Software Architecture for Rapid Development of HLA-Integrated Simulations for Critical Infrastructure Elements under Natural Disasters

G. Kirov, P. Zlateva, and D. Velev

***Abstract*—The paper proposes a software architecture for rapid development of high-level architecture (HLA)-integrated simulations for critical infrastructure elements under natural disasters. It considers the main issues involved in a simulation of complex interdependent systems, which are a part of the critical infrastructure (CI). It investigated an implementation of the object-oriented (OO) concepts into a High-Level Architecture (HLA)-networked simulation, thus addressing sustainability of the HLA methodology into the future. The paper aims at developing an object-oriented layer (OOL) providing a high-level mechanism for HLA data exchange through local objects. It exploits the technical advantages provided by layering, object-oriented programming, strict HLA object interface specification, and data-centric execution while presenting an application for simulation of CI. The proposed architecture can be used as a research platform, in which new models of CI elements and interdependencies under natural disasters can be evaluated.**

***Index Terms*—HLA/RTI, object-oriented programming, critical infrastructure, crisis management, natural disasters.**

I. Introduction

Nowadays, it is registered a growth in number and severity of natural disasters compared to previous years. It is necessary be pointed out that the negative impact of natural disasters on sustainable development of the critical infrastructure (CI) also increases. In this context it is noted that the critical infrastructure generally includes all systems and assets, both physical and virtual, which make vital contributions to national security, economic stability, public health, or safety [1]. Moreover, the infrastructure elements have complex relationships and interdependencies that cross critical infrastructure boundaries. This circumstance increases the range of threats, including threats, including threats caused by natural disasters. It raises the question for effective emergency management due to natural disaster taking into account the critical infrastructure interdependencies about vulnerability and consequences [2].

On other hand, as it is known the interdependencies between the components resulting in the mutual provision of services and use of common communications network, these systems cannot be studied directly. For example, it is not possible to directly test the effect of controlled shut-down procedures for segments of the power grid, the spread of a computer virus, or the spread of a disease vector under different vaccination options without having potentially drastic economic, safety or health impacts. For infrastructure networks, replicating all or even part of the physical infrastructure may be prohibitively expensive. In both these cases, computational modeling and simulation provides a safe and cost-effective alternative that can help enormously in developing needed understanding.

The modeling and analysis of interdependencies between critical infrastructure elements is a relatively new and very important field of study. A number of simulation models have been developed and more are being developed for studying individual aspect of infrastructure elements [3]. The value of these models decreases because they don't consider all aspects of a disaster. The simulation models addressing different aspects of an emergency situation need to be integrated in common framework to provide the whole picture of a situation to planners, trainers, and responders [4], [5].

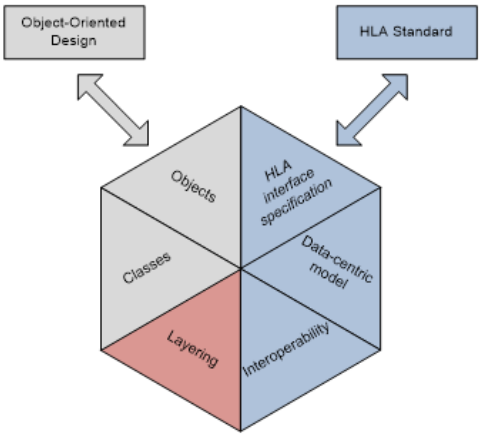

Fig. 1. Properties of the architecture for simulation of complex systems.

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 [6]. 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, HLA/RTI (High Level Architecture/Run Time Infrastructure) standard has a standardized structure for object models, they not

Manuscript received May 28, 2015; revised July 27, 2015. This work was supported in part by the Bulgarian National Science Fund for the support under the Grant NO. DFNI-I02/ 15 from 2014.

G. Kirov and P. Zlateva are with the ISER, Bulgarian Academy of Sciences, Sofia 1113, Bl. 2, Bulgaria (e-mail: g_tk@abv.bg, plamzlateva@abv.bg)

D. Velev is with the University of National and World Economy, Sofia 1700, Bulgaria (e-mail: dgvelev@unwe.bg).

completely correspond to common definitions of object models in object-oriented (OO) analysis and design techniques [7]. Given the current state of shrinking budgets and growing number of interdependent systems, it is obvious to almost any observer in the field of the software simulation that there is a critical need for an architecture that exploits the technical advantages provided by layering, object-oriented programming, strict HLA object interface specification, and data-centric execution (see Fig. 1).

The purpose of the paper is to present architecture for rapid development of HLA-integrated simulations for analysis of the interdependencies between critical infrastructure elements. This architecture allows HLA simulation systems to be integrated easily into larger network-centric systems.

II. Interdependencies and Cascading Effects

In this section, the interdependencies and cascading effects between the infrastructure objects are analyzed. According to one of the most widespread definition, interdependency is a bidirectional relationship between infrastructure objects through which the state of each infrastructure object is influenced by the state of the other. The interrelationship among infrastructure objects is a precondition for cascading effects. The cascading effects can occur when an infrastructure object disruption spreads beyond itself to cause appreciable impact on other infrastructure objects, which in turn cause more effects on still other infrastructure objects. The consequences of the cascading effects due to an infrastructure failure can range from the mild to the catastrophic.

For example, interdependencies exist between different sectors: national energy, water supply system, transport, telecommunication, and emergency services. A power outage can cause an effect on water supply system. The impact of the disruption may not stop at this level. It may go on to adversely affect other critical infrastructure objects (see Fig. 2).

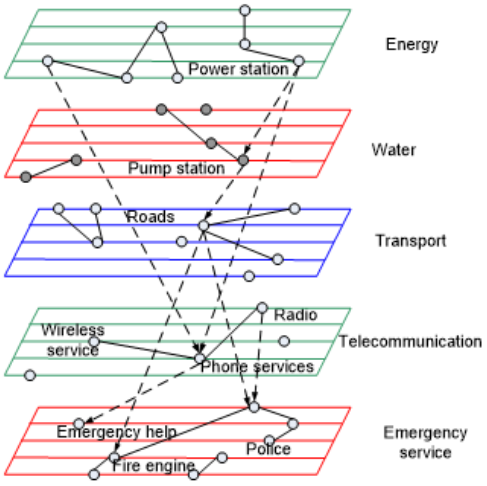

Fig. 2. Critical infrastructure interdependencies.

According to the system analysis there are four classes of interdependencies: physical, cyber, geographic, and logical [8], [9]:

Physical interdependency: two infrastructure objects are physically interdependent if the state of each depends upon the material output of the other. A requirement for this interdependency is a physical reliance on material flow from one infrastructure to another.

Cyber interdependency: two infrastructure objects are cyber interdependent if the state of the one depends on information transmitted through the information system of the other. This is relatively new type of interdependency which basic element is information transfer between infrastructure objects. For example, a lot of infrastructure objects use the SCADA systems for control and analysis. Therefore, the infrastructure objects have an information dependency from the SCADA systems.

Geographic interdependency: two infrastructure objects are geographically interdependent if a local environmental event can change the state in the two objects. The main requirement for this type of interdependency is the existence of a physical proximity. For example, a fire may affect and disrupt all the infrastructure objects located in the area. It is important to notice, that geographic interdependency exists not due to physical connections between infrastructure objects; rather, it arises from the influence the event exerts on all infrastructure objects simultaneously.

Logical interdependency: two infrastructures are logically interdependent if the state of each depends upon the state of the other via some mechanism that is not a physical, cyber, or geographic connection. This type of interdependency shows that infrastructure components may affect societal factors such as public opinion and cultural issues. For example, various regulatory mechanisms can give rise to logical linkage among two or more infrastructure objects [10].

One of the most used ways to present the infrastructure interferences is an interdependency matrix (see Table I).

It presents by coefficients ($r_{i,j}$) the influence or impact, that one infrastructure object can have, either directly or indirectly, upon another. The coefficients are defined by experts, which are responsible for an emergency management.

When the experts fill in the interdependency matrix they have to take into account the next two definitions:

- An infrastructure network *I* is a set of nodes, which are interconnected each other by a relation (connection) presented by function. The relation can be directional or bi-directional. The internal dependencies (connections) in an infrastructure *I* are presented by edges (*a*, *b*), with $a, b \in I$.

TABLE I: Interdependency Matrix

Interdependency matrix				
Infrastructure objects	Object 1	Object 2	…	Object n
Object 1		r1,2	r1,..	r1,n
Object 2	r2,1		r2,..	r2,n
	r.,1	r.,2		r..,n
Object n	rn,1	rn,2	rn,..	

- If *Ii* and *Ij are infrastructure networks*, in which $i \neq j$, $a \in Ii$, and $b \in Ij$, then interdependency are external one. It is defined as relation between infrastructures and presented by edge *(a,b)*. It means that node *b* is dependent upon node *a*: $(a,b) \rightarrow (b,a)$.

The interdependency analysis shows that the experts in the

area of Critical Infrastructure Protection (CIP) have recognized a lot of problems inherent to the complexity of national and multinational infrastructures. As an attempt to address these problems a number of approaches exist for investigation the interdependencies by showing the critical infrastructures and the ways in which they depend on each other. However, there is no quantitative assessment of the degree to which these infrastructures depend upon each other. Moreover, there is no an exact mathematical model, which evaluates the changes to the economy that result from these interdependencies.

Therefore, the above-mentioned considerations raise questions about the approaches for the infrastructure interdependencies. Analytical approaches study the problem based on over-simplified assumptions. This means the final results to be biased towards interdependencies under ideal conditions. The inaccuracy of analytical models focuses the expert attention on simulation to obtain correct results.

III. Distributed Simulation Technologies

The design and execution of distributed simulations has become increasingly important for the analysis of complex systems. In recent years, the Department of Defense (DOD) has invested considerable resources in infrastructures for distributed simulation modeling. The main simulation technologies are based on Distributed Interactive Simulation (DIS) protocol, Aggregate Level Simulation Protocol (ALSP), and High Level Architecture (HLA) (see Fig. 3). While the fundamental structure of each is similar, there are differences that can impact an application developer or the administrator of a distributed simulation exercise.

In 1983 the Defense Advanced Research Projects Agency (DARPA) sponsored the SIMNET (SIMulation NETworking) program to create a new technology to expand the current single task trainers into networked team trainers. SIMNET was tremendously successful, producing over 300 networked simulators.

The first standard for interactive distributed simulation was IEEE 1278.1, also known as the Distributed Interactive Simulation (DIS) protocol. Although DIS was originally developed for military applications, the technology is well suited as a simulator interoperability standard for civil application areas.

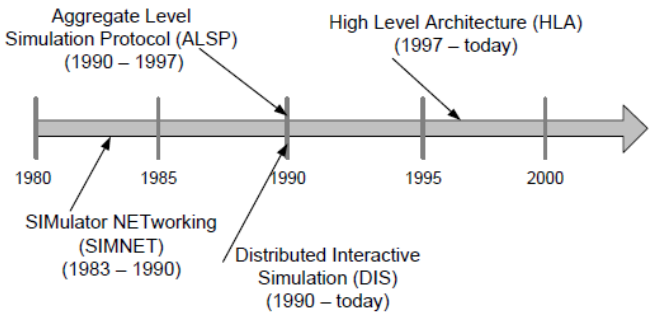

Fig. 3. Historical perspective.

It was based on the use of standard formatted packets, designed for the data required by these specific applications. DIS allows geographically separated simulators to work together, interacting in real-time, to provide predictions just like a single integrated simulator. The technology also allows real entities to be included in the simulation loop. The foundation of DIS is a standard set of messages and rules, called Protocol Data Units (PDUs), used for sending and receiving information across a computer network. The most common message is the Entity State PDU which represents all of the state information about a simulated entity that another simulator needs to know. The fact that there is no central server is perhaps the most surprising DIS characteristic. DIS used broadcast architecture, in which all data is transmitted to all simulators where it can be rejected or accepted depending on the receivers' needs. By eliminating a central server through which all messages pass, DIS dramatically reduces the time needed for a simulator to send important information to another simulator [11].

However, DIS has some drawbacks. Three features in the underlying data transport mechanism cause problems. Firstly, messages can get lost or arrive in the wrong order due to the use of the UDP/IP protocol. Secondly, the messages sent are part of standardised, fixed-sized Protocol Data Units (PDUs), although generic PDUs exist to communicate any type of data. Finally, due to the broadcast mechanism, the scaleability is rather limited. In the case that simulation experiments have to be repeatable, reliable data transfer is crucial [12].

Problems due to the inflexibility and lack of scalability of DIS approach have led to a different approach, the High Level Architecture (HLA), which becomes IEEE 1516 Standard. The HLA defines a set of rules governing how simulations, now referred to as federates, interact with one another. The federates communicate via a communication environment called the Runtime Infrastructure (RTI) and use an Object Model Template (OMT) which describes the format of the data. The HLA does not specify what constitutes an object, nor the rules of how objects interact. This is a key difference between DIS and the HLA.

Besides facilitating interoperability between simulations, the HLA provides the federates a more flexible simulation framework. Unlike DIS where all simulations receive every piece of data broadcast, the HLA federates use data management mechanism based on publishing and subscribing. These facts make it possible to have more simulations on a network at one time because the amount of data being sent is reduced. The simulation software is also simplified because it does not need to process extraneous information.

The Aggregate Level Simulation Protocol (ALSP) is a protocol and supporting software that enables simulations to interoperate with one another. Replaced by the HLA, it was used by the US military to link analytic and training simulations.

The potential advantages of distributed simulation technologies are evident: increased flexibility, building on existing software and communications standards, maximisation of the use of existing simulation assets, and thus reduced costs.

IV. HLA/RTI Simulation Standard

The proposed architecture allows communication between simulation models based on High Level Architecture/Run

Time Infrastructure (HLA/RTI) standard for information exchange. HLA is the IEEE standard for software architecture of interoperable distributed simulations. It aims to establish a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations among themselves, as well as to facilitate the reuse of modeling and simulation (M&S) components [13]. HLA allows experts to combine computer simulations into larger simulation. For instance, the experts might want to combine simulations of critical infrastructures in several different regions of the country. HLA can extend simulation later by adding new models or simulations, for example new models of infrastructures.

HLA provides the technical framework of simulation development in order to guarantee interoperability and reusability, and common services that are applicable to all types of simulations. This simulation architecture is designed for rapid integration of simulation components from various sources. It is comprised of three elements [14]:

- *HLA Rules:* A set of rules which must be followed to achieve proper interaction of simulations in a federation. These describe the responsibilities of simulations and of the runtime infrastructure in HLA federations. HLA rules describe the design goals and constraints for HLA compliant federations and federates.
- *Interface Specification:* Definition of the interface functions between the Run Time Infrastructure (RTI) and the federates in the HLA simulations. Implementation of RTI have a variety of forms. There are currently RTI application programming interfaces in CORBA IDL, C++, and Java as part of the HLA.
- *Object Model Template:* The prescribed common method for recording the information contained in the required HLA Object Model for each federation and simulation in terms of its objects and interactions. The OMT prescribes the allowed structure of FOM.

The relationships of HLA components are shown in Fig. 4. The conceptual model includes simulators, data collectors, passive viewers, and live surrogates simulations. They have a single point of attachment to the RTI. The federate might be a surrogate for human interactions. In this role it can effects the states of the remaining federates in simulation [15], [16].

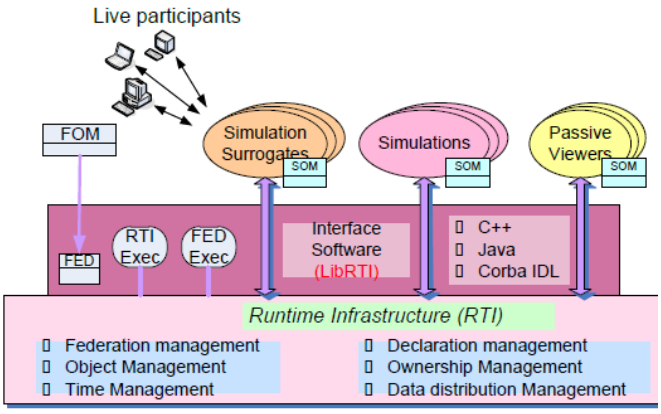

Fig. 4. HLA components.

The framework of the HLA that consists of several functional components:

Federate: All applications participating in a federation are called federates. In reality, this may include simulation models, data collectors, simulators, autonomous agents, passive viewers, or live entity surrogates simulations. The federate is a member of a HLA Federation that can represent a single model of the critical infrastructure or entire national critical infrastructure.

Federation: The combined simulation system created from set of federates that are interconnected with each other. Information exchange in the federation is based on a common object model, called Federation Object Model (FOM). It contains exchange data created by the federation developer that shows the relationships between federates. It means, FOM defines object classes, their attributes and interaction classes that are commonly used and exchanged among federates in the federation. The Simulation Object Model (SOM) is the model that defines objects, attributes and interactions in each federate that can be used from the other federates.

Federation Execution: Simulation session, in which a number of Federates participate, is called a Federation Execution. All simulated entities, such as different infrastructure elements or threats, are referred to as *objects.*

Run Time Infrastructure: RTI is a supporting software that provides 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 does not maintain information about the state of the federates, nor does it handle any semantics associated with the interaction between federates like what coordinate systems to use or what happens during a collision. Also, it does not specify the exact byte layout of data sent across the network. RTI and federates are software components. An RTI may support different federation execution at once.

RtiExec: RTIExec is a global process that controls the creation and destruction of the federation (simulation) in the network. Each federation is characterized by a single FedExec.

FedExec: FedExec is a process that manages federates in the federation. It is responsible for creation, joining and destruction of the federates during the simulation execution. FedExec process is created by the first federate in the federation.

LibRTI: LibRTI is a library that provides communication functions and services specified in the HLA/RTI standard [17] to federate developers. These RTI services are implemented in C++ or Java languages. The software specialists use LibRTI library to call the RTI services. A federate exchange information with the other federates, RtiExec, Fed Exec through LibRTI.

The HLA object model supports information exchange between federates within the federation. The exchange of information takes the form of objects and interactions. Federates communicate with their peers by sending interactions or updating object attributes. Federates do not communicate directly with each other and all communication is administrated by the RTI.

Object classes are comprised of attributes. Object classes describe types of things that can persist. Each object in a moment of time is characterized by a state, which is defined by a set of current values of its attributes. Federate, which manages an object (more precisely, the object attributes), 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.

Interactions classes [16] are comprised of parameters. An interaction is a single action caused by a change in the state of an object from another federation. Interaction classes describe types of events. Objects are similar to interactions in so much as objects are comprised of attributes and interactions are comprised of parameters. The basic difference between objects and interactions is persistence - objects persist, interactions do not.

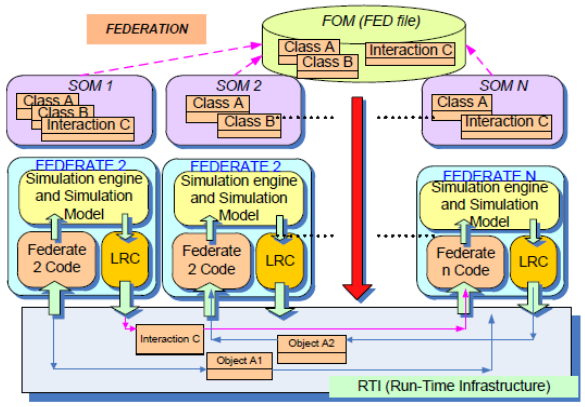

Fig. 5. Traditional HLA simulation.

A traditional HLA federate (Fig. 5) can be presented as an integrated program, consisting of a *simulation model* and *Local RTI Component (LRC)*. The simulation model is a physical, mathematical, or logical representation of processes and systems (user logic), whereas the LRC services it by interacting and synchronizing with other federates. Therefore, the simulation model performs local computing, while the LRC realizes information exchange for the model. It is very difficult to combine the above components in a normal federate, due to the tight coupling of the simulation model and the LRC [6].

In conclusion, the HLA FOM offers an object model that does not completely correspond to common definitions of object models in the object-oriented programming. The main purpose of the HLA FOM is to achieve interoperability between participants (federates) in the simulation, rather than between individual objects from different systems (federates). HLA doesn't support the transference of objects and their behaviors between joined federates.

Therefore, there are fundamental differences between object-oriented programming and HLA. 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. From the other hand, it is also necessary to make a number of assumptions about the HLA interactions between federates in order to fully use object-oriented features such as method invocations [18].

V. Object-Oriented HLA/RTI Simulation Architecture

This section shows sample code that implements the architecture for Object-Oriented (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 performance requirements for 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 object-oriented concept [1] of the HLA architecture is based on a strict hierarchical organization of classes that satisfy the requirements for building complex distributed simulation systems based on HLA standard. Basic class hierarchy is a class StandAlone. It contains features that allow the HLA classes to be integrated into the simulation environment. StandAlone has virtual methods and data members that HLA objects must inherit to be able to participate in the information exchange.

The remaining part of this point is a principle description of the reference model of an OO HLA object - 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. 6).

HLA_Model class contains two sections that are relevant to the HLA/RTI standard. The first section contains declarations of state variables. In the terminology of C ++ these variables are presented as a 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. Virtual functions meet the following basic steps defined by the HLA standard:

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

An example for building a simple OO HLA model of Hurricane is shown. It was developed following the specifications given into the reference model of a HLA object (HLA_Model). A disaster class (Fig. 7) is created initially, and then it inherits the base class StandAlone. It encapsulates common features of all disasters - speed, position, identification and etc.

```
/*************************************************************************
*       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. 6. C++ HLA_Model class.

In the class Disaster are not implemented virtual functions inherited from StandAlone because the simulation application never creates an object of class Disaster. This class is used only to be inherited by classes that are models of specific disaster such as a hurricane (see Fig. 8).

```
/*****************************************************************************
*              Class Disaster inherits the basic class StandAlone
*****************************************************************************/
class Disaster: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_ID;               // identification number
        UINT        m_T0;               // time to occur

        bool        m_status;                       //indicates the status of disaster
        UINT        m_TimeOfCreation;               //time of regist. in standAloneList
        UINT        m_TimeOfRegistrationRTI;        // time of regist. in  RTI
        double      m_zone;                         // disaster zone

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

Fig. 7. C++ Aircraft class.

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

        //disaster track
        vector<POINT3D>   m_Track;

// Virtal methods, which describes the behaviour of the Hurricane
        // it calculates new values for Hurricane  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. 8. C++ F16 class.

An instance of HLA object class Hurricane is created by the constructor of the Class Hurricane. 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. Object instance is a global representation maintained by the LRC. The same object instance is known to all federates by its globally unique handle value.

Like any simulation, HLA simulation is moved by the main simulation loop. The functions of the main simulation loop are related to: time management (real time simulation, fast simulation, etc.), increasing the simulation time with a time step, calling the virtual functions of the StandAlone class. All work on the processing of the OO HLA objects is done by the simulation loop. It is continuously repeated during which the virtual function of all OO HLA simulation objects are executed and attributes are updated. The simulation loop (Fig. 9) calls the virtual functions of the implemented HLA models because of 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. 9. Simulation loop.

The state variables are recalculated on an each time step of the simulation time by the function update(). In the example, the HLA object Hurricane recalculates the new position of the disaster. The updated values of the state variables are sent to the RTI environment by SendUpdate() 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.

VI. OOL Communication Model

The proposed architecture uses a publish-subscribe communication architecture that supports object-oriented updates to HLA object instances. The sequence diagrams (Fig. 10 and Fig. 11) present two parts of the communication model. The figures show the interactions between the model components need to work together to accomplish a communication task.

The publishing application (Fig. 10) creates a new HLA object for each information source (disaster, infrastructure objects, etc.). This object is stored in the static list of objects (standAloneList).

As a result, there is no need to recreate a HLA object upon receiving information from the same source. Therefore, when you need to send updated data for the same information source the program looks for the HLA object that has already been created. Then the application sends data by calling the *sendUpdate()* of the HLA object, thus making the connection between OOL and RTI functions.

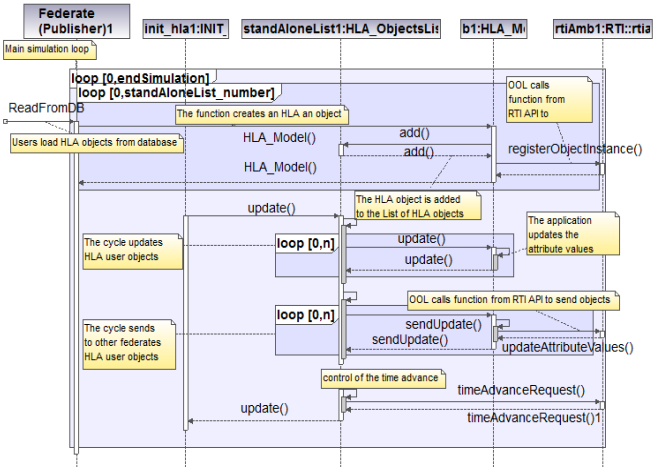

Fig. 10. Publishing application.

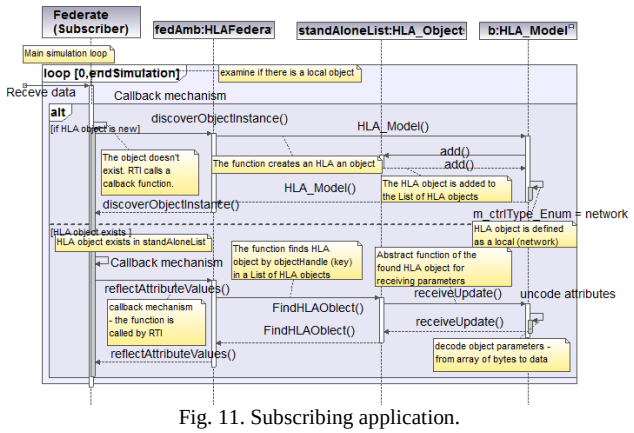

Fig. 11. Subscribing application.

In the subscribing mode (Fig. 11), the application accesses data by the function *receiveUpdate()*. It is a Callback function that has to be implemented by the programmer. The Callback function reads all data of a given HLA instance and searches in the list of HLA objects (standAloneList) if there is an object whose key matches this data. Once the application finds a relevant instance of the class the attribute values are updated.

VII. A Case Study: An Example of Integrated Simulation System

To verify the effectiveness of the new simulation technology, a case study was done on an integrated simulation of critical infrastructure interdependencies and their control mechanisms. The purpose of the integrated simulation is to observe how the critical infrastructure objects behave when unordinary events occur. The integrated simulation system is created from a set of models that are interconnected with each other. The proposed simulation system consists of several functional components [19]:

Simulation models: All simulated entities, such as different infrastructure elements or threats, are referred to as simulation models. It includes models of infrastructure objects, data collectors, and disasters. The simulation models consist of C++ code that access communication services provided by the RTI communication environment. This mechanism allows communication between simulation models based on HLA standard through RTI infrastructure. The communication between infrastructure objects in the integrated simulation is based on a common object model. It contains exchange data created by the developer that shows the relationships between models. Therefore, the common object model defines object classes, their attributes and interaction classes that are commonly used and exchanged among models in the simulation.

Viewer application: The viewer is developed to provide an integrated display environment (Fig. 12). It can act as a passive recipient and display simulation data from the rest of simulation system. The viewer is an important part of the simulation system because it provides analysis tools and playback capabilities. The viewer communicates with the simulation models over TCP/IP protocol that allows different models to reside on separate computers [5].

As a result of repeated execution of simulation, data is collected and analyzed, and the results are documented. The simulation results are presented in Table I. It display simulation time and state variables of the interdependent models exchanged through RTI environment.

A row of the table represents the time series of a state variable. A column represents the set of the simulated variables. The state of the variables at time *t* depends only on the states before *t*. The simulation results can be used for an analysis and assessment of the cascading effects and improving critical infrastructure protection.

VIII. Conclusions

The proposed simulation architecture is based on a distributed framework that can be rapidly implemented with interoperability standards for the modeling and simulation. Together, the framework and interoperability standards can significantly increase the use of modeling and simulation for natural disaster management. In turn, it will help improve the emergency management capabilities. This approach provides a possibility of doing an assessment of the elements of the CI and their interdependencies affected by an emergency situation due to natural disasters (see Table II).

TABLE II: Simulation Results

Space-time graph for disaster simulation							
Hurricane	Latitude	43.2299	43.2269	43.0365	43.9603	42.7368	42.5868
	Longitude	23.3271	23.3336	23.7375	23.8540	24.2699	24.5196
	Speed [km/h]	190	190	220	220	220	220
Power substation	Latitude	43.0295	43.0295	43.0295	43.0295	43.0295	43.0295
	Longitude	24.0012	24.0012	24.0012	24.0012	24.0012	24.0012
	Damages [%]	-	-	15	30	35	35
Residential area	Latitude	42.7514	42.7514	42.7514	42.7514	42.7514	42.7514
	Longitude	24.6549	24.6549	24.6549	24.6549	24.6549	24.6549
	People in disaster area [%]	-	-	-	2	23	58
	Electricity blackout [%]	-	-	-	32	43	56
	Damaged buildings [%]	-	-	-	-	16	23
Simulation time [sec]		300	310	880	1120	1750	2160

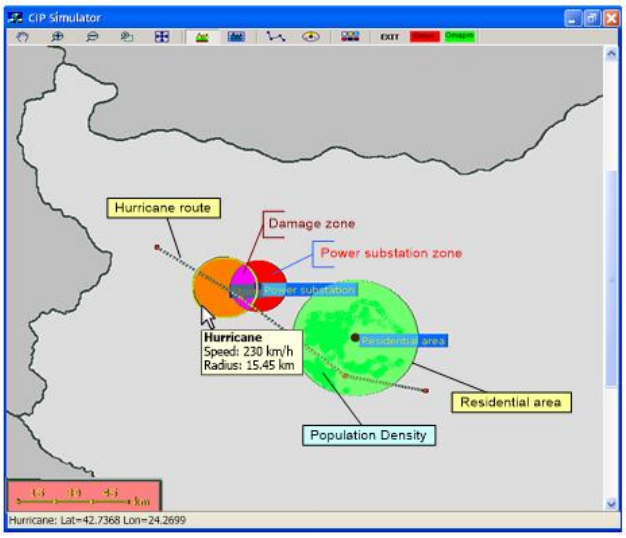

Fig. 12. Viewer application for hurricane simulation.

The main concept relies on the idea of providing a structural methodology and tools for the use of the methods to design, develop, and implement the HLA system, and obtain the interoperation and reuse of the simulation models. The potential advantages of the proposed architecture are evident: increased flexibility, building on existing software and communications standards, and maximisation of the use of existing simulation models.

With HLA/RTI simulation infrastructure interdependencies under natural disasters can be modeled with almost any level of detail desired and the design space can be explored more finely than is possible with analytical-based approaches or measurements. HLA simulation can combine different simulation models easily, and incorporate measured characteristics of infrastructure objects and their interdependencies.

ACKNOWLEDGMENT

The authors express their gratitude to the Bulgarian National Science Fund for the financial support under the Grant No. DFNI-I02/ 15 from 2014, titled "Information System for Integrated Risk Assessment from Natural Disasters".

REFERENCES

[1] P. Edwards, "Millennial reflections on computers as infrastructure," *History & Technology*, vol. 15, pp. 7-29, 1998.
[2] M. Swanson, A. Wohl, L. Pope, T. Grance, J. Hash, and R. Thomas, *Contingency Planning Guide for Information Technology Systems*, NIST Special Publication 800-34, 2002.
[3] S. Jain and C. McLean, "A framework for modelling and simulation of emergency response," in *Proc. 35th Conference on Winter Simulation: Driving Innovation*, New Orleans, Louisiana, pp. 1068-1076, 2003.
[4] S. Jain and R. McLean, "Modelling and simulation of emergency response," Workshop Report, Relevant Standards and Tools, National Institute of Standards and Technology Internal Report, NISTIR-7071, 2003.
[5] G. Kirov and V. Stoyanov, "Software architecture for implementation of complex simulation systems," *Cybernetics and Information Technologies*, vol. 8, no. 4, pp. 57-68, 2008.
[6] D. Chen, S. J. Turner, W. Cai, and M. Xiong, "A decoupled federate architecture for high level architecture-based distributed simulation," *J. Parallel Distrib. Comput.*, vol. 68, pp. 1487-1503, 2008.
[7] A. Tolk, "HLA-OMT versus traditional data and object modeling," in *Proc. Command and Control Research and Technology Symposium*, Annapolis, Maryland, 2001.
[8] S. Rinaldi, J. Peerenboom, and T. Kelly, "Identifying, understanding and analyz-ingcritical infrastructure interdependencies," *IEEE Control Systems Magazine*, pp. 11-25, 2001.
[9] P. Pederson, D. Dudenhoeffer, S. Hartley, and M. Permann, "Critical infrastructure interde-pendency modeling: A survey of U.S. and international research," Prepared for the technical support working group under work for others agreement 05734, 2006.
[10] S. Rinaldi, "Modeling and simulating critical infrastructures and their interdependencies," in *Proc. IEEE 37th Hawaii International Conference on System Sciences*, 2004.
[11] L. Argüello and J. Miró, "Distributed interactive simulation for space projects," *ESA bulletin*, vol. 102, 2000.
[12] G. Jense, H. Kuijpers, and A. Dumay. DIS and HLA: Connecting people, simulations and simulators in the military, space and civil domains. [Online]. Available: http://wwwpa.win.tue.nl/kuijpers/publications/iaf97.pdf.
[13] *IEEE Standard for Modeling and Simulation* (M&S), IEEE Std 1516.2-2000, 2010.
[14] F. Kuhl, R. Weatherly, and J. Dahmann, *Creating Computer Simulation Systems: An Intro-duction to the High Level Architecture*, Prentice Hall PTR, 1999.
[15] J. Kim and T. Kim, "Hierarchical HLA: Mapping hierarchical model structure into hierarchical federation, *M&S-MTSA'06*, Ottawa, Canada, pp. 75-80. 2006.
[16] The HLA. [Online]. Available: http://www.dmso.mil/public/transition/hla
[17] S. Kanai and T. Kishinami, "Multi-disciplinary Distributed Simulation for designing IT Devices by integrating off-the-shelf CAX systems based on HLA," in *Proc. Fall 2004 Simulation Interoperability Workshop*, 2004, pp. 315-325.
[18] B. Möller and F. Antelius, "Object-oriented HLA - Does one size fit all," in *Proc. 2010 Spring Simulation Interoperability Workshop*, 10S-SIW-058, Simulation Interoperability Standards Organization.
[19] G. Kirov, "Integration of simulation models of the elements of the critical infrastructure in common simulation environment," in *Proc. Scientific Support of the Transformation in Security Sector Conference*, CNSDR-BAS, Sofia, 2006, pp. 143-159.

Georgi Kirov is currently an associate professor at the Institute of System Engineering and Robotics at the Bulgarian Academy of Sciences, Sofia, Bulgaria. He holds the M.Sc degree in computer systems from the Sofia Technical University and the Ph.D. degree in soft computing technologies from the Institute of Computer and Communication Systems. His main areas of academic and research interest are distributed information technologies, computer simulation, and risk management.

Plamena Zlateva is currently an associate professor at the Institute of System Engineering and Robotics at the Bulgarian Academy of Sciences, Sofia, Bulgaria. She hods M.Sc. degrees in applied mathematics from the Sofia Technical University and in economics from the Sofia University St. Kl. Ohridski, and Ph.D. degree in manufacturing automation from the Institute of Conrol and System Research - BAS. Her main areas of academic and research interest are control theory, mathematical modeling and risk management.

Dimiter Velev is a professor in the Department of Information Technologies and Communications at the University of National and World Economy, Sofia, Bulgaria. He holds the M.Sc. degree in electroengineering from the Sofia Technical University, Bulgaria and the Ph.D. degree in engineering sciences from the Institute of Modeling Problems in power engineering at the National Academy of Sciences of Ukraine, Kiev, Ukraine. His main areas of academic and research interest are internet-based business systems modeling and development, service oriented architectures, online social networks, cloud computing, web applications development and programming.