TOWARDS A PROTOTYPE WEB-BASED COLLABORATIVE SIMULATION ENVIRONMENT

Andriy Levytskyy and Eugene J.H. Kerckhoffs

Delft University of Technology
Faculty of Information Technology and Systems
Department of Technical Informatics
Zuidplantsoen 4, 2628 BZ Delft, The Netherlands
E-mail: a.levytskyy@cs.tudelft.nl

Keywords

Collaborative environment, Internet, WWW, agents, discrete-event system simulation, distributed simulation.

Abstract

The aim of the research discussed in this paper is to develop and build a generic web-based distributed collaborative environment, allowing the registration of data files, simulation models and simulations in a central modelbase, browsing this modelbase, the meaningful presentation of registered data and models, remote execution of simulations, linking simulations through appropriate wrappers, a.s.o. The paper gives an overview of underlying ideas. Agent-like components are intended to be included to assist users in their collaborative work. As an illustrative (but tentative) example the use of trader and mediator agents are discussed to register data files and provide a meaningful (e.g., graphical) presentation of registered data.

1. Introduction

The purpose of the research discussed in this paper is to support simulations and simulation-related activities at the different levels (physics, circuitry, systems) of the so-called NanoComp project, that investigates the possibilities to achieve revolutionary better computing-system behaviour by optimally exploiting hardware based on quantum devices [DIOC-7]. In the subproject "Quantum Devices", physical phenomena are studied and experimental evaluation of SET devices is done. An appropriate fabrication process of such devices would be needed to test the concepts developed in the other subprojects, but technologically it is not yet feasible to accomplish. Instead, experimental research may be supported by simulation: simulated devices can replace physical devices in the other subprojects. In general, simulated products in one subproject could be employed to support the research in the other subprojects. Figure 1 shows how various subprojects are linked to form one multi-disciplinary program. In principle, the same holds from the viewpoint of underlying simulations and simulation-related activities. The various subprojects include simulations on different

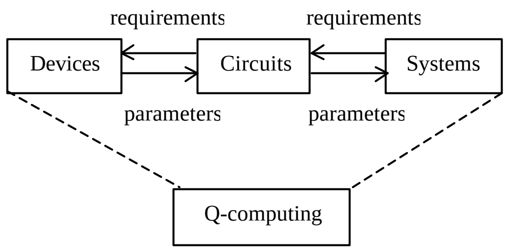


Figure 1. Global view on subproject links.

architecture levels, of different formalisms, in different languages and on different computers. Consequently, providing an appropriate infrastructure [Coe et al. 1998] to link such different project levels to distributed hierarchical compound simulations would be worthy (certainly in the long run) to allow prototyping of research ideas, and to answer "what-if" and optimisation questions.

The described scenario concerns in general a scientific experimentation, where simulation programs are just special cases of computational tools involved in the experimentation. In our design approach, we assume that the environment to be developed should make a number of services available to users through the WWW. These services include access to computational resources (e.g., simulation software) and data resources (e.g., data storage), that are allocated on various hardware and software platforms. Experimentation in such computing environment may involve data retrievals, running executables on remote machines, data-flow between computations, data format conversions, data visualisation, etc. More than one user is expected to use the system's resources at any moment of time. Some resources might temporarily not be available for some reason (e.g., because the resource is occupied to its full capacity, or due to the limitations of external factors, such as load balancing, license restrictions, policy considerations to name a few), which may influence the coarse of the experiment. Consequently, our goal is a generic environment (with web-based interface) for experimentation on distributed heterogeneous resources. The environment should provide support for distributed computations and management of resources, taking into account the above-mentioned external influencing factors; it should support the project life-cycle and feature central project data storage in order to facilitate knowledge exchange inside and outside the project. The generic system should be adjustable to cover a concrete situation at hand (in our case the afore-mentioned NanoComp-project with its specific resources).

Currently, prototypes of other similar collaborative environments are available or in development [Malony et al. 2000, Haupt et al. 2000], however these focus on different problem aspects (web-based front ends, support for data collection, support for running research tools) or implementation aspects (application of web-related technologies, OOP, distributed object technologies, etc.). Our research distinguishes itself by a different approach to resource management, and experiment control and execution.

The paper is organised as follows: section 2 shows our approach to modelling the experimental collaborative environment. Then, in section 3 is described how agent technology is intended to be applied in our approach. Section 4 provides an overview of the architecture for the proposed experimental collaborative environment. Finally, in section 5, future directions are outlined.

2. Modelling, Simulating and Controlling the Environment

2.1. View on the real system

First, we distinguish the generic entities that form the *physical aspect* of the system (i.e., the experimental collaborative environment): dynamic entities – *processes*, which are activities of users (entering and after having been served leaving the system); and static entities - *resources*, that are permanent in the system (e.g., computational tools, DBMS, etc.). Changes in the state of the system are caused by external events (e.g., an entry of a user into the system) and internal events (begin or end of an action). Such system can be described using the discrete-event formalism.

Resources are intended to provide services to the users, and users carry out (compound) activities in which these resources are involved. This leads us to the *information aspect* of the system, which represents the knowledge about activities of users. An activity may consist of one or more atomic tasks (actions) grouped together in a way and order that are crucial to the achievement of the objective of experimentation. Such tasks form the information aspect of a certain process and constitute its body. The latter can be represented as a sequence:

$$[t_1, t_2, \ldots, t_i, \ldots, t_n]$$

where t_i is a task from a process' body. If such a sequence produces good experimental results, it is important to save it for further reuse. In order to reflect optimally this aspect of the user's behaviour, we apply the process-interaction worldview (subset of DEVS).

Finally, we distinguish a *control aspect*, which provides the control of the interactions of the components in the system. This is accomplished by a kernel, which forms the core of the proposed collaborative environment. Since the kernel executes control over the two previous aspects, it inherits the DEVS process interaction formalism.

To summarise, our design approach is based on the process-interaction worldview on the system: multiple user-defined processes interact with resources and compete with each other for resources' services in order to realise their objectives. All the events in the system are controlled by the environment kernel (see subsection 2.2).

2.2. Environment kernel

The core of the environment, named Process-Oriented Kernel (POKer), is implemented in *Python* [Lutz 1996]. It is a discrete-event system solver with the process interaction worldview. The state of the environment at any moment is internally represented by the triple (EL, R, D), where EL is the event list that holds all the processes scheduled to happen in the system; R is a set of resources, and D is a set of (declarations of) process classes and their instances.

All resources are stored in a set as dictionary-like entities, whose bodies are accessible by their *ids*. Currently, a resource's body consists of an "availability attribute" (which indicates whether that resource is available), resource capacity, and a queue for blocked processes (if any) that have tried to acquire a busy resource. On the other hand, each process has an attribute, which holds the id of the resource (if any) being used by it, thus allowing to prevent double taking the same resource (and eventual deadlocking). The kernel's resources are proxy objects for "real" resources, and as such, they reflect the status of their real counterparts.

Every process that exists in the system (and is not blocked or busy with a resource) is assigned an event time to schedule the beginning of execution of its next task, and is accordingly allocated on EL. This feature is useful since it allows combining both simulative and real processes in the system. Besides the event time and the attribute mentioned above, each process is given a distinctive name and contains a body, which is a list of tasks to be accomplished in the system. The body is a process interaction model that captures the information aspect of the user actions in the experimental environment. Consequently, a process is a "dynamic entity" wrapper around the experiment model (i.e., model of user experiment as determined by the process' body) and also provides the internal attributes necessary for its processing inside the kernel. A typical correct body would consist at least of the following tasks: *seize a resource*, *do something* with the resource or *reschedule* itself at *Dt* later, *release the resource*, *create a report* on the coarse of the execution. Additionally, it may include *create a new process class*, *create an instance* of a process class, and *create a new resource*. Implementation of these commands is based on operational semantics study in [Birtwistle and Tofts 1994].

The dictionary of declarations (D) is implemented as a set, which contains class definitions for processes (i.e., their bodies or experiment models). This allows easy creation of multiple instances (processes) of already declared process classes. It also keeps in the same set the ids of already existing processes.

The above shows how the physical and information aspects are represented internally. Furthermore, in controlling the operation of the environment, POKer relies on a number of mechanisms. A s*ynchronisation mechanism* controls the access of processes to resources. A s*cheduling mechanism* manages the processes on the EL. A g*ateway mechanism* provides the connectivity to real resources. Finally, POKer employs a standard discrete-event *timing mechanism* to progress its execution (i.e., pop the next event (process) from EL, initiate the corresponding event (next task from the process' body), react to the event according to the system's state, and repeat until there are no more future events on EL). The integrated mechanisms define possible applications of POKer both as simulator and controller of the experimental collaborative environment.

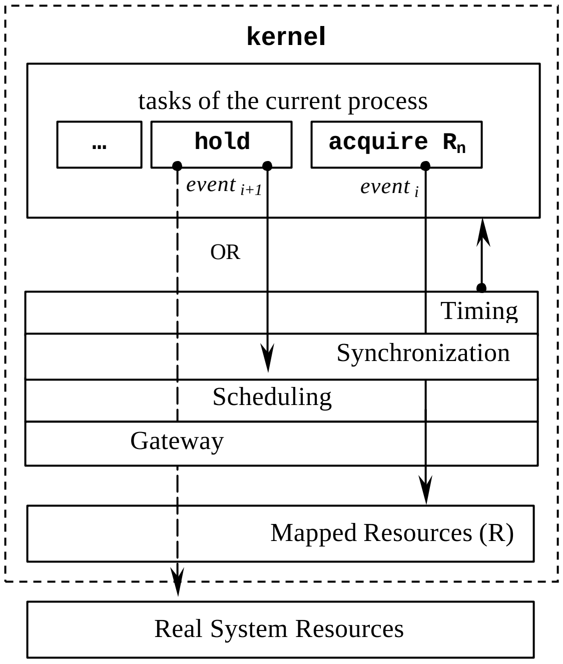
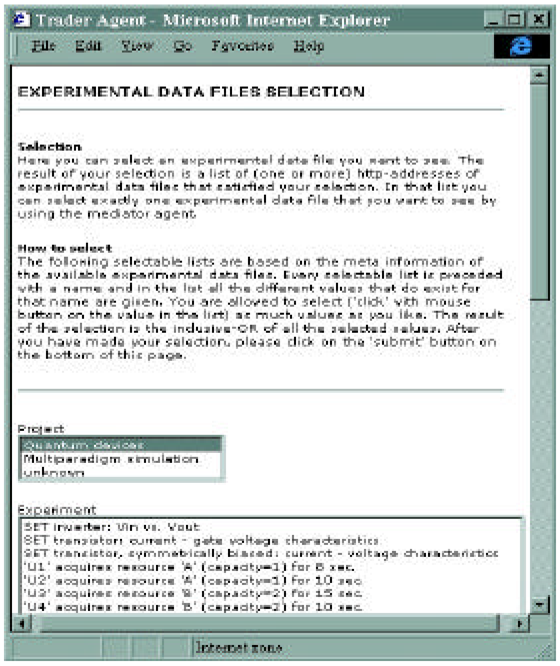


Figure 2. POKer's mechanisms at work.

POKer as Environment Simulator
POKer is capable of simulating 'exclusive access to a limited number of resources' scenarios that dynamically occur during execution of experiment models. As such, POKer can be used as a simulator of our planned experimental environment. Based on the assumed distribution function of the users' arrival times, their process specifications (experiment models), POKer's dictionaries of resources (R) and process declarations (D), through simulation POKer can provide information on how tasks are processed (i.e., their sequence of execution), whether resources are free or blocked, and, based on this, the interaction of processes. These simulations can be used to study and analyse the "real-world" experimental environment (queues of the resources, deadlocks, consequences of changes in the system, etc.).

POKer as Environment Controller
In our research, the emphasis is not on POKer as experimental environment *simulator* but on POKer as experimental environment *controller*. This function of the kernel is enabled by providing processes with a gateway mechanism to external


Trader Agent - Microsoft Internet Explorer
File Edit View Go Favorites Help
Internet zone

EXPERIMENTAL DATA FILES SELECTION

Selection
Here you can select an experimental data file you want to see. The result of your selection is a list of (one or more) http-addresses of experimental data files that satisfied your selection. In that list you can select exactly one experimental data file that you want to see by using the mediator agent.

How to select
The following selectable lists are based on the meta information of the available experimental data files. Every selectable list is preceded with a name and in the list all the different values that do exist for that name are given. You are allowed to select ('click' with mouse button on the value in the list) as much values as you like. The result of the selection is the inclusive-OR of all the selected values. After you have made your selection, please click on the 'submit' button on the bottom of this page.

Project
Quantum devices
Multiparadigm simulation
unknown

Experiment
SET inverter: Vin vs. Vout
SET transistor: current - gate voltage characteristics
SET transistor, symmetrically biased: current - voltage characteristics
'U1' acquires resource 'A' (capacity=1) for 5 sec.
'U2' acquires resource 'A' (capacity=1) for 10 sec
'U3' acquires resource 'B' (capacity=2) for 15 sec
'U4' acquires resource 'B' (capacity=2) for 10 sec

Figure 3. Data-file selection page.

resources (real applications). This mechanism is fired during execution of the `hold` command on the process that has acquired a resource, *which is a proxy object for a real application.* This is demonstrated in figure 2 by the dashed arrow going through the 'gateway mechanism', as opposed to the solid one denoting simulative use of the resource. Then the respective module from the library of proxy modules that wrap "real" resources, is called. The proxy module controls the session with the "real" resource, and maintains the communication with the client-process. Each wrapper stores the interface definition of the server application and knows where to find the implementation of that resource. Currently, custom wrappers are used. Later, we plan to strengthen this mechanism by applying one of the object technology standards [ON].

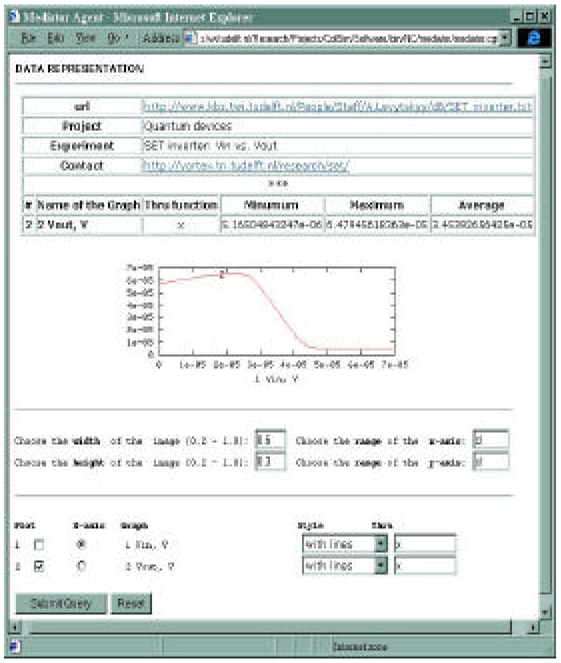
Since POKer itself is just a running operational model, it becomes possible to make fast changes to it, which may influence the operation of the whole system, thus making the environment more flexible.

3. Agent-like Approach

Users, working with the proposed software environment, will have to interact with the parts of the system, such as environment kernel itself and the modelbase. This might include, for example, registration of simulation models and data-files to be added to a modelbase, browsing the modelbase and selecting a model or data-file, presenting data from the selected data-file in a user-specified way, starting simulation of models under user-specified conditions, linking distributed simulations, etc. Previous research [Wong and Hwang 1992], [Kerckhoffs and Vangheluwe 1996] indicates that software agent technology can be applied to provide the user with friendly web-based interfaces to the system, and AI assistance in the execution of the wanted activities. Currently, the system contains trader and mediator agent-like components. In general, a *trader agent* keeps descriptions of available information sources, such as the type of information, the method of access and the location of the information source. It offers the user search mechanisms to identify possibly interesting information sources. A *mediator agent* is positioned between (distributed) information sources and users (or applications). Tasks that are associated with mediation are, for instance, methods to access and merge data from multiple databases or data-files, and computations that support abstraction, generalisation and representation of underlying data. In our system these agents are accessible from the WWW through a browser.

The trader agent enables users within the NanoComp-project and, in general, around the world to share experimental data: users have the possibility to register new experimental data and they can select experimental data. This agent has access to a modelbase that contains the meta-information of the registered experimental data files. Web-users can register an experimental data file by providing the http-address of the file. The trader agent then downloads this file from the provided location and adds all the meta-information of that file to the model base. The data itself is not added to the database, because there is no need for that for the purpose of data file selection, and furthermore, experimental data files can be large, which would complicate their management in a database. Selection of experimental data is done on the basis of this meta-information of the registered data file.

Example: figure 3 shows a web page generated by the trader agent. It summarises the meta-information of registered experimental data files as pairs of `keywords` and `values`. The latter are presented as *'select boxes'* and the former as descriptions of the respective boxes. In the example, the page has boxes with the following keywords: *Project*, *Experiment*, etc. The user selects '*Quantum Devices'* from the box with the list of projects, and clicks the 'Submit' button to request the trader agent to return a list of http-addresses from the available experimental data files, that have *Quantum*



DATA REPRESENTATION

url	http://www.kbs.twi.tudelft.nl/People/Staff/A.Levytskyy/db/SET_inverter.txt
Project	Quantum devices
Experiment	SET inverter: Vin vs. Vout
Contact	http://vortex.tn.tudelft.nl/research/set/

#	Name of the Graph	Thru function	Minimum	Maximum	Average
2	2 Vout, V	x	5.16504843247e-06	6.47945818363e-05	3.46392656425e-05

Choose the **width** of the image (0.2 - 1.8): 0.5 Choose the **range** of the **x-axis**: 0
Choose the **height** of the image (0.2 - 1.8): 0.3 Choose the **range** of the **y-axis**: 0

Plot		X-axis	Graph	Style	Thru
1	☐	●	1 Vin, V	with lines	x
2	☑	○	2 Vout, V	with lines	x

Figure 4. Graphical presentation of the selected experimental data file.

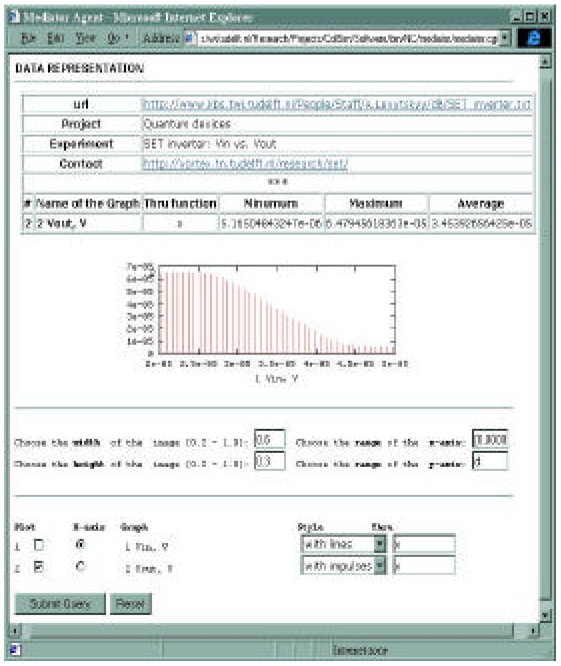
Devices as value for keyword *Project* in their meta-description.

After a user has selected the experimental data-file s/he wants to have access to, its http-address becomes the input for the mediator agent. This agent consecutively retrieves the experimental data file and provides a meaningful (e.g., graphical) presentation of these data. The experimental data will be presented as a graph and the user has the possibility to change the presentation according to his/her preferences. The user can:

- change the size of the graph;
- zoom-in/zoom out on a particular part of the graph;
- choose a data column for the x-axis;
- choose data columns to be plotted;
- pipe data of a column through a function, before plotting;
- customise graph styles for columns (solid line, dashed line, or impulses).

The mediator agent will also offer the user a link to the original data file.

Example: figure 4 demonstrates an example of default graphical representation of the selected data. Besides the graphical representation itself, this page provides the user with meta-information on the data and statistics (the table above the image). Beneath the picture, a number of widgets are available to allow the user to customise the look of the graph. For example, the user altered the range ([2e-05 : 5e-05]), selected impulse in place of line chart, etc. Pressing the "Submit" button returns a new html page with user preferences being applied to the graphical representation for the same data (see figure 5).



DATA REPRESENTATION

url	http://www.kbs.twi.tudelft.nl/People/Staff/A.Levytskyy/db/SET_inverter.txt
Project	Quantum devices
Experiment	SET inverter: Vin vs. Vout
Contact	http://vortex.tn.tudelft.nl/research/set/

#	Name of the Graph	Thru function	Minimum	Maximum	Average
2	2 Vout, V	x	5.16504843247e-06	6.47945818363e-05	3.46392656425e-05

Choose the **width** of the image (0.2 - 1.8): 0.6 Choose the **range** of the **x-axis**: [illegible]
Choose the **height** of the image (0.2 - 1.8): 0.3 Choose the **range** of the **y-axis**: 0

Plot		X-axis	Graph	Style	Thru
1	☐	●	1 Vin, V	with lines	x
2	☑	○	2 Vout, V	with impulses	x

Figure 5. Graphical presentation of the selected experimental data file (continued).

On the basis of user preferences, the mediator agent writes a plotting–script that is used by the visualisation tool *gnuplot* to draw the graph. The mediator agent in turn includes this graph in the (HTML) document, which is returned to the user.

The realised tentative agent-like components allow the user to register, find and visualise data-files. Further on, if the selected file is an experiment model, user can execute it on the environment kernel.

4. Overview of the collaborative environment

The proposed environment consists of three layers (see figure 6):

- High-level front ends
- Environment kernel
- Distributed heterogeneous (hardware and software) resources.

As to the first layer, we intend not to restrict users to particular front-end tools, but to allow them to use custom front-ends by "plugging" these into the system. This, however, requires finalising the specification of the kernel's interface. This task is currently in an initial stage. At the moment, the front-end layer consists of the kernel control utility, which is an administration tool that normally is not accessible for usual users, and

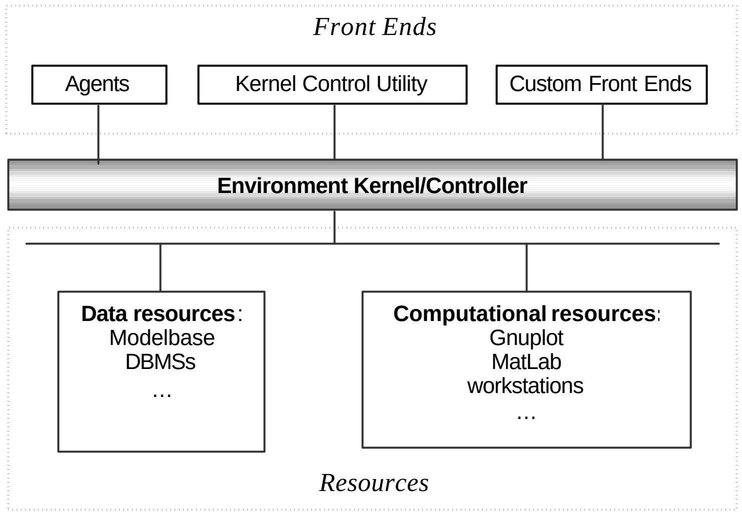

Front Ends
Agents
Kernel Control Utility
Custom Front Ends
Environment Kernel/Controller
Data resources: Modelbase DBMSs …
Computational resources: Gnuplot MatLab workstations …
Resources

Figure 6. Architecture of the system.

agents. The latter carry out various functions: they facilitate the exchange of models and experimental data between participants from inside and outside the NanoComp project. Agents are also used to register experiment models to the model-base and provide intelligent assistance in browsing through registered objects in order to select a model for reuse. Further, a user can proceed with execution of a selected model without having to deal at low level directly with the kernel's API.

The middle layer of the environment consists of one central persistent kernel, which controls access to all the resources in the system. When a user starts an experiment (e.g., via an agent), a low-level experiment description is sent over a socket to the listening thread of the kernel, which collects the incoming data in an internal buffer. POKer regularly removes any new processes from this buffer and schedules them on the EL. A process contains a sequence of events. When an event occurs, POKer executes its controlling function and delegates the task of the process to the respective resource, by binding it with the proxy module that calls the methods of the target resource. Currently, these modules are custom wrappers developed for the applications concerned. When a user leaves the system or works with a resource, the connection between the user and the rest of his/her process is closed, but the process stays in the system as suspended (i.e., it does not influence the queue of events and does not change the state of the system). Every web-user is associated with a process by *cookies*: a general mechanism which server side connections (such as CGI scripts) can use both to store and retrieve information on the client side of the connection [PCS]. When a user comes back in the system, s/he is recognised by the cookie, and the respective process becomes active again (i.e., it is scheduled as an event notice on the EL).

The back-end layer consists of a number of applications (currently accessible via CGI-wrappers). Their state is reflected in their respective proxy resources. Each experiment model can involve execution of applications on different platforms. The output file of one application in the experiment sequence can be the input for the next one, after a suitable format conversion. A CGI-based communication scheme is used to provide the input and output information, and to execute the computational resources. We plan to integrate later on CORBA [OMG] to facilitate the communication between middle and back-end layer.

It might be important to collect status information on the coarse of experiment execution. This would allow users to analyse their experiments post-mortem, which might give new interesting insights. Currently, this is limited to the report generated by the kernel, and the possibility to register the produced experimental data to the modelbase and to visualise the registered data with help of the mediator agent. Because the kernel brings the execution of an experiment at a higher level, it has the potential to facilitate information gathering with respect to data-flow, actions, and intermediate states of the experiment. This facilitates the implementation later on of the "Virtual Product Lifecycle" (VPL) concept.

5. Final remarks

The paper presents some tentative results of research that aims at developing and building a generic multi-user environment with web-based interface for running experiments on distributed heterogeneous resources. We have considered an approach, in which a process-oriented discrete-event system kernel is employed to simulate and especially control the planned collaborative environment. A first prototype of (trader and mediator) agents provides users with a front-end to certain facilities of the system (in particular, the registration of data files and the graphical representation of registered data).

Recent experiences in the subject field provided us with important guidelines on further developments in the project. Next work will focus on the integration in the middle layer of one of the industry standard object technologies, such as CORBA or DCOM [MS], to deal with the distribution of objects (resources). Then, the kernel of the environment shall be a combination of POKer and an ORB (Object Request Broker). The front-end layer will be extended by a web-accessible tool for visual construction of experiment models. Also, we intend to extend the simulation experiment potential of the environment by incorporating HLA (High Level Architecture) [DMSO] to set up generic federations for distributed simulations.

Acknowledgement

The research reported in this paper is done in the framework of the NanoComp-project [DIOC-7], financially sponsored by the TU-Delft.

References

[Birtwistle and Tofts 1994] G. Birtwistle and C. Tofts, “An operational semantics of process-oriented simulation languages: Part 1 πDemos”, Trans. Soc. Comput. Simul., 10(4), December 1994, pp. 299 - 333

[Coe et al. 1998] P. S. Coe, F. W. Howell, R. N. Ibbett and L. M. Williams, “Technical note: a hierarchical computer architecture design and simulation environment”, ACM Transactions on Modeling and Computer Simulation, Vol. 8 (4), Oct. 1998, pp. 431 - 446.

[DIOC-7] NanoComp project homepage: http://nanocom.et.tudelft.nl/

[DMSO] DMSO - HLA project homepage: http://www.dmso.mil/portals/hla.html

[Haupt et al. 2000] T. Haupt, E. Akarsu and G. Fox, “WebFlow: a framework for web based metacomputing”; Elsevier Science, Future Generation Computer Systems, Vol. 16 (5) (2000) pp. 445-451

[Kerckhoffs and Vangheluwe 1996] E. Kerckhoffs and H. Vangheluwe, “A Multi-Agent Architecture for Sharing Knowledge and Experimental Data about Waste Water Treatment Plants through the Internet”, proceedings of Euromedia, 1996, pp. 18 - 26. Society for Computer Simulation International (SCS).

[Lutz 1996] Mark Lutz, “Programming Python”, O'Reilly & Associates, Inc., October 1996.

[Malony et al. 2000] A. D. Malony, J. E. Cuny, J. L. Skidmore and M. J. Sottile. “Computational experiments using distributed tools in a web-based electronic notebook environment”; Elsevier Science, Future Generation Computer Systems, Vol. 16 (5) (2000) pp. 453-464.

[MS] Microsoft's Component Object Model technologies site: http://www.microsoft.com/com

[OMG] OMG homepage: http://www.omg.com

[ON] Object News: http://www.objectnews.com

[PCS] Persistent Client State - Http Cookies: http://home.netscape.com/newsref/std/cookie_spec.html

[Wong and Hwang 1992] Yung-Chang Wong and Shu-Yuen Hwang, “Knowledge-Based Distributed Simulation Generator”; ACM, Proceedings of the 25th annual symposium on Simulation, 1992, pp. 156 - 161.