Dynamic Interactions in HLA Component Model for Multiscale Simulations

Katarzyna Rycerz[1,2], Marian Bubak[1,3], and Peter M.A. Sloot[3]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre CYFRONET AGH, Nawojki 11, 30-950 Kraków, Poland
[3] Faculty of Sciences, Section of Computational Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{kzajac,bubak}@uci.agh.edu.pl, sloot@science.uva.nl
Phone: (+48 12) 617 39 64, Fax: (+48 12) 633 80 54

Abstract. In this paper we present a High Level Architecture (HLA) component model, particularly suitable for distributed multiscale simulations. We also present a preliminary implementation of HLA components and the CompoHLA environment that supports setting up and managing multiscale simulations built in the described model. We propose to integrate solutions from High Level Architecture (such as advanced time and data management) with possibilities given by component technologies (such as reusability and composability) and the Grid (such as joining geographically distributed communities of scientists). This approach will allow users working on multiscale applications to more easily exchange and join the simulations already created. The particular focus of this paper is on the design of a HLA component. We show how to insert simulation logic into a component and make it possible to steer from outside its connections with other components. Its functionality is shown through example of multiscale simulation of dense stellar system.

Keywords: Components, Grid computing, HLA, distributed multiscale simulation, problem solving environments.

1 Introduction and Motivation

Building a simulation system from modules of different scale is an interesting, but non-trivial issue. There are many approaches to combine different models and scales in one problem solution, which is difficult and depends on actual models being combined. When designing problem solving environments for simulation of multiphysics multiscale systems, one can identify issues related to actual connection of two or more models together (this includes support for joining models of different internal time management, efficient data exchange between two modules of different scale etc.). Apart from that, there are also issues related to reusing existing models and their composability. For scientists working on multiscale systems it could be useful to find already existing models needed and connect them either together or to their own models. To facilitate

M. Bubak et al. (Eds.): ICCS 2008, Part II, LNCS 5102, pp. 217–226, 2008.

that, there is a need to wrap their simulations into recombinant components that can be selected and assembled in various combinations to satisfy specific requirements. Also, there is a need for an infrastructure that allows to exchange these components between scientists in relatively easy way across administrative domains.

To summarize, there is a need for a component based model with support specific for multiscale simulations (complex time interactions, efficient data transfer between modules of different scale etc) and for an environment that would allow to build and reuse such components across administrative domains. For the first requirement, we decided to use solutions provided by High Level Architecture (HLA) [1], especially because of its advanced time management mechanism that allows to connect modules of different internal time management. Together with time management, other HLA services (data management, ownership management etc.) give a powerful tool for multiscale simulations [2].

Additionally, HLA partially supports the second requirement – interoperability and composability of simulation models, as it separates actual simulations from the communication infrastructure. Also, each simulation (federate) when connecting to simulation system (federation) is required to have description of objects and events exchanged with others. HLA even provides Management Object Model (MOM) that allows one of the federates in the federation manage connections of the others. However, for a component approach this is not enough, as dynamic joining federates in the federation and easy manipulating their connections from outside by the third party is not directly possible. To satisfy this requirement we propose a HLA-based component model that allows an external module (e.g. builder) to define and control particular behavior of components and their connections on the user request. A user is able not only to dynamically set up multiscale simulation system comprised of chosen HLA components residing on the Grid [3], but also to decide how components will interact with each other by setting up appropriate HLA data and time management and to change nature of their interactions during simulation runtime. To make HLA components more user friendly for their developers, we have designed them in a way that they do not require full knowledge of the quite complicated HLA API.

The third requirement – ability to manipulate components across administrative domains – can be fulfilled by using a Grid infrastructure. As a Grid platform hosting HLA components we have chosen the H2O environment [4].

The main purpose of our approach is to facilitate joining simulations of different scale, so it 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 their own HLA application and want to run it almost unaltered efficiently using the Grid, we suggested using our previous work [5], 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.

In this paper we focus on a dynamic change of the component's time policy and on manipulating objects' exchange between simulation modules (which is

done by switching on/off publish-subscribe mechanism). The functionality of the system will be shown through an example of multiscale simulation of dense stellar systems. The paper is organized as follows: in Section 2 we outline related work, in Section 3 we describe the HLA component model and prototype of its implementation. Section 4 presents the idea of a Grid support system for such model. Section 5 presents an example multiscale simulation from which we have taken the simulation logic for two components used in our experiment as well as the results of the experiment. Summary and future plans are described in Section 6.

2 Related Work

Multiscale simulations are an important and an interesting field of research. Examples include multiphysics capillary growth [6] or modeling colloidal dynamics [7]. The vastly growing number of papers in this area shows the need for an environment that facilitates exchange of developed models between scientists working in that field and for a reusable component solution. Among component standards worth to be mentioned are: Common Component Architecture (CCA) [8] (with its implementations like XCAT [9] or MOCCA [10]), Corba Component Model [11] or Grid Component Model [12] (with its implementation ProActive [13]). However, none of this models provide advanced features for distributed multiscale simulations (in particular they do not support advanced time management mechanism). An important approach to using services and component technology to distributed simulations is described in [14] and is general on distributed simulations, without special focus on multiscale simulations systems. Another important component framework for simulations [15] is specifically designed for partial differential equations.

3 HLA Component Model

The HLA component model differs from popular component models (e.g. CCA [8]), where 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* as shown in the Fig. 1a). Instead, all federates within a federation are connected together and can communicate using HLA mechanisms like time, data, ownership management etc. This is illustrated in the Fig. 1b.

The difference between the component view proposed in this paper and the original HLA approach is that the particular behavior of the component and it's connections are defined and set by an external module on the user request. The main advantage of this approach over original HLA is that it facilitates the user to create federations from federates developed by others. The particular federation, in which a federate is going to take part, does not need to be defined by the federate developer, but can be created later – from outside – in the process of setting up a distributed simulation system. Therefore the presented approach increases reusability and composability of simulations.

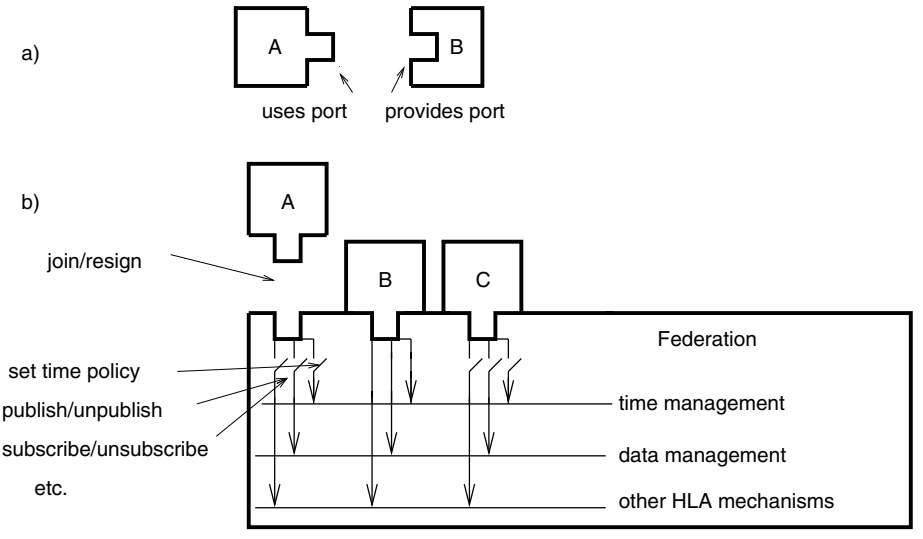

Fig. 1. CCA and HLA component models

Dynamic interactions in the HLA component model. 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 tuple space takes care of sending the appropriate data from the publisher to the subscriber (in most HLA implementations objects and interactions to be exchanged have to be specified before federation creation, but there exists an HLA implementation based on Grid Services [16] which does not have this limitation). HLA also includes advanced time management mechanisms that allows to connect federates with different internal time management. There is no single time for all simulations, but every simulation has its own internal time and some federates can regulate the time of the others (called constrained) if necessary. The communication between federates is done by exchanging time stamped objects and interactions.

In the HLA component model, as shown in the Fig. 1b, it is possible not only to dynamically join federate to a federation from outside [17], but also to steer the connections between federates. Therefore, it is possible to pose a request to one of the components to subscribe to the particular object class (from the objects declared in its description) and to another to publish this object class. In this way one can steer the data flow between simulations during runtime. It is also possible to set one federate to be constrained and another to be regulating. Section 5 provides an example that illustrates this.

HLA component implementation. As a Grid framework we have chosen the H2O [4] platform as it is lightweighted and enables dynamic remote deployment. The HLA component is implemented as a H2O pluglet. In our prototype we have implemented basic requests to start – stop the simulation and requests for federation management: join and resign described in more detail in [17]. In this

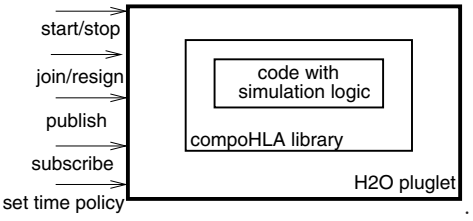

Fig. 2. Relationship between component's developer code (simulation logic), HLA RTI implementation and compoHLA library

paper we focus on chosen requests for time and data management: set time policy for regulating or constrained, subscribe and publish for object class. The set of requests can be easy extended according to HLA possibilities in the future. The component developer has to provide a simulation logic code which is connected with a pluglet by interfacing with the compoHLA library as shown in the Fig. 2.

The CompoHLA library introduces two classes with abstract methods that should be overridden by a component developer. One is the `CompoHLASimulator` class, from which the developer has to inherit and point to the main function starting simulation. There is also the `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 external user (who is going to chose this component to be connected to his simulation system). The developer has to specify how actual simulation data fits into HLA data objects that could possibly be exchanged with other federates. Also, a developer has to override `FederateAmbassador` class callbacks (there are used by RTI to communicate with developer code e.g. when receiving data from other federates) as in the original RTI federate. The simulation developer can also call methods of the `CompoHLAFederate` class which, in turn, uses the HLA `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).

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 the HLA component to be steered from outside (by external requests as described above).

4 CompoHLA – Environment Supporting HLA Component Model

Fig. 3 shows the proposed CompoHLA environment that supports setting and managing applications consisting of HLA components and consists of following elements:

HLA Component Description Repository – stores description of components – including information about data objects and interactions that the

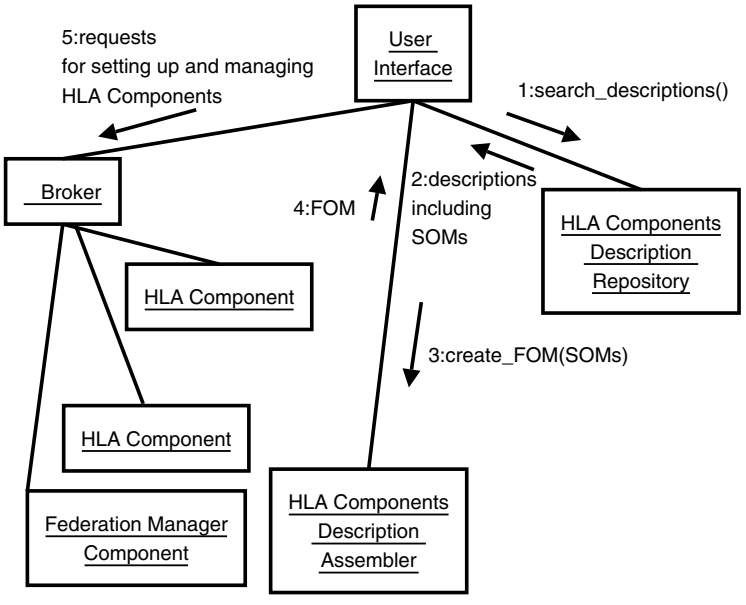

Fig. 3. Collaboration diagram illustrating interactions between basic elements of CompoHLA environment. Simulation Object Model (SOM) contains the description of objects and events that can be exchanged by a HLA component. Fedaration Object Model (FOM) consists of SOMs and describes a federation of HLA Components.

component can exchange with others (which is called Simulation Object Model – SOM), the type of time management that makes sense for this component and additional information that may be useful for 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.).

HLA Components Description Assembler – produces Federation Object Model (FOM) needed to start federation from given SOMs of components that will comprise simulation system.

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.

Federation Manager Component – manages whole federation on the component level and sets up connection with RTI coordination process for federations (`RTIExec` in HLA RTI DoD implementation, `rtig` in Certi RTI)

HLA Components – wrap actual functionality of federates into components. In this paper we particularly focus on this part of the system.

The collaboration between system elements is shown in the Fig. 3. A user can search the HLA Components Description Repository to find HLA components of interest to him. Then he can use the HLA Components Description Assembler to build a federation description (FOM) from the available components' descriptions (SOMs) and pass it to the Builder that sets up the federation from appropriate HLA components. The user than can dynamically change the nature of connections between components using a Builder that makes direct requests (described in this paper) to HLA components.

5 Experiments with an Example Application – Multiscale Multiphysics Scientific Environment (MUSE)

For the purposes of this research we have used simulation modules of different time scale taken from Multiscale Multiphysics Scientific Environment (MUSE) [18] for simulating dense stellar systems like globular clusters and galactic nuclei. However, the presented HLA component approach can be applied to modules of any other multiscale simulation system. Section 3 describes in datail how to wrap new simulation kernels into components.

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: radiative transfer, 3D gas and dust density distribution and stellar spectral energy distribution. In [2] we have shown that distribution of MUSE modules using HLA on the Grid can be beneficial for such applications. In particular, we have shown the usefulness of HLA advanced time management for this kind of simulations.

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 data are sent from macro scale to meso scale as star evolution data (change of star mass) is needed by *dynamics*. In that particular case, no data is needed from *dynamics* to *evolution*.

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 [1] offers many useful features there. The mechanism of *regulating* federate *(evolution)* that controls time flow in *constrained* federate *(dynamics)* is essential. It's main advantage is that it does not require to specify explicitly the period in which the constrained federate should stop and wait for the regulating federate. Instead, the maximal point in time which the constrained federate can reach at certain moment is calculated dynamically according to the position of the regulating federate on the time axis. HLA data management is the second important mechanism useful here. Used together with time management, it allows federates to exchange data objects and interactions with time stamps and assures that objects arrive in appropriate order at the appropriate time of the simulation. To achieve data from *evolution*, the *dynamics* federate subscribes to data object containing information about stars published by *evolution*.

This HLA component model allows these mechanisms to be accessible to the external user who sets up the simulation system from existing *dynamics* and *evolution* components created by someone else. This will allow users working on multiscale simulations to more easily exchange the models already created.

Performance Results. We have created two prototype HLA components for *dynamics* and *evolution* simulations and measured execution time of requests to them. The *dynamics* component was asked to set its time policy to be constrained and to subscribe to the star object class. The *evolution* component was asked to set its time policy to regulating and to publish the star object class. In our prototype, publication and subscription is done to the whole object class, but it is easy to extend it to support subscription and publication of a subset of class attributes. In our implementation we have used H2O v2.1 and HLA Certi implementation v3.2.4. Experiments were done on Dutch Gride DAS3 [19]. The RTI control process was run on a Grid node at the Amsterdam Free University (dual CPU, dual-core, 2.4 GHz AMD Opteron, 4GB RAM), the client was run at Leiden University (dual CPU, single-core, 2.6 GHz AMD Opteron, 4GB RAM). In the first experiment the *dynamics* component was run at University of Amsterdam (dual CPU, dual-core, 2.2 GHz AMD Opteron, 4GB RAM) and the *evolution* component at Delft (dual CPU, single-core, 2.4 GHz AMD Opteron, 4GB RAM). As the requests to *evolution* and *dynamics* were of different kind, in the second experiment we switched locations of *dynamics* (run at Delft) and *evolution* (run at Amsterdam) to compare time of all kinds of requests made to components residing on the same site. The network bandwith between Grid sites is 10 Gbps.

Table 1 shows results of experiments (average of 10 runs) in miliseconds. In both experiments, the time of requests to components residing on the Amsterdam site was slightly shorter then to these in Delft. This is probably because of a slightly different architecture of both sites (see above). It is worth notice that for both sites, the overhead of CompoHLA component layer is rather small – for all requests order of a few milliseconds. Time elapsed of the pure RTI execution depends on the particular HLA implementation. In [2] we have shown that the execution time of sequential and distributed version of MUSE (*evolution* and *dynamics* modules) are comparable and that synchronisation between those two modules does not produce much overhead. In [2] we have also discussed different types of time interactions that can appear between components in multiscale

Table 1. Time of HLA component request execution for *evolution* and *dynamics* components measured from MUSE

action	Amsterdam				Delft			
	whole request		RTI time		whole request		RTI time	
	avr, ms	σ	avr, ms	σ	avr, ms	σ	avr, ms	σ
set time constrained	4.5	0.5	0.1	0.02	7	0.3	0.1	0.02
set time regulating	5	0.3	0.1	0.02	7.5	0.5	0.1	0.03
publish	6	0.4	1	0.1	10	0.5	2	0.1
subscribe	6	0.4	1	0.1	10	0.4	3	0.1

simulation system. Although they are described on the example of MUSE, in general they can appear in applications from any other domain knowledge. We have shown that such systems can benefit from being distributed using HLA. Additionally, using HLA component approach presented in this paper will increase reusability and interoperability of multiscale modules.

6 Summary and Future Work

The main objective of the research presented in this paper was to offer scientists working with multiscale simulations a component model that facilitates joining elements of different scale and providing them with an environment supporting applications in that model. The novelty of proposed approach consists in integrating solutions from HLA, component technologies and the Grid to support multiscale applications. As any of the existing component models does not explicitly address requirements of multiscale simulations, we have presented the idea of a model based on HLA, which enables users to dynamically compose/decompose distributed simulations from multiscale components residing on the Grid. Using the proposed CompoHLA environment a user is able to decide how components will interact with each other (e.g. by setting up appropriate HLA subscription/publication and time management mechanism) and to change nature of their interactions during simulation runtime. 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 in the example of multiscale simulation of dense stellar system – the MUSE environment [18]. The results of experiments show that the execution time of requests are relatively low and that the component layer does not introduce much overhead.

In the future we plan to fully design and implement other modules of the presented support system. Also, we plan to extend the prototype of HLA Component to allow to insert plugins (e.g. build by physicist whom wants to join two already existing components together) that can scale output from one federate to be appropriate an input for another federate.

Acknowledgments. The authors wish to thank Maciej Malawski for discussions on component models and Simon Portegies Zwart for valuable discussions on MUSE. We also acknowledge the access to the DAS3 distributed Computer System. This research was partly funded by EU IST Project CoreGRID and the Polish State Committee for Scientific Research SPUB-M, through the BSIK project Virtual Laboratory for eScience and ACK CYFRONET AGH Grant No. 500–08.

References

1. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) (2004), `http://standards.ieee.org/catalog/olis/compsim.html`
2. Rycerz, K., Bubak, M., Sloot, P.M.A.: Using HLA and Grid for Distributed Multiscale Simulations. In: Parallel Processing and Applied Mathematics: 7th International Conference (PPAM 2007). LNCS, vol. 4967. Springer, Heidelberg (to appear, 2008)

3. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Open Grid Service Infrastructure WG, Global Grid Forum (June 2002)
4. Kurzyniec, D., Wrzosek, T., Drzewiecki, D., Sunderam, V.S.: Towards Self-Organizing Distributed Computing Frameworks: The H2O Approach. Parallel Processing Letters 13(2), 273–290 (2003)
5. Rycerz, K.: Grid-based HLA Simulation Support. PhD thesis, University of Amsterdam, Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Dr. M.T. Bubak (June 2006), http://dare.uva.nl/en/record/192213
6. Szczerba, D., Székely, G., Kurz, H.: A Multiphysics Model of Capillary Growth and Remodeling. In: Alexandrov, V.N., van Albada, G.D., Sloot, P.M.A., Dongarra, J. (eds.) ICCS 2006. LNCS, vol. 3992, pp. 86–93. Springer, Heidelberg (2006)
7. Dzwinel, W., Yuen, D., Boryczko, K.: Bridging diverse physical scales with the discrete-particle paradigm in modeling colloidal dynamics with mesoscopic features. Chemical Engineering Sci. 61, 2169–2185 (2006)
8. Armstrong, R., Kumfert, G., McInnes, L.C., Parker, S., Allan, B., Sottile, M., Epperly, T., Dahlgren, T.: The CCA component model for high-performance scientific computing. Concurr. Comput.: Pract. Exper. 18(2), 215–229 (2006)
9. Krishnan, S., Gannon, D.: XCAT3: A Framework for CCA Components as OGSA Services. In: Proc. Int. Workshop on High-Level Parallel Progr. Models and Supportive Environments (HIPS), Santa Fe, New Mexico, USA, April 2004, pp. 90–97 (2004)
10. Malawski, M., Kurzyniec, D., Sunderam, V.S.: MOCCA – Towards a Distributed CCA Framework for Metacomputing. In: 19th International Parallel and Distributed Processing Symposium (IPDPS 2005), CD-ROM / Abstracts Proceedings, Denver, CA, USA, 4-8 April (2005)
11. CORBA Component Model, v4.0 (2006), http://www.omg.org/technology/documents/formal/components.htm
12. Deliverable D.PM.02 – Proposals for a Grid Component Model (2006), http://www.coregrid.net
13. ProActive project homepage, http://www-sop.inria.fr/oasis/ProActive/
14. Chen, X., Cai, W., Turner, S.J., Wang, Y.: SOAr-DSGrid: Service-Oriented Architecture for Distributed Simulation on the Grid. In: Principles of Advanced and Distributed Simulation (PADS), pp. 65–73 (2006)
15. Parker, S.G.: A component-based architecture for parallel multi-physics PDE simulation. Future Generation Computer Systems 22, 204–216 (2006)
16. Pan, K., Turner, S.J., Cai, W., Li, Z.: A Service Oriented HLA RTI on the Grid. In: IEEE International Conference on Web Services, 9-13 July 2007, pp. 984–992 (2007)
17. Rycerz, K., Bubak, M., Sloot, P.M.A.: HLA Component Based Environment for Distributed Multiscale Simulations (submitted to Special Issue of Scientific Programming on Large-Scale Programming Tools and Environments)
18. MUSE Web page, http://muse.li/
19. The Distributed ASCI Supercomputer 3 web page, http://www.cs.vu.nl/das3