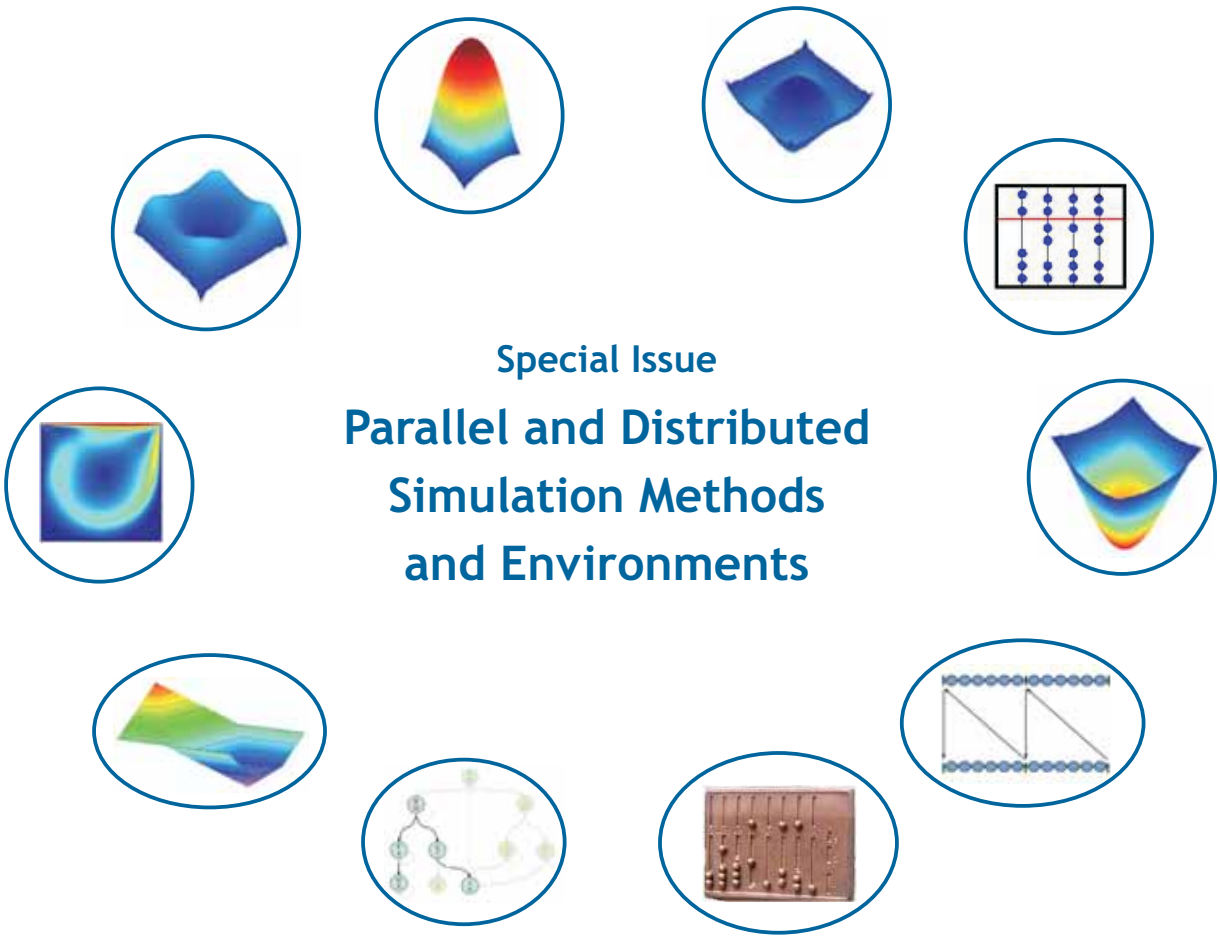
SNE SIMULATION NEWS EUROPE



Special Issue

Parallel and Distributed Simulation Methods and Environments

Volume 16 Number 2 September 2006, ISSN 0929-2268

Journal on Developments and Trends in Modelling and Simulation

Special Issue

ARGESIM

Dear readers,

We are glad to present the first SNE Special Issue - a Special Issue on 'Parallel and Distributed Simulation Methods and Environments'. The idea for special issues was born in ASIM, the German Simulation Society. As there was and as there still is a need for state-of-the-art publications in topics of modelling and simulation, ASIM first tried to publish monographs on this subject. But publication of such books showed disadvantages: too slow production time, too high costs, and lack of publication issues. ASIM, seeking for alternatives, contacted ARGESIM with the idea of SNE Special Issues - while ARGESIM itself thought on Special Issues, because of lack in publication space in the regular SNE issues. Now, one year after the first contact, we can present the first Special Issue, edited by Thorsten & Sven Pawletta from University Wismar, Germany.

The editorial policy of SNE Special Issues is to publish high quality scientific and technical papers concentrating on state-of-the-art and state-of-research in specific modeling and simulation oriented topics in Europe, and interesting papers from the world wide modeling and simulation community. This Special Issue 'Parallel and Distributed Simulation Methods and Environments' (SNE 16/2), will be sent to all ASIM members - together with the regular SNE 16/1 (SNE 46), and sample copies will be sent to other European Simulation Societies; furthermore, it is available on basis of an individual subscription of SNE - SNE Special Issues are open for everybody, for publication and subscription (not only for ASIM). We think also on Special Issues publishing selected papers from EUROSIM conferences.

We hope, you enjoy this Special Issue, which presents state-of-the-art in parallel and distributed simulation, from theory with lookahead formulas via implementation with HLA and other systems to applications in ship desgin and blood flow simulation.

It is planned to publish a SNE Special Issue each year, for 2007 a Special Issue on 'Verification and Validation' (Guest Editor Sigrid Wenzel, University Kassel) is scheduled (SNE 17/2). I would like to thank all people who helped in managing this first Special Issue, especially the Guest Editors, Thorsten and Sven Pawletta from Wismar University.

Felix Breitenecker, Editor-in-Chief SNE; *Felix.Breitenecker@tuwien.ac.*at

Content

SNE Editorial Board

Guest Editors Special Issue *Parallel and Distributed Simulation Methods and Environments*
Thorsten Pawletta, *pawel@mb.hs-wismar.de*
Sven Pawletta, *s.pawletta@et.hs-wismar.de*
Res. Group Computational Engineering and Automation, Wismar University, 23952 Wismar, Germany
WWW.MB.HS-WISMAR.DE/cea

SNE Contact
SNE-Editors / ARGESIM
c/o Inst. f. Analysis and Scientific Computation
Vienna University of Technology
Wiedner Hauptstrasse 8-10, 1040 Vienna, AUSTRIA
Tel + 43 - 1- 58801-10115 or 11455, Fax - 42098
sne@argesim.org; WWW.ARGESIM.ORG

Editorial Info - Impressum
SNE Simulation News Europe ISSN 1015-8685 (0929-2268).
Scope: Development in modelling and simulation, benchmarks on modelling and simulation, membership info for EUROSIM and Simulation Societies.
Editor-in-Chief: Felix Breitenecker, Inst. f. Analysis and Scientific Computing, Mathematical Modelling and Simulation, ,
Vienna University of Technology, Wiedner Hauptstrasse 8-10,
1040 Vienna, Austria; *Felix.Breitenecker@tuwien.ac.at*
Layout: A. Breitenecker, ARGESIM TU Vienna / Linz;
Anna.Breitenecker@liwest.at
Printed by: Grafisches Zentrum TU, Wiedner Hauptstr. 8-10, A-1040, Wien
Publisher: ARGESIM and ASIM
ARGESIM, c/o Inst. for Analysis and Scientific Computation, TU Vienna, Wiedner Hauptstrasse 8-10, A-1040 Vienna, Austria, and
ASIM (German Simulation Society), c/o Wohlfartstr. 21b, 80939München

SNE 16/2, September 2006

Editorial SNE Special Issue

Parallel and Distributed Simulation Methods and Environments

This is the first *Special Issue of SNE*, edited by members of the ASIM working group *Methods of Modeling and Simulation* . The new *SNE Special Issue Series* has been introduced as an extension of the regular SNE. The aim is to publish high quality scientific and technical papers concentrating on a specific topic. Using this approach the SNE Special Issues will present the state of research in specific modeling and simulation oriented topics in Europe, and interesting papers from the world wide modeling and simulation community. This Special Issue of SNE is devoted to *Parallel and Distributed Simulation Methods and Environments* and includes seven selected papers and a call for a benchmark in distributed and parallel simulation.

The development of parallel and distributed simulation methods and software tools has been strongly influenced by *High Level Architecture* (HLA) in recent years. HLA has its origins in the military simulation community. As a consequence of its openness and generic character it has also had a significant impact on non-military applications and is now an IEEE standard for distributed simulation.
The *first paper* by Strassburger (Fraunhofer Institute Magdeburg, Germany) introduces the history of *HLA*, presents its main concepts and discusses recent developments. It provides enough background information for non-experienced readers in this field for the two further HLA related contributions in this journal.

The *second paper* and the *third paper* discuss specific parallel and distributed simulation approaches for *Discrete EVent specified Systems* (DEVS) and the associated simulator algorithms. Zacharewicz, Frydman and Giambiasi (University Marseille, France) investigate new lookahead computation methods in the G-DEVS/HLA environment. G-DEVS is a specific extension of the DEVS theory and of DEVS simulator algorithms for hybrid dynamic systems. Continuous and discrete model components and their associated simulators can be located on different computers and integrated into a global simulation model using HLA technology.

The contribution by Wainer and Glinsky (Carleton University, Ottawa, Canada) investigates parallel simulation techniques for DEVS and *Cell-DEVS* models that combine cellular automata with DEVS theory. In their parallel simulation environment, CD++, the DEVS simulation algorithms are modified and combined with conservative and optimistic synchronization algorithms.

Scientific and Technical Computing Environments (SCEs) such as MATLAB, Scilab or Octave are essential tools in today's computational engineering and science. Especially optimization and simulation are well supported by integrated algorithms and subsystems like Simulink, Scicos or Stateflow. The *fourth paper* by Fink, Pawletta and Lampe (Wismar University, Germany) gives a detailed overview about SCE based parallel processing. In this paper, a new taxonomy on SCE based parallel processing is presented, followed by the identification and assignment of more than 30 existing projects. Furthermore, simulation and optimization applications which have been parallelized under usage of SCEs are discussed. Parallel runtime results as well as general application characteristics are presented.

The *fifth*, *sixth* and *seventh* papers have been motivated by engineering applications.

Stenzel, Pawletta, Ems and Bünning (Wismar University and MTG Marinetechnik GmbH, Hamburg; Germany) describe an application, where existing real-world software components, mainly written in Fortran, have to be integrated into an HLA compliant federation. Fortran/HLA integration approaches are examined in detail, whereas experiences in the field of MATLAB/HLA connectivity serve as design pattern.

The contribution by Eichler, Knöchel, Altmann, Hartong and Hartung (Fraunhofer Institute Dresden and Cadence Design Systems GmbH, Feldkirchen; Germany) describes the coupling of different simulators via TCP/IP network socket connection. The implementation and application of such a co-simulation is described in detail for the simulators MATLAB/Simulink and AMS Designer.

The contribution by Leitner, Wassertheurer, Breitenecker, Hessinger and Holzinger (ARC Seibersdorf research GmbH, Vienna; Vienna University of Technology; Medical University Graz; Austria) presents a *Lattice-Boltzmann model* (LBM) for solving fluid mechanical problems in engineering and biomedical applications. The investigated model is relevant for blood flow simulation because it uses Reynolds and Womersley numbers found in haemodynamics with a realistic time dependent pressure gradient as a boundary condition. A big advantage of LBM is the possibility of easy parallelization.

Therefore different approaches and implementations are discussed and compared with respect to parallelization efficiency.

Furthermore, this SNE Special Issue publishes a call for a benchmark on parallel and distributed simulation tasks. This new *ARGESIM Benchmark* on *Parallel and Distributed Simulation* extends the ARGESIM Comparison on *Parallel Simulation Techniques* from 1994. The three tasks of this benchmark are more general, so that not only simulation software is addressed, so that also different algorithms for solving the tasks can be used, and so that different strategies for parallelization or distribution of the tasks can be set up and compared.

This SNE Special Issue on *Parallel and Distributed Simulation Methods and Environments (*SNE 16/2), will be sent to all ASIM members - together with the regular SNE 16/1 (SNE 46), and sample copies will be sent to other European Simulation Societies (with ordering offer). Furthermore, it is available on basis of an individual subscription of SNE.

It is planned to publish a *SNE Special Issue* each year. We would like to draw your attention to the *Call for Papers* for the next special issue (see below) on *Verification and Validation in Modeling and Simulation*, edited by the ASIM working group *Simulation in Production and Logistics* (Guest Editor Sigrid Wenzel, University Kassel).

Finally, we would like to thank all authors, who have contributed to this special issue, our co-workers at Wismar University for various support, the ARGESIM people at Vienna University of Technology for editorial support, and F. Breitenecker (Editor-in-Chief of SNE) for good co-operation.

Thorsten Pawletta & Sven Pawletta
Guest Editors SNE Special Issue *Parallel and Distributed Simulation Methods and Environments*
Res. Group Computational Engineering and Automation, Wismar University
PF 1210, 23952 Wismar, Germany
WWW.MB.HS-WISMAR.DE/cea

Call for Contributions SNE Special Issue 2007

Verification and Validation in Modelling and Simulation

Simulation is an important method which helps to take right decisions in system planning and operation. Building high-quality simulation models and using the right input data are pre-conditions for achieving significant and usable simulation results.
For this purpose, a simulation model has to be well-defined, consistent, accurate, comprehensive and applicable. The quality criteria can be proved by verification (*building a model in the right way*) and validation (*building the right model*).

The ASIM-Working Group *Simulation in Production and Logistics* which has worked on this topic since three years accommodates the increased significance of verification and validation and will publish the forthcoming Special Issue of Simulation News Europe (SNE) on this topic.

Papers on the following topics will be welcome:

- Procedure Models for Verification and Validation
- Methods for Verification and Validation
- Certification and Accreditation
- Information / Data Acquisition for Simulation Models and their Verification and Validation
- Verification and Validation - Documentation Aspects
- Credibility
- Automatic Verification and Validation
- Case Studies and Practical Experiences

The Guest Editor of this *SNE Special Issue* (SNE 17/2), Prof. Dr. Sigrid Wenzel from University Kassel, invites for submitting contributions.

Contributions should not exceed 8 pages and should be mailed directly to the editor not later than March 31, 2007; contributions will be peer reviewed (templates available at ASIM and ARGESIM web page - WWW.ASIM-GI.ORG, WWW.ARGESIM.ORG).

Sigrid Wenzel
Guest Editor SNE Special Issue
Verification and Validation in Modeling and Simulation
Department of Mechanical Engineering
University of Kassel, Kurt-Wolters-Strasse 3
D-34125 Kassel, Germany
s.wenzel@uni-kassel.de
WWW.UNI-KASSEL.DE/fb15/ipl/pfp/

CALL FOR PAPERS

CALL FOR PAPERS

September 9-13, 2007, Ljubljana, Slovenia

EUROSIM - Federation of European Simulation Societies

EUROSIM 2007

6th EUROSIM Congress on Modelling and Simulation

September 9 - 13, 2007, LJUBLJANA, SLOVENIA

About EUROSIM:

EUROSIM is the Federation of European Simulation Societies and the EUROSIM congress organization (a triennial event) is one of the most important activities of the federation.

For more information about EUROSIM see:
www.eurosim.info

PROGRAMME:

The EUROSIM 2007 scientific programme consists of: Plenary lectures, Regular sessions, Special sessions, Posters, Students' competition and Tutorials. Papers will be published in two Proceedings Volumes: Volume 1: Book of Abstracts, Volume 2: DVD volume with full papers and multimedia files.

SCOPE AND TOPICS:

The scope includes all aspects of continuous, discrete (event) and hybrid modelling, simulation, identification and optimisation approaches. So the common denominator is problems solving with modelling and simulation in a way that can be useful also for solving other problems in similar or different areas. Contributions from technical (engineering) areas but also from nontechnical areas are welcome.

M&S methods and technologies: modelling and simulation of complex, large scale, distributed, hybrid, hierarchical, stochastic, control, expert, adaptive, fuzzy, decision support, multivariable, multiagent, reconfigurable, agent based, knowledge based, real time, queuing systems, scheduling, parallel processing concepts, high performance computing, M&S system architectures, neural networks, model validation and verification, simulation life-cycle evolution, genetic algorithms, man-in-the loop simulation, hardware-in-the loop simulation, nested simulation models, distributed enterprise simulation, data mining, bond graphs, simulation with Petri nets, discrete event simulation, statistic modelling, component based modelling, object oriented modelling, mathematical /numerical methods in simulation, graphical modelling, nano technology modelling, embedded and firmware modelling, middleware architecture modelling, visualisation, graphics and animation, modelling and simulation tools, WEB based simulation, human behaviour representation techniques, virtual reality and virtual environments, CAD/CAM/CIM/CAE, experiential digital media, future of M&S

M&S applications: aerospace, automotive systems and transportation, agriculture, architecture, biopharmacy, biomedicine, bioinformatics, genomics, business, applied chemistry, civil engineering, communications, ecological and environmental systems, economics, econometrics, economics of M&S, education, electrical engineering, geophysical systems, industrial processes, logistics, manufacturing systems, maintenance, reliability, marine systems, materials modelling and simulation, mechanical engineering, mechatronics, meteorology/climate, military systems, organisational processes, process engineering, traffic/transportation, power systems, applied psychology, computational fluid dynamics, training simulators, social systems biology, sciences, robotics, water management and treatment, mobile robotics, seismism, pulp & paper, supply chains, lifecycle management and plant data

VENUE:

University of Ljubljana, Faculty of Electrical Engineering, Ljubljana, Slovenia

DEADLINES:

Proposal for special sessions and tutorials: **1 Feb. 2007**

Submission of extended abstracts: **9 April 2007**

Notification of acceptance: **30 May 2007**

Early registration: **11 June 2007**

Submission of camera-ready papers: **9 July 2007**

Hotel Reservation: **27 July 2007**

CONTACTS:

Borut Zupančič, congress chair
Rihard Karba, IPC chair
University of Ljubljana, Faculty of Electrical Engineering
Tržaška 25, SI-1000 Ljubljana, Slovenia
Phone: +386 1 4768 306
E-mail: borut.zupancic@fe.uni-lj.si
E-mail: rihard.karba@fe.uni-lj.si

Alenka Kregar, registration, accommodation
Cankarjev dom, Cultural and Congress Centre
Prešernova 10, SI-1000 Ljubljana, Slovenia
Phone:+386 1 241 7133
Fax: +386 1 241 7296
E-mail: alenka.kregar@cd-cc.si

CONGRESS COMMITTEE:

Borut Zupančič, president of EUROSIM, chair

Rihard Karba, president of SLOSIM

Tomaž Slivnik, Univ. of Lj., Fac. of El. Eng., dean

Felix Breitenecker, president of ASIM

INTERNATIONAL PROGRAMME COMMITTEE:

R. Karba (SI), chair
D. Al – Dabass (UK),
M. Alexik (SK),
I. Bausch-Gall (DE),
L. Bobrowski (PL),
W. Borutzky, (DE)
J. Božikov (HR),
F. Breitenecker (AT),
P. Bunus (SE),
P. Cafuta (SI),
R. Cant (UK),
A. Carvalho Brito (PT),
G. Cedersund (SE),
F. Cellier (CH),
V. Čerić (HR),
E. Dahlquist (SE),
B. Elmegaard (DK),
P. Fritzson (SE),
J.M. Giron-Sierra (ES),
Y. Hamam (FR),
F. Hartescu (RO),
A. Heemink (NL),
V. Hlupic (UK),
F. Javier Otamendi (ES),
A. Jávor (HU),
K. Jezernik (SI),
Đ. Juričić (SI),
K. Juslin (FI),
E. Juuso (FI),
H. Karatza (GR),
E. Kindler (CZ),
M. Kljajić (SI),
M. Klug (AT),
J. Kocijan (SI),
J. Kunovsky (CZ),
F. Lebon (FR),
B.H. Li (CN),
H.X. Lin (NL),
F. Maceri (IT),
W. Maurer (CH),
Y. Merkuryev (LV),
A. Munitić (HR),
D. Murray-Smith (UK),
S. Oharu (JP),
A. Orsoni (UK),
K. Panreck (DE),
T. Pawletta (DE),
H. Pierreval (FR),
J. Pollard (UK),
C.Z. Radulescu (RO),
M. Radulescu (RO),
F. Rocaries (FR),
P. Schwarz (DE),
M. Savastano (IT),
W. Smari (US),
F. Stanciulescu (RO),
G. Szucs (HU),
M. Šnorek (CZ),
I. Troch (AT),
S. Wenzel (DE),
W. Wiechert (DE),
E. Williams (US),
R. Zobel (UK, TH),
B. Zupančič (SI),
L. Žlajpah (SI)

ORGANISERS:

- SLOSIM - Slovene Society for Simulation and Modelling
- University of Ljubljana, Faculty of Electrical Engineering
- EUROSIM member societies: ASIM, CROSSIM, CSSS, DBSS, FRANCOSIM, HSS, ISCS, SIMS, UKSIM, AES, PSCS, ROMSIM

CO-SPONSORS:

- CASS Chinese Association for System Simulation,
- ECMS European Council for Modelling and Simulation,
- JSST Japan Society for Simulation Technology,
- LSS Latvian Simulation Society,
- SCS The Society for Modeling and Simulation Int.

EXHIBITION:

Exhibitors with software, hardware and books from the area of M&S are cordially invited to participate.

Overview about the High Level Architecture for Modelling and Simulation and Recent Developments

Steffen Straßburger, *steffen.strassburger@iff.fraunhofer.de*
Fraunhofer Institute for Factory Operation and Automation, Magdeburg, Germany

The High Level Architecture for Modeling and Simulation, or HLA for short, is an IEEE standard for distributed simulation. It focuses on interoperability and reusability of the components (called federates) and offers time management interoperability as well as sophisticated data distribution concepts. HLA has its origin in the military simulation community where one of its major tasks is the networking of military training simulators. However, due to its openness and generic character it also has a large impact on non-military distributed simulation applications. Due to these facts, HLA can still be regarded the state-of-the-art standard for distributed simulation. This article introduces the background and history of HLA, introduces its main concepts, and discusses recent developments. A summary and evaluation of the future of HLA concludes this contribution.

SNE 16/2, September 2006

Introduction and Motivation

The development of the *High Level Architecture for Modeling and Simulation* (HLA) was initiated by the *U.S. Department of Defense* (DoD) in 1995 out of the need for a common high-level simulation architecture. The standard was supposed to facilitate the interoperability and reusability of all types of simulation used and sponsored by the DoD.

The necessity of the standard is derived from the complexity and variety of simulation applications in use and the manifold of expectations towards simulation applications. They include different levels of abstraction, different levels of interactivity, different temporal behavior, etc. In essence, no single monolithic simulation application could fulfill all requirements of all users.

Considering the different simulation applica-tions in use, no one could foresee all their potential usage and combinations in advance. Thus, the idea of a modular, composable approach for building federations of simulations was born which eventually led to the development of the HLA.

HLA's main objective was to provide an open architecture offering services for interoperability and reusability. The architecture has no limitations towards a specific simulation paradigm. It is not even limited to simulation applications, rather it offers interoperability to all kinds of programs. However, HLA provides specific interoperability support services to accommodate specific needs of simulation applications. With that, HLA supersedes general interoperability standards like CORBA or DCOM.

Initiated as a standard in the military simulation community the development of the HLA has been overseen by the *Defense Modeling and Simulation Office* (DMSO) for the U.S. DoD. DMSO has deliberately taken a very open approach in the definition and accessibility of HLA and has sponsored publicly available software implementations of the HLA software.

With this policy DMSO has ensured a broad community involvement in the development of HLA, which can be seen as a cornerstone to its rather good acceptance and adoption. In the military simulation domain HLA is a mandatory standard not only in the U.S., but also throughout most NATO countries.

HLA involvement of the civilian simulation community has mostly originated from academia [1] and has been rather research oriented. Significant efforts have been focused on using HLA as a standard for interoperability between commercial of the shelf simulation packages [2,3,4].

An important joint research project in the field of civilian HLA applications was the *IMS Mission project*. Its focus was on adopting HLA as a standard for design, planning and operation of globally distributed enterprises. One outcome was a concept and solution for distributed supply chain simulation [5].

Serious practical applications of HLA have been investigated by several companies, among them Daimler Chrysler in the automotive sector [6]. Especially for the automotive industry with its large supplier networks and rather advanced use of digital planning and simulation methods within their *Digital Factory* efforts, HLA can play a substantial role for providing plug-and-play simulation interoperability.

1 A Short History of HLA

Research in the field of distributed simulation has a long tradition. *Parallel distributed event simulation* (PDES) is one important branch of distributed simulation primarily driven from the civilian simulation community which aims at performance and speedup issues. Conservative and optimistic synchronization protocols were developed to handle possible causality violations between simulations. In the military simulation community, the *Distributed Interactive Simulation* (DIS) technology was developed primarily for the connection of real-time training simulators. DIS is defined in the IEEE 1278 standard since 1993. Another standard for the connection of constructive military simulations was the *Aggregate Level Simulation Protocol* (ALSP).

Outside the simulation domain, standards for distributed computing like the *Parallel Virtual Machine* (PVM) and the *Message Passing Interface* (MPI) have been developed which also influenced the field of distributed simulation. The HLA combines its predecessor technologies from the military sector, DIS and ALSP, and is the designated standard architecture for all U.S. DoD modeling and simulation activities. The HLA development started in 1995 with the *DoD Modeling & Simulation Master Plan* which demanded to 'establish a common high-level simulation architecture to facilitate the interoperability of all types of models and simulations ..., as well as to facilitate the reuse of M&S components'.

The timeline of the development of the HLA is depicted in Figure 1. The base definition of the HLA 1.0 Standard in August 1996 can be regarded the first stable HLA definition. Shortly after its release, different versions of the RTI software developed under DMSO sponsorship became publicly available. This software was distributed freely in the community and included an RTI Help Desk as a support infrastructure for maintaining the software. The next major release of the HLA standard was HLA 1.3 in February 1998. This version of the standard is still quite commonly in use today in many simulation applications. Two RTI developments following this 1.3 release of the standard were made publicly available in the following time, the first being RTI 1.3 in 1998, the next being RTI 1.3 NG in 1999. The latter release offered improved performance and is still available today from the *Virtual Technology Corporation* as RTI NG Pro. The HLA 1.3 release formed the base for further standardization efforts. Among them was the adoption of HLA by the OMG as facility for distributed simulation.

The most important standardization activity was the release of the IEEE version of the HLA standard. This release is in most parts similar to HLA 1.3, but contains several needed improvements which surfaced in the practical use of HLA 1.3 [7]. Also, some modifications needed in order to comply with IEEE requirements were made.

The year 2002 marked the end of a transition phase in which DMSO had led (and sponsored) the efforts to develop HLA. Having become an IEEE standard the further development of HLA was given into the hands of the *Simulation Interoperability Standards Organization* (SISO).

SISO originated over ten years ago with the *Distributed Interactive Simulation* (DIS) Workshops and was since that time focused on creating standards for simulation interoperability. SISO is a volunteer organization with members from industry, military and academia.

Besides hosting the *Simulation Interoperability Workshops* (SIW) which are organized three time a year (two in the U.S.A, one in Europe) SISO hosts a *Standards Activity Committee* (SAC) which oversees the work of several *Product Development Groups* (PDG). PDGs are the actual groups of people developing standards for simulation interoperability. Most of their work is based on HLA, including its future refinement and development of standards.

The most important PDG is the *HLA-Evolved Initiative* as it oversees the review of the IEEE 1516 specification. Many new potential HLA requirements have been identified based on feedback from the various domains and application areas. The PDG seeks to address these requirements via a formal open review of the IEEE 1516 series of specifications.

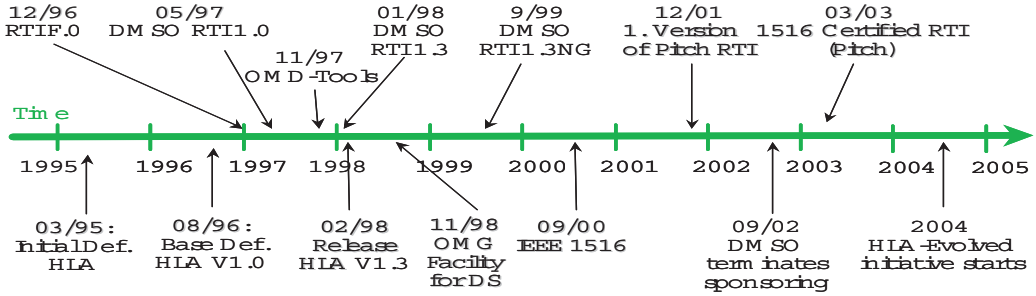


Figure 1: HAL development timeline.

As part of this process, the PDG will incorporate those aspects raised in the DoD Interpretations Document for IEEE 1516 [8] and a Dynamic Link Compatible HLA API for IEEE 1516.1.

2 Major Concepts of HLA

In order to facilitate interoperability and reusability, HLA differentiates between the simulation functionality provided by the members of the distributed simulation and a set of basic services for data exchange, communication and synchronization. Figure 2 gives an functional overview of a distributed simulation under the HLA paradigm.

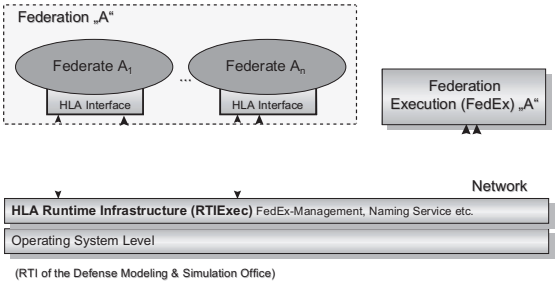


Figure 2: Functional view of a distributed simulation under HLA.

In HLA, individual simulations and other participants of a distributed simulation are referred to as *federates*. Federates which are supposed to co-operate together under certain guidelines and a defined object model form a so-called *federation*. Federates use a *common runtime infrastructure* (RTI) for communication. The RTI is a piece of software which can be regarded as a distributed operating system add-on. HLA defines a bi-directional interface between federates and the RTI. A single run is referred to as a *federation execution*.

The current version of the *High Level Architecture for Modeling and Simulation*, or HLA for short, is formally defined in the three key documents of IEEE standard 1516.

These documents are

- 1516-2000: Framework and Rules
- 1516.1-2000: Federate Interface Specification
- 1516.2-2000: Object Model Template (OMT) Specification

All three elements are briefly discussed in detail in the following sections.

2.1 HLA Rules

The HLA Rules define the required behavior of a federation and its federates and are thus part of the formal HLA compliance definition. There are 5 rules for federations and 5 for federates ([11]).

Rules for federations:

1. Federations shall have an HLA FOM, documented in accordance with the HLA OMT.
2. In a federation, all simulation-associated object instance representations shall be in the federates, not in the runtime infrastructure.
3. During a federation execution, all exchange of FOM data among federates shall occur via the RTI.
4. During a federation execution, joined federates shall interact with the RTI in accordance with the HLA interface specification.
5. During a federation execution, an instance attribute shall be owned by at most one joined federate at any given time.

Rules for federates:

6. Federates shall have an HLA SOM, documented in accordance with the HLA OMT.
7. Federates shall be able to update and/or reflect any instance attributes and send and/or receive interactions, as specified in their SOMs.
8. Federates shall be able to transfer and/or accept ownership of instance attributes dynamically during a federation execution, as specified in their SOMs.
9. Federates shall be able to vary the conditions under which they provide updates of instance attributes, as specified in their SOMs.
10. Federates shall be able to manage local time in a way that will allow them to coordinate data exchange with other members of a federation.

2.2 HLA Federate Interface Specification

The *HLA Federate Interface Specification* describes the services which federates have to use for communicating with other federates via a runtime infrastructure (RTI). The interface specification describes which services can be used by a federate and which services it has to provide [12].

This bi-directional character of the interface is encapsulated into an ambassador paradigm. A federate communicates with the RTI using its *RTI ambassador*. Conversely, the RTI communicates with a federate via its *federate ambassador*. From the federate programmer's point of view, these ambassadors are objects and the communication among the participants is performed by calling methods of these objects. Thus, the services defined in the interface specification are either methods of the RTI ambassador or of the federate ambassador.

The interface specification defines six categories of services, which will be briefly described in the following sections. A special advantage of HLA compared to other technologies are the time management and the data distribution management services.

The time management services provide a mechanism for coordinating simulation clocks of simulations using a wide variety of time advance mechanisms. In comparison with other technologies, where time management/synchronization is only available to a certain type of simulation, HLA provides a general solution for all types of simulations.

The services provided in the data distribution category provide new mechanisms for efficiently transferring data among certain federates and for reducing the amount of data transferred. They are special in that regard, in that previous technologies (like DIS) are usually based on broadcast principles for distributing data.

Federation Management

The main focus of the services in the Federation Management service group is the coordination of federation-wide activities during a federation execution. They are used by federates to initiate, join, resign, and manage a federation execution.

Interface services include:

- *Create/Destroy Federation Execution*: These service are used to create and destroy federation execution. Usually the first federate joining a federation execution has the task of creating it. The last federate leaving a federation execution commonly destroys it.
- *Join/Resign Federation Execution*: These services are used by federates to join a federation execution and to resign from it once the federate has completed its tasks.
- *Services to save and restore federation executions*: These services can be used to save and restore the state of the federation. It should be noted that these service only coordinate the save/restore process. The internal state saving mechanisms have to be implemented by the federates themselves.

Declaration Management

Federates shall use declaration management ser-vices to declare their intention to generate and receive information. A federate shall invoke appropriate declaration management services before it can register or discover object instances, update or reflect instance attribute values, and send or receive interactions.

With that, declaration management could also be seen as an 'interest management'. Federates specify, which data types they would like to send or receive. The publishing and subscribing of data types (object and interaction classes with their attributes and parameters) has to be performed in accordance with the SOMs and the FOM. Although declarations can be changed dynamically during a federation execution, the declaration management belongs to the initialization phase of a federation.

Interface services include:

- *Publish Object Class Attributes/Interaction Class*: These services are used to announce that a federate intends to generate the specified object and interaction classes 'later on' during a federation execution.
- *Subscribe Object Class Attributes/ Interaction Class*: These services are used to announce that a federate is interested in the specified object and interaction classes and would like to receive infor-mation about these classes from now on.
- *Start (Stop) Registration For Object Class/ Turn Interactions On (Off)*: Using these callback functions to the federate, the RTI can inform a federate whether other federates are interested in the object classes and interactions it has published. These services implement the so-called 'advisory switches' which inform federates of the relevance of their publications. Federates can chose to ignore these switches and register object instances/ send interactions regardless of whether other federates are interested.

Figure 3 gives an example of services that two federates might use to manage their subscription and publications.

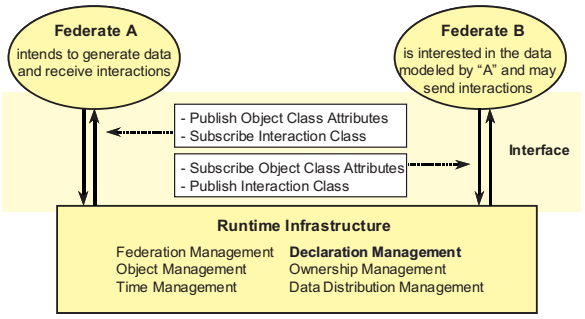


Figure 3: Declaration management (adopted from [9])

Object Management

This group of the interface specification provides services for the registration, modification, and deletion of object instances and the sending and receiving of interactions. The services of this group provide the necessary functionality for all data exchange among federates.

RTI services include:

- *Register Object Instance/Discover Object Instance*: Each object that is relevant to a federation execution needs to be registered with the RTI using the Register Object Instance service. Interested federates will be notified of the existence of such an object instance via the *Discover Object Instance callback* to their federate ambassador.
- *Update/Reflect Attribute Values*: After informing the RTI about the existence of an object instance, the registering federate can start sending updates for this object via the *Update Attribute Values service*. Interested federates will receive updates via the Reflect Attribute Values callback to their federate ambassador.
- *Send/Receive Interaction*: Interactions can be sent via the Send Interaction service and are received via the Receive Interaction callback service.
- *Delete Object Instance*: This service removes an object instance from a federation execution.
- *Change Transport and Ordering Mechanisms*: Object updates and interactions are transported using certain transportation and ordering mechanisms which can be changed at runtime. Transportation types include reliable and besteffort transmission, ordering mechanisms include time stamp order and receive order.

Figure 4 gives an example for the usage of the services introduced in this section.

Ownership Management

Ownership management can be used by federates and the RTI to transfer ownership of attribute instances among federates. The ability to transfer ownership is intended to support the cooperative modeling of a given object instance across a federation.

The services provided by this group support both push and pull mechanisms for ownership transfer.

RTI services include:

- *Negotiated Attribute Ownership Divestiture/ Request Attribute Ownership Assumption*: The service Negotiated Attribute Ownership Divestiture is intended for federates that want to wishes to divest itself of the instance attribute (push). Request Attribute Ownership Assumption is the corresponding callback to the federate ambassador for informing the federate about the request.
- *Attribute Ownership Acquisition/ Request Attribute Ownership Release*: The service Attribute Owner-ship Acquisition is intended for federates that want to become owner of a certain set of attributes (pull). Request Attribute Ownership Release is the corresponding callback to the federate ambassa-dor for informing a remote federate about the re-quest.
- *Attribute Ownership Divestiture Notification/ Acquisition Notification*: These services inform the federates about the success of their respective requests.

Figure 5 illustrates the push mechanism outlined above.

Time Management

Time management is concerned with the mechanisms used by simulations to advance through simulation time. Time advances are coordinated by the RTI with object management services so that information is delivered to federates in a causally correct and ordered fashion. *HLA Time Management* provides mechanisms to support all major types of regimes to advance simula-tion time, such as (scaled) real-time and as-fast-as-possible simulations.

An important design principle that is used to allow this functionality is time management transparency. This means the local time management mechanism used in a certain federate does not have to concern other federates. For instance, a federate using an event-oriented mechanism does not need to know whether the federate with which it is interacting is also using an event-oriented mechanisms, or (say) a time-stepped mechanism.

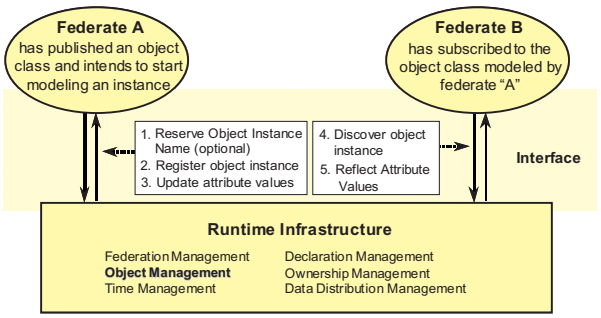


Figure 4: Object management (adopted from [9]).

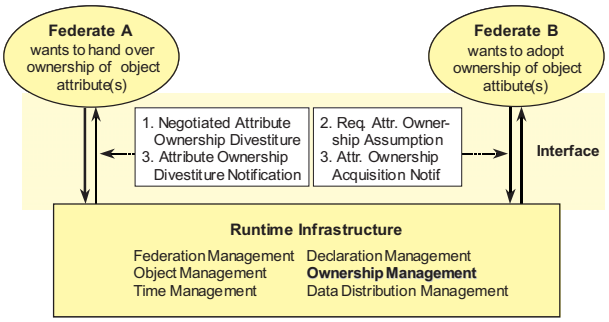


Figure 5: Ownership management (adopted from [9]).

An HLA federation may even include federates using HLA Time Management services to co-ordinate their time advances, and others, that do not. In such an environment, the RTI must determine those federates that must be considered when coordinating time advances. Therefore HLA introduces two boolean flags that determine the federate's time management characteristics. They are called time-constrained and time-regulating flags. A time regulating federate is one that wishes send time-stamped messages to other federates and thus influence their time advancement. A time-constrained federate is one that wishes to be able to receive time-stamped messages, and thus subordinates itself to the federation time advancement.

The HLA time management services are strongly related to the services for exchanging messages, e.g., attribute updates and interactions. There are two general ordering types for messages under HLA: receive-order (RO) and time-stamp-order (TSO). Receive-ordered messages are simply placed in a queue when they arrive, and are immediately eligible for delivery to the federate. TSO messages are assigned a time-stamp by the sending federate, and are delivered to each receiving federate in the order of non-decreasing time stamps. Incoming TSO messages are placed into a queue within the RTI, but are not eligible for delivery to the federate until the RTI can guarantee that there will be no TSO messages for that federate with a smaller time stamp.

In order to allow the RTI to perform time management, a federate must use one of the following time management services (as appropriate for the internal time advance mechanism of the federate):

- *Next Event Request (NER)*. Event driven federates need to process local and external events, i.e., events generated by other federates, in time-stamp-order. The federate time, i.e., its logical simulation time, typically advances to the time stamp of each event as it is processed. An event driven federate will typically use the *Next Event Request* service when it has completed all simulation activity at its current logical time in order to advance to the time stamp of its next local event.
- *Time Advance Request (TAR)*. Time step driven federates make time advances in time steps with some fixed duration of simulation time. The simulator does not advance to the next time step until all simulation activities within the current time step have been completed. This type of federate will usually use the *Time Advance Request* service to request to advance its logical time to the next time step.
- *Flush Queue Request (FQR)*. FQR can be used for optimistically synchronized federates to request the out-of-order delivery of events. HLA supports optimistic federates while maintaining time management transparency. Specifically, the HLA time management services do not require all federates to support a rollback and recovery capability even if one federate is using optimistic event processing. FQR is used by optimistic federates to receive all buffered messages (although there might be some messages at a later point in time which carry a smaller time stamp). In such a case the federate will have to use the other important service Retract, which can be used to cancel a previously sent message. The RTI ensures that 'optimistic' messages are only received by optimistic federates, as long as there is a possibility of a later cancellation of that message.

Using one of these services a typical synchronization loop of an HLA federate would work in the following three step order:

1. Request advancement of logical time by calling the appropriate RTI service (e.g., *NextEvent*, ...)
2. Receive zero or more messages from the RTI (e.g., receive *Reflect Attribute Value* or receive *Interaction callback* from the RTI)
3. Receive a *Time Advance Grant* callback from the RTI to indicate that the federate's logical time has been advanced.

A more detailed discussion of HLA Time Management can be found in [10].

Data Distribution Management

The *data distribution management* (DDM) services provide a mechanism to reduce both the transmission and the reception of irrelevant data. Whereas declaration management services provide information on data relevance at the class attribute level, data distribution management services add a capability to further refine the data requirements at the instance attribute level.

This is achieved be defining multi-dimensional routing spaces. The producers of data (the sending federates) are expected to specify an update region associated with a specific attribute update or interaction. This is the region in which the update or interaction is relevant. Receiving federates have to specify which regions they are interested in (subscription regions). The actual data transfer only takes place if the update and subscription regions for a specific update or interaction overlap (Figure 6).

The usage of DDM is optional, but provides a sophisticated means for the minimization of the amount of transferred data and thus the network load.

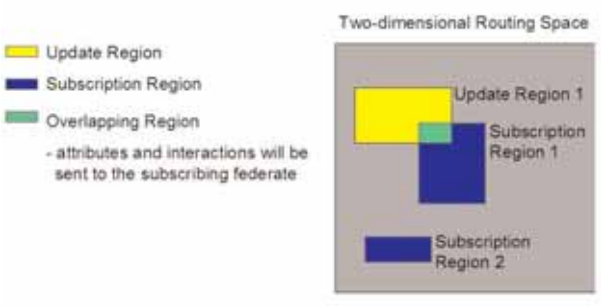


Figure 6: Data distribution management - example of routing spaces.

2.3 HLA Object Model Template Specification

The *Object Model Template* (OMT) defines the way in which federations and federates have to be documented. The HLA object models are the formal definition of the data that is transferred between federates [13] and thus are one of the main vehicles for interoperability in HLA. While the HLA interface specification provides for the technical interoperability between software systems regardless of platform and language (the 'transmission line', the object model template (OMT) defines the 'language' spoken over that line).

HLA applies an object oriented world view which is slightly different from the one known from the area of object oriented programming (OOP). In HLA, two types of classes exist: object classes and interaction classes. Object classes describe the simulated entities with their attributes. Interaction classes describe the relationships between different object classes, i.e., their interactions, and can have parameters associated with them. In contrast to OOP, HLA object models do not specify the methods of objects, since in the common case the behavioral description is nothing that needs to be transferred between federates.
It should be noted that this object oriented world view does only define how federates have to represent themselves to other federates. The object oriented world view does not dictate any internal representation inside the federate, i.e., it merely defines the interface to the outside world.

The HLA specification requires that each individual federate provides a so-called simulation object model (SOM) which is produced according to the OMT. The SOM of a federate defines its modeling capabilities in terms of what kind of data the federate is providing to other federates and what it is expecting to receive from others.

In addition to each federate's SOM, the HLA specification also requires that for each federation a so-called federation object model (FOM) is provided. The FOM is a superset of the information from the individual SOMs of the federates. It thus contains all the classes defined by the individual participants of the federation and gives a description of all shared information. The FOM can be seen as a contract among *n* simulations to satisfy the objectives of a specific federation.

In general, the object models under HLA describe:
- The set of objects chosen to represent the real world for a specific simulation/federation.
- The attributes, associations, and interactions of these objects.
- The level of detail at which these objects represent the real world, including spatial and temporal resolution.

Both the SOM and the FOM are based on a format specified in the OMT, which is a general template specifying the tables that need to be documented:
- *Object Class Structure Table*: This table lists the namespace of all simulation/federation object classes and describes their class-subclass relationships. Thus it contains the (static) object class descriptions of a federate/federation and supports hierarchical class structures. There is no mechanism for multiple inheritance.
- *Interaction Class Structure Table*: This table describes the 'dynamics' among objects by depicting all possible types of interactions among them. It also supports class-subclass relationships.
- *Attribute/Parameter Table*: This table gives detailed information about objects and interactions by specifying the 'features' of object attributes and interaction parameters in a simulation/federation.
- *Data Type Table*: This table specifies details of the data representation in the object model. This is esp. important since HLA allows the specification of complex and enumerated datatypes.
- *FOM/SOM Lexicon*: Each term listed in one of the above tables (e.g., object class names) has to be described in a verbal form in this part of the OMT. The FOM/SOM lexicon is essential to ensure that the semantics of the terms used in an HLA object model are understood and documented.

The issue of defining semantic interoperability is very difficult to solve. For a general, non-application specific architecture like HLA, it is necessary to keep application specific definitions separated from the actual architecture definition. In its predecessor DIS this guideline had not been taken account of, leading to a mixture of network protocol and application specific definitions.

In HLA a strict separation of syntactic and semantic interoperability has been followed. HLA provides the syntax for interoperability. For solving the semantic interoperability, HLA provides the framework for its definition, i.e., the templates for establishing the object models (SOM and FOM), but the task of filling the contents is left to the federation developers.

Since establishing FOMs and agreeing upon common definitions and understandings of certain terms which might be contained in a specific FOM is an effort and time consuming task, the notion of reference FOMs was soon introduced. Reference FOMs are not part of the actual HLA definition. They are usually established by a group of people from a certain niche of the simulation community, summarizing all the semantic definitions agreed upon in this group. One example for such a reference FOM is the Real-time Platform Reference FOM (RPR-FOM), which has been developed in one of the SISO PDGs. The RPR-FOM provides the definitions commonly used in the real-time simulation community.
The process of developing object models is supported by different existing tools (e.g., the Object Model Development Tool (OMDT) by AEgis, the Visual OMT by Pitch). These tools provide an intuitive user interface for creating object models and allow the conversion between the HLA 1.3 format of the OMT and the new XML-based IEEE 1516 representation.

3 Recent Developments

This section introduces recent developments and ongoing efforts that go beyond the existing HLA standard trying to improve it or to come up with alternative solutions.

3.1 COTS Simulation Package Interoperability

Soon after the creation of HLA it became obvious that its applicability would not be limited to military applications. A majority of HLA's concepts could also form the basis for a much needed simulation interoperability standard in the civilian simulation community, the manufacturing, logistics, and transportation industry being example target application areas.

Since simulation models in industry are mainly designed and developed in *commercial-off-the-shelf* (COTS) simulation packages, the prerequisites in this sector are different. As HLA itself is not focused on coupling models created in COTS simulation packages, ways have been investigated to adopt HLA for the usage with these packages [2,3,14]. Principal solutions have been developed for several packages, SLX being one of the first [15].

Ongoing standardization activities concentrate of defining standardized ways of providing HLA based interoperability for COTS packages. The challenge here is not in adopting HLA as such, but coming up with an easy and standardized approach for a certain class of simulation problems. The necessity of these efforts is derived from the different possibilities of using the HLA standard, e.g., a simple matter like entity passing from one model to another can be solved in different ways: an entity being passed could be modeled as an HLA interaction sent from a sink in the first model to a source in the second model. An equivalent solution could model the entities as HLA object instances and use ownership management services to pass the entities [16]. Both solutions are valid HLA-based solutions, but they are not interoperable.

The SISO PDG on *Commercial Off-the-Shelf Simulation Package Interoperability* is devoting its efforts to solving these problems [17]. Their approach is based on establishing so-called interoperability reference models (IRM). These IRMs describe different classes of commonly faced problems when adopting HLA for a COTS package and a standardized way to solve them. It is anticipated that COTS package vendors adopt these IRMs when creating HLA interfaces for their packages, thus achieving full interoperability for the designated problem classes.

3.2 SISO Standardization Activities

SISO is devoted on continuously creating standards and solutions for simulation interoperability issues. As already discussed, the HLA-Evolved PDG is in charge of revising the HLA standard within the cyclic 5-year review process of IEEE. Other important PDGs are briefly described in the following.

BOM PDG - *Base Object Model Specification*. Base Object Models (BOMs) can be considered as reusable packages of information representing independent patterns of simulation interplay, and are intended to be used as building blocks in the development and extension of simulations [18].
The BOM is intended as a component-based standard for describing a reusable piece part of a federation or an individual federate. BOMs provide developers and users a modular approach for defining and adding new capabilities to a federate or federation, and in quickly composing object models. BOM elements include object classes, interaction classes, patterns of interplay, state machines, and events.
SRML PDG - *Simulation Reference Markup Language*. SRML is an XML-based language for describing and executing web-based simulation models.

SRML is defined as an XML schema that adds simulation behavior to XML documents.

The intention of the PDG is to standardize SRML as the interchange of self-describing simulation models that include content and behavior, as well as standardizing the description of how that language would operate in a simulator. The fundamental premise is that an open XML-based standard language and execution environment for simulations will benefit the simulation industry in a similar way to that in which the standardization of HTML and the web browser have benefited the general computing industry.

3.3 Standardization Activities outside SISO

The *German Armed Forces Technical Centre for Communications and Electronics,* WTD 81, has taken a very active role in the evaluation of the applicability of HLA and has introduced several interesting concepts that go beyond it.
Initially, the efforts started with the sponsored development of GERTICO. GERTICO is an acronym for 'GErman Run-Time Infrastructure based on COrba'. With this effort, the WTD has propagated its preference for building HLA on top of an existing well-known standard like CORBA.

In a related effort, the WTD has devised the concept of pSISA. pSISA is the Proposed Standard Interface for Simulation Applications. Its purpose is to foster the reusability of the HLA interface code and to ease the implementation of HLA compliant applications. The major design goal is the complete encapsulation of the RTI in a object oriented shell. The application programming interface (API) accessible to the application is largely based on the object model to convey. As a result, the API is object oriented and can be built by a code generator from the HLA simulation object model (SOM) to provide C++ classes in one-to-one correspondence with the SOM object.

Simulations built on pSISA are thus expected to be independent from the underlying communication infrastructure. The latter can be based on CORBA, HLA, or any other upcoming standard [19].
Further discussions and standardization efforts outside SISO are driven by several national groups outside USA, e. g. the HLA Competence Centre in Magdeburg, Germany with its annual HLA-Forum [20].

4 Evaluation and Summary

This section attempts to give an unbiased evalua-tion of the current state of HLA and its potential for the future.

4.1 Is HLA worth the effort?

An often heard opinion about HLA is that it is too complex to use, too heavy in its performance characteristics and that there is too much overhead involved in using it. Admittedly, HLA with its federate interface specification is indeed one of the most complex standards out there. Consequently, there is certainly a rather high learning curve between getting a first glance at the standard and having the first federation running.

However, with the objectives in mind that HLA was developed for, the author has the strong conviction that it would be difficult if not impossible to devise a standard with less complexity still fulfilling all requirements HLA fulfills. The answer to the question, if it is always HLA that is needed to network two simulators highly depends on the circumstances. If there is a very high certainty that the two simulators must be networked for a single purpose only and it is highly unlikely that they ever be re-used again, then there are certainly solutions for networking them with less effort and overhead. Otherwise the effort for creating a standardized interoperable version of both simulators with a high degree of reusability is certainly worth it.

4.2 Will HLA ever become a mainstream technology?

Having been involved with HLA on both sides, academia and industry, the author postulates the thesis that HLA is far from becoming a mainstream standard for civilian simulation applications. This also concurs with the observation made by Boer [14] in his PhD thesis. One main reason can certainly be seen in the degree of simulation usage itself. Even though much progress has been made, simulation is still a niche technology not applied in day-to-day business in today's industry.

Take the example of the automotive industry: Simulation has received a major push with the digital factory initiatives of many OEMs. Still, their efforts are mainly focused on the introduction of digital planning methods. Digital planning also involves more simulation than in earlier times, but it is still tackled with a monolithic approach: All data is centrally stored in one planning database. Planning and simulation is bound to the tools of one software vendor. Interoperability between vendors and their (simulation) tools is not yet an issue which is on the demand list of the OEMs. Considering the increasing globalization and networking with supplier structures, it will become an issue rather soon.

Therefore the author has the strong believe that there is a rather good potential for usage of HLA in civilian applications. However, it will only become a main-

stream technology if the standard is incorporated into commercial simulation systems by its vendors. This is the prerequisite for making it a commodity technology that can be used like any other plug-and-play standard today.

4.3 Summary

Looking back at about 10 years of history, HLA can certainly be regarded a success. It continues to be the leading simulation interoperability standard and is constantly being maintained and improved by the community itself.

HLA's open approach to define interfaces and functionalities of an infrastructure software rather than providing a black box implementation has allowed software vendors to create their own HLA implementations and become established in the distributed simulation market.

Although HLA adoption in the non-military sector has been rather cautious, good work is underway to standardize user-friendly HLA-usage with COTS simulation packages. The adoption of the HLA standard by COTS package vendors will be the prerequisite for continuing HLA's success in this community.

References

[1] T. Schulze, S. Straßburger, U. Klein: *Migration of HLA into Civil Domains: Solutions and Prototypes for Transportation Applications*. In: SIMULATION, Vol. 73, No. 5, pp. 296-303, November 1999.

[2] S. Straßburger: *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. Ghent: SCS-Europe BVBA, 2001. ISBN 1-565552180.

[3] M.D. Ryde, S.J.E.Taylor: *Issues in Using COTS Simulation Packages for the Interoperation of Models.* In: Proceedings of the 2003 Winter Simulation Conference, eds. S. Chick, P. J. Sánchez, D. Ferrin, D. J. Morrice, pp. 772-777. Dec. 7-10, 2003. New Orleans, USA.

[4] S. Taylor, B. Gan, S. Straßburger, A. Verbraeck: *HLA-CSPIF Panel on Commercial Off-the-Shelf Distributed Simulation*. In: Proceedings of the 2003 Winter Simulation Conference, eds. S. Chick, P. J. Sánchez, D. Ferrin, D. J. Morrice, pp. 881-887. December 7-10, 2003. New Orleans, USA.

[5] M. Rabe, F.-W. Jaekel: *Non Military use of HLA within Distributed Manufacturing Scenarios*. In: Proceedings Simulation und Visualisierung '01, (Eds.) T. Schulze, V. Hinz, S. Schlechtweg. Magdeburg, 22.03-23.03.

[6] S. Straßburger, G. Schmidgall, S. Haasis: *Distributed Manufacturing Simulation as an Enabling Technology for the Digital Factory.* In: Journal of Advanced Manufacturing Systems (JAMS). Vol. 2, No. 1 (2003) 111-126.

[7] Pitch Technologies AB: *Differences between HLA 1.3 and HLA 1516*. Available online at WWW.PITCH.SE/hla/abouthla1516.asp

[8] *DoD Interpretations of the IEEE 1516-2000 series of standards, IEEE Std 1516-2000, IEEE Std 1516.1-2000, and IEEE Std 1516.2-2000*. HTTPS://WWW.DMSO.MIL/public/library/projects/hla/rti/DoD_interps_1516_Release_2.doc

[9] J. Dahmann: *HLA Tutorial*. 1997 Spring Simulation Interoperability Workshop, Mar. 3-7, 1997, Orlando.

[10] R. M. Fujimoto: *Parallel and Distributed Simulation Systems*, Wiley Interscience, 2000.

[11] *IEEE 1516-2000*: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules.

[12] *IEEE 1516.1-2000*: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) Federate Interface Specification.

[13] *IEEE 1516.2-2000*: IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT)

[14] C. A. Boer: *Distributed Simulation in Industry*. Rotterdam: Erasmus Research Institute of Management, 2005. ISBN 90-5892-093-3.

[15] S. Straßburger, T. Schulze, U. Klein, J.O. Henriksen: *Internet-based Simulation using off-the-shelf Simulation Tools and HLA*. In: Proceedings of the 1998 Winter Simulation Conference, eds. D.J. Medeiros and E. Watson, 1669-1676. SCS, Washington, D.C.

[16] S. Straßburger, A. Hamm, G. Schmidgall, S. Haasis: U*sing HLA Ownership Management in Distributed Material Flow Simulations*. In: Proceedings of the 2002 European Simulation Interoperability Work-shop. June 2002. London,

[17] *SISO-Homepage of the CSPI-PDG*. Available onlineat WWW.SISOSTDS.ORG/index.php?tg=articles&idx=More&article=43&topics=21

[18] *BOMs Homepage*. Available online at WWW.BOMS.INFO.

[19] T. Usländer, R. Herzog, K. Pixius, H.-P. Menzler: *A CORBA infrastructure plugged into a German pSISA architecture*. Simulation Interoperability Workshop, Fall 2000.

[20] *HLA-Kompetenzzentrum Magdeburg*. Available online at WWW.KOMPETENZZENTRUM-HLA.DE

Corresponding author: Steffen Straßburger
Fraunhofer Institute for Factory Operation and Automation, Sandtorstrasse 22, 39106 Magdeburg, Germany
steffen.strassburger@iff.fraunhofer.de

Received: April 27,2006
Accepted: May 19, 2006

Lookahead Computation in G-DEVS/HLA Environment

Gregory Zacharewicz, Claudia Frydman, Norbert Giambiasi
Université Paul Cézanne, Marseille, France
{*gregory.zacharewicz; norbert.giambiasi; claudia.frydman*}*@lsis.org*

In this article, we present new methods to evaluate lookahead of DEVS/G-DEVS federates participating in a HLA federation. We propose first an algorithm to compute the lookahead according to the current state of a DEVS/G-DEVS model. This solution is designed for models with lifetime function depending on one state variable. Then, we extend this computation to models with lifetime functions defined with several state variables. We use the Dijkstra graph theory search to compute the different values of state variables and a mathematical function analysis to determine the lookahead for the model states. Finally, we illustrate with an example how this solution extends the range of DEVS/G-DEVS models that can be involved into distributed simulations and we present some simulation results.

Introduction

On the one hand, G-DEVS [7] lies in its ability to develop uniform discrete event executable specifications for hybrid dynamic systems with a scientifically controlled degree of accuracy. Hence, models of continuous and discrete components can be represented with the same formalism using only a continuous time representation.
On the other hand, HLA [16] allows integrating distributed simulations, located on several computers with different operating systems, into a global simulation. HLA-compliant distributed simulations intercommunicate by exchanging messages eventually synchronized.
A first DEVS/HLA compliant environment was proposed by Zeigler et al. in [20, 21]. In this environment, distributed DEVS simulations intercommunicate through the interface (RTI) specified by HLA. In [14], Lake et al. have proposed a DEVS/HLA environment improvement by using the HLA lookahead. In [18], we have proposed a DEVS/HLA environment using the HLA lookahead without moving the management of the coupling relations from the RTI level to the federate level as in [14].

The focus of this article is to improve the DEVS/HLA environment proposed in [18]. For that purpose, in a first part, we compute a lookahead depending on the current state of DEVS models with lifetime function depending on only one state variable. It allows increasing the value of the HLA lookahead.
Then, we propose going further in the improvement of the HLA lookahead computation. This computation tackles DEVS/G-DEVS models for which state lifetimes are functions of more than one state variable. This lookahead computation is based on the shortest and longest path search algorithms in a graph.
This improvement permits to compute non-zero HLA lookahead values from models with complex lifetime functions. This result is significant because the use of greatest values for the lookahead improves the performances of distributed simulation according to literature on distributed discrete event simulation [5].
This article is organized as follows. Section 1 gives a brief recall on DEVS/G-DEVS formalisms and HLA standard. Section 2 recalls previous DEVS/HLA mapping. Section 3 exposes the approach proposed for improving the lookahead computation of the DEVS/G-DEVS HLA environment. Finally, we conclude by giving some simulation results that illustrate the performances of the proposed algorithm.

1 Recall

1.1 Generalized Discrete Event System Specification (G-DEVS)

Traditional discrete event abstraction (e.g. DEVS) approximates observed input-output signals as piecewise constant trajectories. G-DEVS defines abstractions of signals with piecewise polynomial trajectories ([7]). Thus, G-DEVS defines coefficient-event as a list of values representing the polynomial coefficients that approximate the input-output trajectory. Therefore, a DEVS model is a zero order G-DEVS model (the input-output trajectories are piecewise constants). Formally, G-DEVS represents a dynamic system (DES_N) as an n order discrete event model expressed as a structure:

$$DES_N = < XM, YM, S, \delta_{int}, \delta_{ext}, \lambda, D, Coef >$$

The following mappings are required:

$XM = A^{n+1}$, where A is a subset of integers or real numbers that represents external input events
$YM = A^{n+1}$, represents output events
$S = Q \times (A^{n+1})$, is the set of sequential model states

There Q is a set of state variables, and A^{n+1} is a subset of state variables that stores last input coefficient event.
For all total state $(q, (a_n, a_{n-1}, \dots, a_0), e)$, with e being elapsed time in S, $0 \le e \le D(S)$, and a continuous polynomial input segment $w : < t_1, t_2 > \rightarrow x$, the following functions are defined.

The internal transition function: that defines the autonomous state changes for the transient states, (i.e. states for which lifetime is a finite value):

$$\delta_{int}(S) = \delta_{int}\ (q, (a_n, a_{n-1}, \dots, a_0)) = Straj_{q,x}\ (t_1 + D\ (q, (a_n, a_{n-1}, \dots, a_0)), x)$$

with $x = a_n t^n + a_{n-1} t^{n-1} + \dots + a_1 t + a_0$, and *Straj* is the model state trajectory with

$$\vee\ q \in Q\ and\ \vee\ w : <t_1, t_2> \rightarrow x,$$
$$Straj_{q,x} :\ <t_1, t_2> \rightarrow Q$$

The external transition function: that defines the state changes caused by external events:

$$\delta_{ext}(S, e, XM) = \delta_{ext}\ (q, (a_n, a_{n-1}, \dots, a_0), e, (a'_n, a'_{n-1}, \dots, a'_0)) = Straj_{q,x}\ ((t_1 + e), x')$$

with $Coef(x) = (a_n, a_{n-1}, \dots, a_0)$ and $Coef(x') = (a'_n, a'_{n-1}, \dots, a'_0)$

Coef : function to associates n-coefficient of all continuous polynomial function segments w over a time interval $< t_i, t_j >$, to the $(n+1)$ constants values $(a_n, a_{n-1}, \dots, a_0)$ such as:

$$w(t) = a_n t^n + a_{n-1} t^{n-1} + \dots + a_1 t + a_0$$

***Coef*$^{-1}$**: the inverse function of *Coef* is applied to transform an output event in piecewise continuous polynomial trajectory:

$$Coef^{-1}(a_n, a_{n-1}, \dots, a_0) = a_n t^n + a_{n-1} t^{n-1} + \dots + a_1 t + a_0$$

The output function: triggered by autonomous state changes, it produces output events:

$$\lambda(S) = \lambda\ (q, (a_n, a_{n-1}, \dots, a_0)) = (a'_n, a'_{n-1}, \dots, a'_0)$$

The function defining the lifetime of states: that represents the maximum length or lifetime of a state, with *Otraj* is the model output trajectory:

$$D(S) = D\ (q, (a_n, a_{n-1}, \dots, a_0)) = \min\ \{\ e \mid Coef\ (\ Otraj_{q,x}\ (t_1)) <> + Coef\ (\ Otraj_{q,x}\ ((t_1 + e)\}$$
$$Otraj_{q,w} :\ < t_1, t_2 > \rightarrow Y$$

1.2 DEVS / G-DEVS Coupled Model

Zeigler has introduced, in [23], the concept of coupled model. Every basic model of a coupled model interacts with the other models to produce a global behaviour. The basic models are, either atomic models, or coupled models stored in a library. The model coupling is done using a hierarchical approach.

A discrete event coupled model (DEVS or G-DEVS) is defined by the following structure:

$$\text{MC} = < X, Y, D, \{Md \mid d \in D\}, EIC, EOC, IC, Select >$$

- X: set of external events,
- Y: set of output events,
- D: set of components names,
- Md: DEVS/G-DEVS models,
- EIC: External Input Coupling relations,
- EOC: External Output Coupling relations,
- IC: Internal Coupling relations,
- $Select$: defines priorities between simultaneous events intended for different components.

Note that to allow the coupling of different degree models ports, Giambiasi et al. have defined in [7], a coupling model component to transform the polynomial order of events exchanged.

1.3 DEVS / G-DEVS Simulator

The concept of abstract simulator has been proposed in [23] to define the simulation semantics of the formalism. The architecture of the simulator is derived from the hierarchical model structure.
The processors involved in a hierarchical simulation are *Simulators*, which insures the simulation of the atomic models, *Coordinators*, which insures the routing of messages between coupled models, and the *Root Coordinator*, which insures the global management of the simulation (e.g. Figure 1.a, without considering crosses out).
The simulation runs by exchanging specific messages (corresponding to different kind of events) between the different processors.

1.4 The High Level Architecture (HLA)

The *High Level Architecture* (HLA) is a software architecture specification for global simulations that can include a variety of simulation programs implemented on distant computers and/or to reuse existing simulations by interconnecting them ([6]).

Dr. Straßburger presents in this journal an overview of this specification ([16]).

Implementation components

An HLA federation simulation is composed of federates and a *Runtime Infrastructure* (RTI) ([11]).
A federate is a HLA-compliant program, the code of that federate keeps its original features but must be extended by other functions to communicate with other members of the federation. These functions, contained in the HLA-specified class code of *FederateAmbassador*, make interpretable by a local process the information received resulting from the federation. Therefore, the federate program code must inherit of *FederateAmbassador* to complete abstract methods defined in this class used to receive information from the RTI.

The RTI supplies services required by a simulation, it routes messages exchanged between federates. It is composed of two parts.

The *Local RTI Components code* (LRC, Figure 1b) supplies external features to the federate for using RTI call back services such as the handle of objects and the time management. The implementation is the class *RTIAmbassador*, this class is used to transform the data coming from the federate in an intelligible format for the federation. The federate program calls the functions of *RTIAmbassador* to send data to the federation or to ask information to the RTI. Each LRC contains two queues, a FIFO queue and a time stamp queue to store data before delivering to the federate.

Finally, the *Central RTI Component* (CRC, Figure 1b) manages the federation notably by using the information supplied by the FOM [16] to define Objects and Interactions classes participating in the federation. *Object class* contains object-oriented data shared in the federation that persists during the run time, *Interaction class* data are just sent and received.

A federate can, through the services proposed by the RTI, 'Publish' and 'Subscribe' to a class of shared data. 'Publish' allows to diffuse the creation of object instances and the update of the attributes of these instances. 'Subscribe' is the intention of a federate to reflect attributes of certain classes published by other federates.

HLA time management

In order to respect the temporal causality relations in the simulation stated in [15], HLA [4, 5] proposes classical conservative [1, 2] or optimistic [12] synchronization mechanisms. We focus in this article on conservative synchronisation and event driven mechanism.

We recall here the time management notions from [9, 10, 11], implemented in the 1516 compliant RTI implementation, that will be exploited in the following of this article:

Lookahead: Delay given by influencers federates to the RTI. They certify to the RTI not to emit message until their actual time plus their lookahead.

GALT (Greatest Available Logical Time): Time stamp, computed by the RTI, until influenced federates will not receive information from the federation (i.e. minimum lookahead of its influencers federates).

NMR *NextMessageRequest(t)* (*NMR*(*t*)): Federate function to ask for grant to the RTI, to deal an event time stamped *t*. If the RTI callbacks the federate with *TimeAdvanceGrant*(*t*), this federate is sure to have received all events at $t' \leq t$ and can emit events time stamped $t'' > t$.

NMRA *NextMessageRequestAvailable(t)* (*NMRA*(*t*)): differs from *NMR*(*t*) in the call-back function. *TimeAdvanceGrant*(*t*) answer to *NMRA*(*t*), ensures the federate to have received all events at $t' < t$ and allows it to emit events at $t \leq t''$. In return, the federate is not sure to have received all events time stamped *t*.

LITS (Least Incoming Time Stamp): Federate LITS is a lower bound until which the federate will receive no message, this value is calculated from its GALT and the messages in transit not received yet by the federate (i.e. messages stored in the LRC queue).

2. Previous DEVS/HLA Mapping

2.1 Components Mapping

Zeigler et al., in [20, 21, 22], present a first integration of DEVS Coordinators in a HLA-compliant architecture. They map local coupled models in HLA federates whose coordinators of higher level will have responsibility to communicate with a T*ime Manager federate*. TM routes messages between distributed coordinators. This federation of coordinators defines a global distributed coupled model.

2.2 Integrating Algorithms

As recalled in the previous section, deterministic distributed simulations require synchronization mechanisms in order to treat events in respect to causality. In consequence, DEVS/HLA federates must include integrating algorithms to communicate with the RTI (i.e. in order to handle received messages from the federation and to emit messages in a HLA format).

Zeigler et al. have proposed in [22] a first integrating algorithm of DEVS models into a HLA-compliant environment. To guarantee the global synchronization of *Local Coordinators*, this approach exploits conservative algorithm of [1,2] mechanism available in HLA [11]. In [14], Lake et al. have given a second approach for mapping DEVS into HLA that resolves *deadlock* problems encountered in the first solution. To this end, this approach notably uses the *NMRA*(*t*) service proposed by HLA instead of *NMR*(*t*). This two solutions use a zero or negligible value of HLA lookahead for every federate ([4]).

Reference [14] also introduced another approach that uses a not negligible lookahead by globally broadcasting event messages among federates and giving to each federate a global view of DEVS coupling relations. So, the federates decide to treat or not an event regarding to their history of received events and to their knowledge of coupling relations. This DEVS/ HLA environment uses a non-zero constant for the lookahead. However, some responsibilities of the RTI are transferred to the federates, what bypasses some RTI functions.

3 New G-DEVS/HLA Mapping

3.1 Components Mapping

We have proposed, in [18], an environment for creating DEVS/G-DEVS models HLA compliant. This environment proposes two-step for distributing models (and simulators associated to).

In the first step, the GDEVS coupled model is flattened. The hierarchical structure of a model is a user facility, which is not necessary adapted to a simulation purpose. This new simulation structure decreases the algorithm complexity and so increases simulation performance regarding to the hierarchical one as stated by Kim et al. and Glinsky et al. in [8, 13]. The flattening of the structure induces eliminating the crossed out *Coordinators* on Figure 1a.

In the second step, the flattened G-DEVS simulation structure is split into coupled model by federate (Figure 1b) in order to build an HLA federation (i.e. a distributed G-DEVS coupled model). The environment conforms to [22] mapping of *Local Coordinator* and *Simulators* into HLA federates, but does not use the *Time Manager federate*. It maps directly the *Root Coordinator* into the RTI. The reason of this mapping is the specification of interface (RTI) proposes services that enclose those defined in the *DEVS Root Coordinator*. Thus, the *global distributed model* (i.e. the federation) is constituted of federates intercommunicating.

The G-DEVS models federates intercommunicate by publishing/subscribing to HLA interactions that map the coupling relations of the global distributed coupled model. This information is routed between federates by the RTI in respect to time management and FOM description.

3.2 G-DEVS/HLA Integrating Algorithms

From the first algorithm of [14], we have proposed in [18] a solution integrating the use of the HLA lookahead. This solution can be applied to G-DEVS or DEVS coupled model. It considers a local G-DEVS coupled model integrated in a HLA federate. This federate communicates with other G-DEVS models within the federation. We set the federate lookahead as follows, where *S* is the set of model states:

$$Lookahead = \text{Min} \{ D(s) / s \in S \ (1) \}$$

We assume to use G-DEVS models with $D(S) > 0$ to define a non-zero lookahead.

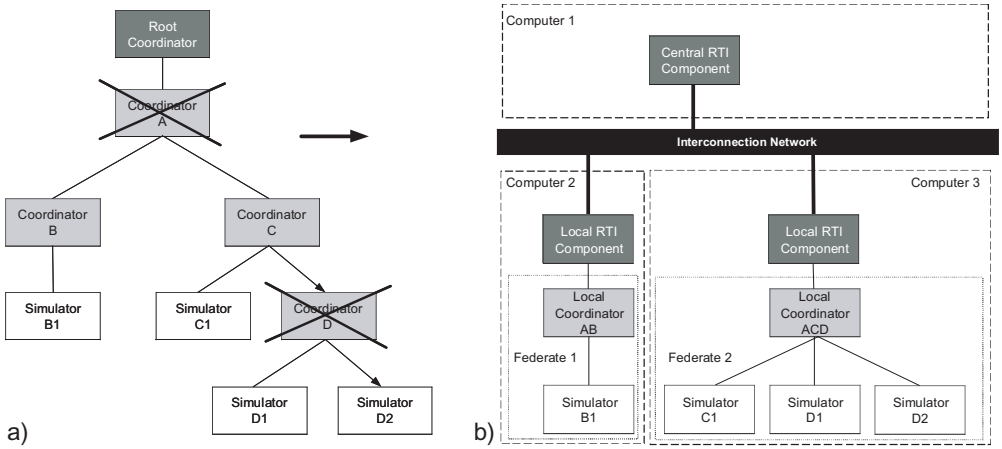


Figure 1:. Components mapping for GDEVS / HLA.

Moreover, we state that, in the case of simultaneous events, we choose to treat first the internal event, then, after having emitted an output event and done a state change, we process simultaneous external events using a confluent function.

We recall from [18], in the Figure 2 pseudo-code, the federate algorithm to communicate with the RTI. The initial settings define that the actual logical time of this federate is `Tact` and it possesses a next local event planned in its local event list at `TnextLocal`. It uses the *queryLITS*() RTI service defined in the HLA standard. Using this service, a federate can preserve a non-zero lookahead value by treating local events with timestamps earlier than LITS (that evolves depending on influencers federates data delivering behaviour and processing speed). In consequence, a non-zero lookahead federate frees of constraint federates under its influence for a period equal to the lookahead. Thus, this situation increases the parallelism of the global simulation.

It should be noted that our pseudo-code is designed for the 1516 version of HLA specification and the 1516 compliant RTI implementation.

```
Do queryLITS()
       If (TNextLocal ? LITS)
          ComputeOutput() // associated to next local internal event timestamped TNextLocal
          SendInteraction(TnextLocal) // send output without reducing federate Lookahead
          NMRA(TNextLocal) //RTI 1516 NextMessageRequestAvailable(TNextLocal) and wait for RTI answer
       Else // if (TNextLocal > LITS)
          NMRA(TnextLocal - Lookahead)
          WaitUntil(RTI responds callback)
          If (TimeAdvanceGrant (TNextLocal - Lookahead))
             queryLITS()
             If ((TNextLocal) > LITS)
                ModifyLookahead(zero)
             Else // If ((TNextLocal) ≤ LITS)
                ComputeOutput() // associated to next local internal event timestamped TNextLocal
                SendInteraction(TnextLocal) // send output without reducing federate Lookahead
             NMRA(TNextLocal) // then wait for RTI answer
          Else If (ReceiveInteraction(T'? (TnextLocal - Lookahead)) & TimeAdvanceGrant(T'))
             Do
                NMRA(T'+ε) // guaranty to have received simultaneous event timestamped T'.
                WaitUntil(RTI responds callback)
                If (TimeAdvanceGrant(T'+ε))
                   ComputeExternalTransition() // associated to external events timestamped T'
                   Break to beginning // with new TnextLocal.
                Else If ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                   AddtoSimultaneousMessageList()
             While (TimeAdvanceGrant(T') < T'+ε)
       WaitUntil(RTI responds callback)
       If (TimeAdvanceGrant(TNextLocal))
          If (output not already sends with positive lookahead)
             ComputeOutput() // associated to next local internal event timestamped TNextLocal
             SendInteraction(TNextLocal) // send output with zero federate Lookahead
          ComputeInternalTransition() // associated to internal event timestamped TNextLocal
          Do
             NMRA(TNextLocal+ε) // guaranty to have received simultaneous event timestamped TNextLocal.
             WaitUntil(RTI responds callback)
             If (TimeAdvanceGrant(TNextLocal+ε))
                ComputeExternalTransition() // associated to eventual external event(s) timestamped T
                ModifyLookahead(min of D(S))
                Break to beginning
             Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                AddtoSimultaneousMessageList()
          While (TimeAdvanceGrant(TNextLocal) < TNextLocal +ε)
       Else If ReceiveInteraction(T<TNextLocal) & TimeAdvanceGrant(T)
          Do
             NMRA(T+ε) // guaranty to have received simultaneous event timestamped T.
             WaitUntil(RTI responds callback)
             If (TimeAdvanceGrant(T+ε))
                ComputeExternalTransition() // associated to external event(s) timestamped T
                Break to beginning // with new TnextLocal.
             Else if ((ReceiveInteraction(T') & TimeAdvanceGrant(T'))
                AddtoSimultaneousMessageList()
          While (TimeAdvanceGrant(T) < T +ε)

While (Simulation not end)
```

Figure 2. Federate algorithm.

3.3 First G-DEVS/HLA Lookahead Computation Improvement

In the two last integrating algorithms [14, 18] presented in the above section, the lookahead is set to the minimum of all the states lifetime $D(s)$ of the model. Indeed, in DEVS/G-DEVS, output events are produced by output function $\lambda(s)$ associated to internal transitions δ_{int} that occur when a state lifetime $D(s)$ is elapsed. Therefore, these solutions always consider the worst case. In concrete term, the event to be earliest emitted will not have a time stamp lower than the minimum states lifetime as defined before, but this solution does not take into account the behaviour of the model (i.e. its current state).

We have proposed in [19] a first improvement in the lookahead computation in the case of G-DEVS models with only one state variable, named 'phase', and a constant lifetime function defined for each symbolic value of the phase. Lookaheads relative to the current state (of the model simulated) are computed by considering the reachable state list by external transitions δ_{ext} for each state.

The lookahead relative to the phase is set to the minimum lifetime of the current state reachable state list, what gives a relative value superior (or equal in worst case) to the solution of the previous section that was using a unique lookahead value during the simulation.

```
depth_First_Search (graph, Initial_State)
x       //   considered state
MinD    //   minimum lifetime function D(s) value
Succeding_States_List // List of reachable states
       // from a considered state by an External Trans.
x <- Initial State
Min_D <- D(x)
Succeding_States_List <- Get_Succeding_States(graph,x)
Do
   If (Existing_not_explored_State(Succeding_States_List))
    x <- First_State_not_Explored(Succeding_States_List)
    Succeding_States_List <- Get_Succeding_States(graph,x)
     Mark_Explored(x)
     MinD <- min(D(x), Min_D)
  Else // x is a leaf or next states already explored
   // Go up to 1st preced.state with not-explored child
   While (x=! Initila_State
  !(Existing_Not_Explored_State(Succeding_States_List)))
    Do
     x <- Prededing_State(x)
     Succeding_States_List <-
Get_Succeding_States(graph,x)
    EndWhile

  If (Existing_Not_Explored_State(Succeding_States_List))
    x <- First_State_Not_Explored(Succeding_States_List)
    Succeding_States_List <- Get_Succeding_States(graph,x)
      Mark_Explored(x)
      MinD <- min(D(x), Min_D)
    EndIf
 EndIf
While (x=! Initial_State &&
  !(Existing_Not_Explored_State(Succeding_States_List)))
```

Figure 3: G-DEVS model current state relative lookahead.

In more details, for a G-DEVS model, the next output event to be emitted is associated to an upcoming internal transition. As a result, we have to find the sooner next internal transition that could be executed from the current state of the model. We propose to use a graph search to explore and to determine for each state all reachable states by a sequence of external transitions.

For that purpose, we defined an algorithm that explores, from a considered state, the graph of reachable states in order to compute a state relative lookahead. In Figure 3, we present a pseudo-code algorithm of this solution that is based on oriented graphs classical depth-first search algorithm.
We use the list of adjacencies of a considered node of the graph to obtain the *Succeding_States_List*. This algorithm computes the relative lookahead for an *Initial_State*, which is equal to *Min_D* at the end of the graph exploration (i.e. *Min_D* is the minimal $D(S)$ of reachable states).

Let us focus on Figure 4 that represents a simple DEVS atomic model (i.e. a G-DEVS model of zero order) with the graphical representation of [17]. The discrete state of the models considered in this subsection is defined only by the phase state variable (with values represented by circles). For that reason, a lifetime value can be associated to each *phase* value (referenced by numbers inside circles). Solid arcs represent external transitions δ_{ext} ; for instance, mark com?o1 on an arc of this type describes that the model state will transit by receiving an input event of o1 value on the input port com. Dotted arcs represent internal transitions δ_{int} ; if it elapses lifetime length in the source phase of this type of arc, mark out!set shows, for instance, that the model state will transit and emit an output event of set value on output port out. Triangles represent input and output ports.

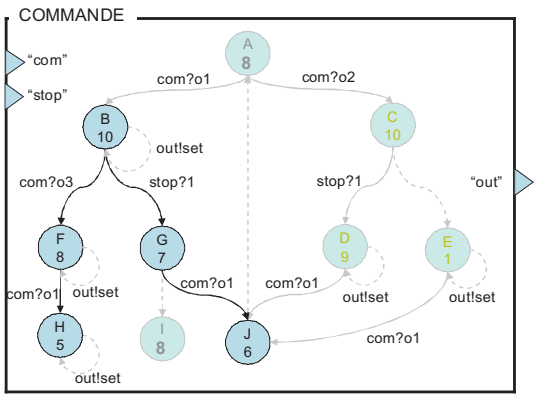


Figure 4: DEVS model current state relative lookahead

If we consider an absolute lookahead not depending on the current state, the lookahead of the Figure 4 example is equal to one time unit. If B state is the current state of the example, considering the current state relative lookahead, the lookahead can be increased to five times units (Figure 4 minimum lifetime of not shadowed states). Moreover, the computation of all lookahead values is done before run time and so does not affect simulation performance.

The restriction of the computation presented in this subsection comes from the fact that it can only be applied to models with $D(S)$ functions depending on only one state variable called the phase. Because DEVS/G-DEVS formalism can express more complex models, we propose in the next subsection to generalise this computation for models with $D(S)$ function depending on several state variables.

3.4 Second G-DEVS/HLA Lookahead Computation Improvement

In the following, we define an extension of the computation of the lookahead in order to consider G-DEVS models with state defined by

$$S = Q \times (A^{n+1}) \quad \text{with}$$

$\boldsymbol{Q}$ **(*phase*, *sigma*,** $\boldsymbol{B^n}$**)** where *sigma* is the lifetime function $D(S)$ of the current state, *phase* is a state variable with symbolic values and B^n $(b_0, \ldots, b_n)$ is a (n)-tuple set of discrete state variables.

$\boldsymbol{A^{n+1}}$: state variables $(a_0,\ldots, a_{n+1})$ stores, the $(n+1)$-tuple polynomial coefficients of the last external event occurred with A subset of real numbers or integers ([7]).

The ***phase*** defines explicit subset of state set, which allows representing graphically the DEVS/G-DEVS models as stated in [17].

The $\boldsymbol{B^n}$ n-tuple finite set of integer valued state variables completes the definition of the considered G-DEVS model state.

The values of the state variables are modified by the δ_{ext} and δ_{int} transition functions. Note that we do not consider the elapsed time in the current state to change the values of the next state.

Moreover, in the G-DEVS models considered, each lifetime is a mathematical function of $\boldsymbol{D}(phase, B^n)$, it does not depend on A^{n+1} (e.g. Figure 5a).

Path search in G-DEVS models

The lookahead is the minimum delay to emit an output event, which corresponds to the earliest next $\lambda(s)$ among the reachable phase values by a sequence of δ_{ext}.

Path search algorithms seem to be suited to analyze the variation of state variables involved as parameters of $\boldsymbol{D}(s)$, because the considered G-DEVS models can be represented by nodes/arcs. The only mismatch comes from classical graph path search al-gorithms only consider one variable (i.e. that is the path weight between two nodes), but G-DEVS models graphs can possess more than one state variable. In the considered model, there are n state variables B.

A key to this mismatch is to decompose a considered G-DEVS model (e.g. Figure 5a) into as many sub-models as the model contains state variables B. From each submodel with $S = ((phase, sigma, B), A^{n+1})$, we create an oriented graph by representing the phase values of the G-DEVS model as nodes and the δ_{ext} as edges (e.g. Figures 5 b, c, d).

The edges of a G-DEVS sub-model are weighted by the part of the δ_{ext} function that handles the considered B state variable. Therefore, it implies that the state variables of B^n are independent in the expression of δ_{ext} (i.e. each B variable must only be dependent on constant values or on itself in the δ_{ext} functions). We can apply on the obtained oriented graphs a path search algorithm to track the variations of state variable B.

Dijkstra path search

Considering an oriented graph (obtained from a G-DEVS sub-model) and the phase value $phase_i$, we define the following function, which computes the shortest path (in terms of a considered b_j of B^n) to reach each other $phase_k$ by a sequence of δ_{ext} :

$$\text{ShortestPath}(phase_i, b_j, phase_k) = \min\ b_j \text{ in } phase_k$$
$$/\ \text{considering an initial value of } b_j \text{ in } phase_i$$
$$\text{and } k \in \text{ reachable phase value list of } phase_i$$

For example in Figure 5b, the shortest path from phase A to the others phase values, considering state variable b_1, is 10 for reaching phase B, 2 for C, 10 for D and not defined for E because it is not linked from A by external transition.

To implement the ShortestPath function, we applied the Dijkstra Algorithm ([3]) that fulfils requirements of the function. The limitation is that this algorithm is not suited for graphs that contain circuits with edges of negative weight; indeed the looping of such circuits decreases iteratively the weight of the path. As a result, the considered G-DEVS models must contain only B variables defined on $\boldsymbol{R}^+$ and δ_{ext} functions that only increment the state variables of B^n. We notice that others algorithms (e.g. Warshall and Floyd) allow the use of negative weight edges but the studied graphs still must no contain negative weight circuit.

We use the modified Dijkstra Algorithm (by changing the values searched from Min to Max) to find the longest path from a considered phase value to all others. This search computes the state variable B maximal values for each reachable phase value. It implies a restriction on graphs type; it can be applied only to acyclic graphs (i.e. without circuits) because finding the longest path in a cyclic graph has been shown to be an NP-hard problem. Notice that cyclic graphs can be considered only if all state lifetimes $\boldsymbol{D}(s)$ contain no decreasing part since we do not search for the maximum of the state variables.

State lifetime $\boldsymbol{D}(s)$ analysis

Using min/Max values of state variables of B^n (obtained from shortest/longest path computation), we exploit mathematical backgrounds to study the variations of state lifetime $\boldsymbol{D}(s)$. We consider $\boldsymbol{D}(s)$ as real-valued functions of several variables. The functions must be continuous, defined and derivable in all points of the state variables values range in order to determine their minimal value regarding to min/Max values of the state variables of B^n. To respect these definitions, we bound the study to linear state lifetimes $\boldsymbol{D}(s)$ defined by independents state variables (i.e. a linear function of several variables $b_1, b_2, \ldots, b_n$) is described in the following:

$$f(b_1, b_2, \ldots, b_n) = \alpha_0 + \alpha_1 b_1 + \ldots + \alpha_n b_n$$
with $\alpha_0, \alpha_1, \ldots, \alpha_n$ *constant real values*

Taking into account the restrictions on state lifetime $\boldsymbol{D}(s)$, we can compute a minimum value of state lifetime of a reachable phase value from a considered one regarding to min/Max values of the state variables. More formally, for a phase value $phase_k$ and the state variables $b_1, b_2, \ldots, b_n$, the state lifetime is defined by the following formula:

$$\boldsymbol{D}((phase_k, sigma, B^n), A^{n+1}) = \alpha_0 + \alpha_1 b_1 + \ldots + \alpha_n b_n$$

if $\alpha_i < 0$ consider maximum value of b_i, $i = 0, \ldots, n$

if $\alpha_i \leq 0$ consider minimum value of b_i, $i = 0, \ldots, n$

By repeating this computation for each reachable $phase_k$ from a considered $phase_i$, we calculate the lookahead of the considered phase value equal to the minimum of all the reachable phase value state lifetime $\boldsymbol{D}(s)$ (we do not consider *sigma* and A^{n+1}) by:

$$Lookahaed\ (phase_i) = \min \boldsymbol{D}(phase_k, \min/\mathrm{Max}\ (B^n))$$
for all k from reachable phase value list of $phase_i$

If some G-DEVS models federates in a G-DEVS coupled model federation do not respect the restriction on state lifetime $\boldsymbol{D}(s)$ and on the state variables variations or if state lifetime $\boldsymbol{D}(s)$ computation concludes to a negative value result, then no minimum of state lifetime $\boldsymbol{D}(s)$ can be computed. We set, in that case, the lookahead of these G-DEVS model federates equal to a minimum value ε negligible regarding to values taken by state lifetime $\boldsymbol{D}(s)$. Thereby, the simulation is constrained and slowed (lookahead).

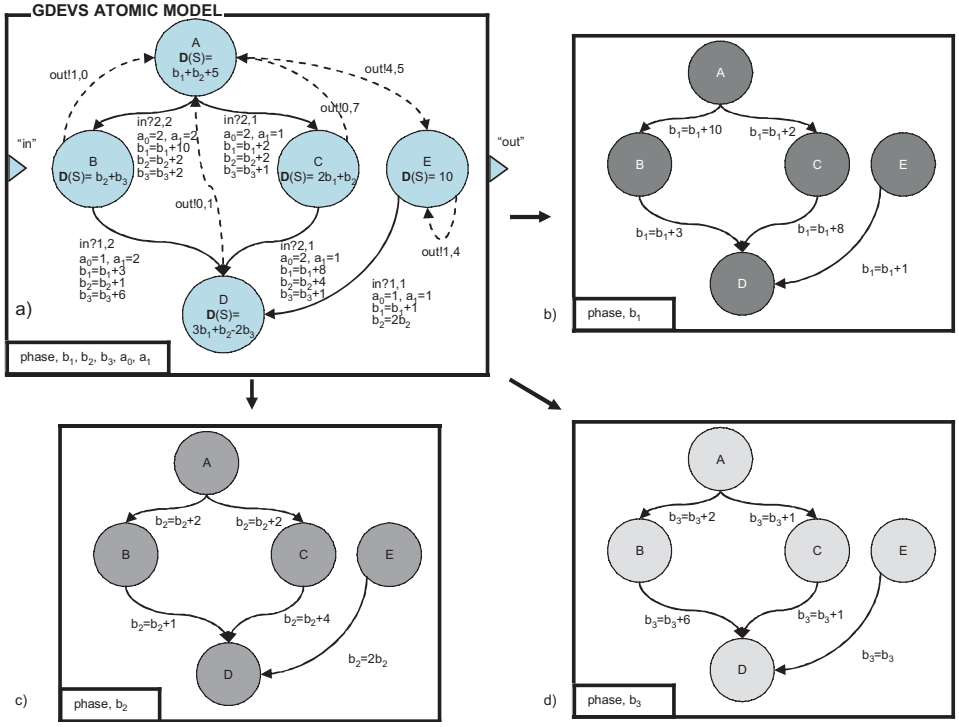


Figure 5: Lookahead computing in G-DEVS model with complex lifetime function.

Lookahead computation example

Figure 5a-example is an order 1 G-DEVS model. We consider an initial state with phase =A and $b_1=b_2 = b_3= a_0 = a_1= 0$.

We focus on the computation of the lookahead of phase A.

As a result, we determine the reachable phase values from A by a sequence of external transitions that are B, C, and D. The state lifetime $\boldsymbol{D}(s)$ values of these phase values are dependent on the external transition passed from A to attain the considered phase value.
For instance, state lifetime $\boldsymbol{D}(s)$ of phase D is equal to $3b_1+b_2-2b_3$. It implies to consider the minimum value of b_1 and b_2 and the maximum value of b_3. Djikstra algorithms find out the extreme values of the state variables for each phase value.
We focus first on the submodel represented in Figure 5b that only considers the phase and b_1. From the initial state, we compute that b_1 minimum value is 10 time units in phase B, 2 for C, 10 for D and not defined for E. By repeating the same process, we compute b_2 minimum, values in Figure 5c. The minimum of b_2 is 2 for B, 2 for C, 3 for D and not defined for E.
In Figure 5d: b_3 minimum is 2 for B, 1 for C, 2 for D and not defined for E. Because the $\boldsymbol{D}(s)$ function of phase D contains a subtraction, it is necessary to compute b_3 maximum value; in Figure 5d that is equal to 8 for D. Using these values, we compute the minimum values of $D(s)$ functions for each reachable phase value from A:

min $\boldsymbol{D}$(A, min/Max (B^n)) = $b_1 + b_2 + 5 = 0+0+5 = 5$

min $\boldsymbol{D}$(B, min/Max (B^n)) = $b_2 + b_3 = 2+2 = 4$

min $\boldsymbol{D}$(C, min/Max (B^n)) = $2b_1 + b_2 = 4+2 = 6$

min $\boldsymbol{D}$(D, min/Max (B^n)) = $3b_1+b_2-2b_3 = 30+3-16=17$

The lookahead of phase A is equal to the minimum $\boldsymbol{D}(s)$ of all reachable phase values, thus set to 4 time units.
Using the same approach, we can compute the HLA lookahead for all phase values of the model. The lookahead is employed in the communication algorithm defined in [18] and recalled in Figure 2.
The limitation of this solution comes from its non-generic capabilities to handle all DEVS formalised models. In more details, this improvement does not allow to extend the lookahead computation to all kinds of DEVS/G-DEVS models; e.g. models with non-explicit phase, state variables defined on $\boldsymbol{R}$ and nonlinear $\boldsymbol{D}(s)$ functions are not considered in this study.

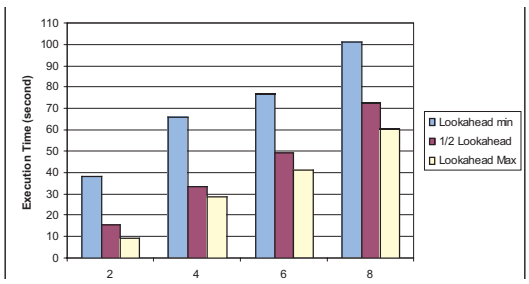


Figure 6. Execution time versus Lookahead value

4 Simulation results

We have implemented the algorithm of Figure 2 on two distributed Pentium 4-based computers with 2.4 GHz, 256 Mb RAM, Windows XP OS, interconnected by a 10 Mbps LAN. We ran G-DEVS coupled models federations, of 2, 4, 6 and 8 G-DEVS federates distributed on the two computers, in order to measure the influence of the lookahead value on the execution time. In the tests, each federate contained a G-DEVS atomic model and published/subscribed to coupling HLA interactions to define the federation as a 'closed-chain' of coupled model federates. The code was developed in Java and the RTI (running on a third similar computer of the LAN) was the pRTI1516 of Pitch.
Figure 6 shows that G-DEVS federation execution is speeded up using a maximum lookahead (computed from the lifetime of the G-DEVS models phase as presented in 3.3 and 3.4). This assertion is done regarding the run time of the same federates models with half-reduced lookahead (representing the first solution proposed in 3.2 with a unique lookahead) and with negligible (min) lookahead (representing the previous solutions recalled in 2.2).
The experiment also deduces that the speedup is nearly linear in number of federates. Thus, federates with a negligible lookahead value always produce an important overhead regarding federates with a maximal lookahead. This overhead increases with the federation size and appears clearly in Figure 6 to slow significantly the simulation as stated theoretically.

5 Future work

The lookahead computation algorithm is still under the scope of our studies. We are working on the improvement of this computation, particularly in the case of computing a longest path. Because computing the longest path is restrictive on the class of G-DEVS models that can be handled, we try to compute it using algorithms of estimate for the longest path proposed in the literature.
We are also studying other kind of state lifetime $D(s)$ functions to be considered (e.g. real valued of several variables interaction functions, distance functions, constrained functions).

Conclusions

In this article, we have presented a new HLA lookahead computing algorithm for distributed G-DEVS/DEVS models that uses the Dijkstra path search in a graph.

It considers G-DEVS models with explicit phase and $\boldsymbol{D}(s)$ depending on several state variables instead of previous solutions that were considering DEVS models with $\boldsymbol{D}(s)$ depending only on one state variable. In addition, a benchmark experiment has been performed to confirm the speedup of the G-DEVS federation execution due to the new lookahead computation.
Finally, this improvement extends the class of G-DEVS models that can be involved in a G-DEVS federation. These models can be, more generally, coupled with heterogeneous HLA-compliant programs that respect, of their sides, the distributed time management constraints and the event exchanged format.

References

[1] R. E. Bryant: *Simulation of packet communication architecture computer systems.* Technical Report MIT/LCS/TR-188, MIT, 1977.

[2] K. M. Chandy, J. Misra: *Distributed simulation: A case study in design and verification of distributed programs*. IEEE Transactions on Software Engineering, 5(5):440-452.

[3] E.W. Dijkstra: *A note on two problems in connexion with graphs.* Numerische Mathematik, 1:269-271, 1959.

[4] R. M. Fujimoto: *Zero lookahead and repeatability in the high level architecture.* In Spring Simulation Interoperability Workshop (SIW), number 97S-SIW-046, Orlando, FL, 1997.

[5] R. M. Fujimoto: *Time management in the high level architecture.* Simulation, 71(6):388-400.

[6] R. M. Fujimoto: *Parallel discrete event simulation*. Wiley Interscience, New York, 2000.

[7] N. Giambiasi, B. Escude, S. Ghosh: *G-DEVS A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems.* Transactions of the Society for Computer Simulation International, 17(3):120-134, 2000.

[8] E. Glinsky, G. A. Wainer: *DEVStone: a Benchmarking Technique for Studying Performance of DEVS Modeling and Simulation Environments.* DS-RT 2005: 265-272, Montreal CA, 2005.

[9] IEEE std 1516-2000: *IEEE Standard for Modeling and Simulation (M&S) - High Level Architecture (HLA) - Framework and Rules*. IEEE, NY, NY, USA, 2001.

[10] IEEE std 1516.1-2000. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Object Model Template (OMT) Specification*. IEEE, NY, NY, USA, 2001.

[11] IEEE std 1516.2-2000. *IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification.* IEEE, NY, NY, USA, 2001.

[12] D.R.Jefferson: *Virtual Time*. ACM Trans. Prog. Lang. and Syst. 7(3):404-425, 1985.

[13] K. Kim, W. Kang, B. Sagong, H. Seo: *Efficient Distributed Simulation of Hierarchical DEVS Models: Transforming Model Structure into a Non-Hierarchical One.* 33rd ASS, 2000 Washington, D.C. p. 227, 2000.

[14] T. Lake, B.P. Zeigler, H.S. Sarjoughian, J. Nutaro: *DEVS Simulation and HLA Lookhead.* In Simulation Interoperability Workshop (SIW), number 00S-SIW-160, Orlando, FL, 2000.

[15] L. Lamport: *Time, clocks and the ordering of events in a distributed system*. Communication of the ACM, 21(7):558-565, July 1978.

[16] S. Straßburger: *Overview about the High Level Architecture for Modelling and Simulation and Recent Developments"*. SNE 16/2, Special Issue *Parallel and Distributed Simulation Methods and Environments*, September 2006, pp 5-14.

[17] H. S. Song, T.G. Kim: *The DEVS framework for discrete event systems control*. In 5th Conf. on AI, Simulation and Planning in High Autono mous Systems:228-234, Gainesville, USA, 1994.

[18] G. Zacharewicz, N. Giambiasi and C. Frydman: *Improving the DEVS/HLA Environment*. In DEVS Integrative M&S Symposium, DEVS'05, 2005 SCS Spring Simulation Multiconf. SpringSim'05, San Diego, CA, USA, April 3-7 2005.

[19] G. Zacharewicz, N. Giambiasi and C. Frydman: *A New Algorithm for the HLA Lookahead Computing in the DEVS/HLA Environment.* In SISO European Simulation Interoperability Workshop (EUROSIW), number 05E-SIW-028, Toulouse, France, 2005.

[20] B. P. Zeigler, J.S. Lee: *Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment.* SPIE, 3369 (Enabling Technology for Simulation Science II):49-58, A.F. Sisti Ed, 1998

[21] B. P. Zeigler B.P., G. Ball, and al.: *The DEVS/HLA Distributed Simulation Environment And Its Support for Predictive Filtering.* Technical Report Dept., DARPA Contract N6133997K-0007, UA, Tucson, AZ, 1998.

[22] B. P. Zeigler, G. Ball, H.J. Cho and J.S. Lee: *Implementation of the DEVS formalism over the HLA/RTI: Problems and solutions.* In Simulation Interoperation Workshop (SIW), number 99SSIW-065, Orlando, FL, 1999.

[23] B. P. Zeigler, H. Praehofer, T.G. Kim: *Theory of Modeling and Simulation.* 2nd Edition, Academic Press, New York, NY, 2000.

Corresponding author: Gregory Zacharewicz,
gregory.zacharewicz@lsis.org
Gregory Zacharewicz, Claudia Frydman, Norbert Giambias
LSIS, Université Paul Cézanne, Marseille, France
UMR CNRS 6168 Université Paul Cézanne
Marseille, France
{norbert.giambiasi; claudia.frydman}@lsis.org

Received: March 4, 2006
Revised: June 8, 2006
Accepted: July 6, 2006

Parallel Simulation Techniques for DEVS/Cell-DEVS Models and the CD++ Toolkit

Gabriel Wainer, Ezequiel Glinsky, Carleton University, Ottawa, Canada
WWW.SCE.CARLETON.CA/faculty/wainer

DEVS is a sound formal modelling and simulation (M&S) framework based on generic dynamic system concepts. Cell-DEVS is a formalism for cell-shaped models based on DEVS. This work presents a new simulation technique for execution of DEVS and Cell-DEVS models in parallel/distributed environments. The parallel simulator is based on Time Warp, and developed as a new simulation engine for CD++, an M&S toolkit that implements DEVS and Cell-DEVS theories. The technique uses a non-hierarchical approach that simplifies the structure of the simulator and reduces the communication overhead. The results obtained allowed us to achieve considerable speedups.

SNE 16/2, September 2006

Introduction

The widespread use of M&S is leading to execution of larger and more complex systems. One way of handling this complexity is to devote more memory and processor cycles through the use of multiple resources [1]. *Parallel discrete event simulation* (PDES) studies the execution of discrete event models in parallel or distributed computers [1]. The main concern of this community was to reduce execution time of applications by using multiple processors, and a large number of synchronization algorithms were developed [1]. Most of these algorithms are based on Chandy-Misra-Bryant [2, 3] and Time Warp [4], which introduced fundamental ideas that are still used.

Another way to attack these problems considered using the DEVS formalism [5] as the modelling framework for PDES [6, 7, 8, 9]. DEVS is a sound formal framework based on generic dynamic systems concepts that supports provably correct, efficient, event-based simulation. DEVS enables the construction of models in a hierarchical, modular fashion, allowing component reuse and reducing development and testing time.

Cell-DEVS [10] combines cellular automata [11] with DEVS theory, improving timing definition. Individual cells are defined as DEVS models and coupled to form complete cell spaces. *CD++* [12] is an M&S tool that implements DEVS and Cell-DEVS theory. A hierarchical, conservative parallel simulation mechanism has been implemented in CD++, showed improved results for both DEVS and Cell-DEVS [8]. However, its degree of parallelism and speedups are bounded. Here, we introduce a new technique for optimistic simulation of large, complex DEVS and Cell-DEVS models in CD++. The technique combines the *Time Warp* synchronization mechanism and the DEVS abstract simulators. We introduce two new classes of DEVS processors that carry out the simulation efficiently across multiple processors. In our approach, the hierarchy of the simulation objects is flattened to reduce communication overheads, using a flat simulation approach that eliminates the need for intermediate coordinators [7, 13]. Consequently, it reduces the overhead of message passing, improving the overall performance of the simulation.

1 DEVS and Cell-DEVS

The *DEVS* formalism [5] provides a framework for the definition of hierarchical modular models, allowing for model reuse and development time reduction. A DEVS model is described as a composite of models, each of them being behavioural (*atomic*) or structural (*coupled*). P-DEVS [6] provides a flexible way of dealing with simultaneous events. An atomic DEVS model is defined as:

$$M = < X_M, Y_M, S, \delta_{ext}, \delta_{int}, \delta_{con}, \lambda, ta >$$

At any given time, an atomic model is in state s during a period defined by $ta(s)$. When that time expires, an internal transition takes place; the system outputs the value $\lambda(s)$ and then it changes to the state specified by $\delta_{int}(s)$. If one or more external events (X_M) occur before $ta(s)$, the new state is given by the external transition function, $\delta_{ext}(s, e, X_M)$, which uses a bag of events to allow multiple events to be processed simultaneously. If external and internal transitions are in conflict (an external event is received at this time), the new state is given by $\delta_{con}(s)$.

Coupled models are defined as a set of basic components (atomic or coupled) interconnected through the model's interfaces. The model's coupling defines how to convert the outputs of a model into inputs for the others. A coupled model is:

$$CM = < X, Y, D, \{M_d \mid d \in D\}, EIC, EOC, IC >$$

X is the set of inputs, Y is the set of outputs, D is a set of the component names; for each $d \in D$, M_d is a basic DEVS model; the external input couplings set (*EIC*) defines how to connect external inputs to components; the external output couplings set (*EOC*) defines how to connect component to external outputs; and the internal couplings set (*IC*) defines how to interconnect components.

Cell-DEVS [10] allows the specification of executable cell spaces with explicit timing delays, which allows easy definition of complex behaviour in physical systems. A parallel Cell-DEVS atomic model [14] can be defined as:

$$TDC = < X^b, Y^b, S, N, d, \tau, \tau_{con}, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, D >$$

Each cell uses a set of N inputs to compute the next state. These values are received through a well-defined interface (X^b and Y^b), activating a local function (τ, τ_{con}), which uses the cell's inputs and present state (S). d defines the kind of delay and D is the state's duration function. The model advances through the activation of δ_{int}, δ_{ext}, δ_{con}, λ, and D, as in other DEVS models.

After the behaviour for a cell is defined, the complete cell space will be constructed by building a coupled Cell-DEVS model:

$$GCC = < X_{list}, Y_{list}, X, Y, n, \{t_1, ..., t_n\}, N, C, B, Z >$$

The cell space is a coupled model composed of an n-sized array of $t_1 \times ... \times t_n$ atomic cells (C). Each of them is connected to the cells defined by the neighbourhood (N). As the cell space is finite, the border (B) can have different behaviour than the rest of the space. X is the set of input events and Y is the set of output events. X_{list} and Y_{list} are the lists of input and output couplings. Finally, the Z function defines the internal and external coupling of cells in the model. CD++ [12] implements DEVS and Cell-DEVS formalisms. Atomic models can be defined in C++ or an interpreted graphical notation, while coupled and Cell-DEVS models are defined using a built-in specification language.

2 Optimistic PDES of DEVS Models

As mentioned earlier, we want to combine advanced DEVS simulators with PDES. In PDES, the simulation is subdivided in smaller parts running on different nodes. Each subpart is a sequential simulation, usually referred to as a *Logical Process* (LP), which groups one or more objects running in a node [1]. Simulation objects communicate through timestamped messages. Objects located on different LPs have to traverse the boundaries of the LPs to interact with each other. *Synchronization* is key in these cases, as the difference in execution speeds can mix events with different timestamps, causing causality problems (i.e., an event in the past affects the present). *Conservative* synchronization algorithms avoid violating causality constraints at all times [2, 3]. Although many conservative algorithms are currently found in real-world applications, they have two main disadvantages ([1]): it is not possible to take full advantage of the concurrency in the application, and the simulator has to be specifically designed to exploit concurrency, leading to a complex, tedious design process.

Optimistic synchronization, instead in [4], allows some causality errors to occur, but provides a detection/ recovery mechanism. Optimistic algorithms enable greater degree of parallelism, and they do not rely on application-specific data.

2.1 The CD++ simulator

CD++ was built as a class hierarchy in C++, where each class corresponds to a simulation entity using the basic concepts defined in [5]. There are two basic abstract classes: *Model* and *Processor*. The former is used to represent the behaviour of the atomic and coupled models, while the latter implements the simulation mechanisms. *Simulators* manage the atomic models. *Coordinators* manage coupled models. The *Root Coordinator* manages global aspects. CD++ was redesigned to provide parallel execution of DEVS and Cell-DEVS [8]. The parallel version of CD++ was built on top of Warped [15], a simulation kernel that provides an implementation of *Time Warp* with different optimizations. Warped uses the MPI message passing standard for communication [16]. Although Parallel CD++ showed speedups in the execution of both DEVS and Cell-DEVS models, a single *Root Coordinator* still acts as a global scheduler for every node in the simulation.

Another problem is that most DEVS simulators are hierarchical, creating a one-to-one correspondence between model components and simulation objects. Since the simulation advances by exchanging messages between simulation objects, communication costs can be considerable. Flat simulation mechanisms, instead, try to reduce this overhead by simplifying the underlying simulator structure, while keeping the model definition and preserving the separation between model and simulator.

Flat simulation approaches have been implemented in distributed [7] and stand-alone [13] environments.

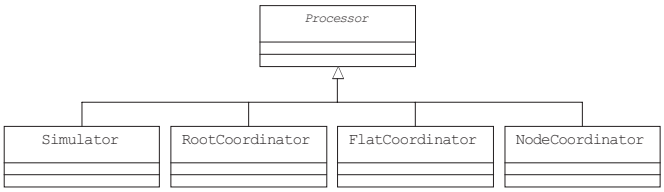


Figure 1: New Processor class hierarchy [17]

There are two basic abstract classes in CD++: *Model* and *Processor*. The former is used to represent the behavior of the atomic and coupled models, while the latter implements the simulation mechanisms. *Simulators* manage the atomic models. *Coordinators* manage coupled models. The *Root Coordinator* manages global aspects (starting/stopping the simulation, communication with environment). This reflects the clear distinction between model and simulator. We took advantage of this separation of concerns by focusing on the processors' class hierarchy only. All classes inheriting from model remain unchanged from those defined in earlier versions of the tool, allowing direct reuse of existing models.

Two new classes are introduced [17]: *Flat Coordinator* (FC) and *Node Coordinator* (NC). Additionally, we modified the *Simulator* and *Root Coordinator* classes (Figure 1). The algorithms we defined are based on those in [6] and [14]. The *Root Coordinator* only handles I/O operations, and starts/stops the simulation. The NC is in charge of synchronization and time management for the LP. The FC is responsible for receiving, translating, and sending messages between its descendants, using a flat data structure with coupling information for every component.

In order to run the model on a distributed environment, we need to indicate the nodes that will participate in the simulation, and how they are allocated to each processor. Figure 2 shows an example where two atomic models run on *Processor 0*, three atomic models run on *Processor 1*, and the remaining two models on *Processor 2*. Node coordinators handle inter-processor communication.

During the creation and registration of each Simulator object, they are associated to the corresponding LP. NCs can communicate with each other using inter-LP messaging. The Root Coordinator executes on one LP, and it forwards messages from the environment to the corresponding NC. On the other hand, when a NC processes an output that must be sent back to the environment, it is sent to the Root Coordinator.

2.2 Abstract simulators in CD++

We will describe the simulation mechanism by presenting the behaviour of each *Processor*. The simulation is message-driven. Different messages can be exchanged among processors: `init` (initialization), `q` (external input), `y` (output), `@` (collect), `*` (internal transition), and `done`.

Simulator

A Simulator is created for each atomic component or cell. It is responsible of executing the functions of the associated model, as follows.

```
1 when (init, 0) message is received
2    initialize model's variables
3    t_L = 0
4    t = ta (s)
5    send (done,t) to parent flat coord.
6 end when
```

When the initialization message is received, variables are initialized (lines 2 and 3) and the simulator informs its parent the time of the next scheduled internal transition (line 5).

When a simulator receives a collect message `(@,t)`, it generates an output, which is sent to the parent flat coordinator (lines 3-5).

When an external message (`q`, `t`) is received, it is stored in the bag of external events (line 12). These messages will be used later, when the external transition is triggered.

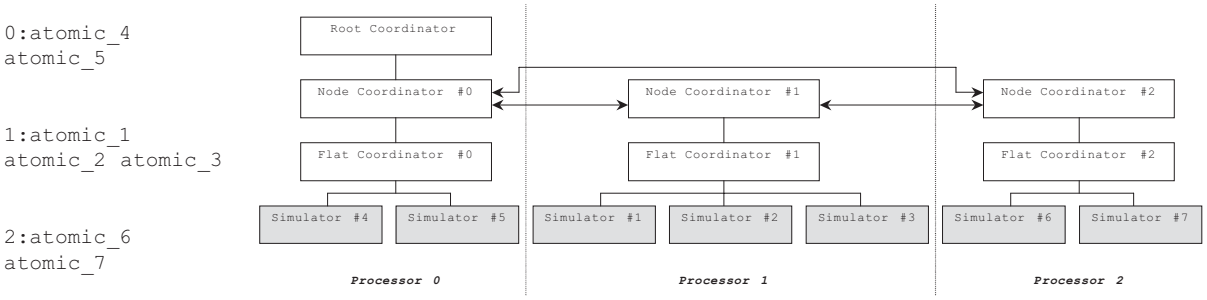


Figure 2. Model partition file for CD++.

```
 1 when a (@, t) message is received
 2   if t = t_N then
 3    y = λ(s)
 4    send (y,t) to parent flat coord.
 5    send (done,t) to parent flat coord.
 6   else
 7    raise error
 8   end if
 9 end when
10 /*********************************/
11 when a (q, t) message is received
12   add event q to the bag
13 end when
```

An internal message `(*,t)` triggers the execution of a transition function. The simulator executes one of the three transition functions based on `t` (elapsed time since the last transition), t_N (time of the next scheduled transition), and the contents of bag of external events, as showed in the following code fragment. If $t<t_N$ (lines 2-6) and the bag of external events contains at least one element, the external transition is executed. If $t=t_N$ (lines 7-9), it is time for the internal transition. However, if the bag is not empty and $t=t_N$ (lines 10-13) the confluent transition has to be executed. In every case, after executing the corresponding transition, a done message is sent to the parent flat coordinator, indicating the next scheduled transition time (lines 17-19).

```
 1 when a (*, t) message is received
 2   case t_L ≤ t < t_N
 3      e = t - t_L
 4      s = δ_ext (s, e, bag)
 5      empty bag
 6   end case
 7   case t = t_N and bag is empty
 8      s = δ_int (s)
 9   end case
10   case t = t_N and bag is not empty
11      s = δ_con (s, bag)
12      empty bag
13   end case
14   case  t > t_N or t < t_L
15      raise error
16   end case
17   t_L = t
18   t_N = t_L + ta (s)
19   send (done, t_N) to parent flat coord.
20 end when
```

Flat coordinator

A *flat coordinator* has one or more *simulator children* (in charge of the atomic components), and one *parent node coordinator*. The *flat coordinator* uses model coupling information to translate output events into input events. Additionally, it synchronizes models that are imminent in this logical process using a structure called synchronize set.

When the initialization message is received, the *flat coordinator* forwards it to all its children to complete the initialization phase (lines 3-5). Using the `done` messages received from them, the minimum time of next change is computed and communicated to the parent node coordinator (lines 6-8).

```
1 when (init, 0) message is received from
            parent node coordinator
2    t_L = 0
3    for each child simulator s_i
4          send (init, 0) to child s_i
5    end for each
6    wait until all done messages receiv.
7    t_N = minimum t_N of all components
8    send (done, t_N) to parent node coord.
9 end when
```

When a collect message `(@,t)` is received, it is sent to all dependant simulators with minimum `t` (lines 3-7). Once all the responses are received (line 8), a `done` message is sent to the parent node coordinator. Simulators that have been scheduled for a transition are cached in the synchronize set.

```
 1 when a (@, t) message is received from
              parent node coordinator
 2  if t = t_N then
 3   t_L = t
 4   for each imminent child s_i with min. t_N
 5      send (@, t) to child s_i
 6      cache i in the synchronize set
 7   end for each
 8   wait all done messages received
 9   send (done, t) to parent node coord.
10  else
11    raise error
12  end if
13 end when
```

```
 1 when (y,t) message received from child i
 2   if destination of y is the environment
 3   send (y, t) to parent node coordinator
 4   else
 5    for each influencee j of child i
 6       q = z_i,j (y)
 7       if (j is a local processor) then
 8          send (q, t) to child j
 9          cache j in the synchronize set
10       else
11         send (q,t) to parent node coord.
12       end if
13     end for each
14   end if
15 end when
```

If the destination of the output `(y,t)` message is the environment, the message is sent to the parent node coordinator (lines 2-3; above).

If not, all influencees of the message are computed using the function z_{ij}, and one or more `(q,t)` messages are sent accordingly (lines 5-13).

For destination processors on the same LP, messages are sent directly to the simulator (lines 8-9). Messages for remote simulators are sent to the parent node coordinator (line 11), which will forward them to the corresponding LPs. Local components with scheduled transitions are cached in the synchronize set.

When an external message (q,t) is received in a flat coordinator, it is stored in a bag of events.

```
1 when (q,t) message received from
                parent node coordinator
2  if destination of q msg is local then
3      add event q to the bag
4   else
5      raise error
6   end if
7 end when
```

Upon receiving an internal message (*,t), the *flat coordinator* sends the external messages stored in the bag to the corresponding components (lines 3 to 8 in the following fragment). All the receivers of these messages are added to the synchronize set. Then, an internal message is sent to all components in the synchronize set. After all done messages are received back from these components, the time of the next event is calculated and a done message is sent to the *node coordinator* (lines 13-17).

```
 1 when (*,t) message is received from
          parent node coordinator
 2   if tL ≤ t ≤ tN then
 3      for each q ∈ bag
 4         for each local receiver sj of q
 5           send (q, t) to sj
 6           cache j in the synchronize set
 7         end for each
 8      end for each
 9      empty bag
10      for each i ∈ synchronize set
11         send (*, t) to i
12      end for each
13      wait all done messages received
14      tL = t
15      tN = minimum tN of all components
16      clear the synchronize set
17     send (done,tN) to parent node coord.
18   else
19      raise an error
20   end if
21 end when
```

Node coordinator

One *node coordinator* is located on each LP, and it has one flat coordinator child. Node coordinators drive inter-LP communication, and advance the simulation time in the local LP based on the information received from the *root coordinator* and from its dependant flat coordinator.

The algorithms describing its behaviour are described next.

The initialization message, sent by the *root coordinator*, triggers the simulation in each LP. An initialization message (init,0) is sent to the flat coordinator (line 2), which will forward it to every simulator.

```
 1 when a (init, 0) message is received
          from root coordinator
 2   send (init, 0) to child flat coord.
 3   wait for done message to be received
          from flat coordinator
 4   sort queue of events by arrival time
 5   t = min (tN of flat coordinator,
          time of first event in queue)
 6   if t = tN of queue then
 7      for each q in queue with time t
 8         send (q, t) to flat coordinator
 9      end for each
10   end if
11   send (@, t) to child flat coordinator
12   next-message-type = *
13 end when
```

The first simulation cycle starts after a (done,t) message is received. The time for the first collect message is determined by the minimum between the first element in queue of external events and the time of next change reported by the *flat coordinator* (lines 3-5). next-message-type is used to determine which type of message has to be sent. If the message to be sent is a collect (lines 6-17), the process is analogous to the initialization phase: minimum time t is computed, events with time t are sent (if any), the collect message is sent (line 16) and the next message type is set to internal (line 17).

When an internal (*,t) message has to be sent to finish the current simulation cycle, the type of the next message is set to collect (line 4).

```
 1 when a (done, t) message is received
          from child flat coordinator
 2   if next-message-type = * then
 3      send (*, t) to child flat coord.
 4      next-message-type = @
 5   else
 6    t= min (tN of flat coordinator,
          time of first event in queue)
 7      if t > stop simulation time then
 8         stop simulation in this LP
 9      else
10      if t=tN of first event in queue then
11          for each q in queue with time t
12              send (q, t) to flat coord.
13          end for each
14         end if
15      end if
16      send (@, t) to child flat coord.
17      next-message-type = *
18   end if
19 end when
```

An external message (q, t) can be received in a *node coordinator* either from another (remote) *node coordinator* or from its dependant *flat coordinator*. In the first case, this event must be sent to the dependant *flat coordinator* (line 3). This happens when a remote atomic component sends an output through a port connected to an atomic component executing in the local LP. As we have shown earlier, this message is forwarded by the *flat coordinator* to the corresponding simulator.

The timestamp t of a message received from a remote *node coordinator* might be lower than the current time in this LP. In such a case, the LP has to recover by performing a rollback. The rollback has to bring that object back to a state whose time is equal or smaller than the time of the straggler. In addition, the messages that were (incorrectly) transmitted from this node coordinator have to be cancelled (anti-messages have to be sent to the destination objects). In the second case, the message must be sent to a remote LP. Thus, it is necessary to determine which *node coordinator* is in charge of that LP, and then to send the message using inter-process communication (lines 5-6).

When a *node coordinator* receives an output message from its child (lines 10-16), a message has to be sent to the environment. The parameter send-outputs-from-NC determines whether outputs must be processed directly by the *node coordinator* (line 12), or via the *root coordinator* (line 14). The first alternative reduces the number of messages required to process an output (messages do not have to travel through the *root coordinator*) but requires some post-processing if the outputs of multiple node coordinators have to be merged. The second alternative centralizes the actual processing of outputs in the *root coordinator*; it does not require any post-processing but the overhead is larger.

```
 1 when a (q, t) message is received
 2   if destination q is local
 3      send (q, t) to child flat coord.
 4   else
 5    dest_nc=node coordinator running
          atomic model that must receive q
 6    send (q,t) to node coord. dest_nc
 7   end if
 8 end when
 9 /***********************************/
10 when (y,t) message is received from
        child flat coordinator
11  if send-outputs-from-NC
12     send output (y, t) to environment
13  else
14  send output (y,t) to parent root coord.
15  end if
16 end when
```

When an external event is received from the *root coordinator*, the event is stored in timestamp order. The destination simulator for that event will eventually receive it when that time is reached by the LP.

```
1 when a (q, t) message is received from
          parent root coordinator
2   add q to the sorted queue of events
3 end when
```

Root coordinator

The *root coordinator* is responsible for starting the simulation, dealing with external events, and sending outputs back to the environment. It starts the simulation by sending initialization messages to every *node coordinator*, located on the different logical processes that form the simulation.

```
1 for each child node coordinator nc i
2   send (init, 0) to nc i
3 end for each
```

External events are received from the environment in the *root coordinator*, which sends an external event to *node coordinators* that have one or more atomic model that should receive that message (lines 3-6 in the next code fragment).

```
 1  when (q,t) is received from environment
 2   t_L = t
 3   for each child node coord. nc_i sharing
 4     LP with destination atomic model of q
 5       send (q, t) to nc i
 6   end for each
 7  end when
 8  when a (y, t) is received from child NC
 9   t_L = t
10    send (y, t) to environment
11 end when
```

Output messages received by the *root coordinator* are sent back to the environment. This code is never executed if the parameter send-outputs-from-NC is set; otherwise, the *root coordinator* consolidates the processing of output messages.

Figure 3 summarizes the flow of messages using the previous algorithms. The *root coordinator* is in charge of starting the simulation process by sending initialization messages. When an output is sent from an atomic component to another, we can identify two different cases: Simulators execute on the same or on different LPs. In the first case, the FC on that LP takes care of the situation. In the second case, a Simulator on LP_i has to send an output to a Simulator on LP_j. FC_i identifies that the destination Simulator is not its descendant.

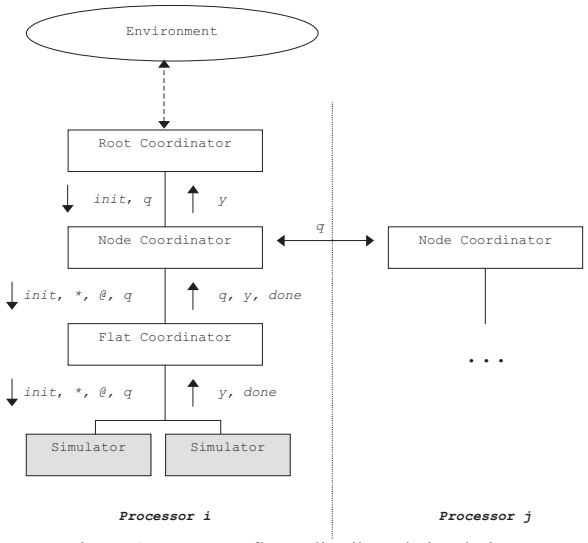


Figure 3: Message flow: distributed simulation.

Thus, it forwards the message to its parent NC_i, which identifies the corresponding LPj and forwards the message. Inter-LP communication can lead to violations to the local causality constraint; in that case, a rollback is triggered.

2.3 Rollbacks in CD++

We will show the execution behaviour of the simulator presented in the previous section by showing how to execute a 10 ×10 Cell-DEVS model divided in two LPs. Figure 4 shows the initialization phase. The first simulation cycle is started by the *Root Coordinator*, which sends an initialization message to the NCs in LP_0 and LP_1 (1 and 2). When (init,0) is received in a NC, it is forwarded to the FC (1.1 and 2.1). Then, the FCs forward the messages to their Simulators (1.2–1.51, and 2.2–2.51), triggering an initialization function.

After computing the time for the next change (using the *ta*(*s*) function), every simulator sends a done message to its parent FC reporting its time of next change. For example, S_1 indicates that there is an internal transition function to be executed at time 100 (message 1.52), whereas S_2 reports no scheduled internal transition (message 1.53, which contains infinity, and represents that the model is in a passive state). After receiving all done messages, the FC sends a done message to its parent NC (messages 1.103 and 2.103) with the minimum time of its components (in this case, 100 for both LPs).

Then, the NC checks for external messages to be sent, and it sends the first collect message to collect the outputs of the imminent components. Figure 5 shows how NCs send the first message to their FCs (1.1 and 2.1), which forward a collect message to imminent descendants (Simulators with next change = 100). For example, in LP_0 it is sent to S_1 (1.2) but not to S_2.

When receiving a collect message, Simulators execute their output functions and send the result/done messages to its parent (1.20 and 2.18). FC translates output messages and sends external messages to the corresponding local influencees or to the local NC. FC sends a done message to the NC completing the collect phase.

At this point, the NC is ready to send an *internal* message (*) to start the next phase. The cycle is similar to the *collect* phase: the FC forwards the *internal* message only to Simulators that have a scheduled transition for the current simulation time (100). Simulators execute the internal, external, or confluent transition function according to the current time, the time of next change and the state of the bag of events. Done messages are sent to inform the time for the next transition.

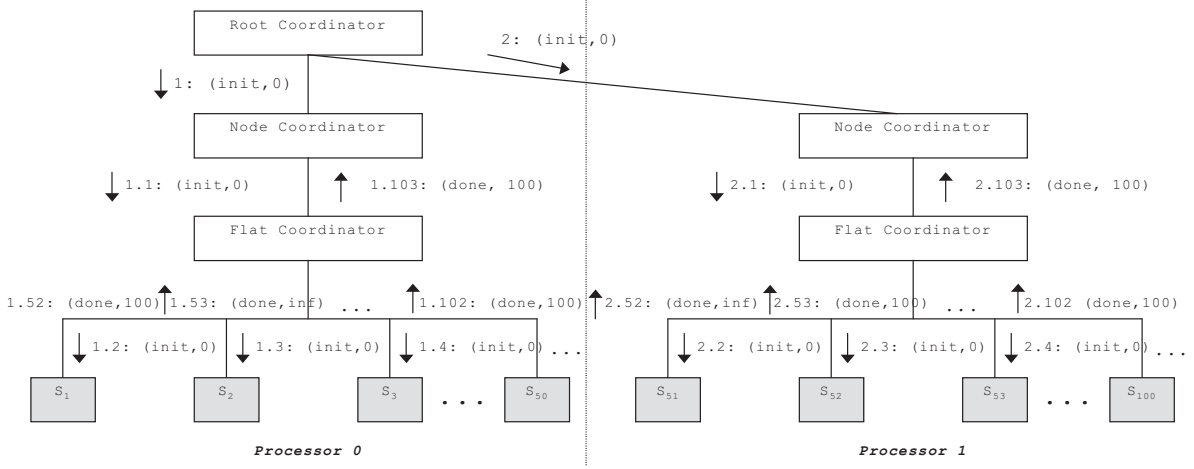


Figure 4: Initialization phase in sample Cell-DEVS model.

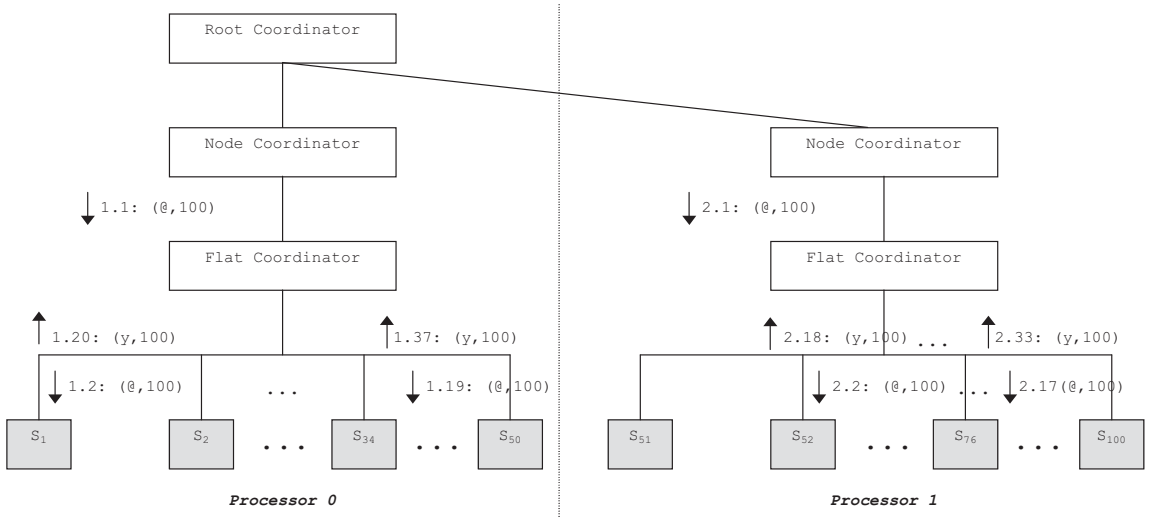


Figure 5: Collect phase in sample Cell-DEVS model.

NCs are in charge of inter-LP communication. If the message has a timestamp greater than the local time in the destination, simulation continues. However, if there is a violation to causality, a rollback has to be executed. Figure 6 shows the scenario for a straggler message in processor P_0. The local times are t_0=280, and t_1=210. In P_0, the NC sent an internal message, which FC forwarded to S_1 (1 and 1.1). In P_1, NC sent a collect message (2), which after being forwarded (2.1 to 2.14) resulted in an *output* from S_{52} (2.15) that has to be sent to S_1. This message is forwarded as an external message, q, from the FC (2.16) to the NC. Then, the NC in P_1 forwards it to the NC in P_0 (2.17). The timestamp of the message, 210, is smaller than the time at the local processor (280), triggering a rollback in P_0.

Figure 7a shows the state of the NC's input, state, and output queues at the moment of receiving a straggler with t=210. Figure 7b depicts the NC's queues after the rollback was completed: then, the NC can return to process events, starting by the one that caused the rollback.

We defined the previous algorithms using different services provided by Warped. Figure 8 shows the new class diagram of the DEVS processors along with some of methods that implement the algorithms previously described. *Processor* is an abstract class that is derived from *TimeWarp* class. *Processor* provides basic functionality and data that are common to all DEVS processors in the application. It defines the methods *initialize*, *executeProcess* and *finalize* as well as other methods and variables.

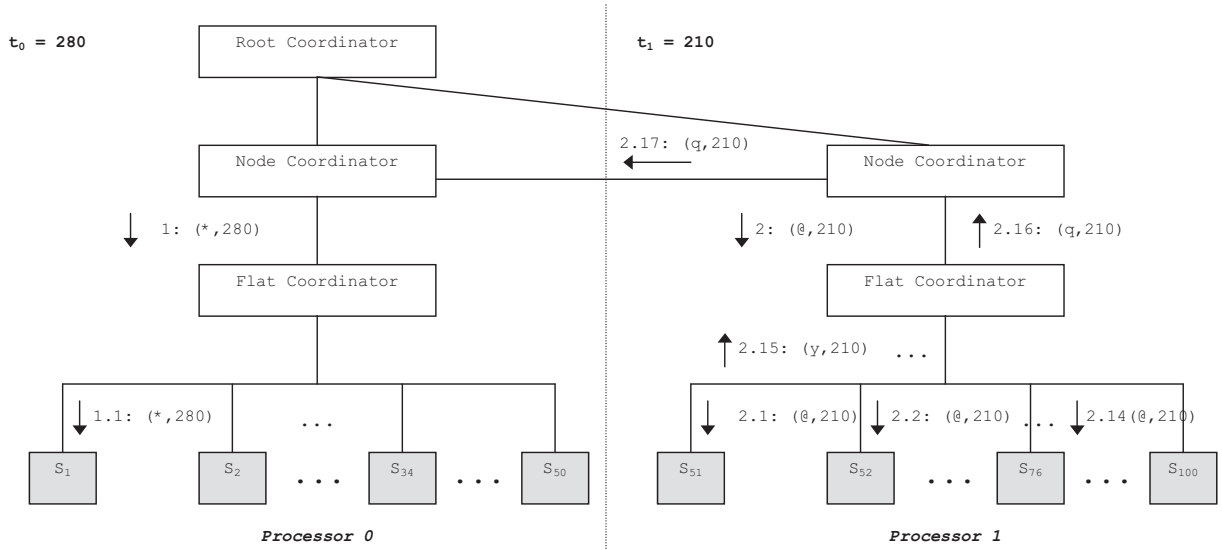


Figure 6: Straggler message received during the simulation of a Cell-DEVS model.

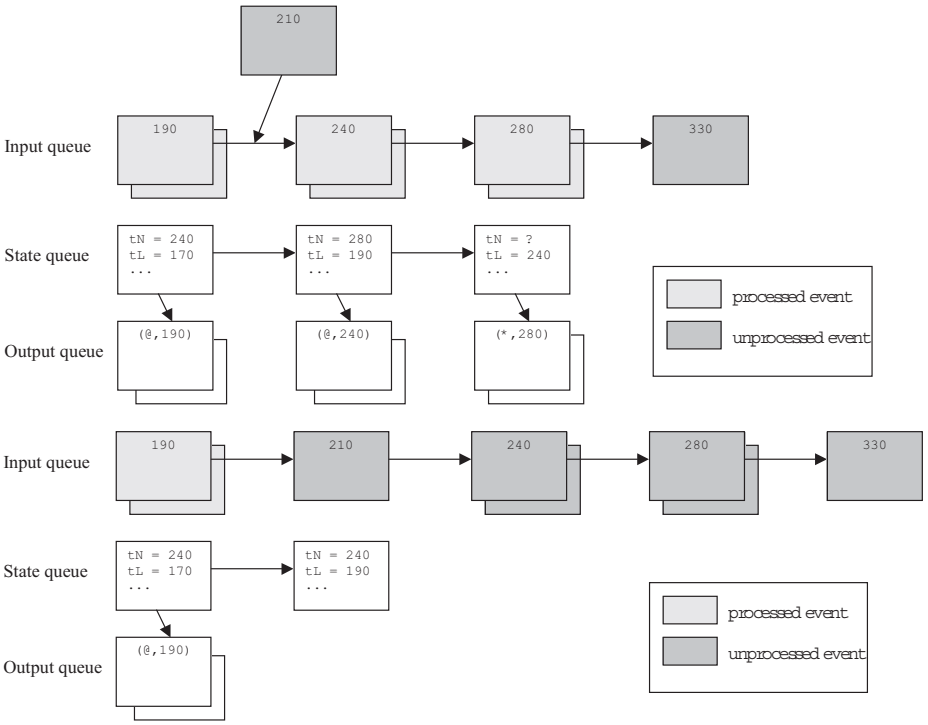


Figure 7: (a - top) Reception of a straggler message in a NC, (b - bottom) State of the NC after the rollback.

In general, processor includes the definition of:
- send methods for each type of message. These methods use, in turn, *sendEvent* in *TimeWarp*.
- Time management methods (e.g., *timeNext()*, *timeLast()*, *timeNext(VTime)*, *timeLast(VTime))*, to report and update the time of the next scheduled change, the time of last change, etc.
- *Initialize, finalize*, and some debugging methods.
- *ExecuteProcess()*, which defines the behaviour of any DEVS processor.
- *rollbackCheck()*, which is called in the receive method, and checks for straggler messages.
- Basic variables, such as the model associated to this processor, its parent, id and descriptors.

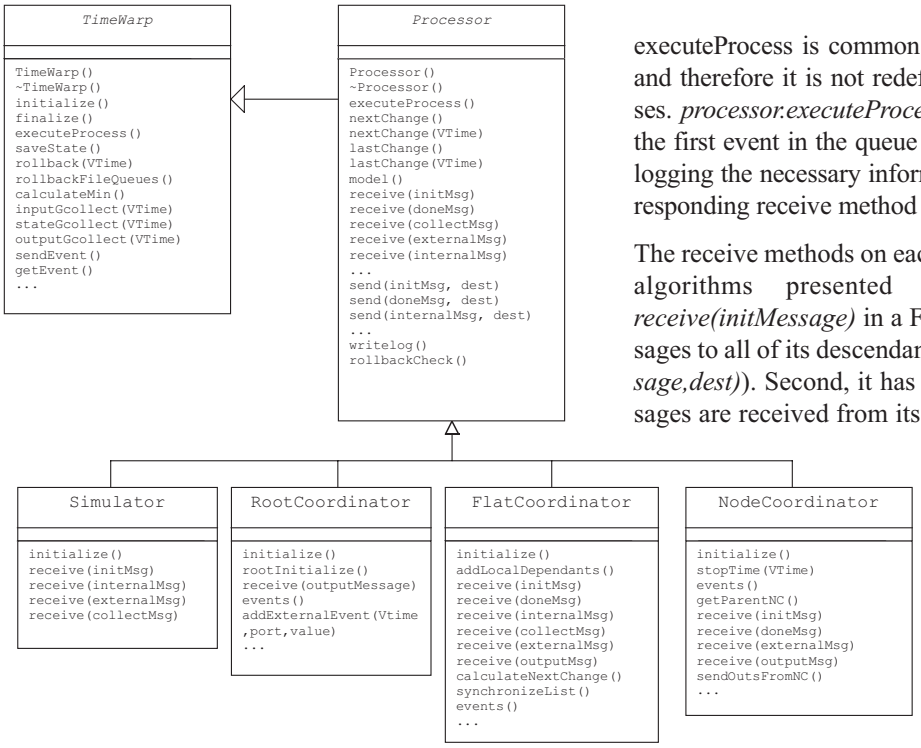


Figure 8: Class diagram for the new DEVS processors.

executeProcess is common to every DEVS processor, and therefore it is not redefined by any of its subclasses. *processor.executeProcess()* is in charge of getting the first event in the queue of events (using *getEvent*), logging the necessary information, and calling the corresponding receive method based on the message type.

The receive methods on each processor implement the algorithms presented earlier. For example, *receive(initMessage)* in a FC sends initialization messages to all of its descendants (using the *send(initMessage,dest)*). Second, it has to wait until all done messages are received from its dependant Simulators.

nodeCoordinator keeps track of the number of done messages it has received (using *doneCount()*). Finally, it determines and updates the time of next change (using *nextChange(VTime)*), and sends this value to its parent NC (using *send (doneMsg,dest)*).

The *receive* (*initMessage*) method on *Simulator*, in contrast, initializes the model variables, computes the time for the next transition (using time advance function, *ta*) and sends a *done* message to its parent.

3 Simulation Experiments

We carried out different performance tests to analyze the results obtained with the new algorithms. To provide uniform means for the overhead, we used the DEVStone benchmark, a synthetic model generator that automatically creates models [18]. DEVStone uses three different types of models with variations in their internal and external structure: LI models, with a low level of interconnections for each coupled model; HI models with a high level of input couplings, HO models with high level of coupling and numerous outputs.
Table 1 shows the parameters we used for different tests. Each model is executed using a different number of levels in the modeling hierarchy (Depth), and different number of submodels on each level (Width). Likewise, different execution times are used for the transition functions, using the DEVStone benchmark.

	Type	Depth	Width	δ_{int}	δ_{ext}
A	LI	3	10	50 ms	50 ms
B	LI	10	3	50 ms	50 ms
C	LI	5	5	50 ms	50 ms
D	LI	10	10	50 ms	50 ms
E	HI	3	6	50 ms	50 ms
F	HI	6	3	50 ms	50 ms
G	HI	5	5	50 ms	50 ms
H	HI	6	6	50 ms	50 ms

Table 1: Simulation parameters.

The following figures show some of the overhead results obtained for these different execution times. The experiments were executed in a single processor, allowing us to measure the pure overhead incurred by our simulator.

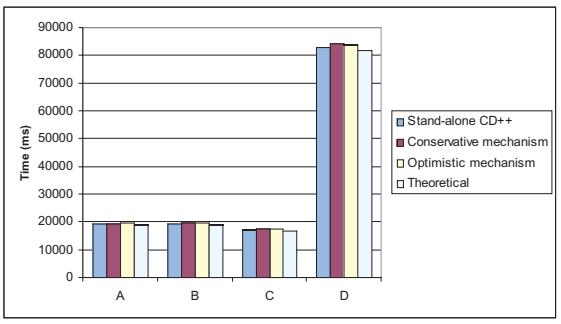


Figure 9: Execution times for LI models.

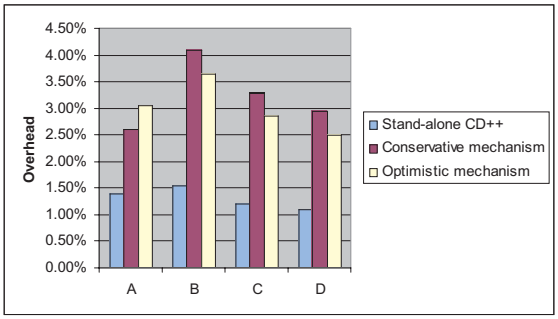


Figure 10: Overhead for LI models

In Figure 9 and Figure 10, we present the total execution time and the overhead for models A-D. We can see that the stand-alone engine outperforms the optimistic one, because the optimistic simulator is more complex. In this case, we need extra synchronization, saving states, input, and output queues, etc. Although the overhead associated with those tasks can be considerable, the optimistic simulator still outperformed the conservative simulator for models B, C, and D.
This is a consequence of the reduction in communication overhead incurred by the flat simulator. In model A, the hierarchical conservative engine performs better than the flat, optimistic engine as a consequence of the structure (3x10) of model A. In this case, the reduction in messages exchanged is not that important.
Figure 11 illustrates the results for HI models, which are similar to those obtained for LI.

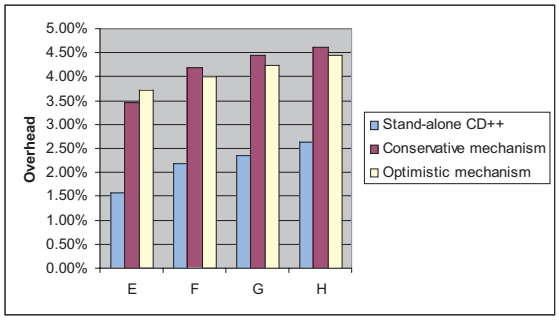


Figure 11: Overhead for HI models

We executed several Cell-DEVS models using different cell spaces on one and four processors. As we can see in Figure 12, the execution time for the model running on one processor varies from 30.7 to 90.8 seconds. When running the model in parallel on 4 processors, the execution time is smaller (between 18.1 and 47.5 seconds); in some cases, the optimistic simulator allows to reduce the execution time in ~50%. Here, the speedup has been affected by the communication costs, as the tests were executed over a relatively slow network, a 10 Mbit/s hub.

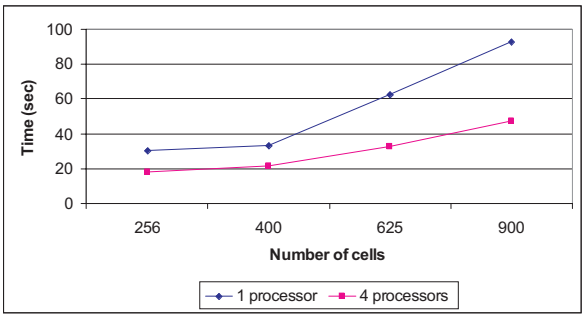


Figure 12: Execution times (1 vs. 4 processors).

We are interested in analyzing the performance of our simulator for larger Cell-DEVS. Figure 13 shows the execution times for different configurations for a cell space of 50x50, using different initial values (*life A-D*). The execution times significantly reduce on 8 processors. When a 50x50 model is executed on a single processor, only one LP is created. Hence, a single instance of a FC is in charge of the 2500 Simulators, and a single NC is in charge of scheduling tasks for the entire model. In contrast, the distribution on 8 processors allows a smaller structure associated with each LP (312 Simulators).

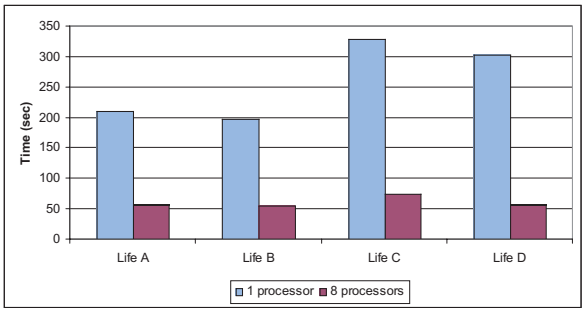


Figure 13: Execution times (50x50 model).

The test uses a sample Cell-DEVS model to study the performance of a firefly model, in which most of the cells change frequently, producing increased processor load. We execute models with 400 and 900 cells, with two initial configurations (models 1 to 4). The optimistic simulator running on a single processor achieves almost the same performance as the conservative simulator running on 4 processors, which shows the increased communication costs.

Figure 14 shows that the optimistic simulator allows significant speedups: 2.91 for 20x20 models, 3.17 for 30x30 models. The speedup factor obtained by executing the simulation on 4 processors using the optimistic approach instead of the equivalent partitioning for the conservative approach is 2.45 for 20x20 and 30x30 models.

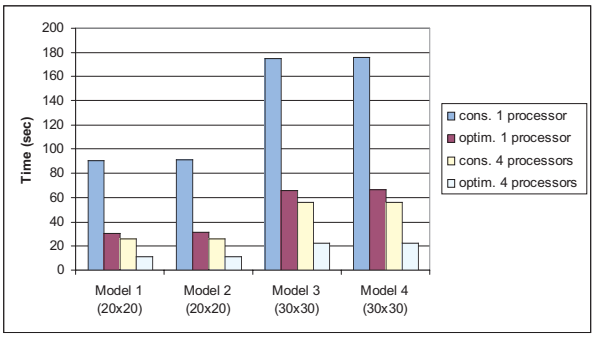


Figure 14: Execution times using conservative and optimistic simulators (1-4 processors).

4 Conclusions

We have introduced a new flat simulation technique for DEVS and Cell-DEVS based on Time Warp, a well-known optimistic synchronization protocol. Our efforts address the need for efficient, fast execution of models using parallel and distributed simulation. We propose an optimistic distributed mechanism that enables achieving higher degrees of parallelism than previous efforts, which only allowed exploiting parallelism in a limited way.

Under our new approach, scheduling tasks are distributed on the Logical Processes; each *Node Coordinator* is in charge of the scheduling tasks for the local simulation objects. Node Coordinators advance the simulation optimistically, assuming that there will be no straggler events. In case of detecting a violation to the local causality constraint, a rollback mechanism allows recovering from it.

Using DEVStone, we compared the overhead of our new technique with the overhead of previous implementations. Although the overhead associated with synchronization tasks implemented by our simulator can be considerable, it still outperformed previous alternatives for some models in single-processor executions. This is a consequence of the flat mechanism implemented in our engine, which outweighs the increased overhead associated with its more complex implementation.

More importantly, we showed that when executing different types of DEVS models, the overhead of Warped/MPI is reasonable small (2.5%-5%). This is a promising result, as the amount of speedup time achievable by these simulators is considerable, and having a constrained overhead in the kernel permits a better utilization of the computing resources.

We showed that the execution times for a particular Cell-DEVS model can be reduced using distributed simulation. Different model sizes where considered, ranging from 256 to 2500 cells.

The execution of the model in a distributed environment allowed achieving better performance than stand-alone execution. Using distributed environments, our simulator outperforms other alternatives and achieves considerable speedups.

References

[1] R. M. Fujimoto: *Parallel and Distribution Simulation Systems*. Wiley. 1999.

[2] R. E. Bryant: *Simulation of Packet Communication Architecture Computer Systems*. MIT, Cambridge, MA. USA. 1977.

[3] K. Chandy, J. Misra: *Distributed Simulation: A Case Study in Design and Verification of Distributed- Programs.* IEEE Transactions on Software Engineering, pp. 440-452. 1979.

[4] D. R. Jefferson: *Virtual time*. ACM Transactions on Programming Languages and Systems. vol. 7(3), pp. 404-425. July, 1985.

[5] B. Zeigler, T. Kim, H. Praehofer: *Theory of Modelling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems*. Academic Press. 2000.

[6] A. C. Chow, B. P. Zeigler: *DEVS: A parallel, hierarchical, modular modelling formalism*. Proceedings of the Winter Computer Simulation Conference. Orlando, FL. USA. 1994.

[7] K. H. Kim, Y. R. Seong, T. G. Kim, K. H. Park: *Distributed Simulation of Hierarchical DEVS Models: Hierarchical Scheduling Locally and Time Warp Globally*. Transactions of the Society for Modelling and Simulation International. vol. 13(3), pp. 135-154. 1996.

[8] A. Troccoli, G. Wainer: *Implementing Parallel Cell-DEVS.* Proceedings of the Annual Simulation Symposium. Washington DC, USA. 2003.

[9] B. Zeigler, Y. Moon, D. Kim, G. Ball: T*he DEVS Environment for High-Performance Modelling and Simulation*. IEEE Computational Science and Engineering. 4 (3), pp. 61 -71. 1997.

[10] G. Wainer, N. Giambiasi. *N-Dimensional Cell-DEVS*. In Discrete Events Systems: Theory and Applications, Kluwer. Vol. 12, No. 1. January 2002. pp. 135-157.

[11] S. Wolfram: *A new kind of science*. Wolfram Media, Inc.

[12] G. Wainer, G. *CD++: a toolkit to develop DEVS models*. Software - Practice and Experience. vol. 32, pp. 1261-1306. 2002.

[13] E. Glinsky, G. Wainer: *Performance analysis of DEVS environments.* Proceedings of AI Simulation and Planning. Lisbon, Portugal. 2002.

[14] G. Wainer: I*mproved cellular models with parallel Cell-DEVS*. Transactions of the SCS. vol 17 (2). June 2000.

[15] D. Martin, T. McBrayer, P. Wilsey: *WARPED: Time Warp Simulation Kernel for Analysis and Application Development*. Proceedings of the 29th Hawaii International Conference on System Sciences. 1996.

[16] J. Dongarra, J. et al. *MPI: The Complete Reference*. The MIT Press. 1996.

[17] E. Glinsky, G. Wainer: *New Parallel simulation techniques of DEVS and Cell-DEVS in CD++*. In Proceedings of the 38th IEEE/SCS Annual Simulation Symposium. Huntsville, AL. 2006.

[18] E. Glinsky, G. Wainer: *DEVSTONE: a Benchmarking Technique for Studying Performance of DEVS Modelling and Simulation Environments*. Procedings of IEEE/DS-RT. Montréal, QC. 2005.

Corresponding author: Gabriel Wainer,
Dept. of Systems and Computer Engineering,
Carleton University
4456 Mackenzie Building, 1125 Colonel By Drive
Ottawa, ON. K1S 5B6, CANADA.
WWW.SCE.CARLETON.CA/faculty/wainer

Received: June 5, 2006
Revised: July 1, 2006
Accepted: July 15, 2006

SCE based Parallel Processing and Applications in Simulation

René Fink, Sven Pawletta, Thorsten Pawletta, Wismar University, Germany
WWW.MB.HS-WISMAR.DE/cea
Berrnhard Lampe, University of Rostock, Germany; *bernhard.lampe@uni-rostock.de*

In this paper, an overview of SCE based parallel processing is presented, and a new taxonomy for this field is discussed. More than 30 SCE based parallel processing projects are listed and categorized with respect to the new taxonomy. For the Multi-SCE class, performance parameters of several packages are presented in terms of latency and bandwidth. Furthermore, characterization schemes of parallel simulation and optimization applications are presented as well as performance results and characteristics of selected applications being parallelized under usage of Multi-SCE packages.

SNE 16/2, September 2006

Introduction

Nowadays, *Scientific and Technical Computing Environments* (SCEs) like MATLAB, Scilab or Octave are well established in numerical computations. Particularly simulation and optimization applications are well supported by these environments, often by specialized subsystems and toolboxes. Before the introduction of SCEs, the development of scientific and technical computations was exclusively compiler based, so the process of programming, compiling, linking and testing had to go through several time consuming iterations. Advantages of SCEs in contrast to compiler based programming techniques are the way of interactive working, integrated numerical and visualization libraries, extendability and a very high level programming language.
Due to these advantages, rapid software prototyping was enabled in the field of computational science and engineering. One major drawback of SCE based programming is that interactive programming is based on an interpreter, so program execution in SCEs is significantly slower than program execution of compiled programs. To weaken this drawback, several possibilities exist, e.g. code optimization, compilation or parallel processing.

SCE based parallel processing was first investigated by The MathWorks Inc. in the middle of the 1980's. These investigations showed disappointing performance results, published in 1995 ([1]). Beginning in the same year, first results of other research projects were published ([2, 3]), offering better performance. In a period of more than eleven years until today, several research and non-research projects concerning SCE based parallel processing were executed and produced a number of extensions to several SCEs, mostly MATLAB. In 2004, the MathWorks Inc. released the *Distributed Computing Toolbox* for Matlab ([4]), following the increasing demand for parallel processing in SCEs.

In an ongoing research project at the University of Applied Sciences Wismar, SCE based parallel processing techniques are investigated ([*]). Main motivations for this research are simulation and optimization problems, which cause extensive computations. Essential results of up to now investigations are presented in this paper. Section 1 introduces parallel processing, focusing on a basic taxonomy, recent parallel hardware architectures and programming techniques. Section 2 gives an overview of SCEs, including history, representatives, characteristics and possibilities of program acceleration. In Section 3, SCE based parallel processing is discussed. In this section, a new taxonomy on SCE based parallel processing is presented, followed by the characterization of existing projects and their assignment to this taxonomy.
From the formerly presented taxonomy, Section 4 focuses the Multi-SCE class. In this section, results of Multi-SCE communication performance analysis are presented. Furthermore, new Multi-SCE prototypes are introduced and compared with existing Multi-SCE projects regarding communication performance. In Section 5, characterization schemes for simulation and optimization applications with respect to parallel processing are presented. Aspects of characterization are parallelism level, granularity and programming model applicability. In Section 6, simulation and optimization applications are presented, which have been parallelized under usage of Multi-SCEs. This presentation includes application characteristics as well as parallel runtime results. In Section 7, major facts of preceding sections are summarized.

1 Parallel Architectures and Programming Models

1.1 Basic Taxonomy

In the field of parallel processing a large number of taxonomies have been developed. Usually, taxonomy scopes are limited either to hardware or software.

In this article, a common taxonomy for both hardware and software structures is used. It is based on a conjunction of a very general interpretation of the well-known Flynn taxonomy ([5]) and the usual classification of memory structures into shared and distributed units.
Flynn distinguishes hardware architectures by the number of instruction streams simultaneously applied to one or more data streams via processing elements, leading to the four basic classes SISD, SIMD, MISD and MIMD. In [6] it is shown that this principle is also applicable to software structures. For instance, a process or thread can be seen as processing element, where a code sequence is applied as instruction stream to a data stream. In this sense, Flynn's classes can be described and interpreted as follows. In the *SISD* class, a single instruction stream (SI) is applied to a single data stream (SD) by one processing element, resulting in completely sequential processing. The *SISD* class contains no parallelism, but shows the delimitation of sequential structures to parallel structures. In the *SIMD* class, a single instruction stream is applied to multiple data streams by multiple processing elements. Since only one instruction stream exists, processing elements must work synchronously (synchronous parallelism). In the *MISD* class, multiple instruction streams are applied to a single data stream by multiple processing elements. In the *MIMD* class, multiple instructions streams are applied to multiple data streams using multiple processing elements. Since there are multiple independent instruction streams, processing elements do not have to work synchronously (asynchronous parallelism).

Parallel processing hardware is usually classified into shared and distributed memory architecture classes (*SHM* and *DM*). On a more abstract level, these concepts can be used for further refinement of the above general interpretation of Flynn's taxonomy. In this sense, the *SHM* class describes a shared memory structure, where multiple processing elements have access to a common memory space. In contrast, the *DM* class describes a distributed memory structure with completely individual memory spaces for each processing element.

1.2 Hardware Architectures

The classification following Flynn's taxonomy combined with memory structures is a common theoretical entrance to characterize parallel hardware architectures. But with respect to real hardware, not all classes of Flynn's taxonomy are actually relevant today. The *SISD* hardware class has no relevance for parallel processing since it contains no parallelism at all.

The *MISD* hardware class is typically considered to be empty because no hardware structures exist following this processing scheme, though Flynn disagrees with that. The *SIMD* hardware class was important for large-scale parallel processing until the 1990's, but disappeared almost completely today in this area. Thus, *MIMD* is the only class that is leftover. Actually, a large variety of *MIMD* hardware exists and can be further classified with respect to memory structures.

Regarding memory structure, real *MIMD* architectures can be divided into three subclasses: shared, distributed and hybrid memory. Thereby, the memory structure strongly influences the scalability and communication performance of an architecture. Generally, distributed memory architectures are well scalable, but show low communication performance. On the other hand, shared memory architectures have limited scalability, but high communication performance. In hybrid memory architectures, shared memory structures on lower hierarchy levels are combined with distributed structures on higher levels. Therefore, hybrid memory architectures are well scalable and show high communication performance on lower hierarchy levels.

In a hardware architectures investigation of *SNE Comparison CP1* contributions from 1994 until 2003 ([7]), being intended to attain a hardware overview in small scale parallel processing (especially in simulation domains), three types of hardware have been identified: Cluster Computers, Symmetric Multiprocessors and Transputers. *Cluster Computers* consist of independent processors with own physical memory, connected by a standard interconnect. They can be categorized as distributed memory *MIMD* systems. *Symmetric Multiprocessors* (SMP) consist of multiple homogeneous processors accessing a common physical memory. They can be classified as shared memory *MIMD* systems. *Transputers* consist of multiple processors with own physical memory, containing additional supporting hardware for fast interconnection. They can be classified as distributed memory *MIMD* systems. Transputers were well-established in small-scale parallel computing during the 1990's, but disappeared almost completely today.

In a second hardware architectures investigation of systems appearing on the recent Top500 list, showing the 500 most performing parallel computers, also three types of hardware have been identified (WWW.TOP500.ORG): Cluster Computers, Constellations and Massively Parallel Computers.

Constellations consist of multiple SMPs, connected by a standard interconnect. Constellations can be classified as hybrid memory *MIMD* systems. *Massively Parallel Computers* consist of a very large number of processors (>1.000) with partly own and common physical memory, connected by a fast interconnect. They can be classified as distributed or hybrid memory *MIMD* systems.

The two investigations of parallel processing hardware show that recently only *MIMD* systems have practical relevance, both in small scale and large scale parallel processing. Hardware architecture differences regarding the described taxonomy can only be found in memory structures. Small scale parallel processing is dominated by distributed and shared memory architectures. In large scale parallel processing distributed and hybrid memory architectures are dominant. Shared memory systems are no longer present in the Top500 list because of their limited scalability. One can assume that hybrid memory architectures will also prevail in small-scale parallel processing in the near future. The main reason for this assumption is the establishment of multi core CPUs as commodity off-the-shelf (COTS) components, enabling installations of reasonable small scale *Constellations*.

1.3 Parallel Programming

As an interface between the programmer and the underlying parallel hardware structures, parallel programming models are used. Parallel programming models are supported either on the level of programming languages, compilers or applied libraries. Following Skillicorn ([8]), the development of parallel programs requires the fulfillment of four tasks:

- *Partitioning*: divide a problem into multiple partial problems
- *Instantiation and Mapping*: process startup and assignment to processors
- *Communication*: data exchange between processes
- *Synchronization*: management of process states

In a parallel programming model, these programming tasks can be explicit or implicit (visible or invisible in the program code). The distinction of parallel programming models is based on their set of explicit and implicit programming tasks. Programming models with only implicit programming tasks are referred to as implicit models, while programming models requiring at least one explicit task are referred to as explicit models.

Examples of implicit programming models are data parallel languages like FORTRAN 90, parallelizing compilers like Intel or SGI compiler suites and the application of parallel numerical libraries like ScaLAPACK. Widespread models for explicit parallel programming are:

- Shared Memory Programming
- Message Passing Programming
- Remote Procedure Call (RPC) Programming

In all of these models, partitioning must be done explicitly, while explicity of instantiation and mapping is system dependent. The explicit or implicit representation of synchronization and communication tasks is the main distinctive feature of the presented models, as shown in Table 1.

Prog. Model	Partitioning	Inst. and Mapping	Communication	Synchronization
Shared Memory	explicit	explicit or implicit	implicit	explicit
Message Passing	explicit	explicit or implicit	explicit	implicit
RPC	explicit	explicit or implicit	implicit	implicit

Table 1: Programming tasks in explicit parallel programming models.

In *Shared Memory Programming*, process communication is done implicitly via a common memory address space. On the other hand, process synchronization has to be organized explicitly, in the simplest case to avoid concurrent write accesses to the same addresses. Common shared memory programming standards are OpenMP and POSIX threads.

In *Message Passing Programming*, process communication has to be implemented explicitly by send/receive operations. In contrast to shared memory programming, process synchronization is completely implicit due to the causality of message transfer (a receiving process blocks until a desired message is sent). Well-know standards and libraries for message passing programming are MPI and PVM.

Unlike shared memory and message passing programming, *RPC Programming* is not originated in parallel programming. Classical synchronous RPC is a usual model in distributed programming. It is based on a client-server structure, where one server process serves multiple client processes. To make RPC usable in parallel programming, this scheme must be inverted, resulting in multiple server processes serving one client exclusively.

In such a scenario, the client splits up one problem into multiple partial problems, subsequently processed by the servers. After finishing all partial problems, the results are collected by the client. This programming scheme is also referred to as embarrassingly parallel or task parallel.
When using RPC in parallel programming, the classical synchronous RPC semantic must be extended either to asynchronous scalar RPC (non-blocking RPC [9]) or synchronous vectorial RPC (multiple RPC at the same time [6]). In all RPC variants, communication takes place implicitly by parameter passing to and from the remote procedure.

Regarding the introduced basic taxonomy, shared memory and message passing programming can be seen as native programming models on top of physical shared and distributed memory architectures, respectively. But of course, arbitrary mapping of programming models to certain physical architectures is possible by intermediate layers. The RPC programming model has no direct counterpart on the hardware level, but can be mapped by shared memory and message passing programming as well.

2 Scientific and Technical Computing Enviroments

A *Scientific and Technical Computing Environment* is a software system that enables interactive numerical scientific and technical computations and visualizations. The term SCE has been introduced in [6] as an abbreviation of *Scientific and Technical Computing and Visualization Environment*.
Before the introduction of SCEs, scientific and technical computations were executed by compiler based program development. To develop a program, the process of coding, compiling, linking and testing had to be iterated several times. Furthermore, to visualize computational results, additional programs or libraries were necessary.

The first major event of SCE history took place in 1977, when Research Systems Inc. released IDL (*Interactive Data Language*), enabling interpretative data analysis and visualization. Also in the late 1970's, Cleve Moler developed the first version of MATLAB, a matrix based interpretative programming system for linear algebra problems. In 1984, the first commercial version of MATLAB was released by The MathWorks Inc., focussing technical computations and visualizations on the upcoming PC. At the same time, further commercial SCEs like Ctrl-C, MatrixX and Gauss were released, partly derived from the first MATLAB version and with MATLAB - similar syntax.

During the 1990's, MATLAB was growing to a proprietary quasi-standard in computational engineering. In the same decade, several non-commercial SCEs arised (e.g. Octave in 1993, Scilab in 1994), partly with MATLAB compatible syntax and function sets. Nevertheless, MATLAB remained a standard, which is revealed by the large set of MATLAB extensions provided by The MathWorks Inc. and third parties. Table 2 summarizes today's available SCEs.

SCE	Producer	Initial Release
IDL	Research Systems	1977, commercial
Gauss	Aptech Systems	1983, commercial
Matlab	The MathWorks	1984, commercial
O-Matrix	Harmonic Software	1992, commercial
Octave	J. W. Eaton	1993, non commercial
Scilab	INRIA	1994, non commercial
Rlab	I. Searle	1994, non commercial
Tela	P. Janhunen	1994, non commercial
Euler	R. Grothmann	1996, non commercial
Yorick	D. H. Munro	1996, non commercial

Table 2: Today's available SCEs, producers and initial release date.

2.1 SCE Characteristics

The following properties denote software systems as SCEs:
- Interactive way of working
- Integrated numerical libraries
- Integrated visualization libraries
- Extendable function set
- Matrix oriented high level programming language

Interactive way of working allows immediate data analysis as well as tests of new code lines or modules. Integrated numerical libraries provide users immediate access to well tested algorithms, while visualization libraries support instant visual data analysis. The extendable function set enables the addition of user defined functions or third party products to the SCE.
The SCE's high-level language allows variables to be defined, redefined or resized at any point in a program. The data type of variables is determined implicitly and can change during program execution. Basic data types are double precision matrices, extendable to complex or non-complex multidimensional arrays. Consequently, operators and integrated functions are array based, which results in compact data parallel code.
Figure 1 shows the basic structure of SCEs: data processing follows the classical von Neumann (SISD) scheme, processing one instruction stream over one data stream.

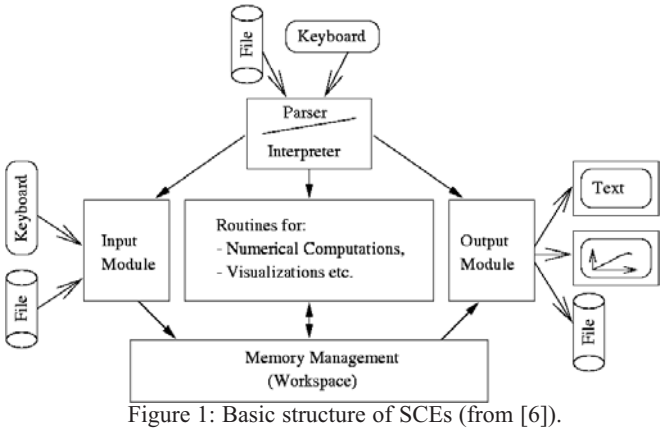


Figure 1: Basic structure of SCEs (from [6]).

For this purpose, SCEs contain data stream input and output modules as well as an instruction stream input module represented by the interpreter. A memory management module storing internal data and a computational module containing internal and external function sets are further essential SCE components.

2.2 Acceleration of SCE Program Execution

As already mentioned, the usage of SCEs for scientific and technical program development takes considerably less time compared to compiler-based development. On the other hand, program execution in SCEs is significantly slower than execution of compiled programs. SCE code acceleration offers possibilities to weaken this drawback.

To accelerate SCE programs, profilers are useful. A profiler allows time analysis of every code line and simplifies identification of time-consuming program structures. Today, profilers are components of commercial SCEs like MATLAB, IDL or Gauss, only.

Several approaches to accelerate SCE program execution exist, also in combination with each other:

- Program optimization
- Compilation of SCE code
- Re-implementation in compilable language
- Parallel processing of SCE code

In case of *program optimization*, time-consuming code lines are substituted by more efficient program structures. Substitutions are for instance data pre-allocation instead of data resizements or data parallel operations instead of loops.

If *compilation of SCE code* is used, complete SCE programs or parts of them are automatically translated into compilable intermediate code and fed to a compiler subsequently (explicit compilation). Automatic compilation is also used by just-in-time compilers compiling program structures temporarily, so that time consuming interpretation is avoided (implicit compilation). Beside acceleration, compilation of SCE code is useful for SCE program deployment as a stand-alone application.

If *re-implementation of SCE code* is applied, time-consuming program structures are transformed manually into compilable code and compiled to machine code subsequently. Since prototype development already happened within the SCE, code re-implementation takes considerably less time than direct compiler based development. For the re-implementation approach, an interface between SCE and a compilable language is necessary. For instance, interfaces are provided by MATLAB (C, Fortran), Octave (C++) or Scilab (C, Fortran).

In *parallel processing of SCE code*, time-consuming program structures run in parallel on multiple processors. Discussions on SCE based parallel processing are topics of the following sections.

3 SCE - based Parallel Processing

3.1 Taxonomy

The combination of parallel processing and SCE based working can be realized in several ways. To ease comparisons of SCE based parallel processing variants, coarsely classifying taxonomies are proposed. In literature, two sources of taxonomies can be found. The first source ([6]) uses a SCE spanning taxonomy distinguishing four categories:

- *Compilation Approach*: compilation of SCE code into parallel machine code; execution on a parallel system
- *Coupling Approach*: coupling of SCE and parallel system; remote execution of parallel routines
- *Parallel-SCE Approach*: SCE runs on a parallel system; local execution of parallel routines
- *Multi-SCE Approach*: coupling of multiple SCEs, altogether representing a parallel system

The second source ([10]) uses a MATLAB based taxonomy and also distinguishes four categories:

- *Embarrassingly Parallel*: coupling of multiple MATLAB instances with one superordinated instance; RPC programming model.
- *Message Passing*: coupling multiple MATLAB instances without superordinated instances; message passing programming model.

- *Backend Support*: MATLAB as a front end to a parallel system executing parallel routines.
- *MATLAB Compiler*: compilation of MATLAB code into parallel machine code; execution on a parallel system.

A comparison of both taxonomies shows that in two cases categories of one taxonomy can be merged into one category of the other taxonomy. Firstly, Pawletta's *Coupling Approach* and *Parallel-SCE Approach* can be merged into Choy's *Backend Support*. Secondly, Choy's *Embarrassingly Parallel* category and *Message Passing* category can be merged into Pawletta's *Multi-SCE Approach*. Furthermore, the compiler-based category is the same in both taxonomies.

Therefore, both existing taxonomies can be brought together, resulting in a new taxonomy on SCE based parallel processing (see Figure 2):

- *Compilation Approach*: compilation of SCE code into parallel machine code; execution on a parallel system
- *Front End Approach*: SCE as a front end to a parallel system which executes parallel routines
- *Multi-SCE Approach*: coupling of multiple SCEs, altogether representing a parallel system

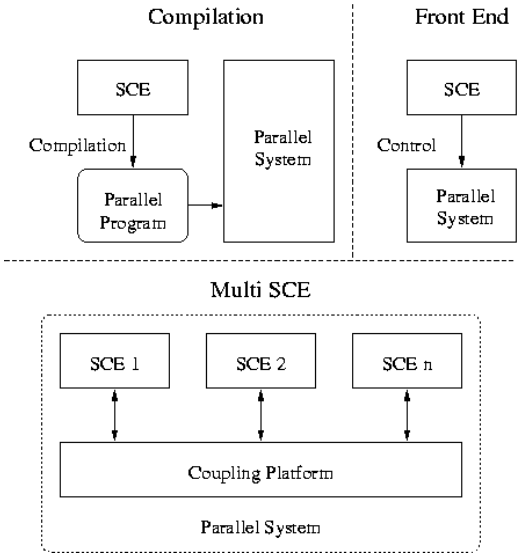


Figure 2: New taxonomy on SCE based parallel processing.

3.2 Compilation Approach

Following the compilation approach, sequential SCE programs or parts of them are translated into compilable intermediate code.

For the intermediate code, scalar languages like C or Fortran 77, and data parallel languages like Fortran 90 are used. Subsequently, the intermediate code is compiled by a parallelizing compiler or linked against parallel numerical libraries after compilation. To create a parallel program by compilation, no modification of SCE code is necessary, which indicates an implicit parallel programming model. For parallel execution of a compiled SCE program, no SCE is necessary. Therefore, SCEs are just utilized as development environments in this approach.

Table 3 summarizes projects following the compilation approach. Compiler based projects focusing on parallel signal processing hardware are neglected in this summarization due to their high specialization.

Project	SCE	Intermediate Code	Rel.
Conlab [11] Compiler	Conlab (MATLAB)	C incl. Message Passing Routines	1993
Falcon [12]	MATLAB	Fortran 90	1996
Menhir [13]	MATLAB	C or Fortran 77 incl. parallel linear algebra routines	1998
Otter [14]	MATLAB	C incl. parallel linear algebra or signal processing routines	1998

Table 3: Projects following the compilation approach.

The Conlab Compiler represents an exception in this summarization, since sequential SCE code has to be modified to run in parallel. Conlab is a development environment for parallel algorithms with a syntax similar to MATLAB. Additionally, Conlab contains directives for parallel processing, e.g. process management, message passing and shared memory techniques. Conlab itself can only simulate parallel processing, but the Conlab compiler is able to translate Conlab (or extended MATLAB) code into real parallel programs.

3.3 Front End Approach

Using the front end approach, the SCE represents a user interface to a parallel processing platform. Whithin the SCE, users may call parallel routines interactively, often without necessity of special parallel computing knowledge. The parallel processing platform can be represented by a remote computer or a local parallel workstation. That means there is a distinction between remote and local SCE front ends. Since no modification of SCE code regarding parallel programming tasks is necessary (see Section 1.2), an implicit parallel programming model exists.

Using this approach, SCEs are development and runtime environments, but to run a parallel program, only a single SCE instance is necessary.
For remote front ends, coupling follows the client server model. The client is represented by the SCE, while the remote parallel system acts as server. Servers provide mostly linear algebra routines like BLAS or ScaLAPACK, but integration of user-defined routines is possible. Table 4 summarizes projects following the remote front end approach.

Project	SCE	Parallel Routines	Rel.
Netsolve [15]	MATLAB, Octave	linear algebra, user defined ro utines	1997
PLAPACK Server Interface [16]	MATLAB	linear algebra	1998
Matlab*P [10]	MATLAB	linear algebra, user defined ro utines	1999

Table 4: Projects following the front end approach (remote coupling).

In the field of local front ends, the SCE's computational module contains parallel numerical routines, executed on the local parallel computer or on a compound of local and remote computers. Parallel numerical standard routines as well as user-defined routines are possible. In the area of standard routines, internal SCE functions often use SIMD capabilities of modern CPUs like MMX or SSE extensions, for example in basic linear algebra, signal processing or trigonometry.
Standard routines which facilitate capabilities of MIMD structures like SMP or multi core CPUs are rarely integrated into SCEs. Currently, only IDL is known to facilitate such structures by its Multi-Threading libraries. User defined parallel routines can easily be integrated into the SCE's computational module by usage of SCE interfaces to compilable languages (see Section 2.2).

3.4 Multi-SCE Approach

The Multi-SCE approach is characterized by multiple conventional SCE instances being connected by a coupling platform, altogether representing a Multi-SCE. The coupling platform consists of a physical layer like Ethernet, shared memory or disk units and of a software layer providing low level programming access to physical devices.
Suitable low-level services to build up Multi-SCEs are:

- *Operating System services* like Sockets, System V IPC, File I/O
- *External middleware services* like Java sockets, PVM, MPI, CORBA, DCOM, HLA
- *SCE internal middleware services* like Matlab engine

On top of these low level services a high level interface has to be realized within the Multi-SCE. On one hand, high level interfaces have to support a well-defined parallel programming model (see Section 1.3). On the other hand, the interface should be well integrated into the matrix oriented SCE programming paradigm (see Section 2.1). To fulfill both demands, it is necessary to adapt the classical programming models to the matrix-oriented paradigm.

In real Multi-SCEs the following adaptations are found:

- Shared Memory Prog. → Shared Array Prog.
- Message Passing Prog. → Array Passing Prog.
- asynchronous RPC → vectorial RPC

Using the Multi-SCE approach, explicit parallel programming can be done in the SCE typical way of interactive working. In this approach, the whole SCE compound is both development and runtime environment. Hence, parallel program execution requires multiple SCE instances.
Due to its structure, the Multi-SCE approach is well suited for cluster and grid platforms and therefore attractive to a broad user community. This fact is reflected by the large number of Multi-SCE packages released during the last decade, being listed in Table 5. Since some Multi-SCE packages do not adapt parallel programming models in the way described above, programming models in Table 5 are classified by general terms, i.e. Message Passing (MP), Shared Memory (SHM) and RPC.

In [6] also combinations of SCE based parallel processing approaches have been proposed. The usage of such hybrid approaches offers ways to use the Multi-SCE approach only in development phases, while switching to the front end approach in production phases. On the other hand, hierarchies like front end approach on lower levels and Multi-SCE approach on higher levels are imaginable. Up to now, hybrid approaches have not been investigated in detail.

4 Multi-SCE Performance Analysis

To investigate Multi-SCE packages quantitatively, communication performance measurements have been executed. The aim of measurements applied is the ascertainment of comparable Multi-SCE spanning performance parameters.

Project	SCE	Low Level Service	Prog. Mod.	Rel.
PT Toolbox [17]	MATLAB	external (PVM)	MP	1995
DP Toolbox [2]	MATLAB	external (PVM)	MP, RPC	1995
MultiMatlab [3]	MATLAB	external (MPI)	MP	1996
Paralize [18]	MATLAB	OS (File I/O)	RPC	1997
PVM Tbx. [19]	Scilab	external (PVM)	MP	1998
PMI [20]	MATLAB	internal (engine)	RPC	1999
Matmarks [21]	MATLAB	external (ThreadMarks)	SHM	1999
PVM Tbx. [22]	MATLAB	external (PVM)	MP	1999
Cornell MT Tbx. [23]	MATLAB	external (MPI)	MP	2000
PLab [24]	MATLAB	OS (Sockets)	RPC	2000
ParMatlab [25]	MATLAB	OS (Sockets)	RPC	2001
MPI Toolbox [22]	MATLAB	external (MPI)	MP	2001
IDLPVM [26]	IDL	external (PVM)	MP	2001
MatlabMPI [27]	MATLAB	OS (File I/O)	MP	2001
Beolab Tbx. [28]	MATLAB	internal (engine)	RPC	2002
Parallelization Tk. [29]	MATLAB	internal (engine)	RPC	2002
DistributePP [30]	MATLAB	OS (File I/O)	RPC	2002
MPIDL [31]	IDL	external (MPI)	MP	2003
Parallel [32]	Gauss	external (Java sockets)	RPC	2003
Parallel Octave [33]	Octave	external (MPI)	MP	2003
Distributed Octave [34]	Octave	external (MPI)	RPC	2004
MPI Toolbox [35]	Octave	external (MPI)	MP	2004
DC Toolbox [36]	MATLAB	external (Java Jini, MPI)	MP, RPC	2004
pMatlab [37]	MATLAB	OS (File I/O)	SHM	2005
MDiCE [38]	MATLAB	OS (Sockets)	RPC	2005
Matlab2Matlab [39]	MATLAB	external (Java sockets)	RPC	2005
GAMMA [40]	MATLAB	external (Global arrays)	SHM	2006

Table 5: Projects following the Multi-SCE approach.

For parameter ascertainment, a programming model independent measurement method has been developed and applied. Measurements have been executed both with existing Multi-SCE packages and new Multi-SCE prototypes.

4.1 Measurement Method and Platform

Communication performance measurements have been applied using a round trip time (RTT) method with a one-to-one communication scheme.

Within this method, a master-slave program structure has been used, where the slave process reflects messages issued by the master process. To determine average round trip times, the send/receive process is repeated several times, altogether embedded into time measurement in the master process. The method described is well suited for packages using a message passing programming model. For RPC and shared memory models, this method has to be adapted to deliver comparable results.

In RPC programming, time measurements have been executed using a dummy function on the slave side, which just returns a single input parameter. On the master side, the dummy function is called remotely with a message as input parameter, resulting in the same message transfer as described above.

In shared memory programming, explicit synchronization has been applied. In the master process, a shared variable is written, followed by sending a synchronization signal to the slave. Subsequently, the master process blocks execution until receiving a synchronization signal from the slave, finalizing the send/receive process by a shared variable read. The slave process behaves analogically, so message transfer follows the scheme described above.

Round trip time measurements have been executed with different data sizes, resulting in a dependence of round trip time to data size. If round trip time is linearly dependent on message size, which is the case in executed investigations, linear regression of round trip time at different data sizes leads to communication performance significant parameters *byte rate* and *latency*. The following formula describes the dependencies:

$$t_{RoundTrip} = 2 \cdot \left(\frac{DataSize}{ByteRate} + t_{Latency} \right)$$

As measurement platform, two Linux computers running with kernel version 2.4.18 have been used. Computers are driven by a 1,500 MHz AMD Athlon CPU and are connected by Gigabit Ethernet.

4.2 Performance Results of Existing Multi-SCE Packages

Out of all 24 listed Multi-SCE packages in Table 5, only 8 packages could be investigated regarding communication performance. The remaining 16 packages were either outdated (not adapted to recent SCEs) or not available. Results for investigated Multi-SCE pakkages are summarized in Table 6.

The results presented show that communication performance of Multi-SCEs heavily depends on the used low-level service but weakly depends on the applied SCE.

Package, SCE	Low Level Service	Prog. Mod.	Byte Rate [MB/s]	Latency [ms]
DP Tbx. v1.5, MATLAB v7.1	PVM v3.4	MP	11.9	1.1
PVM Tbx., Scilab v2.7	PVM v3.4	MP	12.0	0.2
MPI Tbx., MATLAB v7.1	LAM v7.1 (MPI)	MP	36.7	0.1
MPI Tbx., Octave v2.1	LAM v7.1 (MPI)	MP	37.2	0.1
DC Tbx. v2.0, MATLAB v7.1	Jini	RPC	3.6	421.9
	MPICH2 v1.0	MP	34.1	0.3
MATLAB MPI v1.2 MATLAB v7.1	NFS	MP	25.5	44.8
Beolab Tbx., MATLAB v7.1	Matlab engine	RPC	37.1	81.8
Parallelization Tk. v1.2, MATLAB v7.1	Matlab engine	RPC	37.2	25.3

Table 6: Communication performance of existing Multi-SCE packages.

So byte rates of packages using PVM (DP Toolbox and PVM/Scilab) are nearly the same although SCEs differ. The same effect can be seen on packages using LAM MPI. Furthermore, packages that do not use message passing libraries show significant higher latencies than message passing library based packages.

4.3 Performance Results of New Multi-SCE Prototypes

For future designs of Multi-SCE packages, new Multi-SCE prototypes have been developed using MATLAB as targeted SCE. Aims of these prototype developments are feasibility studies and performance improvements compared to existing packages.

Up to now, five prototypes have proofed to be feasible (communication performance results of prototypes are shown in Table 7):

- Fast PVM prototype
- MPICH2 prototype
- Shared Arrays on top of MPICH2 prototype
- Matlab engine plus threads prototype
- Java sockets based Message Passing prototype

The fast PVM prototype is a component of the latest DP Toolbox release (v1.7.2), which has been entirely rewritten.

Prototype	Low Level Service	Prog. Mod.	Byte Rate [MB/s]	Latency [ms]
Fast PVM	PVM v3.4	MP	13.3	0.3
MPICH2	MPICH2 v1.0	MP	29.9	0.1
Shared Arrays	MPICH2 v1.0	SHM	11.0	1.7
Engine plus threads	Matlab engine, threads	RPC	31.0	20.9
Java MP	Java sockets, threads	MP	1.4	2.4

Table 7: Communication performance of Multi-SCE prototypes (SCE: Matlab v7.1).

It shows a slightly higher byte rate and lower latency than the interface used in DP v1.5.

The MPICH2 prototype is the first freely available MATLAB toolbox facilitating the MPICH2 library. Prototypes using other MPI libraries (e.g. LAM or MPICH) have shown to be unstable with MATLAB. Compared to Table 6, the MPICH2 prototype shows very low latency.

The *Shared Arrays prototype* is built on top of the MPICH2 prototype. It is intended to investigate mappings of parallel programming models, especially from message passing to shared memory programming. Since a distributed memory process model is the basis of this prototype, shared memory can only be virtual. In this prototype, a server process manages all shared variables, while shared memory accessing processes act as clients, issuing *put memory* and *get memory* commands. Compared to the MPICH2 prototype, which uses the same low-level service, this prototype shows lower byte rate and higher latency. This is caused by frequent send/receive commands on client/server communication as well as by additional barrier synchronization effort.

The *thread using MATLAB engine prototype* represents a new approach to control multiple Matlab Engines simultaneously. Matlab engine is a library that allows the invocation and control of a remote MATLAB instance out of a C program. To execute a remote command within the MATLAB instance, the blocking function *engEvalString()* has to be called. To use MATLAB engines in parallel computing, this blokking behavior needs to be circumvented. Existing Multi-SCE packages that use Matlab engines bypass blocking by misusing engine internal functionalities (write/read on engine internal file pointers, not working on Win32 MATLAB), introduced by PMI.

The thread using approach encapsulates all MATLAB engine calls into threads, leading to an engine implementation independent interface, working also with Win32 MATLAB. In comparison with other engine based pakkages, this prototype offers the lowest latency.
The *Java based message passing package* is a prototype of a platform independent message passing library, working with all MATLAB architectures (e.g. Win32, Linux, Mac, Solaris). Inside this library, TCP Socket accesses are encapsulated within threads leading to a communication that follows the message passing paradigm. This approach disappoints with the lowest byte rate out of all prototypes.

5 Characteristis of Simulation Applications

Since simulation applications are well supported in SCEs, for instance by ODE and PDE solvers or simulation subsystems like Simulink, Stateflow or Scicos, they are focused in application based investigations. Furthermore, simulation experiments and simulation based optimizations belong to the most time consuming applications in SCEs today.

To characterize investigated applications with respect to parallel processing, three qualitative criteria are used:

- Parallelism level
- Granularity
- Programming model applicability

5.1 Parallelism Level

From a programmer's point of view, simulation based optimization is accomplished at several program levels. On the highest level, an optimization method tries to find optimal input parameters of an objective function that has to be minimized. Within the optimization method, objective functions are called to calculate objective values for current input parameter sets. Within the objective function, results of simulation runs are used to calculate objective values.

Following this view, parallelism in simulation and optimization can occur on the described program levels, subsequently referred to as parallelism levels.

Parallelism levels can be categorized in the following way, ordered from highest to lowest program level:

- Independent optimization runs
- Parallel optimization methods
- Independent simulation runs
- Distributed models

Independent optimization runs are placed on the highest parallelism level. They occur, for instance if optimizations on multiple model operating points are necessary. In *parallel optimization methods* multiple objective values are calculated independently during one iteration step. Examples of parallel optimization methods are evolutionary strategies or Monte Carlo methods. *Independent simulation runs* do not have to be embedded into objective functions necessarily, but can also represent stand-alone simulation experiments. They occur, for instance in parameter studies. *Distributed models* are placed on the lowest parallelism level. At this level, a model is split up into partial models, subsequently simulated by multiple cooperative processes. In discrete event simulation, those processes are referred to as *logical processes* ([41]).

5.2 Granularity

In terms of parallel processing, granularity denotes the ratio of computation effort to communication effort in a parallel program. Distinctions are made between coarse, middle and fine granular programs. A program shows coarse granularity, if communication effort is negligible compared to computation effort. On the other hand, fine granularity exists, if communication effort dominates seriously over computation effort. Granularity represents a qualitative measure and is also hardware dependent. So programs that show middle granularity on one hardware, could be classified as fine granular on another hardware.

Regarding parallelism level, granularity tends to decrease from the highest to the lowest level.

5.3 Programming Model Applicability

Programming model applicability means the applicability of a parallel programming model to parallelize certain application structures. Since RPC programming is mappable onto shared memory and message passing programming (see sect. 1.3), every application being parallelizable by RPC is also parallelizable by shared memory or message passing programming.

Hence, a classification regarding program model applicability distinguishes two classes of applications:

- *RPC capable*: parallelizable by message passing, shared memory or RPC programming
- *Non RPC capable*: parallelizable only by message passing or shared memory programming

In simulation based applications, programming model applicability classification can be assigned unambiguously to parallelism levels, as shown in Table 8.

Parallelism level	Programming model applicability
Independent optimization runs	RPC capable
Parallel optimization methods	RPC capable
Independent simulation runs	RPC capable
Distributed models	non RPC capable

Table 8: Relation of parallelism level and programming model applicability in simulation and optimization applications.

6 Application Performance Analysis

In this section, a number of optimization and simulation applications, being parallelized on basis of the Multi-SCE approach are discussed. Application based investigations comprise benchmark problems as well as real life applications from industry and research. In Table 9, applications are listed including their parallelism level and granularity degree. In every application, parallelization took place only on the highest possible parallelization level.

Application	Parallelism Level	Granularity
Parameter optimization of an exhaust gas model [42]	independent optimization runs	coarse
Evolutionary safety testing of embedded control software [43]	parallel optimization method	coarse
Parameter study of a dynamic system [7]	independent simulation runs	coarse
Simulation of Cavity Flow by the Lattice Boltzmann Method [44]	distributed models	middle
Solution of a PDE by finite differences [7]	distributed models	fine
Simulation of coupled ODE systems [7]	distributed models	fine

Table 9: Parallelism level and granularity of investigated applications.

All investigations have been performed on a cluster computer, representing a distributed memory MIMD architecture. Cluster nodes are driven by a 1.500 MHz AMD Athlon CPU and are connected by Gigabit Ethernet. Operating system is GNU/Linux with kernel version 2.4.18.

The *parameter optimization of an exhaust gas model* represents a real life application from automotive industry. Typically, the optimization has to be performed for a large number of operating points of the same simulation model. Therefore, the parallelism level is independent optimization runs. With a number of 12 processors, a speedup of 10.3 has been reached on the test platform. As expected, this application shows the highest efficiency of 0.9 (efficiency = speedup divided by number of processors) among all selected applications, caused by parallelism level and granularity. A more detailed representation of this application can be found in [42].

The *evolutionary safety testing of embedded control* is also a real life application from automotive industry. In contrast to the first example it is a single optimization problem with embedded simulation. Nevertheless, the parallelism level is still in the optimization because an evolutionary search algorithm is used. Therefore, in each optimization step multiple simulation based objective values can be calculated simultaneously.
The general structure of this application is well suited for parallelization in principle. But the investigation of a certain application configuration has shown only a speedup of 4.7 with 12 processors on the test platform, leading to an efficiency of 0.4. The reason for this weak result is a relatively small-scaled embedded simulation in combination with very high latencies of the applied Multi-SCE package (see Table 10 and Table 6). This example illustrates that the relation of parallelism level and granularity is only of structural nature and does not guarantee good speedup values in each application case.

The *parameter study of a dynamic system* is a benchmark problem of the *SNE Comparison CP1*. In this example, a mass spring system has to be simulated with varying parameters, while system responses have to be averaged. Hence, the parallelism level is independent simulation runs. On the test platform, the investigated implementation has reached a speedup of 5.6 under usage of 8 processors, corresponding to an efficiency of 0.7. Some more details of the benchmark solution are published in [45].

The simulation of *Cavity Flow by the Lattice Boltzmann Method* is also a typical benchmark problem. In the example investigated, an incompressible fluid is bounded in a square cavity, driven by a uniform flow on the top boundary. Parallelization takes place by domain decomposition of the cavity in *y*-direction. In analogy to distributions in discrete event and continuous models, the applied decomposition can be seen as a model distribution approach.

Because of more or less strong dependencies between distributed model parts such applications are structurally not coarse grained. The investigated configuration has reached a speedup of 5.5 using 12 processors, corresponding an efficiency of 0.5.

The *solution of a PDE by finite differences* is a second benchmark problem from the *SNE Comparison CP1*. In this example, the motion of a swinging rope, described by a partial differential equation, is simulated under usage of the finite difference method. Similar to the above Lattice Boltzmann problem, parallelization takes place on the model level by domain decomposition of the rope. In this example, the computation effort in each distributed model part becomes very small compared to the necessary communication effort. Therefore, the application has not been speeded up on the test platform successfully. Actually, run time increases dramatically in comparison to the sequential solution. As in every non coarse grained application, the reachable speedup strongly depends on the run time platform applied. Therefore, also successful solutions of this benchmark have been published within the scope of *SNE Comparison CP1*.

The *simulation of coupled ODE systems* is a third benchmark problem from the *SNE Comparison CP1*. In this example, five coupled predator-prey systems are simulated. This benchmark problem is an example of the model distribution approach in continuous simulation. Due to the usually tight coupling between continuous model parts, a successful parallelization of such problems is mostly impossible on distributed memory MIMD systems. Therefore, also with this benchmark no speedup has been reached on the test platform.

Table 10 summarizes performance results of investigated simulation and optimization based application problems and shows the involved Multi-SCE packages.

Investigations of program parallelization efforts have shown that for RPC capable applications RPC programming is the most convenient model. Due to the absence of explicit communication and synchronization, RPC programming results in the lowest number of code lines compared to message passing and shared memory programming. Prerequisite for applying RPC programming is a modular, that means strictly function based, sequential program. In real life applications, this condition is often violated, for example by script programming or by accessing global variables.

Application	Multi-SCE package	np	Speed up	Effi-cieny
Parameter optimization of an exhaust gas model [42]	DP Tbx. v1.5.0	12	10.3	0.9
Evolutionary safety testing of embedded control software [43]	DC Tbx. V2.0	12	4.7	0.4
Parameter study of a dynamic system [7]	DP Tbx. v1.5.0	8	5.6	0.7
Simulation of Cavity Flow by the Lattice Boltzmann Method [44]	DP Tbx. v1.7.0	12	5.5	0.5
Solution of a PDE by finite differences [7]	DP Tbx. v1.5.0	8	<1	-
Simulation of coupled ODE systems [7]	DP Tbx. v1.5.0	5	<1	-

Table 10: Performance of investigated applications (np: number of processors).

In such cases, message passing programming can be applied more efficiently, since there are no restrictions with respect to script or function execution and variable accesses.

7 Conclusion

Regarding parallel processing in general, it was shown that hardware specific classifications of Flynn and memory structure could be seen as basic taxonomies for hardware and software structures, as well.

The analysis of hardware structures in small scale and large scale parallel processing has demonstrated that in both domains MIMD systems dominate, so further distinctions with respect to memory structure are necessary.

It was assumed that in small scale parallel processing constellations consisting of multi core workstations will prevail in the near future. The discussion of programming models has shown that beside the two common explicit parallel programming models, shared memory and message passing programming, RPC programming is a convenient model for specific problems.

It was demonstrated that for the usage of RPC programming in parallel processing, certain adaptations to the classical RPC programming have to be made, namely asynchronous or vectorial RPC.

With respect to scientific and technical computing environments, typical denoting features of such systems have been presented. Besides SCE based parallel processing, several other ways of SCE program acceleration have been discussed.

Regarding SCE based parallel processing, a new taxonomy combining two existing taxonomies have been presented. Principles of all classes of the new taxonomy have been discussed and more than 30 representatives have been identified and categorized.

In the field of the Multi-SCE approach, a comparable communication performance measurement method for all parallel programming models was developed enabling comparisons on the basis of two parameters: latency and byte rate.
Results of communication performance measurements of existing Multi-SCE packages have been presented. New Multi-SCE prototypes have been discussed and communication performance has been compared to existing packages.

With respect to applications, general characteristics of parallel simulation and optimization problems have been discussed. Especially, a parallelization level based scheme has been presented, allowing the characterization of both parallel optimization and simulation problems.
Regarding certain applications being parallelized under usage of Multi-SCE packages, formerly introduced characteristics have been applied. Parallel performance results have been presented and discussed with respect to application's characteristics.

References

[1] C. Moler: *Why there isn't a parallel MATLAB*. Matlab News & Notes, Spring 1995.

[2] S. Pawletta, W. Drewelow, P. Dünow, T. Pawletta, M. Süße: *A MATLAB toolbox for distributed and parallel processing*. 2nd International MATLAB Conference, Cambridge, 10/1995.

[3] A. E. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, L. N. Trefethen: *MultiMATLAB: MATLAB on Multiple Processors*. Technical Report, Cornell Theory Center, 1996.

[4] The MathWorks, Inc.: *Distributed Computing Toolbox For Use with MAT LAB.* User's Guide, Version 1, 11/2004.

[5] M. J. Flynn: *Some Computer Organizations and Their Effectiveness*. In IEEE Transactions on Computers, Vol. C-21, No. 9, 09/1972.

[6] S. Pawletta: *Erweiterung eines wissenschaftlich-technischen Berechnungs- und Visualisierungssystems zu einer Entwicklungsumgebung für parallele Applikationen*. Dissertation, Universität Rostock, 06/1998.

[7] F. Breitenecker, I. Husinsky, G. Schuster: *Comparison of parallel simulation techniques*. In Simulation News Europe, Issue 10, 03/1994.

[8] D. B. Skillicorn, D. Talia: *Models and Languages for Parallel Computing*. In ACM Computing Surveys, Vol. 30, No. 2, 06/1998.

[9] A. L. Ananda, B. H. Tay, E. K. Kohn: *A Survey of Asynchronous Remote Procedure Calls*. In ACM SIGOPS Operating Systems Review, Vol. 26, Issue 2, 04/1992.

[10] R. Choy, A. Edelman: *Parallel MATLAB - Doing it Right*. Technical Report, Computer Science AI Laboratory, Massachusetts Institute of Technology, 11/2003.

[11] P. Drakenberg, P. Jacobson, B. Kagström: *A CONLAB Compiler for a Distributed Memory Multicomputer*. In Proc. 6th SIAM Conference on Parallel Processing for Scientific Computing, 03/1993.

[12] L. DeRose, D. Padua: *A MATLAB to Fortran 90 Translator and its Effectiveness*. In Proc. 10th International Conference on Supercomputing, 01/1996.

[13] S. Chauveau, F. Bodin: *Menhir - An Environment for High Performance Matlab*. In 4th International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers, 05/1998.

[14] M. J. Quinn, A. Malishevsky, N. Seelam: *Otter: Bridging the Gap between MATLAB and ScaLAPACK*. In Proc. 7th IEEE Inter national Symposium on High Performance Distributed Computing, 07/1998.

[15] K. Seymour, A. YarKhan, S. Agrawal, J. Dongarra: *NetSolve: Grid Enabling Scientific Computing*. In Grid Computing: The New Frontier of High Performance Computing, Vol. 14, Elsevier, Advances in Parallel Computing, 05/2005.

[16] G. Morrow, R. van de Geijn: *A Parallel Linear Algebra Server for Matlab-like Environments*. In Proc. 1998 ACM/IEEE conference on Supercomputing, 11/1998.
[17] V. P. Pauca, J. Hollingsworth, K. Liu: *User's Guide for the Parallel Toolbox for MATLAB*. Technical Report, Wake Forest University, 05/1995.
[18] T. Abrahamson: *Paralize*. MATLAB Central File Exchange, 11/1997.
[19] Scilab Group: *PVM parallel toolbox*. Scilab Documentation (4.0 Version), 02/2006.
[20] D. Lee: *PMI*. MATLAB Central File Exchange, 03/1999.
[21] G. Almasi, C. Cascaval, D. A. Padua: *MATmarks - A Shared Memory Environment for MATLAB Programming.* In Proc. 8th IEEE International Symposium on High Performance Distributed Computing, 08/1999.
[22] J. F. Baldomero: *Message Passing under MATLAB*. Advanced Simulation Technologies Confer-ence (ASTC), Seattle Washington, 04/2001.
[23] J. Zollweg: *Cornell Multitask Toolbox for MATLAB*. Cornell Theory Center, 2006.
[24] U. Kjems: *PLab*. Technical University of Denmark, 11/2000.
[25] L. Andrade: *parmatlab*. MATLAB Central File Exchange, 04/2001.
[26] Kilvarock Corp.: *IDL to PVM interface*. 08/2001.
[27] J. Kepner: *Parallel Programming with MatlabMPI*. High Performance Embedded Computing Workshop, MIT Lincoln Lab., 11/2001.
[28] T. Abrahamsson: *Beolab Toolbox for v6.5*. MATLAB Central File Exchange, 01/2002.
[29] E. Heiberg: *MATLAB Parallelization Toolkit 1.20*. MATLAB Central File Exchange, 01/2002.
[30] M. D. DeVore: *DistributePP*. MATLAB Central File Exchange, 02/2002.
[31] D. Mastrovito: *MPIDL Overview*. Princeton Plasma Physics Laboratory, 01/2003.
[32] M. Ford: *Parallel V1.1: Multi workspace GAUSS plus networking*. Forward Computing and Control Pty. Ltd., 09/2003.
[33] T. Obara: *Parallel Octave*. Tohoku University, 12/2003.
[34] J. D. Cole: *Distributed Octave*. Transient Research, 04/2004.
[35] J. F. Baldomero: *MPI Toolbox for Octave*. In 6th International Meeting on High Performance Computing for Computational Science, Valencia, 06/2004.
[36] The MathWorks, Inc.: *Distributed Computing Toolbox For Use with MATLAB*. User's Guide, Version 2, 03/2006.
[37] H. Kim, J. Mullen: *Introduction to Parallel Programming and pMatlab v0.7*. MIT Lincoln Lab., 02/2005.
[38] D. Gruber, C. Kastinger, T. Mayerdorfer, S. Zorn-Pauli: *MDiCE R2.0 - MultiMDiCE*. Carinthia Tech Inst., Klagenfurt, 07/2005.
[39] B. Phelan: *Matlab 2 Matlab : Distributed Computing Toolbox*. XTargets, 11/2005.
[40] R. Panuganti et al: *GAMMA: Global Arrays meets MATLAB*. Technical Report, Ohio State University, 01/2006.
[41] H. Mehl: *Methoden verteilter Simulation*. Vieweg, Braunschweig/Wiesbaden, 1994.
[42] R. Fink, S. Pawletta, M. Schultalbers: *Matlab-based parallel optimization with integrated simulation.* In 5th EUROSIM Congress, ESIEE Paris, 09/2004.
[43] H. Pohlheim, M. Conrad, A. Griep: *Evolutionary Safety Testing of Embedded Control Software by Automatically Generating Compact Test Data Sequences*. In SAE 2005 Transactions Journal of Passenger Cars: Mechanical Systems, 02/2006.
[44] S. Hou, Q. Zou, S. Chen, G. D. Doolen, A. C. Cogley: *Simulation of Cavity Flow by the Lattice Boltzmann Method*. J. Comp. Physics, Vol. 118, Issue 2, 05/1995.
[45] R. Fink, S. Pawletta, T. Pawletta: *A Matlab based Solution to ARGESIM 'Comparison of Parallel Simulation Techniques' using DP-Toolbox*. In SNE, Issue 38/39, 12/2003.

[*] The research is partly supported by BMBF and BM M-V (HWP310 - FHW13).

Corresponding author: René Fink
Rene Fink, Sven Pawletta, Thorsten Pawletta
Res. Group Computational Engineering and Automation, Wismar University, University of Technology, Business and Design, PF 1210, 23952 Wismar, Germany
WWW.MB.HS-WISMAR.DE/cea
Bernhard Lampe, Institute of Automation
University of Rostock, Richard-Wagner-Str. 31/H.8
18119 Rostock, Germany; *bernhard.lampe@uni-rostock.de*

Received: June 15, 2006
Revised: June 29, 2006
Accepted: July 20, 2006

HLA Applied to Military Ship Design Process

Christian Stenzel, Sven Pawletta, Wismar University, Germany
christianstenzel@gmx.net, WWW.MB.HS-WISMAR.DE/cea
Richard Ems, Petra Bünning, MTG Marinetechnik GmbH, Hamburg, Germany
{*Richard.Ems; Petra.Buenning*}*@mtg-marinetechnik.de*

This article reports on an ongoing project in the field of naval architecture where existing implementations for simulating surface vessels in seaway have to be integrated into an HLA compliant distributed simulation and visualization federation. Like in other engineering fields, existing Fortran codes for extensive numerical problems play an important role in the ship design process. Unfortunately, relevant RTI implementations provide language bindings only for C++ and Java. Therefore, it is currently not straightforward to build HLA federates using existing Fortran codes. The article points out possible coupling variants between an RTI and Fortran code and discusses pros and cons. It is also shown how this research is influenced by experiences from related efforts to provide Matlab/HLA connectivity. Subsequently, the current implementation state of the simulation federation is presented. Finally, the overall development effort is evaluated and possible ways towards complexity reduction are pointed out.

Introduction

In 2002, after successful completion of the *Simulation Based Design and Virtual Prototyping* (SBDVP) program, the *NATO Naval Armaments Group* NG6 has charged the formal sub-group SG61 with establishing standards for modeling and simulation in naval ship acquisition. One of the objectives of this subgroup is the development of the *Virtual Ships* (VS) STANAG ([2], [5]). The VS STANAG is currently in the final draft state and in preparation for ratification. What is technically most important, is that the VS STANAG defines a simulation architecture for virtual ships based on the HLA standard. The standardization activities of the NATO indicate that also in Europe the HLA technology becomes important as strategic market factor for simulation and military suppliers in future.

The MTG Marinetechnik GmbH is an independent center of excellence for planning and designing surface warships and operates already since 1966 primarily for the German Navy. In cooperation with the *Federal Office of Defense Technology and Procurement* (BWB) MTG already deals with the topic Simulation Based Design and Virtual Prototyping for a while. Thus, the computer-aided model VORGES was already conceived for the development and evaluation of ship designs.
The main objective of VORGES is to point out possible realization variants for the future ship design. Furthermore, VORGES should support the selection of a ready to build solution as well as the process of construction, proving, and operation control ([9]). During this ship design process the simulation tools and models utilized have to interoperate.

Thereby, it has to be distinguished between non-runtime and runtime interoperability. *Non-runtime interoperability* can be achieved through use of shared databases, common access to *Product Data Management* (PDM) systems, and data exchange standards ([2]). *Runtime interoperability* has to be achieved via HLA technology because that is mandatory to become compliant with the upcoming VS STANAG.

Therefore, MTG started a project in cooperation with Wismar University in 2005 to explore feasibility and effort of HLA connectivity between existing simulation software. MTG decided to start with a medium scale problem - called *SIMBELFederation* - of connecting two existing Fortran codes for simulating ship motion in seaway and computing seaway heights in a spatially bounded region with a third visualization component. At first view it seems that only the technical problem of linking Fortran code with an RTI implementation has to be overcome to solve the overall problem. After more general examination one realizes that the problem belongs to an HLA application domain with specific characteristics.

In [8] three dissimilar approaches to build HLA based federates are distinguished, which fulfill the specific needs of different application domains:

1.) *Implementation (programming) of federates using common object-oriented programming languages*. That is the approach for which HLA RTIs are originally designed for. Consequently, all relevant RTI implementations provide C++ and Java language bindings.
This approach seems to meet the needs of a wide range of defense simulation applications because that is the domain for which HLA and RTI development was initiated by the U.S. Department of Defense.

2.) *Implementation (modeling) of federates using simulation tools*. This approach is still more a need than reality from the perspective of possible industrial application domains, where it is common to utilize simulation tools. Although extensive research on interfacing HLA RTIs from so-called COTS (*commercial off-the-shelf*) simulation packages has produced solutions in principle ([10]), HLA support is not widely provided by today's COTS.
3.) Beside the above domains, there exists a community in various fields of engineering where neither C++ and Java programming nor modeling with COTS is the preferred approach. Characteristic problems in this domain are extensive numerical simulations and other computations, which are traditionally coded in Fortran. In some areas (e.g. control engineering) Fortran coding has been replaced by computing environments like MATLAB.

The *SIMBELFederation* problem obviously belongs to the last domain. Unfortunately, also in this domain, HLA connectivity is not a matter of course yet. But at least for the MATLAB computing environment there exist experiences from earlier research and also some commercial solutions are available in the meanwhile. Some essential aspects of MATLAB/HLA integration and the current state of the art in this branch are summarized in Section 1. Basic coupling variants between Fortran and HLA RTIs are discussed in Section 2. Section 3 outlines the *SIMBELFederation* problem and its current implementation state. In the Conclusions Section, the overall development effort is discussed and possible ways towards complexity reduction are pointed out.

1 MATLAB/HLA Connectivity

In 1998 the development of an *HLA toolbox* for MATLAB was started at the University of Rostock. Essential design challenges and solutions were published in [7], [8]. A generalization to the entire class of SCEs (*Scientific and Technical Computing Environments*) can be found in [6]. Results of this project which are also meaningful for the problem of Fortran/HLA connectivity are summarized in the following subsections.
In 2005, the first two commercial solutions for MATLAB/HLA connectivity have been released. They are viewed in the last subsection briefly.

1.1 RTI Linkage

The concrete structure of an RTI implementation is vendor dependent. However, the connection between an application and a certain RTI is always realized in a uniform way. For C++ coded applications, RTI implementations provide interface libraries, which have to be linked. This technique can also be used to build a connection between MATLAB and an RTI.
For extension purposes, MATLAB provides a number of external interfaces. One of them is the so-called MEX-interface, which allows dynamic linkage of C code. By this way also C++ libraries can be linked if C style name mangling is enforced (extern ‘C’).

1.2 Procedural HLA Interface

The HLA interface is specified in an object-oriented manner and consists of two fundamental classes which define the *RTI Ambassador* and the *Federate Ambassador*. But as a rule, an application is not simultaneously federate in more than one federation. Hence, only single RTI and Federate Ambassador instances are needed per application.
Consequently, in a MATLAB/HLA integration it is possible to instantiate the two ambassadors below the MEX-interface on the C/C++ layer. Then, on the MATLAB layer only a procedural interface is required to access RTI services and to provide federate services.
For simple handling of an HLA interface in MATLAB, the original very long designations of the RTI and federate services should be replaced by abbreviations. This is in particular necessary for the interactive way of working in MATLAB. Since from the huge number of federate services typically only a few are needed in an application, a MATLAB/HLA-interface should provide predefined federate services.

1.3 Vectorization

RTI and federate services perform scalar operations with elementary data objects as parameter as it is usual in conventional programming. However, efficient MATLAB programming is based on vectorization, whereby code complexity is reduced considerably. Therefore, the routines of a MATLAB/HLA-interface should be vectorized as much as possible.
Due to vectorization not only application code is simplified but also the complexity of a MATLAB/HLA-interface is reduced. For example, in the HLA Toolbox presented in [8] which is based on a DMSO RTI more than 80 auxiliary routines of the primary C++ interface became unnecessary in the MATLAB/HLA interface.

1.4 Commercial Tools

The first commercial product for MATLAB/HLA connectivity was the HLA Toolbox from ForwardSim, Inc. released in May 2005 ([1]). It provides a procedural MATLAB/HLA interface with some features stated in Subsection 1.2. By now, the toolbox supports only HLA 1.3 compliant RTI implementations.

Since July 2005, MÄK Technologies, Inc. offers the product The MÄK HLA/DIS Toolbox for MATLAB and Simulink ([4]). But in the above sense this product is not a direct MATLAB/HLA-interface. Rather the toolbox provides a MATLAB interface to VR-Link. VR-Link is also a product of MÄK Technologies which provides a higher middleware-independent interface. VR-Link can run on top of DIS, HLA 1.3, and IEEE 1516 compliant middleware.

2 Fortran/HLA Connectivity

In various fields of engineering, especially where complex numerical problems have to be solved, Fortran is still the dominating programming language. This fact will probably not change in future because there are a large number of well-tested Fortran codes; the language is continuously refined and standardized. Furthermore, Fortran is the most common language in High Performance Computing (HPC).

In the original HLA main application areas, which are *Distributed Virtual Training Environments* (DTVE) and wargaming, Fortran has no importance. Therefore, established RTI implementations offer programming interfaces for C++ and Java, but not for Fortran.
Currently, development of HLA federates based on Fortran or by using existing Fortran codes is only possilbe by applying one of two indirect methods: The first method requires an appropriate modularization of the Fortran code to allow subsequent integration into an HLA-capable language environment. Then, all HLA specific parts have to be implemented in this language environment. The second method is to employ an HLA interface within Fortran that permits access to an external RTI implementation. Problem specific as well as HLA specific parts are coded in Fortran.

2.1 Integration of Fortran Routines into HLA-Capable Language Environments

Appropriate language environments for this approach are C++, Java, and Matlab. In the following, basic solutions for these three language environments are discussed.
On the object code level, usual Fortran and C++ compilers have very different naming conventions for subroutines and methods, respectively. Therefore, linkage of Fortran and C++ object code is impossible without taking special precautions. But most C++ compilers can provide C-style naming on demand. C and Fortran naming are not identical but the differences are small and can be handled on source code level.

The following code fragments illustrate, how Fortran routines can be integrated into a C++ program:

```
C  Computing routine implemented in Fortran
   SUBROUTINE COMPUTE(result)
     DOUBLE PRECISION result
     ...
     END
/* External declaration of the Fortran computing
   routine as C++ function with C-style naming */
extern "C"{
    extern void compute_(double *result);
}
/* Usage of the Fortran computing routine
    in a C++ program                          */
int main () {
    ...
    compute_(&result);
    rtiamb.updateAttributeValues(...);
    ...
}
```

The code fragments are based on name conventions of the GNU compiler suite and are therefore not generally valid. Hence, this approach is compiler dependent.
Furthermore, differences between representations of data objects in memory in Fortran and C++ have to be taken into consideration. Especially, if arrays are passed as parameters, conversion from the column-oriented representation in Fortran to the row-oriented representation in C++ is required. Such conversions are potentially error-prone.

Java offers integration of extern software via the *Java Native Interface* (JNI). However, only C/C++ is directly supported by JNI. Due to that fact, integration of Fortran routines is only possible by using additional C/C++ wrappers. Hence, for this approach all problems already discussed for the C++ integration have to be solved. In addition, some more platform specific conversions for correct parameter passing between C/C++ and Java need to be accomplished. Consequently, error-proneness increases and runtime performance decreases.

In contrast to C++ and Java, MATLAB with its MEX-Interface supports the integration of Fortran routines very well. Parameter passing between Fortran and MATLAB is also well-supported by the MEX-Interface. No extensive conversions are necessary because MATLAB uses the same representation of array data types like Fortran.
Main advantage of the approach to integrate Fortran routines into HLA-capable language environments is that it is not necessary to implement an HLA interface within Fortran. Thus, realization of HLA federates on basis of existing small and medium Fortran codes is possible without greater effort. The approach is not suitable if HLA federates have to be build using large and complex Fortran codes.

In such cases, it can be difficult to modularize the codes into subroutines in such a way that all HLA parts can be done on top of these routines in the integrating language environment. From a structural and technical point of view, implementations based on the integration approach have to be characterized as ad hoc solutions. The approach enforces the separation of problem specific from HLA specific implementation, which can lead to insufficient structures. The Fortran integration technique into C++ and particular into Java is very wasteful and error-prone.

2.2 Fortran Integrated HLA Access

The integration approach discussed in the previous subsection does not provide direct HLA access within Fortran. But actual Fortran integrated HLA access can be realized in a similar way, as presented for MATLAB in Section 1. Therefore, in analogy, the necessary software layer, which realizes the Fortran/HLA connectivity, is called *HLA toolbox for Fortran*. Prototypes of such an HLA toolbox are currently developed at Wismar University. Essential design issues are discussed subsequently.

A Fortran/HLA toolbox has to work on top of common RTI implementations, which provide C++ and Java interfaces. The C++ interfaces are clearly preferred to build up a Fortran/HLA toolbox. If Java interfaces were used, additional mapping and conversion problems would occur like in the integration approach discussed in the previous subsection.
A Fortran/HLA toolbox which is internally based on a C++ interface has to link an RTI class library. That can be done by mapping the relevant C++ class methods to ordinary functions for which C style naming is enforced.
Like in the MATLAB/HLA toolbox, the *RTI Ambassador* as well as the *Federate Ambassador* can be internally instantiated in the C++ layer. Thus, it becomes possible to build up a purely procedural Fortran interface consisting of subroutines to access RTI services and to provide federate services. As in MATLAB, the very long designations of the RTI and the federate services should be replaced by abbreviations, because it is common Fortran-style to use short designators. Likewise, a Fortran/HLA toolbox should provide predefined federate services. Further design issues depend on the concrete Fortran version. Relevant versions are FORTRAN 77 and Fortran 90/95. FORTRAN 77 does not support vectorization and optional subroutine parameters. Therefore, the interface of an HLA toolbox for FORTRAN 77 cannot be simplified so much as it is possible for Fortran 90/95 and MATLAB.

3 The SIMBELFederation

Figure 1 shows a simplified architecture of the so-called *SIMBELFederation*. It is a medium scale problem consisting of real-world software components used in the ship design process. It serves as test problem for a stepwise examination of suitable solutions for integrating existing software with the HLA technology, especially Fortran codes of various scale. As result of this ongoing research, it should be possible for future HLA projects to form a realistic estimate of the overall development effort and reachable runtime performance in advance.

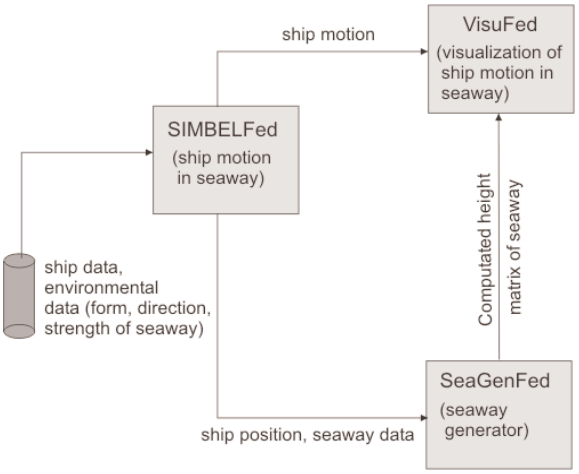


Figure 1: Simulation compound SIMBELFederation.

The problem examined consists of three components, which should work together in an HLA federation. *SIMBELFed* is a simulator federate which computes the motion and position of a ship in seaway. The computing federate *SeaGenFed* generates a spatial representation of the seaway around the ship position. The visualization of the ship motion in seaway takes place in the federate *VisuFed*.
The implementation state subsequently discussed is based on the DMSO RTI-1.3NG. Further investigations will also include commercial RTI implementations.

3.1 SIMBELFed

The federate *SIMBELFed* has to integrate the simulation package *SIMBEL* which is used to compute hydrodynamic forces and moments of monohull and multihull ships in different seaways. This permits conclusions about the suitability of ship designs in the operating range supposed. Already since the end of the 1980s, the simulation package *SIMBEL* is developed in cooperation with TU Hamburg-Harburg and the MTG. It is coded in FORTRAN 77 completely and comprises currently approx. 50.000 lines of code.

Because of this high complexity, a first step was to realize an offline-coupling of the *SIMBEL* simulator and the federation through files (Figure 2). In this case, the *SIMBELFed* federate has to contain only simple Fortran routines for reading out simulation data. Consequently, the appropriate integration approach in terms of Section 2 is to insert the Fortran routines into a C++ program, where necessary HLA parts have to be implemented. But this approach is not suitable for the intended online integration of the *SIMBEL* simulator. In that case, a fully operational Fortran/HLA-interface will be necessary.

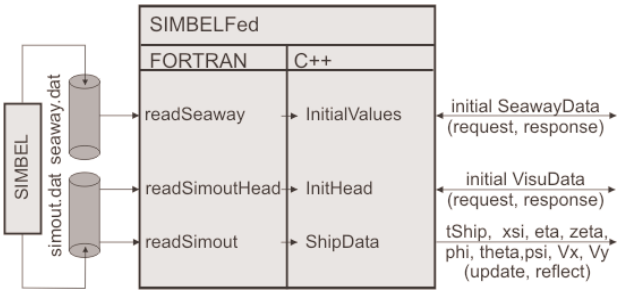


Figure 2: Current state of federate *SIMBELFed*.

During initialization *SIMBELFed* reads the file seaway.dat using the Fortran routine readSeaway. The content of this file describes initial environmental data like form, direction, and strength of seaway. The data are hold in the C++ class InitialValues. The file simout.dat contains among others header information about the simulated ship type, which is also read during initialization. All initial data are made available to the federation by the HLA request/response mechanism.

After initialization, the simulated ship motion data are read time stepwise from the file simout.dat. It contains motion data of six degrees of freedom, where *xsi*, *eta*, and *zeta* represent the translations and *phi*, *theta*, and *psi* the rotations in the directions *x*, *y*, and *z*. Furthermore, the ship velocity *Vx* in *x* and *Vy* in *y* direction as well as the timestamp *tShip* are included. After reading, the data are made available to the federation by the HLA update/reflect mechanism.

3.2 SeaGenFed

For the generation of a seaway height matrix also existing Fortran code should be used. It permits the computation of complex natural seaways on basis of a JONSWAP spectrum. The amplitudes of the individual wave components are additively overlaid for each time step and point of the seaway height matrix. The computation effort rises quadratically with the grid size. Currently, grid sizes up to 64x64 points can be computed in real time on a standard PC. The demand for computability of the seaway height matrix in real time arises from the presence of the visualization component *VisuFed* in the federation. If it is necessary to generate seaways of higher accuracy in the future, parallel processing has to be employed. With this in mind, the placement of the seaway generation into a separate federate is very meaningful.

Currently, the federate *SeaGenFed* contains only sequential FORTRAN 77 code, which could be easily integrated into a C++ program. The necessary HLA parts are implemented on the C++ level (see Figure 3). If in future the federate is to employ parallel processing, at least parts of the exiting code have to be ported to Fortran 90. Then it is more efficient to implement also HLA parts in Fortran following the approach described in Section 2.2.

All input data of the federate *SeaGenFed* are obtained from the federate *SIMBELFed*. The initial environmental data *SeawayData* are received during initialization. The current central ship position presented by *xsi* and *eta* is transferred time stepwise. On that basis, *SeaGenFed* computes a seaway height matrix *zmatrix*, which is spatially centered around the central ship position. Subsequently, *zmatrix* is communicated together with its position in the *xy*-plane and with a timestamp *tSeaway* using the HLA update/reflect mechanism.

3.3 VisuFed

The federate *VisuFed* is the primary interface of the federation to the user. It visualizes the moving ship in seaway as a photorealistic three-dimensional representation. Additionally, two-dimensional representations of the ship motion can be displayed.

Recent visualization software is mostly written in C++ or Java. Therefore, there usually exists no HLA connectivity problem. That applies also to the federate *VisuFed* presented in Figure 4, which is currently based on visualization software written in Java.

4 Conclusions

Due to the establishment of new standards the HLA technology will play an important role also in the military ship design process.

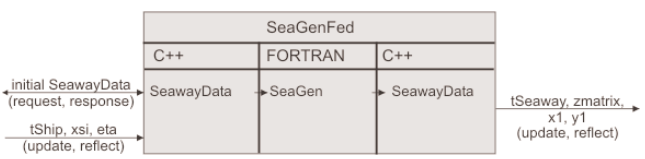


Figure 3: Current state of federate *SeaGenFed.*

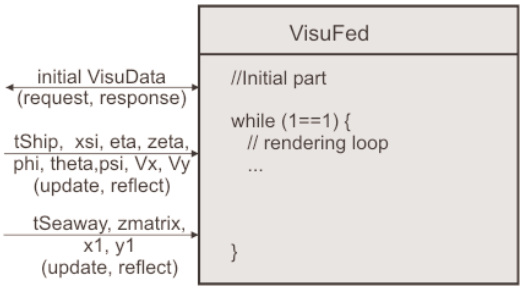


Figure 4: Current state of federate *VisuFed.*

On the other side, simulation- and other engineering software have been developed with large effort over many years and have to be reused as much as possible in future. Therefore, it is necessary to investigate how HLA connectivity can be reached for existing software, and which development effort has to be spent for that.

The research presented in this article is based on earlier works in the field of Matlab/HLA connectivity, which have provided important patterns for the development of Fortran/HLA connectivity. Both, MATLAB and Fortran are essential language environments in many engineering fields. For Fortran two basic coupling approaches were examined:

- Fortran integration in HLA-capable language environments like C++, Java, and MATLAB,
- Fortran integrated HLA access according to the pattern of an HLA toolbox.

The approach exposed first can only be viewed as ad hoc solution for small and medium scale problems. For C++ and Java as integrating language environments it is potentially error-prone and difficult to handle for application developers, because special knowledge in different language environments is required.

The second approach is well suited for problems of arbitrary scale. Application developers do not have to use different programming languages, what is profitable regarding error-proneness and implementation effort. Essential design issues of an HLA toolbox for Fortran were discussed.

Further on, the application problem SIMBELFederation was introduced. The presented current implementation state is based on the approach of integrating Fortran routines into C++ as HLA-capable language environment. With it the proof of concept has been provided, it is possible to connect existing real-world Fortran codes with the HLA technology and the concrete realtime requirements of the application are not violated by the distributed processing approach. As a matter of fact, the correct and effective usage of the HLA concepts depends on deep knowledge in the field of distributed simulation. No interface technology can eliminate this prerequisite.

The discussed Fortran/HLA connectivity approaches are based directly on the HLA interface specification. More than 130 HLA services require a considerable amount of initial training. Therefore, a number of software packages exist defining an interface layer above the HLA service set for complexity reduction. Typical representatives are VR-Link from MÄK technologies (see 2.4, and pSISA [3]), developed by the German Armed Forces (WTD 81). Similar to RTI implementations, C++ language bindings are provided. Therefore, the integration techniques examined in this article can also be applied to realize Fortran connectivity to these packages.

References

[1] *HLA Toolbox - The MATLAB interface to HLA*. Brochure, ForwardSim, Inc., Sainte-Foy, QC, 2005.

[2] K. J. de Kraker, J. Duncan, E.-W. Budde, R. Reading, R.: *NATO Standards for Virtual Ships*. Procs. Fall 2005 Simulation Interoperability Workshop, Paper 05F-SIW-020, Orlando, FL, 2005.

[3] U. Krosta, H.-P. Menzler, K. Pixius: *Implementierung von HLA-Schnittstellen mittels pSISA*. HLA Forum, Magdeburg, 2001.

[4] *MÄK HLA/DIS Toolbox for MATLAB and Simulink*. Broch., MÄK Technologies, Cambridge, MA, 2005.

[5] *NATO STANAG for Virtual Ships*, Study Draft v0.10, Military Agency for Standardisation.

[6] S. Pawletta, W. Drewelow, T. Pawletta: *On the Integration of HLA into SCEs*. TRANSACTIONS of SCS, Vol. 18, No. 2 (2001), pp. 92-97.

[7] S. Pawletta, B. Lampe, T. Pawletta, W. Drewelow: *Eine HLA-Toolbox für Matlab*. In Proc. Simulation und Visualisierung, Magdeburg 2000, SCS Int., pp. 31-44.

[8] S. Pawletta, T. Pawletta, W. Drewelow: *HLA-based Simulation within an Interactive Engineering Environment*. In Proc. 4th IEEE Workshop on Distributed Simulation and Real-Time Applications, San Francisco 2000, pp. 97-102.

[9] H. Schütz, H.: *Wohin steuert die maritime Rüstung in Deutschland?* In Marineforum 04/2003.

[10] S. Straßburger: *Distributed Simulation Based on the High Level Architecture in Civilian Application Domains*. Advances in Simulation, SCS-Europe BVBA Ghent, Belgium, 2001.

Corresponding author: Christian Stenzel
Christian Stenzel, Sven Pawletta, Group Computational Engineering and Automation, Wismar University, PF 1210, 23952 Wismar, Germany; WWW.MB.HS-WISMAR.DE/cea; *christianstenzel@gmx.net*
Richard Ems, Petra Bünning, MTG Marinetechnik GmbH Wandsbeker Königsstraße 62, 22041 Hamburg, Germany {*Richard.Ems; Petra.Buenning*}*@mtg-marinetechnik.de*

Received: June 17, 2006
Revised: July 7, 2006
Accepted: July 20, 2006

Co-simulation of Matlab/Simulink with AMS Designer in System-on-Chip Design

U. Eichler, U. Knöchel, S. Altmann, Fraunhofer Institute for Integrated Circuits, Dresden, Germany
{*eichler, knoechel, altmann*}*@eas.iis.fraunhofer.de*
W. Hartong, J. Hartung, Cadence Design Systems GmbH, Feldkirchen, Germany
{*hartong, juergenh*}*@cadence.com*

With increasing complexity of Systems-on-Chips (SoC), system level design and simulation is a necessity. In an ideal top-down design flow, the system level model is used as executable specification for the block implementation, which is supported by mixed-signal simulators. This contribution describes a link between the system-level simulator MATLAB/Simulink and mixed-signal simulation in Virtuoso AMS Designer by a socket based co-simulation. The implementation of the co-simulation is described in detail, including user interface, protocol, synchronization and cross-platform support. The application of the co-simulation is illustrated by a wireless LAN system. While the RF subsystem of the WLAN receiver is modeled in Virtuoso AMS Designer, Simulink provides standard compliant testbenches and adequate visualization tools. The presented simulator coupling, as a special case of distributed simulation, provides a functional parallelization of the involved tools.

Introduction

Today's integrated circuits often contain analog and digital signal processing as well as microcontrollers and memory. These complex *Systems-on-Chips* (SoCs) provide high functionality and enable the design of smart products for a mass market. Chip designs have to be verified well before the expensive manufacturing of the first prototype in silicon is started. Simulation is the key element in chip verification.
The design flow is usually divided into different abstraction levels as shown in Figure 1. At system level functionality and performance of the whole system are specified and evaluated by system-level simulations. Afterwards, the system is partitioned into hard- and software, analog and digital parts - possibly in several steps. At circuit level these blocks are implemented as analog or mixed-signal circuits or gate netlists. Finally, a layout is designed. After each step the description of the design is more detailed than before.

In modern design flows, simulation support is available for all design levels and system parts. A wide range of tools offer dedicated solutions for specific design problems. Each tool is optimized for a specific level of abstraction and application area. Even though there is a certain range of overlap between the tools, difficulties arise when effects have to be analyzed that span different design levels. While the interfaces between block level and transistor circuit are well established by mixed-signal simulators, the link toward system level is still weak in most environments, although there are approaches e.g. for using simulator coupling in digital design flows ([5]).

MATLAB and its simulation toolbox Simulink are used in system design for many applications including communication, automotive and control systems. The visualization of results is well supported by comprehensive functions for signal analysis and monitoring.

In an ideal top-down design flow the system level model will be used as executable specification for the detailed block implementation using mixed-signal languages like VHDL-AMS or Verilog-AMS. In a following step the implementation proceeds down towards circuit level. Since most of these designs consist of analog and digital parts, their implementation is usually supported by mixed-signal simulators like Virtuoso AMS Designer or ADVance MS. Thus, a very accurate analysis of analog and mixed-signal circuits is provided. On the other hand this accuracy naturally reduces the simulation speed, so that it is often impractical to verify the whole system behavior on a very detailed level of abstraction.

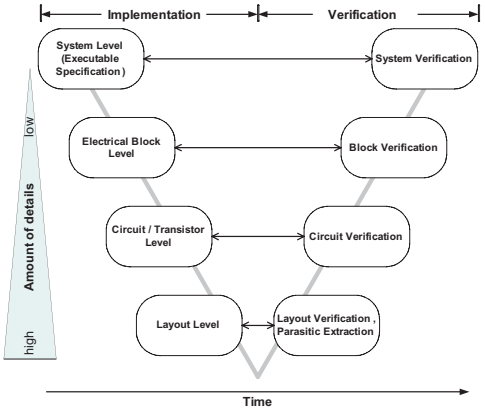


Figure 1: Design levels.

A verification of the implemented blocks against system level is difficult, due to the missing link. On the other hand, individually testing the designed circuits often does not ensure a working system, and designs fail due to problems at the interfaces. A direct link to the system environment can help to reduce interface problems and increase design efficiency and quality.
This was the main motivation for the development of the AMS Designer - Simulink co-simulation feature. Designers of analog and mixed-signal systems can evaluate their designs within a system model that can be reused from system design. The powerful Simulink model libraries simplify the design of module testbenches. System designers may include block or circuit level models of critical analog modules in the system simulation to analyze the performance impact and to adjust analog and digital parts, e.g. by digital predistortion of signals to compensate the non-linearity of a succeeding analog amplifier. Some of the existing solutions for multi-level simulation have been evaluated and improved within the project DETAILS ([*]). It is focused on an integrated simulation flow from system-level to mixed-signal and RF circuit implementation.

1 Concept and Implementation

When linking two simulators for co-simulation three main aspects of implementation have to be considered: the coupling of the different simulation algorithms, the choice of an appropriate user interface that integrates well in the simulators' handling concepts, and how both simulators should communicate. In the presented simulator coupling the mixed-signal simulator *Virtuoso AMS Designer* is used for the block and circuit level simulation. MATLAB/Simulink acts as sys-tem level simulator. Because both tools calculate the time-dependent behavior of the analyzed model (time-domain simulation), the coupling algorithm is focused on synchronization which is discussed in Section *Synchronization* below.
The simulator coupling user interface has to provide a simple way for the user to define the border between both model parts residing in the two different simulation environments. That concerns the choice of the signals to be transferred, their data types and the sampling mode to be used. Special coupler modules have been introduced for both simulation environments. The coupler module represents the model part that resides in the other simulator. Signals that should be transferred to or from the other model part are connected to the input or output ports of the coupler (see Figure 2). This approach allows to keep the modular structure of a model when splitting it for co-simulation.

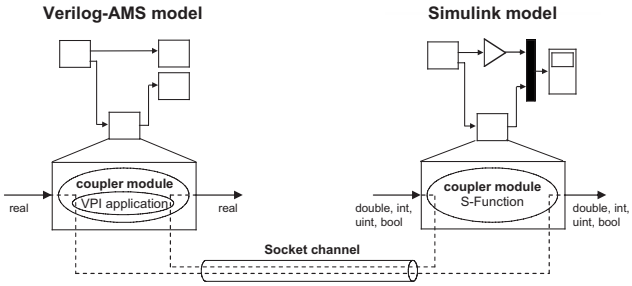


Figure 2: Co-simulation principle.

The coupler modules can be easily inserted and parameterized using the well-known graphical user interfaces of both simulation environments.

For the communication between the coupled simulators a TCP/IP network socket connection is used, allowing the co-simulation to be run on a single machine as well as on different hosts in a network. Different operating systems and platforms are supported in one co-simulation run (cross-platform simulation, see also Section *Platform Support* below). When running on different machines, the co-simulation may profit from better memory utilization because more memory is available per simulator.

1.1 Virtuoso Coupler Module

For AMS Designer the actual coupler module is written in *Verilog-AMS*. It provides port definitions, contribution statements for port access, and intermediate variables of type *real* for each input and output port. These variables are read and written by the VPI application - a dynamically loaded library which is written in C and contains the main coupling functionality. VPI, the *Verilog Procedural Interface*, provides several functions to interact with the simulation engine and to access Verilog objects like variables, modules and ports. The VPI application is started by a user-defined system task called at initialization of the coupler module:

```
initial $couple_init(CouplerToSimulink,hostName);
```

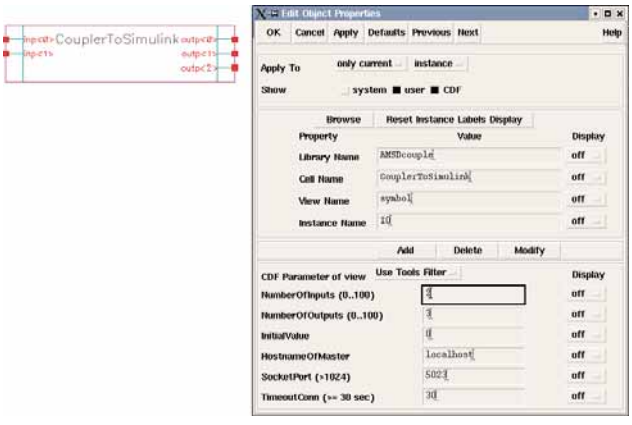


Figure 3: Virtuoso AMS Designer coupler module.

In each step, the simulation data received from Simulink is written to the Verilog module's output variables, and the input variables are read. Inside the Verilog coupler module these variables has to be mapped to the module ports. Currently, there are two coupler modules implemented. The first one is a pure digital module and maps data directly to its ports of type *wreal*. The second one is shown in Figure 3 and has analog electrical ports. It uses an interpolation algorithm to write the received data to its output ports.

Symbol and parameter dialog of the Verilog-AMS coupler module in Virtuoso AMS Designer are shown in Figure 3. The following parameters are provided:

- *NumberOfInputs, NumberOfOutputs*: The number of module ports per direction. The symbol view is changed according to these settings.
- *InitialValue:* The starting value for the inter polation algorithm. It is provided at the coupler module's output ports at simulation time t_0 (Figure 5, 6).
- *HostnameOfMaster:*The name of the local or remote host where Simulink is started.
- *SocketPort:*The TCP/IP port number for the socket connection. Both simulators must use the same setting in order to communicate.
- *TimeoutConn:* Time before terminating simulation if no data is received from the other simulator.

1.2 Simulink Coupler Module

The C-based s-function API was used to implement the coupler module on Simulink side. The advantage is that common functions for protocol and socket access could be shared by VPI application and s-function. The S-Function code is compiled to a shared library and contains the entire functionality of the coupler except the parameter dialog which was created using the *Simulink Mask Editor* (see Figure 4). The coupler module can be executed either at a possibly variable sample rate, inherited from the connected blocks, or at a constant, user-defined sample rate.

With the following three parameters the sampling mode can be controlled:

- *FrameMode*:Toggles between framed and unframed synchronization (see below).
- *FrameSize*: The number of samples per frame. If the frame size should be inherited from the connected blocks, this value is set to -1.

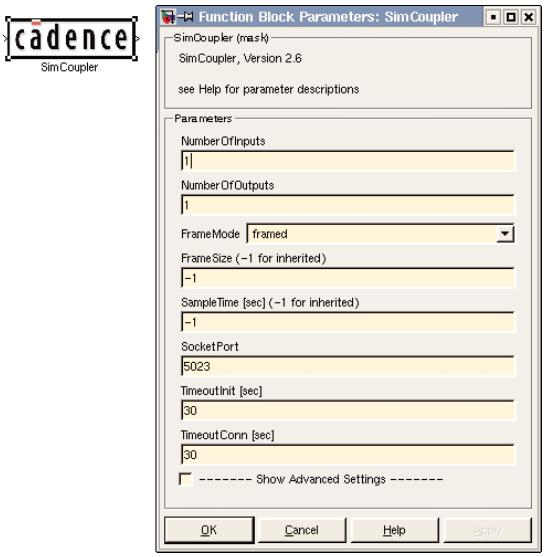


Figure 4: Simulink coupler module.

- *SampleTime*: This value defines a fixed sampling or frame period for the coupler block. If set to -1, sample time is inherited from the connected blocks.

1.3 Synchronization

When splitting a simulation task into several parts for co-simulation, the synchronization of the involved simulation tools is an important issue. This comprises the choice of an appropriate synchronization algorithm as well as its implementation (for overview, see [4]).
A synchronization algorithm should strongly depend on the scheduling schemes used by the coupled simulators. In our case these are a dataflow-like scheduling algorithm with fixed or variable time steps for Simulink and a combination of discrete-event control with a continuous-time analog solver for the mixed-signal simulator AMS Designer. Here, the set of applicable synchronization schemes is mostly determined by Simulink, because in dataflow simulation a block is not executed until all of its inputs are calculated by their drivers. This results in a fixed execution order of all blocks that is repeated for each sample period. Thus, also the Verilog-AMS model represented by the coupler block has to be executed accordingly to this order and has to calculate its outputsfor that sample period. The synchronization time points are given by Simulink. This scheme is a conservative synchronization approach ([3], [1]).
With these preconditions an implementation using the master-slave principle was the most preferable one due to its simplicity and its only small communica-tion overhead. Consequently, Simulink was chosen to take the master role for synchronization and connection setup.

The simulators exchange data frames with blocking access to the socket connection. Simulation time advance is controlled by the master simulator and is always positive for both simulators. That means, Simulink calculates the coupler module's input data for the current time step, sends this data together with the time of the next sampling point to the Verilog coupler. AMS Designer advances simulation until this time and sends back its output data to Simulink.

Inside the Verilog-AMS coupler module the discrete-timed data received from Simulink has to be mapped to the continuous time axis of the analog part. This is done by an interpolation algorithm that calculates additional values between the received samples if requested by the analog solver. The interpolation is done linearly starting at simulation time t_0 with the value of the parameter *InitialValue*. In the opposite direction the input signals of the Verilog coupler are sampled at the times given by Simulink. This can influence the co-simulation performance significantly. Thus, the sampling rate should be chosen carefully. If it is too low, signal changes with smaller time constants are lost. If the sampling rate is too high, simulation performance decreases.

With its signal processing blockset Simulink provides a special signal type - so-called frame-based signals. These compose several successive samples to a single frame and transmit them all at once. Frame-based signals can help modeling multi-rate systems and increase simulation performance significantly due to the reduced communication effort between connected blocks ([6]). This feature had to be considered also for the proposed co-simulation. Supporting Simulink's different sampling modes - from variable sample rates to frame-based signal processing - was a challenge for the co-simulation implementation.

For more flexibility two different synchronization algorithms for framed and unframed data are used. In unframed mode (Figure 5) the Simulink coupler module does not exchange data at simulation time t_0.

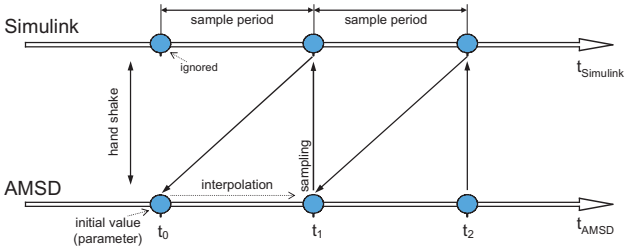


Figure 5: Sample-based synchronization.

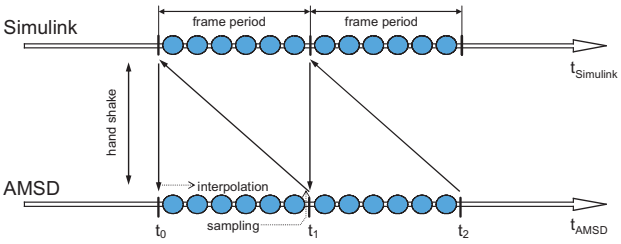


Figure 6: Frame-based synchronization.

The first input sample of the coupler module is ignored and a value of 0 is written to the outputs. The input sample of the second sampling period is then sent to AMS Designer together with the current simulation time t_1. AMS Designer advances simulation until t_1, samples the coupler module's input signals and sends these values back to Simulink. Due to the interpolation algorithm in the Verilog coupler module, the signal values received at time t_0 are not achieved until t_1 is reached. Thus, the introduced delay of one sample period is compensated, and time axes of both simulators are absolutely synchronous. Because the Simulink coupler module only needs to know the current simulation time, the synchronization scheme for the unframed mode allows using the coupler also in models with variable sample time.

In framed mode whole data frames are exchanged between the simulators at the equidistant frame sample points. It is important to note, that Simulink always generates frames for the following frame period. Thus, the first frame is sent at simulation time t_0 from Simulink to AMS Designer for the interval from t_0 to t_1 (Figure 6). The Simulink coupler module has to know the next synchronization point t_{n+1} when sending a data frame. This is only possible with fixed sampling rates as used in framed mode. Within the VPI application of the Verilog-AMS coupler the incoming data frame is split into the original data points. The Verilog simulation works subsequently on this input data. The generated output data is again collected into a frame and sent back to Simulink once the frame is complete. At sample level AMS Designer shows a delay of one sample period compared to Simulink resulting from the linear interpolation between the sample points (see above). In both, framed and unframed mode, there appears no delay between the input and output ports of the Simulink coupler module.

For the analog co-simulation, the use of the two different schemes provides the most suitable synchronization in both, framed and unframed mode. The unframed synchronization also allows variable sample times in Simulink. The occurring delays are negligible.

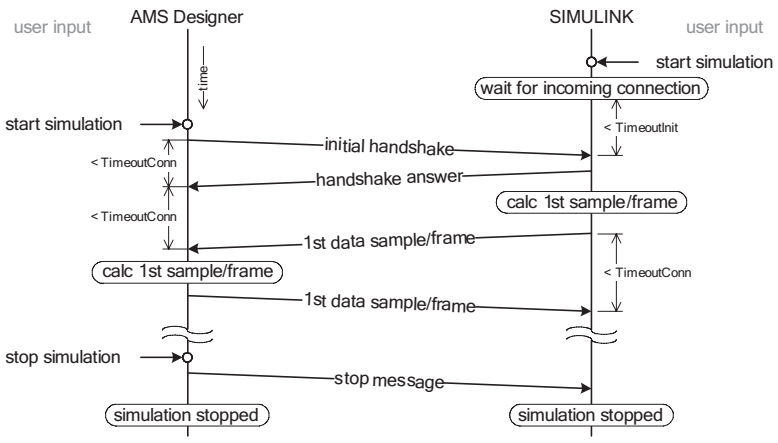


Figure 7: Protocol message flow.

1.4 Protocol

The implemented master-slave coupling principle has only few requirements concerning the underlying communication protocol. There are initial handshake messages exchanged at co-simulation setup and data messages containing simulation data (see Figure 7). When starting a co-simulation, the master simulator acts as server waiting for client requests. After the client simulator has sent an initial handshake message, the master sends a handshake reply containing information on its *Endian byte format* (see below) and the number, data types and dimensions of its coupler module's ports.

If the client simulator accepts the received settings, simulation starts with the first data frame from the master simulator. The data messages contain a flag describing the simulation status. It is mainly used to signalize errors or the end of simulation. The maximum time a simulator waits for an incoming message can be set by a parameter of the coupler module.

1.5 Platform Support

The current implementation of the simulator coupling supports the Linux, Solaris, and Windows operating systems and the Sparc and x86 processor architectures. To enable cross-platform simulation, it was necessary to consider the different Endian formats of the target platforms. Data is stored in sequences of bytes assembled of eight bits. To store numbers like integers or doubles using more than eight bits, several consecutive bytes are used. Different processor architectures use different byte orders inside those multi-byte numbers.

There are two common formats: *Little Endian*, used by Intel processors and *Big Endian* used by Sparc or Motorola processors and for protocol data in TCP/IP networks.

The cross-platform support provides an automated detection of the Endian format on both machines. A conversion is done only in the case of different formats to minimize the simulation overhead.

For *Little Endian* the least significant byte is stored at the first position (the lowest address) and for *Big Endian* the most significant byte comes first. To perform a cross-platform co-simulation, the Endian format of the master machine must be detected and - if necessary - transferred data must be converted when sent/received from the other simulator. In the current implementation the initial handshake messages contain a flag that indicates the Endian format of the simulator and are sent always in Big Endian format. The subsequent data messages are sent in the Endian format of the master simulator. That means, during simulation only the client simulator converts data if the Endian formats of client and master are different.

Endian format detection is done by casting the first byte of a long integer variable with value 1 to a one-byte character and checking whether it's value is 0 (Big Endian) or 1 (Little Endian). The data conversion simply re-orders the bytes of each double or integer in the reverse order.

2 Application Example: Wireless LAN Transceiver

In this section the application of the co-simulation for modeling a wireless LAN IEEE 802.11b physical layer transmission system on different levels of abstraction is shown.

Figure 8 shows the Simulink top-level schematic of a wireless LAN system level model. Binary random data is encoded and modulated in the transmitter part. The OFDM signal is then transferred to a channel model.

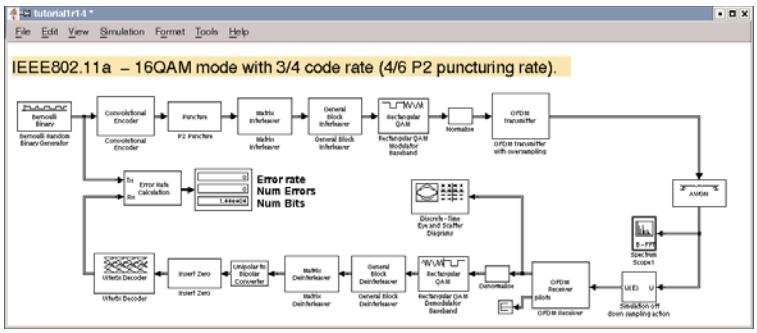


Figure 8: End-to-end system-level simulation with Simulink.

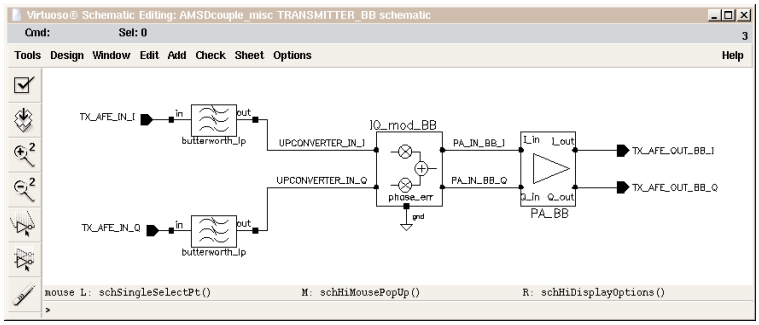


Figure 9: Behavioral model of the transmitter RF frontend.

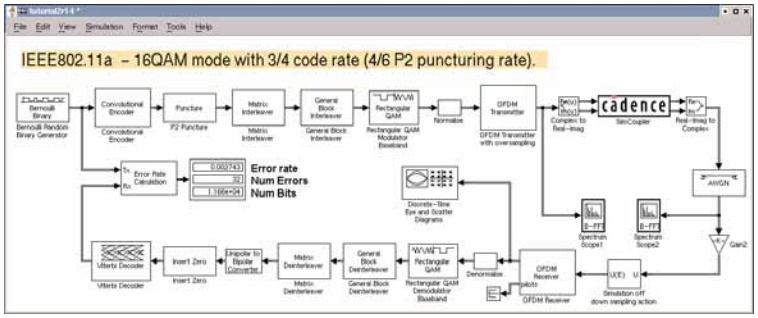


Figure 10: Simulink model with modules for co-simulation.

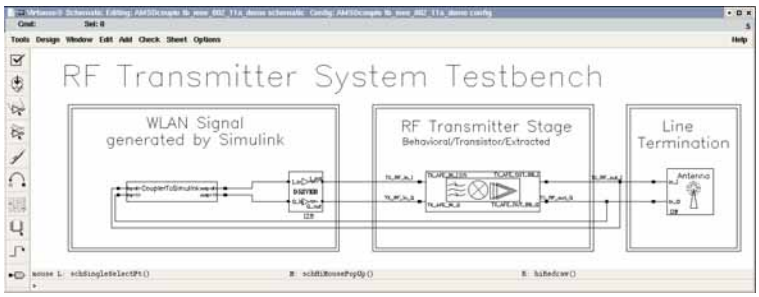


Figure 11: AMSD testbench with couple module.

In the example a White Gaussian Noise channel is used. The receiver blocks demodulate and decode the channel output. Finally the received bits are compared with the original bit stream to compute the bit error rate.

This standard compliant model of the wireless LAN link is built with modules from the Simulink communication and signal-processing toolboxes. The sample model contains only the digital parts of the transmission system. Effects originating from the analog RF parts of transmitter and receiver are not considered in the current simulation.

The RF frontend was designed using SpectreRF and AMS Designer within the Cadence Virtuoso environment. Figure 9 depicts the behavioral model of the RF transmitter module, which filters, up converts and amplifies the signal.

For simulation speed-up, the RF parts are modeled in complex baseband domain as Verilog-A behavioral models. It is possible to switch the abstraction level as far down as transistor level. However, the simulation performance will be lower in this case.

One- and two-tone sources are typically used in this environment as stimuli for the analysis of the RF sub-systems. Characteristics of the design are for example intercept points, noise figures and corner frequencies.

In most cases, it is much easier to handle more realistic stimuli, like modulated signals and corresponding DSP post-processing blocks for performance evaluation, on system level using Simulink.

With the co-simulation those tests are set up easily without modifying the environment setup too much.

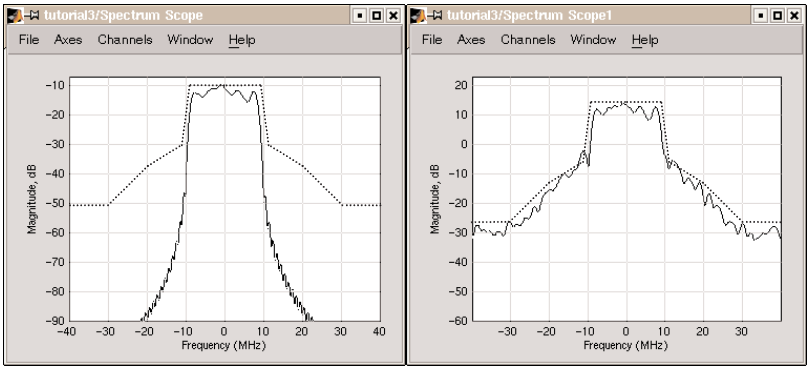


Figure 12: Co-simulation results, spectral masks.

A coupler module is used to link the RF transmitter model into the Simulink system-level schematic (Figure 10). In AMS Designer the RF frontend model is embedded in a separate testbench containing the corresponding coupler module and a simple antenna model (Figure 11). The input of this coupler module is sent through the socket connection to the output of the Simulink coupler module and is therefore connected to the output of the transmitter model.

Figure 12 depicts the frequency characteristic of the transmitted OFDM signal and the spectral mask (dotted) for IEEE 802.11a. The transmitted signal must be within this mask to fulfill the specification. The left-hand plot shows the signal generated by the digital baseband in Simulink. After passing the RF frontend some deviations can be observed, caused by nonlinear behavior (right-hand plot). The co-simulation can now be used to improve the system model by optimizing the parameters of RF front-end and DSP part.

3 Summary and Outlook

The presented simulator coupling enables the co-simulation of MATLAB/Simulink and the mixed-signal simulator Virtuoso AMS Designer. Its main advantage comprises the possibility to integrate design verification steps into system level simulation by increasing simulation accuracy of selected parts of a model - if necessary down to circuit level. Here, the general tradeoff between simulation accuracy and performance has to be taken into account. This application scenario was demonstrated by a WLAN transceiver model.
Furthermore, the coupling allows to use special features of one simulator in a co-simulation, e.g. Simulink blocks for stimuli generation and postprocessing, AMS Designer for multi-language mixed-signal simulation. Cadence Design Systems is providing this coupling feature within the current software release. It has been successfully tested by several major design companies.

References

[1] U. Donath et al.: *Parallel Multi-Level Simulation with a Conservative Approach*. J. Systems Analysis - Modelling - Simulation 21(1995), pp. 187-201

[2] R. Frevert et al.: *Modeling and Simulation for RF System Design*. ISBN 0-387-27584-3, Dordrecht, Springer, 2005

[3] D. Kim, C.-E. Rhee, S. Ha: *Combined Data-Driven and Event-Driven Scheduling Technique for Fast Distributed Cosimulation*. IEEE Trans. on VLSI Systems, Vol. 10, No. 5, pp. 672-678, Oct. 2002

[4] P. Le Marrec et al.: *Hardware, Software and Mechanical Cosimulation for Automotive Applications*. Proc. 9th IEEE Int. Workshop on Rapid System Proto-typing, pp. 202-206, Leuven, June 1998

[5] S. Wielens, S. Altmann, J. Haufe, P. Schneider: *Integration of Prototypes into the Design Flow of Digital Hardware for Applications in Mechatronics and Telecommunication*. Proc. Model-Based Design Conf. 2005, pp. 55-60, Munich, June 2005

[6] *Simulink Signal Processing Blockset*, WWW.MATHWORKS.COM/products/sigprocblockset/

[7] *Cadence RF Design Methodology Kit*. WWW.CADENCE.COM/products/kits/RF_Design/

[*] The presented work was partly funded by the project DETAILS, promoted by the German BMBF (Sign 01M3071) within the initiative 'Mobile Internet'.

Corresponding author: U. Eichler
U. Eichler, U. Knöchel, S. Altmann
Fraunhofer Institute for Integrated Circuits (IIS),
Branch Lab Design Automation (EAS)
Zeunerstraße 38, 01069 Dresden, Germany
{eichler, knoechel, altmann}@eas.iis.fraunhofer.de
W. Hartong, J. Hartung, Cadence Design Systems GmbH
Mozartstrasse 2, 85622 Feldkirchen, Germany
{hartong, juergenh}@cadence.com

Received: May 2, 2006
Revised: June 8, 2006
Accepted: June 20, 2006

Parallel Computation in Blood Flow Simulation using the Lattice Boltzmann Method

Daniel Leitner, Siegfried Wassertheurer, ARC Seibersdorf research GmbH, Vienna, Austria
{*daniel.leitner, siegfried.wassertheurer*}*@arcsmed.a*
Felix Breitenecker, Vienna University of Technology, Vienna, Austria
Felix.Breitenecker@tuwien.ac.at
Michael Hessinger, Andreas Holzinger, Medical University Graz
{*michael.hessinger, andreas.holzinger*}*@meduni-graz.at*

Lattice Boltzmann Models (LBM) are widely used to solve fluid mechanical problems in engineering and biomedical applications. First a brief introduction of LBM is given and an example model with three spatial dimensions is introduced. The model is relevant for blood flow simulation because it uses Reynolds and Womersley numbers found in hemodynamics with a realistic time dependent pressure gradient as a boundary condition. A big advantage of LBM is the possibility of easy parallelization. Therefore different approaches of implementations are discussed. To test parallelization, the example model is used as a benchmark. The simulation times are compared calculating the problem in parallel on one to four processors.

SNE 16/2, September 2006

Introduction

In the western industrial countries cardiovascular diseases are the most frequent cause of death. Therefore a lot of research is done to get a better understanding of the *cardiovascular system* (CVS). To simulate the CVS, various models of different accuracy are used and often coupled together to describe the circulation on different spatial and temporal scales [1].

In this work a LBM is used to simulate the blood flow in three spatial dimensions, solving the Navier-Stokes equation with the Lattice Bhatnagar-Gross-Krook (LBGK) method ([2, 3]). The main advantages of the LBGK method are that it is simple to implement and to parallelize which enables an efficient computation. Furthermore it is a bottom up approach. Thus the algorithm can be interpreted physically in every step, which makes the method very intuitive.

The calculations in computer fluid dynamics (CFD) and specially blood flow simulation in three spatial dimensions are very time consuming. Adequate computer systems often make use of multiple CPUs. Therefore it is fundamental for algorithms in CFD to support parallelization. In this work the LBGK method in three dimensions with 15 degrees of freedoms is tested under these aspects.

The LBGK method is tested on a Dell Precision 670 machine containing two Intel Xeon dual core processors with 2.8 GHz.

The test case simulates unsteady flow in a rectangle. The time dependent pressure gradient, fluid viscosity and the resulting Reynolds numbers lie in a range relevant for blood flow simulation.

The results of the simulation are in best accordance with the analytical results obtained by Womersley, see [4]. The example shows the benefits of the LBGK method for blood flow simulation and thus acts as a relevant example for comparing computation times of multiple CPUs.

1 The LBGK D3Q15 Model for Blood Flow Simulation

For simulating the flow field we use a LBGK model [2, 3], which is proved to be capable of dealing with pulsative flow within the range of Reynolds and Womersley numbers existing in large arteries [5, 6].

The LBGK model is based on a statistical description of a fluid in terms of the Boltzmann equation. The Boltzmann equation with single relaxation time is given by

$$\frac{\partial f}{\partial t} + \xi \cdot \nabla f = -\frac{1}{\lambda}(f - f^{eq})$$

This equation is discretised in the spatial domain, in velocity space and in time, yielding

$$f_i(x + c\cdot \mathbf{e}_i\Delta t, t+t) - f_i(x,t) = -\frac{1}{\lambda}(f_i(x,t) - f_i^{eq}(x,t))$$

where $c = Dx/Dt$, Dx is the lattice grid spacing, and Dt the time step.

The particle distribution functions f_i evolve on a regular grid and represent particles travelling on the link $\mathbf{e}_i$ (Figure 1), thus $f_i(x,t)$ refers to the particle distribution on the lattice node x at time t on the link $\mathbf{e}_i$.
Note that $f_0(x,t)$ represents the particles resting at node x, thus $i = \{0,\ldots, 14\}$ in the D3Q15 LBGK method. In the name D3Q15, D3 is referring to the three spatial dimensions, Q15 to the 15 degrees of freedom in a node.

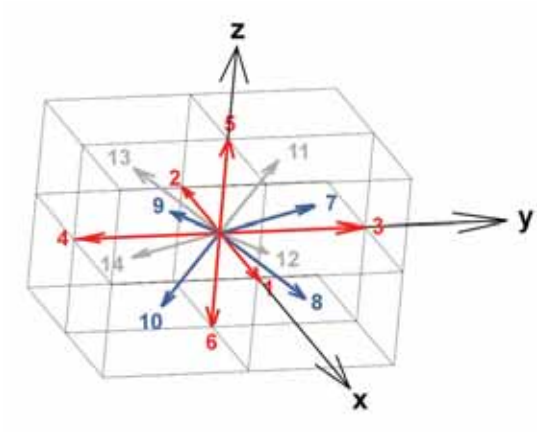


Figure 1: The velocity directions in the LBGK D3Q15 model

The equilibrium density distribution $f^{eq}(x,t)$ depends solely on the density $\rho(x,t)$ and the velocity $u(x,t)$ of a lattice node x. The density ρ and the velocity u are obtained from the density distribution function f_i:

$$\rho(x,t) = \sum_i f_i(x,t)$$
$$\rho(x,t)u(x,t) = \sum_i c\cdot \mathbf{e}_i f_i(x,t)$$

The equilibrium is defined as

$$f_i^{eq}(\rho,u) = \rho(\omega_i + 3\omega_i \mathbf{e}_i\cdot u + \frac{9}{2}\omega_i(\mathbf{e}_i\cdot u)^2 - \frac{3}{2}\omega_i u\cdot u)\cdot$$

with the weight coefficients

$$\omega_i = \begin{cases} 2/9, & i = 0 \\ 1/9, & i = 1,\ldots,6 \\ 1/72, & i = 7,\ldots,14 \end{cases}$$

The mass and momentum equations can be derived from the model via multiscale expansion as

$$\frac{\partial\rho}{\partial t} + \nabla\cdot(\rho u) = 0$$
$$\frac{\partial(\rho u)}{\partial t} + \nabla\cdot(\rho u u) = -\nabla p + v(\nabla^2(\rho u) + \nabla(\nabla\cdot(\rho u)))$$
$$p = c_s{}^2\rho, \quad c_s = \frac{c}{\sqrt{3}}, \quad v = (2\tau - 1)c^2\frac{Dt}{6}$$

where p is the pressure, c_s is the speed of sound, and v is the kinematic viscosity. The mass and momentum equations are exactly the same as the compressible Navier-Stokes equation, if the density variation is small enough.

Thus the compressible Navier-Stokes equation is recovered in the incompressible limit. If the density fluctuation is assumed to be negligible, the incompressible Navier-Stokes equation can derived directly via the Chapman-Enskog procedure. Because of the expansion in the velocity term the lattice Boltzmann method is only applicable to low Mach number hydrodynamics.

2 Implementation Notes for Parallel Computation

The strictly local nature of the LBGK method enables an easy parallelization of the algorithm.
The pseudo code for a one processor machine can be easily formulated, as can be seen in the following code snippet:

```
while(running) {
  for each node {  calc kinetic equ }
  for each node {  calc equilibrium }
}
```

Furthermore note, that the discretised Boltzmann equation (see before) is normally formulated as kinetic equation ($\omega = 1/\lambda$):

$$f_i(x + c \cdot \mathbf{e}_i \Delta t, t + \Delta t) = (1-\omega) f_i(x,t) - \omega f_i^{eq}(x,t)$$

To adjust the method for multiple CPUs the set of nodes must be simply distributed on the threads, each running on one processor. In each calculated time step the threads must wait for each other two times:

```
while(running) {
  for each thread { calc kinetic
  equation for all nodes }
wait for all threads
  for each thread { calc equilibrium
  for all nodes }
wait for all threads
```

The way the nodes are distributed on the threads is important. Neighbouring nodes should be processed from one thread for optimal cache usage. Special care must be taken when there are complex boundary nodes which need more computation time. The nodes should be distributed in a way that every thread needs exactly the same time to calculate its nodes.

For the computation a data structure is needed to store the densities, equilibria and information about neighborhood. Basically there are three approaches for the representation of the states: lattice, list, and object.

Lattice Implementation

The simplest approach is to store all states in a three - dimensional lattice and a three-dimensional array, respectively.
Two lattices are needed. In one time step the new values are calculated from an input lattice A into a lattice B, in the next step from B to A and so on, see Figure 2. There is no need to store extra information about neighborhood because the position of the data in the lattice and therefore the position in memory is known.

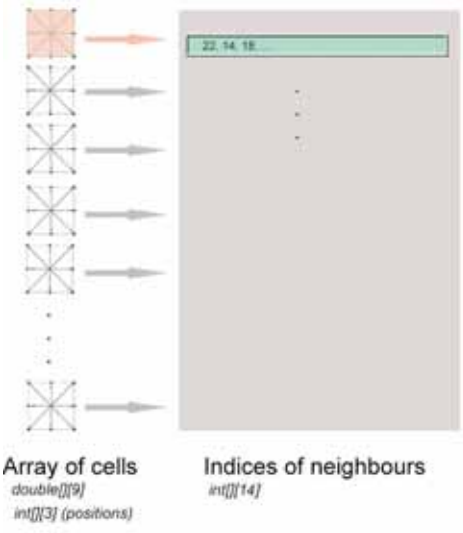


Figure 3: Implementation with a list.

List Implementation

A major drawback of the method with lattices is when small structures lie in a big volume, like arteries in tissue. Most of the nodes are boundary nodes and there is only a small percentage of fluid nodes.
The idea is to calculate and store only relevant nodes, which are fluid nodes and no slip nodes neighbouring them. The data of these relevant nodes are written into a list. Two sets of states are stored for every node, old states and new states. In addition extra information about neighbourhood is needed, see Figure 3. Thus the indices of the neighbours must be stored in a table for each node in the list.

Further the positions of the nodes in space are needed, therefore three additional values must be stored for every node. Note that in the implementation with lattices this information is provided in a natural way.

Implementation with Objects

A more intuitive but slightly slower and more storage demanding approach is to represent every node as an object. The states and positions in space are stored within the object. Neighbourhood is realized by storing the references to the neighbours, see Figure 4.

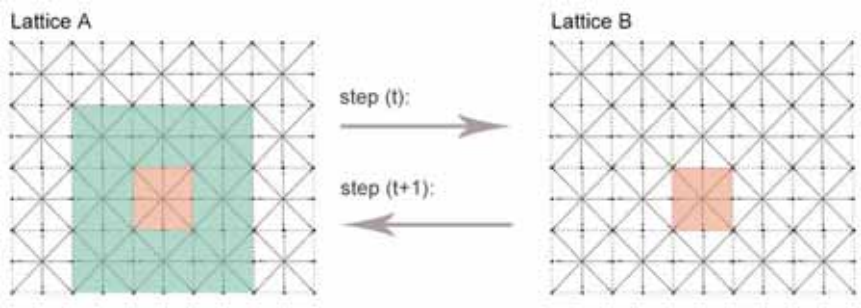


Figure 2: Implementation with two lattices.

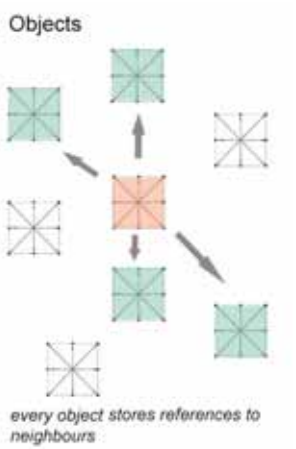


Figure 4: Object oriented implementation.

This needs more memory on 64 Bit machines because a reference needs 64 Bits while an integer used for an index of a list needs only 32 Bits of memory.

The object oriented approach has been used in this work, because it simplifies experiments with new node types.

3 Simulation Results

The example models are boxes of 20*20 nodes with a length of 20, 100, 200 nodes. The boxes are surrounded with no-slip nodes describing the walls. At the top and bottom of the box a special boundary condition is applied for describing the time dependent pressure gradient. For more information about boundary conditions for LBGK methods the reader may refer to [2]. An approach to model elastic walls of arteries is given in [7].
The results of the simulation are presented in Figure 5. They are in best accordance with analytic solutions presented by Womersley.

The simulation is done on a Dell Precision 670 containing two Intel Xeon dual core processors with 2.8 GHz with Windows XP SP2 and J2SE 5 Update 7.

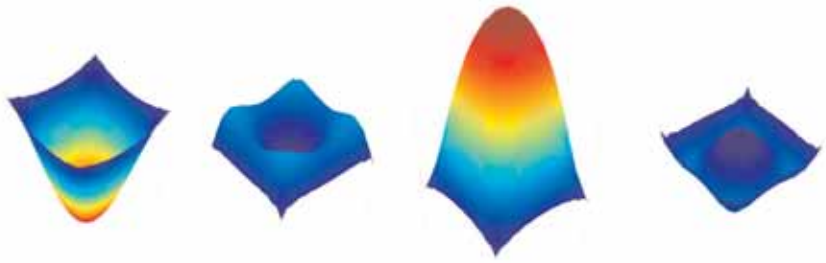


Figure 5: Velocity profiles of unsteady flow.

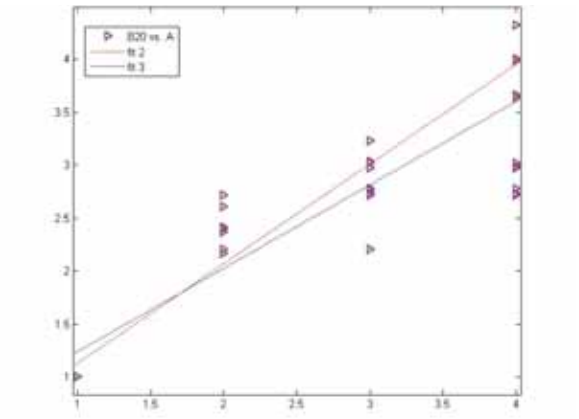


Figure 6: Simulation times of a 20*20*20 box (8000 nodes).

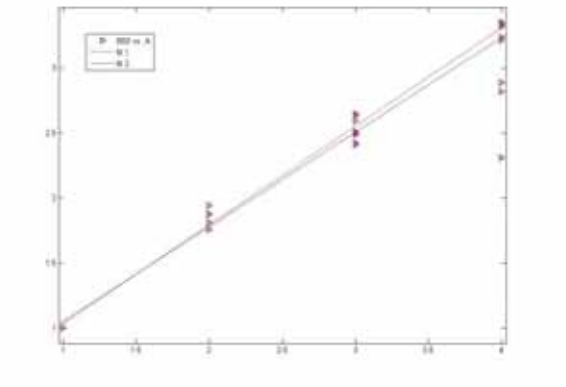


Figure 7: Simulation times of a 20*20*50 box (20000) nodes.

Time is measured over 100 time steps. The simulation is done 20 times on one to four processors. The run times are compared to the calculation time of one processor. Thus ideally two processors should work exactly with twice the perfomance as one.
The blue line in Figures 6-8 is the regression line of all sample points, the red line is the regression line of all points except the three worst results.

Simulation of the Box with 20*20*20 nodes: The smallest experiment with only 8000 nodes works very well and scales nearly linear. Eye catching is that two CPUs work have more than twice the performance than one CPU (see Figure 6). An explanation for this is the architecture of the computer. When two threads are used they run on the cells of only one CPU, while the second CPU can serve the operation system.

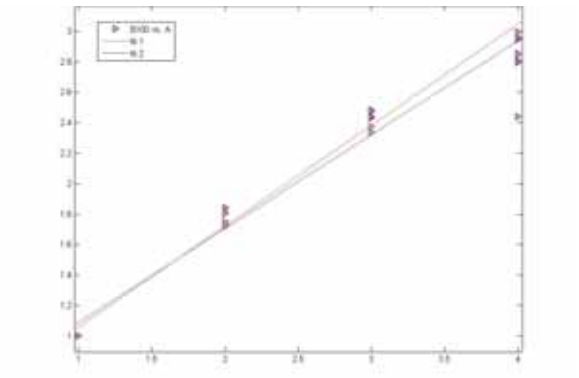


Figure 8: Simulation times of a 20*20*100 box (40000) nodes.

The threads on the two cells can use the cache in an optimal way. In this small example four threads are most of the times really four times faster than one thread.

Simulation of the Box with 20*20*50 nodes: The slightly bigger example with 20000 nodes shows a different behaviour. Because the simulation time is longer, more influences from the operation system affect the simulation time. Further performance is lost because the cache usage is not as good as in the smaller model. This results in slower simulation times than in the first example. Four threads are only three times faster than one thread (see Figure 7).

Simulation of the Box with 20*20*100 nodes: The biggest example with 40000 nodes shows similar behaviour than the model with 20000 nodes. Four threads have nearly the same performance as three threads (see Figure 8). The reason for this behaviour is that system services take a lot of time from one processor and therefore all other threads have to wait.

4 Summary

The LBGK D3Q15 method is described, which is a widely used method for fluid mechanical application. This work mainly focuses on the application of this method to hemodynamics and parallelization. A relevant example for blood flow simulation is described and the LBGK D3Q15 method is used for the calculation of the problem.
The model is used as a benchmark to test the ability of parallelization of the method. Different approaches of implantation are discussed. The object oriented approach is favoured.
For the computation a Dell Precision 670 workstation containing two Intel Xeon dual core processors with 2.8 GHz is used. The method scales linearly as expected for small models (8000 nodes).
For larger models (20000 or 40000 nodes) the simulation time is strongly influenced by services of the underlying operation system when working with four threads.

References

[1] D. Leitner, J. Kropf, S. Wassertheurer, F. Breitenecker: *Lattice Boltzmann Methode zur Simulation vom Strömungsverhalten in Arterien*. In (U. Rüde et al., eds.) Proc.18th Symp.Simulationtechnique ASIM 2005, Frontiers in Simulation, SCS Publishing House, Erlangen, 2005; pp 768-774.

[2] S. Succi: *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.

[3] D.A. Wolf-Gladrow: *Lattice-Gas Cellular Automata and Lattice Boltzmann Models - An In-troduction*. Lecture Notes in Mathematics. Springer, 2002.

[4] D.A. McDonald: *Blood Flow in Arteries*. Edward Arnold, 1974.

[5] A. M. Artoli, D. Kandhai, H. C. Hoefsloot, A. G. Hoekstra, P. M. A. Sloot: *Lattice bgk simulations of flow in a symmetric bifurcation*. Future Gener. Comput. Syst., 20(6):909-916, 2004.

[6] A.M. Artoli, A.G. Hoeksta, 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.

[7] D. Leitner, S. Wasssertheurer, M. Hessinger, A. Holzinger: *A Lattice Boltzmann Model for pulsative blood flow in elastic vessels*. Elektronik und Informationstechnik, heft 4, 152-155, 2006.

Corresponding author: Daniel Leitner
Daniel Leitner, Siegfried Wassertheurer,
ARC Seibersdorf research GmbH, Biomedical Engineering,
Floragasse 7, 1040 Vienna, Austria;
{daniel.leitner, siegfried.wassertheurer}@arcsmed.at

Felix Breitenecker, Vienna University of Technology,
Inst. for Analysis and Scientific Computing,
Wiedner Hauptstr. 8, 1040 Vienna, Austria;
fbreiten@osiris.tuwien.ac.at

Michael Hessinger, Andreas Holzinger, Medical University Graz, Auenbruggerplatz 2/4, 8036 Graz, Austria
{michael.hessinger, andreas.holzinger}@meduni-graz.at

Received: May 15, 2006
Revised: July 10, 2006
Accepted: July 20, 2006

ARGESIM Benchmark on Parallel and Distributed Simulation

Felix Breitenecker, Gerhard Höfinger,Vienna University of Technology, Austria
{*Felix.Breitenecker, Gerhard.Hoefinger*}*@tuwien.ac.at*
René Fink, Sven Pawletta, Thorsten Pawletta, Wismar University, Germany
WWW.MB.HS-WISMAR.DE/cea

This new ARGESIM Benchmark addresses benefits of parallel and distributed computing in the area of continuous, discrete, and hybrid simulation and in related areas. The benchmark may be of interest for users of all types of parallel and distributed facilities. The spectrum may range from parallelisation strategies and strategies for distributing tasks, via general purpose programming languages to simulation languages, andfrom networks of workstations, via special parallel computers, to very high performance computers.

Introduction

In 1994, ARGESIM has set up the *ARGESIM Comparison on Parallel Simulation Techniques* (CP1). There, three test examples have been chosen to investigate the types of parallelisation techniques best suited to particular types of simulation tasks. The new *ARGESIM Benchmark on Parallel and Distributed Simulation* (CP2) extends the previous comparison, addressing not only simulation software and predefined given algorithms, but also allowing use of different algorithms for solving the tasks and comparing different strategies for parallelisation or distribution of the tasks.

1 Contribution to the Benchmark CP2

The *ARGESIM Benchmark on Parallel and Distributed Simulation* tests benefits of parallel and distributed simulation with three case studies:

- Monte Carlo - Study
- Lattice Boltzmann Simulation
-• PDE Solution

Participation at this benchmark requires:

- Documentation of the algorithms for solving the case studies (one or more algorithms)
- Documentation of the strategy for parallelising or distributing the case studies (one or more strategies)
- Serial solution of the case studies
- Parallel / distributed solutions of the case studies
- Determination and documentation of efficiency of parallelisation

In detail, a contribution to this benchmark should for each case study describe first the approach or the algorithms for calculating solutions, followed by information about the method of parallelisation or distribution of tasks and subtasks. It is highly appreciated, if more than one solution for a particular case study is given, either using different parallelisation strategies or strategies for distribution, or by using different hardware environments, or by using different algorithms for calculating solutions. In the following, results of the case studies should be presented, based on a comparison of a serial solution and the parallel / distributed simulation of each case study.

For quantitative comparison of serial solution and parallel or distributed solutions, performance should be assessed in terms of the relative speed-up factor, a numerical value found by dividing the time for serial solution by the time for the parallel solutions (speed-up factor $\boldsymbol{f}$). Measurements of time, whenever necessary, should be in terms of the total elapsed time for running the task. Furthermore, a rough indication should be provided for the (time) effort for implementing a parallel or distributed simulation (at best compared with implementation time for the serial solution).

Contributions to this benchmark will be published in the journal SNE – *Simulation News Europe*. Solutions sent in should not exceed three SNE pages and will be reviewed by the editorial board and by authors of the benchmark.

2 Case Study 1 – Monte Carlo Study

The first case study is a Monte Carlo study. In a damped dynamic mass – spring system the damping factor is randomly disturbed, and the mean of a sample of dynamic outputs is to be calculated.
The second order mass-spring system is described by the following ODE, where the damping factor d should be chosen as a random quantity uniformly distributed in [800, 1200]:

$$m\,\ddot{x}(t) + d\,\dot{x}(t) + k\,x(t) = 0$$

$$x(0) = 0,\ \dot{x}(0) = 0.1,\ k = 9000,\ m = 450$$

The task is to calculate a sample of M = 1000 results $x(t, d_i)$ of the motion (Figure 1 shows $x(t,1000)$) and to calculate the mean motion $x_{mean}(t)$ over the time interval [0, 2] with a resolution (stepsize) of 0.01 (n = 200 steps):

$$x_i(t_j) = x(t, d_i), \quad j = 0,\ldots n,\ i = 1,\ldots, m$$

$$x_{mean}(t) = \frac{1}{M}\sum_{k=1}^{M} x_i(t_j)$$

As the model is a linear one, the solution can be provided also analytically, not only by using an ODE solver:

$$x(t,d) = Ke^{-\alpha t}\sin(\omega t)$$

$$\alpha = \frac{d}{2m},\ \omega^2 = \frac{k}{m} - \alpha^2,\ K = \frac{\dot{x}(0)}{\omega}$$

While the ODE may be basis for a parallelisation of the varying damping factor, the analytical formula may be a basis for parallelisation of the 201 time instants, where a solution is to be calculated.

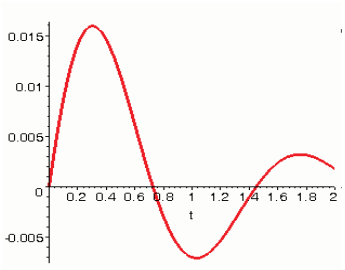


Figure 1: Plot of the analytical solution of the second order mass – spring system with d = 1000.

For documentation, we ask for a precise description of the parallelisation strategy used, and for comparison of the solutions we ask for a plot of the mean motion $x_{mean}(t)$ and of values for the speed-up factors $\boldsymbol{f}$.

3 Case Study 2 – Lattice Boltzmann Simulation

The second case study addresses the lattice Boltzmann method (LBM) for fluid flows, which is widespread in parallel simulation domains today. The method is derived from lattice gas cellular automata in which space, time, particle velocity and particle occupation state are all discrete. In LBM, particle occupation state on nodes is replaced by single-particle distribution functions (real values).

The case study is based on the famous cavity flow problem published by Hou et al in J. Comput. Phys. 118 (1995), where the behaviour of an incompressible fluid in a square enclosure, driven by a constant stream on the top boundary is examined (see Figure 2).

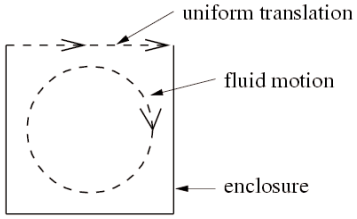


Figure 2: Lid-driven cavity flow.

For a description of the geometry matrix g, cell types are divided into wall cells (W), driving cells (D) and fluid cells (F). For a lattice size of 2x2, g is given at right.

$$g_{2,2} = \begin{pmatrix} D & D & D & D \\ W & F & F & W \\ W & F & F & W \\ W & W & W & W \end{pmatrix}$$

The uniform translation on top of the cavity is given as $u_{0x} = 0.1$, $u_{0y} = 0$, where the Reynolds number is Re = 1000. At any grid point, the initial macroscopic velocity is $u_x = 0$, $u_y = 0$ and the initial density is $\rho = 1$.

The task is, to simulate the cavity flow with lattice size 257 x 257 for a number of 350.000 iterations. After this number of iterations, steady state is reached. Simulation results are shown in Figure 3.

For documentation, we ask for a precise description of the parallelisation strategy used, and for comparison of the solutions we ask for a plot of relative macroscopic velocity magnitudes (u/u_0) at steady state and for values of the speed-up factors $\boldsymbol{f}$ (please note, that also a serial solution is necessary for this purpose).

A problem discussion in detail and links to sequential reference implementations as well as to introductory materials for the lattice Boltzmann method are provided at WWW.MB.HS-WISMAR.DE/cea/lbm.

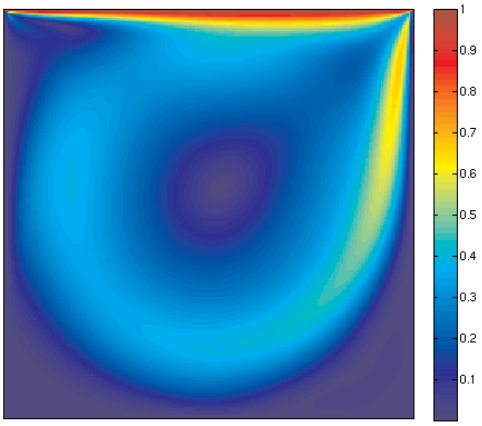


Figure 3: Relative macroscopic velocity magnitude (u/u_0) in cavity flow after 350000 iterations on a 257x257 grid.

4 Case Study 3 – Solution of a Partial Differential Equation

The third case study is based on a second order partial differential equation describing a swinging string with length L, fixed at both ends, excited at the beginning (surface plot shown in Figure 4):

$$u_{xx}(t,x) = \frac{1}{v^2}u_{tt}(t,x)$$

$$u(0,t) = u(L,t) = 0, \quad u_t(x,0) = 0$$
$$u(0 \le x \le \frac{L}{2},0) = 2\frac{h}{L}x,\ u(\frac{L}{2} \le x \le L,0) = 2h(1-\frac{1}{L}x)$$
$$v = 0.6, L = 0.5, h = 0.05$$

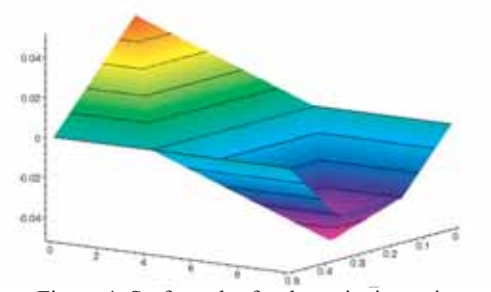


Figure 4: Surface plot for the swinging string – excitation in dependency of space and time.

One approach for solving this PDE is the method of lines, using discretisation of space. Discretising the space into N equidistant intervals and replacing the differential quotient $u_{xx}(t,x)$ by a central difference quotient, a set of weakly coupled ODEs replaces the PDE:

$$\frac{k^2}{v^2}\ddot{u}_i(t) = u_{i-1}(t) - 2u_i(t) + u_{i+1}(t), i = 1, N-1$$
$$u_i(0) = 2\frac{h}{N}i,\ i = 0,\ldots,\frac{N}{2},$$
$$u_i(0) = 2h(1-\frac{i}{N}),\ i = \frac{N}{2},\ldots,N, \quad \dot{u}_i(0) = 0$$

Also an analytical solution (approximation) can be calculated because of linearity. A classical separation approach $u(t,x) = X(x)\ T(t)$ can be used for calculating the solution. This yields with given initial and boundary conditions a solution with a Fourier series ([3]; Figure 5 and Figure 6 show lines in x and t, calculated with Fourier series cut at 100 summands):

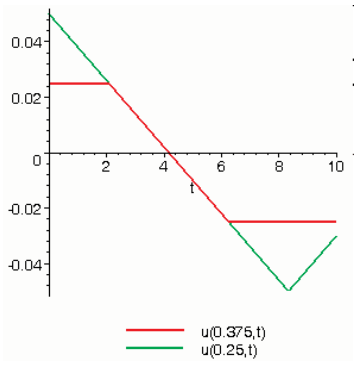


Figure 5: Solution of the PDE, excitation over time at $x = 0.375$ and $x = 0.25$.

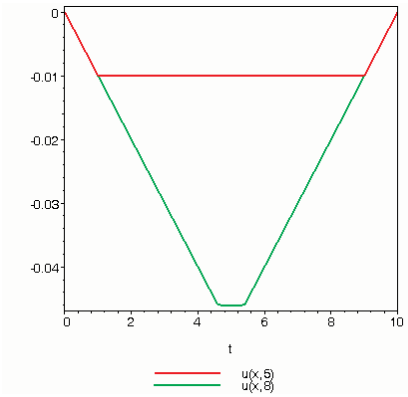


Figure 6: Solution of the PDE, excitation over space at $t = 5$ and $t = 8$.

$$u(x,t) = \frac{8h}{\pi^2}\sum_{j=0}^{\infty}\frac{(-1)^j}{(2j+1)^2}\sin\left(\frac{(2j+1)\pi x}{L}\right)\cos\left(\frac{(2j+1)\pi vt}{L}\right)$$

In principle, also discretisation of space and time may be suitable. For instance, using for space discretisation a central difference quotient as in method of lines, and using for time backwards difference quotients (as well for PDE and for initial condition) yields a linear system for $u(t_k, x_i)$, which may be parallelised for solution.

Of course, other algorithms for solving the PDE may be used, with varying grids etc, which can be parallelised / distributed appropriately.

In general, the system is to be solved with a spatial discretisation of $N = 500$ lines at the interval [0, 10] with time discretisation of 0.01s ($m = 1000$). For documentation, we ask for a precise description of the parallelisation strategy used, and for comparison of solutions we ask for plots of the lines $u(x=3L/4, t)$, $u(x=L/2, t)$ and $u(x, t=15)$, $u(x, t=30)$, and of a surface plot (excitation versus space and time). Furthermore, values for the speed-up factors ***f*** should be given.

References

[1] F. Breitenecker, I. Husinsky, G. Schuster: *Comparison of Parallel Simulation Techniques –Definition.* Simulation News Europe SNE 10, March 1994.

[2] S. Hou, Q. ou, S. Chen, G. D. Doolen, A.C. Cogley: *Simulation of Cavity Flow by the Lattice Boltzmann Method.* J. Comput. Phys. Vol. 118, no. 2, May 1995.

[3] Ch. B. Lang, N. Pucker: *Mathematische Methoden in der Physik*; Akad. Verlag, Spektrum HTB, Heidelberg, 1998 ISBN 3-8274-0225-5

Corresponding author: Felix Breitenecker
Felix Breitenecker, Gerhard Höfinger
Vienna University of Technology,
Inst. f. Analysis and Scientific Computing,
Wiedner Hauptstrasse 8-10, 1040 Vienna,
Austria; {*Felix.Breitenecker, gerhard.Hoefinger*}*@tuwien.ac.at*
René Fink, Sven Pawletta, Thorsten Pawletta, Res. Group Computational Engineering and Automation
Wismar University, Phillip Müller Straße,
23952 Wismar, Germany;
{*s.pawletta, r.fink*}*@et.hs-wismar.de*
pawel@mb.hs-wismar.de,

Received: June 15, 2006
Revised: August 30, 2006
Accepted: September 15, 2006

ASIM - Buchreihen / ASIM Book Series

REPORTS

Fortschritte in der Simulationstechnik (FS) / Series Frontiers in Simulation (FS) - Monographs, Proceedings:

W. Borutzky: *Bond Graphs Methodology for Modelling Multidisciplinary Dynamic Systems*. FS 14, ISBN 3-936150-33-8, 2005.

M. Becker, H. Szczerbicka (eds.): *19th Symposium Simulation Techniques*. Proceedings Tagung ASIM 2007, Hannover; FS 16, ISBN 3-936150-49-4, 2006.

S. Wenzel (Hrsg.): *12. Fachtagung Simulation in Produktion und Logistik*. Proceedings Tagung ASIM SPL 2006; ISBN 3-936150-48-6, 2006.

F. Hülsemann, M. Kowarschik; U. Rüde: *18th Symposium Simulation Techniques*. Proceedings Tagung ASIM 2005 Erlangen; FS 15, ISBN 3-936150-41-, 2005.

Available / Verfügbar: SCS Publishing House e.V., Erlangen, WWW.SCS-PUBLISHINGHOUSE.DE
Download ASIM Website WWW.ASIM-GI.ORG (partly; for ASIM members)

Fortschrittsberichte Simulation (FB) / Advances Simulation (AS) / ASIM Mitteilung (AM) ARGESIM Reports (AR) - Special Monographs, PhD Theses, Workshop Proceedings

C. Deatcu, S. Pawletta, T. Pawletta (eds.): *Modelling, Control and Simulation in Automotive and Process Automation*. Proceedings ASIM Workshop Wismar 2006, ARGESIM Report 31, AM 101; ISBN 3-901-608-31-1, 2006.

H. Ecker: *Suppression of Self-excited Vibrations in Mechanical Systems by Parametric Stiffness Excitation*. ARGESIM Report FB 11, ISBN 3-901-608-61-3, 2006.

M. Gyimesi: *Simulation Service Providing als Webservice zur Simulation Diskreter Prozesse*. ARGESIM Report FB 13, ISBN 3-901-608-63-X, 2006.

J. Wöckl: *Hybrider Modellbildungszugang für biologische Abwasserreinigungsprozesse*. ARGESIM Report FB 14, ISBN 3-901608-64-8, 2006.

Available / Verfügbar: ARGESIM/ASIM Publisher, TU Vienna, WWW.ARGESIM.ORG
Download / Bestellung zum Mitgliederpreis € 10.- ASIM Website WWW.ASIM-GI.ORG

REPORTS

Reihen der ASIM-Fachgruppen / Series of ASIM Working Groups

S. Collisi-Böhmer, O. Rose, K. Weiß, S. Wenzel (Hrsg.): *Qualitätskriterien für die Simulation in Produktion und Logistik*. AMB 102, Springer, Heidelberg, 2006; ISBN 3-540-35272-4.

M. Rabe, S. Spiekermann, S. Wenzel (Hrsg.): *Verifikation und Validierung für die Simulation in Produktion und Logistik*. AMB 103, Springer, Heidelberg, 2006; ISBN 3-540-35281-3.

A. Gnauck (Hrsg.): *Modellierung und Simulation von Ökosystemen - Workshop Kölpinsee 2004*. AMB 93, Shaker Verlag, Aachen, 2006; 3-8322-5203-7

Available / Verfügbar: Bookstore / Buchhandlung, ermäßigter Bezug für ASIM Mitglieder
Info at ASIM webite WWW.ASIM-GI.ORG

COMSOL MULTIPHYSICS™

Proceedings CD der Konferenz zur Multiphysik-Simulation

ANWENDUNGSBEREICHE:

- Akustik und Fluid-Struktur-Interaktion
- Brennstoffzellen
- Chemietechnologie und Biotechnologie
- COMSOL Multiphysics™ in der Lehre
- Elektromagnetische Wellen
- Geowissenschaften
- Grundlegende Analysen, Optimierung, numerische Methoden
- Halbleiter
- Mikrosystemtechnik
- Statische und quasi-statische Elektromagnetik
- Strömungssimulation
- Strukturmechanik
- Wärmetransport

Bestellen Sie hier Ihre kostenlose Proceedings CD mit Vorträgen, Präsentationen und Beispielmodellen zur Multiphysik-Simulation:

TITEL, NACHNAME _____

VORNAME _____

FIRMA / UNIVERSITÄT _____

ABTEILUNG _____

ADRESSE _____

PLZ, ORT _____

TELEFON, FAX _____

EMAIL _____

www.comsol.de

Fax: +49(0)551 / 99 721- 29
www.comsol.de/conference2005/cd/