



UvA-DARE (Digital Academic Repository)

Grid-based HLA simulation support

Rycerz, K.J.

Publication date

2006

Document Version

Final published version

[Link to publication](#)

Citation for published version (APA):

Rycerz, K. J. (2006). *Grid-based HLA simulation support*. [Thesis, externally prepared, Universiteit van Amsterdam].

General rights

It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations

If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: <https://uba.uva.nl/en/contact>, or a letter to: Library of the University of Amsterdam, Secretariat, P.O. Box 19185, 1000 GD Amsterdam, The Netherlands. You will be contacted as soon as possible.

Grid-based HLA Simulation Support

Grid-based HLA Simulation Support

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Universiteit van Amsterdam
op gezag van de Rector Magnificus
prof. mr. P. F. van der Heijden
ten overstaan van een door het college voor promoties ingestelde
commissie, in het openbaar te verdedigen in de Aula der Universiteit
op dinsdag 13 juni 2006, te 10:00 uur

door

Katarzyna Julia Rycerz

geboren te Kraków, Polen

Promotiecommissie:

Promotor: prof. dr. P. M. A. Sloot

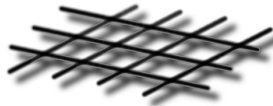
Co-promotor: dr. M. Bubak

Overige leden: prof. drs. M. Boasson
dr. A.G. Hoekstra
prof. dr. R.J. Meijer
dr. S.J. Turner
prof. dr. K. Zieliński

Faculteit: Faculteit der Natuurwetenschappen, Wiskunde en Informatica



Section Computational Science



Universiteit van Amsterdam



vl-e



The Work described in this thesis has been carried out in two institutions: at the Section Computational Science of the University van Amsterdam and Institute of Computer Science, AGH Krakow. It was financially supported by the University of Amsterdam and AGH Krakow.

This research is partly funded by the European Commission IST-2001-32243 Project CrossGrid (www.eu-crossgrid.org) and partially by Project IST-2002-004265 CoreGRID (www.coregrid.net). This work was also carried out in the context of the Virtual Laboratory for e-Science project (www.vl-e.nl). Part of this project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

Copyright © 2006 by Katarzyna Rycerz
Author contact: kzajac@agh.edu.pl

ISBN: 83-923723-0-1 (978-83-923723-0-1)

Typeset with \LaTeX 2_ε

Cover designed by: Krzysztof Radoszek

Published by: Nowy Projekt (<http://www.nowyprojekt.pl>)

To my parents

Contents

1	Research Area and Roadmap	13
1.1	Trends in distributed interactive simulations	13
1.1.1	Application push	14
1.1.2	Technology pull	16
1.2	Environments for distributed interactive simulation	19
1.2.1	Available systems supporting distributed simulations	19
1.2.2	Importance of High Level Architecture	19
1.3	Requirements of distributed interactive simulations	21
1.4	Aims and scope of the thesis	23
1.5	Research roadmap	24
1.6	Methodology	28
1.7	Organization of this thesis	28
2	Analysis of Approaches Supporting Distributed Interactive Simulations	31
2.1	Environments for distributed simulations and their runtime steering	32
2.1.1	Computational Steering Environment	32
2.1.2	Collaborative User Migration, User Library for Visualization and Steering	33
2.1.3	Visualization of Parallel Simulation Algorithms for Extended Research	34
2.1.4	Visualisation Interface Toolkit	36
2.1.5	Cactus Problem Solving Environment	37
2.1.6	Discover	39
2.1.7	TENT	40
2.1.8	High Level Architecture	41
2.1.9	JSIM	43

2.1.10	Interactive Simulation Systems Conductor	44
2.1.11	Amsterdam Parallel Simulation System	45
2.1.12	Virtual Radiology Explorer	46
2.1.13	Summary	47
2.2	High Performance Simulations in a Metacomputing Environment	49
2.2.1	Introduction	49
2.2.2	Synthetic Forces Express project	50
2.2.3	Airflow simulation	52
2.2.4	Numerical relativity (Cactus application)	52
2.2.5	Summary	53
2.3	Grid solutions for distributed interactive simulations	55
2.3.1	Evolution of Grid architecture	55
2.3.2	CrossGrid	59
2.3.3	Reality Grid	61
2.3.4	GEMSS	62
2.3.5	Virtual Laboratory	63
2.3.6	DSGrid	64
2.3.7	Summary	65
2.4	HLA in Grid and Web services environments	65
2.4.1	Towards interoperability of distributed interactive simulations	65
2.4.2	Fault tolerance	67
2.4.3	Peer to peer collaborative applications	68
2.4.4	Management of HLA-based simulations on the Grid	69
2.4.5	Summary	70
2.5	Summary and conclusions	71
3	Grid HLA Management System	73
3.1	Background	74
3.1.1	High Level Architecture	74
3.1.2	Service Oriented Architecture	76
3.1.3	Possibilities and limitations for HLA-based simulations on the Grid	78
3.2	Grid HLA Management System	79
3.2.1	Requirements	79
3.2.2	Functionality	80
3.2.3	Automatic HLA application setup within G-HLAM	82
3.2.4	Summary	84
3.3	Analysis of G-HLAM System	85
3.3.1	Petri Net	85
3.3.2	Using Petri Nets for deadlock detection	86
3.3.3	A Petri Net based model for the G-HLAM system	86
3.4	Summary	90

4	Interfacing Services	93
4.1	The role of HLA Interfacing Services	94
4.2	HLA-Speaking Service for a single process	95
4.2.1	Functionality	95
4.2.2	GridHLAController Library routines for a single process	96
4.2.3	User federate setup in <i>HLA-Speaking Service</i>	96
4.3	HLA Speaking Service for multiple processes	97
4.3.1	Functionality	97
4.3.2	GridHLAController Library routines for multiple federates	99
4.3.3	Application setup – component diagram	100
4.4	Evolution of <i>HLA-Speaking Service</i> – discussion	100
4.5	Interfacing RTI control process	102
4.6	Summary	103
5	Migration Services	105
5.1	Overview of possible migration approaches	106
5.2	Migration scenarios	107
5.2.1	Single process version	107
5.2.2	Multiple processes version	108
5.2.3	Discussion	109
5.3	Collaboration between the <i>HLA-Speaking Service</i> and user processes during migration	109
5.3.1	Single process version	110
5.3.2	Multiple processes version	110
5.4	Summary	112
6	Monitoring Services	115
6.1	Obtaining information from the infrastructure	116
6.1.1	Functionality of general <i>Benchmark Service</i>	117
6.1.2	<i>Benchmark Service</i> set for interactive simulation	118
6.2	Monitoring of HLA Grid application federates with OCM-G	119
6.2.1	Grid-enabled OMIS-Compliant Monitor	120
6.2.2	Monitoring HLA-based applications	121
6.3	Summary	123
7	Performance Results and Their Analysis	125
7.1	Migration overhead	126
7.1.1	Overhead of migration operations	126
7.1.2	Impact of migration on the performance of an interactive appli- cation	132
7.1.3	Conclusions	134
7.2	Benchmark results	134
7.2.1	Experiment description	134
7.2.2	Conclusions	135
7.3	Monitoring overhead	136

7.3.1	Experiment description	136
7.3.2	Conclusions	136
7.4	Summary	137
8	Application of G-HLAM to N-body simulation	139
8.1	Background	140
8.2	Description of the N -body application	141
8.2.1	Simulation details	141
8.2.2	Inter-module communication	142
8.2.3	Types of interactions with the user in a loop	143
8.3	N -body simulation in G-HLAM	144
8.3.1	Detailed structure	144
8.3.2	Results	145
8.4	Summary and conclusions	154
9	Application of G-HLAM to Vascular Reconstruction	157
9.1	Vascular reconstruction application	158
9.1.1	Background	158
9.1.2	Application modules	158
9.1.3	Scenario with a single user and a single simulation	160
9.1.4	Scenario with a single user and multiple simulations	161
9.1.5	Collaborative environment scenario	162
9.1.6	HLA-based communication	163
9.2	Using G-HLAM	164
9.3	Results	165
9.4	Summary and conclusions	173
10	Summary, Conclusions and Future Research	175
10.1	Summary	175
10.2	Conclusions	177
10.3	Future research directions	179
10.3.1	Possible extension of the G-HLAM functionality	180
10.3.2	Technological migration possibilities	181
A	Operations of HLA-Speaking Porttype	183
A.1	Version for a single process	183
A.2	Version for multiple processes	183
B	Grid HLA Controller Library	185
B.1	Initialization routines	185
B.1.1	A single process version	185
B.1.2	A multiple processes version	186
B.2	Migration routines	186
	References	189

CONTENTS

v

English Summary	205
Nederlandse samenvatting	209
Streszczenie po polsku	213
Publications	217
Acknowledgments	223

List of Figures

1.1	Sequence diagram illustrating batch applications – a user starts the application and gets the results when the job is finished.	15
1.2	Sequence diagram illustrating the one-way interaction model – the user first starts the application with parameter A, then stops it in the middle of execution and restarts with parameter B.	16
1.3	Sequence diagram illustrating the two way interaction model – the user is not only able to see the progress of the simulation when receiving its output online, but can also can steer the running application	17
1.4	The Grid HLA Management System is a middleware between distributed interactive simulations built over HLA and a Grid infrastructure	23
1.5	The roadmap of research – the issues and their solutions described in this thesis.	25
2.1	Architecture of the Computational Steering Environment [40] implemented as a set of processes - called satellites - that communicate via a central manager.	32
2.2	Functionality of Collaborative User Migration, User Library for Visualization and Steering [92] – it allows to add interactive steering and visualization to an existing parallel or serial program (task).	34
2.3	Architecture of the Visualization of Parallel simulation algorithms for Extended Research system [136] containing DUAL SERVER for two kinds of clients: simulation and visualisation.	35
2.4	Architecture of Visualisation Interface Toolkit [173] introduces a concept similar to SOA - a service announcement protocol (SEAP) server is used to register visualisations services that can then be found by simulations.	36
2.5	Connection between simulation and visualisation in Cactus [7] using stream driver for transferring HDF5 data.	38

2.6	A three-tier architecture of Discover [100] composed of detachable thin clients at the front-end, a peer-to-peer network of servers in the middle, and the Distributed Interactive Object Substrate (DIOS++) at the back-end.	39
2.7	Basic components of TENT [148] – a system for building and management of complex distributed simulations. Application modules (i.e. simulation engines, visualisation modules) are connected to the system using wrappers.	41
2.8	Basic components of High Level Architecture Runtime Infrastructure [78]. Elements of distributed simulation (federates) communicate via RTI using <i>libRTI</i> library. Federates are controlled by <i>RTIExec</i> and <i>FedExec</i> processes.	42
2.9	Architecture of JSIM [90], consisting of three layered groups of packages – the foundation layer, engine layer and environment layer.	43
2.10	Architecture of ISS-Conductor [193] consists of Module Agents for controlling application modules and Communication Agents for communication between Module Agents and actual modules.	44
2.11	APSYS simulation kernel provides an API for the user simulation code and communicates with the inter-process communication library (PVM or MPI) using asynchronous communication.	45
2.12	Architecture of Virtual Radiology Explorer [14] includes components for speech recognition, tracking of user movement and advanced visualization.	46
2.13	Architecture of SF–Express Application consists of nodes performing actual simulation (S), performing communication (R), providing interest management functions (I) and data servers (D).	51
2.14	First version of Globus architecture [70], consists of four layers: fabric, connectivity, resource and collective.	56
2.15	OGSI-based Grid architecture [61] introduced the concept of extending Web services to the transient and stateful Grid services with a notification mechanism. The aim of OGSA is to define a set of higher–level services built over OGSI.	57
2.16	WSRF-based Grid architecture [180] – instead of directly extending Web services, one has to model a Stateful Resource assigned to that service	58
2.17	The CrossGrid architecture [38] consisting of three layers – applications, tools and services.	60
2.18	Architecture of WEDS [37], designed to let scientists remotely deploy instances of a pre-existing codes across multiple resources and giving steering, visualisation and workflow functionality.	61
2.19	The GEMSS [17] middleware exposes applications installed on various Grid hosts as services which support a common set of methods for data staging, remote job management, error recovery and QoS support.	62

2.20	VL architecture enables researchers at different locations to work in an interactive way, as in any laboratory.	64
2.21	Architecture of web-enabled RTI introduces a Web service wrapper over existing RTI functionality.	67
2.22	Architecture of FT-RSS allows the individual configuration of FT mechanisms to be included in federates and federations.	68
2.23	Architecture of FederationGrid based on the concept of a fractal Grid, comprised of hierarchical HLA federations.	69
2.24	Architecture of a framework for large-scale distributed simulations with Grid services – simulation models are encapsulated in Grid services. Following dynamic discovery of RTIExec Service, the simulation is set up and runs using HLA RTI communication.	70
3.1	Exchanging data objects between HLA federates within an RTI tuple space. The arrows indicate the direction of control flow between the RTI and the applications.	75
3.2	The HLA object model – federates use libRTI (which communicates with the <i>RtiExec</i> , a <i>FedExec</i> , and other federates) to invoke HLA services. . .	76
3.3	Service Oriented Architecture have a mechanism enabling a consumer to discover a service provider under the context sought by the consumer.	77
3.4	Grid HLA Management System Architecture (G-HLAM) consists of services which control the whole HLA application and the services that should be installed on each HLA-enabled Grid site.	80
3.5	Type of messages in UML sequence diagram	82
3.6	UML sequence diagram – interactions between <i>Broker Service</i> , <i>Registry</i> , <i>HLA-Speaking Service</i> , <i>RTIExec Service</i> and <i>GridFTP server</i> during start up of HLA application.	83
3.7	The Petri net describing the G-HLAM system – simplified model showing how monitoring of an application triggers migration decision.	87
3.8	The Petri Net describing migration in G-HLAM. Federate 1 (<i>fed1</i>) represents the federates that will be moved during migration, while federate 2 (<i>fed2</i>) represents the federates that have joined the federation, but will not be moved.	89
4.1	The role of <i>HLA-Speaking Service</i> and <i>RTIExec Service</i> . Services are located in a management level – they are responsible for managing execution of the federates and the control RTIExec process.	94
4.2	Architecture of <i>HLA-Speaking Service</i> for a single federate process – the service provides the HLA-Speaking Porttype - with operations supporting user federate process startup and saving or restoring its state. . .	95
4.3	UML component diagram example	96
4.4	UML component diagram of <i>HLA-Speaking service</i> for a single process showing how elements of the service architecture cooperate together when initiating user federate process.	97

4.5	Conceptual view of an <i>HLA Speaking Service</i> for multiple federate processes. The service provides the HLA–Speaking Porttype - with operations supporting user federate processes startup and saving or restoring their state. Control federation is used to communicate the service with user processes.	99
4.6	UML component diagram illustrating setup of multiple federate processes by the <i>HLA Speaking Service</i> – describes how elements of the service architecture cooperate together during application setup.	101
4.7	Management of <i>RTIExec</i> endpoints – cooperation between <i>RTIExec Factory</i> , <i>RTIExec Service</i> and actual <i>RTIExec</i> process.	102
5.1	UML sequence diagram illustrating federate migration for a single federate scenario - cooperation between services of G-HLAM system and RTI.	107
5.2	UML sequence diagram illustrating federate migration for multiple federates scenario – cooperation between services of G-HLAM system and RTI.	108
5.3	UML collaboration diagram example.	110
5.4	UML collaboration diagram for the Migration library, the RTI library and the <i>HLA Speaking Service</i> layer while executing a saving request.	111
5.5	Multiple–process management algorithm	112
5.6	UML collaboration diagram for the Migration library, the RTI library and the <i>HLA–Speaking Service</i> layer for multiple processes.	113
6.1	UML collaboration diagram of <i>Benchmark Services</i> in the Grid services framework – interactions between services and actual benchmark federate processes.	117
6.2	Hierarchical architecture of the OCM-G monitoring system [12]. Local Monitors are used to connect the system to monitored processes. Tool gets monitoring data from Main Service Manager.	119
6.3	Architecture of OCM-G based monitoring system for HLA–based application. Wrappers are used to connect HLA legacy code with OCM-G.	121
6.4	Workflow (thin lines) and dataflow (thick lines) during setup of OCM-G based monitoring of HLA applications.	123
7.1	Time of saving federation state in Experiment 1 as a function of the number of receiving federates	128
7.2	Time of resigning from federation during migration in Experiment 1 as a function of the number of receiving federates	129
7.3	Time of rejoining to federation during migration in Experiment 1 as a function of the number of receiving federates	130
7.4	Time of restoring of federation state during migration in Experiment 1 as a function of the number of receiving federates	130
7.5	Time of migration in Experiment 1 as a function of the number of receiving federates	132

7.6	Impact of migration on response time	134
8.1	The N -body application architecture – The simulation processes data from a database and sends its output to the visualization (thick arrows). The user can interact with the simulation by changing its parameters during runtime (thin arrows).	141
8.2	Communication between the user, the simulation and the visualization: at each timestep the simulation sends output (snapshot) to the visualisation/interaction module, which is then forwarded to the user interface. The user can react according to his needs by invoking steering commands (pause,change parameters, resume).	143
8.3	Interaction without changes to the simulation – the user pauses and resumes sending output, but does not make any changes to simulation state.	144
8.4	Interaction with changes to the simulation – the user makes changes to simulation state, so the rollback of the simulation that managed to advance in the background is needed.	144
8.5	Application with G-HLAM – visualization/interaction federate communicates with the MPI root process of the simulation federate using the HLA RTI. Additionally, HLA control federation is used to interface MPI simulation processes on the Grid site to the <i>HLA-Speaking Service</i> and G-HLAM.	145
8.6	Overhead of resigning from the control federation	147
8.7	Resigning time as a function of the size of the federation	149
8.8	Overhead of joining control federation	150
8.9	Total migration time as a function of the number of migrated MPI federates	151
8.10	Migration impact on execution time for a scenario of <i>passive</i> interaction with the N -body simulation – migration during passive examination	152
8.11	Migration impact on execution time for the scenario of <i>active</i> interaction with the N -body simulation.	153
8.12	Migration impact on execution time for the scenario of <i>active</i> interaction with the N -body simulation – migration during active examination.	154
9.1	Vascular reconstruction application scenario with a single user and a single simulation consisting of four modules: data analysis, editing tool, simulation and visualisation.	160
9.2	Vascular reconstruction application scenario with a single user and multiple simulations – the user uses a 3D editing tool to create different meshes with different bypasses and run the simulation simultaneously.	161
9.3	Vascular reconstruction application scenario with multiple user and multiple simulations. Each user starts a number of simulations and can own (control) each of them. The user can also interact with simulations he does not own.	162

9.4	G-HLAM for the medical application – example for three simulations and two users. Application federation connects all simulations and visualisations. On each site, there is a control federation that interfaces simulation to G-HLAM.	165
9.5	Time of saving application federation state as a function of the number of modules in the collaborative environment	168
9.6	Time of resigning from the application federation during migration in Experiment 1 as a function of the number of modules in the collaborative environment	168
9.7	Time of rejoining the federation during migration as a function of the number of modules in the collaborative environment	169
9.8	Time of restoring federation state during migration as a function of the number of modules in the collaborative environment	170
9.9	Total migration time as a function of the number of modules in collaborative environment federates	171
9.10	Impact of migration on simulation performance within collaborative environment	172
B.1	Sample RTI Federate simulation loop	188

List of Tables

2.1	Main features of the interactive simulation and visualization environments	49
2.2	Summary of the techniques used to redesign HPC simulations for the Grid	54
2.3	Comparison of the interactive grid computing projects	66
2.4	Different aspects of applying Grid to modeling and simulation	71
3.1	Limitations and possibilities of HLA and Grid for interactive simulations	78
3.2	Summary of requirements and their solutions in G-HLAM	84
3.3	Four-bit state of a federate	88
4.1	Comparison of the two versions of <i>HLA-Speaking Service</i>	100
7.1	Grid testbed infrastructure for Experiment 1	127
7.2	Grid testbed infrastructure for Experiment 2	132
7.3	Benchmark results	135
7.4	Testbed infrastructure on Grid	136
7.5	Overhead of OCM-G	137
8.1	Grid testbed infrastructure.	145
8.2	Coefficients of approximating function $Ax^2 + Bx + C$ of resigning time for various initial sizes of the federation	148
8.3	Execution time of various migration stages independent of the number of MPI processes	150
9.1	Grid testbed infrastructure	166
9.2	Execution time of various migration stages independent of the number of modules in the collaborative environment	171

10.1 Summary of problems and their solutions in G-HLAM	177
--	-----

List of Acronims

APC	Application Client Machine
API	Application Programmer Interface
APSYS	Amsterdam Parallel Simulation System
ASCI	Advanced School for Computing and Imaging
ASP	Application Service Provider
BOM	Base Object Model
CAS	Community Authorization Service
CFD	Computational Fluid Dynamics
CMF	Collaborative Metaprogramming Framework
COTS	Commercial Off-The-Shelf
CSE	Computational Steering Environment
CSM	Computation Structural Mechanics
CT	Computer Tomography
CUMULVS	Collaborative User Migration, User Library for Visualization and Steering
DAS2	Distributed ASCI Supercomputer
DDM	Data Distribution Management
DIS	Distributed Interactive Simulation
DM	Data Management
DMS	Distribute Manufacturing Simulation
DoD	Department of Defense
DRM	Distributed Resource Management system
DRMS	Decentralized Resource Management System
EDA	Event Driven Architecture
EDG	European DataGrid
EZ	Economic Affairs
FATS	Firearm Training Systems

GARA	General-purpose Architecture for Reservation and Allocation
GHCL	Grid HLA Controller library
GIIS	Grid Index Information Service
GIS	Grid Information Services
GRAM	Globus Resource Allocation Manager
GRIS	Grid Resource Information Service
GSAP	Grid Service Access Point
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
GVK	Grid Visualisation Kernel
GVT	Global Virtual Time
HEP	High Energy Physics
HLA	High Level Architecture
HPC	High Performance Computing
HPF	High Performance Fortran
HTC	High Throughput Computing
ISS-Conductor	Interactive Simulation Systems Conductor
JADE	Java Agent DEvelopment Framework
JIMS	JMX infrastructure Monitoring System
JMX	Java Management eXtension
JNI	Java Native Interface
JSIM	Java-based Simulation and Animation Environment
LBM	Lattice Boltzmann method
LM	Local monitor
LP	Logical Process
LRC	Local Replica Catalogue
LVT	Local Virtual Time
MAGNETAR	Metaprogrammable AGent NETwork ARchitecture
MD	Migrating Desktop
MDA	Model Driven Architecture
MDS	Meta Directory Service
MOM	Management Object Model
MPI	Message Passing Interface
NT	Network Transfer facility
OCM-G	Grid-enabled OMIS-compliant Monitor
OGSA	Open Grid Service Architecture
OGSI	Open Grid Service Infrastructure
OMIS	On-line Monitoring Interface Specification
OMT	Object Model Template
OO	Object Oriented
PDE	Partial Differential Equation
PGO	Parametrized Graphics Object
PPC	Performance Prediction Component

PVM	Parallel Virtual Machine
QoS	Quality of Service
RAS	Roaming Access Server
RDC	Remote Display Client
RDS	Remote Display Server
RFT	Reliable File Transfer
RLI	Replica Location Index
RMC	Replica Metadata Catalogue
RPC	Remote Procedure Call
RSL	Resource Specification Language
RTI	Runtime Infrastructure
SA	Software Agent
SEAP	Service Announcement Protocol
SM	Service Manager
SMod	Simulation Module
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SPEEDES	Synchronous Parallel Environment for Emulation and Discrete-Event Simulation
SP	Synchronization Point
UAV	Unmanned Aerial Vehicle
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
VE	Virtual Environment
VIMCO	Virtual Lab Information Management for Cooperation
VIPER	Visualization of Parallel simulation algorithms for Extended Research
ViSE	The Virtual Simulation and Exploration Environment
VISIT	Visualisation Interface Toolkit
VL	Virtual Laboratory
VNC	Virtual Network Computing
VO	Virtual Organisation
VR	Virtual Reality
VRE	Virtual Reality Explorer
Vtk	Visualisation Toolkit
WAN	Wide Area Network
WEDS	WSRF-based Environment for Distributed Simulation
WS	Web Services
WSDL	Web Service Description Language
WSRF	WS-Resource Framework
XML	eXtensible Modeling Language
XMSF	eXtensible Modeling Simulation Framework

Research Area and Roadmap

The subject of this thesis is the concept and development of a system for the management of High Level Architecture–based distributed simulations running on the Grid. In this Chapter we present an overview of the current trends in distributed simulations, analyze resulting requirements, present motivation for applying the HLA and outline the goals of the thesis, its scientific contribution and methodology. Our purpose is to explain the motivation and roadmap for our research.

1.1 Trends in distributed interactive simulations

Distributed simulations are an interesting field of research in modern computer science, being developed by various communities [63] – scientific (high performance computing) [95], defense [5], business (e.g. manufacturing [110]; transportation [153]; simulations for real–time planning, scheduling and control [50, 19]), and the Internet gaming industry [105]. While the high performance computing community is mainly concerned with reducing execution time of distributed simulations – i.e. by improving synchronization algorithms (so-called time management [80]), the defense and business communities are concerned with accessing distributed data and integrating separate training simulations in order to facilitate interoperability and software reuse [165, 110]. Last but not least, a track of research and development efforts has arisen in the area of distributed multiuser gaming where there is a need for scalable real-time interactive systems [105, 32]. In this field, the distributed simulation technology may have the greatest economic and social impact.

As new technologies become available we need to identify their potential to support these kinds of applications. In some cases new technologies are developed in response to specific needs (application push), in others the technologies are developed first and suitable applications are then sought (technology pull). Neither technology pull nor application push is superior with respect to one another. There is interaction between

user needs and technology in the innovative process - as technology progresses, the users see greater possibilities, so their demands change [138]. In this Section, the potential for applying distributed simulations is discussed.

1.1.1 Application push

The parallel and distributed simulation field of research is very broad and covers various types of applications. *Parallel* simulations are usually thought of as applications executed on a tightly-coupled supercomputer or cluster, where the main reason for distributing execution is to reduce the time needed to execute the simulation or to run much larger problems in the same amount of time.

The term *distributed* simulation refers to a geographically distributed system which is often integrated from several different simulators into a single simulation environment.

The areas of *parallel* and *distributed* simulations are overlapping [42] as the large-scale distributed computing systems provide possibilities for the exploitation of large-scale parallelism. This field of research includes various application types of which the most representative are [138]:

- model distribution – includes distribution of the execution of a one large simulation model [64] or many separate models on different computers [196],
- simulation distribution – includes replication of the same simulation [9] or distribution of multiple simulation scenarios [189],
- data management – includes linking models to remote databases [93] or real-time systems [46],
- collaborative environments - include simulations that can be interactively steered by many users [111], models or simulation software that can be shared [177], remote meetings between modelers and users during model development or use [164] and searching and downloading components for building models [127].

Two widely-used architectures for distributed simulation are the *client-server* and the *peer-to-peer* approaches. The client-server approach involves executing the distributed simulation on one or more server computers, where the simulation computation is executed and is available for clients (e.g. users). This approach is typically used in distributed simulations for multiplayer gaming. Centralized management of simulation computation greatly simplifies the management of the distributed simulation system itself and facilitates its monitoring, e.g., to detect cheating. Peer-to-peer systems have no such servers, and the simulation is distributed across many machines, often interconnected by a wide area network. This approach is often applied in distributed simulations used for defense. The advantage of simulations distribution is the fact that the physical location of required personnel and/or resources (e.g., databases, computational power or specialized hardware) which are usually also distributed does not need to be changed.

Distributed simulations can be *interactive*. In general, *interactivity* is the dialog that occurs between a human being and a computer program [179]. On the other hand, programs that run without an immediate user involvement are usually called *batch* or *background* programs. The idea of batch applications is shown in Fig. 1.1,

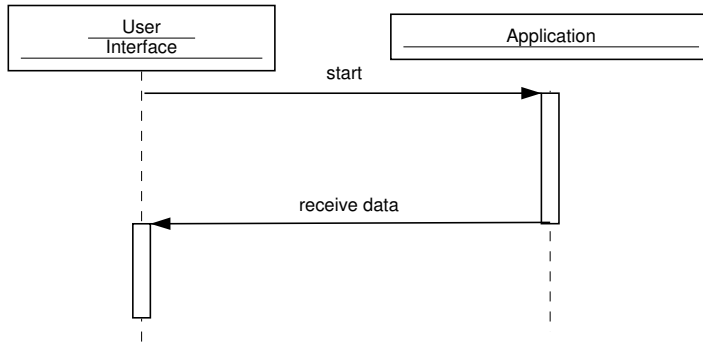


Figure 1.1: Sequence diagram illustrating batch applications – a user starts the application and gets the results when the job is finished.

where a user starts the application and gets the results when the job is finished.

In a *one-way* interaction model, a user can see the output of the simulation while it is running and stop it any time the partial results seem unsatisfactory (e.g. the result seems not to converge). This is shown in the sequence diagram in the Fig. 1.2, where the user interface acting on the user's behalf starts the simulation. Immediately after starting, the application sends data to the user interface. This enables the user to stop the simulation anytime he finds the partial results uninteresting. This approach differs from batch processing, where the user has to wait for the simulation to finish in order to see the results. This model is suitable for parameter study problems [2, 99], where there is no need to change parameters of the simulation while it is running, but only to check the output for specific parameters. This is illustrated in Fig. 1.2, where the user first starts the application with parameter A, then stops it in the middle of execution and restarts with parameter B.

The *two-way* interaction model is shown in Fig. 1.3, where the user is not only able to see the progress of the simulation when receiving its output online, but can also can steer the running application, either providing some input data online as requested by the application itself or by changing its parameters during runtime. This model is sufficient for runtime steering simulations, where the parameter space is not known before the applications starts and therefore a parameter study approach cannot be taken. This kinds of simulations are also referred to as "person in the loop" since they require a user in the processing loop. For example in blood flow simulation based on the Lattice-Boltzmann method [155], which is a particle-based approach to simulating fluid flows [10], the user can modify the conditions of the blood flow e.g. by inserting or changing positions of bypasses.

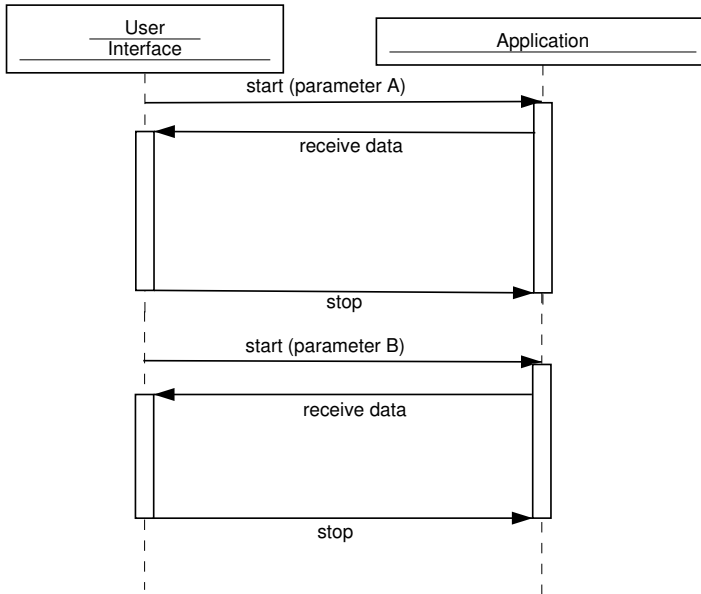


Figure 1.2: Sequence diagram illustrating the one-way interaction model – the user first starts the application with parameter A, then stops it in the middle of execution and restarts with parameter B.

In many aspects *Collaborative environments* [21] have much in common with distributed interactive simulations. An example is the Virtual Laboratory for e-Science project [172] that provides a framework for groups of scientists, engineers and scientific organizations that interact and cooperate with each other towards the achievement of a common experiment. Also, *multiplayer games* [32, 105]. are a very good examples of collaborative environments as they support a great amount of players interacting in real time, simultaneously, in a persistent-state virtual world. Collaborative environments can be divided into different groups with respect to their requirements of type of interaction. For instance, when taking games as an example, first-person shooters require fast response times to player commands. Role-playing games often tolerate command latency or temporary inconsistencies. Real-time strategy games can tolerate rather large command latencies but, in turn, require support for updating each game client on hundreds of dynamic game objects at each tick of the simulation clock.

1.1.2 Technology pull

Application development is also determined by the available technology. Below, we briefly present modern technologies that currently influence the area of distributed simulations. A detailed description of those technologies can be found in Section 2.3.1

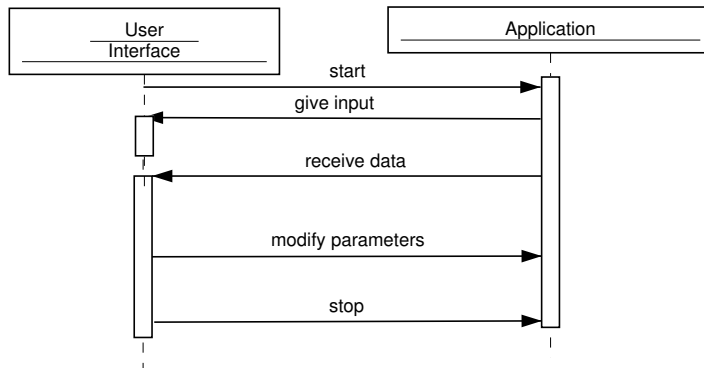


Figure 1.3: Sequence diagram illustrating the two way interaction model – the user is not only able to see the progress of the simulation when receiving its output online, but can also can steer the running application

and Section 3.1.2.

Web services

The Web services [178] concept aims at achieving interoperability between different technologies. Web services provide a standard means of interoperating between different software applications running on a variety of platforms and/or frameworks. Web services are characterized by their great interoperability and extensibility thanks to the use of the eXtensible Modeling Language (XML) [184], and they can then be combined in a loosely coupled way in order to perform complex operations. Programs providing simple services can interact with one another in order to deliver sophisticated added-value services. In order to achieve interoperability, Web services expose their interfaces described in an XML-based language called the Web Service Description Language (WSDL) [181] published in a Universal Description, Discovery and Integration (UDDI)[168] registry, so they can be found by clients. The main protocol defining the kind of communication to a Web service is the Simple Object Access Protocol (SOAP)[157]. Web services are stateless entities, so the results of any operation invoked remain the same every time for the same arguments.

Grid

The Grid is one of the most important concepts which strongly influence the area of scientific distributed computing. Ian Foster [58] presented a checkpoint list defining the Grid as a system that coordinates resources which are not subject to centralized control, but where coordination should be performed by means of standard, open, general-purpose protocols and should deliver nontrivial qualities of service.

By looking at ongoing experiments in this field one can distinguish the following main profiles [42]:

- Computational Grids – access to a large-scale high-performance computing service, with transparent access to the underlying parallel and distributed servers [154].
- Scientific data Grids – access to large scientific data sets, with optimized data transfers and interactions for data processing and manipulation [45],
- Virtual organizations and collaborative virtual spaces – access to virtual environments for resource sharing, online interaction and collaborative virtual environments [3],
- Information, knowledge and semantic Grids – large and geographically-distributed information repositories available for searching and data mining as well as for intelligent knowledge management and decision support [150].

The interest in Grid is growing as network speed is increasing more than computer speed [160]. One of the most important Grid technologies is the Globus Toolkit [70]. The first version of Globus included a tool for job submission called Grid Resource Allocation Management (GRAM), efficient file transfer (GridFTP), Grid Security Infrastructure (GSI) and Metadirectory Service (MDS). Basing on experience from the first version of Globus, the Grid community has undertaken an effort to define standard "InterGrid" protocols and to achieve interoperability between distributed systems. This idea was realized by extending the Web services concept with a state and possibility of dynamic creation. The Open Grid Services Infrastructure (OGSI) and the Open Grid Services Architecture (OGSA) [61] are among the results of this push. In OGSI the primary purposes are to manage the creation and termination of stateful services. The main focus is on the definition of abstract interfaces that allow services to cooperate without concern for the actual protocols they use. Basing on OGSI experience, its authors proposed the WS-Resource Framework (WSRF) which aims at converging the concepts of Grid and Web services. Both OGSI and WSRF are concerned with how to manipulate stateful resources through a Web services interface. The difference is in modeling – OGSI encapsulates service state in a Grid service which extends the concept of Web services, while WSRF uses a classical Web service as an interface to a stateful WS-Resource [180].

Apart from work of the Globus Community, there are also other approaches to Grid computing. The Grid also can be built using pure Web services or a Components Environment e.g. the Common Component Architecture [31]. CoreGRID [36] is a sample project aiming at building the Grid from components.

The global standardization effort for grid computing is being done by a Global Grid Forum [69] community of users, developers and vendors.

1.2 Environments for distributed interactive simulation

1.2.1 Available systems supporting distributed simulations

There are many system supporting development and execution of distributed simulations. Environments such as CSE [40], CUMULVS [92], VISIT [173], VIPER [136] CACTUS [7], and ISS-Conductor [193] allow for efficient runtime steering of simulations. CSE and VIPER support steering of single simulations through many visualizations using TCP; CUMULVS allows for building and steering parallel simulations by means of PVM, VIPER supports steering parallel simulations using remote procedure call (RPC), CACTUS is an advanced framework for parallel simulations written in various standards (MPI, PVM, Corba) and ISS-Conductor is an agent system supporting the development and execution of interactive simulations built over HLA.

JSIM [90], which is a representative system for Web-based simulation, allows the development of simulation models and their subsequent execution in the Java Beans technology.

The HLA [78] standard allows not only for efficient distribution of simulation and its steering, but also provides time management support. Additionally, HLA enables composing simulation systems from components with different time management (event-driven and time-driven). A more detailed description of those systems can be found in Chapter 2 (Section 2.1). For each system, we analyze the type of support for distributed simulations and the amount of effort that has to be invested in order to take advantage of the Grid technology.

1.2.2 Importance of High Level Architecture

After analyzing the various environments mentioned above (described in more detail in the Section 2.1), we decided to choose the High Level Architecture (HLA)[78]. Our main motivation was that HLA is an IEEE standard, well recognized in the area of distributed simulations and offers all necessary functionality for simulation developers such as simulation scalability, interoperability, and support for time and data distribution management. Also, to our best knowledge, it is the only standard that has ability to connect simulations with different time management (i.e time-driven or event-driven) in one system. Many companies are currently working on more scalable and efficient implementations of the standard [140]. There is also an open source version released recently [123]. HLA can be used for any kind of simulation systems. Although it originates from defense technology, there is a growing interest from non-military areas like manufacturing, transportation and gaming industries. The more detailed motivation for HLA and its comparison with other systems is described in Section 2.1.13.

Examples of HAL-based applications can be divided into the following groups:

Scientific simulations allows scientists to build agent-based frameworks for supporting interactive systems like ISS-Conductor, used for surgery planning [192] or

the combination of HLA with Java Agent DEvelopment Framework (JADE) to build Distributed Virtual Environments [176]. Another example is simulation of thin-film production [95].

Battlefield simulations link different types of forces at multiple physical locations to create a realistic and complex virtual world. This kind of applications are the primary purpose of HLA which was designed by the U.S. Department of Defense. Examples are:

- trainers of U.S. Special Operations Forces including Navigation/Fire Control Officer Testbed and Combat simulators [5]
- trainer of Air Force Research Laboratory – a four-ship F-16 simulation facility, networked via Distributed Interactive Simulation (DIS) and/or Native high-level architecture (HLA) technologies [6],
- CSC Advanced Marine’s ship maneuvering and navigation simulation software (Virtual Ship) [39],
- Firearm Training Systems (FATS) battlefield simulations [55]. FATS has fielded over 750 training systems that include HLA capability [55],
- the The U.S. Air Force Overseer UAV/UGV Control Interface project for graphical control of swarms of Unmanned Aerial Vehicles (UAVs) using the OpenSkies technology [125] – a scalable architecture for massive multiplayer networking built over HLA developed by the Cybernet Systems Corporation [43],
- military tactical trainers developed by the Mak company [106] allow the student to explore current and future combat operations, practice tactical decision making and better understand the impact of logistics support on tactical operations,
- a virtual reality laboratory, war gaming capability and battle space software for military training and testing developed by SMARTlab which serves as the modeling and simulation center for Honeywell Defense and Space Electronic Systems in Albuquerque, New Mexico [156].

Simulations in telecommunication– an example is a large scale cluster of distributed satellites that collaboratively share dispersed data assets [97].

Supply chain simulations manage material and information flow, from manufacturers through distributors to customers. Examples:

- SJUL – a distributed simulation used in the training of medical transport controllers developed by Pitch AB in cooperation with UhC in Skövde [153],
- Dukat – also used in the training of transport controllers and developed by Pitch AB in cooperation with UhC in Skövde [52],
- KLOTS – a system that supports traffic safety planning [91].

Multiplayer internet games that build massive multiplayer (10,000) virtual worlds. Examples:

- games [105] built using FederationX [57] – an HLA-based implementation of an architecture for the construction of large-scale distributed agent-based applications and supporting the Collaborative Metaprogramming Framework (CMF) [56]:
 - ExForce - a role-playing strategy game,
 - Nebula - a game network for board games and puzzles,
 - Pulsar - an online stock tracking and trading system,
 - Fortuna - a live casino embedded in a virtual world.

These games are developed by the Magnetar Games corporation [105].

- games using the OpenSkies technology [125]:
 - Engima: Rising Tide – the massively multiplayer component of Tesseract Games’ naval combat simulation [54],
 - Starfleet Command II– the multiplayer component of Taldren’s tactical space combat simulation [159].

HLA is used for many applications – its additional and important feature is that the local time management mechanism of one simulation component (federate) is not visible to other federates. Hence, all forms of time management (time-driven, event-driven, parallel discrete event, real-time-driven) may be linked together. HLA separates the communication infrastructure from the actual simulation. Additionally, it introduces a uniform way of describing events and objects being exchanged between federates.

All of these features support interoperability between various simulations. This feature can be used to build systems for simulation integration – i.e. Distribute Manufacturing Simulation (DMS) system [110] for integration of various commercial off-the-shelf (COTS) manufacturing software. Furthermore, Taylor and Turner [165] have shown that the HLA-based approach to interoperability between COTS packages performs better than other solutions.

It is also worth noticing that in all the listed HLA-based simulations interactivity plays an important role.

1.3 Requirements of distributed interactive simulations

In this thesis we focus on the class of simulations characterized by two main features:

- A simulation can be parallel or distributed and it is a part of a distributed application together with its visualization and interaction modules that can also be distributed.
- A human interacts with the simulation during runtime. This kind of simulation is also called a "human in the loop" simulation, because the parameters cannot be determined before the application starts (as in the case of parameter study applications), but have to be entered by the user in a processing loop during actual execution.

Basing on the characteristics of simulations described in Section 1.1.1, we have defined the following requirements:

1. synchronization (time management) for event-driven or time-driven simulations – every component of the distributed simulation should be able to control the execution time of other components,
2. ability to connect simulations with different time management in one system – i.e time-driven and event-driven simulations should be able to form components of one distributed simulation if necessary,
3. efficient and convenient data distribution management – enabling the exchange of data and/or events between distributed simulation components,
4. scalability – the distributed simulation should scale along with the number of its components,
5. access to (often specific) distributed resources e.g. databases, computational power, visualization devices.
6. a certain level of quality in the execution environment that allows for near-real-time communication between distributed components. This requirement follows the High Performance Computing (HPC) nature of distributed interactive simulations,
7. support for legacy simulations.

There is a gap between the requirements of interactive distributed simulations described in this Section and the available technological solutions described in Section 1.1.2. Some of these requirements (synchronization management, data distribution support, scalability and the ability to connect time- and even-driven simulations into one system) can be fulfilled by existing systems supporting such simulations (mentioned in Section 1.2 and described in more detail in the next Chapter). At the time of writing this thesis, up to our best knowledge, HLA is the only system that supports all requirements 1-4. Additionally, Grid solutions are a promising approach to the requirement of accessing distributed resources (requirement 5). However, there still remains the issue of execution environment efficiency (requirements 6) not only for newly created applications, but also for legacy systems (requirements 7).

1.4 Aims and scope of the thesis

In this thesis we focus on the issue of efficient usage of modern computing solutions for the purposes of HLA-based distributed interactive simulations.

In our research the central hypothesis can be formulated as: *interactive distributed simulations can significantly benefit from the Grid environment and there exist solution for achieving their effective execution by filling the gap between interactive distributed simulations and the Grid infrastructure.* We describe the scientific and technological issues involved in the design of an environment for the purpose of effective execution of distributed interactive simulations.

The scientific contributions which support the central hypothesis are:

- analysis of requirements of distributed interactive simulations,
- evaluation of existing solutions for those requirements,
- analysis of possible technologies useful with regard to these requirements,
- proof that HLA is an appropriate standard for such simulations, when there is a need to run them on the Grid,
- proof that the Grid environment can be beneficial for such simulations and provides them with new opportunities not previously available. This contribution is realized through:
 - analyzing possibilities offered by the Grid and defining requirements for their efficient usage,
 - design, implementation and feasibility study of a solution for efficient usage of the Grid for distributed interactive simulations.

Distributed Interactive Simulation	Application level
High Level Architecture (HLA)	
Grid HLA Management System	Middleware level
Grid Infrastructure (OGSI, Gram, GSI, GridFTP)	

Figure 1.4: The Grid HLA Management System is a middleware between distributed interactive simulations built over HLA and a Grid infrastructure

The main result of this thesis is the development of the Grid HLA Management System (G-HLAM) which supports efficient execution of HLA-based distributed interactive simulations in a Grid environment (Fig. 1.4). We will show that G-HLAM

fills the gap between HLA-based distributed interactive simulations and the Grid infrastructure in an optimal way.

1.5 Research roadmap

In this Section we present a roadmap of the research described in this thesis, (see Fig. 1.5). We have started with analysis of existing technologies and approaches that offer very promising approaches to distributed simulations. The details of the analysis are presented in Chapter 2. Basing on it, we have chosen HLA as a support for distributed simulations that can be run on a Grid. On the one hand, HLA, an important standard, offers advanced functionality for simulations: support for geographically-distributed components, time management and efficient data distribution management. On the other hand, Grid technology extends the set of possible distributed resources that can be used and facilitates access to them. The Grid can be useful for those simulations that consist of components with different functionalities, requiring various, often geographically-dispersed resources such as databases, computational power or visualization devices. Additionally, Grid services together with the Web services concept introduce the definition of abstract interfaces that allow services to cooperate without too much concern for the actual protocols being used and their internal technology. The advantages and disadvantages of HLA as well as of Grid technology are depicted on the top of Fig 1.5. The natural question in our research is how to join both concepts to allow better and more convenient execution of simulations.

As described in [190], we think that the most coarse-grain approach is to address the issue of the discovery of HLA federate components. This discovery mechanism in HLA was one of our initial points of focus: HLA Runtime Infrastructure (RTI) uses either a multicast discovery protocol, which does not scale well on large Wide Area Network (WAN) Grid distributed environments, or relies on the specification of an RTIExec's (which is a control process of RTI) endpoint in a configuration file. We have found this approach not very convenient, because the endpoint has to be known in advance and manually inserted into each file system the HLA federate has access to. We have decided to solve this problem by proposing an RTIExec service which handles the RTIExec coordination process of RTI. The Broker Service learns about existing RTIExec services from the Registry Service and sets up the whole application as described in Chapter 3 (Section 3.2.3) and [191].

Next, we investigated how to turn user application federates into Grid services. The solution of making each federate into a separate Grid service depending on its functionality would make sense, if the developed federates were quite general and there was a chance that they would also be needed by other applications. However, achieving reusability of federates that encapsulate simulation models is not a trivial problem and although touched upon in [194, 185, 41], still remains to be solved. This solution would also make it impossible to enable Grid use for HLA legacy applications, which is one of our aims.

Finally, we have decided to build a Grid service (called an *HLA-Speaking Service*)

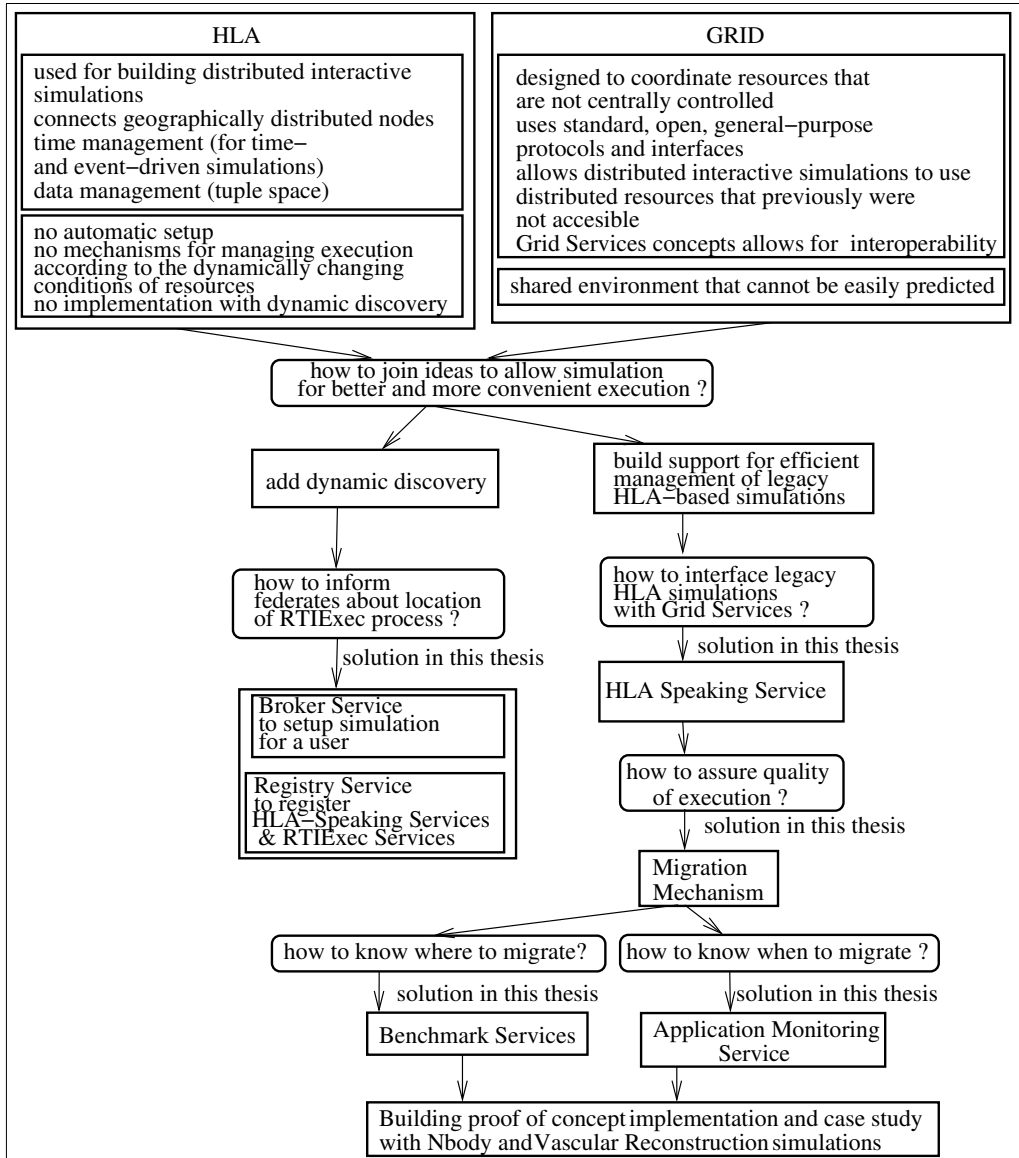


Figure 1.5: The roadmap of research – the issues and their solutions described in this thesis.

that is general for all HLA federates responsible for running federate code on the site where it resides. The service factory should know about various implementations of the HLA standard on its site and expose this info, so the entity that sets up the application (in our case the *Broker Service*) can find the appropriate version and ask the factory service to create an appropriate HLA-Speaking Service.

The main problems that arise when developing the HLA-Speaking Service were: interfacing service to federates, building an easy-to-use API for interaction of the user code with the service, managing multiple processes and using the whole Grid site efficiently. The first prototype of the service designed for a single process was described in [191, 7] and a version for multiple processes was described in [142]. Our solution can be found in Chapter 4 of this thesis.

Our goal was to provide a certain quality of the Grid environment to allow HLA-based simulations to be executed efficiently. It is often very difficult to predict the performance of a task in a shared Grid environment. Although various QoS mechanisms such as guaranteed services [73] or differentiated services [47] are technically possible, they are often not present in Grid environments and require support from network devices as well as additional effort to set them up, which can limit access to some of the resources. The General-purpose Architecture for Reservation and Allocation (GARA) [67] together with its GARNET testbed [68] is a project that integrates Quality of Service (QoS) into Globus [70]. Unfortunately, although proposed, GARA is not present in current Grid solutions. If no reservation or priority mechanisms are available, HPC applications require support for dynamic adaptation to the changes in the execution environment and thus require a system that controls their execution. Unfortunately, The Globus Resource Allocation Manager (GRAM) [70], which is a Grid interface to local management systems, does not support all of the plentiful features of each underlying batch system. A variety of useful features, such as the ability to checkpoint are missing. Condor-G [166], used by the DataGrid resource broker [45], has the same disadvantages as GRAM. Additionally, in the current OGSA [61, 70], which actually exposes GRAM functionality as a service, the same problems arise.

Therefore, we have decided to build a migration mechanism to allow federates to migrate when their performance is not satisfactory. There are several HLA-specific issues [26] concerning the design of migration support for HLA components of interactive applications. The system should be able to suspend the federate without generating HLA-specific errors (e.g. by resigning from a federation). There are also other HLA-specific issues concerning the consistency of HLA services such as data management, ownership management and time management - e.g. when federate A regulates the time of federate B and federate A is selected for migration - during migration, federate B must behave in a consistent way.

Migration of one of the federates without affecting the behavior of other federates is a nontrivial task. The HLA specification hides the actual location of data and messages, therefore using the HLA Management Object Model (MOM) [78] for obtaining information about a federate's internal state cannot guarantee that this state is up to date. A different approach is to write a new implementation of the HLA stan-

dard. This, however, would be a huge task and would also force the developer to use a selected implementation of the Runtime Infrastructure (RTI) library.

We have decided to use the HLA specification Federation Management API to save and restore the HLA internal state[78]. This API contains routines designed to freeze the internal state of the HLA communication bus from the beginning of the saving process up until complete restoration. It assures that all RTI services such as *Data Distribution Management (DDM)*, *Ownership management* and *Time Management* are not affected during the saving/restoring process. No operations that change the RTI internal state are allowed during the saving process. This solution is not completely transparent to the developer, since it requires using the HLA API and it does not allow for saving user-specific data, which we need to add. However, it assures state consistency between all federates with relatively little overhead. A solution built over the HLA API and allowing the storage of user data was published in [191] and is described in Chapter 5 of this thesis.

The next step was to answer the following question: how to know when the simulation is behaving badly? According to the HLA standard [78], HLA-based applications can be monitored and controlled by the Management Object Model (MOM). Although MOM enables monitoring invocations of HLA functions, it does not allow the programmer to introduce performance metrics (e.g. invocation time). We were therefore looking for a monitoring system that is Grid-aware, introduces low monitoring overhead, is flexible, secure and transparent to the simulation developer. Finally, we have chosen The Grid-enabled OMIS-compliant Monitor (OCM-G), since it fulfills all the above requirements and complies with the On-line Monitoring Interface Specification (OMIS)[103]. Our goal also was to make the solution independent of the actual HLA implementation. The concept of monitoring HLA simulations was published in [141] and is described in Chapter 6 (Section 6.2) of this thesis.

Last but not least, we had to consider the problem of the destination to which a federate should be migrated. In order to make this decision, the system needs to monitor the host and the network infrastructure. Although various monitoring tools are currently being developed [104, 65], it is extremely difficult to monitor WAN connections. Furthermore, the HLA API hides the actual communication between components of the distributed simulation. The tuple space concept conceals the actual structure and order of messages sent from publishers to subscribers. One possible solution to that problem is to implement online HLA-based benchmarks and make decisions about migration basing on their results. The adequate solution was published in [143] and is described in Chapter 6 (Section 6.1) of this thesis.

Proof of concept implementations for the solutions presented in this thesis – performance results of particular elements of G-HLAM are presented in Section 7. We present a feasibility study for the G-HLAM system based on a sample real-world interactive simulation. Our proof-of-concept implementation of the G-HLAM system was tested on two kinds of real simulations. Chapter 8 describes an interactive parallel N-body simulation that allows for exploration of partial simulation results with the possibility of rollback to previous simulation timesteps triggered by the user. Chapter 9 shows how G-HLAM can be used for the collaborative environment for

surgical pre-operative planning previously developed [155].

Additionally, during our work we have also considered the use of Grid core services for the transport of actual federate data. Initially we intended [167] to experiment with Globus GridFTP and Globus I/O implementation extensions for interfacing with the RTI logical bus. The main features of GridFTP that improve data transfer are data stream parallelization and tuning of the TCP buffer. This problem, however, is more technical than scientific and it boils down to developing another, more efficient implementation of HLA. Moreover, only simulations that exchange huge amounts of data would benefit from this solution. Since many companies are currently engaged in developing more and more efficient HLA implementations, we have decided to no longer pursue this.

Another idea was to directly expose HLA functionality as Grid Services interface. The similar solution, but with pure Web services was done in [134]. In this solution, HLA communication between federates residing on different sites is realised by Web services protocols. The invocation of an RTI routine is sent to remote federate and translated to actual RTI request already at remote site as shown in the next Chapter in Fig 2.21. We decided not to go in this direction due to performance issues. Instead we focused on making use of existing effective solutions for HLA implementations [140, 123] and on allowing legacy codes build on them to take advantage of Grid,

According to our analysis of existing approaches made in Chapter 2, especially in Section 2.4, there was no solution that allowed to run HLA simulations on the Grid in efficient way also with support for legacy codes, which is the result of this thesis.

1.6 Methodology

In this thesis we present the analysis, design, implementation and feasibility study of the Grid HLA Management System. Our analysis is based on distributed simulation requirements and the lack of appropriate existing solutions which fully support those requirements. The design of the system is based on the Services Oriented Architecture with stateful services defined by the OGSA standard [61]. To model the architecture of the system and its components, we use the Unified Modeling Language (UML) [169] as it is a well known standard for illustrating system design. To analyze the behavior of the system and show that the system is deadlock-free we use the Petri Net formalism [128]. We also build and describe a proof-of-concept implementation, present performance results and outline a case study: runtime steering of parallel simulations for modeling dense stellar system.

1.7 Organization of this thesis

This thesis is organized as follows: in Chapter 2 we present various approaches to efficient execution of distributed simulations, we analyze how existing environments can be used to allow them to benefit from Grid environments and describe two projects aiming at development of support for simulations: CrossGrid and I-GASP. We also

present techniques for tuning HPC simulations to take advantage of the Grid without sacrificing performance.

In Chapter 3 we give an overview of the architecture of the Grid HLA Management System (G-HLAM) and use the Petri Net formalism to analyze the behavior of the system and to show that our solution is deadlock-free.

In subsequent Chapters we describe the components of G-HLAM: in Chapter 4 we focus in more detail on the interface between actual HLA applications and G-HLAM by presenting the *HLA-Speaking Service* designed for single and multiple processes and the *RTIexec service* that manages the coordination process of each HLA-based application. The *HLA Speaking Service* is responsible for execution of application code on the site on which it resides. It can submit, save and restore the execution of simulation federates. We present two cases of this service. The first is a prototype able to manage only one federation process. The second is a full service that can deal with multiple federation processes.

In Chapter 5 we focus on the part of the system that is responsible for migration of an HLA-connected component or components of the distributed simulation in a Grid environment. We also present migration support routines of the GridHLAController runtime support library for easy integration of HLA simulations into the Grid services Framework.

Chapter 6 presents monitoring services: an HLA-based benchmark suite that allows G-HLAM to obtain information about the behavior of HLA-based simulations in the Grid infrastructure and the *Application Monitoring Service* that monitors the performance of HLA-based applications in order to decide when to migrate its parts to achieve more efficient execution.

Chapter 7 describes performance results of the G-HLAM system. First, we describe various experiments with the migration mechanism, next we present the *Benchmark Services* results, and finally we outline the performance of *Application Monitoring Services*. The experiments presented in this Chapter are based on test applications, while a case study involving real applications is presented in Chapter 8 and Chapter 9. We show how a sample interactive simulation from the real world can benefit from the Grid using the solution proposed in this thesis. In Chapter 8 we present the architecture of a N-body simulation and its associated types of interaction. Subsequently, we present how the application can be run within the Grid HLA Management System (G-HLAM). Finally, we show results from three different experiments on how G-HLAM can improve the performance of a "human in the loop" application. In Chapter 9 we show how the CrossGrid Medical Application can be run within G-HLAM. A summary, discussion and conclusion are provided in Chapter 10.

Note on coauthors of published papers

The results presented in the chapters of this thesis formed the basis to some of the published papers, as indicated at the beginning of each chapter. Apart from promoter professor Peter Sloot and co-promoter doctor Marian Bubak, coauthors of this papers were (in alphabetical order):

- Bartosz Baliś - one of authors of OCM-G system used in this thesis, also author

of many suggestion and comments of using it,

- Allesia Gualandris and Simon Portegies Zwart – results produced by Allesia and Simon are not presented in this thesis. In paper [8] we published joined results.
- Maciej Malawski - coauthor of the CrossGrid architecture presented in Section 2.3.2, author of many comments and suggestions during our discussions on the Grid, OGSA and HLA,
- Robert Szymacha - author of wrappers that allow to monitor HLA applications with OCM-G
- Alfredo Tirado Ramos – results produced by Alfredo are not presented in this thesis. In paper [8] we published joined results. Alfredo is also author of many comments and suggestions during our discussions on the Grid, OGSA and HLA.

Analysis of Approaches Supporting Distributed Interactive Simulations

This Chapter presents the state of the art for existing approaches to efficient execution of interactive, distributed simulations. Its aim is to determine to what extent the requirements described in the previous Chapter are already addressed, to find areas where existing solutions are lacking and also to choose the best environment that fits our requirements in order to provide a starting point for our research.

Furthermore, our goal is to outline the evolution of concepts for running distributed interactive simulations according to the emergence of new concepts in computer science. We want to analyse how the environment we will choose as our starting point is already integrated into those concepts. Therefore, this Chapter is divided into three parts:

- non-Grid environments that were designed to support interactive distributed simulations,
- techniques for porting High Performance Simulations to the metacomputing environment at an early stage of development of Grid concepts,
- mature solutions for running simulations on the Grid

In Section 2.1 we describe environments supporting the development, execution and/or steering of simulations. For each of these environments, we analyse the advantages and disadvantages for adapting Grid solutions. In Section 2.2 we outline the ideas that were used in the past to port HPC simulations to the Grid seen as a large, distributed metacomputing environment. In Section 2.3, we review current efforts of using Grid concepts for distributed simulations. Finally, in Section 2.4 we describe efforts of using Grid and Web solutions to make interactive distributed simulations more interoperable, fault-tolerant and efficient. The aim of this overview is

to present the background of our research and to choose the best existing solution as a basis for distributed interactive simulations that need to run on the Grid.

2.1 Environments for distributed simulations and their runtime steering

This Section presents environments for supporting distributed interactive simulations. Regarding requirements stated in the previous Chapter (Section 1.3), for each of the given examples we analyze two different constraints:

1. Type of support for distributed simulations: architecture of the system, the way a simulation can be distributed and steered, protocols used.
2. Amount of effort that has to be invested in order to take advantage of Grid technology; particularly: architecture scalability, support for execution in a distributed environment, possible ways of porting to the Grid.

2.1.1 Computational Steering Environment

Functionality

The aim of the Computational Steering Environment (CSE) [40] is to provide scientific end users with an environment in which they can easily define interactive interfaces to ongoing simulations. The CSE architecture, an example of which is shown in Fig. 2.1 is implemented as a set of processes - called satellites - which implement stan-

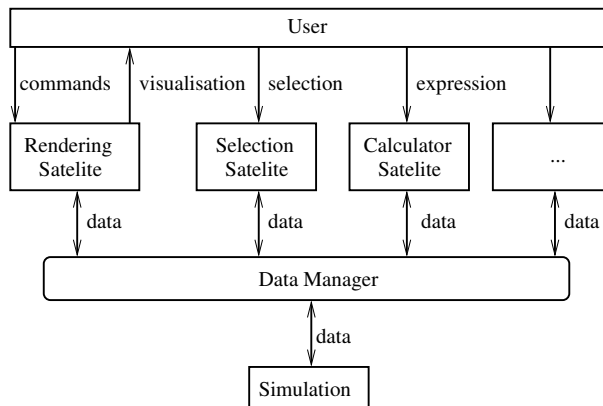


Figure 2.1: Architecture of the Computational Steering Environment [40] implemented as a set of processes - called satellites - that communicate via a central manager.

dard visualization operations. The simulation is also seen by the system as a satellite. Satellites cooperate by sending and receiving data from a central data manager

which, in turn, notifies all interested satellites about data mutations. The most predominant satellite is the interactive graphics editing tool called Parametrized Graphics Object (PGO) editor, which allows the end user to sketch out visualizations. A two-way binding between visualization and data is achieved by binding the sketch to data within the data manager. In Fig. 2.1, three sample satellites (apart from the simulation satellite) are shown. These are as follows: the rendering satellite which can react to user commands (e.g. drag, pick, etc.), the calculator satellite that can calculate a given expression and the selection satellite that can perform the appropriate selection.

Protocols

CSE uses the TCP/IP protocol as a communication layer between satellites.

Possibility of adapting to the Grid

The main disadvantage of the CSE is its centralization, which hampers its scalability in Grid environments. However, the idea of a data manager and satellites can be somehow extended (e.g. by building hierarchical or distributed data sets).

2.1.2 Collaborative User Migration, User Library for Visualization and Steering

Functionality

Collaborative User Migration, User Library for Visualization and Steering (CUMULVS) [92] allows the programmer to add interactive steering and visualization to an existing parallel or serial program (task) as shown in Fig. 2.2. With CUMULVS, each of the collaborators can start up an independent view program that will connect to the running simulation program. Viewers allow scientists to browse through the various data fields being computed and observe the ongoing convergence toward a solution. CUMULVS supports a variety of visualization systems from commercial packages such as AVS to public domain environments such as Tcl/Tk. It collects information and coordinates communication between the various viewers and the simulation program. The result is presented as a uniform field of data even if the actual data are distributed across a set of parallel tasks.

CUMULVS also allows an application program to perform user-directed checkpointing and automated restarts of parallel programs using checkpointing, even across a heterogeneous cluster of machines. A single user library interface routine passes control to CUMULVS periodically, to transparently handle the viewer attachment / detachment protocols, the selection and extraction of data, and the updating of steering parameters. CUMULVS allows each front-end viewer to interactively select the granularity and extent of data that it desires to view. The functioning of CUMULVS is shown in Fig. 2.2. CUMULVS interface joins various viewers with elements of distributed or parallel simulations (tasks).

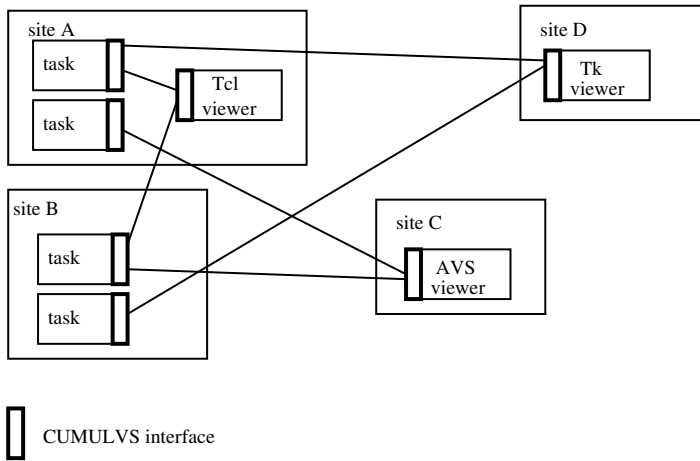


Figure 2.2: Functionality of Collaborative User Migration, User Library for Visualization and Steering [92] – it allows to add interactive steering and visualization to an existing parallel or serial program (task).

Protocols

Currently, CUMULVS uses PVM [135] as its message passing substrate; it allows for pairs of anonymous tasks to communicate with each other without both tasks being started at the same time. MPI does not allow these dynamics, so porting CUMULVS ideas to MPI would not be easy.

Possibility of adapting to the Grid

In the past [92], several experiments were conducted to explore the feasibility of porting CUMULVS to the Globus/Nexus [70] environment as an alternative to the PVM message-passing substrate. There were plans to extend CUMULVS to provide better support for MPI applications which use MPICH-G [116] on Globus. However, the most recent version of CUMULVS supports only the PVM library. The obvious advantages of this tool are its fault tolerance and checkpointing ability. One major disadvantage is the lack of support for MPI.

2.1.3 Visualization of Parallel Simulation Algorithms for Extended Research

Functionality

The system called Visualization of Parallel simulation algorithms for Extended Research (VIPER) [136] is a conceptual prototype for on-line visualization for parallel simulation algorithms. VIPER breaks down the structure of an application cycle into

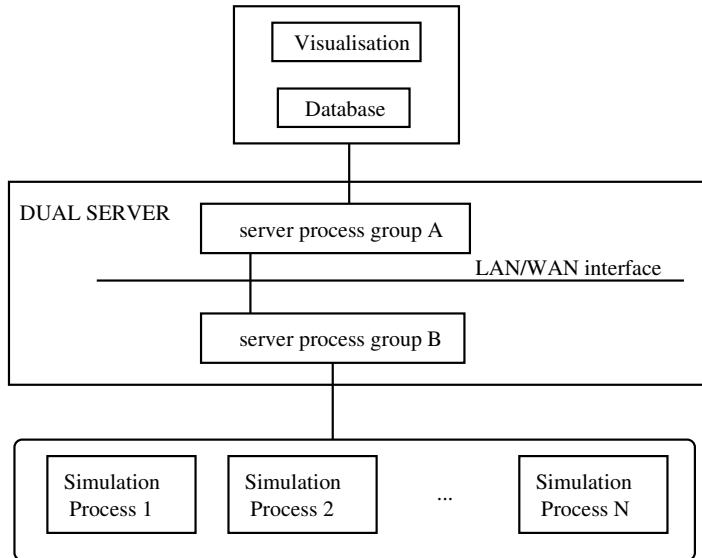


Figure 2.3: Architecture of the Visualization of Parallel simulation algorithms for Extended Research system [136] containing DUAL SERVER for two kinds of clients: simulation and visualisation.

three distinct logical components: computational units, connectivity units and visualization units, which compose a pipeline architecture as shown in Fig. 2.3. VIPER follows a new DUAL SERVER model, which is based on a client/server/client architecture. One client is the parallel simulation algorithm, the other client is the visualization system. On request, DUAL SERVER executes different kinds of services for both clients. The processes involved are split into three groups: parallel simulation algorithm running on a remote parallel computer system or a cluster of workstations, visualization system running on a graphical workstation and the distributed application DUAL SERVER running on processors which reside in a LAN and on a remote parallel computer system in a WAN or LAN. Additionally, to process and re-process scientific data, the visualization unit maintains a database which holds data from the distributed and replicated data structures of the parallel simulation algorithm, handed out by the DUAL SERVER. To enable online visualization, the DUAL SERVER offers an infrastructure to extract data out of the computational unit during simulation time, to transfer data across a WAN (or LAN) and to hand out data to the visualization unit.

VIPER propagates an object-based view of the whole application cycle. The so-called objects are built from distributed and replicated data structures of the parallel simulation algorithm, which are based on parameters of the mathematical model and numerical methods. Because of the object-based view of VIPER, interaction is similar to execution of operators on objects. To steer the simulation process, a modification

operator is applied to the parameters of the mathematical model or to the parameters of the numerical methods. Each object (parameter) is associated with so-called synchronization points (SPs). These synchronization points are executed during simulation time that means sending a signal (request) to the DUAL SERVER. The DUAL SERVER extracts data out of the computational unit, provided that the synchronization point and the relevant objects are switched on.

Protocols

RPCs (remote procedure calls) and the XDR [182] protocol are used to implement interprocess communication between WAN and LAN distributed processes of the DUAL SERVER. Again, RPCs are used to communicate with the visualization unit.

Possibility of adapting to the Grid

The architecture of the DUAL SERVER is quite interesting for development in a Grid environment. However, according to the VIPER project Web pages, only a prototype implementation exists and it appears that the project has been discontinued.

2.1.4 Visualisation Interface Toolkit

Functionality

Visualisation Interface Toolkit (VISIT) [173] is a library for point-to-point communication between two independent applications (such as a simulation and a visualization) using a client-server model. The architecture of VISIT is shown in Fig. 2.3. Bundled with VISIT is a simple name service called SEAP (Service Announcement

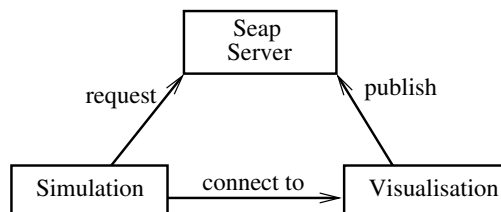


Figure 2.4: Architecture of Visualisation Interface Toolkit [173] introduces a concept similar to SOA - a service announcement protocol (SEAP) server is used to register visualisations services that can then be found by simulations.

Protocol). A visualization can register its service(s) on a SEAP server. The simulation can then query this information (consisting of a hostname and a port number) and use it to connect to the visualization.

Furthermore, VISIT provides the Vbroker tool for attaching multiple visualizations to a single simulation. It accomplishes this by forwarding all send-requests from the client to all attached servers. Receive-requests, however, are only forwarded to a

single server Vbroker enables switching between servers. This means that it allows for multiple passive viewers but only one visualization may steer the application.

The simulation-related API is reduced to a few function calls. Language bindings are available for C, FORTRAN and Perl. On the visualization side, VISIT has language bindings for C and Perl. The distribution contains demo clients and servers written in all the supported languages. A complete server for AVS/Express [11] is also included.

Protocols

The current implementation of VISIT uses TCP/IP sockets for connecting simulations and visualizations. Besides that, it is also possible for the simulation to write data to a file, which a visualization can then read, using the same send/receive calls as when a socket connection is established. This is intended to be used for offline visualizations, where the simulation data is recorded in advance.

Possibility of adapting to the Grid

VISIT is a quite simple tool, where the data transfer is over IP, no parallel libraries are supported and no sophisticated error recovery protocols are provided. However, the idea of thinking about a visualization module as a server which can be registered as a service is very similar to contemporary Service Oriented Architecture and Grid services [61] concepts.

2.1.5 Cactus Problem Solving Environment

Functionality

Cactus [7] is an open-source problem solving environment designed for scientists and engineers. The Cactus code was originally developed to provide a framework for the numerical solution of Einstein's equations, one of the most complex sets of partial differential equations in physics. Cactus has since evolved into a general purpose, open-source environment that provides a unified, modular and parallel computational framework for scientists and engineers. The name Cactus comes from the design of a central core, which connects to application modules - or thorns - through an extensible interface. Thorns can implement custom-developed scientific or engineering applications, such as the Einstein solvers, or other applications such as computational fluid dynamics. There exists a standard computational toolkit, consisting of thorns which provide a range of capabilities, such as parallel I/O, data distribution, or checkpointing. A thorn is the basic working module within Cactus. All user-supplied code goes into thorns, which are, by and large, independent of each other. Thorns communicate with each other via calls to the flesh API, plus, occasionally, via custom APIs of other thorns. One of the key concepts which relates to thorns is the concept of implementation. Relationships among thorns are all based upon relationships among the implementations they provide. In principle, it should be possible to swap one thorn

providing an implementation for another thorn providing that implementation, without affecting any other thorn. An implementation defines a group of variables and parameters, which are used to implement some functionality.

Cactus is a problem solving environment for a wide range of issues, here we concentrate on its support for parallel or distributed simulations and their visualisation. Among other features, Cactus includes a mechanism for building distributed parallel simulations. In Cactus, different thorns can be used to implement different parallel paradigms, such as PVM, Pthreads [119], OpenMP [124], CORBA [34], MPICH-G etc. Cactus can be compiled with as many driver thorns as required (subject to availability), with the one actually used chosen by the user at runtime through a parameter file. A detailed description of how Cactus deals with remote steering and visualisation of simulation can be found below in description of used protocols.

Protocols

Cactus provides the ability to stream online data from a running simulation via TCP/IP socket communications. Multiple visualization clients can connect to a running Cactus executable via a socket from any remote machine on the Grid, then request arbitrary data from the running simulation and display simulation results in real time. This can be done in two ways: by an HTTP control interface or through socket connections. The approach for streaming arbitrary data of any type is based on the HDF5 [76] I/O library and is shown in Fig. 2.5. Cactus provides a stream

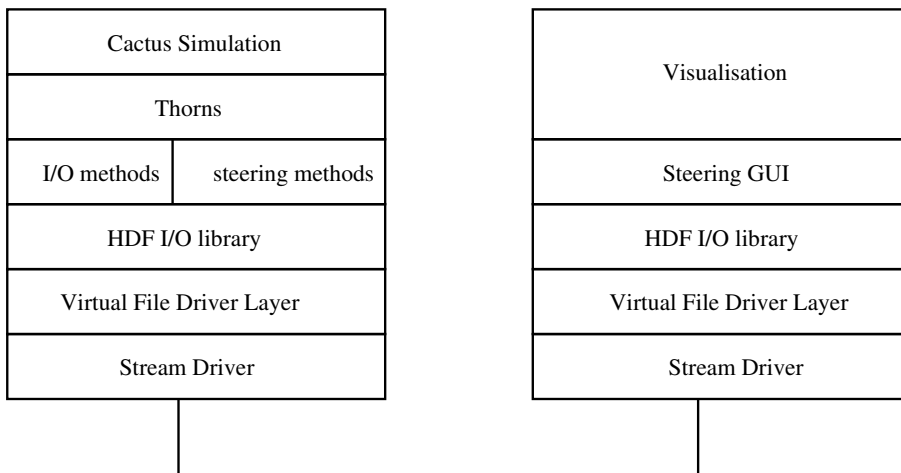


Figure 2.5: Connection between simulation and visualisation in Cactus [7] using stream driver for transferring HDF5 data.

driver which holds the HDF5 data to be streamed out of the Cactus simulation as an in-memory HDF5 file. Upon activation, the entire file is sent through a socket to the connected client. In the client application, the same driver is used to reconstruct the

file which then can be accessed in usual ways to read the HDF5 datasets. The stream driver is capable of sending data simultaneously to multiple clients. Data streaming can be conducted in a bidirectional way: Cactus writes parameters to an HDF5 file which is then streamed to a connected steering client. After some user interaction, this client sends back a modified version of the parameter file that is read and evaluated by Cactus.

Possibility of adapting to the Grid

Cactus already provides support for Grid-enabled MPI – MPICH-G. Its main disadvantage is that the parallelization is limited to domain decomposition. However, because of the modular architecture of Cactus, it appears that adding extended functionality would be quite easy for application developers.

2.1.6 Discover

Functionality

Discover [100] is a interactive and collaborative system that enables geographically–

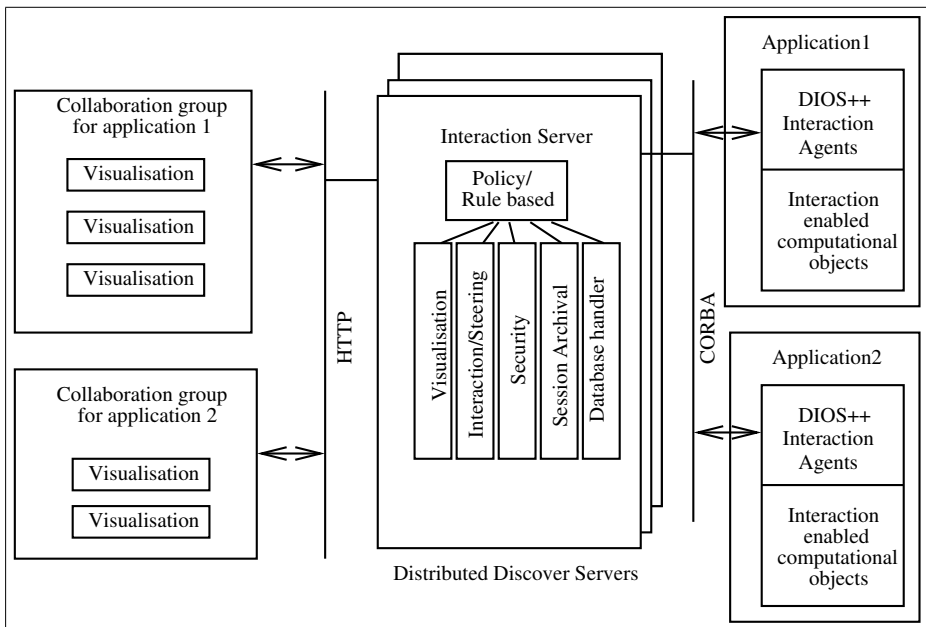


Figure 2.6: A three-tier architecture of Discover [100] composed of detachable thin clients at the front-end, a peer-to-peer network of servers in the middle, and the Distributed Interactive Object Substrate (DIOS++) at the back-end.

distributed scientists and engineers to collaboratively monitor, and control high performance parallel/distributed applications using web-based portals. As shown in Fig. 2.6, Discover provides a three-tier architecture composed of detachable thin clients at the front-end, a peer-to-peer network of servers in the middle, and the Distributed Interactive Object Substrate (DIOS++) at the back-end.

An important part of Discover is a rule-based visualization system. Rules are decoupled from the system and can be externally injected to manage such visualization behavior at runtime, as autonomically selecting the appropriate visualization routines and methods and adjusting the extraction threshold.

To allow rule-based management, DIOS++ provides autonomic objects that extend application computational objects with sensors to monitor the state of the objects. It also contains actuators to modify the state of the objects, access policies to control accesses to sensors and actuators and rule agents to enable rule-based autonomic self-management.

Protocols

Discover uses HTTP for connection with visualisation thin clients and CORBA for communication with application engines.

Possibility of adapting to the Grid

Discover already possesses a distributed and scalable peer-to-peer type of architecture. However, it can still be extended to take advantage from Grid technology, which would enable it to be automatically set up and effectively run in a non-centrally controlled environment consisting of different administrative domains.

2.1.7 TENT

Functionality

TENT [148] is a software integration and workflow management system that helps improve the building and management of process chains for complex simulations in distributed environments. TENT allows for online steering, and visualization of simulations. The TENT architecture is shown in Fig. 2.7. Wrappers are used to interface application modules (e.g. Computation Fluid Dynamics (CFD), Computation Structural Mechanics (CSM), visualisation or filters) with the system. The system consists of base components which include: modules for controlling workflows; factories for starting system and applications in the distributed environment; the name server as the central information service. There are also support components – additional services for special application scenarios not covered by the basic functionality. Examples include: a data server for storing data files, a monitoring and reporting component, and several special control components (e.g., for coupled simulations like the ones parallelized with MPI). The system is controlled by a user through a GUI.

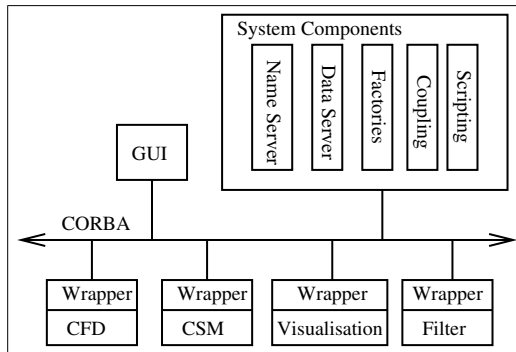


Figure 2.7: Basic components of TENT [148] – a system for building and management of complex distributed simulations. Application modules (i.e. simulation engines, visualisation modules) are connected to the system using wrappers.

Protocols

TENT uses CORBA for communication between parts of the system.

Possibility of adapting to the Grid

TENT is already Grid-enabled. The Globus Toolkit version 2 is used for resource selection, for starting applications, and for data transfer and security. The system supports MPICH-G2 simulations.

2.1.8 High Level Architecture

Functionality

The High Level Architecture (HLA) standard [78] defines an infrastructure for developing distributed interactive simulations. The HLA standard is defined by three components: Federation Rules, the HLA Interface Specification, and the Object Model Template (OMT).

In HLA terminology each component of a distributed simulation is called a federate and can form federation with other federates. The communication is done via the Runtime Infrastructure (RTI) as shown in Fig. 2.8. In DoD reference implementation of the Runtime Infrastructure each federate uses the *libRTI* library to communicate with other federates. There are also two coordination processes - each federation is managed by its *FedExec*. All federations and *FedExec* processes are managed by *RTIexec*.

The Federation Rules describe the responsibilities of federates and their relationships with the RTI. The interface specification identifies how federates will interact with the federation and, ultimately, with one another by means of RTI.

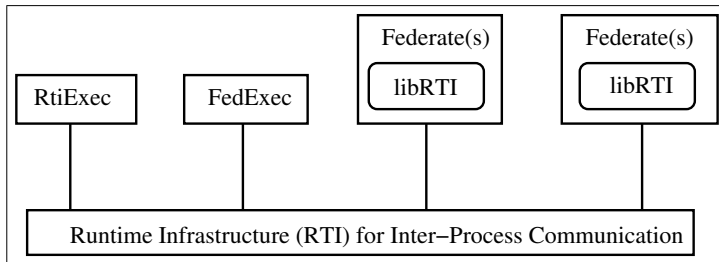


Figure 2.8: Basic components of High Level Architecture Runtime Infrastructure [78]. Elements of distributed simulation (federates) communicate via RTI using *libRTI* library. Federates are controlled by *RtiExec* and *FedExec* processes.

The most important services provided by RTI to support interactive applications are *data distribution management* - in charge of efficient routing of data between federates, *time management* - controlling message ordering, and *ownership management* - transferring ownership that controls rights to changing attributes of objects that are exchanged between federations.

All objects and interactions managed by a federate and visible outside the federate are described according to the standard OMT. The OMT provides a common method for representing HLA Object Model information. In HLA the local time management mechanism of one simulation component (federate) is not visible to other federates. Hence, all forms of time management (time driven, event driven, parallel discrete event driven, real time driven) may be linked together. HLA separates communication infrastructure from actual simulation. Additionally, it introduces a uniform way of description of events and object being exchanged between federates. All of these features allow interoperability between various simulations.

Protocols

The HLA standard does not define an underlying protocol for implementation. In the HLA RTI implementation used in this thesis, RTI, is built over ACE/TAO Real Time CORBA [162].

Possibility of adapting to the Grid

HLA posses advanced mechanisms supporting distributed simulations, so execution of HLA-based applications on the Grid should be natural extension of its usage. However, the HLA standard was developed assuming a certain quality of service in the underlying environment. Therefore, there is a need for adaptation of HLA-based applications to the dynamically changing Grid as well as for automatic setup, dynamic discovery and fault-tolerance mechanisms. A detailed description of these issues is provided in Section 3.1.3.

2.1.9 JSIM

Functionality

Java-based simulation and animation environment (JSIM) [90] is a representative example of Java-based distributed simulation environments for Web based simulations. It allows for developing simulation models and encapsulate them as Java Beans, so

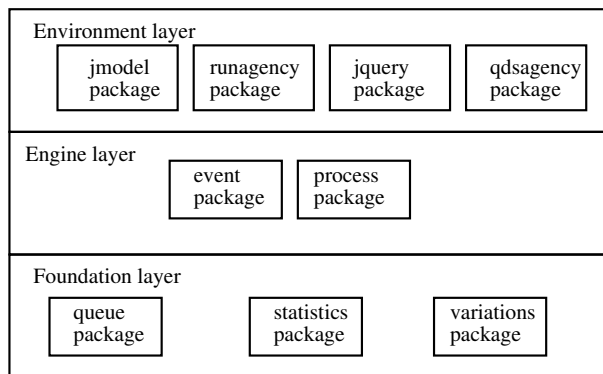


Figure 2.9: Architecture of JSIM [90], consisting of three layered groups of packages – the foundation layer, engine layer and environment layer.

they can be executed and their output can be written to a database. JSIM consists of three layered groups of packages – the foundation layer, engine layer and environment layer as shown in Fig. 2.9. The foundation layer consists of packages useful in simulation in general: the queue package contains various queue models, the statistic package contains classes to collect statistical information, other packages also provide a wide variety of random variations. The engine layer provides two popular simulation paradigms: the event package supports construction of event scheduling type models and the process package provides process interaction type models. The environment layer contains four packages: jmodel provides a visual designer for the process package, jquery allows models to access databases, runagency manages model execution and qdsagency provides a convenient means for generating and accessing simulation data.

Protocols

JSIM uses Java Beans communication.

Possibility of adapting to the Grid

JSIM is a representative example of a system supporting Web-based simulation. The simulation models, encapsulated in Java Beans are located on the Web and executed.

This idea is similar to the Grid concept, where resources are searched for and accessed. However, as for HLA, there is a need for adaptation of JSIM-based applications to the dynamically-changing Grid environment. Furthermore, JSIM is not a standard for distributed interactive simulations.

2.1.10 Interactive Simulation Systems Conductor

Functionality

Interactive Simulation Systems Conductor (ISSConductor) [193] is an agent oriented component framework for Interactive Simulation Systems. As shown in Fig. 2.10,

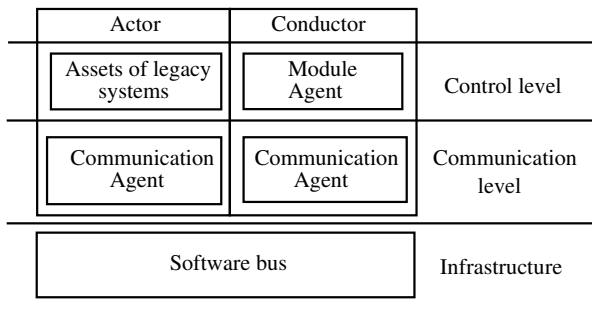


Figure 2.10: Architecture of ISS-Conductor [193] consists of Module Agents for controlling application modules and Communication Agents for communication between Module Agents and actual modules.

the system introduces two kinds of agents: Module Agents that are specific for different modules of interactive application like simulation, visualization and interaction and Communication Agents that are used for communication between Module Agents and actual modules. Module Agent uses an extended finite state machine to model the run-time behavior of a component, and adopts first order logic to represent the interaction constraints between components and to implement them in the knowledge bases of agents. ISS-Conductor separates the basic computational functions of a component from its run-time behavior controls, and provides a high level interface for users to design interaction scenarios. The framework is very general and can be used for various interactive applications.

Protocols

ISS-conductor is built on top of HLA described in the previous Section.

Possibility of adapting to the Grid

Currently, ISS-Conductor is not Grid-enabled, however it can easily take advantage of the system presented in this thesis, since it is built over the HLA standard.

2.1.11 Amsterdam Parallel Simulation System

Functionality

The Amsterdam Parallel Simulation System (APSYS) [84, 126] is an optimistic Time Warp parallel discrete event simulation kernel. APSIS has been designed for complex

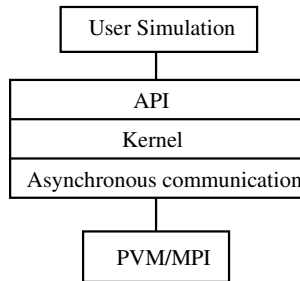


Figure 2.11: APSIS simulation kernel provides an API for the user simulation code and communicates with the inter-process communication library (PVM or MPI) using asynchronous communication.

system simulations, such as asynchronous cellular automata. It is oriented towards simulations that have a very large and slowly changing state – for such models it provides an incremental state saving algorithm. The APSIS architecture is shown in Fig. 2.11. At the bottom the kernel communicates with the inter-process communication library (PVM or MPI) using asynchronous communication. At the top, it provides an API for the user simulation code which contains the following groups of functions:

- support routines: initialisation, termination, parameter tuning, etc,
- topology routines,
- query routines: obtaining the Logical Process (LP) identifier, the current values of Local Virtual Time (LVT), Global Virtual Time (GVT), wallclock time, etc,
- event routines: sending, retracting, receiving,
- state saving.

Protocols

APSYS can use PVM or MPI as its communication infrastructure

Possibility of adapting to the Grid

APSYS is already Grid-enabled. In [84] various techniques were introduced to improve the performance of the APSIS kernel running on the Grid with MPICH-G2:

- introducing routing processes performing wide area communication and separating them from simulation processes,
- introducing multiple routing processes per cluster,
- message aggregation in a routing process,
- shifting the task of computing the simulation's (GVT) from the simulation processes to routing processes.

2.1.12 Virtual Radiology Explorer

Functionality

Virtual Radiology Explorer [14] is a Virtual Reality System that allows a user to in-

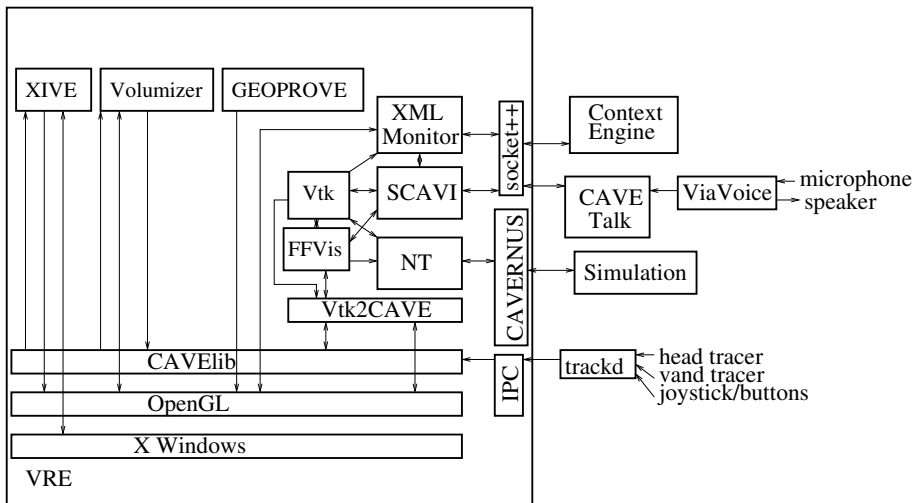


Figure 2.12: Architecture of Virtual Radiology Explorer [14] includes components for speech recognition, tracking of user movement and advanced visualization.

teract with a running simulation and allows for interactive exploration of its results. As shown in Fig. 2.12, VRE includes many different components for speech recognition, tracking of user movement and advanced visualization. The VRE architecture is composed from various modules. XiVE [183] is a library that allows for using X Windows in a Virtual Environment (VE). OpenGL [122] and Volumizer [174] are rendering libraries. GEOPROVE [15] - GEOMETRIC PROBES in VE - allows for taking measurements in VE. The XML-based [184] monitor of VRE internal state provides information about the state over the network. Vtk [175] is a visualisation toolkit used by the Vtk2CAVE [14] library that allows for rendering Vtk objects with CAVElib [28]

applications. SCAVI [146] – Speech, CAVE and Vtk Interaction – allows for interaction with Vtk actors using voice, wand, joystick and buttons, requires CAVETalk [14], ViaVoice [171] and Context Engine [14] for speech recognition. Network Transfer facility (NT) [14] connects VRE with simulation using CAVERNUS library. FFVis is an interactive flow field visualisation utility [14].

Using VRE the user can explore simulation results, manipulate the simulation objects and control simulation execution – start it with various parameters as well as pause, resume and stop.

VRE has been deployed for the virtual reality (VR) and desktop projection modalities. The interaction capabilities of both modalities are compared in [195], where it is shown that VR and desktop within the same interaction–visualisation medium could help satisfy the needs of a wider range of VRE users.

Protocols

Communication with the simulation is done by using the CAVERNUS [29] library

Possibility of adapting to the Grid

VRE is an integrated interaction and visualization component used in the CrossGrid medical application [155] – see also Section 9.1.2.

2.1.13 Summary

In this Section we have presented existing environments that support interactive steering of simulations. For each of presented systems we described the architecture and protocols used to connect simulations with visualizations. For each of the systems we also analyzed the possibilities of adapting it to the Grid environment. A summary of the features of the presented systems is show in Tab. 2.1. CSE, CUMULVS, VIPER, VISIT, Cactus, Discover and TENT focus more on support for steering simulations without much concern for advanced simulation composition from distributed components which often use different time management. As shown in Tab. 2.1, HLA is a standard in that field, allowing for interoperability between different types of simulations. The drawback of HLA is that it does not adapt to changing environments like the Grid. ISS-Conductor is a high–level system built over HLA. JSIM is a good example of a system supporting Web–based simulation, but is not a standard and similarly to HLA does not have the ability to adapt to a changing environment. Additionally, it does not allow for linking event-driven and time-driven simulations in one system. APSIS is already Grid-aware and, thanks to advanced techniques can achieve good performance. However, it is dedicated to parallel discrete event simulations – especially asynchronous ones, with a slowly changing state, which is not the case for interactive simulations. VRE is oriented towards advanced visualization of simulation results and does not posses advanced features for distribution of simulations.

System	Type of simulation and its distribution	Protocol	Porting to the Grid issue
CSE	runtime steering – multiple visualizations for one simulation	TCP/IP	scalability issue
CUMULVS	runtime steering of parallel simulations (PVM)	PVM	porting to MPI issue
VIPER	runtime steering of parallel simulations	RPC	interesting architecture, but no longer supported
VISIT	runtime steering – multiple visualizations for one simulation	TCP/IP (additionally SEAP protocol)	no support for parallel or distributed simulations, but idea similar to SOA
CACTUS	runtime steering of parallel or distributed simulations PVM, Pthreads, OpenMP, CORBA, MPICH-G	two possibilities: 1) HTTP 2) HDF5 format over TCP/IP	support for MPICH-G
Discover	parallel and distributed applications in general	HTTP, CORBA	scalable architecture that could be extended to allow automatic setup and effective run in non-centrally controlled Grid environment
TENT	parallel and distributed simulations	CORBA	TENT is already Grid enabled by using GTv2 and MPICH-G2 features [148]
HLA RTI	standard for distributed simulation includes synchronization mechanism for event-driven and time-driven simulations	CORBA ACE TAO	no adaptation to changing environment, no automatic setup, no dynamic discovery
JSIM	a system for Web based simulations – support for distributed, event driven simulations	communication between Java Beans	no adaptation to changing execution environment
ISS-Conductor	distributed simulations	HLA	same as for HLA

System	Type of simulation and its distribution	Protocol	Porting to the Grid issue
APSYS	discrete-event, distributed/parallel simulations	PVM/MPI	APSYS is Grid-aware by using MPICH-G and advanced performance improving techniques described in [84]
VRE	advanced visualisation for parallel simulation	Cavernus	used in CrossGrid medical application [155] as visualization module

Table 2.1: Main features of the interactive simulation and visualization environments

Finally, we have chosen HLA as a base for distributed interactive simulations running on the Grid. It is a well recognized standard and satisfies most requirements listed in Section 1.3: it offers all the necessary functionality for simulation developers such as support for time management (synchronization) and data distribution management. Its additional and important feature is that the local time management mechanism of one simulation component (federate) is not visible to other federates. Hence, all forms of time management (time-driven, event-driven, parallel discrete event, real-time-driven) may be linked together. HLA also allows to build scalable simulation systems. It separates the communication infrastructure from the actual simulation. Additionally, it introduces a uniform way of describing events and objects being exchanged between federates. All of these features allow interoperability between various simulations. Although HLA originates from the defense technology, there is a growing interest from non-military areas like manufacturing, transportation and gaming industries. Therefore companies are currently working on more scalable and efficient implementations of the standard [140]. Recently, open source implementation was also released [123]. The overview of the applications that use HLA is described in Section 1.2.2.

2.2 High Performance Simulations in a Metacomputing Environment

2.2.1 Introduction

In this Section we present approaches to interactive simulations in the metacomputing Grid environment. The solutions described illustrate the evolution of concepts related to running simulations on the Grid.

In the early days, the Grid was seen as a global metacomputer [60] that could be used instead of expensive supercomputers. It could also be viewed as a large distributed-memory parallel computer consisting of multiple (groups of) processors which exchange data across communication links. This vision is closely related to the

first version of the Globus Toolkit [70] together with a Globus-enabled MPI implementation (MPICH-G) [116] which became a de-facto standard for Grid computing [62].

An example of a metacomputing system is the Polder Metacomputing Environment [85]. Polder offers high-performance computing and interactive simulation support in a wide area network. The features offered include efficient management of resources, in particular multi-level scheduling and migration of tasks that use PVM or sockets. Its dynamic task migration component called Dynamite continuously ensures optimal task allocation of parallel applications that make use of the PVM library.

In parallel computers, a proprietary interconnection network is provided by a vendor; in clusters the links may be commodity networks such as Fast Ethernet. However, in distributed supercomputing the communication links are relatively slow, with high latency times and high potential for failure. Therefore the idea of using the Grid for running High Performance Simulations forces the simulation developers to invent various techniques that deal with the problem of bandwidth and latency in Grid environments. This Section shows examples of Grid applications and systems making use of these techniques.

In [59] a range of decomposition techniques is presented to construct a distributed supercomputing application:

- pipelining or dataflow decomposition,
- functional decomposition – it is useful when the simulation can be divided into parts, which execute more efficiently on different types of architectures,
- domain decomposition – this approach is particularly difficult to parallelize in Grid environments, especially with a tightly coupled simulation.

In the last case one can think of dividing the problem into parts which do not need to communicate with one another very often. Another useful technique is overlapping communication and computation (i.e the exchange of data between different tasks is carried out in smaller subdomains).

In this Section we present examples of successful porting of applications to the Grid despite their HPC requirements.

2.2.2 Synthetic Forces Express project

One example of an application ported from a classic parallel architecture to a distributed Grid environment is the Synthetic Forces Express project (SF-Express) [59] which uses the Distributed Interactive Simulation (DIS) [48] technique to model the behavior and movement of hundreds or thousands of entities for military training, analysis and planning.

Techniques of adapting to the Grid environment

In order to effectively run the simulation in a distributed environment, the entities were distributed over a large number of simulation nodes. SF-Express used a technique called *region of interest management* to reduce the amount of communication. The entities were assigned to nodes in a way that preserved physical locality as much as possible. The nodes were then organized into groups and a router node associated with each group was charged with propagating to other routers only those events, which might be of interests to their groups. For example tanks in Germany did not require information about naval vessels conducting training exercises in the Pacific.

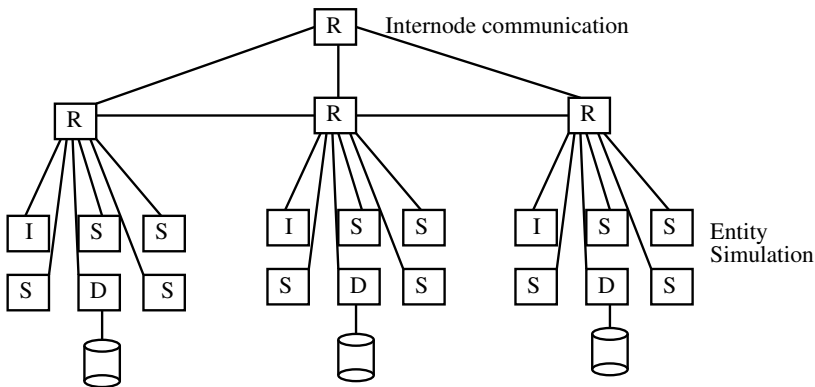


Figure 2.13: Architecture of SF-Express Application consists of nodes performing actual simulation (S), performing communication (R), providing interest management functions (I) and data servers (D).

The architecture of the SF-express is shown in Fig. 2.13 and consists of four kinds of nodes : *entity simulators*, marked with "S", performing actual simulation, *routers*, marked with "R", performing efficient communication between regions, *data servers*, marked with "D", used to achieve uniform IO performance and *interest management nodes*, marked with "I", providing interest management functions.

Grid technology used

SF-Express used the Globus Toolkit version 1 [70] infrastructure; namely it utilized the Globus Resource Allocation Manager (GRAM) to initiate an SF-Express computation so that the computation could be started from a single point. It also used the Globus Heartbeat Monitor, which provided a mechanism for tracking the state of an SF-Express computation. The SF-Express developers have also experimented with the use of the Globus communication mechanisms, including MPI over Nexus [70].

2.2.3 Airflow simulation

The aim of this application was to simulate airflow around an airborne vehicle [13] using a computational fluid dynamics (CFD) algorithm. In this application the CFD scheme divides the entire problem domain into a set of meshes that overlap one another by one or more cells. The solution proceeds by updating, at each iteration, the boundaries on each mesh with interpolated data from overlapping meshes. Additionally, geometrically complex shapes are broken into simpler groups. The parallelization is done by domain distribution. Meshes form groups that are assigned to different processors. Therefore, it is only necessary to communicate some of the boundary information between processors.

Techniques of adapting to the Grid environment.

In order to achieve good performance in the distributed environment the following techniques were used:

Advanced partitioning and load balancing - a METIS [13] graph partitioner was used to assign grids to processors in a way that balanced load and minimized communication. One of the main problems was that on the Grid the size of domain may interact with the various cache sizes in unpredictable ways. Therefore, instead of estimating the work required by each process from the size of its data domain, the amount of work was actually measured.

Overlapping communication and computation - In the original parallel scheme, all communicating processors first exchange boundary values and once the exchange is completed, the internal values are computed at each of the processes separately. In the new scheme, latency tolerance is obtained by delaying the boundary value update by one additional timestep. The boundary value exchange is initiated at the beginning of a timestep, but the values are not used until the beginning of the next timestep. It allows the overlap of computation and communication for as much as the duration of one entire timestep.

Grid technology used

Globus was used as the enabling device for geographically-distributed computation. Its performance was tested on three separate SGI Origin2000 machines: two located at the NASA Ames Research Center and the third at the Argonne National Laboratory. A maximum of 8 processors were used on each machine. The application used Globus version 1.0.0 as middleware and Grid enabled MPI version 1 – MPICH-G as the message-passing library.

2.2.4 Numerical relativity (Cactus application)

The application was described in [8] as an example of how to deal with latency and bandwidth in a Grid environment. It is oriented towards numerical relativity problems – the model used solves Einstein’s equations for the evolution of gravitational

wave spacetimes. This application is an example of a large scale computing problem: it contains many coupled, nonlinear hyperbolic and elliptic partial differential equations (PDEs) and requires resources with computational power and large memory space. The code is based on finite differencing techniques and functions discretized on a regular mesh. The application uses the Cactus framework presented earlier in Section 2.1.5, in particular its MPICH-G thorns.

Techniques for adapting to the Grid environment

The authors described three different techniques dealing with Grid environment conditions as described below:

Overlapping communication and computation was gain by redistributing mesh points so that the WAN communicating processors had fewer mesh points and could overlap communication with computation on the other processors.

Compression was achieved by a Cactus thorn compressing messages prior to transmission.

Infrequent synchronization – A ghostzone is a part of memory of a local processor where remote data needed by the local processor are fetched and are used together with actual local data for calculations. Larger ghostzones lead to an increase in communication granularity at the cost of replicated computation and increased memory usage. This reduces the number of messages (and hence the costs related to communication latency) while the total amount of data exchanged remains constant.

Grid technology used

The application is parallelized with the Grid-enabled MPI version 2 - MPICH-G2. The test were performed on four supercomputers (three SGI Origins at NASA and one IBM Power-SP at SDSC). The measured bandwidth between the two sites was 3 MB/s, while the bandwidth between machines at each site was 100MB/s. The results proved that presented techniques were successful.

2.2.5 Summary

This Section presented techniques used to redesign HPC simulations so that they can achieve good performance in a Grid distributed environment. We presented three example applications: SF-Express – a military simulation application, Airflow (CFD) simulation and a numerical relativity simulation. The reviews presented here show that although efficient execution of HPC simulations on the Grid is not a trivial task, it can be successfully achieved by applying various methods summarized in Tab. 2.2. The region of interest management technique is based on grouping of processes in a way that those communicating more frequently are closer. It also introduces so-called router nodes that send data between processes that are farther apart. Communication and computation overlapping is a technique of performing computations while sending results from the previous step of calculations. Advanced load balancing is

performed by special applications called partitioners which assign processes to processors to balance load and minimize communication. Infrequent synchronization means that data exchange between farther processes is less frequent than between closer processes. As described above, achieving high performance of simulations in

Technique	Application
region of interest management	SF-Express
communication and computation overlapping	Airflow, Numerical relativity
advanced load balancing	Airflow
data compression	Numerical relativity
infrequent synchronization	Numerical relativity

Table 2.2: Summary of the techniques used to redesign HPC simulations for the Grid

Wide Area Networks with high latency times is not a trivial issue. Various methods have to be applied either on the user level (as described in this Section) or on the environment level (as, for example, in APSIS - see Section 2.1.11).

The HLA, which we have chosen as a basis for distributed interactive simulations to be run on the Grid, includes some facilities to improve performance in WANs. HLA is a more efficient successor of DIS, which was used in the SF-Express application, therefore techniques for achieving high performance are shifted to the middleware (HLA itself) and do not need to be performed on the application level (as for the SF-Express case). This is done mostly by reduction of unnecessary network traffic, which is not trivial taking into account the advanced functionality provided by HLA. Methods used in Declaration Management (DM) service [78] such as the publish/subscribe mechanism ensure that events of a particular type are transported only to those federates that have expressed interest in them. Furthermore, the Data Distribution Management (DDM) service [78] allows for interest management which means that areas of interest are described by routing spaces, which are used to send data only to the federates that have subscribed to events occurring within particular regions. This is particularly useful when simulating a collection of physical entities where an entity can only reacts to and acts upon events that occur within a certain distance of this entity.

However, the techniques described in this Section (also those provided by HLA) do not cover all aspects of the issue of running distributed interactive simulations on the Grid. This is because today's Grid is seen not only as a big metacomputing system, but also includes a set of interoperable services that provide various functionality to the users. This new approach gives even more possibilities for interactive distributed simulations as described in the next Section.

2.3 Grid solutions for distributed interactive simulations

In this Section we present modern approaches to the Grid – the environment we will use for running interactive, distributed simulations. We begin with presenting Grid concepts and evolution of its architecture and technology. Next, we outline goals and functionality of chosen Grid projects aiming at developing an environment for interactive and distributed simulations. We focus on the interactivity aspect of those projects and concepts for its realisation. The goal of the analysis is to check what the Grid community has already done in this field and how to extend its efforts.

2.3.1 Evolution of Grid architecture

In Section 1.1.2 we have briefly described Grid as a technology that influences area of distributed simulations. In this Section we present Grid architecture and its evolution in more detail. The first widely recognized definition of Grid and its architecture was described in [62]. This paper also describes a roadmap for Grid technology that is based on the Globus Toolkit [70] (the paper is adequate to its version 1 and 2). As can be seen in Fig. 2.14, Grid is divided into four basic layers – they are: fabric, connectivity, resource and collective [62].

The fabric layer consists of local software that controls single resources – for example operating systems, queue systems or local developer support.

The aim of the connectivity layer is to join Grid resources – it consists of communication protocols and security – Grid Security Infrastructure (GSI) [70].

The next layer (resource) is responsible for sharing single resources – controlling access to them and their use. Here, we can place Globus Toolkit [70] components such as Grid Resource Allocation Management (GRAM), efficient file transfer (GridFTP), and an information service called the Meta Directory Service (MDS) – namely its lower level Grid Resource Information Server (GRIS). This level also includes monitoring services and the Local Replica Catalogue (LRC) [45] – joint Globus and DataGrid software that collects information about local files that need to be replicated to other sites in order to make access to them faster.

The collective layer coordinates multiple resources. It contains ubiquitous infrastructure services, examples of which are:

- index part of MDS – Grid Index Information Server (GIIS) used to collect data from GRIS-es of all sites,
- Coallocation managers (such as Globus DUROC [70] that allocates resources for running the MPI Grid version [116]),
- Resource Brokers – like the DataGrid Resource Broker [45] or the Condor matchmaker [99],
- higher parts of Replica Catalogue: Replica Location Index (RLI) and Replica Metadata Catalogue (RMC) used for collecting data from LRC of all sites [45],

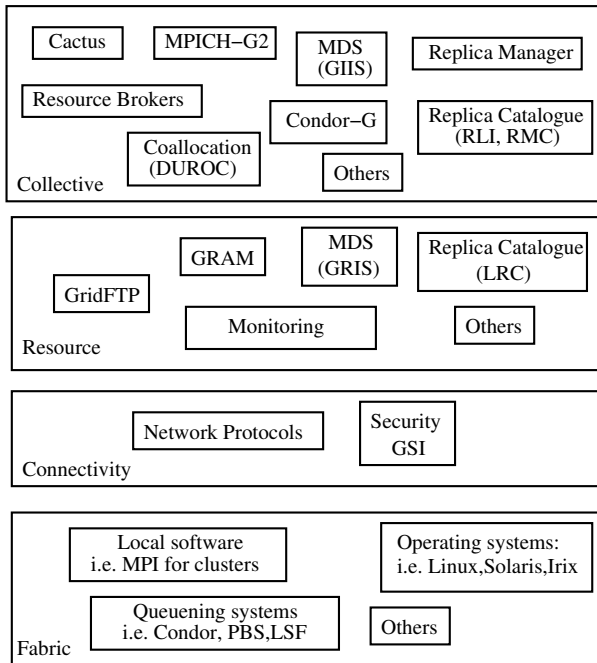


Figure 2.14: First version of Globus architecture [70], consists of four layers: fabric, connectivity, resource and collective.

- DataGrid Replica Manager [45] that performs replication of files basing on Replica Catalogue information using GridFTP
- Grid Version of the Condor queue system (Condor-G) [166]

Furthermore, this layer contains support for the application developer, such as the Cactus Problem Solving Environment described in detail in Section 2.1.5, and the Grid version of MPI (MPICH-G2) [116]. This layer can also contain application-specific distributed services .

The next step in Grid vision was an attempt to define standard "InterGrid" protocols to allow different distributed systems to interoperate. Such was the motivation for adapting Web services concepts [178] and Service Oriented Architecture (SOA) to the Grid.

The Web services concept is a modern solution stemming from the industry. It allows the publishing of service functionality, so it can be found by clients and used without bothering about the actual technologies a service is built with, provided that it exposes its interface as a so-called Web service interface. Original Web services are stateless – for the same arguments they always return the same results – there are no internal states that can influence the answer.

The Open Grid Services Architecture (OGSA) and the underlying Open Grid Services Infrastructure (OGSI) [61] were the first results of merging Web services and Grid concepts. The resulting Grid Architecture is depicted in Fig. 2.15. As can be

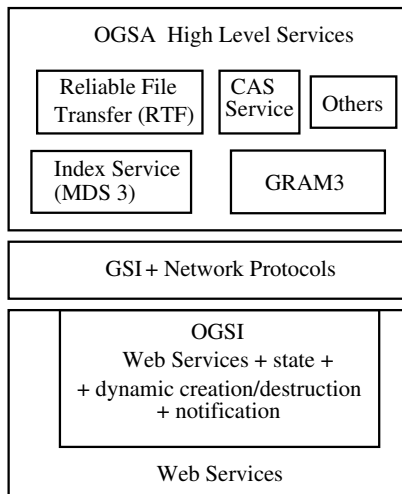


Figure 2.15: OGSI-based Grid architecture [61] introduced the concept of extending Web services to the transient and stateful Grid services with a notification mechanism. The aim of OGSA is to define a set of higher-level services built over OGSI.

seen, OGSI introduced the concept of extending Web services to the transient and stateful Grid services with a notification mechanism. The aim of OGSA is to define a set of higher-level services, among which are those with the same functionality as in the original Globus Toolkit version 2 - for data management Reliable File Transfer (RFT), resource management (GRAM3) and information management (i.e. MDS3), but implemented as a Grid services version (Globus Toolkit version 3). There is also a high-level security service called the Community Authorization Service (CAS). The list of OGSA GT3 services can be found in [70].

Recently, the OGSI authors have changed the idea of Grid services to the Web Services Resource Framework [180]. The main idea remains the same, that is to converge concepts of Grid and Web services, but in a slightly different way. In order to add state to the Web service, instead of directly extending Web services, one has to model a Stateful Resource assigned to that service as shown in Fig. 2.16. In this way, the classical Web services can support features similar to those defined by OGSI. The concept of the upper level (OGSA) does not change. The full list of OGSA Services implemented in WSRF can be found in [70].

Apart from work of the Globus Community, there are also other approaches to Grid computing. The Grid also can be built using pure Web services or a Components Environment e.g. the Common Component Architecture [31], H2O [96], ProActivecite [133] or MOCCA [108] which is a CCA-compliant framework for the H2O resource

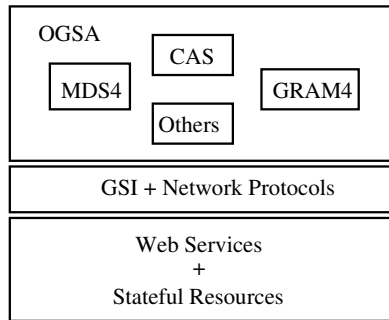


Figure 2.16: WSRF-based Grid architecture [180] – instead of directly extending Web services, one has to model a Stateful Resource assigned to that service

sharing platform. CoreGRID [36] is a sample project aiming at building the Grid from components.

It is also worth to mention Virtual Organisation (VO) as one of the most important concepts introduced by Grid researchers [62]. VO is the basic unit of Grid users organisation; it is a set of individuals and/or institutions which are related to each other by some level of trust and rules of sharing the resources, services, and applications.

Note on the evolution of Grid technology

The research described in the thesis started in October 2001. It was also partly supported by CrossGrid project that started in March 2002 and the Virtual Laboratory Project that started in January 2004. Grid concepts and technology have evolved significantly since then. Initially, the mainstream Grid technology was based on Globus Toolkit version 2. At the beginning of year 2002, new concepts of using Web Services technology in Grids emerged. The results have converged into Globus Toolkit version 3 with OGSA and OGSi specifications, where the key role was played by the concept of the Grid service – a Web service extended with a state. OGSi was chosen as a platform for the system presented in this thesis. In the meantime, at the beginning of 2004, OGSi authors change this concept into the Web Service Resource Framework (WSRF) [180] that became the successor of OGSi. Due to the fact that WSRF was still at a very early stage of development when this thesis was being written and that the Grid technology changed very dynamically, we decided to stay with the original OGSi/OGSA concepts. However, the system allows easy migration to WSRF, if necessary in the future. The results of this thesis are independent of underlying technology (OGSi or WSRF).

The Grid is a promising approach for various applications. Its idea seems to be most suitable for projects that aim to develop Grid infrastructure for compute- and data-intensive applications. Apart from that, there is also the concept for using Grid for distributed simulations, which is not a trivial issue. Below, we present the various approaches to that concept.

2.3.2 CrossGrid

Project Goals

The CrossGrid Project [38] is oriented towards compute- and data-intensive simulations which are characterized by the interaction with a person in a processing loop. Each of CrossGrid applications require a response from the Grid to an action by a human of the different time scales. These applications are: simulation and visualization for surgical procedures, flooding crisis team decision support system, distributed data analysis in high-energy physics (HEP) and air pollution combined with weather forecasting.

Functionality

The medical application is a distributed near-real-time simulation with a user interacting in virtual reality (VR) or other interactive display environments. A 3-D model of arteries is the input to a blood flow simulation. The medical application requires a distributed environment consisting of simulation, interaction and visualization components which will allow the user to change simulation parameters in near-real time [155]. The application uses the Grid Visualization Kernel (GVK) [94] as a component that connects simulation with the visualization. The solutions presented in this thesis were mainly driven by the requirements of the medical application.

Flood forecasting starts with meteorological simulations conducted at different resolutions, from mesoscale to storm-scale. Selected hydrological models are then used to determine water discharges from the affected area, and with this information hydraulic models simulate flow through various river structures. An interactive Grid system fulfilling the needs of this application allows experts to prepare and control the cascades of meteorological, hydrological and hydraulic simulations basing on the assumption that each preceding step of the cascade produces input for the next simulation. The details can be found in [79].

Distributed data analysis in HEP addresses access to large distributed databases in the Grid environment and development of distributed data mining techniques suited to the HEP field. Data mining services based on supervised and unsupervised learning with processing on worker nodes and on the database side are being elaborated. HEP and air pollution modeling applications require support from a Grid interactive system that allows for on-line progress monitoring of their results in order to help operators decide about further job execution (e.g. interrupting the execution or letting it finish). In this application interactivity is realized using CrossGrid middleware that, in turn, uses Condor Bypass [25] solution.

To summarize, three types of interactive simulations were defined in CrossGrid - firstly, near-real-time simulation with a user in the loop, next a cascade of simulations and, last but not least, on-line progress monitoring. These applications pose specific requirements for the CrossGrid architecture which is shown in Fig. 2.17 and described in [24]. The requirements are addressed by the Grid services developed in CrossGrid.

Fig 2.17 shows how the CrossGrid applications, tools and services communicate

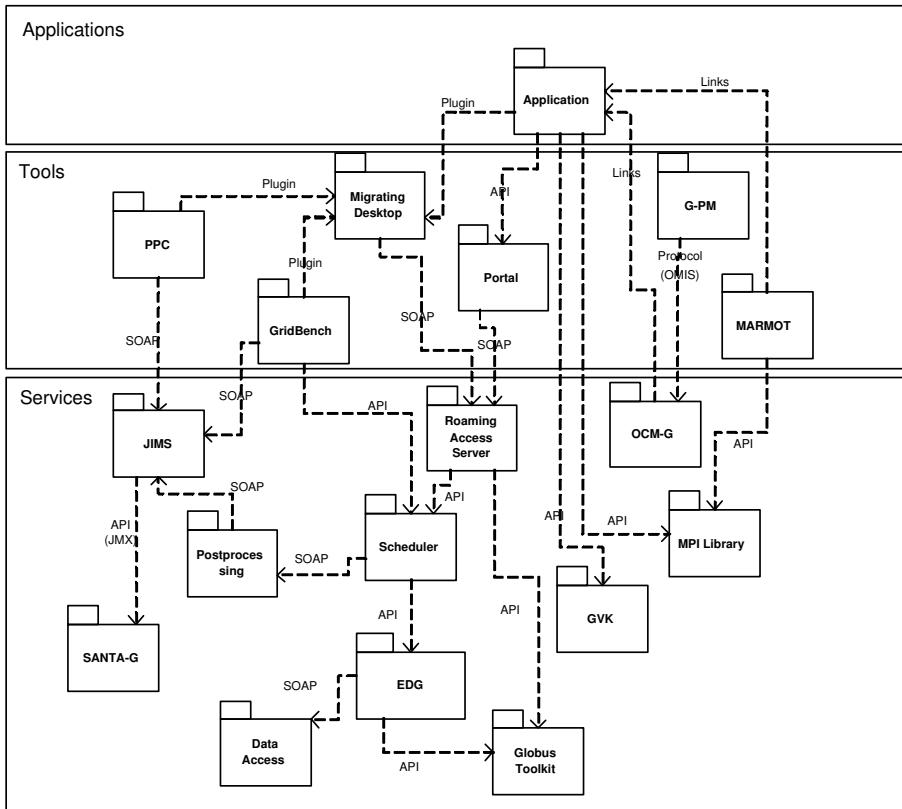


Figure 2.17: The CrossGrid architecture [38] consisting of three layers – applications, tools and services.

with each other. The arrow directed from an element A to B means that A uses B. Simple Object Access Protocol (SOAP) [157] indicates Web services interface, Application Programmer Interface (API) indicates usage of library routines, JMX means use of Java Management Extension technology [87], OMIS is Online Monitoring Interface Specification protocol [103], *links* means that a tool is linked to an application, *plugin* is used to extend Migrating Desktop with an application- or a tool-specific user interface.

User-friendly access to Grid resources for remote users is handled by the Migrating Desktop together with the Roaming Access Server and schedulers adapted for interactive applications. Grid monitoring services provide information on infrastructure execution – for that purpose the JMX infrastructure Monitoring System (JIMS) and Grid-enabled System Area Networks Trace Analysis (SANTA-G) were developed. Also, there is application execution monitoring called Grid-enabled OMIS-Compliant Monitor (OCM-G). Data Access services are used to optimize the time needed to trans-

fer large data files from tertiary storage to computing nodes. Data Access are used by European DataGrid (EDG) data management solutions, which, in turn, are used by Scheduler. The creation of interactive Grid applications also calls for specialized tools that facilitate the process of code development and optimization. Such tools, perfected within CrossGrid, are Marmot for MPI verification, Performance Prediction Component (PPC), Grid Benchmarks for testing the behavior of applications in a Grid environment, a performance analysis tool (G-PM) that can perform on-line measurements and a desktop for interactive use as a service.

2.3.3 Reality Grid

Project Goals

RealityGrid [137] proposes to extend the concept of a Virtual Reality centre across the Grid and links it to massive computational resources at high performance computing centres and experimental facilities using Grid technology to closely couple high throughput experimentation and visualization.

Functionality

RealityGrid middleware, called WSRF-based Environment for Distributed Simula-

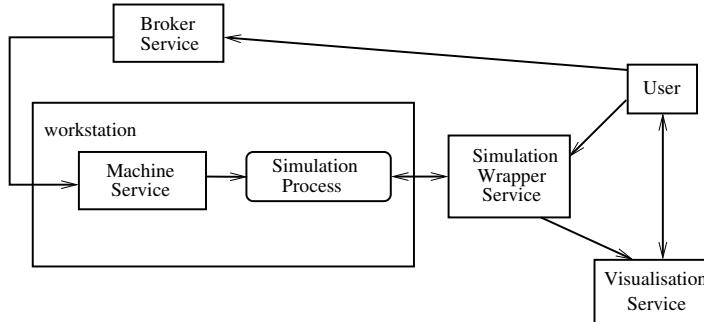


Figure 2.18: Architecture of WEDS [37], designed to let scientists remotely deploy instances of a pre-existing codes across multiple resources and giving steering, visualisation and workflow functionality.

tion (WEDS) [37], is a WSRF-based middleware scheme designed to let scientists remotely deploy single or multiple instances of a pre-existing codes across multiple resources and giving steering, visualisation and workflow functionality with only simple modifications to program code. WEDS is built around four key WSRF services. Machine Services represent individual machine resources: they make available information about machine specification and load. Wrapper Service represents a single simulation. File Service runs on the user machine and serves requests for input files. Broker Service acts as a registry, linking together all of the above services and storing

their addresses. The conceptual architecture can be found in Fig. 2.18. A user uses the Broker Service to set up the simulation. Machine Service then launches the simulation process and the user can interact with a Wrapper Service of this simulation. The output data is sent to the Visualisation Service and can also be modified by the user.

2.3.4 GEMSS

Project goals

The GEMSS Project [17] is concerned with the creation of medical Grid service prototypes and their evaluation in a secure service-oriented infrastructure for distributed on-demand supercomputing - the GEMSS testbed. The medical prototype applications include maxillo-facial surgery simulation, neuro-surgery support, radio-surgery planning, inhaled drug-delivery simulation, cardiovascular simulation and tomographic image reconstruction.

Functionality

The GEMSS middleware exposes medical simulation applications installed on various

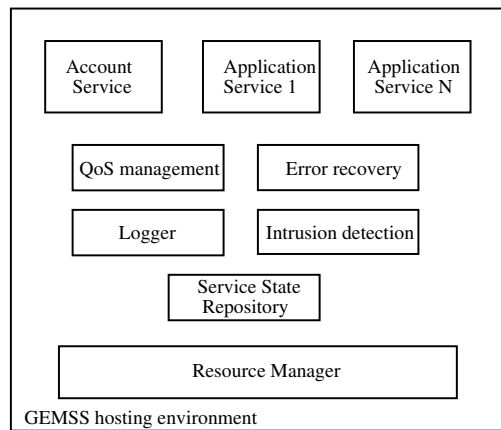


Figure 2.19: The GEMSS [17] middleware exposes applications installed on various Grid hosts as services which support a common set of methods for data staging, remote job management, error recovery and QoS support.

Grid hosts as services which support a common set of methods for data staging, remote job management, error recovery and QoS support. The architecture of the GEMSS server is shown in Fig. 2.19. Medical simulation applications are exposed as Web services. The quality of service management component handles reservation with the resource manager (job scheduler) and provides input to the quality of service

negotiation process. In order to compute various QoS properties (e.g. service completion time) on the basis of the specified performance parameters, a machine-specific performance model has to be provided. This model is derived from a database relating typical problem parameters to resource needs such as main memory, disk space and running time. The database stores data from test cases. The error recovery component handles checkpointing and re-starting of services if required. The logger component manages a database for logging information and performs level event logging for intrusion detection. The service state repository component manages a conversational state database that contains information about any client-service conversation allowing it to be resumed at a later date.

2.3.5 Virtual Laboratory

Project Goals

The Virtual Laboratory environment (VL) [172] provides a framework for groups of scientists, engineers and scientific organizations that interact and cooperate with each other towards the achievement of a common experiment. Such an experimental environment enables researchers at different locations to work in an interactive way, as in any laboratory, i.e. the scientists are able to create and conduct the experiments in the same natural and efficient way as if they were in their own laboratory.

One of the most important characteristics of the experimental domains is the manipulation of large data sets produced by the experiment devices. To be able to handle the resulting experiment data sets, three main requirements are supported within the VL architecture:

- Proper management of large data sets: e.g. storage, handling, integration, and retrieval of large data sets.
- Information sharing and exchange for collaboration activities: scientists are able to share both the devices used to perform the experiments and the data sets generated by those experiments. They must also be able to look at these data sets and compare them to the ones from previous experiments or other public databases, in order to find similarities and patterns.
- Distributed resource management: the management of resources must be properly considered in order to meet the high performance and massive computation and storage requirements.

Functionality

The VL architecture is shown in Fig. 2.20 - its middleware enables users to access low-level distributed computing resources by following components:

- The Virtual Lab Information Management for Cooperation (VIMCO) component provides the functionality to store and retrieve both large data sets and data

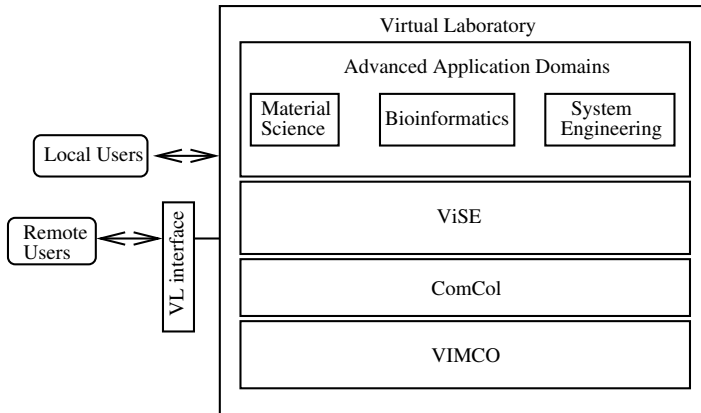


Figure 2.20: VL architecture enables researchers at different locations to work in an interactive way, as in any laboratory.

analysis results, advanced functionality for intelligent information integration and facilities for information sharing.

- The Communication and Collaboration (ComCol) component provides the appropriate mechanisms for data and process handling based on the Grid technology.
- The Virtual Simulation and Exploration Environment (ViSE) component offers a generic Virtual Simulation and Exploration environment where 3D visualization techniques are provided to analyze large data sets.

The VL middleware enables VL users to access low-level distributed computing resources. This is achieved by providing a VL user interface which enables scientists to define and execute the experiments.

2.3.6 DSGrid

Project Goals

The DSGrid [51] project focuses on large-scale distributed simulation on Grids. The goal is to build a distributed collaborative simulation environment where researchers with different domain knowledge located at different places develop, modify, assemble and execute distributed simulation components over the Grid.

Functionality

The project aims at: support for collaborative development of simulation applications, mechanisms for fault tolerant, coordinated, secured simulation executions, advanced model and service discovery mechanisms, novel resource management and load balancing mechanisms to meet the differing requirements of various models etc. The

project is very promising, but relatively young (started in May 2004) and at the time of writing this thesis is at early stage of development.

2.3.7 Summary

This Section describes projects involved in the area of interactive distributed simulations on the Grid. There are obviously many other projects, as can be found on [35] or [170]. We have chosen those related to interactive distributed simulations and described above how this interactivity is realized. The project summary and comparison is presented in Tab. 2.3. CrossGrid is an application-driven project providing four interactive applications as well as various tools and services for their development and execution. CrossGrid is based not only on Globus software, but also on DataGrid and EGEE. GEMSS is also application-driven – it created medical Grid service prototypes in Web services technology. DSGrid, focused more on middleware, plans to build a distributed collaborative simulation environment for researchers based on Grid and Web services technologies. The aim of Reality Grid is to create a Grid-based VR system built over Web services. Finally, Virtual Laboratory provides a framework for groups of users (scientists, engineers etc.) to interact and cooperate with each other to conduct common experiments. All projects are complementary and introduce useful ideas for interactive and distributed simulations, which we would like to extend. Since we have chosen HLA as our basic middleware, in the next Section we will take a more detailed look at the work related to the aspects of merging Grid and HLA ideas.

2.4 HLA in Grid and Web services environments

Basing on analysis in Section 2.1 we have chosen the High Level Architecture (HLA) as middleware which can take advantage of the Grid to form an advanced environment for distributed interactive simulations. Therefore, in this Section we analyze work related to merging HLA and Grid concepts. Since HLA is a well-known standard in the area of distributed and interactive computing, a lot of effort is being devoted to taking advantage of the emerging Grid and Web solutions in this field. The complementary issues in this matter are: interoperability of different simulation models, fault tolerance, support for peer-to-peer collaborative applications (e.g. games) and effective management of HLA-based simulations. This thesis focuses mostly on the last issue, but the approaches to the related problems are discussed below as well.

2.4.1 Towards interoperability of distributed interactive simulations

The modeling and simulation community has realized that there is no widely accepted standard that can handle both DOD Modeling and Simulation protocols such as HLA [78] and Web/IT standards [178]. It was also recognized that Web-based technologies

Name	Applications	Goals	Grid technology used
CrossGrid	surgical planning, flood crisis team support, distributed data analysis in HEP, air pollution, weather forecasting	to provide various tools and services for execution and development interactive applications on the Grid	Globus v2, DataGrid software, EGEE
Reality Grid	VR applications	to extend the concept of a Virtual Reality centre across the Grid and link it to massive computational resources at high performance computing centers and experimental facilities.	WSRF
GEMSS	The medical prototype applications: maxillo-facial surgery simulation, neurosurgery support, radiosurgery planning, inhaled drug-delivery simulation, cardiovascular simulation and tomographic image reconstruction	to create medical Grid service prototypes and their evaluation in a secure service-oriented infrastructure for distributed on-demand supercomputing	Web and Grid services
VL	material science, bioinformatics, system engineering	to provide a framework for groups of scientists, engineers and scientific organizations that interact and cooperate with each other towards the achievement of a common experiment	Globus v2, DataGrid
DSGrid	distributed simulations in general	to build a distributed collaborative simulation environment where researchers with different domain knowledge and expertise, at different locations develop, modify, assemble and execute distributed simulation components over the Grid.	Web and Grid services

Table 2.3: Comparison of the interactive grid computing projects

can provide an Extensible Modeling and Simulation Framework (XMSF), to support a new generation of interoperable applications. What is more, it was noticed [186] that current solutions in HLA implementations, such as DoD HLA-RTI, do not perform efficiently in Wide Area Networks and that traditional approaches to high-performance

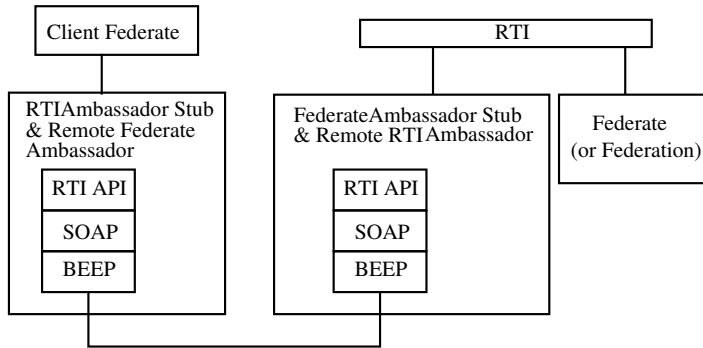


Figure 2.21: Architecture of web-enabled RTI introduces a Web service wrapper over existing RTI functionality.

RTI implementations assume relatively static configurations of federates which are not sufficient for the highly dynamic nature of XMSF. This was the motivation behind the XMSF project [185, 134] aiming to develop a common standard for Web and DOD Modeling and Simulation. As a prototype, a web-enabled RTI has been developed [185], being basically a Web service wrapper over existing RTI functionality as shown in Fig. 2.21. The only difference between this and classical Web service is the usage of the BEEP [185] protocol that enables bi-directional calls and callbacks via the RTI. Web-enabled RTI serves as an example of creating more general XMSF profiles which are planned as documents describing a set of protocols, data and metadata standards used for the application domain and a detailed description of applying protocols and data standards to implementing architecture. Unfortunately, the final specification of XMSF profiles has not yet been defined.

In parallel, research effort is ongoing within the Commercial Off-The-Shelf Simulation (COTS) Package Interoperability Forum (HLA-CSPIF) [41, 165]. As there is no real standard use pattern for the High Level Architecture within the context of this area, the goal of the Forum is to create standards through SISO [152] that will facilitate the interoperation of COTS simulation packages and thus present users of such packages with the benefits of distributed simulations enjoyed by other modeling and simulation communities.

Additionally, interoperability of simulations is also going to be included in the system mentioned in [194] – more details can be found in Section 2.4.4.

2.4.2 Fault tolerance

The Decentralized Resource Management System (DRMS) [53] is a scalable and fault-tolerant framework for HLA-based applications. DRMS is a JXTA-based peer-to-peer system for execution of HLA (High Level Architecture) federations in a decentralized environment. The main advantage of DRMS is that no single point will cause the system to fail, as would be the case in a centralized approach. In the DRMS all

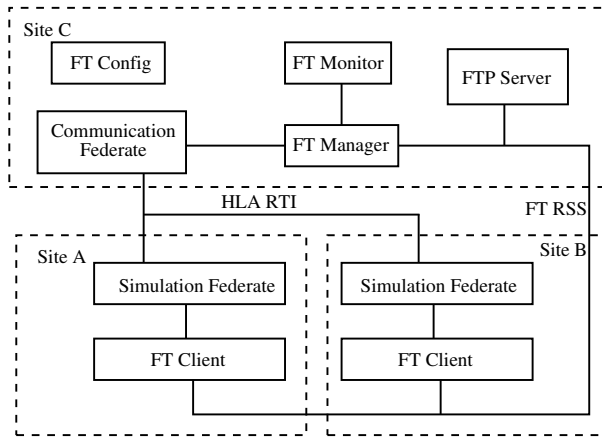


Figure 2.22: Architecture of FT-RSS allows the individual configuration of FT mechanisms to be included in federates and federations.

participating peers are equal in terms of functionality and federates are not stored in a central repository. No matter which or how many peers are joined at a certain moment, a basic level of service will always be upheld.

Work on fault tolerant issues for HLA can be also found in [86], where authors propose the fault tolerant resource sharing system (FT-RSS) for HLA-based simulations shown in Fig. 2.22. The framework and its components allow the individual configuration of FT mechanisms to be included in federates and federations. In the design and implementation of a fault tolerant federation, the developers are supported by an FT configuration tool. During the execution of a distributed HLA simulation using the FT-RSS, the manager and client components of the FT-RSS are responsible for the enforcement of FT mechanisms. A special communication federate is used for sending HLA-specific information from simulation to the manager

2.4.3 Peer to peer collaborative applications

There is also work on another approach that uses the HLA concept to build a scalable peer-to-peer infrastructure. [161] proposes FederationGrid (FedGrid), implementing a scalable Grid supporting real-time collaborative applications. FedGrid is build over the standard High Level Architecture (HLA) and is based on the concept of a fractal Grid, comprised of hierarchical HLA federations. As shown in Fig. 2.23, a *System Federation* is essentially a federation of *Systems* each of which is again a federation of *Engines* communicating using an internal RTI. Federation Grid Engines are implemented using MAGNETAR (Metaprogrammable AGent NETwork ARchitecture). The structure of a MAGNETAR Engine is a single process federation, which uses an internal HLA infrastructure Engine (that acts as a blackboard) to provide communication between agent plugin components.

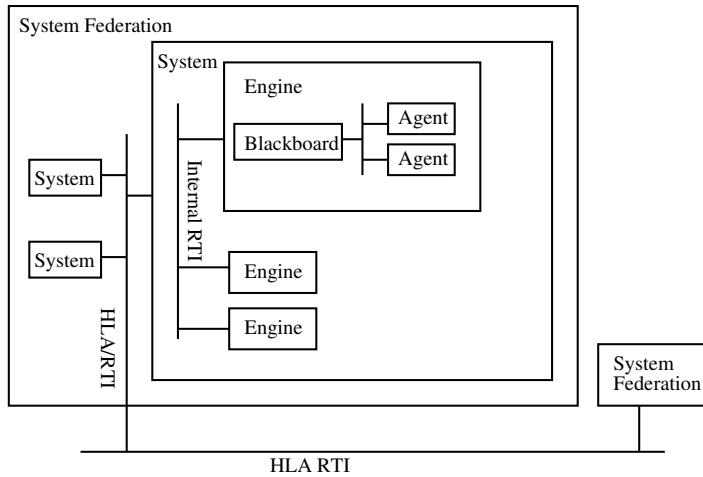


Figure 2.23: Architecture of FederationGrid based on the concept of a fractal Grid, comprised of hierarchical HLA federations.

2.4.4 Management of HLA-based simulations on the Grid

The research which is the most closely related to the subject of this thesis involves effective management of HLA-based simulations in the Grid environment.

The design and implementation of a load management system for running large-scale HLA simulations in a Grid environment based on Globus Toolkit 2 is described in [26]. The authors also present a framework where the modeler can design parallel and distributed simulations with no knowledge of HLA/RTI [188]. In [27] a protocol is presented which supports efficient checkpointing and federate migration for dynamic load balancing. In order to facilitate the use of that protocol, an additional layer between HLA and the actual application is proposed [188]. Although this approach presents a more efficient migration protocol, it is not sufficient for porting HLA legacy code to the Grid.

In [194] a framework for executing large-scale distributed simulations using Grid services is presented. Currently, the framework addresses dynamic discovery of HLA federates. The approach assumes that each federate is a simulation model encapsulated in a Grid service. Fig. 2.22 shows two example simulation models A and B encapsulated in Grid services that are registered in an Index Service together with the RTIExec Service. Following dynamic discovery of RTIExec Service, the simulation is set up and runs using HLA RTI communication.

The presented solution is complementary to the approach taken in this thesis. In [194] there is an assumption that the user is able to build his simulation from existing Grid services (that encapsulate functionality of HLA federates) or can wrap his federates into Grid services (the paper does not describe how to wrap a federate into a Grid service, which is not a trivial issue). On the other hand, this thesis presents a

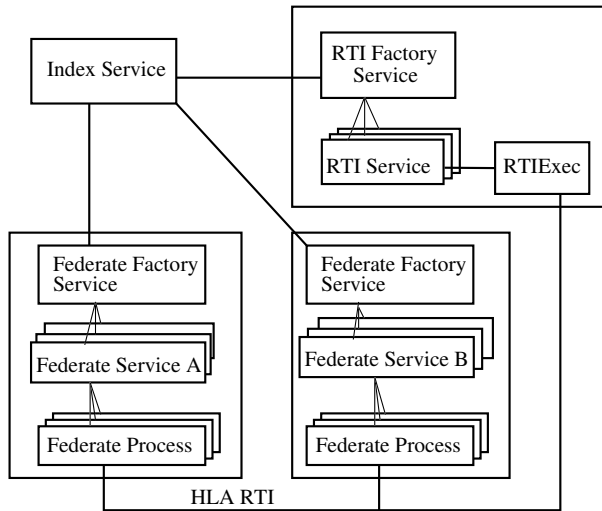


Figure 2.24: Architecture of a framework for large-scale distributed simulations with Grid services – simulation models are encapsulated in Grid services. Following dynamic discovery of RTIExec Service, the simulation is set up and runs using HLA RTI communication.

system that allows a user to simply load his legacy federates into general-purpose so-called *HLA-Speaking* Grid services and manage execution of the simulation. There is no doubt that both approaches are important depending on the simulation requirements.

2.4.5 Summary

The Grid environment provides new possibilities for HLA-based distributed interactive simulations. In this Section, we presented different aspects of applying Grid solutions to the field of modeling and simulation including interoperability of simulations, fault tolerance, building collaborative environments, dynamic application setup and execution management. The issues are summarized in Tab. 2.3. The diversity of work in this field shows that using Grid for modeling and simulation is not a trivial task. On one hand, Grid and Web services provide a possibility for achieving simulation interoperability and for building collaborative environments. It should be noted, however, that executing simulations on the Grid requires dynamic setup, execution management and fault tolerance. According to our central thesis formulated in Section 1.4, our aim is to show that there exist solutions for achieving effective simulation execution on the Grid. Therefore, we focus mainly on the last of the mentioned aspects (execution management). Nevertheless we also describe a mechanism that allows for extending our system to support fault tolerance.

Issue	Solution
simulation interoperability	developing common standards by taking advantage of Web/Grid services which were designed for interoperability
fault tolerance	1) peer-to-peer environment with redundant peers, 2) tools and methods for error detection, error processing and migration
building collaborative environments	a fractal Grid, comprised of hierarchical HLA federations
dynamic setup and management	load management, migration mechanisms, dynamic discovery

Table 2.4: Different aspects of applying Grid to modeling and simulation

2.5 Summary and conclusions

This Section presented various approaches of porting distributed interactive simulations to the Grid. We started with an overview of existing environments supporting such applications, next we presented solutions for efficient execution of High Performance Computing simulations in a Metacomputing Environment, which was the first vision of the Grid. Finally, we presented current research on distributed interactive simulations on the Grid. We also described the CrossGrid project, where the requirements of a medical application provided motivation for this thesis. We mentioned DS-Grid, PowerGrid and GEMSS initiatives in the field of distributed interactive simulations. Subsequently, we described efforts that most closely relate to the subject of this thesis, which is to use the possibilities offered by the Grid in HLA-based interactive simulations. We described the most significant issues in that area: interoperability of different simulation models, fault tolerance, support for peer-to-peer collaborative applications and, lastly, effective management of HLA-based simulations.

The analysis presented in this Chapter allows us to find the most urgent open problems in the field of distributed interactive simulations running on the Grid. It outlines existing solutions and their drawbacks and also helps us choose the best environment that fits our requirements in order to provide a starting point for our research.

To the best of our knowledge, none of the presented solutions focus on efficient execution of legacy HLA simulations on the Grid, which is our primary aim, and therefore they do not fully cover problems and solutions presented in this thesis.

Through comparing different environments supporting distributed and/or interactive simulations described in Tab. 2.1 we have shown, that HLA is a good choice as a basis for distributed simulations because it is a standard and because it possesses ad-

vanced features: scalability, data distribution management, synchronization and the ability to connect simulations with time management. What's more, the related work presented in this Chapter shows that running distributed simulations on the Grid is not a trivial problem and introduces various challenges which need to be overcome. The problems have matured along with the vision of Grid - from efficient running of HPC simulations in a metacomputing environment to advanced Grid projects. Open issues related to combining HLA and Grid concepts include interoperability, development of collaborative environments, fault tolerance and execution management. We have therefore chosen one of the most urgent problems - we focus our work on efficient execution of legacy HLA simulations on the Grid to allow them to use distributed resources in a more effective and convenient way.

Grid HLA Management System

In this Chapter¹ we present the concept of the Grid HLA Management System (G-HLAM) designed to satisfy the requirements presented in Section 1.3 – *efficient execution of scalable HLA-based distributed interactive simulations on the Grid with support for legacy systems*. The motivation and the location of G-HLAM in the context of Grid architecture was shown in Fig. 1.4 in Section 1.4. The objectives of this Chapter are: to describe the functionality of G-HLAM and its modules, to show how an HLA-based application is automatically set up with G-HLAM using those modules and to analyze the system's functionality using a Petri Net [128]. in particular – to check if the system is *deadlock-free*. We also outline the background of G-HLAM in more detail - namely we include a more detailed description of HLA than the one present in Section 2.1.8, since it is our middleware of choice.

This Chapter is organized as follows: in Section 3.1 we describe the HLA and Service Oriented Architecture initiatives that were the basis for design and development of G-HLAM. In Section 3.2 the concept, functionality and architecture of the G-HLAM system are presented as well as the setup of the whole HLA-based application in this system. Finally, Section 3.3 presents Petri Net-based analysis of the system to prove that it behaves correctly and also that deadlocks do not occur.

The solution presented in this thesis allows not only for building HLA-based applications that can be executed on the Grid, but also to easily adapt HLA legacy code. This approach cannot be found in any of the current solutions described in Chapter 2.

¹The results described in this Chapter formed the basis to the paper K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Framework for HLA-Based Interactive Simulations on the Grid. *SIMULATION*, 81(1):67–76, 2005.

3.1 Background

As already mentioned in the previous Chapter in Section 2.1.13, we have chosen the High Level Architecture standard as support for distributed interactive simulations that will run on the Grid. In this Section we include a more detailed description of the standard than the one presented in the Section 2.1.8. HLA is a mature distributed simulation framework, with support for synchronization of simulations components (*time management*), efficient *data distribution management* and the ability to connect simulations using different time management techniques into one, coherent system. Therefore, in our work we have decided to focus on simulations based on this standard. Additionally, we think that the Grid environment offers new possibilities for such simulations by fulfilling their requirement for the ability to access distributed resources. Despite the advantages mentioned above, running HLA-based simulation on the Grid requires additional support for its efficient usage. This was the motivation behind developing the G-HLAM system, which is the subject of this thesis.

3.1.1 High Level Architecture

As already mentioned in Section 2.1.8, HLA is a standard for distributed interactive computing. In this Section we provide a brief overview of its functionality and present basic components of its reference implementation.

HLA basic functionality

The HLA Runtime Infrastructure (RTI) federations [78] are distributed systems that consist of multiple processes (federates) communicating across computer nodes.

As already described in Section 2.1.8, the most important services provided by RTI to support interactive applications are *data distribution management* in charge of efficient routing of data between federates, *time management* controlling message ordering, and *ownership management*, transferring ownership that controls rights to changing attributes of objects exchanged between federations.

The RTI acts as a *tuple space*: in order to send messages, the applications that are plugged into the RTI have to publish well-defined objects in that space. The applications which want to receive messages have to subscribe to those objects. Each time an object is to be sent, the application must call a method that updates the attribute values of this object; RTI then notifies the subscribed applications that the object has been updated. Events are similar to data objects. The difference is that HLA does not distinguish between attributes of events (they are "all or nothing"). As an example, Fig. 3.1 shows the sequence of calls required to transmit data objects between two applications. The arrows indicate the direction of control flow between the RTI and the applications. Before actually sending an object of a given class, Federate A has to publish the class in the RTI tuple space using the `publishObjectClass` function. Federate B, wishing to receive objects, calls the `SubscribeObjectClassAttributes` routine which triggers the RTI to call the

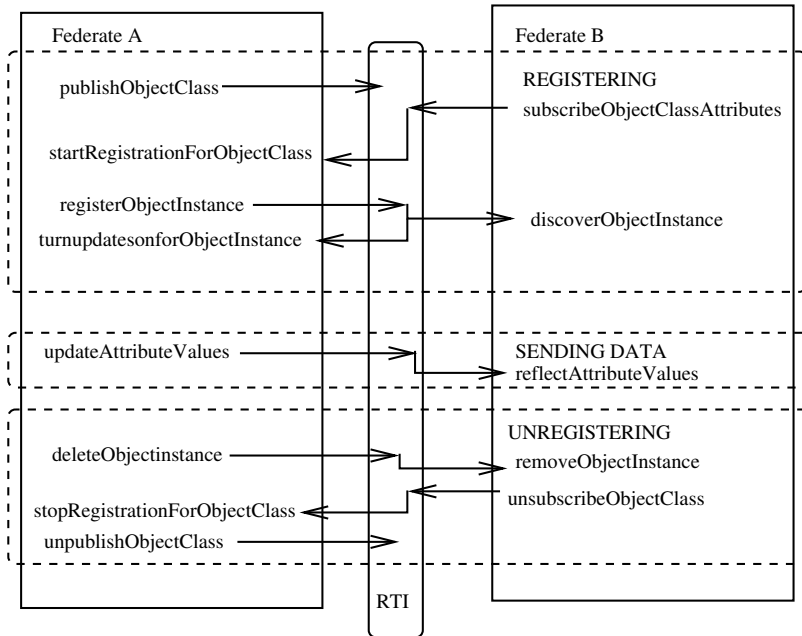


Figure 3.1: Exchanging data objects between HLA federates within an RTI tuple space. The arrows indicate the direction of control flow between the RTI and the applications.

`startRegistrationForObjectClass` callback in Federate A. This callback notifies Federate A about subscription. Subsequently, Federate A can register objects of the published class and Federate B discovers them by means of a callback mechanism. Actual data are sent with the `updateAttributeValues` function and received with the `reflectAttributeValues` callback. When no longer needed, the object can be unregistered with `deleteObjectInstance` and its class can be unpublished with `unpublishObjectClass`.

Basic components of the HLA reference implementation

The RTI implementation developed by the U.S. Department of Defense (DoD) [77] is currently comprised of the main RTI Executive process *RtiExec*, the Federation Executive process *FedExec* and the libRTI library *libRTI*. As illustrated in Fig. 2.8, each executable containing federates incorporates libRTI. The *RtiExec*'s primary purpose is to manage the creation and destruction of *FedExecs*. An *RtiExec* directs the joining of federates to the appropriate federation execution and ensures that each *FedExec* has a unique name. The network address specified by the hostname or the Internet Protocol address of that machine plus the port number used, known as the endpoint of the process, is defined in a federate configuration file. If the endpoint is not specified for the *RtiExec*, a multicast protocol is used for finding the RTI control process.

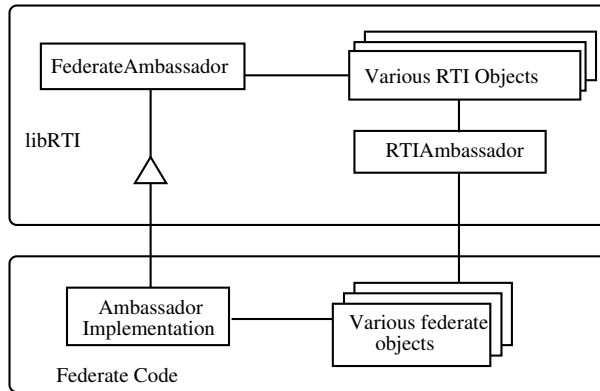


Figure 3.2: The HLA object model – federates use *libRTI* (which communicates with the *RtiExec*, a *FedExec*, and other federates) to invoke HLA services.

Each *FedExec* manages a federation. It allows federates to join and quit, and facilitates data exchange between participating federates. It is created automatically with the federation. *libRTI* provides the RTI services specified in the HLA Interface Specification to federate developers. The class diagrams in Figure 3.2 illustrates RTI and federate core responsibilities. Federates use *libRTI* (which communicates with the *RtiExec*, a *FedExec*, and other federates) to invoke HLA services. The HLA Interface Specification identifies the services provided by *libRTI* to each federate and the obligations each federate has to the federation. Within *libRTI*, the class *RTIAmbassador* bundles the services provided by the RTI. All requests made by a federate on the RTI take the form of an *RTIAmbassador* method call. The abstract class *FederateAmbassador* identifies the callback functions each federate is obliged to provide.

3.1.2 Service Oriented Architecture

The design of Grid HLA Management System is based on the concept of a Service Oriented Architecture (SOA). We have chosen SOA because this kind of architecture provides a powerful and easy way to integrate parts of the system which supply various functionalities. Furthermore, it is easier to integrate the system with existing solutions without bothering about technologies they are build on, provided that they expose their interfaces as Web service interfaces.

The SOA is an architectural style that aims to achieve loose coupling among interacting software components. A service is a unit of work done by a service provider to achieve the desired end results for a service consumer as shown in Fig. 3.3. The idea of SOA differs significantly from that of object-oriented (OO) programming, which suggests binding data and its processing together. The SOA achieves loose coupling by employing two architectural constraints [74]:

- A small set of simple interfaces for all participating software components. Only

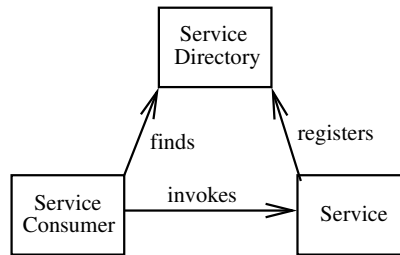


Figure 3.3: Service Oriented Architecture have a mechanism enabling a consumer to discover a service provider under the context sought by the consumer.

generic semantics are encoded at the interfaces. The interfaces should be universally available for all providers and consumers.

- Descriptive messages constrained by an extensible schema delivered through the interfaces. Minimal system behavior (or even none of it) is prescribed by messages. A schema limits the vocabulary and structure of messages. An extensible schema allows new versions of services to be introduced without breaking existing services.

Additionally, the SOA must have a mechanism enabling a consumer to discover a service provider under the context of a service sought by the consumer as shown in Fig. 3.3.

There are also some additional issues regarding SOA. One of the most important issues is the question whether the services should be stateful or not.

In the case of a *stateless* service, each message that a consumer sends to a provider must contain all the necessary information for the provider to be able to process it. This constraint makes the service provider more scalable because the provider does not have to store state information between requests. Each request can be treated as generic. This constraint improves visibility because any monitoring software can inspect one single request and figure out its intention. There are no intermediate states, so recovery from partial failure is also relatively easy. This makes the service more reliable.

On the contrary, *stateful* services require both the consumer and the provider to share the same consumer-specific context, which is either included in or referenced by messages exchanged between the provider and the consumer. For example, stateful services could be useful in a banking system, where sending a security certificate with each request is a serious overhead for both the consumer and the provider. It is much quicker to replace the certificate with a shared token. The drawback of this constraint is that it may reduce the overall scalability of the service provider because it may need to remember the shared context for each consumer. It also increases coupling between the service provider and the consumer and makes switching service providers more difficult.

The design of our system is based on the Service-Oriented Architecture with stateful services defined by the Open Grid Service Architecture described in Section 2.3

3.1.3 Possibilities and limitations for HLA-based simulations on the Grid

As described in Section 1.2.2 and Section 2.1.13, we have decided to choose the HLA [78] as a framework for developing interactive steering, because it provides simulation developers with many useful features needed by time- or event driven- distributed simulations and it satisfies some of the requirements listed in Section 1.3. However, the HLA standard was developed assuming a certain quality of service in the underlying environment (in our case – the Grid) for simulation execution.

As described in Section 2.3.1, the Grid [58] is designed to coordinate resources that are not under central control. Additionally, the Web services [178] concept of abstract interfaces allows for modular design (OGSA) [61]. However, the Grid environment is shared between many users and its conditions can change in an unpredictable way. Therefore, there is a need for a system that adapts HLA-based applications to a dynamically-changing environment and requires fault tolerance mechanisms such as migration of its distributed federates or their monitoring.

	HLA	Grid
Possibilities	used for building interactive simulations, connects geographically distributed nodes, time management (for time- and even-driven simulations), data management (tuple space) scalability interoperability	designed to coordinate resources that are not subject to centralized control, uses standard, open, general-purpose protocols and interfaces, Web services concept of abstract interfaces allows for modular design (OGSA)
Limitations	no automatic setup, no mechanisms for managing execution according to the dynamically changing conditions of computing resources, no implementation with dynamic discovery, no security mechanism	general approach, so no support for interaction

Table 3.1: Limitations and possibilities of HLA and Grid for interactive simulations

In addition, the Grid idea is to facilitate access to computing resources and make it more transparent to the user. Currently, setting up distributed applications based on HLA requires tedious setup and configuration. The HLA standard does not cover aspects of dynamic discovery of HLA federations. For instance, just like other RTI implementations, the current RTI1.3-NG [77] implementation requires common configuration files that specify the location of controlling components (endpoint of RTI control process), as well as a definition of the data to be exchanged, before runtime. Therefore, there is a need for a mechanism that sets up an HLA-based application on geographically distributed system (i.e. the Grid) in a more convenient way. Subsequently, HLA federates should be able to find one another dynamically and transparently to the user. Additionally, the RTI does not provide security mechanisms similar to the one provided by the Grid Security Infrastructure (GSI) [70],

A comparison of limitations and possibilities of HLA and Grid for running interactive simulations is listed in Tab. 3.1.

3.2 Grid HLA Management System

3.2.1 Requirements

In Section 1.3 we defined the main requirements for distributed interactive simulations. Firstly, they need to be *scalable* and require specific functionality such as *time management* for event-driven or time-driven simulations and *data distribution management* for convenient data exchange. Next, it should be possible to *connect simulations with different time management into one system* – i.e time-driven and event-driven simulations should be able to form components of one distributed simulation if necessary. Next, they require *a certain level of quality from the execution environment* that allows for *near-real-time communication* between distributed components. Additionally, the simulations consist of distributed elements with varying functionality and need *access to a variety of distributed resources* and, if possible, *convenient and dynamic setup* for using these resources. Last, but not least we need an environment *supporting legacy simulations*.

To fulfill these requirements, we have designed a system that supports running HLA-based distributed interactive applications in a Grid environment. The components of the system allow for efficient execution of HLA-based applications on the Grid. Requirements such as *scalability*, *support for time and data management* and *the ability to connect simulations with varying time management* are fulfilled by building support for HLA-based simulations that are interfaced with the *HLA-Speaking Service* described in Chapter 4. *Access to distributed resources* is realized by a Grid environment (GRAM, GridFTP, GSI) [70] in which G-HLAM is designed to be run. *Dynamic and automatic setup* within a distributed environment is done by cooperation of the *Broker Service*, the *Registry Service*, the *RTIexec Service* and *HLA-Speaking Services* as described in Section 3.2.3. *Efficient execution* is assured thanks to the *Monitoring Service*, *Benchmark Service* and *Migration Service* as described in Chapters 5 and 6. The system *supports legacy* HLA simulations - The *Monitoring Service*

is transparent to the user and the *Migration Service* does not require the user to port his application from the HLA to any other library as in [188]. Finally, the Service Oriented Architecture of G-HLAM supports *modular design*. G-HLAM modules communicate with Grid service interfaces, which allows them to be developed independently of each other and also, if needed, facilitates adding preexisting services to the system.

3.2.2 Functionality

The system we propose consists of services which control the whole HLA application as well as the components that should be installed on each HLA-enabled Grid site. The system is shown in Fig. 3.4 and its functionality is described below.

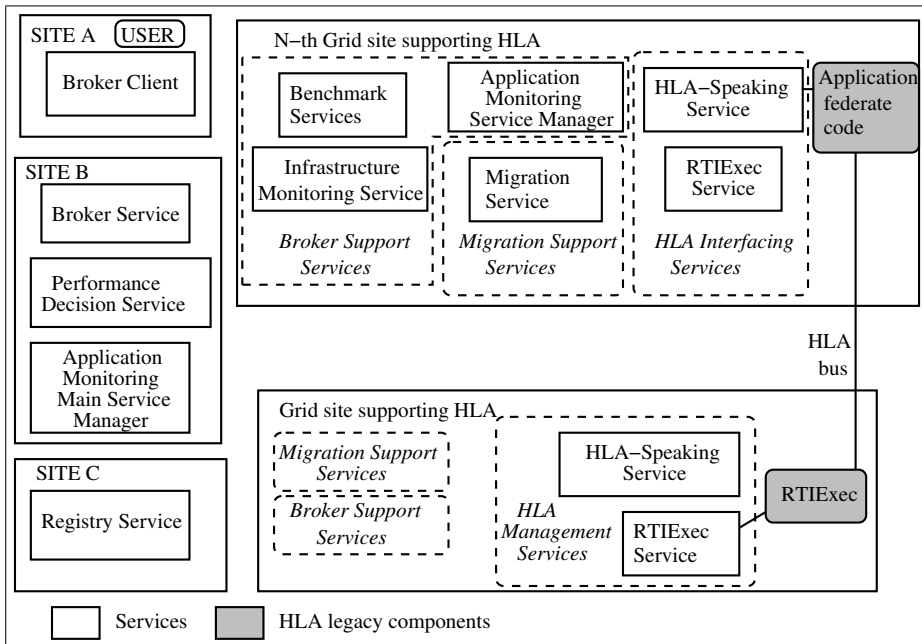


Figure 3.4: Grid HLA Management System Architecture (G-HLAM) consists of services which control the whole HLA application and the services that should be installed on each HLA-enabled Grid site.

Collective services

The group of collective services (global per application) consists of the *Broker Service*, the *Performance Decision Service*, and the *Main Service Manager*. The *Broker Service* manages the whole application and assures that its performance satisfies a user in a processing loop. This is done by communication with other services. The *Main Service*

Manager is a part of the OCM-G monitoring system [12] responsible for gathering information from *Service Managers* residing on each site. The *Performance Decision Service* obtains data from the *Main Service Manager* and decides if migration should be performed. When needed, it triggers the *Broker Service* to perform the action of migration.

There are also services that should reside on each HLA enabled site. They form three groups: *HLA Interfacing Services*, *Broker Support Services* and *Migration Support Services*.

Interfacing HLA

HLA Interfacing Services interface and manage actual HLA installations on each site. This group includes the *HLA-Speaking Service* that represents HLA installations on a particular Grid site and *RTIexec service* which interfaces the RTIexec process described in more detail in the next Chapter. The *HLA-Speaking Service* acts as an interface to the actual application code. It is responsible for running that code, saving it and restoring it when there is a need for migration. The service uses Globus GRAM [70] for job submission. The *RTIexec Service* is responsible for running the RTIexec coordination process of RTI.

Broker support

The aim of the *Broker Support Services* is providing the *Broker Service* with information necessary for decisions about setup and migration of application components. This group includes *Infrastructure Monitoring Services*, *Benchmark Services* and *Application Monitoring Service Managers*.

The *Broker Service* should take into account information from *Infrastructure Monitoring Services* present in the Grid environment that provide network parameters and host load information. In Chapter 6, Section 6.1 we will describe *Benchmark Services* as another possible source of information about HLA-based application behavior. Currently, *Benchmark Services* are based on a scenario where the application consists of a simulation module that produces large amounts of data (simulation output) for the visualization module and receives small amounts of data (requests for parameter changes) from the interaction module. *Application Monitoring Service Managers* are based on the OCM-G monitoring system [12] as described in Chapter 6, Section 6.2. They aim is to receive monitoring data about federation performance and send it to the *Main Service Manager* which, in turn, sends it to the *Performance Decision Service*.

Migration support

The *Migration Service* provides direct support for fault-tolerant and effective performance of HLA-based applications acting as a conductor for *HLA-Speaking Services* on source and destination sites for the required migration. Once the *Broker Service* decides where to migrate, the actual action is performed by the *Migration Service* which

asks the *HLA Speaking Service* to checkpoint states of federates that are running on that site. Subsequently, the *Migration Service* starts the *HLA Speaking Service* on the remote site it wants to migrate to, transfers the code and checkpoint files (using GridFTP) and restarts the federates. A technical description of the HLA migration process can be found in Chapter 5.

3.2.3 Automatic HLA application setup within G-HLAM

In this thesis we use Unified Modeling Language (UML) [169] – in particular, the *sequence diagrams* are used to show time sequences of messages exchanged by entities in the designed system. Fig 3.5 shows types of exchanged messages. There are two types of send messages with their corresponding return messages. *Synchronous message* means that a sender waits until it returns. Therefore, explicit indication of *synchronous return* message is optional. *Asynchronous message* does not block sender. The *asynchronous return* has to be indicated if it exists.

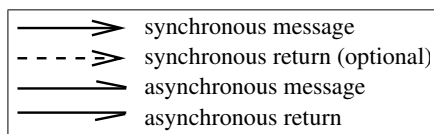


Figure 3.5: Type of messages in UML sequence diagram

A sequence diagram of the HLA application setup is depicted in Fig. 3.6 and illustrates how an application is started. The scenario is as follows: *Broker Client* asks the *Broker Service* to set up the application according to the given description. Then *Broker Service* asks the external *Service Registry* about the available *RTIExec Factory Services*. It chooses one *RTIExec Factory* from the ones proposed by the *Registry Service* and asks it to create the *RTIExec Service*. It might turn out that the service already exists, (it has been created by another application), but the factory knows its endpoint and returns it to the *Broker Service*. Then *Broker Service* asks the *Registry* about existing *HLA Speaking Services Factories* than can run *HLA-Speaking Services* with the appropriate HLA implementation version (determined by user code compilation) and assigns them to the federates from the application description. Each *HLA Speaking Service* is responsible for managing federate processes on its site. The *Broker Service* asks the *HLA Speaking Service Factory* on each of the chosen sites to create a *HLA Speaking Service*, transfers the code of the federates and invokes a start operation on each *HLA Speaking Service* providing the RTIExec endpoint obtained previously. In this way, there is no need to manually set up the federation and the mechanism presented here allows for automatic discovery of available RTIExec services. Subsequently, each *HLA Speaking Service* creates appropriate federate processes that join RTIExec.

Upon being started up, the application is monitored by *Monitoring Services*. In case of bad performance, *Migration Services* perform migration of particular federates.

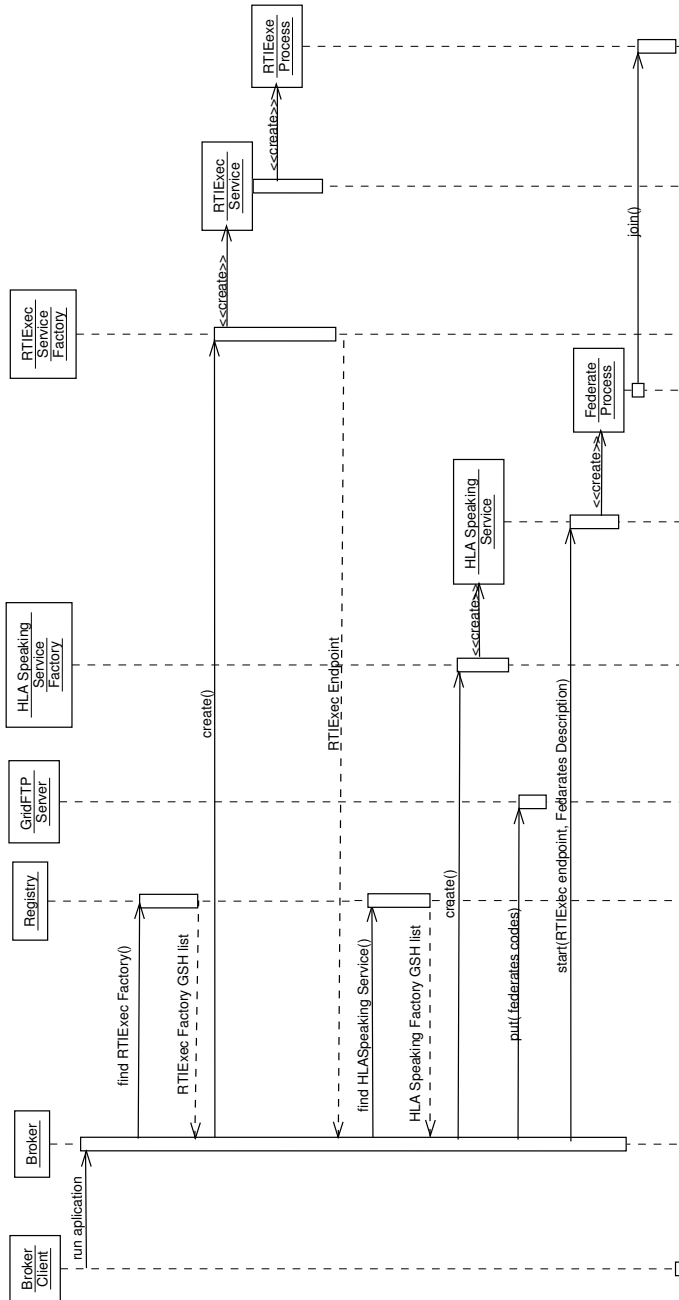


Figure 3.6: UML sequence diagram – interactions between *Broker Service*, *Registry*, *HLA-Speaking Service*, *RTIExec Service* and *GridFTP server* during start up of HLA application.

The solution presented here uses the *Broker Service* to ensure that the same RTIExec process is assigned to each of the federates within the same federation. The *Broker Service* first chooses an RTIExec endpoint and then forwards this endpoint to all federates of the application as it starts them up.

A different approach is presented in [194], where an Index Service (Registry) manages a database with information on which federation is assigned to which RTIExec process. The reservation mechanism is used to store an RTIExec handle in the database and return the same handle for later queries for the same federation. This solution doesn't require any entity responsible for managing the whole application (such as the Broker Service). Each federate, encapsulated in a Grid service can join the system itself by asking the Index Service about an RTIExec endpoint. However, this requires the Index Service to implement a reservation mechanism which is not necessary in our approach (where each application has its own Broker that assigns the same RTIExec to all federates within the same federation).

3.2.4 Summary

Requirement	Solution proposed in G-HLAM
scalability, support for time and data management, ability to connect simulations with different time management	Support for HLA based simulations (interfaced with <i>HLA Speaking Service</i>)
access to distributed resources	Service Oriented Architecture of G-HLAM, Grid environment (GRAM, GridFTP, GSI)
dynamic setup within distributed environment	<i>Broker Service, Registry Service, RTIexec Service, HLA-Speaking Service</i>
efficient execution	<i>Monitoring Service, Benchmark Service, Migration Services</i>
support for legacy HLA simulations	<i>Monitoring Service, Migration Service, HLA Speaking Service</i>

Table 3.2: Summary of requirements and their solutions in G-HLAM

Tab. 3.2 shows how G-HLAM services fulfill requirements of distributed interactive simulations. Support for HLA-based applications satisfies requirements specific for simulation development such as scalability, support for time and data management and the ability to connect simulations with different time management. Grid concepts allow for access to distributed resources. Particular G-HLAM services support convenient setup of whole applications and efficient execution as described in Tab. 3.2. The Service Oriented Architecture of G-HLAM allows for its modular design. Grid service interfaces between G-HLAM modules help them to be developed independently and enable the adding of "off-the-shelf" services to the system if necessary.

3.3 Analysis of G-HLAM System

In order to describe and analyze the behavior of G-HLAM, we use Petri Nets [128]. We have chosen this formalism as it provides a strong theoretical background for analyzing concurrent systems, can be used as a language describing workflow of these systems [98] and provides mechanisms for deadlock detection [109, 83]. There are other formalisms for modeling system behavior such as David Harel statecharts [75] which provide a basis for UML statechart diagrams [169], however Petri Nets seem to be the most convenient mechanism for our purposes.

We will provide a short introduction to the chosen formalism, then present the Petri Net-based model describing G-HLAM behavior and show that deadlocks cannot occur. For our analysis, we have used the Pipe [130] tool.

3.3.1 Petri Net

A Petri net (also known as a place/transition net or P/T net) is one of several mathematical representations of discrete distributed systems. As a modeling language, it graphically depicts the structure of a distributed system as a directed bipartite graph with annotations. As such, a Petri net has *place* nodes, *transition* nodes, and directed arcs connecting places with transitions. Formally, a Petri Net [117] is a 5-tuple $PN = (P, T, F, W, M_0)$ where:

1. P is a set of *places*
2. T is a set of *transitions*
3. $F \subseteq (P \times T) \cup (T \times P)$ is a set of arcs from places to transitions and from transitions to places
4. $W : F \rightarrow 1, 2, 3, \dots$ is a *weighting function* (in this thesis we assume that each arc has weight=1)
5. $M_0 : P \rightarrow 0, 1, 2, 3, \dots$ is a *initial marking* (each place can have 0 or more tokens)
6. $P \cup T \neq \emptyset$ and $P \cap T = \emptyset$

The execution of PN is done by executing (*firing*) its transitions. A place connected with a transition in the sense that an arc leads from the place to the transition, is called an *input place*. A place connected with a transition in the sense that an arc leads from the transition to the place, is called an *output place*. A transition can be fired if all its input places have at least as many tokens as the weight of the arc connecting this place with this transition. The firing changes the marking by subtracting one token from every input place and adding one token to every output place of the transition. In our notation, if M_i is the marking before the transition t is fired and M_j is the marking after it is fired, we write that $M_i[t]M_j$.

3.3.2 Using Petri Nets for deadlock detection

One important feature of PNs is that they can be used to analyze whether a deadlock can occur in the modeled distributed system [109, 83].

In this thesis we use the approach described in [109]. A PN system contains a deadlock if it can reach a state in which no transition can be fired. We use the Petri Net formalism to prove that the functionality of the system is deadlock-free. This analysis is performed by creating a so-called *graph of reachable states*.

Petri Nets can also be described by a sequence of markings which occur. Starting from the initial marking it is possible to compute a set of all markings reachable from that marking (the state space of the PN system) and all paths that the system may follow to move from state to state. The set of all possible sequences forms a connected graph called the *graph of reachable markings* or the *graph of reachable states*. By analyzing the graph, we can determine whether a deadlock can occur – a deadlock occurs if there exists a dead marking (a node with no output arcs in the graph). The absence of such a node is sufficient to guarantee the absence of deadlocks [109].

To provide a formal definition of a *graph of reachable states* we first define the *reachability set* [109]. The *reachability set* of a PN system with initial marking M_0 is denoted $RS(M_0)$, and it is defined as the smallest set of markings such that:

- $M_0 \in RS(M_0)$
- $M_1 \in RS(M_0) \wedge \exists t \in T : M_0[t]M_1 \Rightarrow M_1 \in RS(M_0)$

Given a PN system, and its *reachability set* RS , we call a *reachability graph* $RG(M_0)$ the labelled directed multigraph whose set of nodes is RS , and whose set of arcs A is defined as follows [109]:

- $A \subseteq RS \times RS \times T$
- $(M_i, M_j, t) \in A \Leftrightarrow M_i[t]M_j$

3.3.3 A Petri Net based model for the G-HLAM system

A simplified Petri Net illustrating the behavior of the G-HLAM system is shown in Fig. 3.7. The *Application Monitoring Service* monitors the performance of the HLA-based application and reports to the *Broker Service* if something is wrong. This is shown as the two transitions $AM \rightarrow Monitor$ (normal situation) and $AM \rightarrow report\ bad\ performance$ (this transition is fired in case some anomaly is observed by *Application Monitoring*). Next, the *Broker Service* asks *Benchmark Services* about results and, based on that, makes decisions about migration (in Fig. 3.7 the migration takes place from site A to site B). Subsequently, a method of the *Migration Service* is invoked at site A to perform migration. Migration is performed as described in Chapter 5 (Section 5.2). Firstly, as shown in the sequence diagram in Fig. 5.2, a Control Federation at site A is created and then the actual migration is performed. In order to avoid too much detail in one figure, a detailed Petri Net for modeling migration is shown

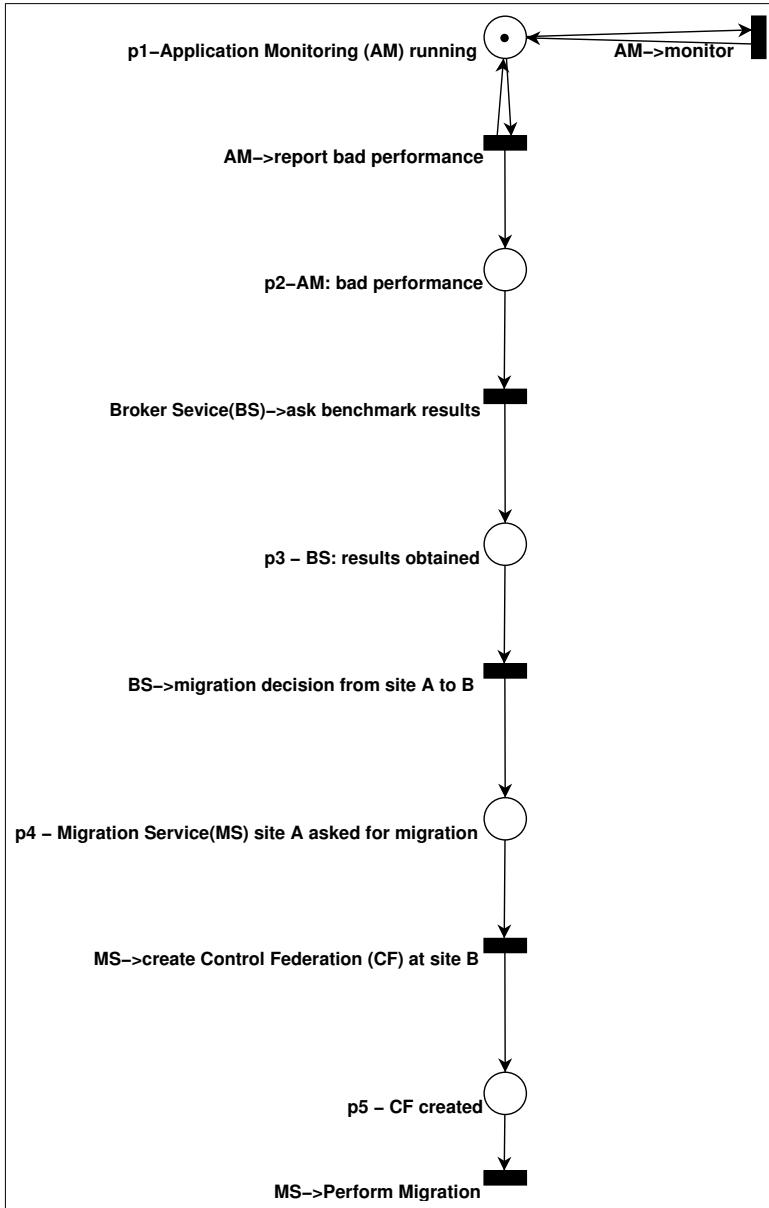


Figure 3.7: The Petri net describing the G-HLAM system – simplified model showing how monitoring of an application triggers migration decision.

in Fig. 3.8. To determine if there is no deadlock present in the net as described in

Section 3.3.2 we needed to build a corresponding graph of reachable states. Building such a graph is an automatic process, so we have used one of the tools available on the web. For that purpose we have chosen Pipe [130] as it was an easy to use, user friendly tool with all functionality needed .

Petri net based model for migration

A Petri net–based model for actual migration for two federates is shown in Fig. 3.8. Federate 1 (indicated as *fed1* in Fig. 3.8) represents the federates that will be moved during migration, while federate 2 (*fed2*) represents the federates that have joined the federation, but will not be moved. Each federate uses the GridHLAController library as an interface to the control federation. In Fig. 3.8 the GridHLAController library is indicated as *control*. More information about the library will be provided in Chapter 4 and Chapter 5; for now it is sufficient to know that it manages a four–bit state of the user federate defined in Table 3.3.

bit number	meaning
0	external request for saving came(1)/not came(0)
1	internal request for saving came(1)/not came(0)
2	external request for restore came(1)/not came(0)
3	internal request for restore came(1)/not came(0)

Table 3.3: Four–bit state of a federate

An external request for saving or restoring is defined as a request coming from the *Migration Service* that invokes `RequestMigrationSave` or `RequestRunWithRestore` operations on the *HLA–Speaking Service*. The internal request is defined as the one sent through the control federation using federation–wide save/restore mechanisms and is used only to freeze the federation.

Detailed description of Petri net based model

From the place *P1 G-HLAM running* there are two transitions that can be fired: *G-HLAM run* in a normal situation and *request for migration* if a migration is requested as described in detail in the previous Section.

Following the request, the *Monitoring Service* invokes the `requestMigrationSave` operation on the *HLA–Speaking Service* described in Section A.1 and A.2. This invokes a request for an external save that is propagated in the HLA bus via the control federation to all federates.

Each federate recognizes (by means of the multiple process management algorithm described in Section 5.3.2) if the request was directed to it. Furthermore, the request is processed only when a federate is in its normal state - that is all bits of its state are set to 0. If these conditions are met, the GridHLAController library sets the first bit in the federate state (by means of the `request_external_save` routine indicated in Fig. 3.8) and the federation–wide save mechanism [78] is used to set the

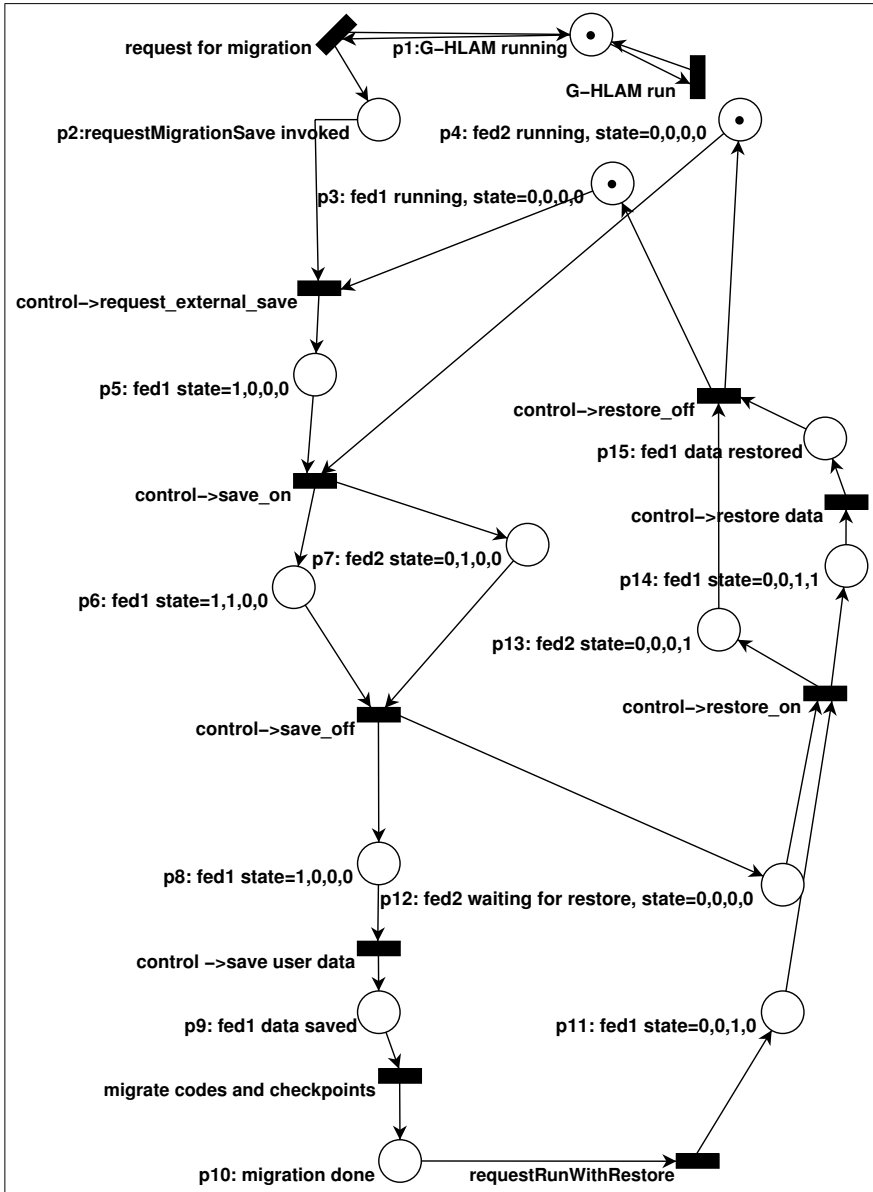


Figure 3.8: The Petri Net describing migration in G-HLAM. Federate 1 (*fed1*) represents the federates that will be moved during migration, while federate 2 (*fed2*) represents the federates that have joined the federation, but will not be moved.

second bit of all federates. In particular, the `save_on` routine of the GridHLAController library uses this mechanism, to notify a federate that other federates have begun saving the internal state of the RTI. Similarly, the `save_off` routine notifies a federate that other federates have finished saving the internal state of the RTI.

Afterwards, the federates that have two first bits set (we will call them *fed1 type* federates as in Fig. 3.8) will be migrated and these with only the second bit set (called *fed2 type* federates) are frozen. After saving the RTI internal state, *fed2 type* federates await the restoration of the federation and *fed1 type* federates save user data and migrate.

When `requestRunWithRestore` is invoked, *fed1 type* federates are started on the destined site, with the third bit set and restore the RTI internal state together with user data. The federation-wide restore mechanism that is used to set the third and fourth bits of federate states allows for synchronization of *fed1 type* and *fed2 type* federates, so no federate will be prematurely restarted before user data is restored. In particular, the `restore_on` routine of the GridHLAController library uses this mechanism to notify a federate that other federates have begun restoring the internal state of the RTI. Similarly, the `restore_off` routine notifies a federate that other federates have finished restoring the internal state of the RTI.

After restarting, the system returns to its initial state and the scenario can be repeated when another request from G-HLAM comes (transition *request for migration* is fired).

As described above, we build the corresponding graph of reachable states and determine that there is no deadlock present in the net as described in Section 3.3.2. We have used the Pipe tool [130] for that purpose.

3.4 Summary

This Chapter constitutes an important part of this thesis as it presents the overall concept and analysis of the Grid HLA Management System (G-HLAM) which is our solution for filling the gap between requirements of legacy simulations based on the HLA standard and the execution environment offered by Grid. This is done by allowing for efficient and convenient execution of such simulations in this environment. G-HLAM is based on SOA and includes services for interfacing legacy code to G-HLAM, migration, performance monitoring and benchmarking. These services are described in more detail in subsequent Chapters.

Since the system supports simulations based on HLA, it enables the use of advanced features of this standard such as support for time and data management, the ability to connect simulations with different time management and scalability. G-HLAM uses distributed resources offered by the Grid environment. It supports dynamic and automatic setup of distributed simulations and their efficient execution. Finally, the Service Oriented Architecture of G-HLAM supports its modular design. G-HLAM modules communicate with Grid service interfaces, which allows them to be developed independently and to be interoperable with preexisting services.

Additionally, we have presented Petri Net–based analysis of the system including its model. We prove that the system runs correctly without the possibility of deadlocks. For our analysis we have used a graph of reachable states generated by the Pipe [130] tool.

Interfacing Services

In this Chapter¹, we describe in more detail the HLA interfacing services shown in Fig. 3.4 of Chapter 3 – namely *HLA-Speaking Service* and *RTIExec Service*, which aim at interfacing HLA-based applications with G-HLAM.

The described services were designed to satisfy the requirements listed in Section 1.3. First, they interface G-HLAM to the HLA application enabling us to take advantage of advanced HLA features (such as *time and data management*, *simulation interoperability*, *scalability*) when running a simulation in the Grid environment. The designed interface does not require the user to port his HLA application to any other library as in [188] – which means that legacy applications are supported.

Next, the services described in this Chapter are used by the *Broker Service* to satisfy the requirement of the *automatic setup* of distributed HLA-based applications as described in detail in Section 3.2.3 of the previous Chapter.

Furthermore, in order to fulfill the requirement for *efficient execution* of distributed simulations on the Grid, the *HLA Speaking Service* is responsible for informing user federates about checkpointing and migration requests which come from the G-HLAM system when the *Monitoring Services* (described in Chapter 6) report bad performance at the current location.

The main problems addressed in this Chapter are:

- building an universal interface of user federates (also those from legacy simulations) to G-HLAM,

¹The results described in this Chapter formed the basis to the following papers:
K. Zając, M. Bubak, M. Malawski, and P. M. A. Sloot. Towards a Grid Management System for HLA-Based Interactive Simulations. In S. J. Turner and S. J. E. Taylor, editors, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 4–11, Delft, The Netherlands, October 2003. IEEE Computer Society.
K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Interactive Grid Computing: Adapting High Level Architecture-based Applications to the Grid. In P. Doerffer and J. Rybicki, editors, *TASK QUARTERLY. Scientific Bulletin of Academic Centre in Gdansk*, TASK Publishing, 2004, 8(4):549-559

- interfacing RTI control processes to the G-HLAM,
- building easy-to-use API for interaction of user code with the service,
- managing multiple federate processes at one Grid site,
- efficiently using the whole Grid site.

We also present the evolution of *HLA-Speaking Service* – from a prototype only able to manage one federation process to a version that can deal with multiple federation processes.

This Chapter is organized as follows: in Section 4.1 we outline the role of the *HLA-Speaking Service*, in Section 4.2 we describe its version for a single process and in Section 4.3 we describe an analogous version for multiple processes. The discussion of service evolution is described in Section 4.4. Section 4.5 describes *the RTIExec Service*. We summarize the Chapter in Section 4.6.

4.1 The role of HLA Interfacing Services

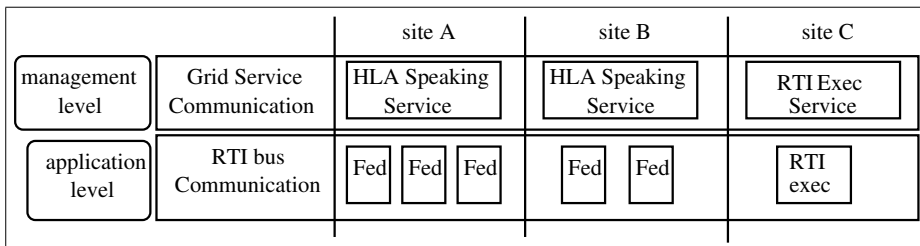


Figure 4.1: The role of *HLA-Speaking Service* and *RTIExec Service*. Services are located in a management level – they are responsible for managing execution of the federates and the control RTIExec process.

In our approach we consider different HLA implementations and their versions installed on Grid sites as resources. We introduce the *HLA-Speaking Service* and the *RTIExec Service* to allow for sharing these resources between members of the Grid community. Those services are located in a management layer – that means they are not responsible for exposing HLA functionality in their interfaces as e.g. in [185], but to manage execution of the federates (in the case of the *HLA-Speaking Service*) and the HLA RTI control RTIExec process (in the case of the *RTIExec Service*) on a particular site. The management functionality of the *HLA-Speaking Service* includes starting the execution of a federate code on their site as well as saving and restoring their states upon request coming from G-HLAM. The management functionality of the *RTIExec Service* includes starting the RTIExec process and providing its location (endpoint), so that user federates can join the HLA RTI.

An example is shown in Fig. 4.1 – on site A the *HLA-Speaking Service* manages three user federates, on site B – two federates and on site C there is an RTIExec Service running the RTIExec process. A HLA-specific API is hidden in the RTI library functionality and is not exposed as a service interface. Instead, the federates and RTIExec communicate using HLA RTI in the same way as in legacy simulations.

4.2 HLA-Speaking Service for a single process

4.2.1 Functionality

As described above, the role of the *HLA-Speaking Service* is to manage user federates on its site. In this Section we present a version of the service suitable for handling single processes (which, however, can contain many federates). The architecture of the *HLA-Speaking Service* is shown in Fig. 4.2. The service has two *porttypes* (in

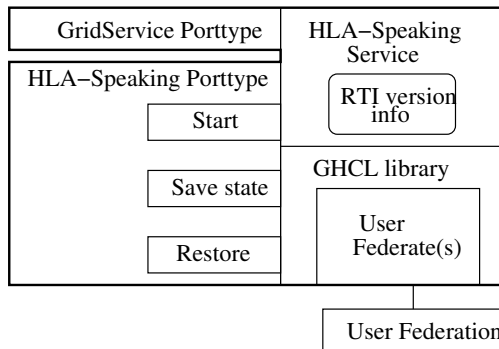


Figure 4.2: Architecture of *HLA-Speaking Service* for a single federate process – the service provides the HLA-Speaking Porttype - with operations supporting user federate process startup and saving or restoring its state.

Grid services terminology a *porttype* is a set of operations offered by a service).

Since it is a Grid service, it provides the GridService Porttype that includes standard OGSi operations like querying service data [61]. Here, service data contains information about the supported version of the HLA implementation, important for the *Broker Service* that sets up the whole HLA application.

Additionally, the service provides the HLA-Speaking Porttype - with operations supporting user federate process startup and saving or restoring its state. The full list of the operations can be found in Section A.1.

A client (e.g. the *Broker Service* or the *Migration Service*) can use these operations to load user federate code into the service, and request for checkpointing and migration of the federate code as shown in detail in Section 5.2. To receive these requests, the user federate uses the GridHLAController library (GHCL) described in the next Section.

4.2.2 GridHLAController Library routines for a single process

The GridHLAController library (GHCL) is an interface between the Grid service layer and the user federate. It is used to notify the user federate about saving and restoring requests which come from the *Migration Service* to the *HLA-Speaking Service* and finally to the user federate. The library contains two groups of routines: initialization functions and migration functions. In this Section we focus on the former group, while the latter can be found in the next Chapter, describing the migration mechanism (Section 5.3).

The group of initialization functions contains startup routines that start the program and register the main simulation loop (the callback defined by the user). There are also connection functions that are used to connect the user federate with the migration engine. The full description of the routines can be found in Section B.1.1.

The next Section describes how the GHCL library is used to set up user federate(s) within the *HLA-Speaking Service*.

4.2.3 User federate setup in HLA-Speaking Service

In this thesis we use UML component diagrams to illustrate cooperations of system's components. The notation used in this kind of diagram is illustrated in Fig. 4.3. The

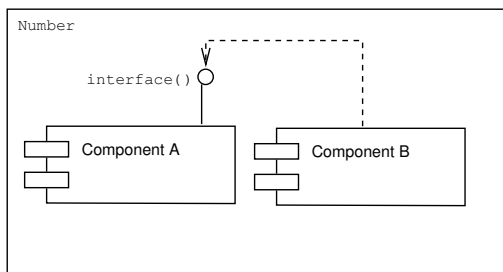


Figure 4.3: UML component diagram example

system consists of a number of components. Each of them can *expose* interface so the others can *use* it. In Fig. 4.3 component A *exposes* interface and component B *uses* that interface. The particular set of cooperating components can appear in the system more than once. The *number* indicates a number of the sets of the components in the rectangle.

Fig. 4.4 shows a UML component diagram of the *HLA-Speaking Service*. It shows how elements of the architecture presented in Fig. 4.2 cooperate together when initiating user federate process. The detailed description of GHCL routines used in this Chapter can be found in Section B.1.1.

The *Broker Service* invokes the `start` operation of the *HLA-Speaking Service* which in turn calls the `start()` function of the GHCL library. This action triggers startup of user code.

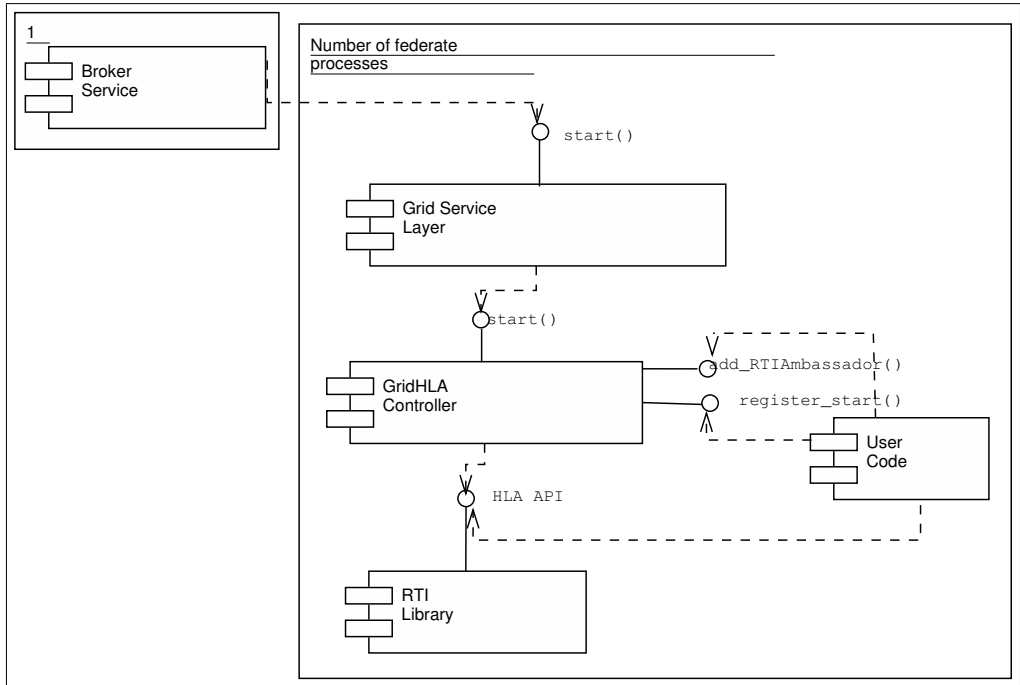


Figure 4.4: UML component diagram of *HLA-Speaking service* for a single process showing how elements of the service architecture cooperate together when initiating user federate process.

User code uses the GHCL library `GridHLAController::register_start()` routine to register the main loop of its simulation and the `GridHLAController::add_RTIAmbassador()` function to inform the GHCL that the state of the user federation (which is related to the `RTIAmbassador` class in the HLA RTI library) has to be saved in case of migration. GHCL uses the HLA API to connect to other user federates within that federation. This is necessary to inform other federates about the migration of the managed federate (see details in Section 5.3). User code utilizes the HLA API as in legacy applications.

4.3 HLA Speaking Service for multiple processes

4.3.1 Functionality

In the previous Section we have shown how to create an universal interface between user federates (also for legacy simulations) and G-HLAM and also how to build an easy-to-use API for interaction of user code with the service. However, the presented solution was suitable only for Services which include single federation pro-

cesses. There is a need to extend this functionality to support many processes and therefore to make more efficient use of the Grid site.

As the first step, we have decided to use the GRAM [70] interface for submission of user processes. The actual commands submitted by the *Broker Service* are based on the Resource Specification Language (RSL) [139] and include information indicated in the RSL specification: executable, standard output and error files, their relative directory, number of processes and information whether it is an MPI job.

However, RSL-based submission in G-HLAM differs from original RSL submission as some of the information needed cannot be provided by the external service (e.g. the *Broker Service*) and is therefore filled by the *HLA-Speaking Service*. This information includes:

- name of the job manager used by GRAM,
- GridFTP server name that is used to fetch code from remote sites (e.g. *Broker Service* site during application setup or the site from which federates are migrated),
- absolute paths,
- version of the HLA implementation and its dynamic libraries. The version is determined by the *HLA-Speaking Service* factory before service creation as described in 3.2.3.

The *HLA-Speaking Service* fills the submitted RSL-like command with necessary information included in the configuration file (changes relative paths to absolute paths, adds necessary environmental variables etc.) and submits this command to the GRAM manager.

Fig. 4.5 shows an *HLA-Speaking Service* architecture supporting multiple federates. As previously, the service has two porttypes. The Grid porttype is the same as for the single-process version. Operations of the *HLA-Speaking Service* Porttype are, as previously, designed to start user federate processes and pass requests for saving and restoring states of the federate code. However, they are extended to support multiple federates and include operations that create control federation, start the control federate, fetch and start user federates codes, save and restore state of indicated processes. The full list of the operations can be found in Section A.2.

In order to forward requests from the *HLA-Speaking Service* to the actual user's federates, we have designed a control federation. Upon being submitted by a service using GRAM, each user process uses the GridHLAController library (GHCL) library to join the control federation. A detailed description of this library can be found in Section 4.3.2. As can be seen in the presented design, the scope of the control federation is limited to the *HLA-Speaking Service* local site, so the user application participates in one control federation for each Grid site it runs on. Apart from that, user federates communicate with one another in a normal way – using RTI in the scope of user-defined federations.

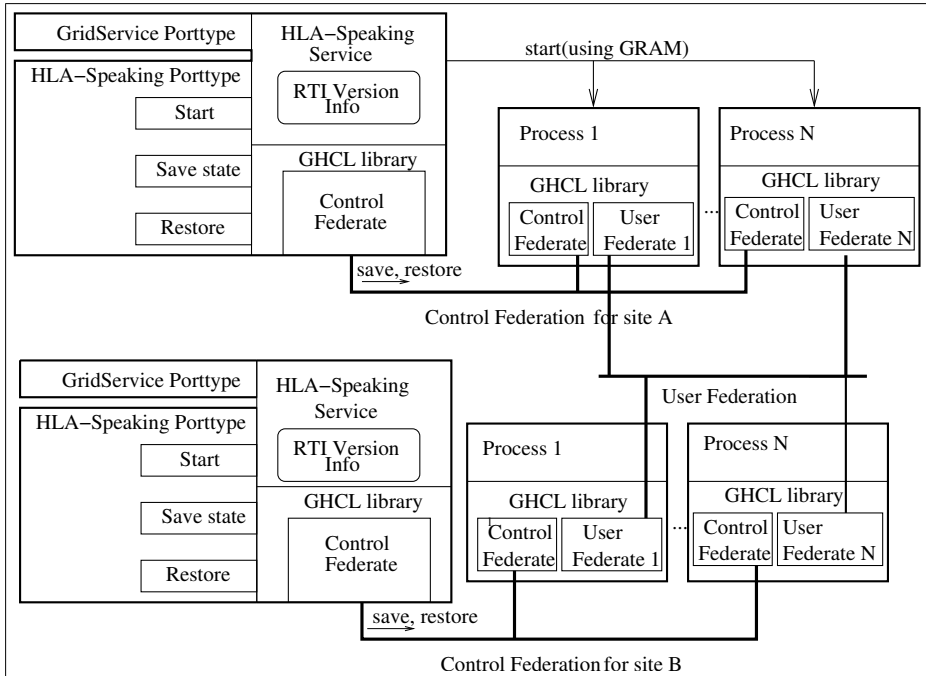


Figure 4.5: Conceptual view of an *HLA Speaking Service* for multiple federate processes. The service provides the *HLA-Speaking Porttype* - with operations supporting user federate processes startup and saving or restoring their state. Control federation is used to communicate the service with user processes.

4.3.2 GridHLAController Library routines for multiple federates

As in the previous Section, the GridHLAController library is divided into two groups of routines: the first supporting initialization and the second related to migration. In this Section we focus on the former group, while details about actual GHCL migration support can be found in the Section 5.3.

The initialization group contains routines that initiate all the necessary functionality in the GridHLAController library, create a control federate in the user process and connect it to the control federation. The group contains also connection functions used to connect the user federates with the migration engine. The full description of the routines can be found in Section B.1.2.

The next Section shows how GHCL library is used to set up multiple processes via the *HLA-Speaking Service*.

4.3.3 Application setup – component diagram

Fig. 4.6 shows a UML component diagram of the *HLA-Speaking Service* and user processes. It describes how elements of the architecture presented in Fig. 4.5 cooperate together during application setup.

The Grid layer starts user federates (by means of a Grid job manager such as GRAM). Subsequently, control federate code is loaded into the *HLA-Speaking Service*. The control federate is treated similarly to the federate process described in Section 4.2. The difference is that this federate is designed for control purposes and therefore is not defined by the user, but rather by the G-HLAM system itself. Therefore its start callback is predefined and does not need to be registered by the `GridHLAController::register_start()` routine. Instead, the control federate joins the control federation using its internal `join_controller()` routine). User federates use the `GridHLAController::init()` routine to initiate all the necessary functionality in the `GridHLAController` library, create their own control federates and connect them to the control federation. As previously, they also use the `GridHLAController::add_RTIambassador()` function to inform the GHCL that the state of user federation has to be saved in case of migration.

4.4 Evolution of *HLA-Speaking Service* – discussion

In two previous Sections we presented two versions of *HLA-Speaking Services*.

Single-process version	Multiple-process version
user code is loaded directly into the (JNI) service	GRAM is used to run user code in separate processes
communication within the same process space	longer communication using HLA RTI (control federation)
requires separate service for each process – unnecessary consumption of computing resources	only requires a single service for all processes on a Grid site
does not obey the common administration policy of the Grid site	by using a job manager complies with the administration policy of the Grid site
difficult to run MPI simulations	facilitates running MPI simulations

Table 4.1: Comparison of the two versions of *HLA-Speaking Service*

The advantages and disadvantages of both are summarized in Tab. 4.1.

The version for single processes uses special programming mechanisms (JNI [88]) to load user code directly into the Service. This results in quicker setup of the federate, since the solution does not require job manager mediation to start the user process. Furthermore, saving and restoring actions takes less time - the user federate and the service communicate within the same process space, instead of using a control federation as is the case for the multiple-process version.

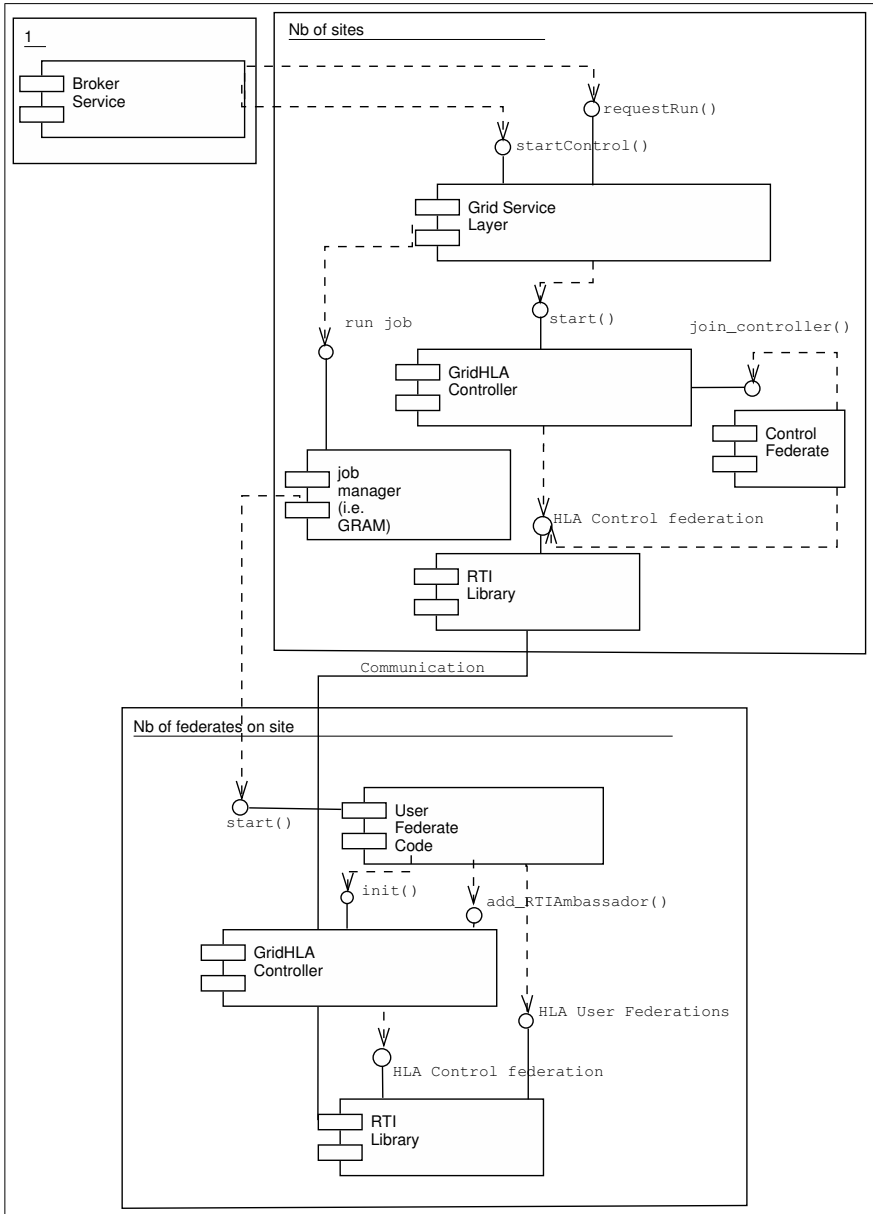


Figure 4.6: UML component diagram illustrating setup of multiple federate processes by the *HLA Speaking Service* – describes how elements of the service architecture cooperate together during application setup.

On the other hand, the version for multiple processes does not require starting a separate Grid service for every user process and therefore it consumes less computing resources. Usage of a job manager via a GRAM interface assures that the Grid site is used efficiently and in accordance with site administration policy (using job managers on Grid sites is usually required by administrators). Additionally, this version enables straightforward execution of HLA simulations which are parallelized using MPI. Using the single-process version would be more difficult for that purpose, because MPI requires starting all processes concurrently.

To summarize, both solutions have their advantages and disadvantages and each of them can be chosen according to user needs. The multiple-process version was created based on experience with the single-process version and, as can be seen from the presented comparison, is more natural in situations when a user with a legacy HLA application wants to utilize Grid resources in order to run it in an efficient way. However, it is worth mentioning that the single-process approach is similar to that described in [194] where a system for dynamic discovery of federates encapsulated within Grid services is presented. It is more natural to use when one wants to add a semantic descriptions of a federate loaded into Grid Service and compose complex distributed simulations from different federates based on this description [23].

4.5 Interfacing RTI control process

The *RTIExec Service* is used by the *Broker Service* to perform automatic setup of an

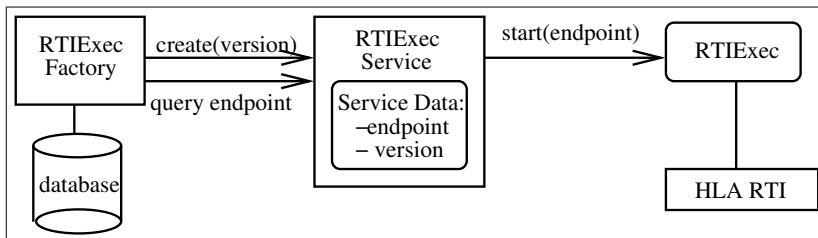


Figure 4.7: Management of *RTIExec* endpoints – cooperation between *RTIExec Factory*, *RTIExec Service* and actual *RTIExec* process.

HLA-based application. The factory stores information about versions of *RTIExec* implementation that can be run on its site. On request it creates an appropriate *RTIExec Service*.

The Service starts the actual *RTIExec* as shown in Fig. 4.7. Its service data includes:

- endpoint of the process – required by federates for communication,
- version number of RTI implementation.

Since the RTIExec process can coordinate many federations, it is not necessary to create separate instances for different applications as this would result in waste of computing resources. Therefore, the *RTIExec Factory* manages a database of existing services containing their versions and endpoints (obtained by querying service data) and assures that there is no attempt to create two RTIExec processes in the same version. If creation of a service with the existing version is requested more than once, the Factory returns an appropriate endpoint obtained from its database.

4.6 Summary

The goal of this Chapter was to present in detail the HLA Interfacing Services, first introduced in Chapter 3, Section 3.2.2. These services allow for cooperation between user HLA applications and the Grid Services framework. An evolution of services was presented: first, we described an *HLA-Speaking Service* which interfaces a single process to the system; subsequently we described the design of a service that manages execution of multiple processes. We also described the *RTIExec Service* responsible for running the RTIExec process that controls RTI execution.

The solutions presented in this Chapter satisfy the requirements stated in Section 1.3 as follows: an universal interface from the G-HLAM to the legacy HLA application ports advanced HLA features (time and data management, interoperability, scalability) to the Grid; functionality of services allows for automatic setup of distributed HLA-based applications, while efficient execution of the managing application is supported by forwarding checkpointing and migration requests which come from the G-HLAM system to user processes.

Summarizing, in order to satisfy the requirements mentioned above the following problems have been solved:

- universal interface to user federates (also those from legacy simulations) to G-HLAM – this was done by building the *HLA-Speaking Service* which can start execution of user legacy code using GRAM and can communicate with that code during execution,
- interfacing RTI control process to G-HLAM – achieved by the *RTIExec Service* which can start the process and return its location, so that user federates can automatically join it,
- designing an easy-to-use API for interaction of user code with the service – fulfilled by the GridHLAController library,
- managing multiple federate processes at one Grid site and using the whole Grid site efficiently – achieved by a GRAM interface to the local job manager.

Migration Services

In this Chapter¹ we present the part of the G-HLAM system which is responsible for migration of a HLA-connected federate(s) of the distributed application in the Grid environment. Migration requests arrive from the *Migration Service* (acting on demand of the *Broker Service*) via the *HLA-Speaking Service* and are processed by user federates. In this Chapter we describe the migration mechanism performed by these services. Their location in the G-HLAM architecture is shown in Fig. 3.4, in Chapter 3.

We focus on the requirement for *efficient execution* of the application in a Grid environment, which is specified in Section 1.3. Our approach proposes a mechanism for adapting application execution to changes in the Grid environment. The adaptation is realized through migration of badly-performing federates. We aim at designing a *scalable* and *universal* solution.

The main problems addressed in this Chapter are:

- to provide the ability of migrating some of the federates without generating errors by other federates within the same federation,
- to support migration of multiple federates,
- to assure consistency of HLA services such as *data management*, *ownership management* and *time management* - e.g. when federate A regulates the time

¹The results described in this Chapter formed the basis to the following papers:

K. Zając, M. Bubak, M. Malawski, and P. M. A. Sloot. Towards a Grid Management System for HLA-Based Interactive Simulations. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 4–11, Delft, The Netherlands, October 2003. IEEE Computer Society.

K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Grid Service for Management of Multiple HLA Federate Processes. Accepted for Parallel Processing and Applied Mathematics (PPAM) 2005 conference, 11-14 Sep 2005, Poznań, Poland.

of another federate B and federate A is selected for migration. During migration federate B must behave in a consistent way.

The Chapter is organized as follows: in Section 5.1 we discuss possible solutions to the problems stated above, in Section 5.2 we present how G-HLAM services introduced in Chapter 3 (Section 3.2.2) cooperate to perform migration and Section 5.3 outlines collaboration between the *HLA-Speaking Service*, the GHCL library and managed federate processes during migration. We conclude the Chapter in Section 5.4

5.1 Overview of possible migration approaches

Migration of one of the federates without affecting the behavior of other federates is a nontrivial task [26]. The HLA specification conceals the actual location of data and messages, therefore using the HLA Management Object Model (MOM) [78] for obtaining information about a federate's internal state cannot guarantee that this state is up-to-date. Therefore, it is difficult to use it for saving the current state for migration purposes.

A different approach is to write a new implementation of the HLA standard. This, however, would be a huge task and would also force the developer to use a selected implementation of the RTI library.

We have therefore decided to use the HLA Federation Management API [78] to save and restore the HLA internal state as it is an easy-to-implement, straightforward solution. This API contains routines designed to freeze the internal state of the HLA communication bus from the beginning of the process saving procedure to their complete restoration. It hides implementation details of HLA (also underlying communication protocol) and supports saving and restoring all HLA data structures. For a user that creates its own HLA data structures, the API provides routines that transform pointers to the unique handles that become valid also after migration. The API assures that all RTI services such as *Data Management*, *Ownership Management* and *Time Management* are not affected during the saving/restoring process. However, no operations that change RTI internal state are allowed during the saving process.

The presented solution is not completely transparent to the developer, since it requires using the HLA API and it does not allow for saving user-specific data. However, it assures state consistency between all federates with relatively little overhead. We propose the GHCL library that not only facilitates to use HLA federation save API, but also allows the user to save his own data.

As stated in Chapter 2, Section 2.4.4, there exists another approach to the presented problem – it is solved by designing a sophisticated migration protocol for HLA-based applications. Unfortunately, the available solutions, although efficient [27], require the user to port his/her application to high-level-library build over HLA [188] and therefore does not support legacy HLA-based simulations.

5.2 Migration scenarios

In this Section, we present how G-HLAM services cooperate during migration. The scenarios are presented for two versions of *HLA-Speaking Services* described in the previous Chapter.

5.2.1 Single process version

The scenario of migration is shown in Fig 5.1. The main services participating in migration are the *Broker Service* which decides where to migrate, the *Migration Service* which performs actual migration and the *HLA-Speaking Service* described in the previous Chapter which interfaces *Migration Service* requests to the actual HLA application.

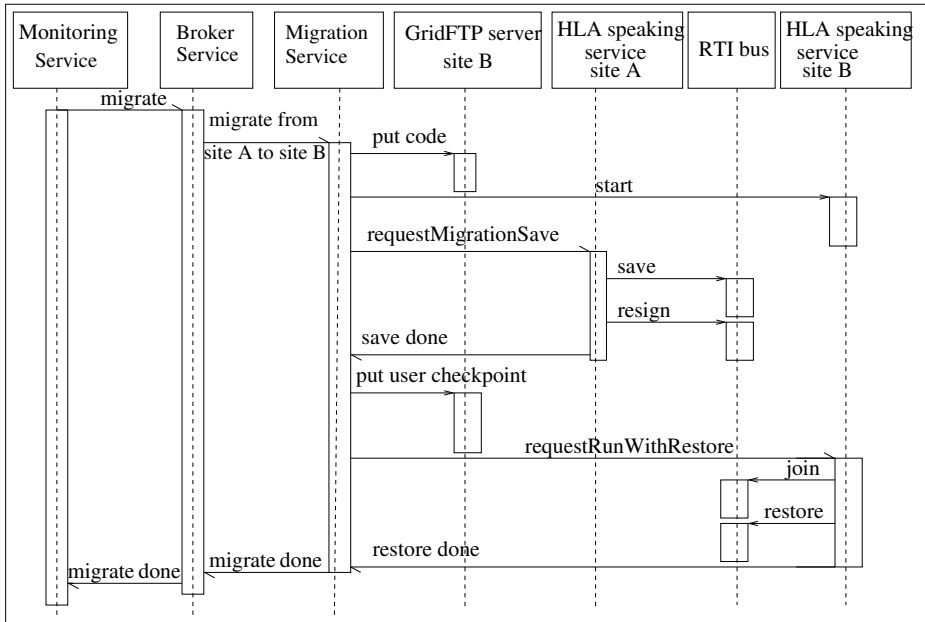


Figure 5.1: UML sequence diagram illustrating federate migration for a single federate scenario - cooperation between services of G-HLAM system and RTI.

If there is a need to migrate, the *Monitoring Service* informs the *Broker Service* which decides that the HLA federate from site A has to be migrated to site B and triggers the *Migration Service* to perform this migration. The *Migration Service* creates the *HLA-Speaking Service* on the new site, transfers federate code by means of the *GridFTP server* and invokes a "start" operation on that service which loads the code into a Java Virtual Machine via the JNI [88] mechanism.

When this is done, the *Migration Service* asks the *HLA-Speaking service* on site A to save the state. The HLA-speaking service forwards this request to its federate code by means of the GridHLAController Library (GHCL) library. The federate invokes an HLA API for federation saving, which suspends the other federates; user data is then saved in a checkpoint file and the federate resigns from the federation.

Subsequently, the *Migration Service* transfers the checkpoint files to site B and asks the *HLA-Speaking Service* there to restore the migrated federate from the available checkpoint file. The newly-created federate joins the federation and invokes GHCL library that restores user-specific data and uses the HLA API to restore the federation RTI internal state.

5.2.2 Multiple processes version

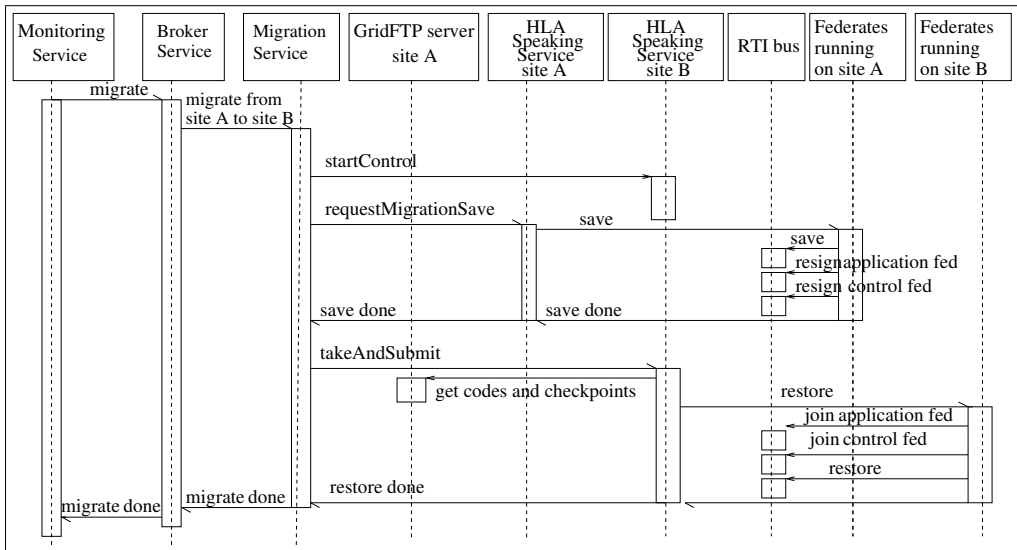


Figure 5.2: UML sequence diagram illustrating federate migration for multiple federates scenario – cooperation between services of G-HLAM system and RTI.

The multiple processes version scenario is shown in Fig 5.2. As previously, the services participating in migration include the *Broker Service*, the *Migration Service* and the *HLA-Speaking Service*.

If there is a need to migrate, the *Monitoring Service* informs the *Broker Service* which decides that the HLA federate from site A has to be migrated to site B and triggers the *Migration Service* to perform this migration. The *Migration Service* creates an *HLA-Speaking Service* on the new site, then asks for creation of a control federate which in turn creates a control federation.

When this is done, the *Migration Service* asks the *HLA-Speaking service* on site A to save its state. The HLA-speaking service forwards this request with the identifiers

of federates that have to be saved to all federates it controls by means of the GridHLA-Controller Library (GHCL). The federates invoke the HLA API for federation saving [78] which suspends other federates. The federates know by their identifiers whether there is a request to save state of their federations and resign from them (the identifiers are described in more detail in Section 4.3 of the previous Chapter).

Once this is done, user data is saved in a checkpoint file and the appropriate federates resign from the federations defined by the user (application federations) as well as from their control federation.

As a next step, the *Migration Service* invokes the `takeAndSubmit` operation of the *HLA-Speaking Service* on site B that triggers it to fetch the user code and checkpoint files from site A to site B and restore federates from the available checkpoint files. The federates join the control federation as well as user-defined (application) federations and invoke the GHCL library which restores user-specific data and uses the HLA API [78] for restoring the application federation RTI internal state.

5.2.3 Discussion

In Section 5.2 we presented migration scenarios for two versions of *HLA-Speaking Services*. As can be seen by comparing Fig. 5.1 and Fig. 5.2, the scenario with the service version for a single process does not include joining and resigning from a control federation during migration and therefore is less time-consuming. This is the price we pay for running separate processes (a solution which has many advantages, as described in Section 4.4 in the previous Chapter).

As shown later in Chapter 7, for the HLA RTI DoD implementation [77] the joining overhead is relatively big. However, this feature is specific to that implementation and the protocols used when joining federations. Since the control federate has local scope (one Grid site), the multicast mechanism can easily be used to reduce joining overhead.

5.3 Collaboration between the *HLA-Speaking Service* and user processes during migration

In this Section, we show how GHCL is used to communicate the *HLA-Speaking Service* with user federates on the example of a saving request. Restoring requests are processed in a very similar way and can be easily produced by small modifications of our example. The example is presented for two versions of the *HLA-Speaking Service*.

The functionality of the GridHLAController library (GHCL) is encapsulated within the GridHLAController class and apart from setup functions mentioned in Section 4.2.2, Section 4.3.2 and described in detail in Section B.1, it also supports migration mechanism. The library contains: group of functions checking if an external request (saving or restoring) from the *Migrator Service* has arrived, group of functions checking if an internal request (saving or restoring) from the *Migrator Service* has arrived,

and the group of routines for saving and restoring user-specific values. The detailed description of those routines can be found in Section B.2.

For our purposes we use UML collaboration diagram which models interactions between the elements of a system. The example is shown in Fig. 5.3. The messages

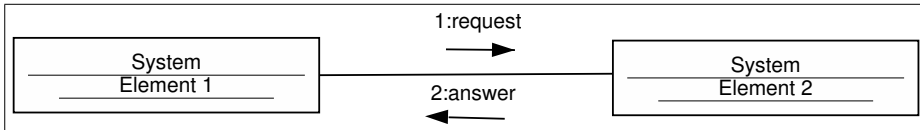


Figure 5.3: UML collaboration diagram example.

between elements are shown as arrows attached to the relationship lines. The sequence of messages is indicated by sequence numbers prepended to message description. In Fig. 5.3 Element 1 makes a request to Element 2 (step 1) and gets the answer from it (step 2).

5.3.1 Single process version

In Fig. 5.4 we describe communication between various levels of software when processing saving requests. In step 1 the *Migration Service* invokes the *requestMigrationSave* operation on the *HLA-Speaking Service*. Subsequently, in step 2, the *HLA-Speaking Service* uses the GHCL library to set up an appropriate state of its state machine. The state machine is checked by the user federate by means of the GHCL routine (step 3), which, in turn, informs the RTI bus about this fact (step 4). In the next (5th) step, other federates are informed in order to freeze and save their internal state. Once this is done, all necessary HLA-specific steps are performed in order to save the internal RTI state. Communication going back from RTI (step 6) proceeds through a callback provided by the user according to the HLA specification (step 7). The callback then invokes a GHCL library routine informing the library about the actual state of the saving process (step 8). When the RTI saving process is complete, the GHCL library saves the external state of user variables and informs the HLA service (step 9), which, in turn, informs the *Migration Service* (step 10).

5.3.2 Multiple processes version

The *HLA-Speaking Service* for multiple processes faces a more complicated task than in the single-process scenario. It has to distinguish between different jobs and to remember pids and nodes of all managed processes in order to save and restore appropriate ones. Therefore in this Section, we first present an algorithm for managing migration when dealing with multiple processes and then we describe a collaboration diagram showing how the *HLA-Speaking Service* interacts with user federates to process saving requests.

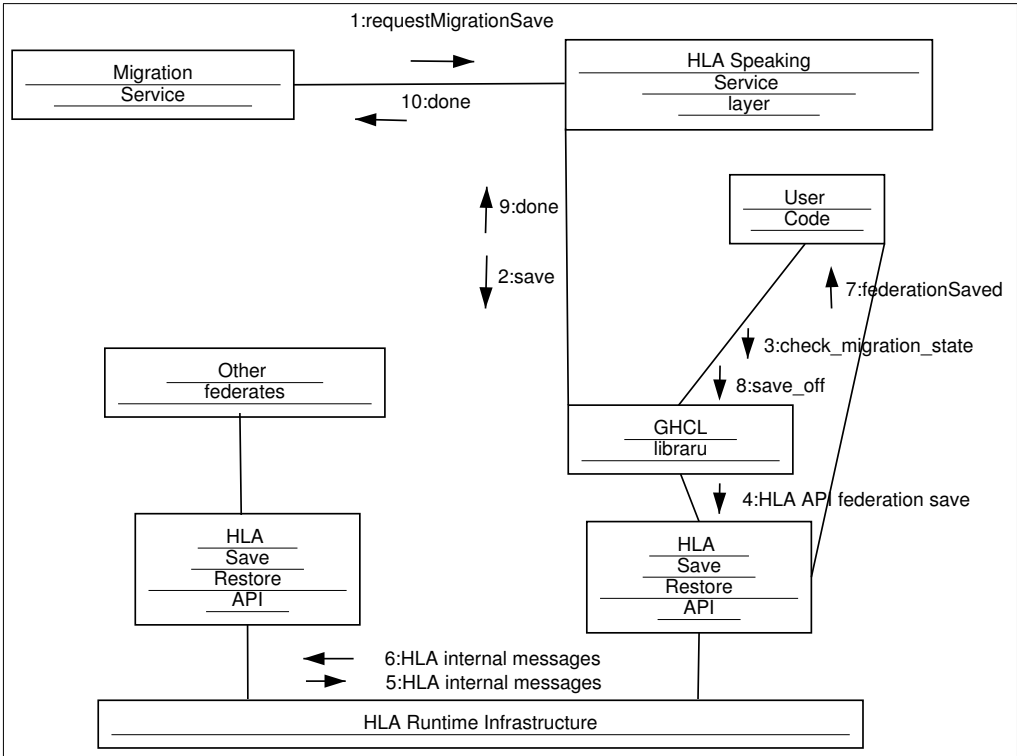


Figure 5.4: UML collaboration diagram for the Migration library, the RTI library and the *HLA Speaking Service* layer while executing a saving request.

Management algorithm

The multiple–process management algorithm, is shown in Fig. 5.5. We assume that jobs specified within one RSL cannot be separated by later migration. The jobs that can or should be separated should be given in separate RSL commands. In particular, we assume that all MPI processes that are started within the same simulation job on one site have to be migrated together since there is a high risk that separating MPI processes on different sites will significantly increase communication overhead. The other reason is that the MPI standard [115] does not provide the possibility of dynamically joining processes in a simple way.

Collaboration between HLA–Speaking Service and user federates

The collaboration diagram in Fig. 5.6 shows the control flow during migration. First, the *Migration Service* invokes the `requestMigrationSave` operation with identifiers of processes that need to be saved (step 1). Subsequently, the *HLA–Speaking Service* layer invokes the *Control Federation* (step 2) which uses the GHCL library (step 3)

1. The *HLA-Speaking Service* submits the given RSL to the the GRAM manager and at the same time generates an **id** of this submission and saves the (**id**, **RSL**) pair in a hash table.
2. Each process determines its pid and hosts and sends this information back to the *HLA-Speaking Service* by means of its control federation.
3. The *HLA-Speaking service* concatenates the received pids and hosts in a connection string and saves (**connection string**, **id**) pairs in a hash table.
4. If there is a request to migrate the process with a specific pid and host (e.g. from the *Application Monitoring Service*) the *HLA-Speaking Service* finds the id of the appropriate submission as well as its RSL command using the hash tables.
5. The *HLA-Speaking Service* sends a saving request together with the submission id to the processes it manages.
6. The processes recognize by their id whether they have to be migrated.
7. Once the saving is done, the *HLA-Speaking Service* returns RSL, hosts and pids of all processes that have to be migrated as output of the `requestMigrationSave()` function. This information is used by *Migration Service* for invoking the `requestRunWithRestore()` command (see description of the *HLA-Speaking* methods above).

Figure 5.5: Multiple-process management algorithm

to send save requests to other federates using HLA data management – namely HLA send interaction (steps 4-6). User federates receive interaction by means of the GHCL library that has previously joined them with the control federation (step 7). Next, user code learns that there is a need to save its state (step 8), similarly as in the previous example – by using the `check_migration_state()` routine of the GHCL library. The library uses HLA save federation API to save internal RTI state (steps 9-11) . Upon successful completion, RTI invokes the [78] callback in user code (step 12). The callback in turn invokes the GHCL library (step 13) which sends information back to the *HLA Speaking Service* (steps 14-19) and finally to the *Migration Service* (step 20).

5.4 Summary

In this Chapter, we described a migration mechanism which allows moving poorly-performing federates to different Grid sites and thereby improving the execution of the whole HLA application. We presented how G-HLAM services, introduced in Section 3.2.2, cooperate to perform migration. We also showed the migration functional-

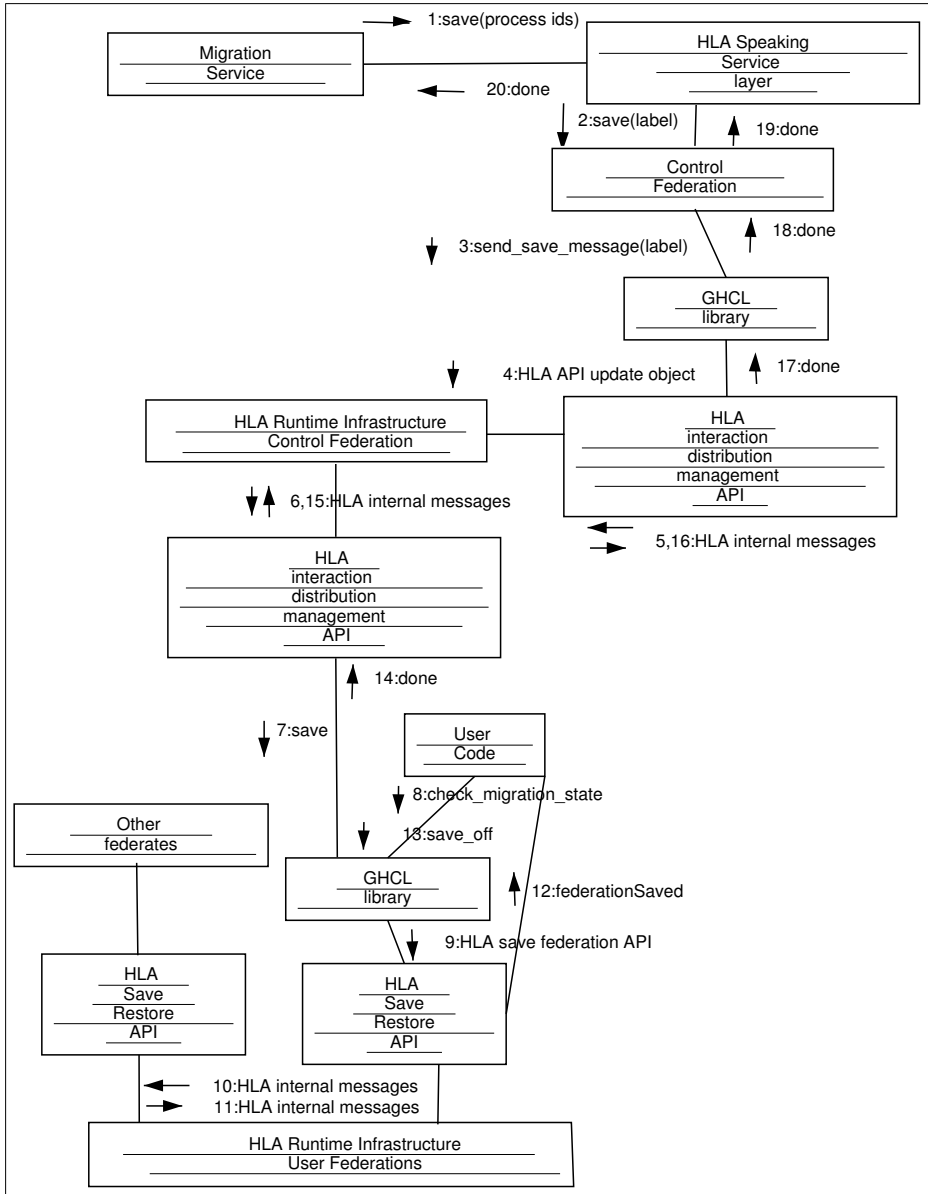


Figure 5.6: UML collaboration diagram for the Migration library, the RTI library and the *HLA-Speaking Service* layer for multiple processes.

ity of the GridHLAControler library (GHCL) and described collaboration between the *HLA-Speaking Service*, the library and managed user processes during migration.

As in the previous Chapter, we presented our solution for two versions of the *HLA-Speaking Service* – a single-process and a multiple-process version. For multiple processes we described a management algorithm implemented by the *HLA-Speaking Service* when performing migration of some of those processes.

Our main goal was to show how to satisfy the requirement for efficient execution of the application in a Grid environment in a scalable and universal way – our solution provides a mechanism of adapting application execution to changes in the Grid environment.

To summarize, the following problems were solved:

- scalability realized by distributed architecture of the G-HLAM system - each site is managed by a separate *HLA-Speaking Service* and *Migration Service*,
- building a universal solution – usage of GHCL does not force the user to port his/her HLA application to any higher-level library - he can easily add GHCL to existing HLA code,
- the ability to migrate some of federates without losing consistency of the whole application – achieved by using HLA save/restore API,
- management of multiple federates realized by the algorithm described in Section 5.3.2.

Performance results of the migration mechanism can be found in Chapter 7, Section 7.1.

The solutions presented in this Chapter can be easily extended for migration between different architectures under two conditions:

- the federate internal data checkpoint file must not be architecture-dependent (this issue is not specified by the HLA standard and is implementation-dependent)
- the user must provide the *Broker Service* with code compiled for different architectures. If this requirement is satisfied, the *Migration Service* can fetch the appropriate code from the Broker's repository instead of from the site of the migrating process.

User data can be restored as in the case of homogeneous environment, the checkpoint file (created by functions of the GHCL library) is not architecture-dependent.

Monitoring Services

This Chapter¹ describes the Broker Support Services shown in Fig. 3.4 in Chapter 3 – namely *Benchmark Services* and *Application Monitoring Services*. They are designed to help the *Broker Service* decide when and where migration should be performed.

The aim of these services is to fulfill the requirement for *efficient execution* of an HLA-based application in a Grid environment, as specified in Chapter 1, Section 1.3. They are used to provide information about planned (in the case of *Benchmark Services*) or existing (in the case of *Application Monitoring Services*) execution of distributed federates of the application.

The main problems addressed in this Chapter are:

- to provide information about HLA application performance on the Grid,
- to decide which part of Grid infrastructure is the best location for migrating poorly-performing federates,
- to support legacy HLA applications,
- to create scalable, flexible and secure solutions with little overhead.

In Section 6.1 we describe infrastructure monitoring services issue and present an HLA-based benchmark suite. This part of the G-HLAM gives information about

¹The results described in this Chapter formed the basis to the following papers:

K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Support for Effective and Fault Tolerant Execution of HLA-Based Applications in the OGSA Framework. In M. Bubak, D. G. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004. 4th International Conference, Kraków, Poland, June 2004*, number 3038 in LNCS, pages 848–855. Springer, 2004.

K. Rycerz, B. Baliś, R. Szymacha, M. Bubak, and P. M. A. Sloot. Monitoring of HLA Grid Application Federates with OCM-G. In S.J. Turner, D. Roberts, and L. Wilson, editors, *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04), IEEE, Budapest, Hungary, October 2004*, pages 125–132, 2004.

planned execution of HLA application and helps the *Broker Service* decide where application federates should be placed. Although various monitoring tools are currently being developed [104, 65], it is extremely difficult (if not impossible) to monitor Wide Area Network (WAN) connections (e.g. different data packages going from the same source to the same destination can be actually routed differently). Furthermore, the actual communication between components of the HLA-based simulation depends on an implementation of the HLA standard. The tuple space concept hides the actual path taken by messages sent from publishers to subscribers. The solution proposed in this thesis is to perform on-line HLA-based benchmarks and make decisions about migration based on their results.

In Section 6.2 we present the *Application Monitoring Service* which monitors the performance of already running HLA-based applications and helps decide when to migrate its parts to achieve more efficient execution. According to the HLA standard [78], HLA-based applications can be monitored and controlled by the Management Object Model (MOM). Although MOM allows for monitoring invocations of HLA functions, it does not enable the programmer to introduce performance metrics (e.g. invocation time). We have decided to use the OCM-G [12] as a monitoring system, since it is already Grid-enabled and has a standardized interface to the potential monitoring tool (On-line Monitoring Interface Specification (OMIS) [103] protocol). Other solutions like PerfMETRICS [30], although useful, do not have those features.

We conclude the Chapter in Section 6.3. Thanks to the Service-Oriented Architecture of G-HLAM, it is easy to plug in existing solutions, if they are available. In the Grid community there is ongoing research on metascheduling systems [113, 71, 187] that can be extended for our purposes and used as a *Broker Service*. This issue, however, is out of scope of this thesis.

6.1 Obtaining information from the infrastructure

In this Section we focus on the part of the framework that supplies the *Broker Service* with information about HLA performance in the given part of the Grid infrastructure. This is particularly useful if we would like to migrate a part of an application to a new site and check if the site is not overloaded and does not have busy connections with other sites. This is different from the situation described in Section 6.2, when an actual application is monitored, so we can see its performance results online.

In order to make brokerage decisions, the system needs to monitor hosts and the network infrastructure. As said above, although various monitoring tools are currently being developed [104, 65], it is extremely difficult to monitor WAN connections. Furthermore, HLA API conceals actual communication between components of the distributed simulation. The tuple space concept hides the actual paths of messages sent from publishers to subscribers.

One possible solution of these problems is to perform on-line HLA-based benchmarks and make decisions about migration basing on their results.

6.1.1 Functionality of general *Benchmark Service*

The *Benchmark Service* should be able to perform measurements that can be then interpreted by the *Broker Service* to find the best location in which to place HLA federates. We focus here on communication performance measurements. Information about host load is also important, but many relevant solutions already exist and are utilized in various Grid systems [99, 104].

In general, the *Benchmark Service* exposes the following operations:

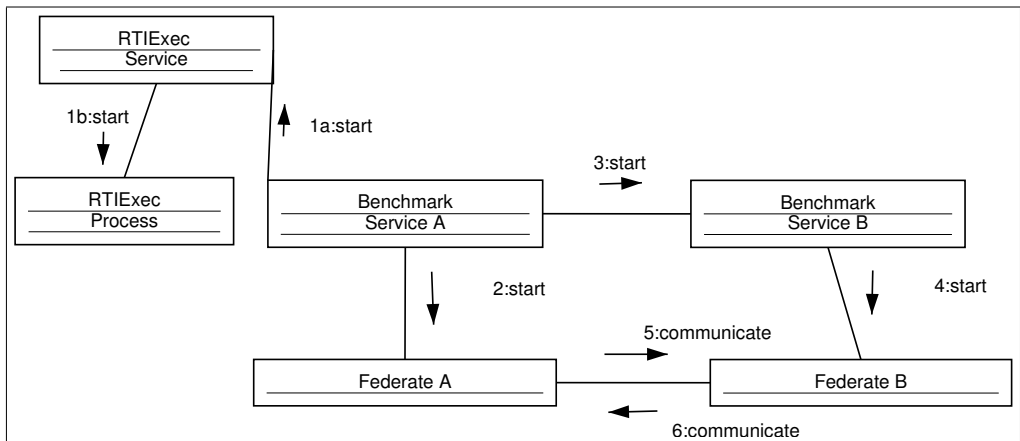


Figure 6.1: UML collaboration diagram of *Benchmark Services* in the Grid services framework – interactions between services and actual benchmark federate processes.

- *start* – if the service is the one which initiates measurements (as Service A in Fig. 6.1), it starts or connects to the existing *RTIExec Service* (step 1 in Fig. 6.1), creates a federation and starts a benchmark federate on its site (step 2). If the service is passive (as service B in the figure), it only starts its federate (see below)
- *measurement set of operations* – the service can implement various operations that perform particular measurements. In general, this mode of operation is used to cooperate with other *Benchmark Services*. The service invokes the *start* operation of the *Benchmark Services* (step 3) which sets up their federates by connecting them to the same federation (step 4). Following that, the *Benchmark Services* cooperate to performs measurements (steps 5, 6), which can include performance of various HLA communication functions such as sending data objects, interactions or transferring ownership as well as more complicated communication schemes. The measurement are repeated to achieve better statistics.
- *scan* – performs the chosen *measurement* operation with the selected list of *Benchmark Service* sets . If there is no response before a specific timeout, the *Benchmark_IVC Service* proceeds to the next site.

- `forscan` – performs the `scan` operation from start time to stop time for each time interval.

This type of supporting measurements is exposed in service data of the Benchmark Service, so two services can be matched (e.g. by the *Broker Service*). It contains information on whether the measurement provided is active (initiated by that service) or passive (the service has to wait for the active counterpart to be started) and also the name of the measurement. The services which can cooperate by running their federates within the same federation share the same measurement name or names.

6.1.2 *Benchmark Service set for interactive simulation*

This Section presents a sample *Benchmark Service* designed to measure the performance of interactive human-in-the-loop simulations. The example is based on the architecture of medical application developed in CrossGrid [155] and can be used to examine possibilities of adapting a Grid environment for this kind of applications.

The application consists of three components : interaction, simulation and visualization. While simulation is being conducted, the user can change its parameters using the interaction module (sending small messages from interaction to simulation) and see results through visualization (simulation sends large updates to visualization). According to this scenario, our benchmark consists of two modules: an integrated Interaction-Visualization Module (IVMod) and a Simulation Module (SMod).

IVMod sends small messages that are human interactive requests for change of SMod parameters. SMod responses are large messages that present the altered state of the simulation to the user.

The HLA benchmark federation consists of two federates, acting as IVMod and SMod respectively. IVMod sends SMod requests in which it asks for certain amounts of data, then measures the time between sending the request and receiving a response. Messages are sent through updates of HLA data objects [78].

The benchmark is managed within the Grid services framework by the *Benchmark_IVMod Service* and the *Benchmark_SMod Service*, which start, control and measure the performance of IVMod and SMod. The *Benchmark_SMod Service* is passive and does not initiate measurements – its role is to start and control SMod federates. This is done by invoking its `start` operation.

The *Benchmark_IVMod Service* is active with a measurement operation called `check`. Operation `check` starts an IVMod federate and submits requests to a specific *Benchmark_SMod Service* by invoking its `start` operation which sets up the SMod part of the benchmark and connects it to the same federation. Following that, the *Benchmark_IVMod Service* initiates measurements between the IVMod federate and the SMod federate as described above.

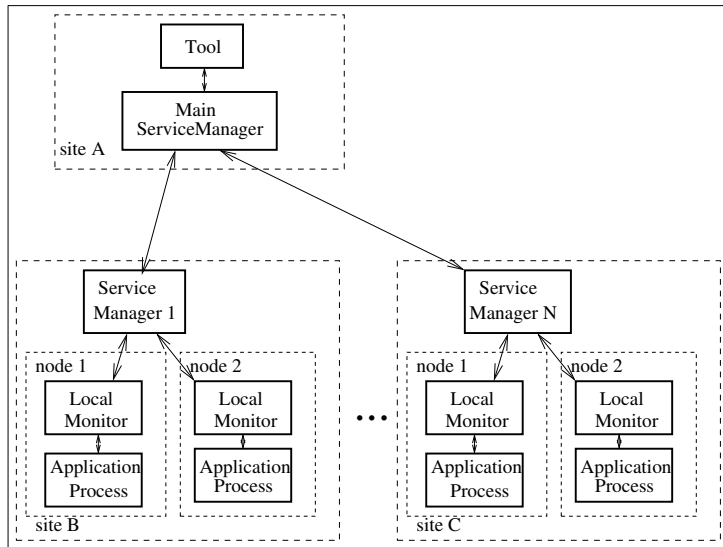


Figure 6.2: Hierarchical architecture of the OCM-G monitoring system [12]. Local Monitors are used to connect the system to monitored processes. Tool gets monitoring data from Main Service Manager.

6.2 Monitoring of HLA Grid application federates with OCM-G

Apart from using benchmarks to judge infrastructure state as described in the previous Section, there is a need to monitor the HLA application itself, to learn when to improve its performance and enable to more efficient execution by migration of federates. As stated above, MOM available in HLA standard [78] allows for monitoring invocations of HLA functions, but it does not enable the programmer to introduce performance metrics.

For our purposes we have chosen the OCM-G system [12] as a monitoring system, since it was designed to be executed in a Grid environment and complies with the On-line Monitoring Interface Specification (OMIS) [103]. Other solutions [30], although useful, do not have those features.

We show how the design concepts of OCM-G enable easy adaptation to monitoring of HLA applications. We also describe the role of monitoring in the whole system for managing execution of HLA-based applications. We discuss implementation issues and present test results for monitoring overhead.

6.2.1 Grid-enabled OMIS-Compliant Monitor

The Grid-enabled OMIS-compliant Monitor (OCM-G) is an infrastructure which enables runtime monitoring of Grid applications. The OCM-G is an autonomous, distributed, decentralized system which exposes monitoring services via a standardized interface called OMIS [103].

The OCM-G is composed of two types of components: per-host Local Monitors and per-site Service Managers. Additionally, there is a Main Service Manager which distributes data to and collects it from site-SMs. The architecture of the OCM-G is shown in Fig. 6.2 [12]. The Main Service Manager is the component which exposes the functionality of the system to performance analysis tools. Thanks to the standardized protocol used between the monitoring system and possible tools, OCM-G can be very easily adapted to the architecture described in Chapter 3, Section 3.2.

The most important features [12] of the OCM-G include:

- *Low monitoring overhead.* This is achieved, among others, by: (1) local buffering and preprocessing of monitoring data, (2) selective runtime instrumentation which can be activated and deactivated at runtime; allowing users to focus on the very set of information that is interesting at a given moment as well as to dynamically change the scope of interest as the application execution progresses. Overall, these concepts enable a significant data rate reduction.
- *Flexibility.* The OCM-G provides a broad range of monitoring services which include support for debugging, performance analysis, visualization, etc. The services are easy to use, yet relatively low-level. The OCM-G does not provide high-level metrics with fixed semantics. Instead, a combination of several services is used to obtain a specific metric. This allows the user to derive metrics with the desired semantics and it also enables user-defined metrics.
- *Security.* The OCM-G uses Globus GSI to enable security in the Grid environment. Moreover, each user runs his/her own instance of the OCM-G which further increases security.
- *Transparent service-oriented operation.* The involvement of the user in the process of preparing his application for monitoring and setting up the monitoring system is reduced to a minimum. First, the user does not need to manually instrument the application (except the special case of user-defined events, when there is no other way to do this); instead, pre-instrumented libraries are provided. Second, the manner of compilation and submission of the application is hardly changed by monitoring, except by using wrappers for compilation and passing additional command-line parameters.

OCM-G has been successfully used with an independent performance analysis tool for Grid applications, namely G-PM [22].

6.2.2 Monitoring HLA-based applications

[12] describes how OCM-G is used to monitor MPI applications. In this Section we describe adapting these solutions to different constraints defined by HLA RTI-based application features. Below, we describe the architecture and functionality of the monitoring system for HLA applications.

Architecture

The architecture of the HLA application monitoring system is presented in Fig. 6.3. We insert additional classes (wrappers) between HLA applications and *RTIAmbassador*/*FederateAmbassadors*. These wrappers communicate with original ambassadors, and send all required monitoring data (invoking HLA methods) to the OCM-G using the OMIS protocol. All these operations are completely transparent to the user application and are performed with no changes to the original HLA RTI code. Our

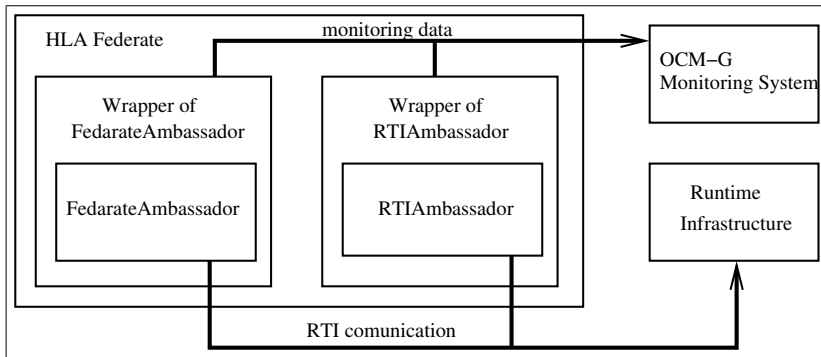


Figure 6.3: Architecture of OCM-G based monitoring system for HLA-based application. Wrappers are used to connect HLA legacy code with OCM-G.

solution depends only on the HLA interface specification, which is well defined and standardized [78], as well as independent of any HLA implementation, so it can be applied to any version of the implementation without any changes to its original installation.

The HLA Monitoring Service is almost transparent to the HLA application. The user only needs to insert one additional line into the source code, initializing the connection to the OCM-G system.

Our system also supports using OCM-G to monitor other user events, e.g. it can send all kinds of messages (integers or floating numbers, character strings) to the monitoring application. This is done using *probes* as described below.

Description of functionality

By using OCM-G, we can gather all information required by the *Performance Decision Service* which communicates with the *Main Service Manager* using the OMIS[103]

protocol. The system can monitor invocations of all HLA services: federation management, data management, data distribution management, time management and ownership management. The system allows for gathering data required for performance measurements such as the amount of data sent, time of data transmission and process identifiers. This enables the *Performance Decision Service* to discover which federates participate in communication within the HLA tuple space and to decide if their location on the Grid is sufficient for achieving the performance required by the user in a processing loop.

The OCM-G architecture allows to control the amount and frequency of monitored data sent from the monitored application to *Performance Decision Service*. It is scalable and suitable for running in the Grid environment. To send data to the OCM-G, we use *probes* in the following way:

- the original method is invoked,
- the HLA-Monitoring Service sends information about node address, time, and operation parameters, using the OMIS protocol.

Of course in order to avoid unnecessary traffic and overhead, data is sent to the monitor only if required by a consumer. This is possible thanks to the OCM-G architecture which allows for selective runtime instrumentation that can be activated and deactivated at runtime. The flow of monitored data is shown in Fig. 6.4 as thick lines. The control flow, external to OCM-G, is depicted as thin lines.

To allow the system presented in Section 3.2 to monitor HLA-based applications, specific steps have to be performed, as shown in Fig. 6.4.

- OCM-G should be initialized in the application code,
- *OCM-G Main Service Manager* should be up and running,
- An application should be started (this step should be performed by the *HLA-Speaking Service* chosen by the Broker Service) with additional parameters pointing to the *Main Service Manager*. During this step, the *Service Manager (SM)* and the *Local Monitor(LM)* are created automatically, if needed, by the application process, or the process can connect to SMs and LMs that have already been created beforehand.
- Now, any component that can communicate using the OMIS [103] protocol, such as the *Performance Decision Service*, can connect to the *Main Service Manager*, subscribe to the required probes and receive monitored events to make decisions about moving the application to another Grid node if necessary.
- If necessary, the *Performance Decision Service* can inform the *Broker Service* about bad performance, causing it to choose another *HLA-Speaking Service*.

As mentioned before, the HLA Monitoring System enables the use of the OCM-G to monitor user events by inserting custom probes into the code. These probes can be

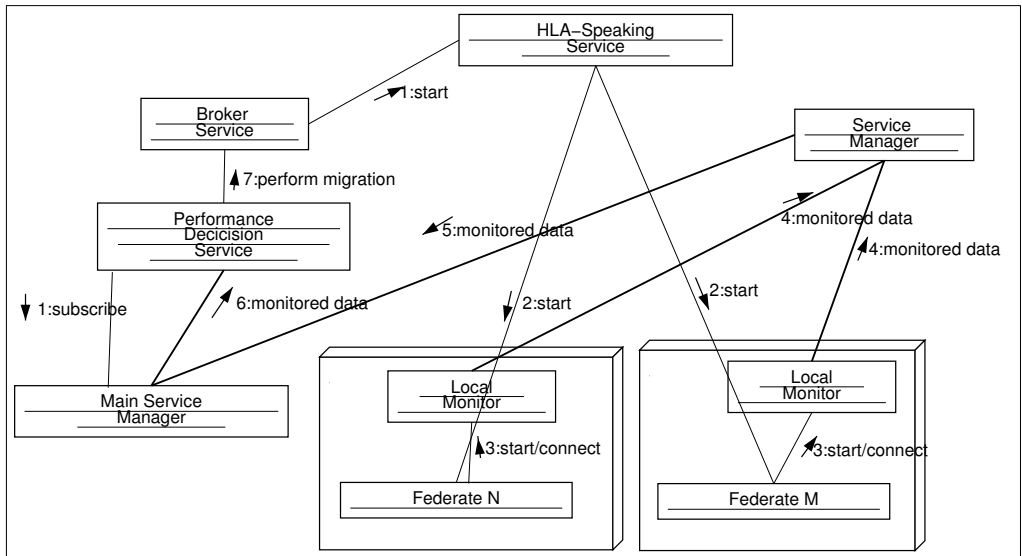


Figure 6.4: Workflow (thin lines) and dataflow (thick lines) during setup of OCM-G based monitoring of HLA applications.

monitored just like any other HLA probes, as described above. In order to subscribe and monitor interesting events, the user can use the *ocmg-tool* which connects to the *Main Service Manager* using the OMIS protocol in the same way as the *Performance Decision Service*.

Implementation

Because HLA is a C++-driven API, OCM-G cannot be used, since its purpose is to monitor MPI applications and it changes the structure of MPI libraries. The structure of a C++ program is more complicated than that of a C program and it may be compiler-dependent. This is why we had to create a way of automatically changing user source code rather than binary libraries (as had been done with MPI via instrumenting MPI libraries). This was accomplished by changing header files and class names using macros. These macros allow us to make the system transparent to the user, who only interacts with the original application.

6.3 Summary

The goal of this Chapter was to present Broker Support Services introduced in Section 3.2 – namely *Benchmark Services* and *Application Monitoring Services*. These services are used to provide efficient execution of HLA-based applications on the Grid in a scalable and transparent way and therefore satisfy the requirements presented

in Section 1.3. The former service can be used to determine where federates should be located to achieve the best performance and the latter indicates when and which of the federates need to be migrated due to poor performance.

The benchmarking case described here is based on a simple interactive simulation scenario, however the general design of *Benchmark Service* interface allows for easy plugging in of other benchmarks. Furthermore, as the architecture of the whole system is based on Grid services, it becomes straightforward to apply external infrastructure monitoring services if they expose their interfaces as web or grid services (i.e. JIMS [104]).

Application Monitoring Service functionality is crucial to decide when to migrate federates (and also which federates to migrate) to achieve more efficient execution. We use OCM-G [12] as a monitoring system, since it is designed for Grid infrastructures and conforms to the existing OMIS monitoring standard [103].

To summarize, the problems mentioned at the beginning of this Chapter have been solved as follows:

- providing information about HLA application performance on the Grid is realized by using OCM-G, which is a Grid-enabled, scalable, flexible and secure monitoring system,
- universal, transparent support for monitoring HLA applications is realised by the monitoring architecture which takes advantage of OMC-G and can be easily used by legacy simulations,
- finding the best location for migrating federates is realized by running *Benchmark Services*. In our case, HLA-based benchmarks have an advantage over monitoring of network connections (such as in [104, 65]). This is due to the high-level communication infrastructure of HLA acting as a tuple space that hides the actual point-to-point communication and therefore makes it difficult to determine which connections are actually used.

Thanks to the presented solutions, we are able to provide *Monitoring Services* fulfilling the requirement for *efficient execution* of HLA-based simulations on the Grid.

The performance results of *Benchmark Services* and the overhead of the OCM-G based application monitoring system will be presented in Chapter 7, Sections 7.2 and 7.3.

Performance Results and Their Analysis

This Chapter¹ presents detailed G-HLAM system performance results. Our implementation consists of the most important services of the system: the *HLA-Speaking Service* described in Chapter 4 and Chapter 5, the *Migration Service* described in Chapter 5, and the *Benchmark Services* together with *Application Monitoring Services* described in Chapter 6.

The goals of performance tests are as follows:

- to show that the system satisfies requirements listed in Section 1.3,
- to examine the performance of particular elements of G-HLAM before the system is applied to real applications (as presented in Chapter 8 and Chapter 9).

Grid technology is constantly evolving (see Section 2.3.1), hence our experiments were performed using various Grid software versions available at the time. The first implementation of single-process version of the *HLA-Speaking Service* was done in GT3 Alpha-3, the next version (for multiple processes) used GT v3.0.2 and later we upgraded both versions to GT v.3.2. We applied GT3.2 when implementing the *Migration Service*, *Benchmark Services* and *Application Monitoring Services*, where we also used the OCM-G monitoring system developed within the CrossGrid project [12].

¹The results described in this Chapter formed the basis to the following papers:

K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Framework for HLA-Based Interactive Simulations on the Grid. *SIMULATION*, 81(1):67–76, 2005.

K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Support for Effective and Fault Tolerant Execution of HLA-Based Applications in the OGSA Framework. In M. Bubak, D. G. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004. 4th International Conference, Kraków, Poland, June 2004*, number 3038 in LNCS, pages 848–855. Springer, 2004.

K. Rycerz, B. Baliś, R. Szymacha, M. Bubak, and P. M. A. Sloot. Monitoring of HLA Grid Application Federates with OCM-G. In S.J. Turner, D. Roberts, and L. Wilson, editors, *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, IEEE, Budapest, Hungary, October 2004, pages 125–132, 2004.

Our hardware configurations were conveniently adapted from the DAS2 [49] testbed infrastructure and the part of the CrossGrid [38] testbed located at CYFRONET. We have chosen DAS2 sites that were available and not overloaded at the time of performing measurements. All sites of DAS2 are very similar so configuration differences should not have any significant impact on results. The CYFRONET part of the infrastructure was always the same. For our experiments we used HLA RTI 1.3v5.

This Chapter is organized as follows: in Section 7.1 we analyse various experiments with the migration mechanism, in Section 7.2 we show *Benchmark Services* results, and finally, in Section 7.3 we measure the overhead of *Application Monitoring Services*. The summary and conclusion can be found in Section 7.4.

7.1 Migration overhead

In this Section we analyse migration overhead. We focus on the single-process version of the *HLA-Speaking Service* because, although simpler, it allows us to verify the concept of improving simulation performance through migration.

The overhead of the version for multiple processes will be presented in the next two Chapters showing the use of G-HLAM for two real applications.

We present impact of different operations performed during migration on its overhead and show how migration can actually improve the performance of interactive simulations.

7.1.1 Overhead of migration operations

In this Section we show the impact of various HLA RTI operations that are performed during migration on the migration overhead. The scenario of migration is described in Section 5.2 and it is as follows: The *Migration Service* creates an *HLA-Speaking Service* on the remote site which it wants to migrate to; subsequently it transfers user code and loads it into the *HLA-Speaking Service* (via JNI). These steps are invisible to the federates. Next, the *Migration Service* requests the local *HLA-Speaking Service* to save its state and transfers the checkpoint file to the remote resource. Finally, it invokes the *restore* operation on the remote service.

The total migration time is given by the following formula:

$$t_{migration} = t_{save} + t_{resign} + t_{transfer} + t_{join} + t_{restore} \quad (7.1)$$

where :

- $t_{migration}$ is the total migration time,
- t_{save} is the time of saving the state of the federation,
- t_{resign} is the time of resigning from the federation,
- $t_{transfer}$ is the time of transferring checkpoints,

- t_{join} is the time of joining a federation following migration
- $t_{restore}$ is the time of restoring state.

Our goal was to measure scalability and answer the question of how the presented values depend on the number of federates in the federation.

Operating System	Red Hat Linux 7.2		
Network	10 Gbps (DAS2) + 155 Mbps (DAS2-CYFRONET)		
Role in experiment 1	Name	CPU	RAM
Migration source	DAS2 Nikhef	Pentium III 1 GHz	1 GB
Migration dest	DAS2 Delft	Pentium III 2GHz	2 GB
Receiving federates	DAS2 Leiden	Pentium III 1 GHz	2 GB
Receiving federates	DAS2 Utrecht	Pentium III 1 GHz	1 GB
Receiving federates	DAS2 VU	Pentium III 1 GHz	2 GB
RTIExec	Cyf Krakow	Xeon 2.4 GHz	1 GB

Table 7.1: Grid testbed infrastructure for Experiment 1

In our experiment the HLA federation consisted of N federates (one publisher and $N-1$ subscribers). We have measured how long the subscribers have to wait for publisher migration. The detailed setup of the experiment is shown in Table 7.1. We measured the impact of various HLA RTI operations that are performed during migration as described in Section 5.2 on the overhead of this migration. The results were averaged over 10 measurements. Error bars indicate the estimated standard deviation.

Saving overhead

Saving overhead is caused by internal federation-wide saving algorithm used to implement the HLA Save/Restore API [78]. It includes synchronization mechanism between federates, which assures that the state does not change while saving. RTIExec control process saves the internal states of all federates in a separate checkpoint files. Also state of FedExec control process (which is one per federation) is saved. Additionally, moved federate saves user data to a checkpoint file.

The results in Fig. 7.1 show that the initial saving overhead independent on number of federates is relatively large in comparison to the growth caused by increasing federates number. This is probably because the initial overhead is dependent on the time of saving user data (which is done only by migrating federate) and saving FedExec process state. Also, the algorithm used by RTIExec to save internal RTI state can introduce some internal overhead.

Additionally, we can observe plateaus in the plot - that probably means that RTIExec uses buffering to process requests coming from federates during saving process.

We have approximated migration overhead by choosing the middle points of the plateaus and finding an approximating linear function for them given by the formula

$$t_{save} = a_{save}n_f + b_{save} \quad (7.2)$$

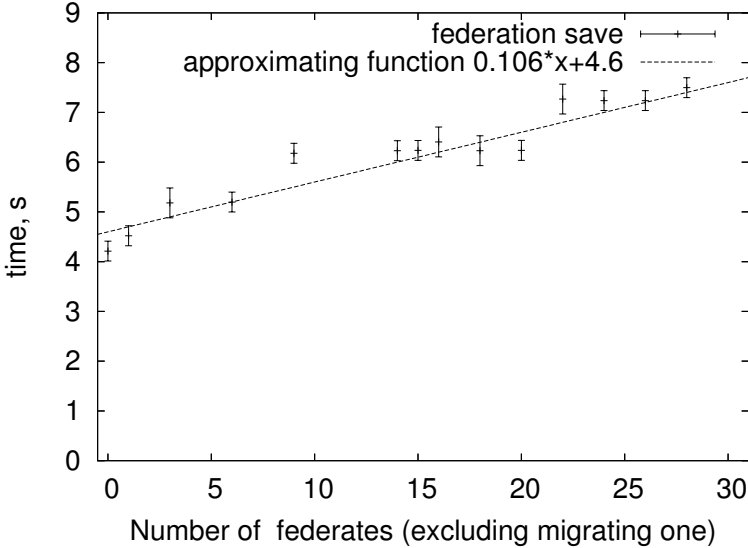


Figure 7.1: Time of saving federation state in Experiment 1 as a function of the number of receiving federates

where n_f is the number of federates. We used the least square method to calculate $a_{save} = 0.106$, $\sigma_a = 0.003$, $b_{save} = 4.6$ and $\sigma_b = 0.1$. The correlation coefficient for all experimental data is 0.958.

Resigning overhead

The complexity of the resigning operation in the HLA RTI implementation when using reliable transport should be linear with respect to the number of federates, because the resigning federate should close all TCP connections to all other federates. Fig. 7.2 shows that resigning time indeed depends linearly on the number of federates. The approximating linear function is given by the following formula:

$$t_{resign} = a_{resign}n_f + b_{resign} \quad (7.3)$$

where n_f is the number of federates. For experimental data we calculated $a_{resign} = 0.3$, $\sigma_a = 0.04$, $b_{resign} = 1.22$ and $\sigma_b = 0.01$ using the least square method. Correlation coefficient is equal 0.996.

Transfer overhead

Transfer time depends on the size of user data checkpoint file. The checkpoint files with internal state of RTI are located at the RTIExec site and do not have to be moved. In our experiment, the total size of user data checkpoint file is not dependent on total federates' number. Transfer overhead equals 3.0 sec with $\sigma = 0.1$.

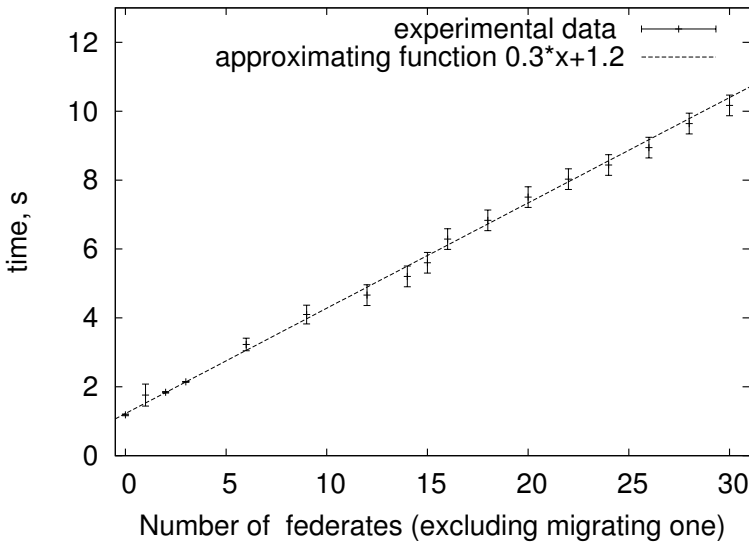


Figure 7.2: Time of resigning from federation during migration in Experiment 1 as a function of the number of receiving federates

Joining overhead

During joining operation the federate informs RTIExec process that has to update information about new federation member and opens TCP connections to all other federates in a federation. Therefore, the joining overhead should be linear with number of federates when using reliable transport in the HLA RTI implementation. Fig. 7.3 shows that the joining federation time depends linearly on the number of federates. The approximating linear function is given by the following formula:

$$t_{join} = a_{join}n_f + b_{join} \quad (7.4)$$

where n_f is the number of federates. For experimental data we calculated $a_{join} = 6.1$, $\sigma_a = 0.1$, $b_{join} = -21$ and $\sigma_b = 3$ using the least square method. Correlation coefficient is equal 0.997. The approximation does not apply to small number of federates because the time of opening small number of sockets (which is the factor that causes linear behavior) is not long enough to dominate time needed for other activities (such as updating RTIExec database of federation members).

Restoring overhead

Restoring overhead is caused by internal federation-wide restoring algorithm used to implement the HLA Save/Restore API [78]. As in the case of saving, it includes synchronization mechanism between federates assuring that the state does not change

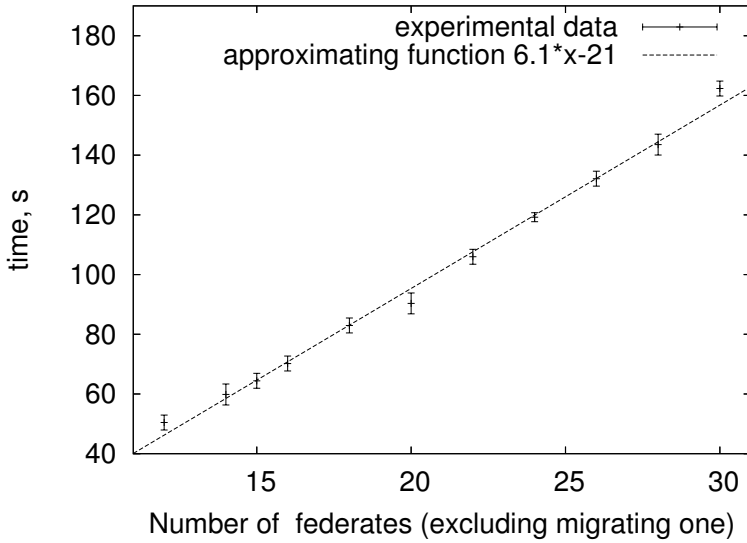


Figure 7.3: Time of rejoining to federation during migration in Experiment 1 as a function of the number of receiving federates

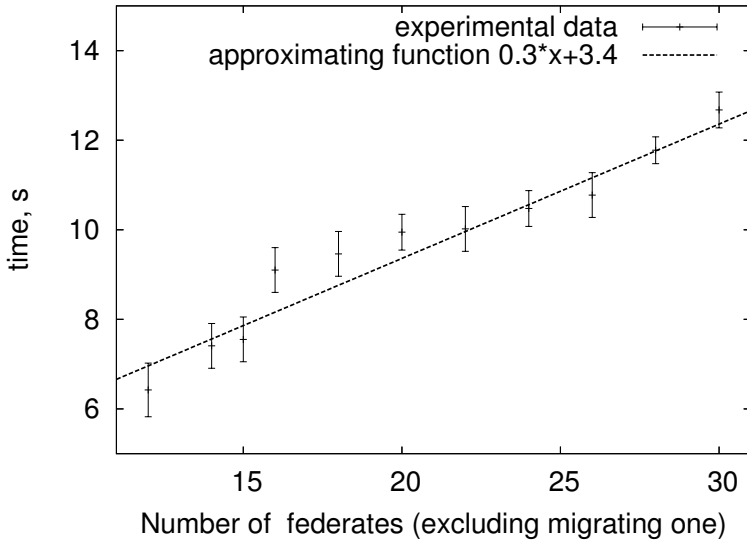


Figure 7.4: Time of restoring of federation state during migration in Experiment 1 as a function of the number of receiving federates

while restoring. RTIExec control process restores the internal states of all federates from checkpoint files. Also state of FedExec control process (which is one per federation) is restored. Additionally, moved federate restores user data from its checkpoint file.

The experimental data shown in the Fig. 7.4. We can observe plateau in the plot - that probably means that RTIExec uses buffering to process requests coming from federates during restoring process. We have found an approximating linear function basing on the middle points of plateaus given by the following formula:

$$t_{restore} = a_{restore}n_f + b_{restore} \quad (7.5)$$

where n_f is the number of federates. For experimental data we calculated $a_{restore} = 0.3$, $\sigma_a = 0.02$, $b_{restore} = 3.4$ and $\sigma_b = 0.4$ using the least square method. The correlation coefficient is equal 0.968

Total migration overhead

From expression 7.1, the total migration time equals:

$$t_{migration} = (a_{save} + a_{resign} + a_{join} + a_{restore}) * n_f + b_{save} + b_{resign} + b_{join} + b_{restore} + t_{transfer} \quad (7.6)$$

With values obtained in the previous Section we can calculate:

$$t_{migration} = 6.8 * n_f - 9 \quad (7.7)$$

and standard deviations:

$$\sigma_a = 0.1, \sigma_b = 3 \quad (7.8)$$

It can be noticed that the main migration overhead (the most significant factor a) is caused by rejoining the federation after migration. This is due to the fact that this operation requires opening TCP sockets to all other federates in the federation, which is costly. If multicast is used, one would expect a reduction of that overhead. Unfortunately, multicasts are not popular in WANs. A promising approach has been taken up by the XMSF network group which is developing multicasts suitable for HLA applications executed in WANs [185].

On the other hand, the values of $a_{migration}$ and $b_{migration}$ can be obtained directly from experimental data. Fig. 7.5 shows that restoring federation time depends linearly on the number of federates. The approximating linear function is given by the following formula:

$$t_{migration} = a_{migration}n_f + b_{migration} \quad (7.9)$$

where n_f is the number of federates. For experimental data we calculated $a_{migration} = 6.7$, $\sigma_a = 0.1$, $b_{migration} = -4$ and $\sigma_b = 2$ using the least square method. The correlation coefficient equals 0.9979.

Comparing it to the values of a and b obtained from the expression 7.7, we can see that the both results are the same in the range of error bars.

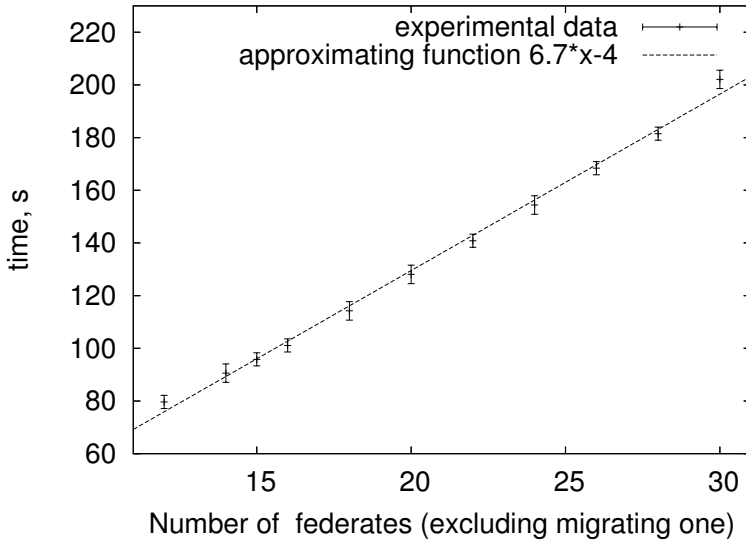


Figure 7.5: Time of migration in Experiment 1 as a function of the number of receiving federates

7.1.2 Impact of migration on the performance of an interactive application

In this Section we show how migration can improve the performance of an application with a human in a processing loop. The detailed setup of the experiment illustrating the migration impact on performance is shown in Table 7.2. For this configuration we chose DAS2 sites that were available and not overloaded at that time. To learn

Operating System	Red Hat Linux 7.2		
Network	10 Gbps (DAS2) + 155 Mbps (DAS2-CYFRONET)		
Role in Experiment 2	Name	CPU	RAM
Migration dest	DAS2 Nikhef	Pentium III 1 GHz	1 GB
Migration source	DAS2 Utrecht	Pentium III 1 GHz	1 GB
Interaction Federate	DAS2 VU	Pentium III 1 GHz	2 GB
RTIExec	Cyf Krakow	Xeon 2.4 GHz	1 GB

Table 7.2: Grid testbed infrastructure for Experiment 2

what can be expected from application performance in a simple interaction – simulation scenario we implemented a sample interactive test application consisting of two HLA federates: an interaction federate and a simulation federate. The interaction is performed as follows: the interaction federate asks the simulation federate for a large amount of data (in our case 0.4 GB) and measures the time between this re-

quest and the response from the simulation. Each response was in fact divided into 1 MB chunks of data due to memory allocation issue – HLA-RTI requires four times more memory than the size of the actual sending/receiving buffer. This is the problem of the particular implementation of HLA (which was used in our experiments), not the HLA standard itself. We suspect that RTI internal data structures, which are useful for implementing data management and data distribution management advanced features are not optimized for sending large volumes of data encapsulated in one object or interaction.

If we define the following:

- $\langle t_{before} \rangle$ is the average interaction time between a simulation and an interaction before migration
- $\langle t_{after} \rangle$ is the average interaction time after migration,
- $\langle t_{before} \rangle > \langle t_{after} \rangle$
- $t_{migration}$ is the migration time
- n_{steps} is the number of question–answer pairs (interactive steps in the loop with a human).

then the migration becomes affordable when:

$$\langle t_{before} \rangle * n_{steps} > \langle t_{after} \rangle * n_{steps} + t_{migration} \quad (7.10)$$

and the number of steps:

$$n_{steps} > \frac{t_{migration}}{\langle t_{before} \rangle - \langle t_{after} \rangle} \quad (7.11)$$

Fig. 7.6 shows the execution time as a function of number of interactive steps. The dotted line shows the time of request–response steps in the case when the simulation was not migrated. In order to create conditions in which migration would be useful, we increased the load of the Grid site where the simulation was executed (Utrecht) by submitting nonrelated jobs. Next, we imitated a Resource Broker and migrated the simulation to another site which was not overloaded (Amsterdam, Nikhef). The experiments were performed at night in order to avoid interference with other users.

In the presented situation it is better to spend some time on migration to another site, from where the response time is shorter. This is shown by the solid line (migration was done between responses 2 and 3). Fig. 7.6 shows that the human gains access time between responses 4 and 5 independent of the time lost on migration. The experiment was repeated 10 times – each time we got slightly different values of response time, but the significant result (a user gain between steps 4 and 5) remains the same.

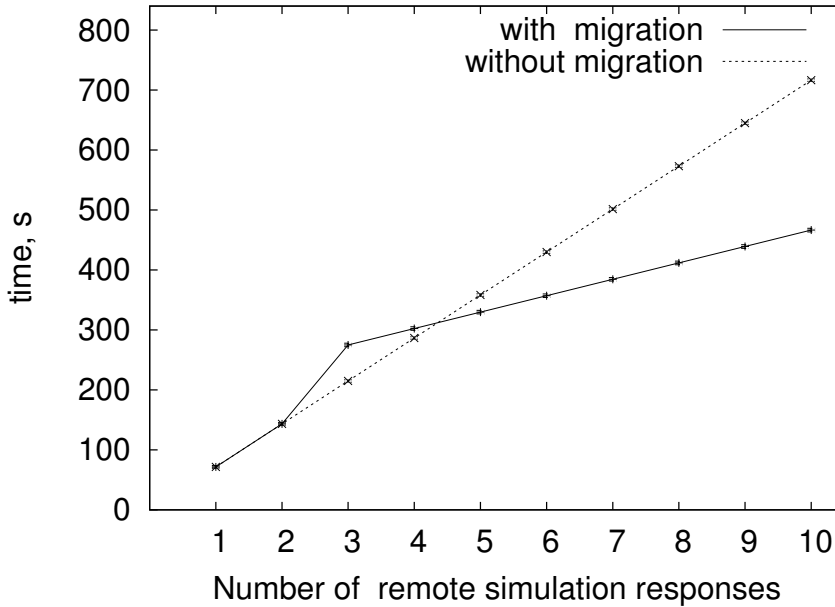


Figure 7.6: Impact of migration on response time

7.1.3 Conclusions

In this Section we have presented results from two experiments. In the first experiment, we show the impact of various HLA RTI operations that are performed during migration on overhead of that migration. In the second experiment we show how migration can improve the performance of an application with a human in a processing loop.

The results show that, in some circumstances, migration can improve the performance of the HLA-based application in the Grid environment and allows us to fulfill the requirements of HPC applications. However, there still remains the question of how to decide when and where to move – the results of *Benchmark Services* and *Monitoring Services* are discussed in the next Section.

7.2 Benchmark results

7.2.1 Experiment description

Our current *Benchmark Services* implementation consists of the *Benchmark_IVC Service* – check and scan operations, as well as the *Benchmark_SC Service* – start operation. The services are described in detail in Section 6.1.

Localization	check operation		actual sent time	
	\bar{x} [s]	σ [s]	\bar{x} [s]	σ [s]
Amsterdam(VU)–Leiden	700	10	660	10
Amsterdam(VU)–Amsterdam (UvA)	450	10	410	10
Amsterdam(VU)–Delft	290	10	250	8
Amsterdam(VU)–Utrecht	377	9	340	9
Total scan operation	1810	30		

Table 7.3: Benchmark results

Measurements were performed on the DutchGrid DAS2 [49] coupled with the Krakow ACC CYFRONET [4] infrastructure as shown in Tab. 7.3.

We set up the *Benchmark_IVC Service* at Vrije University (VU) in Amsterdam and performed the `scan` operation for connections between all other sites of DAS2. The RTIExec process was run in Krakow, at ACC CYFRONET. The first column presents the measured time of the **check** operation, where $t_{check} = t_{ogsasetup} + t_{send}$ as the operation consisted of setting up the federation in the OGSA framework as described in Section 6.1 and sending actual data (10 requests (small message) and 10 responses (400MB each)). Each response was in fact divided into 1MB chunks of data due to memory allocation issues (as described and explained in the previous Section – RTI requires four times more memory than the size of the actual sending/receiving buffer). In the next column we show the time of ten request-response operations made by pure HLA object updates. In the last row of the table we show the total duration of the **scan** operation which is a sum of **check** operations on all given sites: $t_{scan} = \sum_{allsites} t_{check}$. The measurements were taken outside working hours from 9-10 p.m., to avoid additional traffic. The results are an average value from 10 measurements and σ indicates the estimated standard deviation.

7.2.2 Conclusions

The experiment gives us an idea of what to expect from application performance in a simple interaction–simulation scenario and helps decide if it is good enough for running interactive applications (for some applications this can be satisfactory, for others – less so). The results also show that the benchmarks can provide support for decisions on where to place application components (the location of the interaction and visualization component is fixed due to human presence constraints, but the simulation component can be located – for instance – in Delft, according to benchmark results). Some tuning of benchmark parameters is necessary depending on whether they are performed on– or offline.

7.3 Monitoring overhead

7.3.1 Experiment description

Operating System	Red Hat Linux 7.2		
Network	10 Gbps (DAS2)		
Role in experiment	Name	CPU	RAM
Rtiexec	Nikhef	Pentium III 1 GHz	1 GB
OCM-G Main SM	VU	Pentium III 1 GHz	2 GB
Simulation federate	Leiden	Pentium III 1 GHz	2 GB
Interaction federate	Nikhef	Pentium III 1 GHz	1 GB

Table 7.4: Testbed infrastructure on Grid

As with previous tests, our HLA monitoring system was tested on the DAS2 [49] DutchGrid testbed infrastructure.

Two different configurations were used. First, we used only one cluster from the DAS2 located in Nikhef with Pentium III nodes (1 GB RAM each). The nodes were connected by multi-gigabit Myrinet. Next, we used the whole Grid infrastructure as shown in Tab. 7.4. For these configurations we chose DAS2 sites that were available and not overloaded at that time.

First of all, we checked that the functionality described in Section 6.2.2 is provided in the correct way; then we measured the overhead induced by the monitoring system. As an example, we used the application based on interactive applications requirements [155, 72], consisting of integrated interaction and visualization components that interface with a human who steers an MPI simulation component. Interaction components send small messages (commands from the user) to the simulation, which replies by sending large amounts of data containing new images of blood flow. Based on this scenario we implemented a simple application consisting of two HLA federates acting as an interaction federate and a simulation federate. Interaction is performed as follows: the interaction federate asks the simulation federate for a large amount of data (in our case, 100x8 MB packages and 100x80 MB packages) and measures the time between request and response. All programs were executed 10 times. σ indicates estimated standard deviation. The results of tests are shown in Tab. 7.5. Each time we monitored 5000 events of `RTIambassador::tick` function calls and 200 events of `RTIambassador::updateAttributeValue` function calls.

7.3.2 Conclusions

The result of our experiment shows that although the time of execution tends to grow slightly, OCM-G introduces very low overhead. This means that the proposed way of monitoring is useful for G-HLAM.

		no HLA-Monitor	HLA-Monitor compiled	
			no events subscriptions	all events subscriptions
Local cluster	\bar{x} [s]	452	465 (2.87%)	487 (7.74%)
	σ [s]	35	27	43
Grid	\bar{x} [s]	463	470 (1.51%)	503 (8.63%)
	σ [s]	29	32	49

Table 7.5: Overhead of OCM-G

7.4 Summary

This Section contains performance results of the G-HLAM system. We presented experiments showing the overhead of migration operations and its impact on application performance. Next we described results from the *Benchmark Services* and finally we measured the overhead of the *Application Monitoring Service*.

The results show that the most important services of G-HLAM are provide functionality described in previous Chapters. The overhead and complexity of services are reasonable, so the system is able to fulfill the requirements stated in Section 1.3:

- contains services supporting more efficient execution for HLA-based distributed interactive simulations on the Grid,
- provides an interface to legacy HLA applications,
- allows the use of advanced HLA features on the Grid,
- provides scalable solutions with relatively low complexity,
- does not introduce too much overhead.

In this Chapter we prove that the presented services of the G-HLAM system are behaving correctly and effectively and therefore can be used with real applications, as is described and validated in the next two Chapters.

Application of G-HLAM to N-body simulation

In this Chapter¹ we present a feasibility study for the G-HLAM system based on a sample real-world interactive simulation. Our proof-of-concept implementation of the G-HLAM system was tested with a parallel N-body simulation allowing it to benefit from the Grid using the solution proposed in this thesis.

When talking about HLA-based interactive distributed simulations that are two important aspects that should be taken into account. One is a type of *interaction* and the other is a type of *distribution*. In this Chapter, we especially focused on the application example illustrating *interactivity* issue. Therefore, we have described the supported types of interaction with the simulation: *passive* and *active*, exploration of partial simulation results with the possibility of rollback to previous simulation timesteps on user demand. The *passive* exploration fits the one-way interaction model shown in Fig. 1.2, while *active* fits the second interaction model as depicted in Fig. 1.3 in Chapter 1, where the user is not only able to see the progress of the simulation when receiving its output online, but can also can steer the running application by changing its parameters during runtime.

Using this example, we show how an interactive simulation can be run within the Grid HLA Management System (G-HLAM). We also present results from three different experiments showing how G-HLAM can improve the performance of a "human in the loop" application in both interaction models.

¹The results described in this Chapter formed the basis to the following papers:

K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Grid Service for Management of Multiple HLA Federate Processes. Accepted for *Parallel Processing and Applied Mathematics (PPAM) 2005 conference*, 11-14 Sep 2005, Poznań, Poland.

K. Rycerz, A. Tirado-Ramos, A. Gualandris, S. Portegies Zwart, M. Bubak, and P. M. A. Sloot. N-Body simulations on the Grid: HLA versus MPI. Submitted to *Journal of High Performance Computing Applications*.

Subsequently, the next Chapter focuses on the *distribution aspect* of HLA-based applications – we are showing this issue on the representative example of building collaborative environment for users with their visualisation devices located in different sites on the Grid.

8.1 Background

Modeling the detailed behavior of dense stellar systems remains a great challenge for astrophysics. Efficient simulations of galactic nuclei and suburbia, star-forming regions, globular or open star clusters remain interesting issues[131].

There are two main directions in which algorithms for solving astrophysical N -body problems have recently evolved.

On the one hand, approximated models have been developed, examples are Fokker-Planck models [118], gaseous models [101] and Monte Carlo models [89]. They have been applied to simulating of globular clusters and galactic nuclei. These models allow us to follow the global evolution of large systems along their lifetime but at the expense of moderate accuracy and resolution. On the other hand, direct summation methods have been developed to model the dynamics and evolution of collisional systems such as dense star clusters.

One of the features of dense stellar systems is that the stars are so close together that they will occasionally collide with one another and undergo frequent interesting and complex interactions. Therefore, simple integration of Newton's equations of motion for each star under the influence of the gravitational pairwise interactions of all other stars is not enough.

To give an example, even a small group of stars will spontaneously form one or more double stars (so-called binary stars). The changes in the state of such binary stars occur frequently, so they should be modeled by choosing a tiny time step. However, in such a scenario the rest of the system is forced to slow down, resulting in a waste of a significant amount of computer time on all the other stars that don't require such fine resolution. This is not a trivial problem.

Additionally, relaxation effects and close stellar encounters require highly accurate computations of a large amount of forces between particles which, at the present time, can only be achieved with direct integrators. Direct methods have $O(N^2)$ complexity, where N is the number of particles, since they compute the complete set of inter-particle forces at each step and are therefore limited to smaller particle numbers in comparison to approximated methods.

Simulating a globular cluster containing about one million stars is currently one of the most challenging numerical problems in astrophysics. There is a need for a computer infrastructure that will significantly improve the performance of direct methods. We think that the Grid, by providing the possibility of accessing computational resources that were previously inaccessible is a promising environment for such requirements. A performance analysis of different parallelization schemes for direct codes used in the simulation of astrophysical stellar systems for various computer

architectures including the Grid can be found in [72].

In this Chapter we use a sample N -body simulation - a modern integrator (the Hermite scheme [107]), using a variable time step integration scheme. The algorithm described in [82] chooses the time steps in such a way as to guarantee time symmetry in an integration scheme and therefore allows for energy conservation for orbital calculations with variable time steps, which is not an easy task.

This example is quite straightforward for the purpose of presenting a two-way interaction model, where the user can steer the running application by changing its parameters during runtime (e.g. he can examine the behavior of the system after a supernova explosion, which can be easily simulated by changing the mass of the selected star). Another advantage is that the user does not need to restart simulation with different parameters, but can research various systems while continuing with the same simulation.

8.2 Description of the N -body application

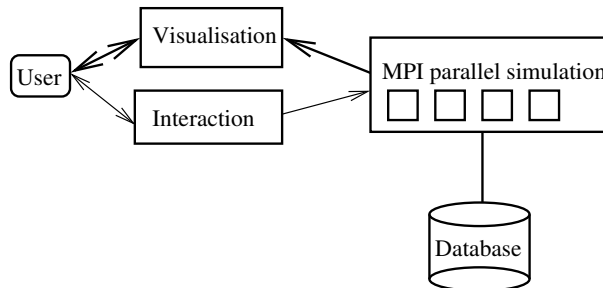


Figure 8.1: The N -body application architecture – The simulation processes data from a database and sends its output to the visualization (thick arrows). The user can interact with the simulation by changing its parameters during runtime (thin arrows).

The N -body simulation is part of the virtual laboratory called Gravitylab [81] which is a powerful software environment for experiments in stellar dynamics of dense stellar systems. The simplified architecture of the application is shown in Fig. 8.1. The simulation processes data from a database and sends its output to the visualization (thick arrows). The user can interact with the simulation by changing its parameters during runtime (thin arrows).

8.2.1 Simulation details

The algorithm of the simulation uses a direct method, where the gravitational force acting on a particle is computed by summing up the contributions from all other par-

ticles according to the Newton's law

$$\vec{F}_i = m_i * \vec{a}_i = -Gm_i \sum_{j=1, j \neq i}^{j=N} \frac{m_j(\vec{r}_i - \vec{r}_j)}{|\vec{r}_i - \vec{r}_j|^3}$$

The numerical method is based on the Hermite [107] scheme, where higher-order derivatives are explicitly computed in order to construct interpolation polynomials of the force.

Firstly, the initial snapshot of N stars is read and the particles are distributed between processes using block distribution. The snapshot consists of information about stars' masses and their positions and velocities in 3D.

Each simulation timestep is calculated as follows: first, the positions and velocities of all particles are predicted at time t (predictor phase) using the values of positions, velocities, accelerations and first derivatives of accelerations (jerks) computed at the previous step. The prediction uses a third-order Taylor expansion.

Next, by means of the predicted quantities, new values for accelerations and jerks at time t are computed. This calculation is the most computationally expensive one of the whole scheme, having N^2 scaling - the processes exchange data about particles using a one-dimensional ring structure. In this step potential energies and the collision times are also calculated. Collisions determine significant changes in local configurations, therefore the calculated collision time is used to calculate the next timestep. This causes the algorithm to have a variable timestep.

Finally, the second and third derivative of the accelerations are calculated using Hermite interpolation based on the values of acceleration and jerk. These correcting factors are added to the predicted positions and velocities (corrector phase) at time t . A detailed algorithm can be found in [82].

After the next output of the simulation is calculated the program checks if a request from the user to roll back to one of the previous timesteps with different star parameters has arrived. If so, the appropriate rollback is performed and the simulations continues from the point indicated by the user. If no rollback request arrives, the simulation proceeds to the next timestep.

8.2.2 Inter-module communication

The inter-module communication between application modules is shown in Fig. 8.2. The visualization/interaction module consists of a user interface and a communication interface that communicates with the simulation through the High Level Architecture Runtime Infrastructure. The user can invoke steering commands (pause updates of simulation data, change simulation parameters and resume updates of simulation data). The simulation produces output at each timestep and sends it back to the visualisation/interaction module (put snapshot). Subsequently, the communication interface sends this data to the user interface so the user can see the output and react to it according to his needs.

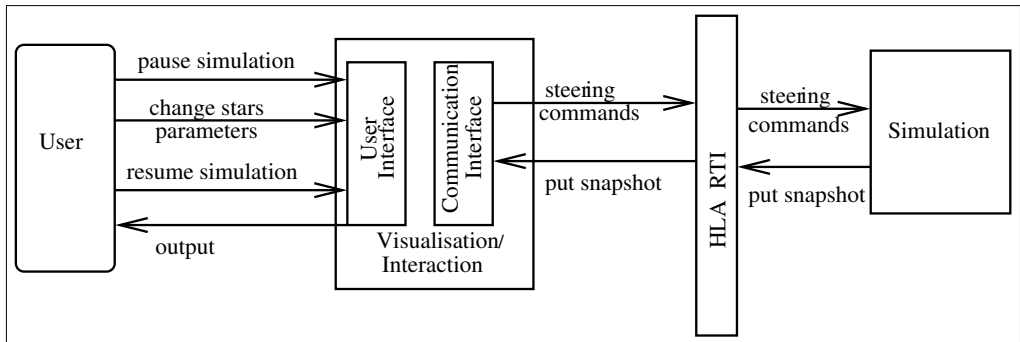


Figure 8.2: Communication between the user, the simulation and the visualization: at each timestep the simulation sends output (snapshot) to the visualisation/interaction module, which is then forwarded to the user interface. The user can react according to his needs by invoking steering commands (pause, change parameters, resume).

8.2.3 Types of interactions with the user in a loop

The output of the simulation is sent online to the visualization module while simulation is progressing, using High Level Architecture (HLA) Runtime Infrastructure [77]. In order to explore results at a chosen point of simulation time, the user can pause the process of receiving simulation output without pausing the whole simulation. In such a case, the visualization enters an *active* mode. This is illustrated in Fig. 8.3, where a simulation produces output in time steps (numbered boxes in the figure). The diagram notation was adopted from [16]. This output is then visualized in *passive* or *active* mode according to the user's needs (in the figure the user has changed mode from *passive* to *active* while watching results from step 2 of the simulation – it is worth noticing that the simulation is already calculating results for step 4). If the user decides to make no changes to the situation in step 2, he can resume the process of receiving output (calculated in the meantime by the simulation). In the presented figure this is done in the middle of calculating step 6 and the user can examine results from that step once they are finished.

A different situation is presented in Fig. 8.4. The user decides to change the simulation at timestep 2 by changing position, mass or velocity of the chosen stars, and informs the simulation about this requirement when it is already calculating step 5. The simulation then rolls back to step 2 and initiates new calculations from that point taking into account the changes introduced by the user (shown as 2') in Fig 8.4.

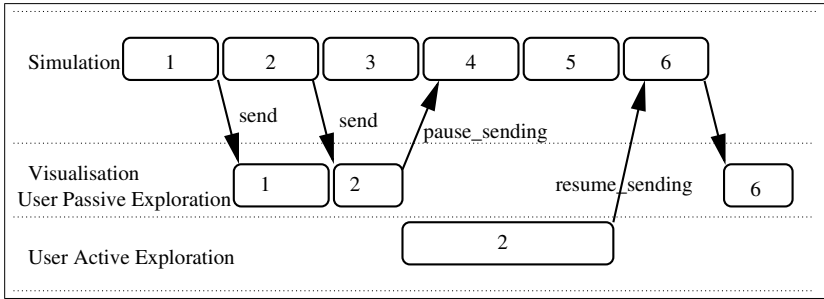


Figure 8.3: Interaction without changes to the simulation – the user pauses and resumes sending output, but does not make any changes to simulation state.

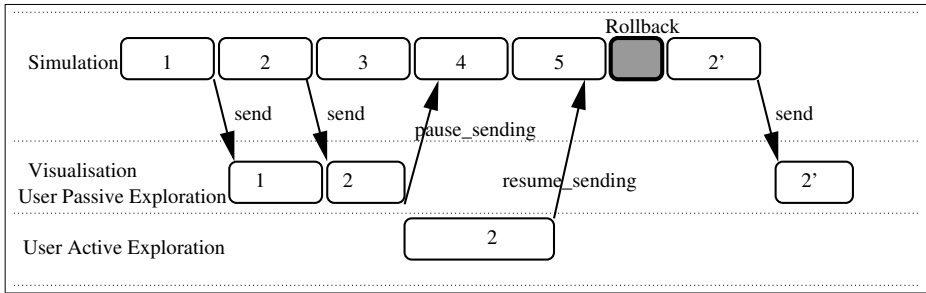


Figure 8.4: Interaction with changes to the simulation – the user makes changes to simulation state, so the rollback of the simulation that managed to advance in the background is needed.

8.3 N-body simulation in G-HLAM

8.3.1 Detailed structure

The application consists of two modules: An MPI N-body simulation module and an integrated visualization–interaction module. Communication between application components is shown in Fig. 8.5. The modules communicate using the HLA RTI bus which connects the visualization/interaction federate with the MPI root process of the simulation federate. Together, those federates form the application federation. Additionally, HLA is used for control of MPI simulation processes on the Grid site as described in Section 4.3. Therefore, all MPI processes join the control federation together with the control federate in the HLA–speaking service. For connecting to the control federate all processes use the GridHLAController library as described in Section 5.3.

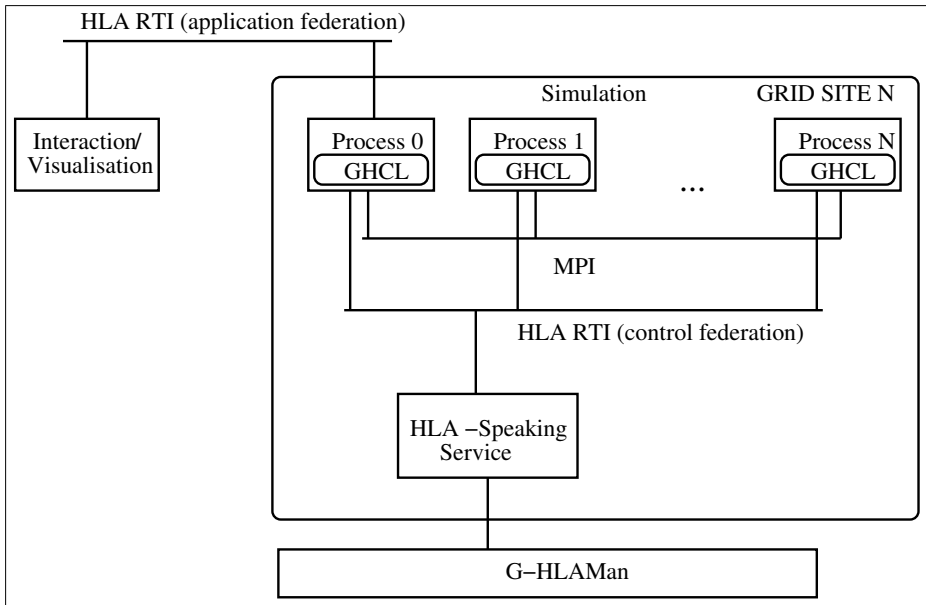


Figure 8.5: Application with G-HLAM – visualization/interaction federate communicates with the MPI root process of the simulation federate using the HLA RTI. Additionally, HLA control federation is used to interface MPI simulation processes on the Grid site to the *HLA-Speaking Service* and G-HLAM.

8.3.2 Results

In Chapter 7 we have presented performance results of the simple test applications. In this Chapter we prove that G-HLAM can be also successfully used with the realistic HLA-based interactive *N*-body simulation that can be actively explored by a user during runtime. We have conducted four different experiments to show how migration can help achieve better performance of the simulation. The experiments were performed on the DutchGrid DAS2 testbed infrastructure and at CYFRONET, Krakow, as shown in Tab. 8.1.

Operating System	Red Hat Enterprise Linux Advanced Server, version 3		
Network	10 Gbps (DAS2) + 155 Mbps (DAS2-Cyfronet)		
Role	Name	CPU	RAM
Migration source	DAS2 UvA, Amsterdam	Pentium III 1 GHz	1 GB
Migration dest	DAS2 Delft	Pentium III 2GHz	2 GB
Interaction federate	Cyfronet Krakow	2.40 GHz Xeon	1 GB
RTIExec	Cyfronet Krakow	2.40 GHz Xeon	1 GB

Table 8.1: Grid testbed infrastructure.

Migration time

According to the scenario presented in Fig. 5.2 in Section 5.2.2, migration time can be described by the following formula:

$$t_{migration} = t_{save} + t_{resignapp} + t_{resigncontrol} + t_{transfer} + t_{joincontrol} + t_{joinapplication} + t_{restore} \quad (8.1)$$

where

- $t_{migration}$ is the total migration time
- t_{save} is the time of saving application federation state
- $t_{resignapp}$ is the time of resigning from the application federation
- $t_{resigncontrol}$ is the time of resigning from the control federation
- $t_{transfer}$ is the time of transferring checkpoints
- $t_{joincontrol}$ is the time of joining the control federation
- $t_{joinapplication}$ is the time of joining the application federation
- $t_{restore}$ is the time of restoring the application federation's state

In our application the time of resigning and joining the control federation depends on the number of all processes in the MPI simulation, since every process contains one control federate. Below we present results of scalability measurements for those durations.

Afterwards, we also present the results from measurements of the time of joining, saving, resigning, restoring the application federation – these durations do not depend on the number of MPI processes, since only the master process contains an application federate as illustrated in Fig. 8.5.

Resigning from the control federation depends on the total number of federates already in the federation. Below, we approximate the complexity of resigning time. Assuming that the federation contains N federates, the time of resigning of the i -th federate can be approximated by the following formula:

$$t_{resign}^i = \sum_{j=1}^i [(N - j) * t_{resignOtherFed} + t_{resignRTIproc}] \quad (8.2)$$

where t_{resign}^n is the time of resigning the federation by the n -th process, $t_{resignOtherFed}$ is the time of closing connection between two federates which are in the same federation and $t_{resignRTIproc}$ is a time of disconnecting federate from RTI control processes (RTIExec and FedExec).

Using the formula for the sum of the first n terms of an arithmetic sequence:

$$t_{resign}^i = i * [N * t_{resignOtherFed} + t_{resignRTIproc}] - \frac{(i + 1) * i}{2} * t_{resignOtherFed} \quad (8.3)$$

and:

$$t_{resign}^i = [(N - \frac{1}{2}) * t_{resignOtherFed} + t_{resignRTIproc}] * i - \frac{1}{2} * t_{resignOtherFed} * i^2 \quad (8.4)$$

Resigning time also includes a constant term C depending on the RTIExec and FedExec federation management algorithm.

$$t_{resign}^i = -\frac{1}{2} * t_{resignOtherFed} * i^2 + [(N - \frac{1}{2}) * t_{resignOtherFed} + t_{resignRTIproc}] * i + C \quad (8.5)$$

from the formula above, we suspect that t_{resign} is $O(N^2)$ and therefore, we are looking for a quadratic function $Ai^2 + Bi + C$.

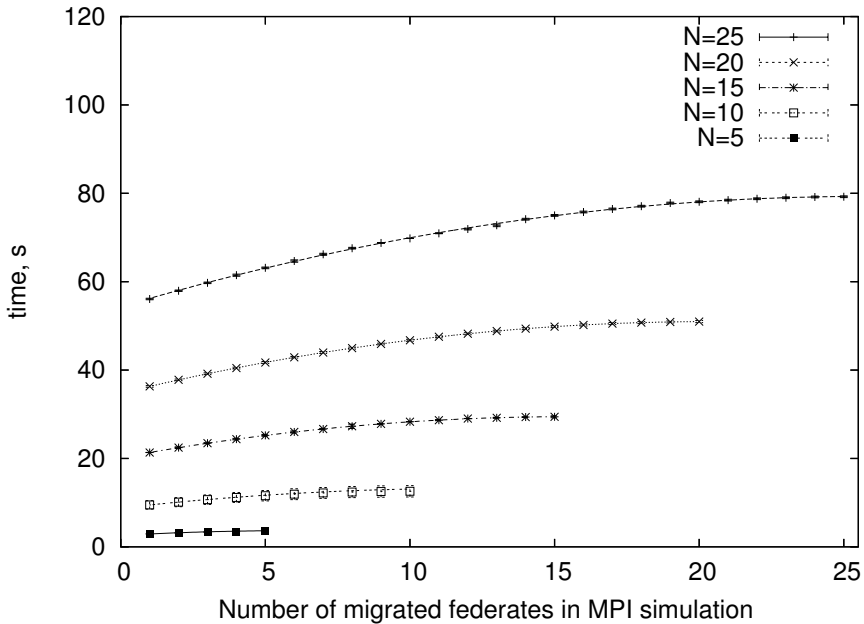


Figure 8.6: Overhead of resigning from the control federation

Fig. 8.6 shows resigning time as a function of the number of MPI processes in the federation. For our tests we used GT v3.2 and HLA RTI 1.3v5. The number of stars used in the simulation was 5000. Results were obtained as an average from 10 measurements. The error bars indicate estimated standard deviation. The figure shows also quadratic functions obtained from the least squares fitting that approximate our results. The obtained constants for various N can be found in Tab. 8.2. The calculated approximation has physical sense for the $i=1..N$.

As can be observed, the coefficients B and C depend on federation size N. Formula 8.5 explains behavior of coefficient B. The growth of initial value C for various N can be explained by the fact that in our experiment all federates are trying to resign at

Federation size	A	σ_A	B	σ_B	C	σ_C
5	-0.03	0.005	0.4	0.06	2.5	0.09
10	-0.03	0.004	0.7	0.05	8	1
15	-0.03	0.005	1.2	0.05	10.2	0.02
20	-0.03	0.001	1.6	0.05	35	0.2
25	-0.03	0.001	2.0	0.02	53.3	0.09

Table 8.2: Coefficients of approximating function $Ax^2 + Bx + C$ of resigning time for various initial sizes of the federation

the same time. Federation management algorithm for dealing with multiple resigning requests implemented within RTIExec and FedExec processes can result in relatively big initial value dependent on the total number of federates that want to resign.

The total resigning time for N federates is therefore given by the following equation (from 8.5):

$$t_{resign}^N = -\frac{1}{2} * t_{resignOtherFed} * N^2 + [(N - \frac{1}{2}) * t_{resignOtherFed} + t_{resignRTIproc}] * N + C(N) \quad (8.6)$$

$$t_{resign}^N = \frac{1}{2} * t_{resignOtherFed} * N^2 + (-\frac{1}{2} * t_{resignOtherFed} + t_{resignRTIproc}) * N + C(N) \quad (8.7)$$

From equation 8.7 we suspect that t_{resign}^N is at least $O(N^2)$. The actual complexity depends on $C(N)$, which, as described above, depends on internal algorithm of HLA Federation management. Fig. 8.7 shows experimental data and approximation function $AN^2 + BN + C$, illustrating the total resigning time depending on N , where N is the size of the federation. From our calculations $A = 0.12$, $\sigma_A = 0.01$, $B = -0.2$, $\sigma_B = 0.05$, $C = 1.0$, and $\sigma_C = 0.1$. As described above, the complexity $O(N^2)$ depends on the used HLA implementation and could be reduced if implementation with more optimal federation management was used. Unfortunately, to our best knowledge, no such implementation is available for free academic use.

Joining the control federation depends on the number of federates that have already joined. Below we approximate the complexity of the joining time. Assuming that the n -th process containing a federate which wants to connect to the federation has to open TCP connections to all existing $n - 1$ processes and connect to RTIExec and FedExec control processes, the joining time for that process can be approximated by the following formula:

$$t_{join}^n = (n - 1) * t_{joinOtherFed} + t_{joinRTIproc} \quad (8.8)$$

where t_{join}^n is the time of joining the federation by the n -th process, $t_{joinOtherFed}$ is the time of opening connection between two federates which are going to be in the same federation and $t_{joinRTIproc}$ is a time of connecting federate to RTI control processes (RTIExec and FedExec). If, in turn, there are N processes which need to join the federation and there are already M processes that have previously joined, then the

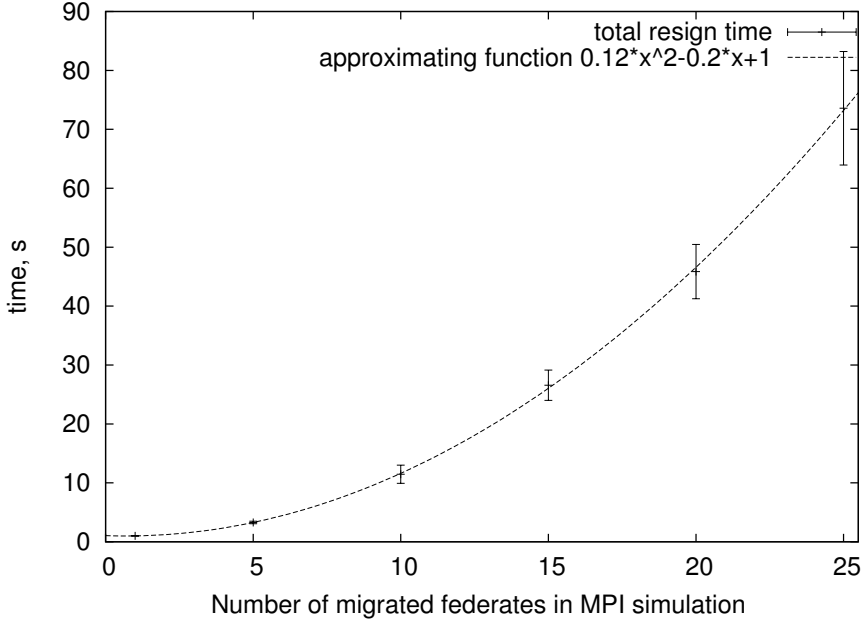


Figure 8.7: Resigning time as a function of the size of the federation

joining time for $process_i$ from $i = 1 \dots N$ is the sum of the joining time for all $i - 1$ processes that have joined before plus its own joining time:

$$t_{join}^i = \sum_{j=1}^i [(M + j - 1) * t_{joinOtherFed} + t_{joinRTIproc}] \quad (8.9)$$

and using the formula for the sum of the first n terms of an arithmetic sequence:

$$t_{join}^i = i * [(M - 1) * t_{joinOtherFed} + t_{joinRTIproc}] + \frac{(1 + i) * i}{2} * t_{joinOtherFed} \quad (8.10)$$

$$t_{join}^i = [(M - \frac{1}{2}) * t_{joinOtherFed} + t_{joinRTIproc}] * i + \frac{1}{2} * t_{joinOtherFed} * i^2 \quad (8.11)$$

Additionally there is always a constant term C in the joining time equation depending on the RTIExec and FedExec federation management algorithm.

$$t_{join}^i = \frac{1}{2} * t_{joinOtherFed} * i^2 + [(M - \frac{1}{2}) * t_{joinOtherFed} + t_{joinRTIproc}] * i + C \quad (8.12)$$

From the formula above we suspect that t_{join} is $O(N^2)$. To study this, we have performed experiments to check if the joining time is indeed a quadratic function of the type $Ai^2 + Bi + C$. Fig. 8.8 shows joining time as a function of number of MPI processes

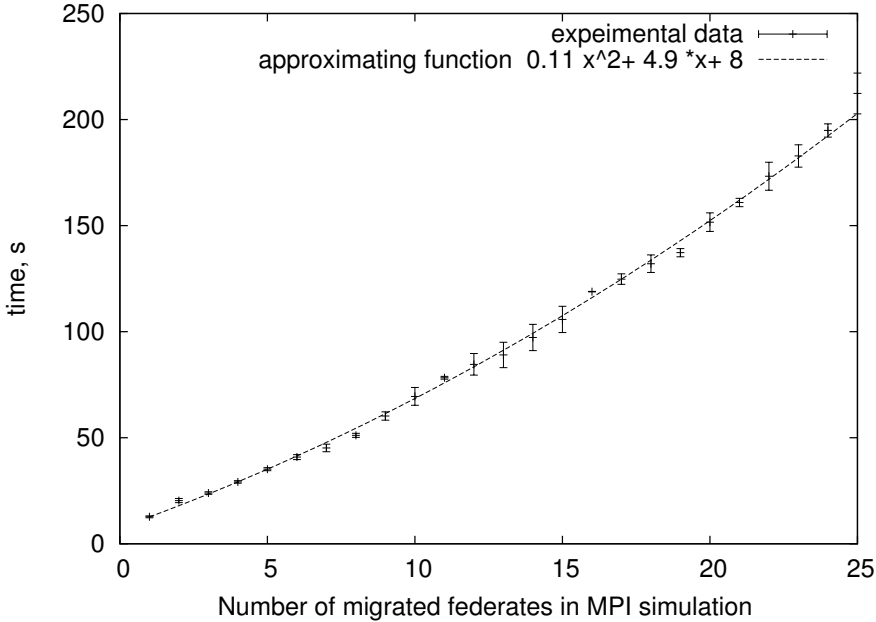


Figure 8.8: Overhead of joining control federation

	\bar{x} [s]	σ [s]
saving time	6	1
application resigning time	4	0.5
transfer time	0.4	0.05
GRAM	2.5	0.1
PBS waiting time	8	0.7
application joining time	13	2
restoring time	8	0.8

Table 8.3: Execution time of various migration stages independent of the number of MPI processes

in the federation. The conditions of the test were the same as above (5000 stars, using GT v3.2 and HLA RTI 1.3v5). The results were obtained as an average from 10 measurements. The error bars indicate estimated standard deviation. We have used least-square fitting to approximate our results, obtaining the following constants: $A = 0.11$, $\sigma_A = 0.01$ $B = 4.9$, $\sigma_B = 0.08$ and $C = 8$, $\sigma_C = 1$.

Time of other activities

Tab. 8.3 shows the activities independent on the number of MPI processes performed during migration. They include saving and restoring the state of the application federation, joining and resigning the application federation, transferring files

(GridFTP overhead) as well as GRAM and PBS overhead. They are independent of the number of MPI federates since only one root MPI process actually joins the application federation in order to communicate with the visualization federate.

The conditions of the test were the same as above (5000 stars, using GT v3.2 and HLA RTI 1.3v5). The results were obtained as an average from 10 measurements. The error bars indicate estimated standard deviation.

Total migration overhead Total migration overhead can be calculated from formula 8.1 and the values derived above. Therefore:

$$t_{migration} = 0.23 * x^2 + 4.7 * x + 50.9 \quad (8.13)$$

On the other hand, values A , B and C can be calculated directly from experimental data. From our experiment $A = 0.2$, $\sigma_A = 0.07$, $B = 4$, $\sigma_B = 0.8$ and $C = 51$, $\sigma_C = 0.6$. These values are equal in the range of error bars.

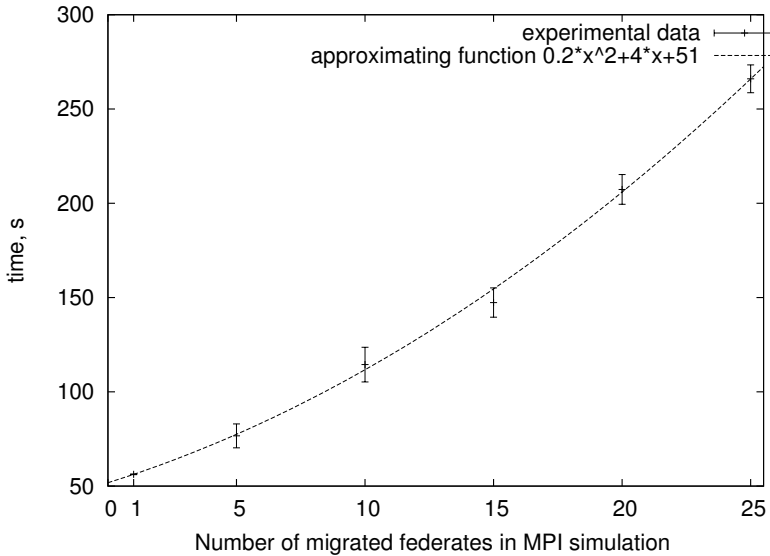


Figure 8.9: Total migration time as a function of the number of migrated MPI federates

As can be observed, the largest overhead in the case of the tested application is for rejoining the control federation. This is due to the fact that in the used RTI implementation, each federate has to open TCP connections to every other federate in the control federation. Reliable multicasting [112] would significantly decrease this overhead.

Passive exploration

As in the previous experiment, for our tests we used GT v3.2, HLA RTI 1.3v5 and

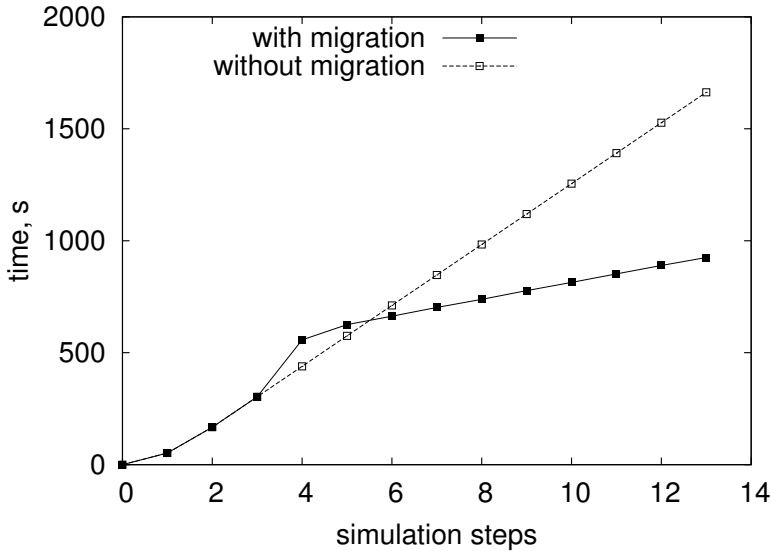


Figure 8.10: Migration impact on execution time for a scenario of *passive* interaction with the *N*-body simulation – migration during passive examination

the testbed infrastructure depicted in Tab. 8.1. The simulation consisted of 20 MPI processes calculating 3D positions and velocities of 24000 stars.

Because the bandwidth available for testing was broad (10 Gbps), communication did not play an important role and calculations were the most time-consuming part of the execution. In order to create conditions in which migration would be useful, we increased the load of the Grid site where the simulation was executed (Amsterdam) by submitting nonrelated, computationally-intensive jobs. Next, we imitated a Resource Broker and migrated the simulation to another site which was not overloaded (Delft). The experiments were performed at night in order to avoid interference with other users.

Fig. 8.10 shows the time as a function of the number of interactive steps with a human in the loop (for the first 13 steps). In each step, the simulation calculates data and sends it to the visualization and interaction modules using HLA. The dashed line shows the execution time of the simulation steps in the case when the simulation was not migrated.

In this situation it is better to spend some time on migration to another site, from where the time of response is shorter as shown by the solid line in the figure. Fig. 8.10 shows that the human gains access time between steps 5 and 6 independent of the time lost on migration (performed between steps 3 and 4). The experiment was repeated 10 times – each time we got slightly different values of response time, but the significant result (a user gain between steps 5 and 6) remains the same.

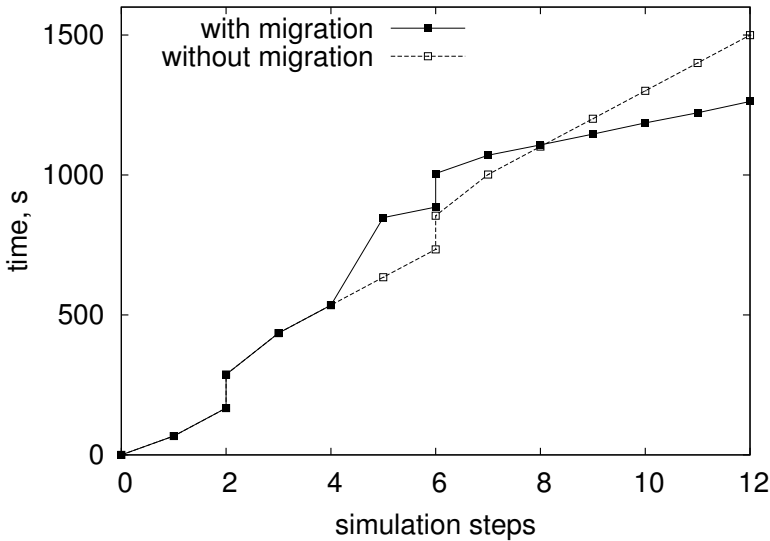
Active exploration

Figure 8.11: Migration impact on execution time for the scenario of *active* interaction with the *N*-body simulation.

In this experiment, we show what happens when a user actively explores simulation output (the parameters and conditions were the same as in the previous case – 20 MPI processes, 24000 stars, the same software versions and testbed infrastructure). The solid line in Fig. 8.11 shows a typical scenario - in step 2 the user explores the simulation output and a rollback is performed between steps 2 and 3. The time of rollback and simulation response is deemed unsatisfactory, therefore migration is performed between steps 4 and 5. Afterwards, the time of obtaining answers from simulation as well as performing rollback become acceptable from the user’s perspective. The dashed line shows what happens under the same conditions when migration is not performed. We can see that the user achieves a net gain at step 8.

The Fig. 8.12 shows what happens if migration is performed **while** the user is actively examining and changing simulation output at a given time. Since the user has temporarily ceased receiving new output from the simulation, he may even fail to notice the migration overhead, unless he explores the visualization output for a sufficiently long time. In Fig. 8.12 the user explores the simulation output at step 2, while migration is performed in the background. Between steps 2 and 3 a rollback is performed at a new place, hence the user achieves an immediate gain.

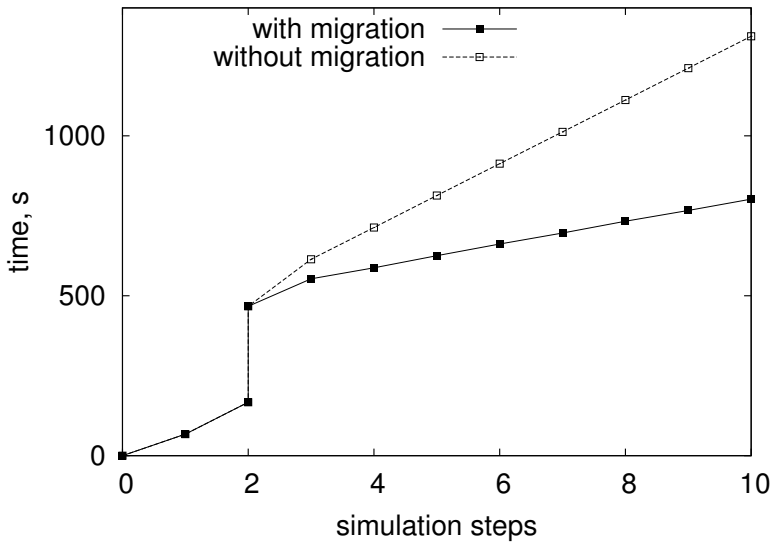


Figure 8.12: Migration impact on execution time for the scenario of *active* interaction with the N -body simulation – migration during active examination.

8.4 Summary and conclusions

In this Chapter we presented how an example interactive simulation can be run within the Grid HLA Management System (G-HLAM). The simulation calculates the N -body problem for a dense stellar system. The user can monitor the output of the simulation online (*passive* exploration) as well as interact with it by changing its parameters during runtime (*active* exploration). Additionally, the simulation has the ability to roll back to a previous timestep on user request.

Apart from a functional description, we presented and described performance results of the G-HLAM system used for this sample simulation. First, we measured how various migration phases impact migration time. Subsequently, results from three different experiments were shown to illustrate how G-HLAM can improve the performance of a "human in the loop" simulation, decreasing the response time of simulation federates by migrating them to a better location.

From our experiments we can draw the following conclusions:

- the G-HLAM system can be used to improve execution of an interactive distributed application running on the Grid by performing migration of its simulation module to a different Grid site,
- during migration the most significant overhead is introduced by rejoining the federation. This issue, however, is specific to the HLA implementation used and

can be reduced by introducing a reliable multicast mechanism [112],

- there are scenarios in which migration improves application execution despite of the overhead it produces. This occurs when the number of simulation steps performed is greater than the ratio of migration time to the difference between the execution time for an average step before and after migration (see equation 7.11),
- *active* exploration mode can be used to conceal migration overhead. When a user interacts with the application in that mode, no data is sent from simulation to visualisation, so the user does not view simulation progress online. Therefore migration can be performed without interfering with the user. When the user exits the *active* exploration mode, the response time from the simulation has already improved.

Application of G-HLAM to Vascular Reconstruction

In this Chapter¹ we show the advantages of using the Grid HLA Management System (G-HLAM) for running a large-scale HLA-based medical application supporting surgical pre-operative planning [1, 155].

While the previous Chapter focused on *interactivity* aspect of HLA-based applications, this Chapter describes the *distribution* aspect of such applications – we are showing this issue on the example of an simulation environment for users with the visualisation devices located in different sites on the Grid. The described application is representative for showing how G-HLAM can be used for *distributed* HLA-based applications. In particular, it will be shown how G-HLAM can be used to steer such an application in terms of performance. The application can be executed in three scenarios: one simulation for a single user, many simulations for a single user and a collaborative environment for many users. Our goal is to verify G-HLAM features and to show that the system can successfully run the complicated application with different scenarios including a collaborative environment, where geographically distributed users interact in a processing loop.

The interactions in the application fit the first interaction model depicted in Fig. 1.2 in Chapter 1. The user can see the output of the simulation while it is running and stop it any time when partial results seem unsatisfactory.

In Section 9.1 we describe in detail the application, its modules and three scenarios of execution. In Section 9.2 we show how to apply G-HLAM for the modules that

¹The results described in this Chapter formed the basis to the paper K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. HLA Grid Based Support for Simulation of Vascular Reconstruction. In S. Gorlatch and M. Danelutt, editors, *Proceedings of the CoreGRID Workshop "Integrated Research in Grid Computing, November 28-30, 2005*, pages 165–174, Pisa, 2005. Technical Report TR-05-22.

use HLA as their communication infrastructure. Section 9.3 presents performance results. The summary and conclusions can be found in Section 9.4.

9.1 Vascular reconstruction application

9.1.1 Background

Computer simulations in medicine are becoming increasingly important area, especially in surgery [192]. Planning vascular operations is difficult task – before the operation the physician has to locate the affected vessels, analyze them and then make decision how to operate. Therefore, simulation and verification of the surgeon’s plans prior to the actual operation is very useful and allows for better treatment of patients [155]. Usually such simulations are very complex, require high–performance execution and near–real–time interaction with users during runtime [129].

Vascular disorders such as stenosis (narrowing of the artery by the accumulation of fat and cholesterol) and aneurysms (weakness of artery wall resulting in a ballooning effect) seriously influence blood flow and can cause serious diseases [18]. Therefore, it is important to improve the flow quality in the affected vessels. Vascular reconstruction is a surgical procedure which redirects blood flow from the affected place using a grafted bridge called a bypass.

Planning such operations requires the surgeon to analyze the structure of the artery and locate the affected areas together with the optimal place in which to insert a bypass. By using computer simulations, surgeons’ decisions can be verified before the actual operation takes place [114]. In this Section we present a virtual reality–based medical training application [1] developed at the Department of Computational Science of the University of Amsterdam. We will describe application modules and outline three scenarios of execution: a scenario when a single user runs only one simulation, an extension for many simulations and, finally, a collaborative environment.

9.1.2 Application modules

Input data for the application are obtained from various medical imaging techniques such as X-ray angiography, computer tomography (CT) or MRI (magnetic resonance imaging). After getting digital images of the patient’s vessels, the data is processed by four modules of the application as described below.

Analysis and segmentation module

The goal of the segmentation process is to automatically find the lumen border between the blood and non-blood in input images [1]. Firstly, a wave front propagator algorithm is used to find an approximation of the centerline of the vessel. Next, the data are divided into a set of 2D slices orthogonal to the centerline. Then, in each slice a contour delineating the lumen border is detected. Finally, the stack of 2D contours is combined to 3D surface model, which is then passed to 3D editing tool.

3D editing tool

This tool allows for editing stereoscopic images and executing experimental visualisation studies on realistic arterial geometries [14]. A user can conduct measurements, pick up a position on the artery for creating a bypass and modify its shape and size. The final stage is generation of a computational mesh. The pre-prepared arterial geometry, including aneurysms, bifurcations, bypasses and stents is converted into a coarse or fine (depending on the user's choices) computational mesh that is then passed to the simulation module.

Simulation module

Blood flow simulation is performed with a parallel flow solver [10] which computes pressure, velocities, and shear stresses. The simulator is based on the Lattice-Boltzmann method (LBM) – a mesoscopic approach for simulating fluid flow based on the kinetic Boltzmann equation. The simulation produces intermediate results and sends them to the visualisation module. The module is parallelized using MPI.

Visualization and exploration module

The visualization/exploration module uses a Virtual Reality Explorer (VRE) [14], where the patient's data is visualized as a 3D stereoscopic image together with a graphical interpretation of the simulation's results. The user can then manipulate the 3D images of arteries, patient's body and blood flow structures in virtual reality. VRE combines natural input modes of context sensitive interaction by voice, hand gestures and direct manipulation of virtual 3D objects. The interactive measurement component of the VRE provides the possibility to quantitatively measure distances, angles, diameters and other parameters characterizing 3D objects in a virtual world, where markers are building blocks of distance, angle, and linestrip measurements. VRE architecture is described in more detail in Section 2.1.12.

Need for the Grid

The application modules require different resources: the segmentation tool requires quick access to an image database, flow simulation requires computational power, editing and visualization tools require specific VR hardware. It is quite unlikely to find those resources at any one geographical site. Additionally, if more simulations need to be run concurrently, one site with computational power may be not sufficient. A similar problem arises when many visualisations (users) located in different places want to observe the same simulation. Therefore, the application modules usually have to be located in geographically-different places and the Grid concept which facilitates access to computing resources may offer a very promising approach here.

9.1.3 Scenario with a single user and a single simulation

In Fig. 9.1 we show a scenario with a single user running one simulation [192]. It

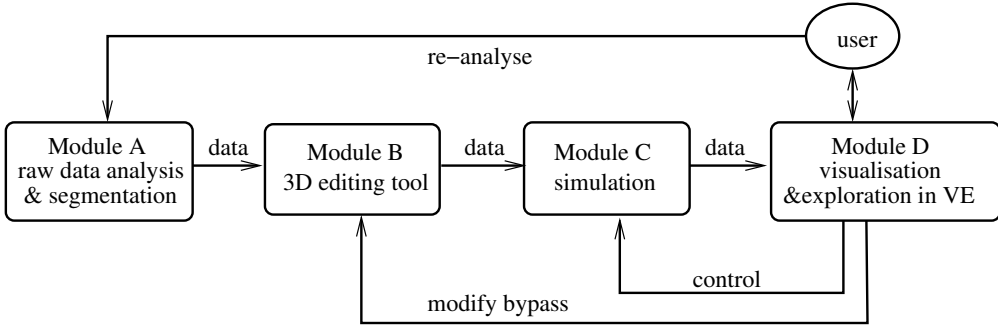


Figure 9.1: Vascular reconstruction application scenario with a single user and a single simulation consisting of four modules: data analysis, editing tool, simulation and visualisation.

shows how the modules described above cooperate with one another. As described, the application consists of the following modules:

- A) tool for analyzing and performing segmentation,
- B) tool for 3D editing and mesh generation,
- C) flow simulation,
- D) online visualization and exploration of results in VE.

The input data for the application is received from X-ray angiography, computer tomography (CT) or MRI (magnetic resonance imaging). Next, a radiologist uses module A to analyze raw digital images of vessel structures. In the first step, the images of the affected vessels are segmented in order to obtain a 3D geometrical description of the arteries of interest. Next, the segmented artery is prepared for blood flow simulations by module B, a 3D editing tool, allowing to define inlets and outlets, to filter and crop parts of the artery, to add a by-pass, and to generate computational meshes as input to the blood flow simulators. Subsequently, by using module C – a dedicated fluid flow solver – the time-dependent blood flow in the artery is computed. The resulting flow, pressure and shear stress fields are then shown to the user using a number of visualization techniques in a Virtual Environment (VE) – module D.

While watching simulation output, the user can control simulation execution (stop / pause / resume). The user also can modify the segmented artery and run another simulation. Additionally, there is the possibility of performing analysis of raw data and segmentation of another part of the vessel.

To summarize, the user can:

- stop/pause the simulation during runtime,

- restart with a different bypass,
- reanalyze raw data and reset segmentation.

The interactions described here fit the first interaction model depicted in Fig. 1.2 in Chapter 1. The user can see the output of the simulation while it is running and stop it any time when partial results seem unsatisfactory.

9.1.4 Scenario with a single user and multiple simulations

In Fig. 9.2. we show a scenario where the user can run more than one simulation.

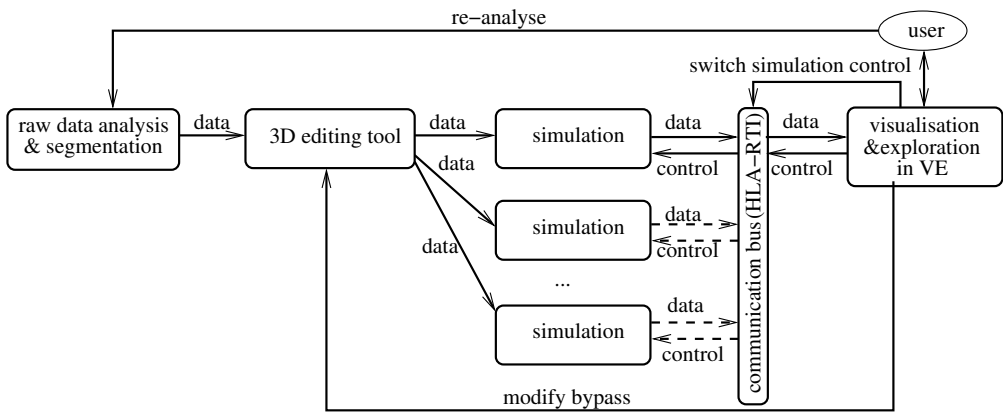


Figure 9.2: Vascular reconstruction application scenario with a single user and multiple simulations – the user uses a 3D editing tool to create different meshes with different bypasses and run the simulation simultaneously.

In this case, the user uses a 3D editing tool to create different meshes with different bypasses and run the simulation simultaneously.

In order not to overload the network, the user can see only one simulation output and control its execution - we thus say that this simulation is running in the foreground. Other simulations run in the background – the user cannot see their output or control their execution unless he decides to swap one of them with the foreground simulation (which is similar to background and foreground processes in Unix systems - a user can interact only with one process at a time, but is able to move background processes into the foreground and vice versa). In Fig. 9.2 the simulation with solid data and control arrows is running in the foreground while other simulations (with dashed arrows) are executed in the background.

If the user decides to control and see the output from another simulation, he has to switch simulations. Following this, the foreground simulation becomes a background simulation and the selected background simulation moves to the foreground. This situation is more general than in Fig. 9.1, where the only simulation available is run in the foreground.

The user can:

- stop/pause the chosen simulation during runtime,
- switch between different simulations (foreground/background),
- restart with different bypass,
- reanalyze raw data and reset segmentation.

9.1.5 Collaborative environment scenario

The most complicated scenario of this medical application usage is when multiple users want to share simulations between them. This kind of situation takes place when a group of surgeons from various hospitals wants to discuss interesting medical cases together and exchange ideas about planning difficult operations. In Fig. 9.3 we show the situation for two users, but this can be easily extended to more. Each user

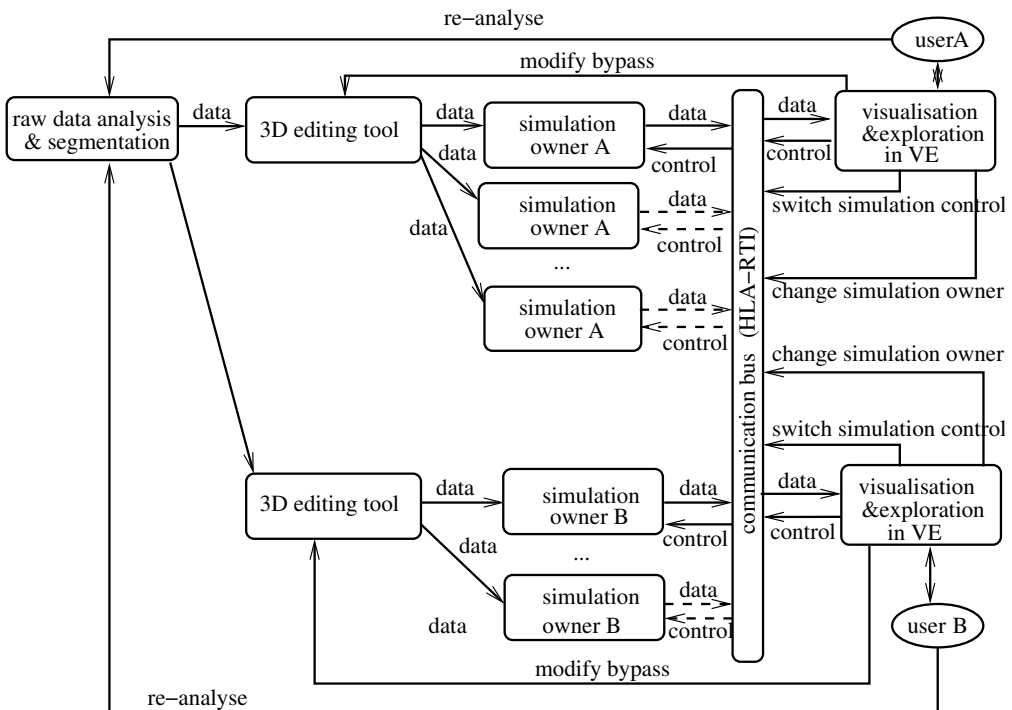


Figure 9.3: Vascular reconstruction application scenario with multiple user and multiple simulations. Each user starts a number of simulations and can own (control) each of them. The user can also interact with simulations he does not own.

starts a number of simulations as in the previous case and becomes their owner - that

means he can fully control each simulation. However, the user can also interact with simulations he does not own, in two ways:

- by requesting the output data of a non-owned simulation – in this case concurrent data access is not a problem, since only one user can actually change the data.
- by requesting to control a non-owned simulation - concurrent data access has to be controlled by a communications bus.

The user can:

- stop/pause the chosen simulation during runtime,
- switch between different simulations owned by him (receiving output),
- switch ownership of simulations (ability to stop/pause a particular simulation),
- restart with a different bypass,
- reanalyze raw data and reset segmentation.

9.1.6 HLA-based communication

For using G-HLAM, the most interesting parts of the application are the simulation module, the visualisation module and HLA-based communication in between. HLA is a very convenient solution for that purpose [192] as it allows for :

- building a geographically-distributed system - as stated in Section 9.1.2, usually there is a need to place the simulation in a different place than visualization, especially if there is more than one of each, as in Fig. 9.3,
- dynamic joining of simulation/visualization with the application – useful when user A in Fig. 9.3 needs to start simulations one by one, or user B joins late and starts his own simulations, but wants to share results with others,
- easy switching between background (indicated with dashed arrows in Fig. 9.3) and foreground (indicated with solid arrows) of simulations - useful when the user decides to obtain output data from a different simulation; by using HLA he can unsubscribe from data currently being received and subscribe to data of the new simulation – *Data Distribution Management* services of HLA are very convenient for that purpose,
- dynamic resigning of simulation/visualization from the application – the user can quit the application or stop some simulations without disturbing others,
- notification - useful for notifying the simulation about user commands i.e. pause, cancellation,

- easy switching between ownership - useful when the user who created a simulation wants to transfer control over it to a different user,
- concurrency control – related to ownership – assures that only one user can control the simulation at any time.

According to scenarios presented above and types of interaction within the application, these HLA features are very useful and fill the applications' requirements for a communication infrastructure between simulation and visualization.

9.2 Using G-HLAM

For the purposes of this case study, we have chosen two modules of the described application: simulation and visualisation/exploration. For describing G-HLAM usage, we use a collaborative environment scenario, since other scenarios are simply specific cases of that scenario. The collaborative environment includes many federates of both simulation types and visualization types connected using HLA. Therefore, the collaborative environment scenario is the most interesting case from our point of view.

G-HLAM usage in the vascular reconstruction application is extended in comparison to the N-body example, which is at the same level of complexity as the first scenario of the medical application (Fig. 9.1). G-HLAM usage is depicted in Fig. 9.4. Now, instead of one parallel simulation such as in the N-body application, we have more simulations that can be placed at different Grid sites. Furthermore, there can be more than one visualization (more than one user). As described above and shown in the figure, the case study application consists of two kinds of modules: an MPI Lattice Boltzman simulation module and an integrated visualization-exploration module. For simplicity, in Fig. 9.4 we present an example with three simulations and two visualizations, but this can be easily extended. Moreover, we outline details of simulation only on Grid site A, however the same structure applies to sites B and C, where simulation is shown as one box.

As can be seen, there are two kinds of HLA federations : one application federation and N internal G-HLAM control federations, where N is the number of Grid sites used by simulation modules.

Application federation. Communication between application components is performed by HLA, so in Fig. 9.4 all simulations and visualizations are connected to the application federation. Simulations are connected to that federation only by their MPI root processes which are responsible for sending data to visualization. This federation is specific to the application and has been designed by its developer.

G-HLAM Control Federations. On each site, there is an *HLA-Speaking Service* for controlling simulation by G-HLAM. The service is an interface between G-HLAM and application federates. It starts application processes on its site and sends save/restore request from G-HLAM when there is a need for migration. A detailed description of the *HLA-Speaking Service* is provided in Chapter 4. Each site has its own control federation for communicating with the *HLA-Speaking service* residing on this site and

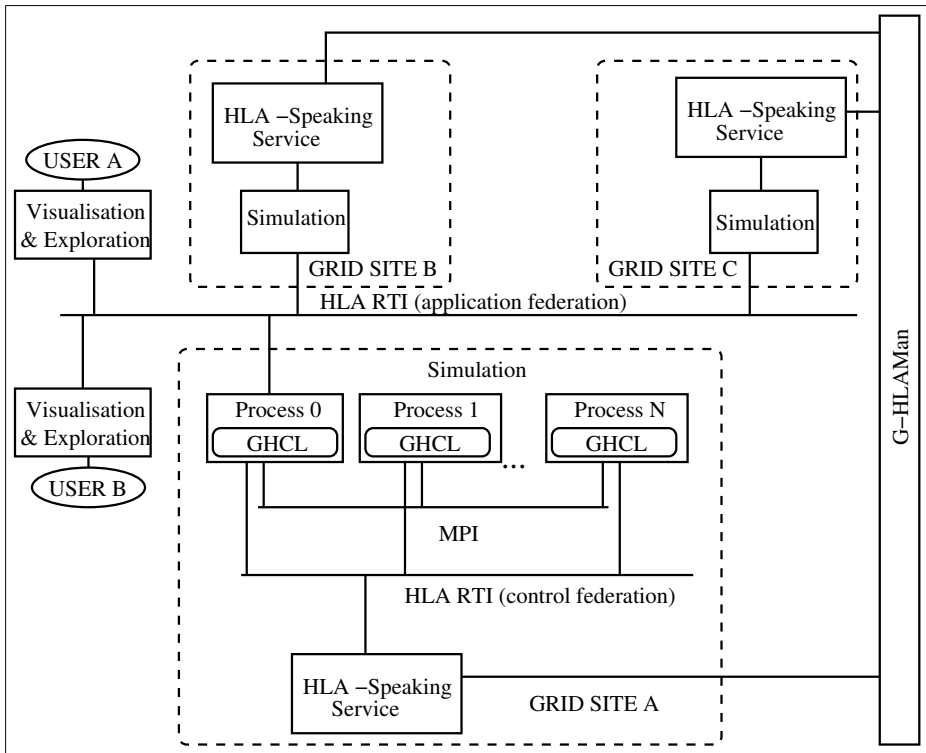


Figure 9.4: G-HLAM for the medical application – example for three simulations and two users. Application federation connects all simulations and visualisations. On each site, there is a control federation that interfaces simulation to G-HLAM.

all simulation processes join respective control federations through the GridHLAController library. The library is an interface between user code and G-HLAM: it passes save and restore requests to user code and assures that all user processes behave correctly, when one of them is migrated. A detailed description of the library and the migration mechanism can be found in Chapter 5.

9.3 Results

While in Chapter 7 we have presented performance results of the simple test applications, in this Chapter we prove that G-HLAM can be also successfully used with the realistic HLA-based collaborative environment. We present results from two experiments using G-HLAM for the prototype vascular reconstruction application. The prototype consisted of two types of modules communicating with HLA: *simulation modules* (MPI parallel simulation) and *visualization-receiver modules* (responsible for receiving data from simulation). As above, we use the last scenario of the applica-

tion (collaborative environment) since other scenarios are just specific cases thereof.

First, we show how migration overhead scales along with the number of simulation and visualization–receiver modules in the collaborative environment. We describe in detail the phases of the migration process as well as their duration. Next, we show how migration improves performance from the point of view of the user - i.e. how sending output data from the simulation changes after migration if the partial simulation results are actually watched by someone.

The experiments were performed on the DutchGrid DAS2 testbed infrastructure and at CYFRONET, Krakow, as shown in Tab. 9.1. In the first experiment we ran four

Operating System	Red Hat Enterprise Linux Advanced Server, version 3		
Network	10 Gbps (DAS2) + 155 Mbps (DAS2-Cyfronet)		
Role	Name	CPU	RAM
Migration source	DAS2 Nikhef	Pentium III 1 GHz	1 GB
Migration destination	DAS2 Leiden	Pentium III 1 GHz	2 GB
other visualizations and simulations	DAS2 Delft	Pentium III 2GHz	2 GB
	DAS2 Utrecht	Pentium III 1 GHz	1 GB
	DAS2 VU	Pentium III 1 GHz	2 GB
RTIexec	Cyf Krakow	Xeon 2.4 GHz	1 GB

Table 9.1: Grid testbed infrastructure

simulations (each containing 12 MPI processes) in our collaborative environment and the number of visualization–receivers varied from 3 to 22. We assigned visualization–receivers for each simulation in the most balanced way possible, so that each simulation had an equal number of visualisation–receivers collecting its data (exact to the remainder of dividing the number of visualizations and the number of simulations). We then migrated one of the simulations.

In the second experiment one simulation was migrated. The number of visualization–receivers was fixed to 25.

We observed that the type of module (simulation or visualization–receiver) does not have any impact on migration time. This is because only one (master) process of the parallel simulation participates in the application federation and its role is equal to a single visualisation–receiver process. Therefore, we plot migration time as a function of all federates in the application federation regardless of their type.

In our experiments, in each step, the simulation produces 52000 velocity vectors of simulated blood flow in 3D space. For our experiments we used GT v3.2 and HLA RTI 1.3v5. The results were obtained as an average from 10 measurements. The error bars indicate estimated standard deviation.

Migration Overhead

As already stated in the previous Chapter, according to the scenario presented in Fig. 5.2 in Section 5.2.2, migration time can be described by the following formula:

$$t_{migration} = t_{save} + t_{resignapp} + t_{resigncontrol} + t_{transfer} + t_{joincontrol} + t_{joinapplication} + t_{restore} \quad (9.1)$$

where

- $t_{migration}$ is the total migration time
- t_{save} is the time of saving application federation state
- $t_{resignapp}$ is the time of resigning from the application federation
- $t_{resigncontrol}$ is the time of resigning from the control federation
- $t_{transfer}$ is the time of transferring checkpoints
- $t_{joincontrol}$ is the time of joining the control federation
- $t_{joinapplication}$ is the time of joining the application federation
- $t_{restore}$ is the time of restoring the application federation's state

In our application the time of saving, resigning, joining and restoring the application federation depends on the number of federate processes in the application federation (i.e. the number of simulation and visualization–receiver modules) as shown in Fig. 9.4. Below we present results of scalability measurements for those durations.

On the other hand, the time of resigning and joining the control federation depends only on the number of all processes in the MPI simulation, which is fixed for this experiment. Therefore the time of resigning and joining the control federation is constant.

Saving the application federation. Saving overhead is caused by internal federation-wide saving algorithm used to implement the HLA Save/Restore API [78]. It includes synchronization mechanism between federates, which assures that the state does not change while saving. RTIExec control process saves the internal states of all federates in a separate checkpoint files. Also state of FedExec control process (which is one per federation) is saved. Additionally, moved federate saves user data to a checkpoint file.

The Fig. 9.5 shows how saving overhead depends on the size of the application federation (that is the number of modules in collaborative environment). We can observe plateaus in the plot - that probably means that RTIExec uses buffering to process requests coming from federates during saving process.

We have approximated migration overhead by choosing the middle points of the plateaus and finding a linear function:

$$t_{save} = a_{save}n_f + b_{save} \quad (9.2)$$

where n_f is the number of modules. For experimental data we used the least squares method to calculate $a_{save} = 0.44$, $\sigma_a = 0.04$, $b_{save} = -1.1$ and $\sigma_b = 0.6$. The correlation coefficient for all experimental data is equal 0.950.

Resign the application federation. The complexity of the resigning operation in the HLA RTI implementation when using reliable transport should be linear with respect to the number of federates in the application federation (number of modules

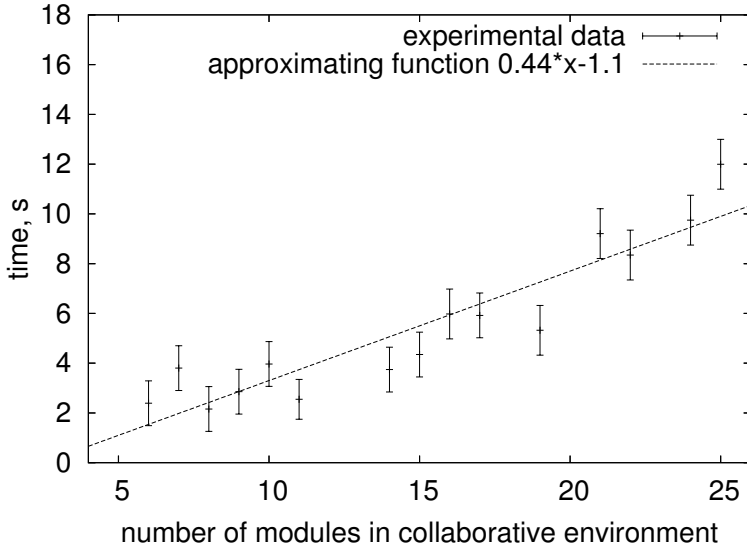


Figure 9.5: Time of saving application federation state as a function of the number of modules in the collaborative environment

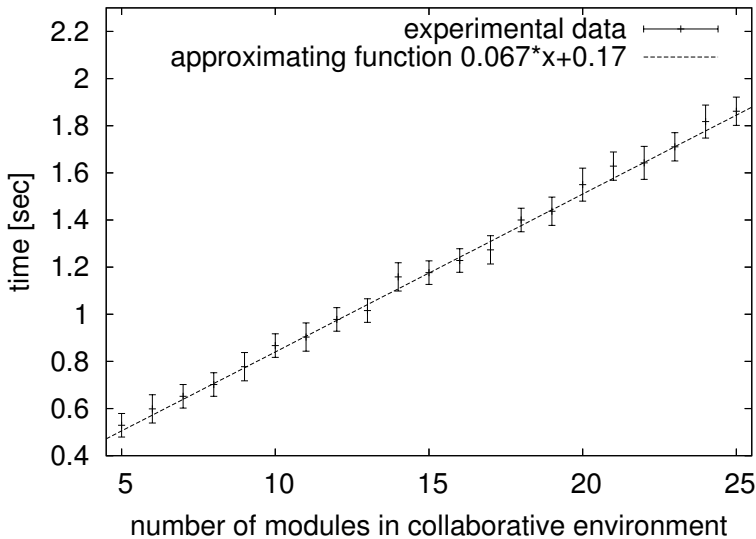


Figure 9.6: Time of resigning from the application federation during migration in Experiment 1 as a function of the number of modules in the collaborative environment

in the collaborative environment). This is because the resigning federate should close all TCP connections to all other federates. Fig. 9.6 shows that resigning time depends linearly on the number of federates. The approximating linear function is given by the following formula:

$$t_{resign} = a_{resign}n_f + b_{resign} \quad (9.3)$$

where n_f is the number of federates. For experimental data we calculated $a_{resign} = 0.067$, $\sigma_a = 0.001$, $b_{resign} = 0.17$ and $\sigma_b = 0.02$ using the least squares method. The correlation coefficient is equal 0.998.

Joining the application federation. The joining operation should be linear with

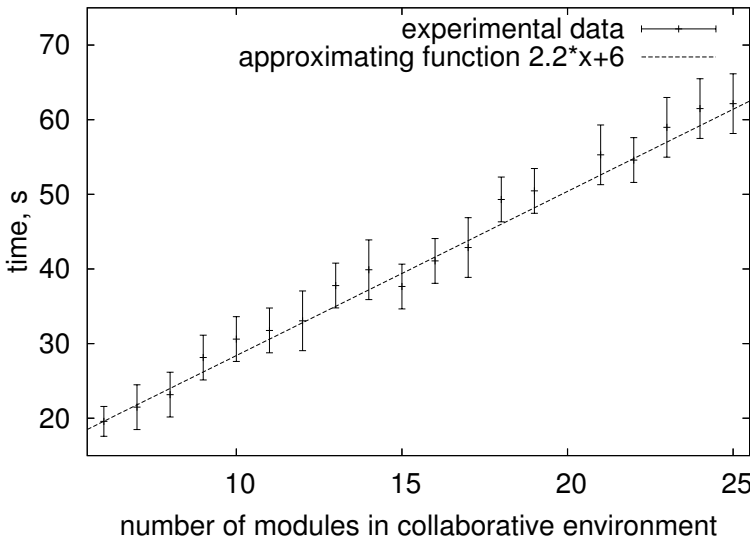


Figure 9.7: Time of rejoining the federation during migration as a function of the number of modules in the collaborative environment

respect to the number of federates in the application federation (modules in the collaborative environment) when using reliable transport in the HLA RTI implementation, since the joining federate has to open TCP connections to all other federates in the federation. Fig. 9.7 shows that joining federation time depends linearly on the number of federates. The approximating linear function is given by the following formula:

$$t_{join} = a_{join}n_f + b_{join} \quad (9.4)$$

where n_f is the number of federates. For experimental data we calculated $a_{join} = 2.2$, $\sigma_a = 0.1$, $b_{join} = 6$ $\sigma_b = 2$ using the least square method. Correlation coefficient is equal 0.994.

Restoring the application federation. Restoring overhead is caused by internal federation-wide restoring algorithm used to implement the HLA Save/Restore

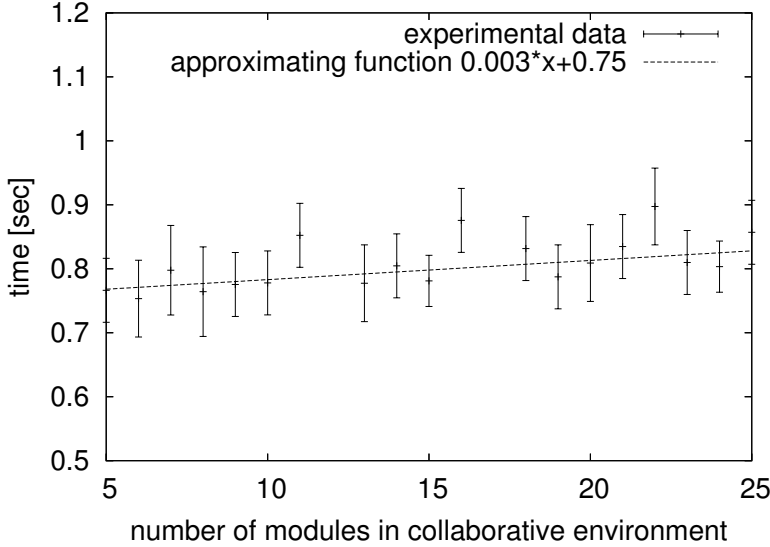


Figure 9.8: Time of restoring federation state during migration as a function of the number of modules in the collaborative environment

API [78]. As in the case of saving, it includes synchronization mechanism between federates assuring that the state does not change while restoring. RTIExec control process restores the internal states of all federates from checkpoint files. Also state of FedExec control process (which is one per federation) is restored. Additionally, moved federate restores user data from its checkpoint file.

The experimental data are shown in the Fig. 9.8. We can observe plateau in the plot - that probably means that RTIExec uses buffering to process requests coming from federates during restoring process. We have found an approximating linear function basing on the middle points of plateaus given by the following formula:

$$t_{restore} = a_{restore}n_f + b_{restore} \quad (9.5)$$

where n_f is the number of federates. For experimental data we calculated $a_{restore} = 0.003$ $\sigma_a = 0.001$ $b_{restore} = 0.75$ $\sigma_b = 0.02$ using the least squares method. Correlation coefficient is equal 0.606.

Time of other activities Tab. 9.2 shows the activities performed during migration independent of the number of modules in the collaborative environment. They include joining and resigning the control federation, transferring files (GridFTP overhead) as well as GRAM and PBS overhead. They are independent of the number of modules (that is simulations and visualisation–receivers in the collaborative environment).

Total migration overhead. Total migration overhead can be calculated from formula 9.1 and the values derived above. Therefore, after adding all a and b factors, we

activity	\bar{x} [s]	σ [s]
control federation resigning time	1.3	0.1
transfer time	0.5	0.05
GRAM submission	4.0	0.1
PBS waiting time	6.0	0.7
control federation joining time	63	3

Table 9.2: Execution time of various migration stages independent of the number of modules in the collaborative environment

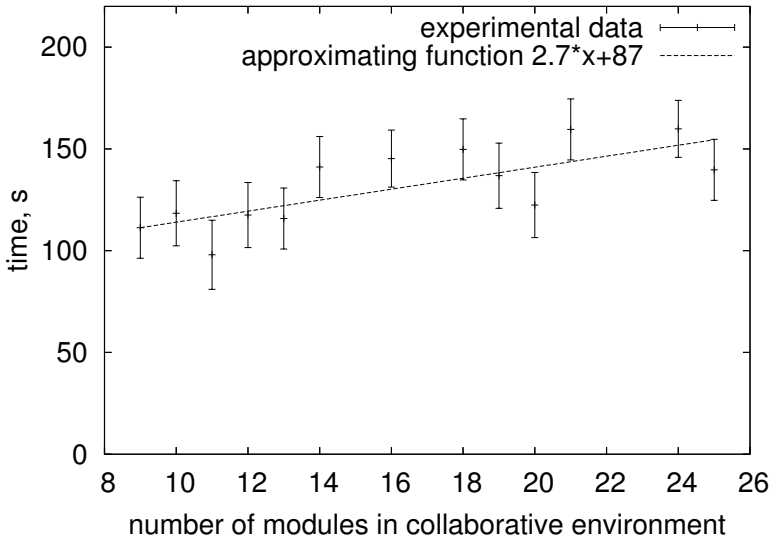


Figure 9.9: Total migration time as a function of the number of modules in collaborative environment federates

obtain:

$$t_{migration} = 2.73 * x + 88.9 \quad (9.6)$$

On the other hand, values A and B can be calculated directly from experimental data shown in the Fig. 9.9 which are combination of linear functions and plateaus. From our experiment $A = 2.7$, $\sigma_A = 0.8$ and $B = 87.0$, $\sigma_B = 13$. These values are equal in the range of error bars. The correlation coefficient for experimental data is equal 0.750.

Again, as can be observed, the largest overhead for the tested application is introduced by rejoining the control federation. The explanation is the same as in previous application case – in the used RTI implementation every federate has to open TCP connections to every other federate in the control federation. As stated before, this could be avoided if the HLA implementation uses reliable multicasting [112].

Impact of migration on performance of simulation within the collaborative environment

In this experiment we show how migration can improve the efficiency of simulation execution when its results are sent on-line to many users.

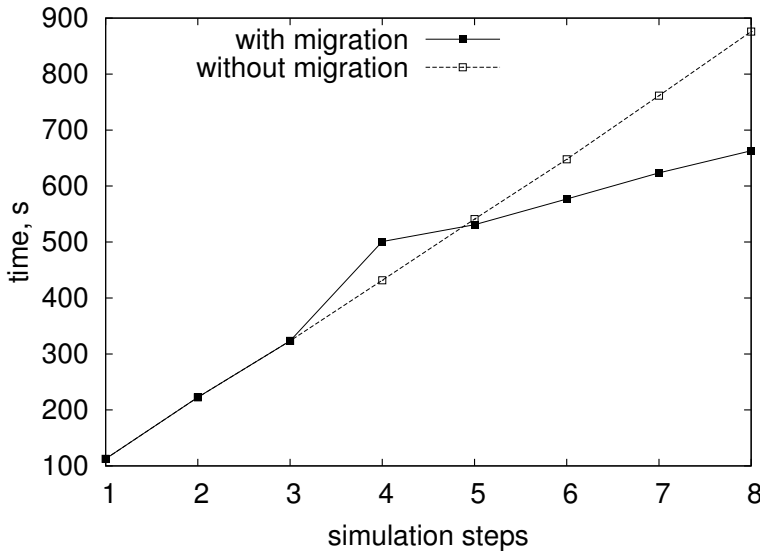


Figure 9.10: Impact of migration on simulation performance within collaborative environment

As in the case described in the previous Chapter, the bandwidth available for testing was broad (10 Gbps), so communication did not play an important role and calculations were the most time-consuming part of the execution. In order to create conditions in which migration would be useful, we increased the load of the Grid site where the simulation was executed (cluster in Amsterdam) by submitting non-related, computationally-intensive jobs. Next, we imitated a Resource Broker and migrated the simulation to another site which was not overloaded (cluster in Leiden). The experiments were performed at night in order to avoid interference with other users and repeated 10 times – each time we got slightly different values of response time, but the significant result (a user gain between steps 4 and 5) remains the same.

Fig. 9.10 shows the time as a function of the number of interactive steps with a human in the loop (for the first 8 steps). At each step, the simulation calculates data and sends it to the 25 visualization-receivers modules using HLA.

The dashed line shows the execution time of the simulation steps in the case when the simulation was not migrated. In this situation it is better to spend some time on migration to another site, from where the response time is shorter, as shown by the solid line in the figure. Fig. 9.10 shows that the human can gain access time between

steps 4 and 5, independently of the time lost on migration (performed between steps 3 and 4).

G-HLAM for N-body vs Vascular Reconstruction applications – discussion

In Chapter 8 and this Chapter we have described how G-HLAM works for two different applications. Both Chapters show that the most important factor that influences migration is rejoining to a federation. In our examples there were two federations: G-HLAM-specific control federation and application-specific federation.

The N-body application consists of an MPI parallel simulation and a visualisation / interaction module. As only one MPI process participates in application federation, the size of this federation is constant. On the contrary, all MPI processes participate in control federation. The most important factor that influences migration time is the overhead of rejoining to the control federation determined by the number of MPI processes. We have shown that the overhead has complexity $O(N^2)$, where N is number of MPI processes. This is because all MPI processes in simulation have to be moved and every MPI process have to reopen TCP connection with every other process.

On the contrary, the application federation of the collaborative environment for vascular reconstruction consists of many federates (each visualisation module is a federate and one MPI process from each simulation module is a federate). The migration time is not only influenced by a number of MPI processes which have to rejoin to the control federation, but also by a number of modules in the collaborative environment (other simulations and visualisations) that belong to application federation. In this Chapter, we have shown how the number of modules in application federation influences the migration time (with constant number of MPI processes – impact of a number of MPI processes on migration time is the same as for N-body application). Our results show that the overhead has complexity $O(N)$, where N is number of modules in collaborative environment. This is because only one MPI process of the whole migrated simulation has to reopen TCP connections with other modules of collaborative environment.

9.4 Summary and conclusions

In this Chapter we presented how the vascular reconstruction application can be run within the Grid HLA Management System (G-HLAM). Three different scenarios were presented: a scenario when a single user receives partial results of the simulation during runtime, a scenario with many users and one simulation, and finally a scenario with many users and many simulations (collaborative environment).

Apart from a functional description, we presented and described performance results of the G-HLAM system used for the last scenario (collaborative environment) since other scenarios are just specific cases thereof.

First, we measured how various migration phases impact migration time in a sample collaborative environment consisting of many simulation and visualisation modules. Subsequently, we presented results from an experiment which illustrates how

G-HLAM can improve the performance of a simulation running within the collaborative environment, decreasing the response time of simulation federates by migrating them to a better location.

Summing up, we can draw the following conclusions:

- the G-HLAM system can be used to improve execution of a simulation running within the collaborative environment on the Grid by performing migration of the simulation module to a different Grid site,
- during migration the most significant overhead is introduced by rejoining the federation. This issue, however, is specific to the HLA implementation used and can be reduced by introducing a reliable multicast mechanism [112],
- In our experiments we present the impact of migration on the foreground simulation (the simulation that actually sends data to one or more visualisations). However, in the presented collaborative environment, background simulations are also possible. In such simulations data is not sent to any of the visualisations and therefore the delay caused by migration can be hidden from the user.

Summary, Conclusions and Future Research

10.1 Summary

Developing runtime steering of distributed simulations is a very important and interesting issue in modern computer science.

In this thesis we have tried to show whether and how the modern scientific and technological solutions in computer science can be used to satisfy the requirements of distributed interactive simulation more accurately and efficiently than before. We aim not only at applications developed from scratch, but also at legacy codes. In order to answer this question, we have to analyze the requirements of distributed interactive simulations. First, distributed interactive simulations require synchronization support (time management) for event-driven and time-driven scenarios and the ability to connect simulations with different scenarios into one system. Next, they require efficient data distribution management and scalability with a number of distributed components. Additionally, they are often distributed using functional decomposition - each of their elements performs different functions and requires separate (specific) computer resources i.e. databases, visualization devices or computational power. Last but not least, this kind of applications requires an efficient environment, so that they can execute in near-real time with a user in the loop.

Our research presented in this thesis focuses on the requirements mentioned above, which are a starting point for the analysis of both the advantages and shortcomings of current scientific and technological solutions. HLA is a good candidate for fulfilling the requirements of synchronization management, data distribution support, scalability and the ability to connect time- and event-driven simulations into one system. Grid solutions are - by definition - a promising approach to accessing distributed resources. However, there still remains the issue of execution environment

efficiency. The HLA standard was developed assuming a certain quality of service in the underlying environment of the simulation execution. The Grid environment, however, is shared by many users and its conditions can change in an unpredictable way. Therefore, there is a need for a system that would adapt applications to this dynamically-changing environment and allow for their efficient execution.

In this thesis, we have proposed a Service Oriented Architecture of the G-HLAM system that fills the gap described above. The presented approach allows not only for building HLA-based applications that can be executed on the Grid, but also for easy adaptation of HLA legacy code.

The dynamic setup of HLA federates was one of our initial points of focus. In the G-HLAM system, we have introduced the *Broker Service* which uses the *Registry Service* to set up an HLA-based application from the *RTIExec Service*, responsible for management of the RTIExec process, and from *HLA-Speaking Services*, interfacing actual HLA federates with G-HLAM. There are two different versions of the *HLA-Speaking Service*: a prototype suitable for management of federates in one process and an extension, managing multiple processes. *HLA-Speaking Services* are a solution to the problem of turning user application federates into Grid services.

In order to provide a certain quality of the execution environment, we have decided to build a migration mechanism to allow federates to migrate when their performance is not satisfactory. The *Migration Service* is responsible for migrating HLA-connected components of the distributed application in the Grid environment. We have proposed user-level checkpointing, implemented through routines of the GridHLAController runtime support library. The library conceals details of RTI internal state checkpointing and provides a user-friendly interface to save and restore all the necessary data.

Efficient migration of HLA-based simulations requires a means of acquiring information about application performance. In this thesis, we have presented an HLA-based benchmark suite that helps the *Broker Service* decide where the application components should be placed. We have also presented the *Application Monitoring Service* which monitors the performance of HLA-based applications to decide when to migrate their parts to achieve more efficient execution. We have chosen the OCM-G system [12] as a monitoring system, since it was designed to be executed in a Grid environment and complies with the Online Monitoring Interface Specification (OMIS)[103]. We have shown how the design concepts of the OCM-G enable easy adaptation to monitoring of HLA, C++-based applications.

Finally, we have described performance results and shown that two sample realistic simulations can benefit from the Grid using the solutions proposed in this thesis. The chosen applications allows to show how G-HLAM deals with their different types of interactions (one-way or two-way interaction between a user and a simulation) and types of distribution (a single simulation for a single user, multiple simulations for a single user, collaborative environment for many users). We have presented the architecture of an N-body dense stellar system application and the supported types of simulation interaction, and shown how the application can be run within the Grid HLA Management System (G-HLAM). Subsequently, we have presented results from

Problem	Solution proposed in G-HLAM
Dynamic setup	Broker Service, Registry Service, RTIexec Service
Interfacing G-HLAM with HLA federates	HLA-Speaking Service
Ability to move federates in case of their poor performance	Migration Service
Acquiring information about actual application performance to decide when to move	Application Monitoring Service
Acquiring information about possible application performance to decide where to move	Benchmark Services

Table 10.1: Summary of problems and their solutions in G-HLAM

three different experiments showing how G-HLAM can improve the performance of "human in the loop" simulations. We have also shown the advantages of using the G-HLAM for running a large-scale HLA-based medical application supporting surgical pre-operative planning. We especially focused on its execution in a collaborative environment for many users. Our experiments show that G-HLAM is useful also for improving performance of such applications. A summary of problems solved by G-HLAM is presented in Tab. 10.1.

10.2 Conclusions

The central hypothesis of this thesis, as described in Section 1.4, is: *interactive distributed simulations can significantly benefit from the Grid environment and there exist solution for achieving their effective execution by filling the gap between interactive simulations and the Grid infrastructure.* Below we explain how scientific results presented in this thesis prove the central hypothesis. Here we will follow list of scientific contributions given in Section 1.4.

Analysis of requirements of distributed interactive simulations were presented in Section 1.3. As stated in Section 10.1, the requirements like synchronization support (time management) for event-driven and time-driven scenarios, the ability to connect simulations with different scenarios into one system, efficient data distribution management and scalability with a number of distributed components are fulfilled by choosing HLA as a support for distributed interactive simulations. Additionally, requirement of accessing distributed computer resources i.e. databases, visualization devices or computational power can be fulfilled by using Grid technology. Last but not least, the concerned simulations require an efficient environment, such that they can execute in near real-time with a user in the loop. We satisfy this requirement by introducing G-HLAM system presented in this thesis.

The evaluation of existing solutions and analysis of possible technologies, useful

with regard to these requirements, were described in Chapter 2, where we presented environments supporting the development, execution and/or steering of simulations. For each of these environments, we have analysed the advantages and disadvantages for adaptation of Grid solutions. We have also outlined the ideas that were used in the past to port HPC simulations to the Grid seen as a large, distributed metacomputing environment. Additionally, we have reviewed current efforts of using Grid concepts for distributed simulations. We have also described efforts of using Grid and Web solutions to make interactive distributed simulations more interoperable, fault-tolerant and efficient. According to our analysis of existing approaches made in Chapter 2, especially in Section 2.4, there was no solution that allowed to run HLA simulations, including legacy codes, on the Grid in efficient way.

The explanation why HLA is an appropriate standard for distributed interactive simulations, when there is a need to run them on the Grid, is described in Section 2.1.8, Section 2.1.13, Section 2.4 and Section 3.1.3. To summarize, HLA is already designed for running scalable geographically-distributed simulations, offers services for efficient data exchange between distributed federates (data management) and synchronization (time management). It allows for building scalable simulation systems and has features that facilitate simulation interoperability. Section 2.4 presents different aspects of merging HLA and Grid concepts: interoperability of different simulation models, fault tolerance, support for peer-to-peer collaborative applications and effective management of HLA-based simulations.

The Grid environment *can* be beneficial for distributed interactive simulations and provides them with new opportunities not previously available. This is realized in two steps.

First, our goal was to analyze possibilities offered by the Grid and defining requirements for their efficient usage, which was described in Section 1.5, Section 2.3, Section 3.1.2 and Section 3.1.3. To summarize, Grid technology extends the set of possible distributed resources that can be used and facilitates access to them. The Grid can be useful for those simulations that consist of components with different functionalities, requiring various, often geographically-dispersed resources. Additionally, Grid services together with the Web services concept introduce the definition of abstract interfaces that allow services to cooperate without too much concern for the actual protocols being used and their internal technology.

Secondly, we have presented the design (Chapters 3–6), implementation (Chapters 7) and feasibility (Chapters 8–9) of the solution for efficient usage of the Grid for distributed interactive simulations. The analysis of requirements have shown that this goal cannot be achieved by simply running HLA applications on the Grid on an "as is" basis. The result is the Grid HLA Management System (G-HLAM). We have presented solutions to many problems that emerged when designing and developing G-HLAM. First, we have addressed the issue of automatic setup of HLA-based simulations in the Grid environment. Next, we have designed and discussed the relation between user application federates and Grid services, the interface of these services to legacy federates, and the management of multiple federate processes in an efficient way. To show that migration can improve the performance of HLA-based simulations,

we built a mechanism that assures consistency of HLA services – such as *data management*, *ownership management* and *time management* - during migration of single or multiple federates within a federation. Furthermore, we have designed mechanisms for providing information about HLA application performance on the Grid as well as information about its hypothetical behaviour in a given part of the Grid infrastructure. It is also worth mentioning that, as shown in Chapters 8 and 9, we succeeded to obtain one of our main goals to support legacy HLA applications and to create scalable, flexible and secure solutions with not much overhead.

We conclude that G-HLAM presented in this thesis fills the gap between HLA-based distributed interactive simulations and the Grid infrastructure in an optimal way. The performance results presented in Chapter 7 and feasibility study with two realistic simulations described in Chapter 8 and 9 showed that the distributed interactive simulations can be run on the Grid more efficiently using the solutions proposed in this thesis.

10.3 Future research directions

When looking at the lifecycle of a simulation being executed on the Grid, one can distinguish different phases - first, the distributed simulation should be composed from appropriate models, next it should be set up and, finally, it should be efficiently executed on the Grid. This thesis focuses on the resource management part of Grid research. We assume that the distributed simulation is provided by a user who wants to benefit from geographically-distributed resources spread across various administrative domains in an efficient way. The results presented in this thesis show that this goal is now achievable.

However, there is also another aspect of Grid computing that can prove useful for simulation developers at the beginning of the simulation lifecycle – there are efforts to use semantic descriptions of software components that exist on the Grid in order to arrange them into workflows of more complex applications [23]. This approach is particularly promising in the field of distributed simulations. There exist simulation models that can be reused in many applications, i.e. the factory model used in the supply chain simulation which manages material and information flow from manufacturers through distributors to customers [165]. There are also ideas to describe interfaces of such models in a universal way [41, 194, 134] to allow for their automatic composition in complex simulations. One of the important initiative that should be mention here is a development of the Base Object Model (BOM) [20] component-based standard for describing a reusable part of a federation or a federate. BOM specification offers an ontology for characterizing elements of a simulation and relationships among conceptual entities within a simulation environment.

Also, in the world of Massive Multiplayer Online Games there is an research effort in the direction of model composition. The idea is to use concepts not only from SOA, but also from Model Driven Architecture (MDA) [121] and Event Driven Architecture (EDA) [33]. An interesting approach, being developed by Magnetar Games

corporation [105], is to build a federated extensible scene graph [66, 132] for holding Federation Object Model (a specification defining the information exchanged at runtime between federates) [78] and Base Object Model (BOM) [20] data.

In general, pursuing this direction of research is very challenging and raises many open questions i.e. regarding the method of defining interfaces of simulation models, the way of discovering them, HLA compliance, conformance to standard reference models (e.g. HLA-CSPIF [41]) etc. The author of this thesis is convinced that both issues (composability of simulation models and management of their actual execution) are complementary and can benefit from each other.

There are several research areas that can be investigated further. We briefly mention possible extensions to G-HLAM functionality and present technological migration issues.

10.3.1 Possible extension of the G-HLAM functionality

Broker Service and Performance Decision Service

In this thesis we have proposed monitoring services that could support a *Broker Service* in its decisions. Performing adequate brokering in spite of the highly-changeable nature of the network and host environments is a nontrivial task. In the Grid community there is ongoing research on metascheduling systems [113, 71, 151, 187] that can be extended for our purposes. Furthermore, by means of *Performance Decision Service* which processes data from the Application Monitoring Service, an HLA-based distributed simulation can be illustrated as a graph of connected federates. The aim of the broker would be to place those federates on the available Grid sites in the most efficient way. This problem is related to the graph partitioning problem, where one looks for separation of vertices of a graph into near-equal-sized components such that the number of edges between the components is minimized. Although this is an NP-complete problem, many heuristic solutions are known [149, 158, 147] and can be applied for efficient distribution of HLA-based simulations over Grid nodes in a way which minimizes communication between them.

Fault tolerance

The migration mechanism presented in this thesis provides a good starting point to achieving fault tolerance in our system, since checkpointing can be used not only in the case of bad performance, but also to create backup when a failure occurs. This can be achieved by extending the *Migration Service* with a periodic checkpointing ability and creating mirror sites for the simulation that can take over if the primary site fails.

10.3.2 Technological migration possibilities

Towards Web Service Resource Framework solutions

The Open Grid Service Infrastructure(OGSI)[61] was chosen as a platform for the development of G-HLAM Services. However, while this thesis was being written, OGSI authors created a new specification called the Web Services Resource Framework (WSRF)[180] which became the successor of OGSI.

Both OGSI and the WS-Resource Framework deal with manipulating stateful resources through Web service interfaces. The difference is in the manner of modeling – OGSI encapsulates service state in a Grid service, which is an extension of a Web service, while WSRF uses classical Web services as an interface to stateful WS-Resources. Both approaches provide essentially equivalent functionality and use semantically similar WSDL interface definitions allowing for creation, addressing, inspection and destruction in essentially the same ways.

Therefore, the G-HLAM architecture will not change since it is completely separate from implementation details. In the case of migration to WSRF, the stateful services described in this thesis, created for management of an HLA-based application, should be changed to WS-Resources interfaced by Web services. Details of refactoring OGSI Grid services to WSRF can be found in [44].

Components architectures solutions

Recently, a lot of effort has been invested in developing Component Based Environments such as CCA [31], H2O [96] and ProActive [133]. Component architectures are a very promising approach due to such features as lightweight environments, dynamic behavior, scalability to suit various environmental requirements etc. One can also think of applying the advantages of these technologies to distributed interactive simulations. The component approach can be useful for defining interoperable and reusable simulation models. Such reusable models can then be automatically arranged into workflows of distributed applications as mentioned in Section 10.2. The systems like Triana [163], Kepler [102] or Tawerna [120] can be used to coordinate such workflows within an distributed simulation environment.

A

Operations of HLA-Speaking Porttype

A.1 Version for a single process

`start` loads the federate code and required libraries using Java Native Interface (JNI) [88].

`requestRun` initiates the federate process execution.

`requestMigrationSave` saves the federate and RTI state to a file.

`requestRunWithRestore` starts the federate from the checkpoint file.

A.2 Version for multiple processes

`startControl(GridFTP location of RTIExec endpoint file)` - fetches RTIExec endpoint file (determined by *Broker Service* and necessary for automatic connection of federates to the RTI – see Section 3.2.3), creates control federation and starts the control federate.

`take(GridFTP location of user federate code)` – fetches federate code. Returns `id` of this request (to be used by `requestRun()` and `requestRunWithRestore()` command described below).

`requestRun(command, id)` - starts the federate codes specified in the command.

`requestMigrationSave(contactString)` – saves state of all processes indicated in `contact string`. The `contact string` consists of a list of pids and hosts of processes. If other processes are also submitted using the same RSL command, all those processes will be saved. The method returns a structure containing the original command used for submission of these processes and lists of their pids and hosts. The host and pid of a process determine the name of a checkpoint file where the state of user data is stored and is used by the next method described below.

`requestRunWithRestore(command, restoreName)` – restores appropriate feder-

ate processes from the checkpoint file. The indicated federates rejoin their federations. The restore name contains pids and hosts of restored processes returned by `requestMigrationSave()` and determines names of checkpoint files as described above.

`takeAndSubmit(command, restoreName, GridFTP location of codes)` **performs two operations:** `take()` and `RequestRun()` (or `take()` and `RequestRunWithRestore()` – if `restorename` is indicated), both as a single operation.

B

Grid HLA Controller Library

The GridHLAController library (GHCL) is an interface between the Grid service layer and the user federate. It is used to notify the user federate about saving and restoring requests which come from the *Migration Service* to the *HLA-Speaking Service* and finally to the user federate. The library contains two group of routines: initialization functions (described in Section B.1) and migration functions (Section B.2).

B.1 Initialization routines

The functionality of the GridHLAController library (GHCL) is encapsulated within the GridHLAController class. This Section describes initialization routines of the library used to connect user federate processe(s) with G-HLAM system.

B.1.1 A single process version

`static int GridHLAController::main (GridHLAController*,...)` starts the program and should be used instead of a standard `main()` function. It provides a pointer to a GridHLAController object that is necessary to call other functions of the library.

`GridHLAController::register_start()` registers the main simulation loop (the callback defined by the user). This function should be invoked inside the `GridHLAController::main()` routine.

`GridHLAController::set_rtiamb_count()` informs the migration engine about the number of federations in which a federate is participating.

`GridHLAController::add_RTIambassador()` provides a reference of RTIambassadors of those federations to the GridHLAController library.

B.1.2 A multiple processes version

`GridHLAController::init()` routine is invoked by a user to initiate all the necessary functionality in the `GridHLAController` library, create a control federate in the user process and connect it to the control federation.

`GridHLAController::set_rtiamb_count()` informs the migration engine about the number of federations in which a federate is participating.

`GridHLAController::add_RTIambassador()` provides a reference of RTIambassadors of those federations to the `GridHLAController` library.

B.2 Migration routines

As described in Section 3.3.3, the library manages four-bit state of the federate process which indicates the appearance of external saving, internal saving, external restoring, and internal restoring requests respectively.

An external request is defined as one coming from the *Migration Service* invoking the `RequestMigrationSave` or `RequestRunWithRestore` operations on the *HLA-Speaking Service*. This means that the federate process should be completely saved and stopped or restored from a checkpoint file. Internal requests are sent using a federation-wide save/restore mechanisms and are only used to freeze the federation.

The migration support functions are divided into following groups:

Checking external migration or restore requests. Those functions check if an external request from the *Migrator Service* has arrived:

`GridHLAController::check_restore_state()` checks if the federate is being restored from the checkpoint file (if the state is set to $[*,*1,*]$ ¹ by the control federate) and if so, performs all the necessary steps to restore the internal state of the RTI.

`GridHLAController::check_migration_state()` checks if a save request has been received (if the state is set to $[1,**,*]$ by the control federate) and performs all the necessary HLA steps to save the internal state of the RTI.

Checking internal RTI saving/restoring requests. This type of routines should be used while implementing RTI callbacks of the Federate Ambassador. They pass information provided by RTI to the `GridHLAController` library:

`GridHLAController::save_on()` notifies GHCL that RTI has sent a request message (change to state $[*1,*,*]$). `GridHLAController::save_off()` notifies GHCL that RTI has finished saving internal data (change to state $[*0,*,*]$).

`GridHLAController::restore_on()` notifies GHCL that RTI has sent a restore request message (change to state $[***,1]$).

`GridHLAController::restore_off()` notifies GHCL that RTI has finished restoring internal data (change to state $[***,0]$).

The Petri Net-based diagram illustrating state transitions is described in Section

¹an asterisk (*) indicates any two-bit state (0/1)

3.3.3.

Saving and restoring user-specific values. These routines provide an API for saving and restoring user-specific values. All user objects that have to be saved must have a defined stream operator that will be invoked by GHCL during saving or restoring process. It is enough to save handles of user created HLA objects, they become accessible after migration. User open files have to be reopened by a user after migration, GHCL allows to checkpoint only user file names and G-HLAM supports automatic transfer of these files together with application codes. A user has also to reopen external connections (i.e. sockets) other than those provided by HLA.

`GridHLAController::init_save()` opens an output checkpoint file.

`GridHLAController::save()` saves the object to the checkpoint file. The object must have a defined stream operator.

`GridHLAController::exit_save()` closes the output checkpoint file.

`GridHLAController::init_restore()` opens the input checkpoint file.

`GridHLAController::restore()` fills an object with data saved in the checkpoint file. The object must have a defined stream operator.

`GridHLAController::exit_restore()` closes the input checkpoint file.

Fig.B.1 presents a sample use of the GHCL library in an RTI federate simulation loop. The code first joins an application-specific federation, then checks if this execution should be restored from the file using

the `GridHLAController::check_restore_state()` routine. If so, appropriate functions for restoring user data are invoked. If not, the publishing and registration of the data object is done normally. Afterwards, the program enters the main simulation loop. Inside the loop the `GridHLAController::check_migration_state()` function is invoked. If it returns true, the GHCL library is used to checkpoint user-specific data and then cleanup is performed (this includes unpublishing and resigning from federation). In other cases, the simulation proceeds normally.

Although the presented example shows a specific algorithm (loop with an explicit loop counter), the design of the GHCL API is flexible enough to be applied to other types of simulations.

```

void My_Federate::PrimarySimulation (
GridHLAController* control,char* federation_name,
char* federate_name)
{
    My_Data *myd;
    int i=0;
    int* begin=new int;

    /*Create and join federation.*/
    CreateAndJoinFederation(federation_name,
        federate_name);

    /*Check if we are restoring
    from the checkpoint.*/

    int restore_state=
    control->check_restore_state();

    /*Create data object.*/
    myd=new My_Data(rti_ambassador);

    if (restore_state){
        /*Data Object is already registered
        - restore its data.*/
        control->restore(myd);
        control->restore(begin);
    }
    else{
        /*We not in restore mode -
        register the data object.*/
        *begin=0;
        myd->Publish();
        myd->Register();
    }

    /*Start simulation loop.*/
    for(i=*begin;i<NUMBER;i++){

        /*Update values of data
        object in tuple space.*/

        myd->Update();

        /*Provide some time for the RTI.*/
        rti_ambassador->tick(1.0, 1.0);

        /*Check if migration request has arrived.
        and save internal RTI state*/
        if (control->check_migration_state()){

            /*Save user specific data */
            control->init_save();
            control->save(fedamb);
            i++;
            control->save(myd);
            control->save(&i);
            control->exit_save();

            /*Unpublish data object, resign from
            the federation execution and attempt
            to destroy.*/
            myd->Delete();
            myd->Unpublish();
            delete myd;
            ResignAndDestroyFederation(
                nam->federation_name,
                nam->federate_name);
            return();
        }
    }

    /*Data object is not longer needed.*/
    myd->Delete();
    myd->Unpublish();
    delete myd;

    /* Resign from the federation
    execution and attempt to destroy.*/

    ResignAndDestroyFederation(
        federation_name, federate_name);
}

```

Figure B.1: Sample RTI Federate simulation loop

References

- [1] L. Abrahamyan, J. A. Schaap, A. G. Hoekstra, D. P. Shamonin, F. M. A. Box, R. J. van der Geest, J. H. C. Reiber, and P. M. A. Sloot. A Problem Solving Environment for Image-Based Computational Hemodynamics. In V. S. Sunderam, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2005: 5th International Conference, Atlanta, GA, USA, Proceedings, Part I*, volume 3514 of *Lecture Notes in Computer Science*, pages 287–294, Berlin, Heidelberg, May 2005. Springer.
- [2] D. Abramson, J. Giddy, and L. Kotler. High Performance Parametric Modeling with Nimrod/G: Killer Application for the Global Grid? In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 520– 528, May 2000.
- [3] Access Grid Project home page. <http://www.accessgrid.org/>.
- [4] ACK Cyfronet. <http://www.cyfronet.krakow.pl>.
- [5] ADST II Transitions Simulators to HLA Using MK Technologies VR-Link. http://www.mak.com/profile_adst.pdf.
- [6] AFRL Warfighter Training Research Division Uses MK RTI for HLA Networking. http://www.mak.com/profile_afrl.pdf.
- [7] G. Allen, W. Bengler, T. Dramlitsch, T. Goodale, H. Hege, G. Lanfermann, A. Merzky, T. Radke, E. Seidel, and J. Shalf. Cactus Tools for Grid Applications. *Cluster Computing*, 4(3):179–188, 2001.
- [8] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel, and B. Toonen. Supporting Efficient Execution in Heterogeneous Distributed Computing

- Environments with Cactus and Globus. In *Proceedings of SC 2001, November 10-16, 2001*. <http://www.globus.org/alliance/publications/papers/>.
- [9] D. Anagnostopoulos and M. Nikolaidou. Executing a Minimum Number of Replications to Support the Reliability of FRTS Predictions. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 138–146, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [10] A. G. Artoli, A. M. Hoekstra and P. M. A. Sloot. Simulation of a Systolic Cycle in a Realistic Artery with the Lattice Boltzmann BGK Method. *Int. J. Mod. Phys. B*, 17:95–98, 2003.
- [11] AVS/Advanced Visual Systems. <http://www.avs.com/>.
- [12] B. Baliś, M. Bubak, W. Funika, T. Szepieniec, R. Wismüller, and M. Radecki. Monitoring Grid Applications with Grid-enabled OMIS Monitor. In F. Riviera, M. Bubak, A. Tato, and R. Doallo, editors, *Proc. First European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 230–239. Springer, Feb. 2003. <http://www.icsr.agh.edu.pl/ocmg>.
- [13] S. Barnard, R. Biswas, S. Saini, R. Van der Wijngaart, M. Yarow, L. Zechter, I. Foster, and O. Larsson. Large Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus. In *The Seventh Symposium on the Frontiers of Massively Parallel Computation*, February 1999.
- [14] R. G. Belleman. *Interactive Exploration in Virtual Environments*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, April 2003. Promotor: Prof. Dr. P. M. A. Sloot.
- [15] R. G. Belleman, J. A. Kaandorp, D. Dijkman, and P. M. A. Sloot. GEOPROVE: Geometric Probes for Virtual Environments. In P. M. A. Sloot, M. Bubak, A. G. Hoekstra, and L. O. Hertzberger, editors, *High-Performance Computing and Networking (HPCN Europe '99)*, Amsterdam, The Netherlands, number 1593 in *Lecture Notes in Computer Science*, pages 817–827, Berlin, April 1999. Springer-Verlag.
- [16] R. G. Belleman and P. M. A. Sloot. The Design of Dynamic Exploration Environments for Computational Steering Simulations. In M. Bubak, J. Mościnski, and M. Noga, editors, *Proceedings of the SGI Users' Conference 2000*, pages 57–74, Krakow, Poland, October 2000. Academic Computer Centre CYFRONET AGH.
- [17] S. Benkner, G. Englebrecht, G. W. Backfrieder, G. Berti, J. Fingberg, G. Kohring, J. G. Schmidt, S. E. Middleton, D. Jones, and J. Fenner. Numerical Simulation for eHealth: Grid-enabled Medical Simulation Services. In *Proceedings of PARCO2003*, pages 705–718, 2003.

- [18] S. Berger and L.-D. Jou. Flows in stenotic vessels. 32:347–382, 2000.
- [19] M. Boasson. Modeling and Simulation in Reactive Systems. In *WPDRTS '96: Proceedings of the 4th International Workshop on Parallel and Distributed Real-Time Systems*, page 27, Washington, DC, USA, 1996. IEEE Computer Society.
- [20] Base Object Model (BOM) home page. <http://www.boms.info/>.
- [21] J. Brooke, T. Eickermann, and U. Woessner. Application Steering in a Collaborative Environment. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 61, Washington, DC, USA, 2003. IEEE Computer Society.
- [22] M. Bubak, W. Funika, R. Wismüller, T. Arodź, and M. Kurdziel. The G-PM Tool for Grid-Oriented Performance Analysis. In F. Riviera, M. Bubak, A. Tato, and R. Doallo, editors, *Proc. First European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 240–248. Springer, Feb. 2003.
- [23] M. Bubak, T. Gubała, M. Kapalka, M. Malawski, and K. Rycerz. Workflow Composer and Service Registry for Grid Applications. *Future Generation Computer Systems*, 21(1):79–86, January 2005.
- [24] M. Bubak, M. Malawski, and K. Zajac. Towards the CrossGrid Architecture. In D. Kranzlmüller, P. , Kacsuk, J. Dongarra, and J. Volker, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, Proc. 9th European PVM/MPI Users*, volume 2474 of *Lecture Notes in Computer Science*, pages 16–24. Springer Verlag, 2002.
- [25] Condor ByPass home page. <http://www.cs.wisc.edu/condor/bypass/>.
- [26] W. Cai, S. J. Turner, and H. Zhao. A Load Management System for Running HLA-based Distributed Simulations over the Grid. In S. J. Turner and S. J. E. Taylor, editors, *Proceedings of the sixth IEEE International Workshop on Distributed Simulation and Real Time Applications (DS-RT 2002)*, pages 7–14, Fort Worth, Texas, October 2002. IEEE Computer Society.
- [27] W. Cai, Z. Yuan, M. Y. H. Low, and S. J. Turner. Federate Migration in HLA-based Simulation. *Future Generation Computer Systems*, 21(1):87–95, January 2005.
- [28] CAVElib home page. <http://www.vrco.com>.
- [29] CAVERNUS library home page. <http://www.ncsa.uiuc.edu/VR/cavernus/>.
- [30] D. Cavitt, C. Overstreet, and K. Maly. A Performance Monitoring Application for Distributed Interactive Simulations (DIS). In *Proceedings of the 29th conference on Winter Simulation*, pages 421–428. ACM Press, 1997.

- [31] The Common Component Architecture Forum, 2004. <http://www.cca-forum.org/>.
- [32] F. R. Cecin, R. Real, R. de Oliveira Jannone, C. F. R. Geyer, M. G. Martins, and J. L. V. Barbosa. FreeMMG: A Scalable and Cheat-Resistant Distribution Model for Internet Games. In *8th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT 2004), 21-23 October 2004, Budapest, Hungary*, pages 83–90. IEEE Computer Society, 2004.
- [33] K. Chandy, J. L. Carmona, and R. Alexander. Event-Driven Architecture: Event Web Building Block. <http://www.developer.com/design/article.php/3490671>.
- [34] CORBA project home page. <http://www.corba.org/>.
- [35] Grid projects funded by the EU under FP6. <http://www.cordis.lu/ist/grids>.
- [36] CoreGrid project home page. <http://www.coregrid.net>.
- [37] P. Coveney, J. Vicary, J. Chin, and M. Harvey. Introducing WEDS: a WSRF-based Environment for Distributed Simulation. NeSC Technical Report UKeS-2004-07 http://www.nesc.ac.uk/technical_papers/.
- [38] CrossGrid project home page. <http://www.eu-crossgrid.org>.
- [39] CSC Chooses MK s VR-Link for Virtual Ship 2000 HLA Compliance. http://www.mak.com/profile_cscadvancedmarine.pdf.
- [40] CSE project home page. <http://www.cwi.nl/projects/cse/cse.html>.
- [41] HLA CSPIF home page. <http://www.cspif.com/>.
- [42] C. J. Cuhna, F. Omer, and D. Pedro. Future Trends in Distributed Applications and Problem Solving Environments. *Future Generation Computer Systems*, 21(6):843–855, June 2005.
- [43] Cybernet Corporation. <http://www.cybernet.com/>.
- [44] K. Czajkowski, D. Ferguson, I. Foster, J. Frey, S. Graham, T. Maguire, D. Snelling, and S. Tuecke. From Open Grid Services Infrastructure to WS-Resource Framework, March 2004. <http://www.globus.org/alliance/publications/papers.php>.
- [45] DataGrid project home page. <http://www.eu-datagrid.org>.
- [46] W. J. Davis. On-line Simulation: Need and Evolving Research Requirements. In *Handbook of simulation*, pages 465–516. John Wiley, 1998.

- [47] Differentiated Quality of Service. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm.
- [48] IEEE Standard for Distributed Interactive Simulation–Application Protocols . <http://standards.ieee.org/catalog/olis/compsim.html>.
- [49] Distributed ASCII Supercomputer. <http://www.cs.vu.nl/das2/>.
- [50] G. R. Drake and J. S. Smith. Simulation system for real-time planning, scheduling, and control. In *WSC '96: Proceedings of the 28th conference on Winter simulation*, pages 1083–1090, New York, NY, USA, 1996. ACM Press.
- [51] DS-Grid Project Home page. <http://www.cs.bham.ac.uk/research/projects/dsgrid/>.
- [52] Information about Ducat simulation. <http://www.pitch.se/defense/default.asp>.
- [53] M. Eklöf, M. Sparf, and F. Moradi. HLA Federation Management in a Peer-to-Peer Environment. In *Proceedings of the 03 European Simulation Interoperability Workshop*, Stockholm, Sweden, 2003. Paper number 03E-SIW-053 available at: <http://www.sisostds.org/>.
- [54] Engima Rising Tide game. <http://www.warfleet.net>.
- [55] FATS Chooses MK VR-Link for all HLA Requirements. http://www.mak.com/profile_fats.pdf.
- [56] Federation Grid. <http://www.federationgrid.org/Overview.htm>.
- [57] FederationX home page. <http://www.federationx.net/Main.htm>.
- [58] I. Foster. What is the Grid? A three checkpoints list. *GridToday Daily News And Information For The Global Grid Community*, 1(6), July 2002.
- [59] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, July 1998.
- [60] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *Intl J. Supercomputer Applications*, 11(2):115–128, 1997.
- [61] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002. <http://www.globus.org/alliance/publications/papers.php>.

- [62] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International J. Supercomputer Applications*, 15(3), 2001. <http://www.globus.org/alliance/publications/papers.php>.
- [63] R. M. Fujimoto. Parallel Simulation: Distributed Simulation Systems. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, pages 124–134. Winter Simulation Conference, 2003.
- [64] H. Gabbar, S. Shinohara, Y. Shimada, and K. Suzuki. Experiment on Distributed Dynamic Simulation for Safety Design of Chemical Plants. *Simulation Modeling Practice and Theory*, pages 109–123, 2003.
- [65] B. Gaidioz, R. Wolski, and B. Tourancheau. Synchronizing Network Probes to Avoid Measurement Intrusiveness with the Network Weather Service. In *9th IEEE Symposium on High Performance Distributed Computing*, pages 147–154, August 2000. <http://www.cs.utk.edu/~rich/publications/nws-period.ps.gz>.
- [66] GameXML project home page. <http://www.gamexml.org/>.
- [67] GARA home page. <http://www-fp.mcs.anl.gov/qos/>.
- [68] GARNET home page.
<http://www-fp.mcs.anl.gov/qos/garnet.html>.
- [69] Global Grid Forum home page. <http://www.gridforum.org>.
- [70] Globus project home page. <http://www.globus.org/>.
- [71] GridLab Resource Management System project home page.
<http://www.gridlab.org/WorkPackages/wp-9/>.
- [72] A. Gualandris, S. P. Zwart, and A. Tirado-Ramos. Performance Analysis of Direct N-body Algorithms on Highly Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*. Submitted, available at:
<http://arxiv.org/abs/astro-ph/0412206>.
- [73] Guaranteeing Quality of Service. http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/qos.htm.
- [74] H. Hao. What is Service-Oriented Architecture?, September 2003.
<http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html>.
- [75] D. Harel. Statecharts: A Visual Formalism for Complex Systems. *Sci. Comput. Programming*, 8:231–274, 1987.
- [76] HDF5 project home page. <http://hdf.ncsa.uiuc.edu/HDF5/>.

- [77] High Level Architecture Run-Time Infrastructure RTI 1.3-Next Generation Programmer's Guide. <https://www.dmsomil/public/transition/hla/rti/>.
- [78] High Level Architecture specification. <http://www.sisostds.org/stdsdev/hla/>.
- [79] L. Hluchy, V. Tran, B. Simo, O. Habala, J. Astalos, and E. Gatial. Flood Forecasting in CrossGrid project. In M. Dikaiakos, editor, *Grid Computing, 2nd European Across Grids Conference*, volume 3165 of *Lecture Notes in Computer Science*, pages 51–60, Nicosia, Cyprus, 2004. Springer.
- [80] J. Huang, M.-C. Tung, L. Hui, and M.-C. Lee. An Approach for an Unified Time Management Mechanism for HLA. *SIMULATION*, 81(1):67–76, 2005.
- [81] P. Hut and J. Makino. The Art of Computational Science. <http://www.artcompsci.org/>.
- [82] P. Hut, J. Makino, and S. McMillan. Building a Better Leapfrog. *Astrophysical Journal Letters*, 443:93–96, 1995.
- [83] M. V. Iordache and P. J. Antsaklis. Generalized Conditions for Liveness Enforcement and Deadlock Prevention in Petri Nets. In *ICATPN '01: Proceedings of the 22nd International Conference on Application and Theory of Petri Nets*, pages 184–203, London, UK, 2001. Springer-Verlag.
- [84] K. Iskra. *Time Warp – from Cluster to Grid*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, June 2005. Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Dr. G.D. van Albada.
- [85] K. A. Iskra, R. G. Belleman, G. D. van Albada, J. Santoso, P. M. A. Sloot, H. E. Bal, H. J. W. Spoelder, and M. Bubak. The Polder Computing Environment, a System for Interactive Distributed Simulation. *Concurrency and Computation: Practice and Experience*, 14:1313–1335, 2002. Special Issue on Grid Computing Environments.
- [86] J. Lütchi and S. Grossman. FT-RSS: A Flexible Framework for Fault Tolerant HLA Federations. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 865–872, Kraków, Poland, June 2004. Springer-Verlag.
- [87] Java Management Extension (JMX) project homepage. <http://java.sun.com/products/JavaManagement/>.
- [88] Java Native Interface. <http://java.sun.com/docs/books/tutorial/nativel1/>.

- [89] K. J. Joshi, F. A. Rasio, and S. Portegies Zwart. Monte Carlo Simulations of Globular Cluster Evolution. I. Method and Test Calculations. *ApJ*, 540:969–982, Sept. 2000.
- [90] JSIM project home page. <http://chief.cs.uga.edu/~jam/jsim/>.
- [91] Information about KLOT simulation. <http://www.pitch.se/klots/default.asp>.
- [92] J. Kohl and P. Papadopoulos. Cumulvs User's Guide Computational Steering And Interactive Visualization In Distributed Applications. <http://www.virtc.com/Products/>.
- [93] S. Korn, G. R. Burns, and D. K. Harrison. The Application of Multiparadigm Simulation Techniques to Manufacturing Processes. *International Journal of Advanced Manufacturing Technology*, pages 869–875, 1999.
- [94] D. Kranzlmuller, G. Kurka, P. Heinzlreiter, and J. Volkert. A Grid Middleware Extension for Scientific Visualization. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, PDPTA '02*, volume 2. CSREA Press, June 2002.
- [95] V. V. Krzhizhanovskaya, P. M. A. Sloot, and Y. E. Gorbachev. Grid-Based Simulation of Industrial Thin-Film Production. *SIMULATION*, 81(1):77–85, 2005.
- [96] 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.
- [97] J. S. Lee, T. Luu, and V. K. Konangi. Design of a Satellite Cluster System in Distributed Simulation. *SIMULATION*, 81(1):57–66, 2005.
- [98] S. Li and B. Song. Normalized Workflow Net (NWF-net): Its Definition and Properties. *Future Generation Computer Systems*, 21(7):1004–1014, July 2005.
- [99] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, June 1988.
- [100] H. Liu, L. Jiang, M. Parashar, and D. Silver. Rule-based Visualization in the Discover Computational Steering Collaboratory. *Future Generation Computer Systems*, 21(1):53 – 59, January 2005.
- [101] P. D. Louis and R. Spurzem. Anisotropic Gaseous Models for the Evolution of Star Clusters. *MNRAS*, 251:408–425, August 1991.

- [102] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows*. To appear.
- [103] T. Ludwig, R. Wismüller, V. Sunderam, and A. Bode. OMIS — On-line Monitoring Interface Specification (Version 2.0). Technical Report TUM-I9733, SFB-Bericht Nr. 342/22/97 A, Technische Universität München, July 1997. <http://wwwbode.in.tum.de/~omis/>.
- [104] B. Ławniczek, G. Majka, P. Słowikowski, K. Zieliński and S. Zieliński. Grid Infrastructure Monitoring Service Framework Jiro/JMX Based Implementation. *Electronic Notes in Theoretical Computer Science*, 82(6), 2003.
- [105] Magnetar Games Corporation. <http://www.magnetargames.com/>.
- [106] MÄK Trainers – Desktop Training Simulations. <http://www.mak.com/>.
- [107] J. Makino and S. Aarseth. On a Hermite Integrator with Ahmad-Cohen Scheme for Gravitational Many-body Problems. *PASJ*, 44:141–151, April 1992.
- [108] 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.
- [109] M. A. Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley and Sons, 1995.
- [110] C. McLean, F. Riddick, and Y. T. Lee. An Architecture and Interfaces for Distributed Manufacturing Simulation. *SIMULATION*, 81(1):15–32, 2005.
- [111] H. Mizuta and Y. Yamagata. Transaction Cycle of Agents and Web-based Gaming Simulation for International Emissions Trading. In E. Yucesan, C.-H. Chen, S. L. Snowden, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 801–806. Piscataway, NJ: IEEE, 2002.
- [112] D. Moen, J. Pullen, and F. Zhao. Implementation of Host-Based Overlay Multicast to Support of Web Based Services for RT-DVS. In *8th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2004), 21-23 October 2004, Budapest, Hungary*, pages 4–11. IEEE Computer Society, 2004.
- [113] H. H. Mohamed and D. H. J. Epema. The design and implementation of the koala co-allocating grid scheduler. In P. M. A. Sloot, A. G. Hoekstra, T. Priol, A. Reinefeld, and M. Bubak, editors, *Advances in Grid Computing - EGC 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 640–650, Heidelberg, February 2005. Springer-Verlag.

- [114] K. Montgomery, M. Stephanides, S. Schendel, and M. Ross. A Case Study Using the Virtual Environment for Reconstructive Surgery. In *VIS '98: Proceedings of the conference on Visualization '98*, pages 431–434, Los Alamitos, CA, USA, 1998. IEEE Computer Society Press.
- [115] MPI Forum home page. <http://www.mpi-forum.org>.
- [116] MPICH-G home page. <http://www.niu3.edu/mpi>.
- [117] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4):541–580, Apr. 1989.
- [118] B. Murphy, H. Cohn, and R. Durisen. Dynamical and Luminosity Evolution of Active Galactic Nuclei - Models with a Mass Spectrum. *ApJ*, 370:66–77, March 1991.
- [119] B. Nichols, D. Buttler, and J. Farrell. *Pthreads Programming A POSIX Standard for Better Multiprocessing*. O'Reilly, 1996.
- [120] T. Oinn, M. Greenwood, M. Addis, M. N. Alpdemir, J. Ferris, K. Glover, C. Goble, A. Goderis, D. Hull, D. Marvin, P. Li, P. Lord, M. R. Pocock, M. Senger, R. Stevens, A. Wipat, and C. Wroe. Taverna: Lessons in creating a workflow environment for the life sciences. *Concurrency and Computation: Practice and Experience, Special Issue on Scientific Workflows*. To appear.
- [121] OMG Model Driven Architecture home page. <http://www.omg.org/mda/>.
- [122] OpenGL standard. <http://www.opengl.org/>.
- [123] Open HLA Project home page. <http://sourceforge.net/projects/ohla>.
- [124] OpenMP project home page. <http://www.openmp.org/>.
- [125] Openskies technology. <http://www.openskies.net/>.
- [126] B. Overeinder. *Distributed Event-driven Simulation - Scheduling Strategies and Resource Management*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, November 2000. Promotor: Prof. Dr. P.M.A. Sloot, Copromotor: Prof. Dr. M. Livny.
- [127] R. J. Paul and S. J. E. Taylor. What Use is Model Reuse: Is There a Crook at the End of the Rainbow? In E. Yucesan, C.-H. Chen, S. L. Snowden, and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 648–652. Piscataway, NJ: IEEE, 2002.
- [128] C. A. Petri. *Kommunikation mit Automaten*. Bonn: Institut für Instrumentelle Mathematik, Schriften des IIM Nr. 2, 1962.

- [129] S. Pieper, J. Rosen, and D. Zeltzer. Interactive Graphics for Plastic Surgery: a Task-level Analysis and Implementation. In *SI3D '92: Proceedings of the 1992 symposium on Interactive 3D graphics*, pages 127–134, New York, NY, USA, 1992. ACM Press.
- [130] Platform Independent Petri Net Editor. <http://pipe2.sourceforge.net/>.
- [131] S. Portegies Zwart, H. Baumgardt, P. Hut, J. Makino, and S. L. W. Mcmillan. Formation of Massive Black Holes through Runaway Collisions in Dense Young Star Clusters. *Nature*, 428:724–726, April 2004.
- [132] Private Communication with Duncan Suttles, President of Magnetar Games Corporation.
- [133] ProActive project homepage. <http://www-sop.inria.fr/oasis/ProActive/>.
- [134] J. M. Pullen, R. Brunton, D. Brutzman, D. Drake, M. Hieb, K. L. Morse, and A. Tolk. Using Web Services to Integrate Heterogeneous Simulations in a Grid Environment. *Future Generation Computer Systems*, 21(1):97–106, January 2005.
- [135] Parallel Virtual Machine home page. http://www.csm.ornl.gov/pvm/pvm_home.html.
- [136] S. Rathmayer and M. Lenke. A Tool for On-line Visualization and Interactive Steering of Parallel HPC Applications. http://wwwbode.cs.tum.edu/Par/appls/res-A/ovid_e.html.
- [137] RealityGrid Project home page. <http://www.realitygrid.org/>.
- [138] S. Robinson. Distributed Simulation and Simulation Practice. *SIMULATION*, 81(1):5–13, 2005.
- [139] RSL – Globus Resource Specification Language v1.0. http://www.globus.org/gram/rsl_spec1.html.
- [140] RTI Verification Status Board. <https://www.dmsomil/public/transition/hla/rti/statusboard>.
- [141] K. Rycerz, B. Baliś, R. Szymacha, M. Bubak, and P. M. A. Sloot. Monitoring of HLA Grid Application Federates with OCM-G. In S. J. Turner, D. Roberts, and L. Wilson, editors, *DS-RT '04: Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 125–132, Washington, DC, USA, 2004. IEEE Computer Society.

- [142] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Interactive Grid Computing: Adapting High Level Architecture-based Applications to the Grid. *TASK QUARTERLY Scientific Bulletin of Academic Centre in Gdansk*, 8(4):549–559, 2004.
- [143] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Support for Effective and Fault Tolerant Execution of HLA based Applications in the OGSA Framework. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 875–882, Krakow, Poland, June 2004. Springer-Verlag.
- [144] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Framework for HLA-Based Interactive Simulations on the Grid. *SIMULATION*, 81(1):67–76, 2005.
- [145] K. Rycerz, A. Tirado-Ramos, A. Gualandris, S. P. Zwart, M. Bubak, and P. M. A. Sloot. N-Body simulations on the Grid: HLA versus MPI. submitted to *Journal of High Performance Computing Applications*.
- [146] Speech, CAVE and Vtk Interaction (SCAVI) project home page. <http://staff.science.uva.nl/~robbel/SCAVI/>.
- [147] S. Schamberger and J. Wierum. Graph Partitioning in Scientific Simulations: Multilevel Schemes vs. Space-Filling Curves. In *Proceedings of the International Conference on Parallel Computing Technologies*, volume 2763 of *Lecture Notes in Computer Science*, pages 167–179. Springer, 2003.
- [148] A. Schreiber, T. Metsch, and H.-P. Kersken. A Problem Solving Environment for Multidisciplinary Coupled Simulations in Computational Grids. *Future Generation Computer Systems*, 21(6):942–952, June 2005.
- [149] M. Selvakkumaran, A. Ranjan, S. Raje, and G. Karypis. Multi-Resource Aware Partitioning Algorithms for FPGAs with Heterogeneous Resources. In *41st Design Automation Conference (DAC04) June 7-14*, pages 741–746, 2004.
- [150] Semantic Grid Community Portal. <http://www.semanticgrid.org/>.
- [151] Silver project home page. <http://supercluster.org/projects/silver/index.shtml>.
- [152] Simulation Interoperability Standards Organization's (SISO). <http://www.sisostds.org/>.
- [153] Information about SJUL simulation. <http://www.pitch.se/defense/default.asp>.
- [154] D. Skillicorn. Motivating Computational Grids. Technical report, Department of Computing and Information Science, Queens University, November 2001.

- [155] P. M. A. Sloot, A. Tirado-Ramos, A. G. Hoekstra, and M. Bubak. Interactive Grid Environment for Non-Invasive Vascular Reconstruction. In *2nd International Workshop on Biomedical Computations on the Grid (BioGrid'04)*, in conjunction with *Fourth IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid2004)*. IEEE, April 2004. (CD-ROM IEEE Catalog # 04EX836C).
- [156] Honeywell SMARTlab Uses MK Tools for Simulation Based Acquisition. http://www.mak.com/profile_honeywell.pdf.
- [157] Simple Object Access Protocol. <http://www.w3.org/tr/soap/>.
- [158] A. J. Soper, C. Walshaw, and M. Cross. A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph Partitioning. *J. Global Optimization*, 29(2):225–241, 2004. (originally published as Univ. Greenwich Tech. Rep. 00/IM/58).
- [159] Starfleet Command II game. <http://www.openskies.net/present/Openskies.htm>.
- [160] G. Stix. The triumph of the light. 284:68–73, January 2001.
- [161] S.Vuong, X. Cai, J. Li, S. Pramanik, D. Suttles, and R. Chen. FedGrid: An HLA Approach to Federating Grids. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004*, volume 3038 of *Lecture Notes in Computer Science*, pages 889–896, Krakow, Poland, June 2004. Springer-Verlag.
- [162] TAO CORBA project home page. <http://www.cs.wustl.edu/~schmidt/TAO.html>.
- [163] I. Taylor, I. Wang, M. Shields, and S. Majithia. Distributed computing with Triana on the Grid. *Concurrency and Computation: Practice and Experience*, 17(1–18), 2005.
- [164] S. J. E. Taylor, S. Robinson, and J. Ladbrook. Towards Collaborative Simulation Modelling: Improving Human-to-human Interaction through Groupware. In D.Al-Dabass, editor, *Proceedings of the 17th European Simulation Multiconference (ESM 2003)*, pages 474–482. SCS, 2003.
- [165] S. J. E. Taylor, S. J. Turner, N. Mustafee, H. Ahlander, and R. Ayani. A Comparison of CMB- and HLA-Based Approaches to Type I Interoperability Reference Model Problems for COTS-Based Distributed Simulation. *SIMULATION*, 81(1):33–43, 2005.
- [166] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing: Making the Global Infrastructure a Reality*, pages 299–335. John Wiley & Sons Inc., December 2002.

- [167] A. Tirado-Ramos, K. Zajac, Z. Zhao, P. M. A. Sloot, D. van Albada, and M. Bubak. Experimental Grid Access for Dynamic Discovery and Data Transfer in Distributed Interactive Simulation Systems. In P. M. A. Sloot and et al., editors, *Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003*, number 2657 in Lecture Notes in Computer Science, pages 207–213. Springer, 2003.
- [168] Universal Description, Discovery and Integration. <http://www.uddi.org>.
- [169] Unified Modeling Language. <http://www.uml.org/>.
- [170] Internet Network Organizations. <http://www.slac.stanford.edu/comp/net/netorg.html>.
- [171] ViaVoice home page. <http://www-3.ibm.com/software/speech/>.
- [172] Virtual Laboratory project home page. <http://www.vl-e.nl/>.
- [173] VISIT project home page. <http://www.fz-juelich.de/zam/visit/>.
- [174] Volumizer software home page. <http://www.sgi.com/software/volumizer>.
- [175] Visualisation Toolkit home page. <http://public.kitware.com/VTK/>.
- [176] L. Wang, S. J. Turner, and F. Wang. Interest Management in Agent-Based Distributed Simulations. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 20–29, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [177] Y.-H. Wang and Y.-C. Liao. Implementation of a Collaborative Web-based Simulation Modeling Environment. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 150–157, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [178] Web Services. <http://www.w3.org/2002/ws/>.
- [179] Web Services Definitions. <http://searchwebservices.techtarget.com/>.
- [180] WS-Resource Framework. <http://www.globus.org/wsrf/>.
- [181] Web Services Description Language. <http://www.w3.org/tr/wsdl/>.
- [182] XDR: External Data Representation Standard, RFC 1832. <http://www.faqs.org/rfcs/rfc1832.html>.
- [183] X11 in Virtual Reality (XiVE) project home page. <http://staff.science.uva.nl/~robbel/XiVE/>.

- [184] EXtensile Modeling Language. <http://www.xml.com/>.
- [185] XMSF project home site. <http://www.movesinstitute.org/xmsf/xmsf.html>.
- [186] XMSF Workshop position paper. <http://www.movesinstitute.org/xmsf/xmsf.html>.
- [187] A. YarKhan and J. J. Dongarra. Biological Sequence Alignment on the Computational Grid Using the GrADS Framework. *Future Generation Computer Systems*, 21(6):980–986, June 2005.
- [188] Z. Yuan, W. Cai, and M. Y. H. Low. A Framework for Executing Parallel Simulation using RTI. In S. J. Turner and S. J. E. Taylor, editors, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 20–28, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [189] E. Yucesan, Y.-C. Luo, C.-H. Chen, and I. Lee. Distributed Web-based Simulation Experiments for Optimization. *Simulation Practice and Theory*, 9:73–90, 2001.
- [190] K. Zajac, A. Tirado-Ramos, Z. Zhao, P. M. A. Sloot, and M. Bubak. Grid Services for HLA-based Distributed Simulation Frameworks. In F. Riviera, M. Bubak, A. G. Tato, and R. Doallo, editors, *Proc. First European Across Grids Conference*, volume 2970 of *Lecture Notes in Computer Science*, pages 147–154, Heidelberg, February 2003. Springer-Verlag.
- [191] K. Zajac, M. Bubak, M. Malawski, and P. M. A. Sloot. Towards a Grid Management System for HLA-Based Interactive Simulations. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 4–11, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [192] Z. Zhao. *An Agent Based Architecture for Constructing Interactive Simulation Systems*. PhD thesis, University of Amsterdam, Amsterdam, The Netherlands, December 2004. Promotor: Prof. Dr. P.M.A. Sloot, Co-promotor: Dr. G.D. van Albada.
- [193] Z. Zhao, G. D. van Albada, A. Tirado-Ramos, K. Zajac, and P. M. A. Sloot. ISS-Studio: a Prototype for a User-friendly Tool for Designing Interactive Experiments in Problem Solving Environments. In P. M. A. Sloot, D. Abrahamson, A. V. Bogdanov, J. J. Dongarra, A. Y. Zomaya, and Y. E. Gorbachev, editors, *Computational Science - ICCS 2003, Melbourne, Australia and St. Petersburg, Russia, Proceedings Part I*, volume 2657 of *Lecture Notes in Computer Science*, pages 679–688. Springer Verlag, June 2003.

-
- [194] W. Zong, Y. Wang, W. Cai, and S. J. Turner. Grid Services and Service Discovery for HLA-Based Distributed Simulation. In *8th IEEE International Workshop on Distributed Simulation and Real-Time Applications (DS-RT 2004), 21-23 October 2004, Budapest, Hungary*, pages 116–124. IEEE Computer Society, 2004.
 - [195] E. V. Zudilova and P. M. A. Sloot. Bringing Combined Interaction to a Problem Solving Environment for Vascular Reconstruction. *Future Generation Computer Systems*, 21(7):1167–1176, July 2005.
 - [196] G. Zulch, U. Jonsson, and J. Fischer. Hierarchical Simulation of Complex Production Systems by Coupling Models. *International Journal of Production Economics*, 77:39–51, 2002.

English Summary

Developing distributed simulations that support runtime steering is a very important and interesting issue in modern computer science. In this thesis, we have tried to show how the modern scientific and technological solutions of computer science can be used for satisfying the distributed interactive simulation requirements more accurately and efficiently than before.

In Chapter 1 we have defined those requirements. First, distributed interactive simulations require synchronization support (time management) for event-driven and time-driven scenarios and the ability to connect simulations with different scenarios into one system. Next, they require efficient data distribution management and scalability with a number of distributed components. Additionally, they are often distributed using functional decomposition - each of their elements performs different functions and requires separated (specific) computer resources i.e. databases, visualization devices or computational power. Last but not least, this kinds of applications require an efficient environment, so that they can execute in near-real time with a user in the loop.

Our research presented in this thesis focuses on the requirements mentioned above, which are a starting point for the analysis of both the advantages and shortcomings of current scientific and technological solutions presented in the Chapter 2. We claim that High Level Architecture HLA is a good candidate for fulfilling the requirements of synchronization management, data distribution support, scalability and the ability to connect time- and even-driven simulations into one system. Grid solutions are - by definition - a promising approach to accessing distributed resources. However, there still remains the issue of execution environment efficiency. The HLA standard was developed assuming a certain quality of service in the underlying environment of the simulation execution. The Grid environment, however, is shared by many users and its conditions can change in an unpredictable way. Therefore, there is a need for a system that can adapt applications to this dynamically-changing environment and allows for their efficient execution.

In Chapter 3, we have proposed a Service Oriented Architecture of the Grid HLA Management System (G-HLAM) system that fills the gap described above. The presented approach allows not only for building HLA-based applications that can be executed on the Grid, but also for easy adaptation of HLA legacy code. In G-HLAM system, we have introduced the *Broker Service* which uses the *Registry Service* to setup HLA-based applications from the *RTIExec Service*, responsible for management of the RTIExec process, and from *HLA-Speaking Services*, interfacing actual HLA federates with G-HLAM. Also, G-HLAM includes *Monitoring Services* for monitoring HLA application performance, *Benchmark Services* for checking how such application will behave on the Grid and *Migration Service* for migration support in the case the performance is not satisfactory. In this Chapter, we have also presented Petri Net based analysis of the system to proof that it behaves correctly and that deadlock does not occur.

In the subsequent Chapters we describe essential parts of the G-HLAM.

In Chapter 4 we have described *HLA-Speaking Service* that interfaces actual HLA application with G-HLAM. Two different versions of the *HLA-Speaking Service* are presented: a prototype suitable for management of federates in one process and an extension, managing multiple processes. *HLA-Speaking Services* are solution to the problem of turning user application federates into Grid services.

In order to provide a certain quality of the execution environment, we have decided to build a migration mechanism to allow federates to migrate when their performance is not satisfactory, which is described in Chapter 5. The *Migration Service* is responsible for migration of HLA-connected components of the distributed application in the Grid environment. We have proposed user-level checkpointing implemented through routines of the GridHLAController runtime support library. The library conceals details of RTI internal state checkpointing and provides a user-friendly interface to save and restore all the necessary data.

Efficient migration of HLA-based simulations requires a means of acquiring information about application performance. In Chapter 6, we have presented an HLA-based benchmark suite that helps the *Broker Service* decide where the application components should be placed. We have also presented the *Application Monitoring Service* which monitors the performance of HLA-based applications to decide when to migrate their parts to achieve more efficient execution. We have chosen the Grid enabled OMIS Compliant Monitor (OCM-G) system as a monitoring system, since it was designed to be executed in a Grid environment and complies with the Online Monitoring Interface Specification (OMIS). We have shown how the design concepts of the OCM-G enable easy adaptation to monitoring of HLA, C++-based applications.

In Chapter 7 we have described implementation and performance results. We have presented analytical models of migration and show how the experimental results fit into this model. The results from experiments concerning benchmarking and monitoring are also included.

Finally, we have shown that two sample realistic simulations can benefit from the Grid using the solutions proposed in this thesis. The chosen applications allows to show how G-HLAM deals with their different types of interactions and distribution.

In Chapter 8 we have presented an N-body simulation as the example using G-HLAM running on the Grid. It shows how G-HLAM deals with applications with different types of interactions including passive and active exploration of results as well as rollback to the previous simulation steps for the user demand. We have shown how the application can be run within G-HLAM. Subsequently, we have presented results from three different experiments showing how our system can improve performance of interactive simulations.

In Chapter 9 we have also shown the advantages of using the G-HLAM for running a large-scale HLA-based medical application supporting surgical pre-operative planning. We have described how G-HLAM deals with various types of its distribution (a single simulation for a single user, multiple simulations for a single user, collaborative environment for many users). We especially focused on its execution in a collaborative environment for many users. Our experiments show that G-HLAM is useful also for improving performance of such applications.

In this thesis we have shown that Grid environments can be beneficial for distributed interactive simulations by providing them with new opportunities not available beforehand. We have also shown that the HLA is an appropriate standard for interactive distributed simulations, when there is a need to run them on the Grid. The presented design, implementation and feasibility study of a G-HLAM system will help distributed interactive simulation developers efficiently apply the concept of Grid computing.

Nederlandse samenvatting

Het ontwikkelen van gedistribueerde simulaties die tijdens de executie bijgestuurd kunnen worden is een zeer belangrijk en interessant probleem in de moderne informaticawetenschap. In dit proefschrift hebben we geprobeerd aan te tonen hoe moderne wetenschappelijke en technologische informaticaoplossingen toegepast kunnen worden om accurater en efficiënter te kunnen voldoen aan de eisen van gedistribueerde interactieve simulaties.

In Hoofdstuk 1 worden deze eisen gedefinieerd. Ten eerste behoeven gedistribueerde interactieve simulaties ondersteuning voor synchronisatie (tijd management) voor zowel event- als tijd-gedreven scenarios, en moeten ze in staat zijn simulaties onder verschillende scenarios met elkaar te verbinden in een enkel systeem. Ten tweede vereisen gedistribueerde interactieve simulaties ondersteuning voor efficiënte gegevensdistributie en schaalbaarheid in het aantal gedistribueerde componenten. Ten derde moeten interactieve simulaties vaak gedistribueerd worden door middel van functionele decompositie - elk van de onderdelen vervult een andere functie en vereist inzet van zeer verschillende computersystemen, zoals databases, visualisatie-apparatuur, of systemen die pure rekenkracht leveren. Tot slot vereisen dergelijke applicaties een zeer efficiënte executie-omgeving, zodanig dat ze (bijna) real-time kunnen draaien en gebruikers bovendien in staat stellen de simulatie bij te sturen.

Het in dit proefschrift gepresenteerde onderzoek richt zich op de bovengenoemde eisen. De eisen zijn het startpunt voor de in Hoofdstuk 2 gepresenteerde analyse van de voor- en nadelen van bestaande wetenschappelijke en technologische oplossingen. Wij stellen dat *High Level Architecture* (HLA) goed kan voldoen aan de eisen van ondersteuning voor synchronisatie en gegevensdistributie, schaalbaarheid, en het in een enkel systeem verbinden van tijd- en event-gedreven simulaties. Grid oplossingen zijn - per definitie - een veelbelovende manier om gedistribueerde systemen aan te wenden. Echter, het probleem van een efficiënte executie-omgeving is daarmee niet van de baan. De HLA standaard is ontwikkeld onder de aanname van een zekere kwaliteit van serviceverlening van de onderliggende executie-omgeving. Een Grid

omgeving wordt echter gedeeld door vele gebruikers, en de omstandigheden daarin kunnen op een zeer onvoorspelbaren manier veranderen. Daarom is het noodzakelijk een systeem te ontwikkelen dat applicaties aan deze dynamisch veranderende omgeving kan aanpassen om efficiëntere executie te bewerkstelligen.

In Hoofdstuk 3 stellen we een service-georiënteerde architectuur van het *Grid HLA Management System (G-HLAM)* voor dat het bovengenoemde gat opvult. De gepresenteerde aanpak maakt het niet alleen mogelijk HLA-gebaseerde applicaties te ontwikkelen die op een Grid gedraaid kunnen worden, maar vereenvoudigd eveneens de conversie van oude HLA applicaties. In het G-HLAM systeem introduceren we de *Broker Service* die de *Registry Service* gebruikt om HLA-gebaseerde applicaties door middel van de *RTIExec Service* en *HLA-Speaking Services* te initiëren. G-HLAM omvat daarnaast *Monitoring Services* om de performance van HLA applicaties in de gaten te houden, *Benchmarking Services* om te bepalen hoe dergelijke applicaties zich zullen gedragen op een Grid, en *Migration Services* die migratie van functionaliteit bewerkstelligt als de behaalde performance niet toereikend is. In dit hoofdstuk presenteren we eveneens een Petri Net-gebaseerde analyse van het systeem om aan te tonen dat het zich correct gedraagt en dat deadlocks niet kunnen voorkomen.

In de volgende hoofdstukken beschrijven we de essentiële onderdelen van het G-HLAM systeem.

In Hoofdstuk 4 beschrijven we de *HLA-Speaking Service* die een HLA applicatie feitelijk verbindt met G-HLAM. We presenteren twee verschillende versies van de *HLA-Speaking Service*: een prototype die geschikt is voor het onderhouden van HLA instanties in een enkel proces, en een uitgebreide versie die meerdere processen onderhoudt. *HLA-Speaking Services* maken het dehalve mogelijk bestaande applicaties om te vormen tot Grid services.

Om een bepaalde kwaliteit van de executie-omgeving te bewerkstelligen, hebben we, zoals beschreven in Hoofdstuk 5, een mechanisme ontwikkeld dat HLA instanties in staat stelt te migreren als de behaalde performance niet toereikend is. De *Migration Service* is verantwoordelijk voor de migratie van HLA-verbonden componenten van de gedistribueerde simulatie draaiend in een Grid omgeving. We stellen *user-level checkpointing* voor, geïmplementeerd met behulp van routines die onderdeel zijn van de *GridHLAController runtime support library*. Deze bibliotheek verbergt de details van *RTI internal state checkpointing* en biedt een gebruikersvriendelijke interface voor het opslaan en terughalen van alle noodzakelijke gegevens.

Efficiënte migratie van HLA-gebaseerde simulaties vereist een manier om informatie te verkrijgen omtrent de performance van applicaties. In Hoofdstuk 6 presenteren we een HLA-gebaseerde benchmark suite die de *Broker Service* helpt bepalen waar de applicatiecomponenten geplaatst moeten worden. We presenteren daarnaast ook de *Application Monitoring Service* die de performance van HLA-gebaseerde applicaties in de gaten houdt om te bepalen op welk moment onderdelen gemigreerd moeten worden. We hebben het *Grid-enabled OMIS Compliant Monitor (OCM-G)* systeem uitgekozen als monitoring systeem, omdat het is ontwikkeld voor Grid omgevingen en overeenkomt met de *Online Monitoring Interface Specification (OMIS)*. We tonen tevens aan hoe de ontwerpprincipes van OCM-G het monitoren van in C++

geschreven HLA-gebaseerde applicaties mogelijk maakt.

In Hoofdstuk 7 beschrijven we implementatie- en performanceresultaten. We presenteren analytische migratiemodellen en geven aan hoe experimentele resultaten in deze modellen passen. Resultaten van experimenten die betrekking hebben op benchmarking en monitoring worden eveneens gegeven.

Tot slot geven we aan dat twee realistische voorbeeldapplicaties in interactieve simulatie baat kunnen hebben bij Grid gebruik, gegeven de in dit proefschrift voorgestelde oplossingen. De gekozen applicaties maken duidelijk hoe G-HLAM omgaat met de verschillende vormen van interactie en distributie.

In Hoofdstuk 8 presenteren we een N-body simulatie ter voorbeeld van G-HLAM executie in een Grid omgeving. Ten eerste laten we zien hoe de applicatie gedraaid kan worden met behulp van G-HLAM. Daarnaast geven we resultaten voor drie verschillende experimenten die aangeven hoe ons systeem de performance van interactieve simulaties kan verbeteren.

In Hoofdstuk 9 laten we de voordelen zien van het gebruik van G-HLAM voor het draaien van HLA-gebaseerde medische applicaties die chirurgische pre-operatieve planning ondersteunen. We laten zien hoe G-HLAM omgaat met de verschillende vormen van distributie (een enkele simulatie voor een enkele gebruiker, meerdere simulaties voor een enkele gebruiker, en een collaboratieve omgeving voor meerdere gebruikers). We richten ons voornamelijk op de de executie in een collaboratieve omgeving voor meerdere gebruikers. Onze experimenten tonen aan dat G-HLAM eveneens geschikt is voor het verbeteren van de performance van dergelijke applicaties.

In dit proefschrift hebben we aangetoond dat Grid omgevingen geschikt kunnen zijn voor gedistribueerde interactieve simulaties door het aanbieden van voorheen niet bestaande technieken. We hebben eveneens aangetoond dat HLA een geschikte standaard is voor gedistribueerde interactieve simulaties die in een Grid omgeving uitgevoerd moeten worden. Het gepresenteerde ontwerp, alsmede de implementatie en de bijbehorende evaluatie van het G-HLAM systeem, zal een bijdrage kunnen leveren in het efficiënt toepassen van Grid computing in gedistribueerde interactieve simulaties.

Translated by Frank Seinstra

Streszczenie po polsku

System wspomagający wykonywanie symulacji HLA w środowisku gridowym

Jednym z ważniejszych zagadnień współczesnej informatyki jest tworzenie rozproszonych interaktywnych programów symulacyjnych, którymi można sterować w czasie wykonania. W pracy podjęto próbę pokazania w jaki sposób metody i narzędzia informatyki – zarówno naukowe, jak i technologiczne – mogą zostać użyte w celu efektywnej realizacji programów symulacyjnych HLA w środowisku gridowym.

W rozdziale 1 zdefiniowano wymagania aplikacji symulacyjnych: zaawansowane techniki synchronizacji (tzw. zarządzanie czasem) zarówno dla symulacji sterowanej czasem ciągłym, jak i zdarzeniami, możliwość połączenia różnych typów symulacji w jeden system, efektywne zarządzanie nierzadko rozproszonymi danymi i ich przesyłaniem, skalowalność wraz z ilością rozproszonych elementów. Gdy każdy z elementów symulacji rozproszonej ma do spełnienia inną funkcję, wymaga odrębnych i specyficznych zasobów – np. szybkiego dostępu do bazy danych, specjalistycznego sprzętu wizualizacyjnego czy też dużej mocy obliczeniowej. Omawiane aplikacje potrzebują również efektywnego środowiska pozwalającego na ich wykonanie w czasie bliskim do rzeczywistego, dogodnym dla użytkownika znajdującego się w pętli obliczeniowej.

Koncentrując się na wymienionych wymaganiach, przeanalizowano istniejące rozwiązania naukowe i technologiczne. Analiza ta została przedstawiona w rozdziale 2. W wyniku tej analizy, stwierdzono też, że standard High Level Architecture (HLA) dobrze spełnia wymogi zarządzania czasem i przesyłania danych, skalowalności, jak również możliwości połączenia symulacji różnych typów w jeden system rozproszony. Pokazano, że rozwiązania gridowe są obiecującym podejściem mającym na celu zwiększenie dostępu do rozproszonych zasobów. Nawet przy zastosowaniu tych rozwiązań ważnym problemem badawczym jest uzyskanie wysokiej efektywności. Standard HLA powstał przy założeniu, że środowisko dla danej aplikacji zapewnia jakość usług i jej

efektywne wykonanie. Zasoby gridu są współdzielone pomiędzy bardzo wielu użytkowników i warunki ich funkcjonowania mogą się zmieniać w sposób trudno przewidywalny i dlatego też potrzebny jest system wspomagający adaptację rozproszonych interaktywnych aplikacji symulacyjnych do tego dynamicznie zmieniającego się środowiska zapewniający ich efektywne wykonanie.

W rozdziale 3 zaproponowano system zarządzania symulacjami HLA realizowanymi na gridzie – Grid HLA Management System (G-HLAM) – spełniający przedstawione powyżej wymagania. Architektura systemu oparta jest na serwisach – Service Oriented Architecture (SOA). Przedstawione rozwiązanie jest możliwe do wykorzystania nie tylko dla tworzenia nowych aplikacji HLA, lecz pozwala ono również na łatwe dostosowanie aplikacji już istniejących do efektywnego działania na gridzie. Architektura systemu G-HLAM zawiera serwis typu broker (*Broker Service*) który używa serwisu rejestru (*Registry Service*) w celu wyboru serwisów obsługujących pojedyncze elementy aplikacji zwanych wg terminologii HLA federatami (federate). Wybrane poprzez brokera serwisy obsługują zarówno proces koordynujący aplikacje HLA (o nazwie RTIExec) - do tego służy *RTIExec Service*, jak również poszczególnych federatów aplikacji - do tego służy *HLA-Speaking Services*. Te ostatnie stanowią interfejs pomiędzy faktycznymi procesami federatów a systemem G-HLAM. Ponadto G-HLAM zawiera serwisy do monitorowania aplikacji HLA (*Monitoring Services*), serwisy służące do testowania wydajności danej konfiguracji środowiska gridowego (*Benchmark Services*) oraz serwisy odpowiedzialne za bezpośrednie przeprowadzenie migracji (*Migration Services*). W rozdziale 3 przedstawiono także analizę systemu w oparciu o formalizm sieci Petriego sprawdzającą poprawność działania systemu i brak zakleszczeń.

W następnych rozdziałach opisano najważniejsze elementy systemu G-HLAM. I tak w rozdziale 4 jest opisany *HLA-Speaking Service*, który jest interfejsem pomiędzy faktycznymi elementami rozproszonej symulacji a systemem G-HLAM. Przedstawiono dwie wersje tego serwisu: prototyp zarządzający federatami w jednym procesie oraz rozszerzenie zarządzające wieloma procesami. Omawiane rozwiązanie jest propozycją sposobu reprezentacji federatów HLA poprzez serwisy gridowe.

Stworzono mechanizm migracji umożliwiający federatom, których wydajność spada, zmianę miejsca wykonania. Mechanizm opisano szczegółowo w rozdziale 5. Migracja elementów aplikacji HLA w środowisku gridowym jest dokonywana przez *Migration Service*. Zaproponowano mechanizm zapisu stanu aplikacji na poziomie użytkownika. W tym celu zaprojektowano bibliotekę o nazwie *GridHLAController library (GHCL)*, która ukrywa szczegóły zapisu wewnętrznego stanu infrastruktury komunikacyjnej HLA (Runtime Infrastructure – RTI) i udostępnia przyjazny dla użytkownika interfejs do zapisu i odczytu danych.

Efektywna migracja symulacji HLA wymaga informacji o wydajności tych symulacji. W rozdziale 6 zaprezentowano serwisy służące do testowania wydajności danej konfiguracji środowiska gridowego (*Benchmark Services*), które mogłyby być pomocne dla brokera w podejmowaniu decyzji o tym, gdzie dany element symulacji mógłby przemiegrować. Ponadto przedstawiono tam serwis służący do monitorowania symulacji, tak, aby można było zdecydować kiedy wydajność spada i należy przeprowadzić

migrację do lepszego miejsca. Do tego celu zaadaptowano system monitorujący o nazwie Grid-enabled OMIS Compliant Monitor (OCM-G), dostosowany do działania w środowisku gridowym i zgodny ze standardem Online Monitoring Interface Specification (OMIS). Pokazano też, jak zaadaptować system OCM-G do monitorowania aplikacji opartych na standardzie HLA napisanych w C++.

W rozdziale 7, po opisie implementacji, przedstawiono analizę wydajności systemu oraz opracowano analityczne modele migracji i wykorzystano wyniki pomiarów do kalibracji tych modeli. Opisano również wyniki z eksperymentów dotyczących benchmarków i monitorowania.

W następnych rozdziałach przedstawiono dwie przykładowe aplikacje, które dzięki propozycjom przedstawionym w tej pracy, mogą efektywnie wykorzystywać technologie gridowe. Przykłady te ilustrują wspomaganie aplikacji z różnymi typami rozproszenia oraz interaktywności przez system G-HLAM.

W rozdziale 8 przedstawiono symulację metodą wielu ciał jako przykład użycia systemu G-HLAM na gridzie dla aplikacji o różnych typach interaktywności, takich jak pasywne oglądanie wyników symulacji napływających w trakcie jej działania, aktywna eksploracja tych wyników, jak również cofanie stanu symulacji do poprzedniego kroku na żądanie użytkownika. Omówiono tu wyniki trzech różnych eksperymentów pokazujące w jaki sposób system G-HLAM może poprawić efektywność symulacji interaktywnej.

W rozdziale 9 pokazano zalety użycia systemu G-HLAM dla aplikacji dużej skali opartej o standard HLA - jest to aplikacja medyczna mająca na celu wspomaganie planowania operacji chirurgicznych układu krążenia. Opisano, w jaki sposób G-HLAM radzi sobie z aplikacją o różnych typach rozproszenia (jedna symulacja dla jednego użytkownika, wiele symulacji dla jednego użytkownika, środowisko dla wielu użytkowników). Skoncentrowano się w szczególności na badaniu środowiska dla wielu użytkowników. Przedstawione eksperymenty pokazują, że G-HLAM może być użyty do znaczącej poprawy wydajności również takiej aplikacji.

W przedstawionej pracy pokazano, że środowiska gridowe oferują nowe możliwości dla interaktywnych symulacji rozproszonych oraz że aplikacje HLA dzięki G-HLAM mogą być realizowane na gridzie wykorzystując te możliwości w sposób efektywny.

Publications

Note: during work on this thesis the author's surname changed from Zajac to Rycerz. Some of the papers were published with the old author's name.

Journal publications

- [1] A. Z. Maksymowicz, M. Bubak, K. Zajac, and M. Magdoń. Impact of Mutation and Overhunting on Population Extinction in the Penna Model. *Computer Physics Communications*, 121-122:693, 1999.
- [2] A. Z. Maksymowicz, M. Bubak, K. Zajac, and M. Magdoń. Simulation of Aging with an Extended Penna Model. *Computer Physics Communications*, 121-122:113–115, 1999.
- [3] M. Bubak, M. Malawski, P. Nowakowski, R. Pajak, and K. Rycerz. CrossGrid: An EU Integrated Project. *Foundations of Computing and Decision Sciences*, 29(1-2), 2004.
- [4] M. Bubak, M. Malawski, K. Rycerz, and M. Turała. CrossGrid - Tools and Services for Interactive Grid Applications. In P. Doerffer and J. Rybicki, editors, *TASK QUARTERLY. Scientific Bulletin of Academic Centre in Gdansk*, volume 8, pages 503–512. TASK Publishing, 2004.
- [5] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Interactive Grid Computing: Adapting High Level Architecture-based Applications to the Grid. In P. Doerffer and J. Rybicki, editors, *TASK QUARTERLY. Scientific Bulletin of Academic Centre in Gdansk*, volume 8, pages 549–559. TASK Publishing, 2004.
- [6] M. Bubak, T. Gubała, M. Kapałka, M. Malawski, and K. Rycerz. Workflow Composer and Service Registry for Grid Applications. *Future Generation Computer Systems*, 21(1):79–86, 2005.

- [7] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Framework for HLA-Based Interactive Simulations on the Grid. *SIMULATION*, 81(1):67–76, 2005.
- [8] K. Rycerz, A. Tirado-Ramos, A. Gualandris, S. Portegies Zwart, M. Bubak, and P. M. A. Sloot. N-Body simulations on the Grid: HLA versus MPI. Submitted to Journal of High Performance Computing Applications.

Book chapters

- [1] P. Brezany, M. Bubak, P. Łuszczek, M. Malawski, and K. Zajac. A Runtime Support for Large-Scale Irregular Computing on Clusters and Grids. In Y. C. Kwong, editor, *Annual Review of Scalable Computing*, volume 5 of *Series on Scalable Computing*, chapter 2, pages 30–64. Singapore University Press and World Scientific, 2003.
- [2] M. Bubak, M. Malawski, G. Młynarczyk, P. Nowakowski, R. Pająk, M. Turała, and K. Zajac. Software Engineering in the CrossGrid Project. In Z. Huzar and M. Mazur, editors, *Topics and Methods of Software Engineering, Chapter XL*, pages 597–611. WNT, 2003. (in Polish).

Publications in IEEE press

- [1] K. Zajac, M. Bubak, M. Malawski, and P. M. A. Sloot. Towards a Grid Management System for HLA-Based Interactive Simulations. In S. J. Turner and S. J. E. Taylor, editor, *Proceedings Seventh IEEE International Symposium on Distributed Simulation and Real Time Applications (DS-RT 2003)*, pages 4–11, Delft, The Netherlands, October 2003. IEEE Computer Society.
- [2] K. Rycerz, B. Baliś, R. Szymacha, M. Bubak, and P. M. A. Sloot. Monitoring of HLA Grid Application Federates with OCM-G. In S.J. Turner, D. Roberts, and L. Wilson, editors, *Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 125–132, Budapest, Hungary, October 2004. IEEE Computer Society.

Publications in Springer Lecture Notes in Computer Science

- [1] P. Brezany, M. Bubak, M. Malawski, and K. Zajac. Advanced Library Support for Irregular and Out-of-Core Parallel Computing. In B. Hertberger, A. G. Hoekstra, and R. Williams, editors, *Proc. Int. Conf. High Performance Computing and Networking, Amsterdam, June 25-27*, number 2110 in Lecture Notes in Computer Science, pages 435–444. Springer, 2001.

-
- [2] P. Brezany, M. Bubak, M. Malawski, and K. Zając. Irregular and Out-of-Core Computing on Clusters. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and Waniewski J., editors, *Parallel Processing and Applied Mathematics 4th International Conference*, number 2328 in Lecture Notes in Computer Science, pages 299–306. Springer, 2001.
- [3] P. Brezany, M. Bubak, M. Malawski, and K. Zając. Irregular and Out-of-Core Parallel Computing an Clusters. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waśniewski, editors, *Proc. Int. Conf. PPAM 2001 - The 4th International Conference on Parallel Processing and Applied Mathematics, Nałęczów, September 9-12, 2001, Poland*, number 2328 in Lecture Notes in Computer Science, pages 299–306. Springer, 2002.
- [4] P. Brezany, M. Bubak, M. Malawski, and K. Zając. Large-Scale Scientific Irregular Computing on Clusters and Grids. In P. M. A. Sloot, C. J. Kenneth Tan, J. J. Dongara, and A. G. Hoekstra, editors, *Proc. Int. Conf. Computational Science - ICCS, Amsterdam, April 21-24*, number 2330 in Lecture Notes in Computer Science, pages 484–493. Springer, 2002.
- [5] M. Bubak, M. Malawski, and K. Zając. Towards the CrossGrid Architecture. In D. Kranzlmüller, P. Kascuk, J. Dongarra, and J. Volkert, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface - 9th European PVM/MPI Users' Group Meeting Linz, Austria, September 29-October 2, 2002, Proceedings*, number 2474 in Lecture Notes in Computer Science, pages 16–24. Springer, 2002.
- [6] A. Tirado-Ramos and K. Zając and Z. Zhao and P. M. A. Sloot and D. G. van Albada and M. Bubak. Experimental Grid Access for Dynamic Discovery and Data Transfer in Distributed Interactive Simulation Systems. In P. M. A. Sloot and et al., editors, *Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003*, number 2657 in Lecture Notes in Computer Science, pages 284–292. Springer, 2003.
- [7] M. Bubak, K. Górka, T. Gubała, M. Malawski, and K. Zając. Component-Based System for Grid Application Workflow Composition. In J Dongara, D. Laforenza, and S Orlando, editors, *Proceedings of Recent Advances in Parallel Virtual Machin and Message Passing Interface, 10-th European PVM/MPI User's Group Meeting, Venice, Italy, Sep.29-Oct.2 2003*, number 2840 in Lecture Notes in Computer Science, pages 611–618. Springer, 2003.
- [8] M. Bubak, M. Malawski, and K. Zając. Architecture of the Grid for Interactive Applications. In P. M. A. Sloot and et al., editors, *Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003*, number 2657 in Lecture Notes in Computer Science, pages 207–213. Springer, 2003.
- [9] Z. Zhao, G. D. van Albada, A. Tirado-Ramos, K. Zając, and P. M. A. Sloot. ISS-Studio: A Prototype for a User-Friendly Tool for Designing Interactive Experiments in Problem Solving Environments. In P. M. A. Sloot and et al., editors,

- Proceedings of Computational Science - ICCS 2003, International Conference Melbourne, Australia and St. Petersburg, Russia, June 2003*, number 2657 in Lecture Notes in Computer Science, pages 679–688. Springer, 2003.
- [10] M. Bubak, K. Górka, T. Gubała, M. Malawski, and K. Zajac. Automatic Flow Bulding for Component Grid Applications. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics. 5th International Conference, PPAM 2003, Częstochowa, Poland, September 2003*, number 3019 in Lecture Notes in Computer Science, pages 804–811. Springer, 2004.
- [11] M. Bubak, T. Gubała, M. Kapałka, M. Malawski, and K. Rycerz. Grid Service Registry for Workflow Composition Framework. In M. Bubak, G. D. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004. 4th International Conference, Kraków, Poland, June 2004*, number 3038 in Lecture Notes in Computer Science, pages 34–41. Springer, 2004.
- [12] M. Bubak, M. Malawski, G. Młynarczyk, P. Nowakowski, R. Pająk, K. Rycerz, and M. Turafa. Software Engineering in the EU CrossGrid Project. In M. D. Dikaiakos, editor, *Grid Computing. Second European AcrossGrid Conference, AcGrids 2004, Nicosia, Cyprus, January 2004, Revisited Papers*, number 3166 in Lecture Notes in Computer Science, pages 169–178. Springer, 2004.
- [13] M. Bubak, M. Malawski, and K. Zajac. The CrossGrid Architecture: Applications, Tools, and Grid Services. In F. F. Rivera and et al., editors, *Proceedings of Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 2003*, number 2970 in Lecture Notes in Computer Science, pages 309–316. Springer, 2004.
- [14] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Support for Effective and Fault Tolerant Execution of HLA-Based Applications in the OGSA Framework. In M. Bubak, D. G. van Albada, P. M. A. Sloot, and J. J. Dongarra, editors, *Computational Science - ICCS 2004. 4th International Conference, Kraków, Poland, June 2004*, number 3038 in Lecture Notes in Computer Science, pages 848–855. Springer, 2004.
- [15] T. Szepieniec, M. Radecki, K. Rycerz, M. Bubak, and M. Malawski. Tools and Services for Interactive Applications on the Grid - The CrossGrid Tutorial. In D. Kranzlmüller, P. Kacsuk, and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface. 11th European PM/MPI Users' Group Meeting, Budapest, Hungary, September 2004*, number 3241 in Lecture Notes in Computer Science, pages 14–17. Springer, 2004.
- [16] K. Zajac, M. Bubak, M. Malawski, and P. M. A. Sloot. Execution and Migration Management of HLA-Based Interactive Simulations on the Grid. In R. Wyrzykowski, J. Dongarra, M. Paprzycki, and J. Waśniewski, editors, *Parallel Processing and Applied Mathematics. 5th International Conference, PPAM 2003, Częstochowa, Poland, September 2003*, number 3019 in Lecture Notes in Computer Science, pages 872–879. Springer, 2004.

-
- [17] K. Zajęc, A. Tirado-Ramos, Z. Zhao, P. M. A. Sloot, and M. Bubak. Grid Services for HLA-Based Distributed Simulation Frameworks. In F. F. Rivera and et al., editors, *Proceedings of Grid Computing, First European Across Grids Conference, Santiago de Compostela, Spain, February 2003*, number 2970 in Lecture Notes in Computer Science, pages 147–154. Springer, 2004.
- [18] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. A Grid Service for Management of Multiple HLA Federate Processes. Accepted for Parallel Processing and Applied Mathematics (PPAM) 2005 conference, 11-14 Sep 2005, Poznań, Poland, to be published in Lecture Notes in Computer Science. Springer.

Other conference proceedings

- [1] M. Bubak, P. Łuszczek, M. Malawski, and K. Zajęc. Irregular and Out-of-Core Computing on SGI Origin 2000. In M. Bubak, J. Mościnski, and M. Noga, editors, *Proceedings of 1st SGI Users Conference*, pages 185–193, Cracow, Poland, October 2000. ACC Cyfronet-AGH.
- [2] M. Bubak, M. Malawski, and K. Zajęc. Current Status of the CrossGrid Architecture. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'02 Grid Workshop, December 11-14, 2002, Cracow, Poland*, pages 134–139, Kraków, 2003. ACC Cyfronet UMM.
- [3] S. Fisher, K. Zajęc, T. Bołd, M. Garbacz, T. Szymocha, M. Malawski, and R. Pająk. DataGrid Tutorial Report. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'02 Grid Workshop, December 11-14, 2002, Cracow, Poland*, pages 226–227, Kraków, 2003. ACC Cyfronet UMM.
- [4] M. Malawski, M. Bubak, and K. Zajęc. Integration of the CrossGrid Services into the OGSA Model. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'02 Grid Workshop, December 11-14, 2002, Cracow, Poland*, pages 140–147, Kraków, 2003. ACC Cyfronet UMM.
- [5] K. Zajęc, M. Bubak, M. Malawski, P. M. A. Sloot, A. Tirado-Ramos, and Z. Zhao. A Proposal of Services For Managing Interactive Grid Applications. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'02 Grid Workshop, December 11-14, 2002, Cracow, Poland*, pages 155–163, Kraków, 2003. ACC Cyfronet UMM.
- [6] P. Brezany, M. Bubak, M. Malawski, and K. Rycerz. Porting Irregular and Out-of-Core Computations on Grids (Extended Abstract). In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'03 Grid Workshop, October 27-29, 2003, Cracow, Poland*, pages 312–313, Kraków, 2004. ACC Cyfronet UMM.
- [7] M. Bubak, T. Gubała, M. Kapałka, and K. Malawski, M. Rycerz. Design of Distributed Grid Workflow Composition System. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'03 Grid Workshop, October 27-29, 2003, Cracow, Poland*, pages 115–124, Kraków, 2004. ACC Cyfronet UMM.

- [8] M. Bubak, M. Malawski, G. Młynarczyk, P. Nowakowski, R. Pająk, K. Rycerz, and M. Turała. Quality Assurance Aspects in the EU CrossGrid Project. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'03 Grid Workshop, October 27-29, 2003, Cracow, Poland*, pages 52–58, Kraków, 2004. ACC Cyfronet UMM.
- [9] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Execution Support for HLA-based Distributed Interactive Applications. In M. Bubak, M. Noga, and M. Turała, editors, *Proceedings of Cracow'03 Grid Workshop, October 27-29, 2003, Cracow, Poland*, pages 167–173, Kraków, 2004. ACC Cyfronet UMM.
- [10] M. Bubak, W. Funika, B. Baliś, T. Gubała, M. Malawski, M. Radecki, K. Rycerz, M. Smętek, and T. Szepieniec. CYFRONET Contribution to CoreGRID: Problem Solving Environments, Tools and GRID Systems. In M. Bubak, M. Turała, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'04, December 13-15 2004*, pages 45–57, Kraków, 2005. ACC-Cyfronet UST.
- [11] T. Gubała, M. Bubak, M. Malawski, and K. Rycerz. Abstract Workflow Composition in K-Wf Grid Project Environment. In M. Bubak, M. Turała, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'04, December 13-15 2004*, pages 85–92, Kraków, 2005. ACC-Cyfronet UST.
- [12] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. HLA Grid Based Support for Simulation of Vascular Reconstruction. In S. Gorlatch and M. Danelutt, editors, *Proceedings of the CoreGRID Workshop "Integrated Research in Grid Computing, November 28-30, 2005*, pages 165–174, Pisa, 2005. Technical Report TR-05-22.
- [13] K. Rycerz, M. Bubak, M. Malawski, and P. M. A. Sloot. Supporting HLA-based Distributed Interactive Simulations on the Grid - Current Status. In M. Bubak, M. Turała, and K. Wiatr, editors, *Proceedings of Cracow Grid Workshop - CGW'04, December 13-15 2004*, page 256, Kraków, 2005. ACC-Cyfronet UST. (Extended abstract).

Acknowledgments

I would like to take this opportunity and thank all the people I have worked with and met during Ph.D. research.

I would like to express my sincere thanks to my promoter professor *Peter Sloot*. I really appreciate his trust in me, his enthusiasm and help during the course of my work. His ideas and opinions have been invaluable and left a profound impact on this thesis. His open mind and critical approach to science were of great help during our discussions and taught me a lot. Peter, I am deeply thankful for your guidance. Thank you for your invitation to Amsterdam and your help during my visits. Furthermore, thank you for continuing to invest time in my research even when I became a young mother, which obviously reduced my availability :-).

This thesis would not have been written without enormous help from my co-promoter doctor *Marian Bubak*. Marian was the one who brought me into the exiting world of science and helped me cope with the harsh realities of the life of a researcher. Although he is a really busy person, he always found time to aid me with papers and chapters of this thesis. (Since he always has so many duties on his head, I suspect that he might even be a hidden member of "The Incredibles":-)). Marian, thank you for all your help during those years - and especially for your personal visits at my home, when I had to stay in and take care of my small son. You are truly one of a kind.

Special thanks are also due to *Piotr Nowakowski*, who – despite being terribly busy – proofed this thesis and most of my papers. Piotr, I am really grateful for all the time you have spent on my work - I do not know how this thesis would look without your help.

I also owe thanks to *Frank Seinstra* for translating the summary into Dutch. Frank, I know that translations are often a very boring job, so – big thanks for all your help!

I would like to acknowledge the efforts of the people who agreed to serve as members of my Ph.D. committee. I thank professor *Maarten Boasson*, doctor *Alfons Hoekstra*, professor *Robert Meijer*, doctor *Stephen Turner* and professor *Krzysztof Zieliński* for the time spent on reading the manuscript, for their suggestions, and for their

presence during the defense itself.

I would like to thank *Kamil Iskra* and *Wojtek Zajdel* who provided me with many hints about writing Ph.D. theses, translating Dutch forms and organizing the defense. Kamil was the one who helped me buy my first laptop, which proved instrumental during my work.

I would like to thank all the people I work with at the Institute of Computer Science AGH and CYFRONET AGH. Special thanks go to professor *Jacek Kitowski*, for many comments on my thesis. I also wish to thank the co-authors of my papers – *Maciek Malawski* for valuable discussions about the Grid, HLA and OGSA, *Bartek Baliś* for his help on the OCM-G system, *Robert Szymacha* for his work on wrappers for monitoring HLA with OCM-G. Thank you, *Tomek Gubała*, *Kamil Górka*, *Michał Kapalka* for interesting discussions about your work on scientific workflows. I also owe thanks to *Tomek Szepieniec* and *Marcin Radecki* - the great coordinators of Cross-Grid Tutorial. Thank you, *Włodzimierz Funika*, *Marcin Smetek*, *Renata Słota*, *Łukasz Dutka*, *Sławek Zieliński*, *Kazek Bałos*, *Zofia Mosurska*, *Milena Zajac* and *Maria Stawiariska*, for all the discussions and time spent together. Special thanks also go to *Paweł Ptaszczyk* for his hints on GRAM and GridFTP and *Dawid Kurzyniec* for help with JNI.

I would like to thank the people from the Section Computational Science, especially my co-authors *Alfredo Tirado Ramos* and *Zhiming Zhao* for long discussions about the Grid, HLA and components even during lunch breaks :-). Also, thank you, *Simon Portegies Zwart* and *Allesia Gualandris* for your help on the N-body application. Special thanks are due to *Elena Zudilowa-Seinstra*, who took care of me, so that I did not feel lonely during my stay. Thank you, *Elena*, *Frank*, *Heidi*, *Alfredo*, *Yan* and *Zhiming* for your hospitality during my visits to Amsterdam. Also, thank you *Robert Belleman* for your help on VRE and HLA. I would like to thank all members of SCS whom I have met - especially *Dick van Albada*, *Alfons Hoekstra*, *Breannán Ó Nualláin*, *Lilit Abrahamyan*, *Lera Krzhizhanovskaya*, *Abdel Artoli*, *Denis Shamonin*, *Roeland Merks*, *Roman Shulakov* and *Piero Spinnato*. Special thanks go to *Erik Hitepeuw* who helped me with all the forms and related paperwork.

I would like to thank all the people I have worked with and met during CrossGrid project. There were so many of them that it is not possible to mention them all. Special thanks go to professor *Michał Turata*, the project leader. I also wish to thank *Stefano Beco* for discussions about interactivity on the Grid. Thank you, *Marek Garbacz*, *Tomek Bołd*, *Tadek Szymocha* for all the work and time spent together. I also would like to thank *Robert Pająk* and *Witek Marton* who worked in CrossGrid office.

Special thanks go to *Fabrizio Gagliardi* and *Steve Fisher* for interesting discussions about DataGrid.

I would like to thank the maintainers of the computing facilities I used for my research. Thank you, *Andrzej Oziębło*, *Piotr Nyczyk*, *Patryk Lasoń* from Cyfronet and *Zeger Hendrikse*, *Kees Verstoep* from DAS2 for your patience and promptness when responding to reported problems. Special thanks go to former and current administrators at the Institute of Computer Science; *Marcin Radecki*, *Grzegorz Janoszka* and *Łukasz Skitał*.

Last, but not least, I would like to thank all of my family: my parents, my husband *Adam*, my sister *Małgosia* and my small son *Pawełek* for all their support and help. Drodzy rodzice - dziękuję Wam za pomoc i wsparcie !

My thanks also go to many other friends who have not been listed here.

Katarzyna Rycerz
Amsterdam, April 2006

