HIGH LEVEL ARCHITECTURE VIRTUAL ASSISTANT FRAMEWORK

Josef Brožek [a], Bhakti Stephan Onggo [b], Antonín Kavička [c]

[a] Faculty of Electrical Engineering and Informatics, University of Pardubice, Czech republic
[b] Lancaster University Management School, University of Lancaster, United Kingdom
[c] Faculty of Electrical Engineering and Informatics, University of Pardubice, Czech republic

[a] mail@jobro.cz, [b] s.onggo@lancaster.ac.uk, [c] antonin.kavicka@upce.cz

ABSTRACT

High Level Architecture provides a standard for abstraction, design, construction, development and operation of distributed computer simulation systems. This paper focuses on the latest version of the standard, i.e. IEEE1516:2010. Users who are not familiar with distributed computer programming may find it difficult to create HLA-compliant models using a tool that assumes some familiarities with distributed computer programming. Hence, a tool that can help such users is useful, especially in encouraging more people to develop HLA-compliant models.

We have developed HLAVA Framework that encapsulates the detailed steps of main HLA interfaces such as the methods for creating logical processes, and management thereof, synchronization methods, and communication protocols - in fact, the framework is a simulation kernel for the distributed simulation logical process, which is compliant with (among others) the HLA standard, but due to a simple interface has only about 10 methods, which have done creation of (potentially) much simpler.

Keywords: HLA, High Level Architecture, Framework, simplification interface, distributed simulation, traffic simulation

1. MOTIVATION

High Level Architecture (HLA) is a standard for the creation and operation of distributed computer simulation systems. The standard does not limit the application domain and simulation modelling method. Hence, HLA has been applied to domains such as military, healthcare, supply chain and many more. HLA has also been used to link models developed using various simulation modelling methods such as discrete-event and agent-based. At a lower level, the standard specifies the method of communication between the nodes (or federates) of a distributed simulation. The standard sets the requirements for the actual transmission format using XML. This enables us to link simulation models written using different programming languages or simulation software. It also enables us to write a wrapper for legacy simulation models (non HLA compliant) by converting the input-output data into an XML format.

The openness, flexibility and many functionalities of the standard may overwhelm novice users or users who are not familiar with distributed computer programming. The main contribution of this paper is to propose a framework that helps users to develop HLA-compliant models more easily so that they can use the full potential of HLA with minimal effort (at the cost of moderate restrictions which will be discussed later). This research is partly motivated by the lack of adoption of HLA in the Czech Republic. We hope this work may attract more people to develop HLA-compliant models.

The remainder of this paper is organised as follows.

2. CHARACTERISTICS OF HLA

This section provides an overview of HLA. The explanation is based on Fujimoto (2000), Kuhl, Dahmann, Weatherly (2000) and standards IEEE1516:2010 (2010). Readers who are familiar with HLA may skip this section.

2.1. History of HLA

HLA was originally created for the military needs of the United States of America Department of Defense (more specifically, it was developed at the Defense Modeling and Simulation Office), where there was a motivation for linking various simulation models into a bigger and more complex simulator. HLA, which was handed over to the army in 1995, met all the military requirements. It enabled the interconnection between different models implemented in different programming languages. It also enabled the simulation to be used in a human-in-the-loop training exercises using the simulation models.

In 1997, the Army released the standard for public use; HLA 1.0 standard was thus issued, which began to be used in universities and space research. Gradually, the standard began to be applied in the private sector too. In 1998, HLA was included in the NATO military simulation standard (STANAG 4603 standard, which has since been updated). The standard was approved in 2000 (IEEE 1516). Another important milestone was a major revision that was already based on real experience, not only in the military but also in the private sector and the university sector, taking place in 2010. HLA-Evolved (IEEE 1516:2010) was thus created. Example of solution on Roberto, Sala-Diakanda, Pastrana and col (2013).

2.2. Basic Terminology

2.2.1. Federation

A federation in HLA refers to an entire distributed simulation (or a complex distributed simulation model). There is one federation during one distributed simulation experiment.

2.2.2. Federate

A federate is an HLA-compliant application that can participate in a distributed simulation experiment. A federate may be in a form of a simulation model, software application, software agent of any type, an input sensor or panel, a display unit, a system for retrieving historical data, etc.

2.2.3. Objects

An object is the information to be exchanged during a distributed simulation experiment. In HLA, an object represents an entity with persistent states.

2.2.4. Attributes

An object has a set of attributes that are relevant to an entity represented by the object. An attribute is basically a data field of a defined data type. Basic data types and attributes are defined in the Object Model Template (OMT). We will explain OMT in Section 2.2.6.

2.2.5. Interactions

An interaction represents an event that may be of interest to two or more federates. Similar to objects, interactions must be pre-defined. Each interaction can have a set of parameters. Interaction can be performed directly through interaction classes, e.g. passing an object to another federate etc.

Federates cannot send interactions to each other directly, but they must make a request to RTI (runtime infrastructure) as shown in Figure 1. We will explain RTI in Section 2.2.7.

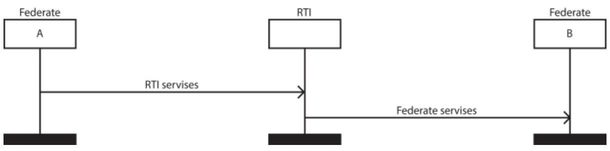

Figure 1: Interactions in HLA

2.2.6. Object Model Template (OMT)

Objects and their attributes as well as interactions and their parameters need to be defined for the whole federation based on the standard set in the OMT (IEEE1516.2:2010). The definitions for Objects and Interactions must be provided in a Federation Object Model (FOM) or Simulation Object Model (SOM). Both are XML documents and are relatively easy to read. In practice, FOM is often built from one or more SOMs.

2.2.7. RTI

RTI (run-time infrastructure) is a middleware that provides communication services to all federates in a federation. It is defined directly by the standard (IEEE1516.3:2010), independent of the platform and language. RTI provides APIs that can be called to perform certain functions such as creating a federation, creating a federate, connecting a federate to the federation, etc. RTI needs FOM to understand the objects and interactions that will be exchanged between federates. Common terminology and the logic of RTI are shown in Figure 2.

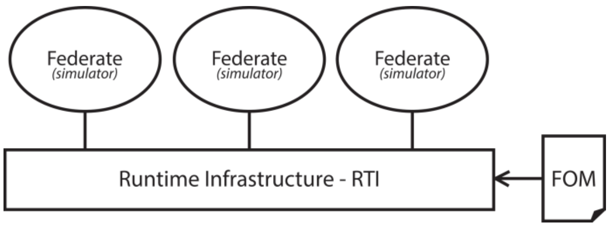

Figure 2: HLA RTI Diagram

A number of vendors have implemented RTI products, for example Pitch RTI, MAK RTI, OpenHLA and many more. In principle, the implementation approach is almost the same as the one shown in Figure 3. A federate usually runs on one computing node (but one node can be shared by multiple federates). For each federate, we need to include the local RTI component. It is a library that needs to be linked to the actual software solution (federates) or to the application wrapper (for legacy or non-compliant software). This component actively manages communication between a federate and a central RTI component. The central RTI component is the busiest part in a federation[1]. It is responsible for managing the communication within a federation, for example, connecting a federate to a federation, managing objects, etc.

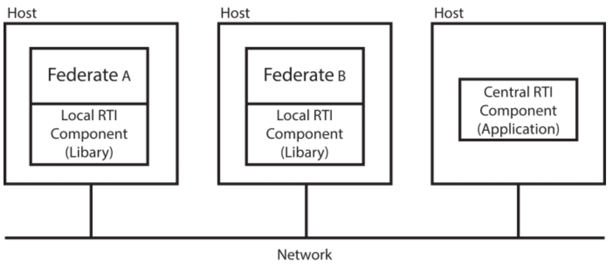
Host
Federate A
Local RTI Component (Libary)
Host
Federate B
Local RTI Component (Libary)
Host
Central RTI Component (Application)
Network

Figure 3: RTI physical implementation

[1] However, it is necessary to note that it is possible to create other than simply linear non-hierarchical simulation models, which makes it possible to use a number of central RTI components in a single simulation.

2.3. Federation Rules

HLA defines 5 rules for federations and 5 rules for federates. These rules must be met even in a situation where we want to declare a model to be partially HLA compliant. In other words, compliance with all ten rules is a necessary condition for us to refer to a simulation model as HLA compliant.

The rules for federations are as follows:

- Federations shall have an HLA federation object model (FOM), documented in accordance with the HLA object model template (OMT).
- All representation of objects in the FOM shall be in the federates, not in the run-time infrastructure (RTI)[2].
- All exchange of FOM data among federates shall occur via the RTI[3].
- Federates shall interact with the run-time infrastructure (RTI) in accordance with the HLA interface specification.
- An attribute of an instance of an object shall be owned by only one federate at any given time. Only this federate is authorized to modify the values of the owned element.

The rules for federates are as follows:

- Federates shall have an HLA simulation object model (SOM), documented in accordance with the HLA object model template (OMT).
- Federates shall be able to update and reflect any attributes of objects in their SOM and send and receive SOM object interactions externally, as specified in their SOM.
- Federates shall be able to transfer and/or accept ownership of an attribute dynamically during a federation execution, as specified in their SOM.
- Federates shall be able to vary the conditions under which they provide updates of attributes of objects, as specified in their SOM.
- Federates shall be able to manage local time in coordination with the RTI requirements (in effect, in coordination with other federates).

[2] The consequence of this rule is the fact that the transfer of the object from one federate to another is performed so that federate1 requests RTI for placing its object in federate2. RTI locks the object in federate1 before the changes and requests federate2 for placing the new object (while passing the object itself). If the location is in federate2, the object is removed from federate1. If not, the object is unlocked in federate1.

[3] The consequence of this rule is that RTI can be equipped with a software upgrade that allows you to track all interactions and potentially current states of most objects.

2.4. Synchronization Methods

Synchronization is provided through RTI, it is therefore not necessary to create additional mechanisms that are normally used in the architecture for distributed simulation models.

The actual synchronization and its algorithms are transparent to the user, we can simply use the services provided. If we want a federate to be synchronized with other federate, we can set which federate is time-regulating and which federate is time-constrained. Alternatively, we can run all federates in a synchronous mode.

2.5. HLA Software Solution

Therefore, to easily understand the benefits of an own framework, it is necessary to specify how the application issue is dealt with.

2.5.1. General Principle

We can design a federate by creating a class that implements (or inherits) one of the federate type classes (for example, NullFederate). This will automatically give the federate access to a number of RTI services such as creating a federation or joining a federation.

Figure 4 shows the details on what happen if a class federate is instantiated during a distributed simulation experiment. First, a pair RTI Ambassador – Federate ambassador will be instantiated for each federate instance. These two components manage the communication between a federate and an RTI. RTI Ambassador is used by the federate to communicate with RTI. The Federate Ambassador is used by the RTI to call any of the callback methods defined in the federate.

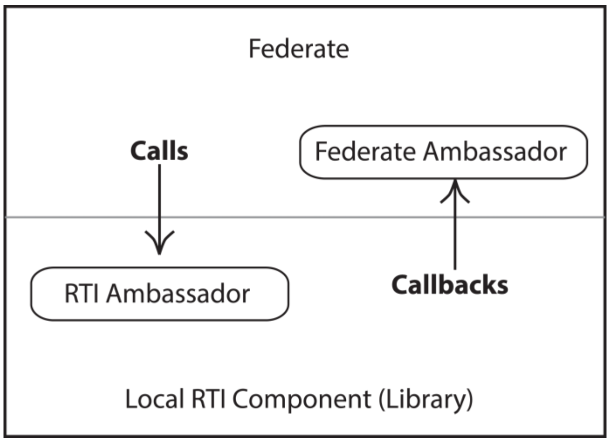

Figure 4: Functioning of local RTI

2.5.2. Federate Ambassador

Federate Ambassador is located in same place as the instance of a running federate. It is generated by default directly when the federate is instantiated. Federate Ambassador allows RTI to invoke a callback method defined in the federate. Since a federate will have access to FOM, it allows the federate to implement appropriate behaviour or to appropriately respond to any callback.

2.5.3. RTI Ambassador

RTI Ambassador is the implementation of a local RTI component discussed previously in Figure 3. During a distributed simulation experiment, each RTI Ambassador is physically located in the same compute node as the corresponding federate. We can use RTI Ambassador to create a federation or to join a federation. It also provides services to allow a federate to communicate with the RTI (i.e., the global RTI component). HLA provides standard for interfaces that have to be implemented in RTI Ambassador and Federate Ambassador which is independent of any RTI software implementation. This standard ensures platform independence[4].

2.5.4. Global RTI Component

The second essential part of RTI that provides communication between RTI ambassadors is the global RTI component. In other words, RTI ambassadors do not communicate directly but via a global RTI component (i.e. the principle described in Section 2.5.3 is valid for some implementations of RTI that do not contain a global RTI component). The global RTI component provides forwarding to all federates that have requested specific information such as an attribute or an interaction. In addition to this forwarding function, this component also provides synchronization, monitoring, administration of the federation and many other supporting functions.

2.6. Overview of Existing RTI software

This section provides an introduction to a number of existing RTI software. Generally, it is possible to characterize various implantations of RTI by specifying whether it is an open or a closed source product; commercial or freely available product; and then according to the HLA standard that the RTI complies with.

2.6.1. Pitch RTI

The commercial product of the Swedish company Pitch Technologies (belonging to the BAE Systems Group) is one of the most powerful variants of RTI, meeting all the HLA standards and is developed with a lot of utilities, more at Pitch websites (2014).

It is very easy to use. It is supplied with libraries for Java and C++ languages (also applicable in C#). To use it, we just need to import a relevant library and then create a sequence of desired objects and serve a quantity of functions.

[4] If both federates have knowledge of OMT, they are able to exchange information with each other through XML. This information can then be forwarded to a federate and RTI (it is worth noting that during the decomposition of the communication process RTI is treated as an active software component, not only as a communication bus - even though it may initially appear so).

Pitch RTI also provides a trial version in which only two federates can run. There may be other limitations that we are not aware of. Currently, this product is used at the University of Pardubice; the proposed software library HLA-VA is designed for Pitch RTI (although the job to write it for a different RTI is trivial).

2.6.2. MÄK RTI

Similar to Pitch RTI, MÄK RTI (more information at MAK websites (2014)) is one of the most widely used commercial RTIs. This software also comes with a trial version in which the maximum number of federates that can run is two. It is a very good and very powerful product that meets the latest standards that is commonly used in the private sector and universities. It supports programs written in Java and C++.

2.6.3. Open HLA

Open HLA is an open source software distributed under the Apache license more at Geeknet (2005). It is a relatively good product for basic experimentation with HLA.

At the time of writing, the development has stopped (i.e. there are no new updates). The latest version does not support all the functionalities that are required by the standard IEEE1516:2010 (e.g. models with optimistic synchronization methods cannot run in Open HLA).

Open HLA launching and compiling scripts are written with the support of the library Ant. It is therefore necessary to have this library installed in the computer (this fact also makes it different from other software where there is an RTI executable .exe file).

2.6.4. CERTI

CERTI is an open source software which is distributed under the GPL (or LGPL) license more at Free software foundation web(2002). Its functionalities are better than that of Open HLA; It fullu supports the HLA 1.3 standard for Java and C++. However, the support for IEEE 1516:2000 is only partially supported and only supports C++.

2.6.5. Other Products

There is a large amount of RTI implementations including products of the U.S. Army and the Chinese People's Liberation Army. Reportedly, RTIs, developed by the Chinese public sector are very good but due to language barriers the author was not able to get acquainted with them. A larger, though not entirely up-to-date, overview of individual RTIs can be found in older resources such as Knight, Corder, Liedel (2001) and report from The simulation interoperability standards organization (2001). A more up-to-date review can be found at Wikipedia (2014), as this resource is updated relatively more regularly to provide the most comprehensive overview of the existing solutions.

3. HLA Virtual Assistant (HLA-VA) Framework

The main motivation for the development of the "virtual assistant" framework is to facilitate easy access to HLA. In this section, we will explain the design and components of the HLA-VA framework, its benefits and limitations.

3.1. Framework Architecture

The framework is built over the libraries that are supplied as a standard to RTI (different RTIs from different vendors have different control libraries). The block diagram of the architecture is shown in Figure 5. The benefit of the design of this framework is that it is possible to replace the relevant control library (the bottom layer in Figure 5) without the need to change the upper layers. The explanation for each layer is as follows.

3.1.1. Isolation of the RTI Library

The actual RTI library is isolated from the rest of the code in the framework via an interface. Parts of the RTI component that are not supplied as a standard in the RTI library are implemented separately. In theory, this design allows us to use an RTI from any provider[5].

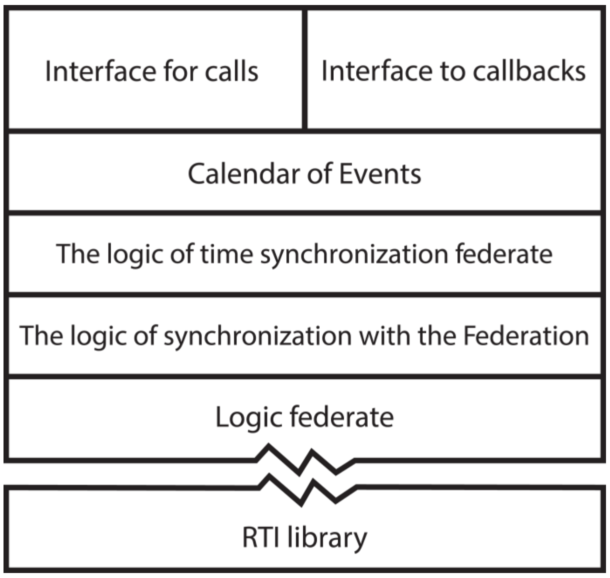
Interface for calls
Interface to callbacks
Calendar of Events
The logic of time synchronization federate
The logic of synchronization with the Federation
Logic federate
RTI library

Figure 5: Block diagram of the framework

3.1.2. Logic of the Federate

This layer deals with the communication between the global RTI component and the local federate. One of the main role of this layer is to make the definition of the information exchanged during distributed simulation execution easier.

[5] We have tested our framework with MAK RTI, open HLA and Pitch RTI. Hence, we are confident that our framework can be used with any compliant RTI solution.

When we write an HLA-compliant model, we often need to deal dynamics arguments[6]. Each message (Objects or Interactions) in HLA is strongly typed and controlled by RTI (as specified in FOM or SOM). If we would like a federate to react upon an incoming message, we need to write a callback method.

The problem is solved by a framework at the program level. Immediately after running a local instance of the framework, it passes through FOM and SOM and selects the interaction classes and types. Consequently, the framework stores them in the hash table.

When a message from RTI arrives, the framework compares its hash fingerprint with those of all message types in the Federation (stored in the hash table) and finds the relevant type. When we identify the type of message, we can start reading - that is, determining the number and type of arguments, their translation from the stream, and select individual data components.

As the number of the existing message types compared to the number of individual sendings is negligible, this method is computationally efficient (the approximated complexity of the search is smaller than the subsequent translation of the data from the stream).

Since we have to assume that the individual Federates can be implemented in different programming languages, it is impossible to rely on programming techniques, such as Reflection, and problems need to be solved by implementation as such.

Hence, we can edit the FOM/SOM and send / receive messages with dynamic arguments without the need to change the code for our federate.

3.1.3. Logic of Synchronization with the Federation

This layer and layer indicated in section 3.1.4 are responsible for time synchronization. Central component of cooperation between both the time synchronizations and also the part available to the user is called Calendar of events (see section 3.1.5.).

Specifically, Logic of Synchronization with the Federation ensures proper functioning of the time synchronization federate (on which it runs) with the Federation. It encapsulate self-programmed and also standard methods specified by HLA enabling the framework to fully control the synchronization (using

[6] Standard communications (or more precisely in the HLA language "Interaction Objects") which takes place within HLA, is defined in the XML format, wherein it differs for different types of communication primarily in the number and types of arguments (however, it is not impossible that messages will have the same types and the number of arguments). However, programming languages often have a problem with coding sufficiently general programs so that we could control incoming communications regardless of the type and number of arguments they have – at the same time, a successful control requirement is also the decoding of messages by type and passing thereof on. This problem is possible to partially solve by using of Lambda expressions, but they are not available in a lot of programing languages.

only the knowledge of the desired synchronization procedure.). This layer passes all the messages (incoming and outgoing) marked as timed and solves the outwards federate requirements for time changes. It is the responsibility of the layer to respond to the changes in GST to be enacted by RTI.

When conservative synchronization is used, each federation makes a request for the local virtual time and after all federates are ready, RTI orders to go to the next lowest required LVT (this is how the easiest global synchronization solved by the methodology with zero look ahead works – more about conservative synchronization with zero look ahead in Fujimoto (2000)). It should be noted that, as standard, the information about the transition to LVT is sent only to those federates that have requested it. The logic of time synchronization of federations is thus responsible not only for forwarding the requests for shifting the time and information about the federate being ready to shift the time, but it also monitors time changes taking place in other federates. In addition, if the mode with look ahead is activated, it provides automatic adjustment of look ahead so that it is as advantageous as possible for the federate and the federation.

3.1.4. Logic of Time Synchronization of the Federate

This layer is responsible for setting up the mode of which a federate runs, for example, real time run, run without time synchronization (e.g., for modelling and use of Monte Carlo simulation), standard mode (i.e., maximum speed run) or speed limits (e.g. for visualization).

This layer is responsible for setting up the mode in which a federate runs. For example, the real time run, run synchronization free run (e.g. for modelling and use of Monte Carlo method), standard mode (i.e. maximum speed run) or speed limits (e.g. for visualization).

The layer consists of two major parts: the application logic running in a standard fibre framework and the time base running in its thread.

The time base may not be used in all the simulation types. However, for some runs (e.g. the real time) it is required because it is the only part that cannot be influenced computationally and the time is the most accurate.

The application part of the layer provides for the synchronization of the local level - monitoring requirements from the user (as inserted into the Calendar of events, and if needed, passes the requirements for the Synchronization Layer with the Federation) serving the requirements of the Federation (by entering them into the Calendar of events) and provides the synchronization with independent time base.

3.1.5. Calendar of Events

It is a standard calendar of events represented by the priority queue that is primarily solved by secured transactions to avoid the risk of inserting and removing across the threads. It is the same as in a monolithic simulator.

Due to the similarity with a monolithic simulator and because there exists only a set of basic user operations such as "Add Event", "Take event" and "Cancel the scheduled event" that makes the entire interface framework very simple.

The behaviour of the other two layers that provide for the synchronization (see above) is transparent to the user. Despite the simple interface, it is possible to reach the synchronization with any time-base derived from real time and with the Federation.

3.1.6. Interface for Calls

It creates a space for calling basic methods necessary for simulation. These include insertion of events into the calendar of events, change of registration in the calendar of events, sending an object or a message to another federate.

Messages are always defined generically to make it possible to insert any type of object into an argument. This object is then the return type of callback.

3.1.7. Interface for Callbacks

It contains Callbacks that must be removed. By default, all Callbacks are implemented the design pattern Observer, it is thus possible to connect to them any number of classes that will process the data.

Callbacks can be viewed as events. The only difference is in the way that leads to evocation. Callbacks are always dependent on the call and they do not occur unbidden.

Basic Callbacks include time change call, incoming message call, incoming object call and information call.

3.2. Services Provided

Basic services that are provided by the framework can thus be summarized as services of the simulation kernel, which includes the time calendar and is able to synchronize the local simulation by the event scanning method.

In addition, support is included for synchronization within distributed simulations. It is possible to forward messages, synchronize the local virtual time with other federates as the global virtual time of the whole federation.

The first major consequence of the creation of the simulation kernel is facilitation of the creation of simulation models for developers who would opt to use this kernel.

Moreover, it is possible to use only a few other methods, and the whole principle of the framework allows developers to create more federates that will communicate with each other and synchronize their time. Therefore, it is possible to create distributed simulation models. At the same time, all models created comply with the HLA standard. The consequence of this fact is that it is possible to connect other simulators implemented without the use of the HLA-VA technology.

3.3. Comparison of Demands of the Development

When comparing individual development methodologies, we assume a distributed simulation model with several federates, while creation of the application is assumed from the very beginning.

3.3.1. Development for HLA without the Framework

Even if we use the absolutely simplest applications developed to comply with the HLA recommendations, namely to operate on the simplest RTI, we need to perform the following steps:

1. Create OMT using a specialized tool, or using a text editor. It is created in XML.
2. Connect all the necessary libraries to the program.
3. Create objects that will implement all methods of the general definition Federate (or NullFederate).
4. Create methods for connecting to the federation, identification of federates and other utility methods.
5. Create methods for operating each individual interaction.
6. Create synchronization methods for the actual simulation logic. While it is indeed possible to use synchronization mechanisms provided by RTI between individual federates, it is still necessary to create a mechanism that will ensure the synchronization within the federate. (The advantage of this step is again the potential reuse of the solution, because if the internal synchronization service is implemented sufficiently generally, it can be used also in other solutions).Set rules for synchronization and solution of the actual synchronization within a particular node.
7. Program the application of the specific model (or logical process).

3.3.2. Development with the Framework

Since the framework has an implemented simulation kernel, there is no need to address synchronization, and it is designed to work as an intermediate layer between the application logic and the federate, so the development process changes to:

1. Create OMT specifications through the XML editor or specialized software.
2. Connect the HLA-VA library to the application.
3. Create the logic of own simulation model.

It is very important to find that all the steps that require knowledge of programming associated with RTI have thus been omitted. At the present moment, it is only the first step that requires some knowledge of HLA (nevertheless, this can be circumvented by using a specialized editor). Still, the development process (especially for a layman) becomes much easier to grasp and implement.

It is this facilitating of the access to HLA that is the greatest advantage of HLA-VA Framework, as it is possible to create HLA-compliant models more quickly. Furthermore, because the RTI library is only connected to framework code in the form of a dynamic link (instead of using static compilation), it is possible to use the framework with (virtually) any RTIs.

3.4. Limitations of the Framework

3.4.1. Limitations on the Choice of Synchronization Methods

Currently, from the user's perspective, we can choose from several synchronization methods in the framework. However, physical implementation is limited to the use of RTI services in the field of conservative synchronization, which is sufficient for the class of tasks that the software is focused on, but it is still a certain limitation for the programmer.

3.4.2. Limitations on the Connection to Federates Implemented without the Framework

By default, HLA assumes that any two federates will be able to cooperate provided that they have a relevant operating method implemented. However, as is clear from the preceding paragraph, if we have an existing federate that contains methods for optimistic synchronization, it is not possible to create a federate through the framework that could cooperate.
In other cases, federates implemented through the framework and without it can cooperate.

3.4.3. Limitations on the Choice of Programming Languages

Since the actual Framework was implemented only in the Java programming language, it enables writing more software components only in Java, C# and C++.
Java support is implicit, and it is possible to link some parts of the framework as libraries.
Support for C# and C++ has been tested in the Windows environment where it was necessary to use Wrapper for the possibility of using the Java code in other programming languages, which can have negative effects on the computational complexity.[7]

[7] Experiments confirmed the extension of the computational time if we compare an identical task implemented solely in one programming language and the same solution implemented using Wrapper and combination of Java and C#. The computational complexity, however, did not increase dramatically enough to make it possible not to recommend such a solution. In the demonstration tests, the solution through Wrapper was useful even if we ran a distributed interactive simulation with real-time synchronization.

4. Prototype

In the early stages of the development of the framework, basic functionalities were tested through direct calls. Using these methodologies, validation of the framework was performed, first in the local environment (more federates running on one local node) and subsequently also in a distributed way within the local network. After basic validation tests were completed, it was decided to build several prototypes.

4.1. Prototype: Motorway Rest Areas

The first prototype designed to test synchronization, connection to the visualization components and testing of the development of the framework in various programming languages (Java, C #) was a simple example of motorway rest areas. The demonstration example contains three federates of two kinds. The federate of motorway is used to generate input streams and its logic determines in which direction the vehicles are going and what percentage of them is heading to an exit with a rest area; the physical situation is shown in Figure No. 6.

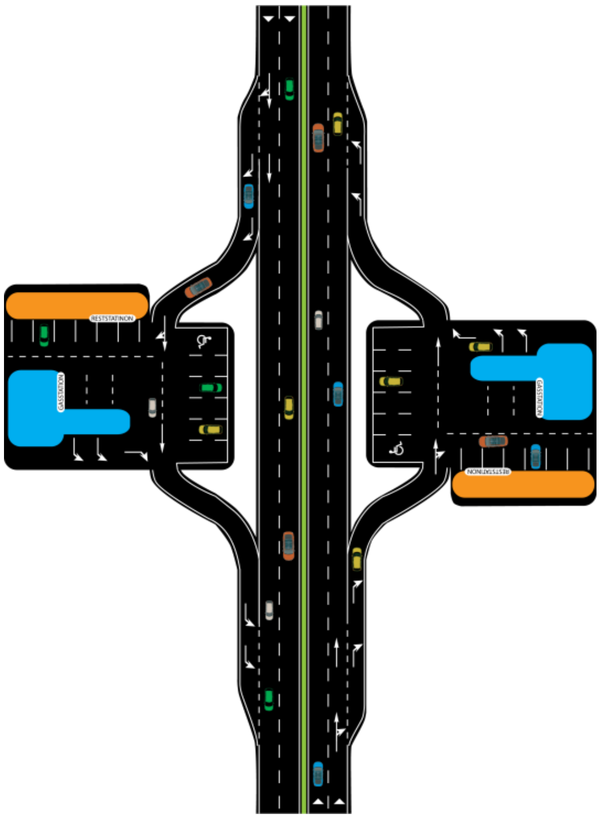
Figure 6: Physical situation

The second type of the federate is a rest area, its internal logic contains only an input from the federate of the highway (similarly, the output is only to the federates of the motorway, it cannot disturb objects by its own) and it is de facto a process that only determines the time during which a given object (in this case, a vehicle) will be delayed. The actual design of the model, including the dividing into individual federates can be seen in Figure 7.

While this may be a relatively trivial demonstration case, by using it, we managed to prove that synchronization solved within the federation at the RTI level (or by calling RTI services) and synchronization solved by default by the framework are functional and the results of their (federates) cooperation are fully in line with expectations..

4.1.1. Testing Methodology

The actual simulation model was programed over the framework, and also in the simulation tool Arena. The outputs of the simulation tool Arena were then taken as the standard.
As for the motorway federate, the selected monitored indicators included verification of the functionality of the generators by comparing individual input streams and also verification of the dividing of the number of vehicles that just pass the section and the number of vehicles that turn off to some of the rest areas (in fact, the number of objects that are passed to the other federates).
As for the system of the rest areas, the selected monitored indicators were the numbers of vehicles and monitoring of their forwarding in time, travel time from the turning to the rest area, up to the arrival and stopping at the rest area (the same for departures) and service/stop time at the rest area.
Results of the solutions within monolithic synchronization in the commercial product Arena and within own programming solutions over the framework reached a high degree of conformity.

Figure 7: Dividing into federates

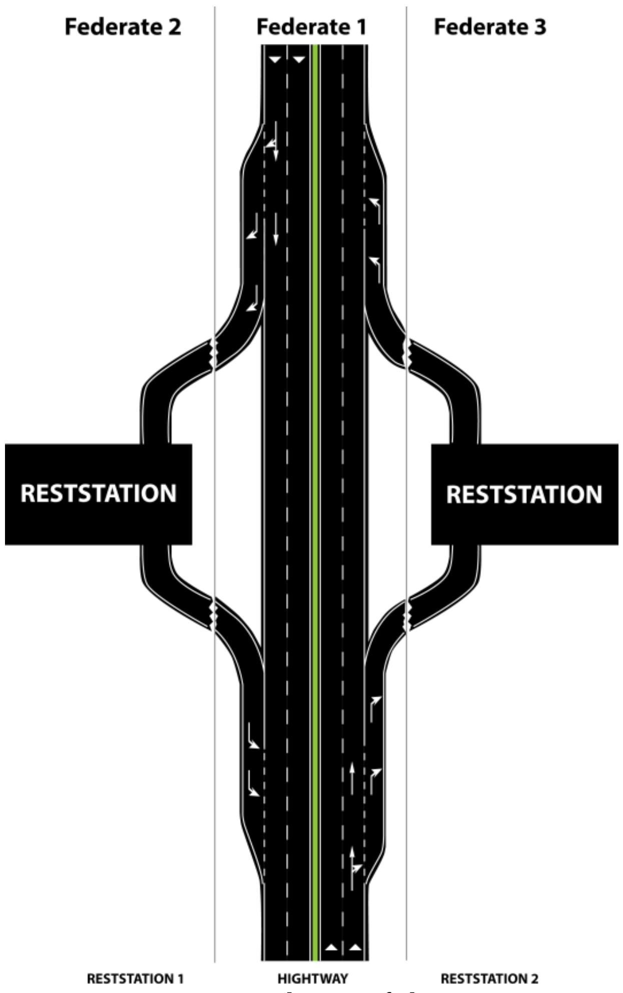

Figure 7: Dividing into federates

4.2. Other Prototypes

For validation and verification of the framework, other prototypes were created reflecting bulk service systems OR queuing systems. The basic principles are, however, very similar to the first demonstration model - they differ only in the area of implementation.

Another suitable prototype model chosen was a multi-storey building whose floors are connected both by a lift and by two staircases. The model is decomposed into three heterogeneous types of logical processes: the operator of the floor (it is a logical process that addresses the movement of people within the individual floors - each floor is operated by one common LP), the operator of vehicles (it is a logical process that addresses the movement between the floors where the objects are passed to the operator of the floor the moment they reach the destination floor) and the operator of doors (addresses the time individual people stay in a particular door - possible to replace with less general terms "surgery", if it was a hospital model, or "shop" if we declared the model to be a department store model. All doors are controlled by only one federate. The specific type of internal doors is entrance doors that serve as a generator).

Another area, for which it was desirable to create a demonstration example, was the possibilities of real-time synchronization and the possibility of interactive interventions into the course of the simulation during its course. Based on these requirements, a third demonstration model was created that reflects the operation in a railway transport system. Federates in this model represent individual stations (these may not be homogeneous) and a special federate (or a set thereof), which is responsible for railway sections between the stations. Operators of each station are enabled to perform interactive interventions into the control of individual signaling devices and railway switches.

For the purpose of this model, it is necessary to say that it is subject to a higher level of abstraction, specifically where it deals with motion characteristics of trains (acceleration, deceleration, driving dynamics). Although the model is designed so that individual characteristics have the potential for future extension so that the system corresponds to reality as much as possible, but currently it uses only the medium speed of the train with an step onset of the speed and step braking distance.

In addition to these examples, the framework is tested in the workplace on a number of demonstration examples ranging from communication testing of communication protocols through an analogue of chat), modelling without time synchronization (testing in turn-based strategy games) and application in the context of larger, potentially real-time simulations.

5. Conclusion

The framework makes it easier for the users to access HLA at the cost of certain restrictions. Prototyping of standard models is easier and faster and, moreover, the user needs no special knowledge of the HLA architecture.

The toll for easy access to the HLA simulations is the limited amount of synchronization methods and some handover protocols.

The framework does not aim to be applicable for large and complex models that consist of high-trained teams - those are assumed to have mastered HLA brilliantly. On the other hand, if a team begins with HLA, or just looks for a methodology though which they would connect existing simulators into the distributed simulation (without trying to specialize purely HLA), or just experiments with simulation technologies (when studying and teaching), this own framework is very useful tool for them as it will help bridge the initial problems with the implementation of distributed simulation models.

Prospects of further development are aimed at the implementation of optimistic synchronization methods and application in practice.

Acknowledgement
This article was created within the project Support for Internships and Professional Activities in Innovation in Tertiary Education at Jan Perner Transport Faculty and Faculty of Electrical Engineering and Informatics of the University of Pardubice, Reg. No.: CZ.1.07/2.4.00/17.0107.

References

Free software foundation, Inc, 2002, *CERTI - Summary [Savannah]*. Available from: http://savannah.nongnu.org/projects/certi [accessed 22 April 2014].

Fujimoto, Richard M., c2000, *Parallel and distributed simulation systems*. New York; John Wiley & Sons. ISBN 04-711-8383-0.

Geeknet, Inc., 2005, *SourceForge.net: Open HLA - Project Web Hosting - Open Source Software.* Available from: http://ohla.sourceforge.net/ [accessed 22 April 2014].

Knight, Pamela, Corder, Aaron, Liedel Ron, 2001, *Evaluation of Run Time Infrastructure (RTI) Implementations.* U.S. Army SMDS. Available from: https://www.scs.org/confernc/hsc/hsc02/hsc/papers/hsc017.pdf [accessed 15 July 2014].

Kuhl, Frederick, Dahmann, Judith, Weatherly Richard, The institute of electrical and electronics engineers, Inc, c2000, Upper Saddle River, NJ; Prentice Hall PTR. ISBN 01-302-2511-8.

Pitch technologies ab, 1993, *Products – Overview*, Available from: http://www.pitch.se/products/products-overview [accessed 22 April 2014].

Rabelo, Luis, Sala-Diakanda, Serge, Pastrana, John, Marin, Mario, Bhide, Sayli, Joledo, Oloruntomi, Bardina, Jorge, 2013. *Simulation Modeling of Space Missions Using the High Level Architecture.* Available from: http://www.hindawi.com/journals/mse/2013/967483/ [accessed 15 July 2014].

The institute of electrical and electronics engineers, Inc, 2010, *IEEE1516:2010: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Framework and Rules*. New York; IEEE. ISBN 978-0-7381-6251-5.

The institute of electrical and electronics engineers, Inc, 2010, *IEEE1516:2010: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Object Model Template (OMT) Specifications*. New York; IEEE. 2010.ISBN 978-0-7381-6249-2.

The institute of electrical and electronics engineers, Inc, 2010, *IEEE1516:2010: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA)-Federate Interface Specification*. New York;IEEE. ISBN 978-0-7381-6247-8.

The simulation interoperability standards organization,. Independent Throughput and Latency Benchmarking for the Evaluation of RTI Implementations, 2001, *The Simulation Interoperability Standards Organization*. Fall. DOI: SISO-01F-SIW-033.

Vtmäk, 2014, *HLA RTI – Run Time Infrastructure – MÄK RTI.* Available from: http://www.mak.com/products/link/mak-rti.html [accessed 22 April 2014].

Wikipedia: the free encyclopedia, 2014, *Run-time infrastructure (simulation).* Available from: http://en.wikipedia.org/wiki/Run-time_infrastructure_%28simulation%29 [accessed 4 April 2014].