DE GRUYTER OPEN

BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **15**, No 5
Special Issue on Control in Transportation Systems

Sofia • 2015

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2015-0022

An Object-Oriented HLA Simulation Study

Georgi Kirov

Institute of Systems Engineering and Robotics, 1113 Sofia
Emails: kirov@iser.bas.bg

Abstract: *The study is dedicated to High Level Architecture* (*HLA*) *standard for software architecture of interoperable distributed simulations. The paper discusses the differences between object-oriented programming and HLA. It presents an extended simulation architecture providing a mechanism for HLA data exchange through Object-Oriented* (*OO*) *objects. This eliminates the complex network programming for HLA distributed simulations. The paper shows a sample code that implements the architecture for OO HLA/RTI simulation.*

Keywords: *High-level architecture, distributed simulation, data-centric organization.*

1. Introduction

Distributed simulation technologies are a paradigm to model dynamic, heterogeneous and spatial distributed systems. They not only aim at speeding up simulations, but also serve as strategic technologies for linking simulation components of various types [1]. Distributed technologies can run with different components installed on different computers linked via a local network, so as to accelerate the execution time of the simulation. Although the contemporary distributed simulation technologies, and especially, High Level Architecture/Run Time Infrastructure (HLA/RTI) standard has a standardized structure for object models, they do not completely correspond to common definitions of object models in Object-Oriented (OO) analysis and design techniques [2, 9].

A number of simulation models has been developed and more are being developed for studying the individual aspect of system components. The value of these models decreases because they do not consider all aspects of system interdependencies. The simulation models, addressing different aspects of the system components need to be integrated in a common framework to provide the whole picture of the behavior of a complex system, which depends on the interaction of several heterogeneous subsystems. In most of the cases, the simulation models are built based on the OO approach. Object-oriented simulation has great intuitive appeal in applications since it is very easy to view the real world as being composed of objects [3]. Typical for these models is that there does not exist integration among the models. They do not easily address highly dynamic complex systems, and heterogeneous and spatial distributed models.

The acquired experience in distributed simulation technologies has called for carrying out in-depth the analysis as to what extent the simulation mechanism is fitting the contemporary challenges and requirements. The protocol-based nature of the distributed simulation technologies shows the following trends:

- the distributed simulation technologies make it easy to connect applications, but not so easy to find, access and work with the information from object-oriented applications;
- lack of an object-oriented infrastructure of the distributed simulation technologies, providing effortless component integration.

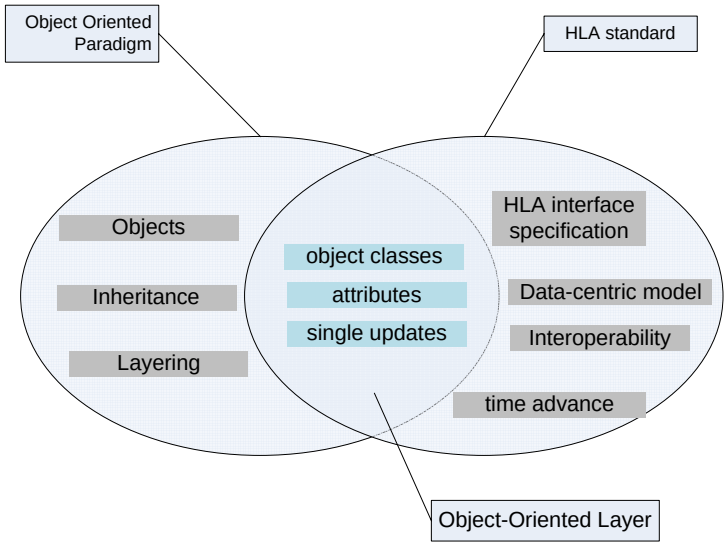


Fig. 1. Mapping of OO paradigm and HLA properties

The study aims at developing an extended software architecture providing a mechanism for HLA data exchange through OO objects (Fig. 1). Manipulating these objects (i.e., creating, modifying, deleting), the applications can work directly at HLA/RTI API. This will eliminate the complex network programming for HLA distributed simulations.

2. HLA/RTI

The problems due to the inflexibility and lack of scalability of the traditional distributed approaches have led to a different approach, the High Level Architecture (HLA), which becomes IEEE 1516 Standard. HLA allows the experts to combine computer simulations into larger simulation. For instance, the experts might want to combine simulations of complex systems in several different regions of the country. HLA can extend the simulation later by adding new models or simulations, for example new models of infrastructures [6]. The HLA defines a set of rules governing how the simulations (applications), now referred to as federates, interact with one another. The federates communicate via a communication environment, called RunTime Infrastructure (RTI) and use an Object Model Template (OMT) which describes the format of the data. The information exchange between federates is based on a common object model, called Federation Object Model (FOM). The HLA federates use data management mechanism based on publishing and subscribing.

Runtime infrastructure is a supporting software that provides an information exchange mechanism between federates in the distributed environment, regarding FOM. It implements a distributed operating system and forms the basic software layer for HLA applications. It neither maintains information about the state of the federates, nor handles any semantics associated with the interaction between federates, like what coordinate systems to use or what happens during a collision. The RTI provides a set of services to the federates for data interchange and synchronization in a coordinated fashion. The RTI services are provided to each federate through its Local RTI Component (LRC) [1, 6].

The *HLA object model* supports the information exchange between federates within the federation. The exchange of information takes the form of *objects and interactions*. The federates communicate with their peers by sending interactions or updating the object attributes. Federates do not communicate directly with each other and all communication is administrated by the RTI. The *Object classes* [6] are comprised of attributes. The Object classes describe types of things that can persist. Each object in a time moment is characterized by a state, which is defined by a set of current values of its attributes. The federate, which manages an object, may alter the state of the object by changing the attribute values. Through RTI services, the federate transmits the new values of the object to all federates in the simulation. In this case it is assumed that the federate updates the attributes. The *Interactions classes* [8] are comprised of parameters. An interaction is a single action caused by a change in the state of an object from another federation. The Interaction classes describe types of events. The objects are similar to interactions in so much as the objects are comprised of attributes, and interactions are comprised of parameters. The basic difference between the objects and interactions is persistence – the objects persist, the interactions do not. In conclusion, the HLA FOM offers an object model that does not completely correspond to the common definitions of object models in object-oriented programming. Most of the OOP functionality can

be mimicked using tailored HLA federation agreements. Most of the HLA functionality can be mimicked using OO classes and methods [7].

3. Extended HLA software architecture

This chapter presents a software architecture that extends the HLA profile with an object-oriented view on a set of related HLA FOM object data, thus providing typical OO-features, such as navigation, inheritance and use of value-types API [4, 5, 7]. The main goal of the architecture is to provide functions and services for working with the traditional HLA (see Fig. 2). Once this is done, we can write the business logic on-top of these abstractions. It can significantly simplify the implementation of HLA interfaces. Our aim is to develop an object-oriented architecture that reduces the lines of the code that need to be written for a HLA application.

3.1. Differences between object-oriented programming and HLA standard

In the HLA terminology, the classes support information for a common description of the objects. The basic idea is to provide data-centric organization, which is an opportunity for information exchange between the distributed applications. This concept differs significantly from the classical object-oriented methodology where the behaviour is an integral part of the objects. In the proposed architecture, attributes of an object-oriented class correspond broadly to the attributes of a HLA object class.

The diversity of HLA and OO object concepts consists in the fact that HLA objects are defined entirely by the attributes, the values of which are exchanged between the federates. The responsibility for updating the attributes of an HLA object is distributed among different federates in a simulation system, whereas OO objects encapsulate the state locally and associate the update responsibilities with methods that are an integrated part of the object's implementation in [2].

In order to add an additional functionality to HLA objects, it is necessary to implement software encapsulations of the data and methods, which contain descriptions about the objects behaviours. The methods affect the values of HLA object attributes. Another difference is that HLA does not support multiple inheritance which does not permit the use of polymorphism. In order to resolve the above mentioned problem, it is necessary to make OO API that provides an ability to request objects with the same interface to react differently depending on the type of the object.

3.2. Extended HLA object model

Fig. 2 shows a high-level model of the proposed architecture. The rtiAmb (rti ambassador) contains customized libraries that access the standard RTI services and it simplifies the design of the simulation model [4, 5]. Another member of the fedAmb architecture provides a common callback mechanism to the programmer, and thus the RTI invokes functions from the OO user methods.

Therefore, there are fundamental differences between object-oriented programming and HLA standard. A number of assumptions about how a federate wants to use HLA services must be made in order to support these services in an object-oriented API.

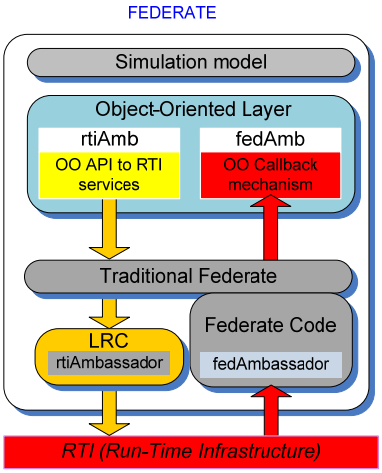


Fig. 2. Conceptual model of HLA software architecture

On the other hand, it is also necessary to make a number of assumptions about HLA interactions between federates in order to fully use object-oriented features, such as method invocations [5, 7]. The proposed OO approach aims at transferring objects between joined federates. It implies shared HLA objects to be presented as local OO objects (C++), i.e., an HLA object instance to be presented as an object-oriented C++ object [7]. For this purpose, the OO objects have to be mapped into FOM data format. Therefore, HLA attribute has to be mapped to C++ class attributes. It imposes the object data to be coded into a network format by a serializing mechanism.

The details of how the object data is encapsulated and propagated are dealt by data serialization (Fig. 3). The concept is based on the idea of representing the object data fields into a network-format for transmission. OO methods for a serialization provide enough capabilities to code the HLA object in the form of a vector of bytes [5]. The publishing federate uses an OO function sendUpDate() to send object data through the RTI. The architecture proposed provides flexible methods to the user for packing update data, and leaves the transmission details transparent. The subscriber federate receives the published data for a given object. Then, the federate calls the function receiveUpDate() of the object distinguished by the key. The function maps HLA FOM object to a class by deserialization, thus making it possible to reconstruct the object-oriented views of the existing data models (mirror objects). Thus, the mirror objects are automatically created at receiving federates as object-oriented instances. A mirror object is a dynamically created block of code that exists in the receiving application on behalf of the OO HLA object, and it exposes all of its features, including all fields, properties, events, and methods.

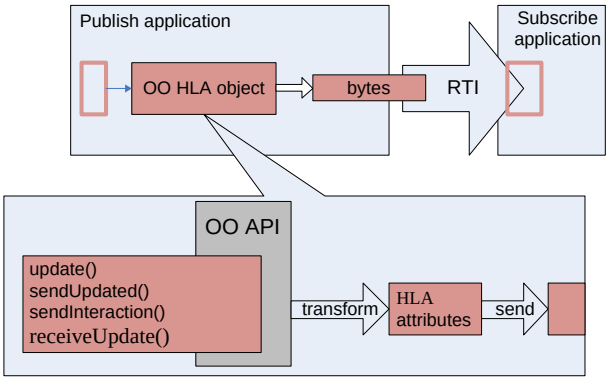


Fig. 3. OO HLA communication model

All work on processing of the OO HLA objects is done by the simulation loop (Fig. 4). It is continuously repeated, during which the virtual functions of all OO HLA objects are executed and the attributes are updated. The simulation loop calls the virtual functions of the implemented HLA models because of the polymorphism obtained by an inheritance StandAlone class.

```
/*****************************************************************************
*           Class SimulationCycle()- simulation cycle
*****************************************************************************/
void HLA_ObjectsList_StandAlone::SimulationCycle()
{
      int i;
      for(i=0;i<list_end;i++)
      {
            //if an HLA object has to be deleted from RTI
            for(i=0;i<list_end;i++)elem[i]->deleteHLAobject();

            //calculate new values for OO HLA objects
            for(i=0;i<list_end;i++)elem[i]->update();

            // send the new values through RTI
            for(i=0;i<list_end;i++)elem[i]->SendUpdate();

            //advance of simulation time
            while(!StandAlone::aaw.advance_time()){};
      }
}
```

Fig. 4. Simulation loop

4. Reference model of an OO HLA object

This section shows a sample code that implements the architecture for OO HLA/RTI simulation. The reference model of HLA object is C++ class, which sets the standard for the construction of HLA objects. It provides the performance requirements for the joint and resign of HLA objects in a common simulation. The reference model implements the principal characteristic of the HLA simulations – modularity and expandability. The remaining part of this point is a principle

description of the reference model of an OO HLA object – the data, methods and working principle. HLA_Model class specifies the structure of the OO HLA object. It inherits the class StandAlone, thereby receiving all features necessary for participation in the HLA simulation (Fig. 5).

HLA_Model class contains two sections that are relevant to the HLA/RTI standard. The first section contains declarations of the state variables. In the terminology of C++ these variables are presented as members-variables (data) to the class HLA_Model. The second section contains virtual functions inherited from the class StandAlone. These virtual functions must be implemented, as they are called sequentially in the simulation loop. The virtual functions meet the following basic steps defined by the HLA standard:

- processing upon receipt of an event (interaction);
- calculation of the state variables for each time step;
- sending the new values for the HLA models through RTI.

```
/*************************************************************************
*     Reference class HLA_Model inherits the basic class StandAlone
*************************************************************************/
class HLA_Model:public StandAlone{
public:
      // State variables, which describes the state of the HLA_Model
      type_specifier    m_member_1;       // state variable
      type_specifier    m_member_2;;      // state variable
      type_specifier    m_member_2;       // state variable

      UINT  m_TimeOfRegistrationRTI;      //time of registration in RTI

      // Virtal methods, which describes the behaviour of the HLA_Model.
      void  update();               //it calculates new values for the objects
      void  sendUpdate();           //it sends the new value through RTI
      void  updateInteraction();    //it receives interaction
      void  receiveUpdate(const RTI::AttributeHandleValuePairSet&
            theAttributes);         //it receivess the nuw values for
                                    //attributes from RTI
      };
```

Fig. 5. C++ HLA_Model class

5. An example for building OO HLA simulation

5.1. Aircraft model

In this section an example for building a simple OO HLA airplane model is presented. It is developed following the specifications given into the reference model of HLA object (HLA_Model). An aircraft class (Fig. 6) is created initially, and then it inherits the base class StandAlone. It encapsulates the common features of all aircraft – speed, position, identification, etc.

In the Aircraft class the virtual functions inherited from StandAlone are not implemented because the simulation application never creates an object of class Aircraft. This class is used only to be inherited by classes that are models of specific aircraft, such as F16 (Fig. 7).

```
/*****************************************************************************
*          Class Aircraft inherits the basic class StandAlone
*****************************************************************************/
class Aircraft:public StandAlone{
public:

// State variables, which describes the state of the HLA_Model
      double      m_speed;          // meters per second
      double      m_PosX;           // position x (longitude)
      double      m_PosY;           // position y (latitude)
      double      m_PosH;           // position h (altitude)
      int         m_SSR_Code;       // identification number
      UINT        m_T0;             // time to take off

      bool        m_status;              //indicates if the plane took off
      UINT        m_TimeOfCreation;      //time of regist. in standAloneList
      UINT        m_TimeOfRegistrationRTI;     // time of regist. in  RTI
      double      m_azimuth;                   //azimut

      //plane track
      vector<POINT3D>   m_Track;

      Aircraft(StandAlone StandAlone1)
            :StandAlone(StandAlone1){}
      };
```

Fig. 6. Aircraft class

```
class F16:public Aircraft
{
public:
      F16(double lon, double lat, double hei, char *name,double speed);
      ~F16();

// Virtal methods, which describes the behaviour of the F16
      // it calculates new values for F16 for the next simulation step
      void  update();

      //get the plane position
      void  getPosition( double & PosX,  double & PosY,  double & PosH);

      //send the new values through RTI
      void  sendUpdate();

      //receive the new values from RTI
      void  receiveUpdate(const RTI::AttributeHandleValuePairSet&
      theAttributes);

      //register a new object into RTI
      void  updateInteraction();
};
```

Fig. 7. F16 class

An instance of HLA object class F16 is created by the constructor of Class F16. It initializes the state variables of the model and carries out registration of the object into the RTI. As a result, an object handle of the objectHandle is returned. It is a unique number that identifies the object instance into RTI. The object instance is global representation maintained by the LRC. The same object instance is known to all federates by its global unique handle value. The state variables are recalculated at each time step of the simulation time by the function UpDate(). In the example, the HLA object F16 recalculates the new position of the airplane. The updated values of the state variables are sent to the RTI environment by SendUpDate() from where all subscribing applications can get them. When an HLA update is received, the corresponding mirror object is updated, enabling the application to receive the value whenever needed [7].

5.2. A conceptual model of distributed simulation

A case study is done on an integrated simulation of an Air Traffic Control (ATC) system. The distributed simulation system is created from a set of models that are interconnected with each other. The proposed simulation system consists of several federates: aircrafts, radars, viewer and analyzing tool. Fig. 8 presents a publish-subscribe communication architecture that supports object-oriented updates to HLA object instances. It shows the interactions between the model components needed to work together to accomplish a communication task. Fig. 9 shows the viewer federate. The viewer is developed to provide an integrated display environment. It can act as a passive recipient and display simulation data from the rest of the simulation system. The viewer uses databases to find geographic coordinates of the static objects (airports, radars, etc.).

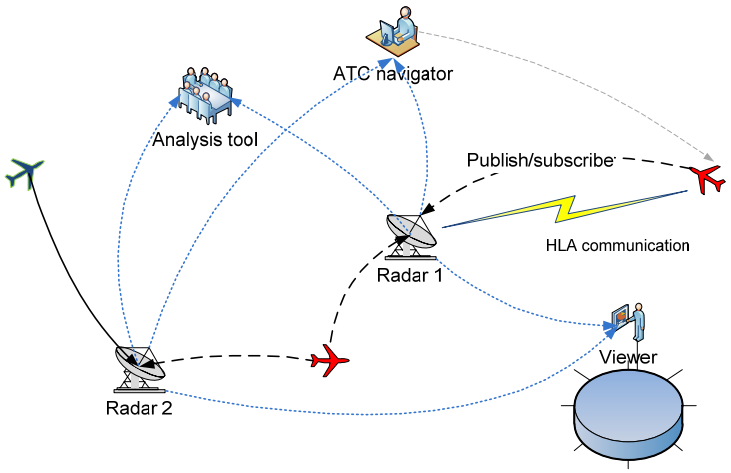


Fig. 8. Publish/Subscribe model

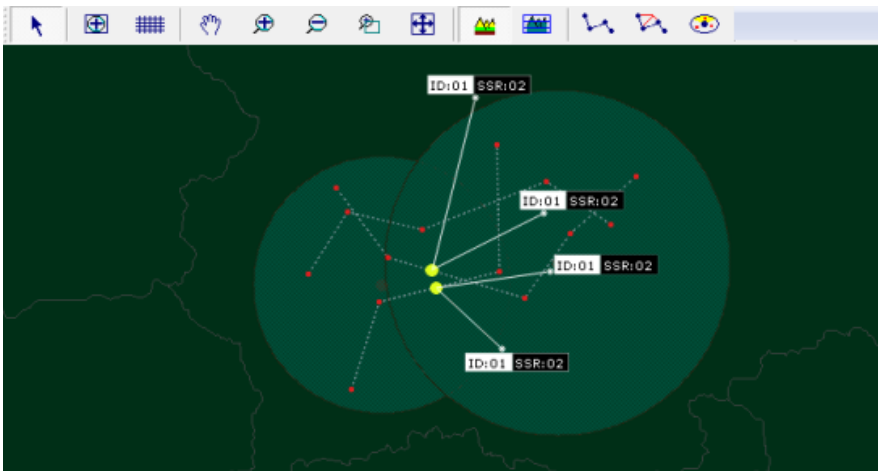


Fig. 9. Viewer federate

6. Conclusion

The study presents an extended software architecture providing a mechanism for HLA data exchange through OO objects. It facilitates the interoperability of all types of models and simulations among them, as well as facilitating the reuse of modelling and simulation components. The benefits of an object-oriented approach to HLA simulation are numerous. The paper shows a sample code that implements the architecture for OO HLA/RTI simulation.

The proposed architecture implements a mechanism for an object transfer that can be summarized in the following steps:

1. Presentation of an HLA object instance as an object-oriented C++ object;
2. Coding the C++ class attributes into a network format (bytes) by a serializing mechanism;
3. Sending the serializing object attributes through the RTI by an OO function;
4. Reconstruction of the object at receiving federation according to the existing data models (mirror objects).

***Acknowledgements*:** This work is partially supported by the project "AComIn: Advanced Computing for Innovation" grant 316087 funded by the European Commission in FP7 Capacity (2012-2016).

References

1. Chena, D., S. J. Turner, W. Cai, M. Xiong. A Decoupled Federate Architecture for High Level Architecture-Based Distributed Simulation. – J. Parallel Distrib. Comput., Vol. **68**, 2008, pp. 1487-1503.
2. IEEE, P 1516. Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules. February 2000.
3. Joines, J. A., S. D. Roberts. An Introduction to Object-Oriented Simulation in C++. – In: Proc. of the 1997 Winter Simulation Conference, S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, Eds., 1997, pp. 78-85.
4. Kirov, G., V. Stoyanov, B. Lazarov. Network-Centric Simulation of Complex Transportation Systems Based on an Extended HLA Integration Architecture. – In: 13th IFAC Symposium on Control in Transportation Systems CTS'2012, TUD COST Action TU1102, 12-14 September 2012, Sofia, Bulgaria, pp. 391-397.
5. Kirov, G., V. Stoyanov. Object-Oriented Architecture for Simulation of Complex Interdependent Systems Based on HLA Standard. – In: 3rd International Conference on Application of Information and Communication Technology and Statistcis in Economy and Education (ICAICTSEE-2013), Sofia, Bulgaria, ISBN 978-954-644-586-5, pp. 279-289.
6. Kuhl, F., R. Weatherly, J. Dahmann. Creating Computer Simulation Systems: An Introduction to the High Level Architecture. Prentice Hall PTR, 1999, ISBN 0130225118.
7. Möller, B., F. Antelius. Object-Oriented HLA – Does One Size Fit All?. – In: Proc. of 2010 Spring Simulation Interoperability Workshop, 10S-SIW-058, Simulation Interoperability Standards Organization, 2010.
8. Run-Time Infrastructure RTI 1.3-Next Generation Programmer's Guide Version 3.2. Department of Defense Modeling and Simulation Office, RTI 1.3NG-V3.2, 7 September 2000.
9. Tolk, A. HLA-OMT Versus Traditional Data and Object Modeling. – In: Command and Control Research and Technology Symposium, Annapolis, Maryland, June, 2001.