European Research Network on Foundations, Software Infrastructures and Applications for large scale distributed, GRID and Peer-to-Peer Technologies
A Network of Excellence funded by the European Commission

# HLA Component Based Environment for Distributed Multiscale Simulations

*Katarzyna Rycerz*[1,2], *Marian Bubak*[1,2], *Peter M.A. Sloot*[3], *Vladimir Getov*[4]
{kzajac,bubak}@agh.edu.pl
{sloot}@science.uva.nl
{V.S.Getov}@wmin.ac.uk
[1]*Institute of Computer Science AGH, al. Mickiewicza 30, 30-059 Kraków, Poland*
[2]*Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland*
[3]*Faculty of Sciences, Section Computational Science, University of Amsterdam Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*
[4]*School of Computer Science University of Westminster Watford Rd, Northwick Park Harrow HA1 3TP, U.K.*

CoreGRID Technical Report
Number TR-0137
April 22, 2008

Institute on Grid Systems, Tools, and Environments

CoreGRID - Network of Excellence
URL: http://www.coregrid.net

CoreGRID is a Network of Excellence funded by the European Commission under the Sixth Framework Programme

Project no. FP6-004265

# HLA Component Based Environment for Distributed Multiscale Simulations

Katarzyna Rycerz[1,2], Marian Bubak[1,2], Peter M.A. Sloot[3], Vladimir Getov[4]
{kzajac,bubak}@agh.edu.pl
{sloot}@science.uva.nl
{V.S.Getov}@wmin.ac.uk
[1]Institute of Computer Science AGH, al. Mickiewicza 30, 30-059
Kraków, Poland
[2]Academic Computer Centre – CYFRONET, Nawojki 11,
30-950 Kraków, Poland
[3]Faculty of Sciences, Section Computational Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
[4]School of Computer Science University of Westminster
Watford Rd, Northwick Park Harrow HA1 3TP, U.K.

*CoreGRID TR-0137*

April 22, 2008

**Abstract**

In this paper we present the Grid environment that supports application building basing on a High Level Architecture (HLA) component model. The proposed model is particularly suitable for distributed multiscale simulations. Original HLA partly supports interoperability and composability of simulation models, where connections between modules (federates) in a simulation system (federation) are defined and set by federates themselves. On the contrary, in the proposed component model the particular behavior of component and it's connections are defined and set by an external module (e.g. builder) on the user request which is more flexible and increases reusability of components. We are also proposing to integrate our HLA component solution with the Grid which will allow users working on distributed simulations to more easily exchange the models already created. The focus of this paper is on design of the HLA component. We show how to insert simulation logic into a component and make possible to steer from outside its connections with other components. Its functionality is shown on the example of multiscale simulation of a stellar system.
Keywords: Components, Grid computing, HLA, distributed simulation

## 1 Introduction

Environments supporting application building from existing software components on the Grid are an interesting topic of research. In this paper we would like to propose such environment oriented towards distributed simulations consisting of modules of different time and space scale (multiscale). The paper describes a High Level Architecture (HLA) component model that defines software modules comprising application to be build.

We have choosen HLA [5] as it is a standard for large scale distributed interactive simulations and offers many advanced features specific for such applications (like time, data and ownership management). It also offers the ability of plugging and unplugging various simulation models ( also with different internal types of time management) to/from

This research work is carried out under the FP6 Network of Excellence CoreGRID funded by the European Commission (Contract IST-2002-004265).

a complex simulation system. Additionally, HLA introduces a uniform way of description of events and objects being exchanged between federates. Also, HLA separates communication runtime infrastructure (RTI) from actual simulation. All these features can be used to create HLA-based component model, where components are independent simulation modules that can be dynamically joined into a coherent whole. The difference between the component view proposed in this paper and an original HLA approach is that the behavior of component and it's connections are defined and set by an external module on the user request. In original HLA, the connections between federates in a federation are defined and set by federates themselves. The proposed approach is more flexible and increases reusability of components as it separates component developers from the users wanting to set up particular distributed simulation system from existing components. It also differs from other popular component models (e.g. CCA [1]), as the federates are not using direct connections. Instead, all federates within a federation are connected together using a tuple space, which they can use for subscribing and publishing events and data objects. The federates can also make use of advanced time management, which is particularly useful for multiscale simulations.

In this paper we propose to build the Grid environment that will support HLA component model. We are using Grid technology [7, 16], as it is oriented towards joining geographically distributed communities of scientists working on similar problems - this will allow users working on distributed simulations to more easily exchange the models already created. Therefore, the attempt to integrate HLA with new possibilities given by both Grid and component technologies is a promising approach. As a Grid platform hosting HLA components we have chosen H2O environment [7].

In this paper we focus on the design of HLA component itself. We show how to insert simulation logic into a component and how to make it possible to steer its connections with other components from outside. The functionality of the system is shown on the example of multiscale simulation of a dense stellar system.

The approach described in this paper is directed to the users that want to create new multiscale simulation systems from existing components or join their own new component to the multiscale system. For the users that have they own HLA application and want to run it almost unaltered efficiently using the Grid, we suggested using our previous work [13], where we have focused on execution management of existing legacy HLA applications and the best usage of available Grid resources, which can be achieved by using provided migration and monitoring services.

This paper is organized as follows: in Section 2 we outline related work, in Section 3 we describe the HLA component model. Section 4 presents the idea of the Grid support system for such model and the design and implementation of a HLA component - the element of the designed system responsible for storing simulation logic and enabling steering its connections from outside of it. Section 5 presents experiment with example multiscale simulation of a dense stellar system. Summary and future plans are described in Section 6.

## 2 Related Work

Building application from existing software modules is a wide range topic. This issue includes defining interoperable and reusable pieces of software – services and component technologies. The most popular services standards include Web Services [15] and its extension with stateful resources [16]. Among component standards worth to be mentioned are: Common Component Architecture (CCA)[1] (with its implementations like XCAT[6] or MOCCA [8]), Fractal [2] and its extension - Grid Component Model [10] (with its implementation ProActive [12]). However, none of this models provides advanced features for distributed multiscale simulations. In particular they do not support advanced time management mechanism, which in our model is achieved by integrating mechanism provided by HLA with component solutions. An important approach to using services and component technology to distributed simulations is described in [3]. However, the proposed solution is addressed in general to distributed simulations, without special focus on multiscale simulations systems. Another worth to be mentioned component framework for simulations [11] is specifically designed for partial differential equations.

## 3 HLA Component Model

As already mentioned in the previous Section, one of the important features of HLA is the ability of plugging and unplugging pieces of functionality to/from a complex application. In that sense it is possible to create a HLA–based component model. Unlike popular component models (e.g. CCA [1]), the federates are not using direct connections (e.g. in CCA one component is connected with other component, when its *uses port* is connected with partner's *provides port*). Instead, all federates within federation are connected together using tuple space, which takes care

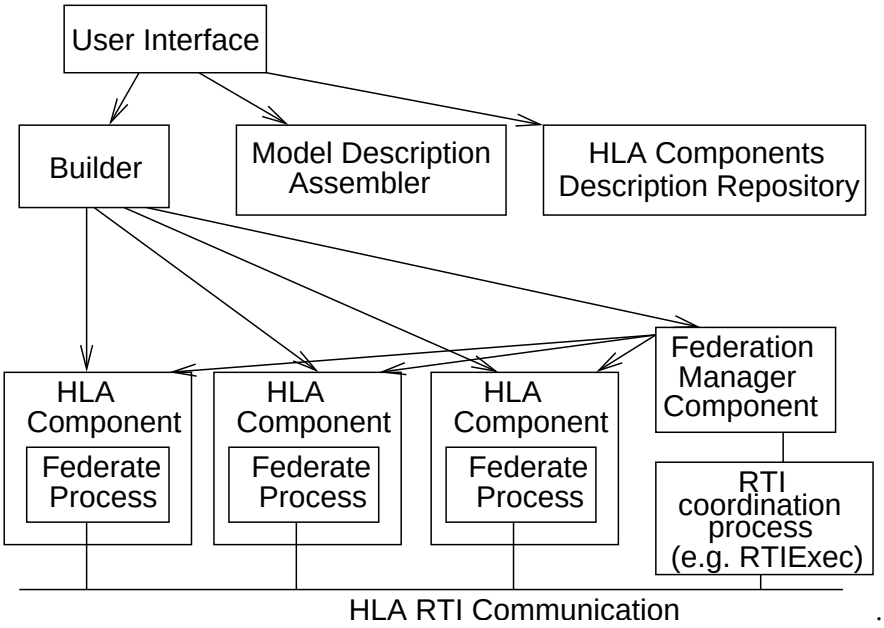


Figure 1: Grid system supporting HLA–based component model.

of sending the appropriate data from the publisher to the subscriber. HLA also includes advanced time management mechanism that allows to connect federates with different internal time management together. It is possible for federates to dynamically subscribe/unsubscribe and publish/unpublish their data as well as dynamically change their use of time management. Additionally, these decisions can not only be taken by the actual federate itself, but also by other federate that can steer subscription/publication mechanism of others.

All these features allow to think about a HLA component as about an entity that can be joined to a set of other components (the set represents a federation) and interact with them by publish/subscribe mechanism of exchanging data and using HLA time management if necessary. If needed, each component can also be executed independently of others. The presented model will be especially useful for the applications that would benefit from HLA (mainly distributed simulations). Because of advanced HLA time management facility that enables to join components of different internal time management and scale, it would be particularly useful for multiscale simulations, on which we would like to focus.

The difference between a component view proposed in this paper and an original HLA approach is that the particular behavior of component and it's connections are defined and set by an external module on the user request. This enables the user to create federations from federates developed by others without changing their implementation. The particular federation, in which a federate is going to take part, does not need to be defined by a federate developer, but can be created later – from outside – in the process of setting up distributed simulation system. Therefore the presented approach increases reusability and composability of simulations.

# 4 Grid system supporting HLA components

In this paper we would like to propose a solution that would support the HLA component model on the Grid. The user will be able to decide how components will interact with each other (e.g. by setting up appropriate subscription/publication and time management mechanism). The user also will be able to plug/unplug components and change nature of their interactions during simulation runtime. Fig.1 shows the proposed support for HLA component model (CompoHLA). Apart from the actual HLA communication level, there is a Grid level consisting of following elements:
**Builder** – sets up a simulation system on behalf of the user. It uses Federation Management Component to create federation and instructs HLA Components to join it. It also can instruct chosen components to set appropriate time management mechanism and subscribe or publish chosen data objects or events.
**HLA Component Description Repository** – stores description of components - including information about data

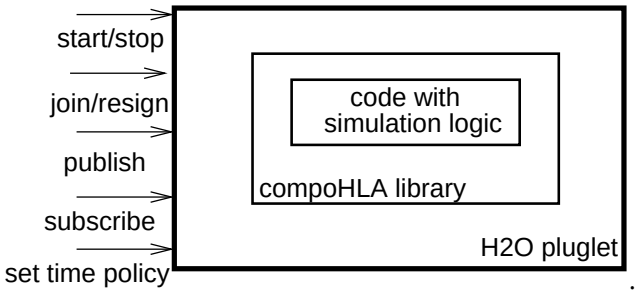


Figure 2: Relationship between component's developer code (simulation logic), HLA RTI implementation and compoHLA library.

objects and interactions that the component can exchange with others (which is called Simulation Object Model), the type of time management that makes sense for this component and additional information that may be useful for the user that wants to set up multiscale system (e.g. units of produced data, scale of simulation time, if rollback is possible, how subscription for particular data affects simulation, average execution time etc.).

**Model Description Assembler** - produces Federation Object Model needed to start federation from given Simulation Object Models of components that will comprise simulation system.

**Federation Manager Component** - manages whole federation on the component level and sets up connection with coordination process for federations.

**HLA Components** – wrap actual functionality of federates into components (described later in this section)

The relations between system elements is shown in the Fig.1. A user can use the HLA Components Description Repository and the Model Description Assembler to build a federation description and pass it to the Builder that sets up the federation from appropriate HLA Components. The user than can dynamically change nature of connections between components using the Builder.

**HLA Component in CompoHLA.** As described in Section 3, a HLA component should be able to be joined to/resigned from federation as well as be able to react on user requests to subscribe/publish appropriate data and use time management mechanism if necessary. In this paper, we present HLA component prototype that is designed as entity that can be requested to join the federation and then to resign from it during component lifetime. Independently from being joined/disjoined each component can be started and stopped during its lifetime. In [14] we described how the user can change interactions (subscription/publication and time management) between components during lifetime. As a Grid framework we have chosen the H2O [7] platform as it is lightweight and enables for dynamic remote deployment. A HLA component is implemented as a H2O pluglet having requests to start, stop, join, resign (described in this paper) and requests to change publications, subscriptions as well as type of time management (described in [14]). The component developer has to provide a simulation logic code which is connected with a pluglet by interfacing the compoHLA library as shown in the Fig.2.

The more detailed relations in the form of a simplified class diagram between HLA RTI, the compoHLA library and a developer code (simulation logic) are shown in the Fig.3. The CompoHLA library introduces two classes with abstract methods that should be overridden by component developer. One is a `CompoHLASimulator` class, from which the developer has to inherit and point to the main function starting a simulation. There is also a `CompoHLADataObject` class that has to be inherited for each data object that is going to be published by the federate and be visible outside for an external user (who is going to chose this component to be connected to his simulation system). The developer has to specify how the actual simulation data fits into HLA data objects that could possibly be exchanged with other federates.

The simulation developer can also call methods of a `CompoHLAFederate` class which, in turn, uses HLA a `RTIambassador` class (main class providing HLA services). The methods include getting info about federate time and requests of time advance as well as checking if stop request came (in order to perform final operations before the simulation exit).

Also, a developer has to override `FederateAmbassador` class callbacks (there are used by RTI to communicate with a developer code e.g. when receiving data from other federates) as in an original RTI federate. The use of the compoHLA library does not free the developer from understanding HLA time management and data exchange mechanisms, but simplifies use of them and allows a HLA component to be steered from outside (by external requests

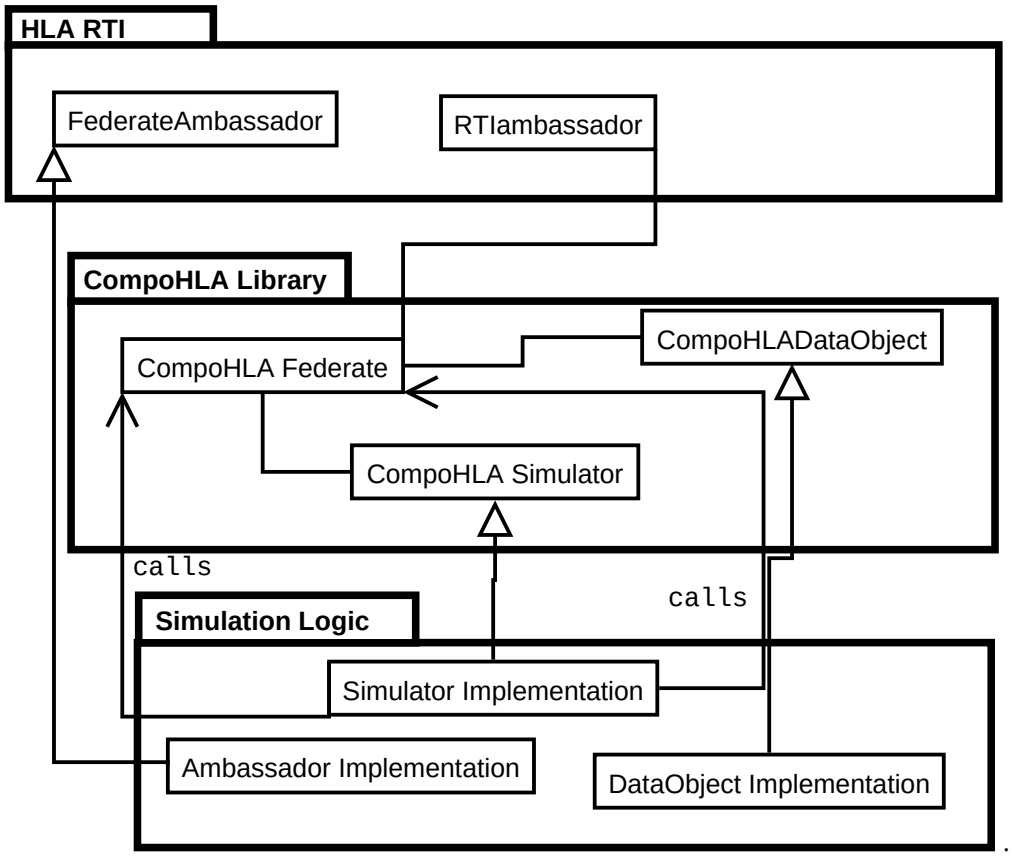


Figure 3: Simplified class diagram illustrating relations between crucial classes of HLA RTI, compoHLA library and component developer code (simulation logic).

as described above).

# 5 Experiments with MUSE

For the purposes of this research we have used simulation modules of different time scale taken from Multiscale Multiphysics Scientific Environment (MUSE)[9] for simulating dense stellar systems like globular clusters and galactic nuclei. The original MUSE consists of the Python scheduler and three simulation modules of different time scale: stellar evolution (in macro scale), stellar dynamics (nbody simulation - in meso scale) and hydro dynamics (simulation of collisions - in micro scale). Also, there are plans to add additional modules. For the purposes of this paper, we have chosen to make components from two MUSE modules: evolution (macro scale) and dynamics (meso scale) that run concurrently. The simulation system has to make sure that dynamics will get update from evolution before it actually passes the appropriate point in time. The HLA time management mechanism [5] of *regulating* federate (evolution) that controls time flow in *constrained* federate (dynamics) could be there very useful [5]. The advantage of HLA Component model is that it should allow this mechanism to be accessible to the external user that would like to set up the simulation system from existing dynamics and evolution components created by someone else.

**Performance Results.** We have created two prototype HLA components for dynamics and evolution simulations and measured execution time of requests to them. In our implementation we have used H2O v2.1 and HLA Certi implementation v3.2.4. Experiments were done on Dutch Grid DAS3 [4]. The RTI control process was run on Grid node at the Amsterdam Free University the dynamics component at University of Amsterdam, the evolution component at Delft, and the client was run at Leiden University. The bandwidth between Grid sites is 10Gbps.

We have tested following scenario: start dynamics – start evolution – join dynamics to federation – join evolution to federation – resign dynamics – resign evolution – stop dynamics – stop evolution. Tab.1 shows results of this

| Request | avr time, sec | $\sigma$ |
|---|---|---|
| start dynamics | 0.008 | 0.001 |
| start evolution | 0.009 | 0.001 |
| join dynamics | 0.181 | 0.003 |
| join evolution | 0.52 | 0.08 |
| resign dynamics | 0.006 | 0.0004 |
| resign evolution | 0.007 | 0.0003 |
| stop dynamics | 0.3 | 0.2 |
| stop evolution | 0.2 | 0.2 |

Table 1: Time of HLA Component request execution for evolution and dynamics components taken from MUSE [9]

experiment (average of 10 runs). In general, execution time of all requests are small. The `start`, `stop` and `resign` requests are similar for both modules. However, as we can see, realization of `join` request by second component is longer then by first component. This can be explained by the fact that joining to the federation that already have some members requires to make connections to these members. These overhead depends on the HLA implementation, not the design of the HLA component. Also performance of `stop` request requires explanation. The request does not stop the simulation immediately, but sends requests to the simulation and waits for it to check if that request came (we would like to give the control to the component developer and let him to save the results of a simulation, if necessary). Therefore, the execution time of `stop` request can vary depending on this waiting time. This is illustrated by quite large $\sigma$. To summarize, execution times of all requests are promising and show that component layer does not introduce much overhead, but we have to have in mind factors independent on the HLA Component design (overhead of particular HLA implementation and the frequency of checking if the `stop` request came in the component developer code).

# 6 Summary and Future Plans

In this paper we have presented the idea of a HLA component model, which enables the user to dynamically compose/decompose distributed simulations from multiscale elements residing on the Grid. We have also shown the architecture of the system supporting such model and build preliminary prototype of a HLA component that stores simulation logic and makes possible to steer from outside its connections with other components. This approach differs from that in original HLA, where all decisions about actual connections are made by federates themselves. The functionality of the prototype is shown on the example of multiscale simulation of a dense stellar system – MUSE environment [9]. The results of the experiment show that that component layer does not introduce much overhead. In the future we plan to fully design and implement other modules of the presented support system.

**Acknowledgments:** The authors wish to thank Maciej Malawski for discussions on component models and Simon Portegies Zwart for valuable discussions on MUSE. This research was also partly funded EU IST Project CoreGRID and the Polish State Committee for Scientific Research SPUB-M.

# References

[1] R. Armstrong, G. Kumfert, L. C. McInnes, S. Parker, B. Allan, M. Sottile, T. Epperly, and T. Dahlgren. The CCA component model for high-performance scientific computing. *Concurr. Comput. : Pract. Exper.*, 18(2):215–229, 2006.

[2] E. Bruneton, T. Coupaye, and J.-B. Stefani. Recursive and dynamic software composition with sharing. In *Proceedings of Seventh International Workshop on Component-Oriented Programming*, June 2002.

[3] X. Chen, W. Cai, S. J. Turner, Y. Wang: SOAr-DSGrid: Service-Oriented Architecture for Distributed Simulation on the Grid. Principles of Advanced and Distributed Simulation (PADS) 2006: 65-73

[4] The Distributed ASCI Supercomputer 3 web page `http://www.cs.vu.nl/das3`

[5] IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA), 2004. http://standards.ieee.org/catalog/olis/compsim.html.

[6] S. Krishnan and D. Gannon. XCAT3: A Framework for CCA Components as OGSA Services. In *Proc. Int. Workshop on High-Level Parallel Progr. Models and Supportive Environments (HIPS)*, pp. 90–97, Santa Fe, NM, USA, 2004.

[7] D. Kurzyniec, T. Wrzosek, D. Drzewiecki, and V. S. Sunderam. Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach. *Parallel Processing Letters*, 13(2):273–290, 2003.

[8] M. Malawski, D. Kurzyniec, and V. S. Sunderam. MOCCA - Towards a Distributed CCA Framework for Metacomputing. In *19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, 4-8 April 2005, Denver, CA, USA*, 2005.

[9] MUSE Web page http://muse.li/

[10] OASIS team: Proposals for a Grid Component Model CoreGRID project Technical report, 2004. http://www.coregrid.net

[11] S.G. Parker. A Component-Based Architecture for Parallel Multi-physics PDE Sim ulation. Future Generation Computer Systems, 22(1-2):204–216, 2006.

[12] ProActive project homepage. http://www-sop.inria.fr/oasis/ProActive/.

[13] K. Rycerz, M. Bubak, P.M.A. Sloot, V. Getov: Problem Solving Environment for Distributed Interactive Simulations in: S. Gorlatch, M. Bubak, and T. Priol (Eds). Achievements in European Reseach on Grid Systems. CoreGRID Integration Workshop 2006 Springer, 2008, pp 55 - 66.

[14] K.Rycerz, M. Bubak, P.M.A. Sloot Dynamic Interactions in HLA Component Model for Multiscale Simulations. Proccedings of International Conference on Computational Science, ICCS 2008. (to appear)

[15] Web Services. http://www.w3.org/2002/ws/.

[16] Web Services Resource Framework. http://www.globus.org/wsrf.