ISS-Studio: A Prototype for a User-Friendly Tool for Designing Interactive Experiments in Problem Solving Environments

Z. Zhao[1], G.D. van Albada[1], A. Tirado-Ramos[1], K. Zajac[2], and P.M.A. Sloot[1]

[1] Section Computational Science
University of Amsterdam
Kruislaan 403, 1098SJ, Amsterdam, the Netherlands
{zhiming, dick, alfredo, sloot}@science.uva.nl
http://www.science.uva.nl/research/scs
[2] Institute of Computer Science, AGH, al.Mickiewicza 30,
30-059 Kraków, Poland
kzajac@uci.agh.edu.pl

Abstract. In Problem Solving Environments (PSE), Interactive Simulation Systems (ISS) are an important interactive mode for studying complex scientific problems. But efficient and user-friendly tools for designing interactive experiments lack in many PSEs. Mechanisms, such as data flow and control flow diagrams, adopted in many current PSEs to specify the component interconnection and interaction scenarios are derived mostly from a data processing perspective, and are not suitable for designing user-centred interactions. ISS-Conductor is an agent-oriented architecture for ISS components. It uses an extended finite state machine to model the run-time behaviour of a component, and adopts first order logic to represent the interaction constraints between components and to implement them in the knowledge bases of agents. ISS-Conductor separates the basic computational functions of a component from its run-time behaviour controls, and provides a high-level interface for users to design interaction scenarios. In this paper, we prototype a user-friendly tool for using components based on ISS-Conductor to design interactive experiments.

1 Introduction

A Problem Solving Environment (PSE) is a complex and integrated computational environment that provides all kinds of facilities needed to solve a given class of problems [1]. In industrial design and scientific research, PSEs have proved themselves by providing solutions to complex problems [2–4]. In general, a PSE consists of a set of functional components or tools for its target problem domain, a user interface for designing and performing experiments, and a run-time environment for computation and for managing resources. A solver for a complex domain problem is decomposed into a number of functional components or tools, which are then interconnected and executed under control of the PSE.

P.M.A. Sloot et al. (Eds.): ICCS 2003, LNCS 2657, pp. 679–688, 2003.

In PSEs, trial solutions are often called *experiments*. In many current PSEs, *data flow* and *control flow* are the mechanisms adopted to specify the interconnection between functional components. In the *data flow* mechanism, data pipelines depict the interconnection between the data channels of components are used to represent the dependencies between components. Processing and transferring data objects are basic interactions between components, in systems like SciRun [5]. In the *control flow* mechanism, interaction and execution dependencies between components are represented by a sequence of actions or tasks that should be performed by components. *Workflow* system is a typical example, such as in DISKWorld [6]. In many *control flow* systems, data state information is also used as a complement for interaction controls.

In PSEs, *human in the loop* experiments are increasingly important for studying complex problem spaces. Interactive Simulation Systems (ISS), which couple simulation modules and visualisation tools together, and include a human user in the run-time loop to manipulate simulation parameters are an example. For complex problems, sophisticated interactions between simulators and interactive visualisation tools are often needed. However, in the current data flow based mechanisms, only data processing related interaction controls can be specified. These are not sufficient for handling synchronisation between parallel components and human interactions. In the control flow mechanisms, tasks sequences are predefined, but user-centred and event-driven interactions are not easily included. A flexible mechanism for describing the interconnection of components is needed in current PSEs, and it should be supported by a user-friendly tool.

In this paper, we continue our work on an agent-oriented middleware for constructing Interactive Simulation Systems [7]. First, we will review our earlier work on an agent-oriented ISS architecture, and then we will analyse the design requirements for a user-friendly tool for constructing interactive experiments. After that we will use an example discuss how such tool should be implemented.

2 ISS-Studio: Design Overview

2.1 Earlier Work: An Agent Oriented Framework

In our earlier work, we have invented an agent-oriented software architecture, Interactive Simulation System Conductor (ISS-Conductor), for implementing and interconnecting distributed interactive simulation components [7, 8]. In ISS-Conductor, we use a layered interconnection mechanism: at the lower-level, messages between modules are carried by Communication Agents (ComAs), and at the higher-level, application logic is controlled by Module Agents (MAs). *Components* are major units in an ISS application. In ISS-Conductor, each component contains two parts: an *Actor* and a *Conductor*, both of which contain a *ComA*. The *Actor* realises computational functions of a component, and the *Conductor* contains a MA for controlling run-time behaviour of the component. At run-time, the *Actor* and *Conductor* of a component are separate processes, which communicate with each other and the other components via a *software bus*. The

software bus is normally the run-time infrastructure of the communication middleware adopted by the ComAs. The interaction scenarios between modules are represented as knowledge bases, which can be bound to MAs at run-time. Fig. 1 depicts the architecture.

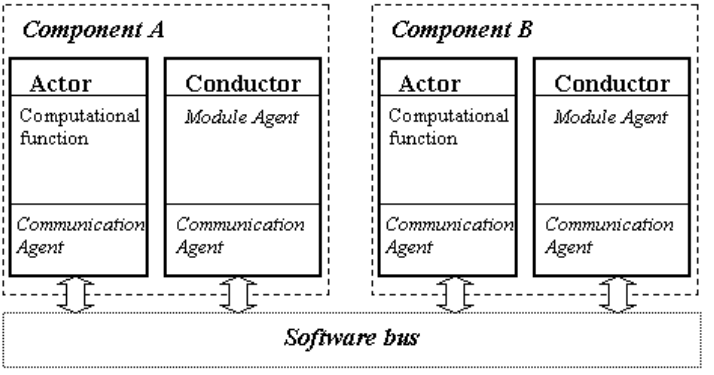

Fig. 1. ISS-Conductor components and their interconnection.

In the current implementation of ISS-Conductor, the Run Time Infrastructure (RTI) 1.3NGV5 of High Level Architecture (HLA) [9] is the communication interface between ComAs, and Amzi Prolog [10] is used to implement the reasoning engines in the MAs. ISS-Conductor is part of Polder [11], a computing environment built by the University of Amsterdam. In Polder, ISS-Conductor is the emerging framework architecture for Interactive Simulation Systems. In the rest of this paper, we will describe a prototype of a user-friendly tool, called ISS-Studio, for developing ISS by using ISS-Conductor components.

2.2 Functional Requirements for ISS-Studio

ISS-Studio aims to be a user-friendly tool for scientists to design interactive ISS-Conductor experiments. It must support different configurations for experiments: multiple simulation and interactive visualisation modules can be combined, and users can be geographically distributed. ISS-Studio should be user-friendly, flexible, easy to use, and robust. It should provide an integrated environment, which covers most basic procedures for designing and executing experiments, such as problem analysis, scenario design and execution management. It should be an open environment; components can be shared between organisations. For portability, ISS-Studio should support most popular computational infrastructures for executing experiments. Decision support for experiments should also be provided.

2.3 Basic Functional Modules

Managing components, designing experiments, and executing experiments are the three main functionalities to be provided by ISS-Studio. Fig. 2 shows the basic module-diagram of ISS-Studio. The *component management* sub-system provides an interface for managing software components, such as storing, querying, outsourcing, and updating. The *experiment management* sub-system supports basic procedures to design and validate experiments. Iterative development mode is a popular and practical approach for designing and implementing software applications. The steps depicted in Fig. 3 will be considered as basic procedures for experiment development life cycle in ISS-Studio. The *execution management* sub-system executes an experiment on the computational infrastructure. The current implementation of ISS-Conductor is based on HLA. ISS-Conductor is also being integrated with Globus [12] to provide grid access. The interaction between RTI and Globus is one of our considerations for the migration, which is discussed in a separate paper [13]. This sub-system provides interface to execute experiments and monitor their resource consumption and progresses.

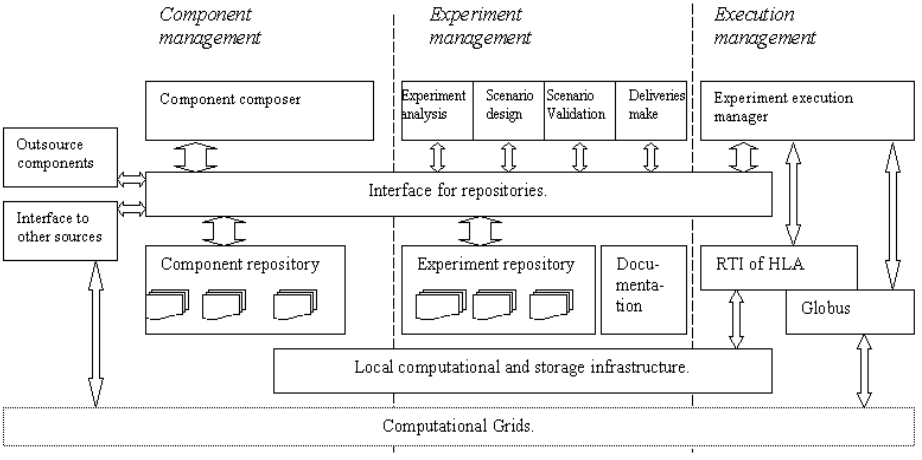

Fig. 2. Functional modules in ISS-Studio.

3 Design

3.1 Component Management

Components are stored in a repository, which provides an interface for searching, browsing and updating. To share software resources with other organisations, the repository will also provide an interface for outsourcing and insourcing components based on a service architecture.

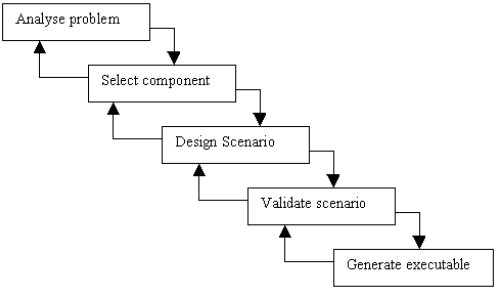

Fig. 3. An iterative model for designing interactive experiments in PSEs.

In the repository, each component has an *actor*, a *conductor* and an interface specification. In the interface specification, actions and states of the *component*, data object interface, execution requirements, and the version of the implementation are described. XML [14] is considered as basic language for the specification. ISS-Studio should provide an interface for incorporating normal programs into the ISS-Conductor architecture, for updating available components, and for refining their interface specification.

3.2 Experiment Management

The *experiment management* sub-system assists users at each main step of the experiment life cycle depicted in Fig. 3. To help users to build a solution and map it onto suitable components, a *decision support agent* is desirable. The decision support agent can search the experiment repository and find similar experiments as examples for the user, and search for proper components. If no proper components are available in the local repository, the decision support agent should contact the repositories of the other PSEs.

Considering components are the major units in a PSE, composing interaction scenarios between these components becomes an important step in designing experiments. The *extended finite state machine* [7] is adopted as the basic mechanism in ISS-Conductor to model run-time behaviour and represent interaction scenarios. Therefore, an interaction sequence chart will be a suitable visual interface for specifying interaction cases of experiments. An experiment may contain multiple scenarios; composing global scenario-switch and detailed interaction sequences in each scenario in a hierarchical way will be necessary.

From the sequence chart, dependencies between components can be automatically generated and represented in Prolog, which is the representation language of the knowledge bases of the MAs. To validate the scenario, a simulation tool will be provided. This simulation tool will execute all knowledge bases in a multi-threaded way, and track the run-time behaviour of all modules.

Finally, an executable of the experiment can be generated. It can be executed both inside and outside ISS-Studio. This sub-system will also include a documen-

tation management tool to generate and maintain all necessary documentation for an experiment.

3.3 Execution Management

The *execution management* sub-system provides an interface for users to execute experiments. The interface will be able to interact with job scheduling tools, such as PBS [15], to submit computing jobs and monitor the progress of the execution. In this sub-system, general tools for visualising experiment federation, and resource consumption, like software traffic, will be provided. During each execution, log files will be maintained.

4 Prototype

4.1 An Example

In this section, we use an example to discuss how ISS-Studio will be implemented to support interactive experiment design. In surgery, verifying an operation plan is a difficult task, even for expert surgeons. Computer simulation may help a surgeon to validate his treatment, but it is almost impossible to let a computer simulate all possible solutions. In an interactive experiment, a human expert is put into the simulation cycle to let him apply his expertise to confine the problem space. For complex cases, more than one expert can attend the experiment at same time, and they can be at different locations. In the experiment, a simulation program on a parallel computer system simulates the patient's blood flow. One or more visualisation modules present the simulated results together with the body's geometrical information obtained from a medical scanner (such as CT or MRI). The visualisation and interaction modules can be executed in immersive virtual environments such as (CAVE) [16] or desktops, from which a user can study the results of a trial treatment, and modify it when necessary [17, 18]. The simulation and interactive visualisation modules are connected through a high performance network. Due to the massive computation of the simulation, the experiment also includes a storage module, which can store intermediate simulation results and allow users to browse their earlier interactions.

4.2 Components and Scenario Design

The *component composer* tool should provide a direct manipulation GUI to edit elements of a component, such as the action set of the *actor*, the knowledge base of the *conductor* and the interface specification of the component. The interface specification can be automatically adapted when the user changes the action definition of the component. Fig. 4 shows an example interface.

Table. 1 specifies basic scenarios of the experiment. The *scenario design* tool provides a two-level view for specifying scenarios. At the top level, a global scenario-switch graph is designed, and each scenario can be zoomed into a detailed lower level. At the lower level, popup windows are provided to assist

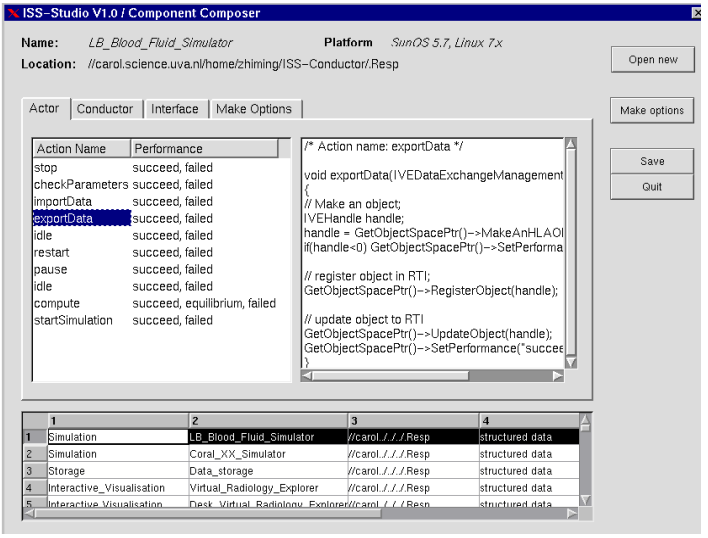

Fig. 4. The user interface of *Component Composer*.

Table 1. Scenarios definition of the experiment.

Scenario 1: live_Visualisation	
Involved components:	*Scenario specification:*
LB_Blood_Fluid_Sim	*LB_Blood_Fluid_Sim* computes and regularly updates a data object, called *trial_object*;
VRE	*VRE* and *Desktop_IV* visualise *trial_object* and refresh their
Desktop_IV	visualisation pipelines when *trial_object* has been updated;
Storage	*Storage* stores *trial_object* to disk when received a request.
Scenario 2: making_Trial_Operation	
Involved components:	*Scenario specification:*
VRE	*VRE* generates and updates an object, called *plan_Object*;
	Only *VRE* is allowed to update *plan_Object*;
Desktop_IV	*VRE* and *Desktop_IV* visualise *plan_object*, and refresh their
	visualisation pipelines when *plan_object* has been updated;
LB_Blood_Fluid_Sim	*LB_Blood_Fluid_Sim* adapts its computing parameters when received *plan_Object*.
Scenario 3: review_Experiment	
Involved components:	*Scenario specification:*
Storage	*Storage* generates and updates a object, called *history_Object*,
	when received a request;
VRE	*VRE* and *Desktop_IV* visualise *history_Object*, and refresh their
Desktop_IV	visualisation pipelines when *history_Object* has been updated;
	Only *VRE* is allowed to send requests to *Storage* to ask for updating *history_Object*.

users to select states and actions from a component and define its transition constraints with the state of the other components. Fig. 5 depicts a prototype GUI. When the user selects *scenario validation* from the interface, the scenario will be evaluated using a simulation program. The visualisation of the scenario simulation helps the user to validate his scenario design.

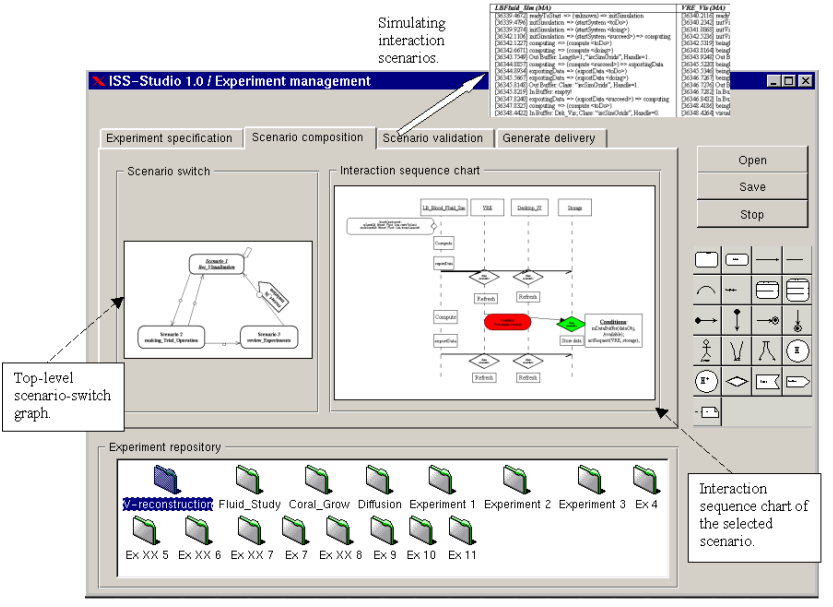

Fig. 5. The user interface of visually scenario designing.

4.3 Execution Management

Once the experiment has been designed, it can be executed. The execution tool assists users to execute the experiment on available computational resources. By providing the execution tool with a web-based interface, it can be linked an eventual portal-based interface of the PSE to support remote access of the experiment.

5 Discussion and Conclusion

In PSEs, ISSs are becoming an important mode for testing trial solutions to complex problems. A user-friendly tool is needed for scientists to efficiently design *the human in the loop* experiments. In this paper, we have prototyped such a tool for constructing ISS. First, we reviewed our earlier work, ISS-Conductor,

an agent oriented architecture for ISSs. Then we proposed a tool, ISS-Studio, for visually designing interactive experiments. ISS-Studio has not been fully implemented yet, but by prototyping the key features, we can already draw some conclusions:

1. The layered interconnection mechanism adopted in ISS-Conductor allows developers to specify application logic at an independent level. It provides possibilities to design experiment scenarios by using a visual interface.
2. In ISS-Conductor run-time behaviour of the system is modelled as an *extended finite state machine*, which can be visually represented by mechanisms such as interaction sequence chart. The conditions in the sequence chart can be automatically translated into first order logic to generate knowledge bases for MAs.

6 Future Work

As a next step, we will implement the ISS-Studio and include it in an existing PSE. We will also work on version 2 of ISS-Conductor which will support computational grids resource access.

Acknowledgements.

This research is partly supported by the European research project "CrossGrid".

References

1. John R. Rice Efstratios Gallopoulos, Elias Houstis. Computer as thinker doer: Problem-solving environments for computational science. *IEEE Computational Science and Engineering*, 2:13–23, 1994.
2. Marc Vass, Clifford A. Shaffer, and John J. Tyson. The jigcell model builder: A tool for modeling intra-cellular regulatory networks. In *Naren Ramakrishnan, Layne T. Watson, Submitted for review to HPC 2003*, 2003.
3. L. Boloni, D.C. Marinescu, J.R. Rice, P. Tsompanopoulou, and E.A. Vavalis. Agent based scientific simulation and modeling. *Concurrency: practice and experience*, 12:845–861, 2000.
4. Geoffrey Fox, Sung-Hoon Ko, Marlon Pierce, Ozgur Balsoy, Jake Kim, Sangmi Lee, Kangseok Kim, Sangyoon Oh, Xi Rao, Mustafa Varank, Hasan Bulut, Gurhan Gunduz, Xiaohong Qiu, Shrideep Pallickara, Ahmet Uyar, and Choonhan Youn. Grid services for earthquake science. *Concurrency: practice and experience*, 14:371–393, 2002.
5. C. Johnson, S. Parker, and D. Weinstein. Large-scale computational science applications using the scirun problem solving environment. In *Proceedings of Supercomputer*, 2000.
6. K. A. Hawick, H. A. James, and P. D. Coddington. A reconfigurable component-based problem solving environment. In *Proc. of Hawaii International Conference on System Sciences (HICSS-34)*, 2000.

7. Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. State update and sce-nario switch in an agent based solution to constructing interactive simulation systems. In *Proceedings of the Communication Networks and Distributed Systems Model-ing and Simulation Conference*, pages 3–10, San Anto-nio, US, January 2002.
8. Z. Zhao, R.G. Belleman, G.D. van Albada, and P.M.A. Sloot. Ag-ive an agent based solution to constructing interactive simulation systems. In *Proceedings of the second inter-action conference of computational science (ICCS02)*, Amsterdam, NL, April 2002.
9. Defence Modelling and Simulation Office (DMSO). High level architecture (hla) homepage. In *http://hla.dmso.mil/*, 2002.
10. Amzi Inc. Amzi prolog homepage. In *http://www.amzi.com*, 2002.
11. K.A. Iskra, R.G. Belleman, G.D. van Albada, J. Santoso, P.M.A. Sloot, H.E. Bal, H.J.W. Spoelder, and M. Bubak. The polder computing environment, a system for interactive distributed simulation. *Concurrency and Computation: Practice and Experience((Special Issue on Grid Computing Environments) in press)*, 2002.
12. The globus project homepage. In *http://www.globus.org/*, 2002.
13. Katarzyna Zajac, Alfredo Tirado-Ramos, Zhiming Zhao, Peter Sloot, and Marian Bubak. Grid services for hla-based distributed simulation frameworks. In *First European Across Grids Conference*, San Deigo, US, 2003.
14. World Wide Web Consortium (W3C). Extensible markup language (xml). In *http://www.w3.org/*, 2002.
15. Veridian System. Portable batch system homepage. In *http://www.openpbs.org/*, 2002.
16. C. Cruz-Neira, D.J. Sandin, and T.A. DeFanti. Surround-screen projection-based virtual reality: The design and implementation of the cave. In *SIGGRAPH '93 Computer Graphics Conference*, pages 135–142, 1993.
17. R.G. Belleman, J.A. Kaandorp, D. Dijkman, and P.M.A. Sloot. Geoprove: Geometric probes for virtual environments. In *proceedings of High-Performance Computing and Networking (HPCN Europe '99)*, pages 817–827, Amsterdam, The Netherlands, in series Lecture Notes in Computer Science. Springer-Verlag, Berlin, ISBN 3-540-65821-1, 1999.
18. R.G. Belleman and P.M.A. Sloot. Simulated vascular reconstruction in a virtual operating theatre. In *Computer Assisted Radiology and Surgery (Excerpta Medica, International Congress Series 1230)*, pages 938–944, Elsevier Science B.V., Berlin, Germany, 2001.