi: 10.1109/ICDCSW.2003.1203555.

[6] Wang Xinren, Jia Rongzhen, Peng Xiaoyuan, and Feng Qin, Flight Real-time Simulation System and Technology, Press of Beihang University, Beijing, 2003.

[7] PrismTech Limited., DDS Version 3.4 C++ Reference Guide, PrismTech Solutions Americas, Inc., MA, USA, 2008.

[8] SAIC, RTI-NG 1.3 V3.2 Release Notes, Science Applications International Corp, 2000.

[9] Federal Aviation Authority, Airplane Simulator Qualification, Advisory Circular (AC) 120-40C, 1995.