An agent based architecture for constructing Interactive Simulation Systems
Zhao, Z.

Citation for published version (APA):
Chapter 6

Towards an intelligent planning environment for interactive simulations

In the previous chapters, we discussed the architecture of ISS-Conductor and its utilisation in constructing interactive simulation systems. The layered integration mechanism in ISS-Conductor improves the flexibility of controlling the application logic of an ISS. Yet, the difficulties of describing Petri net based scenario nets may also hamper the introduction of ISS-Conductor in Problem Solving Environments. In this chapter, we discuss an approach to this problem in a proposed environment called Interactive Simulation System Studio*.

6.1 Introduction

Problem Solving Environments integrate computing technologies and provide an abstract environment for scientists to do research on various problem domains. Since the 1980s, Problem Solving Environments have become an important subject in the community of High Performance Computing [182–185]. Depending on the target domain of the system and the freedom that the scientists are allowed to customise the system interaction, a PSE may have different guises, e.g. an Interactive Simulation System with customisable configurations [186], or a library of solvers and its necessary user interface as in [16]. But at an abstract level one can always distinguish three main functional subsystems in a PSE: an environment for analysing problems and designing experiments, a collection of necessary software resources for building experiments and an environment for executing the experiments.

PSEs play a key role in the emergence of computer simulations, and in particular interactive simulations, as an important experimental paradigm for problems that are

*Parts of this chapter have been published in Z. Zhao, G.D. van Albada, A. Tirado-Ramos, K.Z. Zajac and P.M.A. Sloot. “ISS-Studio: a prototype for a user-friendly tool for designing interactive experiments in Problem Solving Environments”, in the proceedings of ICCS 2003, Melbourne, Australia and St. Petersburg, Russia, Part I, in series Lecture Notes in Computer Science, June, 2003.
Towards an intelligent planning environment for interactive simulations

difficult to solve using conventional methodologies like experiments using normal lab instruments. As we discussed in the earlier chapters, the complexity of implementing an ISS lies in three main aspects: developing valid simulation or visualisation kernels, coupling distributed modules of the system, and controlling their run-time activities. Employing the simulation or visualisation kernels from legacy systems can reduce both the risks and the costs of the development of an ISS. However, the customised integration mechanism resulting from such construction paradigm introduces a strong dependency between the constituent system modules and hinders the further deployment of the system in PSEs. One of the solutions is to use software component technologies: industrial components, e.g. Java beans and DCOM, and scientific computing components, e.g. CCA, have been used to encapsulate the simulation and visualisation systems and to facilitate the interoperability between them [16,89,90,184]. Most of the available architectures provide a description mechanism to specify the functionality of the components, e.g. the SOM (Simulation Object Model) in HLA and the SIDL (Scientific Interface Description Language) in the CCA, and an integration mechanism for assembling the components and for realising their run-time binding. In those architectures, the interface specifications are basically used to promote the interoperability between components; an explicit layer for controlling overall interactions is not defined. Using these architectures, complex activity constraints, e.g. multi-user interactive simulation, are often difficult to describe at the flow control level. Therefore low level component programming is needed, which still hampers the further introduction of ISSs in PSEs. Hiding these low level assembling and programming details from a scientist and allowing him to plan an experiment at a high level is desirable. Since the planning procedure will be partly automated by the system, we call this the intelligent planning of interactive simulations.

The intelligent planning is basically approached by mechanisms which support automatic (or semi-automatic) selection of components and derivation of the coupling details between them. The research on this subject received a great deal of attention after significant progress was achieved on the reusability and interoperability of the simulation components [187–189]. An efficient mechanism for selecting software components has been considered as a necessary step to approach the intelligent planning. A number of technologies were reviewed in [187], e.g. based on key words, facets, signatures, behaviour and semantics. One of the conclusions drawn from the paper is that semantic level component matching is essential to improve the searching efficiency. A number of researchers studied the feasibility of automating the compatibility check between the Simulation Object Model (SOM) and the Federation Object Model (FOM) of an HLA application, but most of the matching mechanisms are limited to the syntactical level, e.g. in [188]. Using predefined templates, e.g. Process Flow Templates [15], is a straightforward way to facilitate the composition of interactions between components. But the templates are mostly composed manually by domain specialists. The burgeoning applications of Service Oriented Computing [190] paradigms are an important force to push the research on automatic flow composition. One of the motivations is to compose the flow between intermediate services and to provide a transparent binding interface for the service requester. A number of
researchers have studied this problem in both architectures of web services and Grid services [189,191,192]. The basic idea is to distinguish the dependencies between the services according to their pre and post conditions on data, and describe them using a workflow description language.

The ISS-Conductor architecture provides solutions for encapsulating legacy simulation and visualisation tools, and for orchestrating their activities at run time. A Petri net-based control mechanism for component activities supports the description of sophisticated interactive scenarios. Automatically planning of ISS-Conductor based experiments exhibits a number of differences from the other related work. First, the capability descriptions of ISS-Conductor components are based on state machines; they provide extended information for the simulation object models, and thus it becomes feasible to include more sophisticated matching mechanisms than in [188]. Second, the execution of an ISS-Conductor system is based on HLA, but the interaction scenarios between the components are based on Petri nets; they provide semantics to complement the object model based composition with the activity constraints between components. One of the aims in this chapter is to study the feasibility of composing scenarios which support human-in-the-loop computing.

In this chapter, Interactive Simulation System Studio (ISS-Studio), a framework for deploying ISS-Conductor components in constructing interactive simulation based experiments will be proposed. First, we give an overview of ISS-Studio and enumerate its desired functionality. ISS-Studio is proposed specifically for the ISS-Conductor compliant software resources. It intends to work with existing generic PSE frameworks to enhance their services for supporting interactive simulation based experiments. We will discuss this issue using an example of Grid-based Virtual Laboratory Amsterdam (VLAM-G) [15,19], a general PSE framework developed at UvA. After that we discuss the basic procedures to automate the story composition for an ISS-Conductor based system. Finally, some experiments and earlier results will be presented.

6.2 A global picture

The main goals to propose ISS-Studio is to facilitate the development of ISS-Conductor based components and to simplify the construction of interactive simulation systems. In this section, we will first describe the desired functionality of ISS-Studio and then discuss the design requirements for them.

6.2.1 Proposed functional subsystems

From the lifecycle of developing ISS-Conductor based components and interaction stories for the integrated systems, the functionality of ISS-Studio is grouped into four subsystems: component management, knowledge management, experiment planning and run-time experiment management.
Component management

Incorporating existing standalone tools which are designed for a specific problem as reusable and customisable solvers for a spectrum of other problems [92] is an important way to enrich the software resource of a PSE. The first subsystem will aid component developers to incorporate legacy simulation or interactive visualisation programs into the ISS-Conductor architecture. The component management subsystem provides tools for component developers to construct and maintain components, e.g. to define a component capability, to develop code and to debug. The component products, including the capability specification, the source, the documentation and the binary are stored in repositories with version control. Services for retrieving and updating components from the repositories are also provided.

Knowledge management

An efficient reuse of the software components depends not only on the nature of the components but also on the mechanisms for searching and retrieving them from the repositories where they are stored. As we mentioned, conventional search techniques do not capture the run-time semantics of the components. In ISS-Conductor components, the actions are complemented with pre and post conditions: the requirements and influences on the data objects, but they do not guarantee that the retrieved actions provide the semantics that the component searching process needs because of the possible diverse meaning of the actions and data classes. One of the solutions is to synchronise the meaning of the vocabularies used in different repositories using a knowledge-based backbone; the concepts used for describing software resources, e.g. components and experiments, are associated with certain ontologies. A knowledge management subsystem is proposed for this function.

Experiment planning

The third subsystem is to plan ISS based experiments. It intends to aid a scientist to develop an interactive simulation based experiment at each phase of the lifecycle. The subsystem needs to provide a user-friendly environment and supports intelligent planning of the experiments. In the next section, we will come back to this point.

Execution management

An interactive experiment is executed on computing resources, e.g. supercomputers, clusters or high performance virtual reality environments. The fourth subsystem processes the resource requirements of an experiment and generates suitable job description for different types of computation resources. The execution management subsystem also provides an interface to interact with tools for execution monitoring and job migration.
6.2.2 Design requirements

The system must meet a number of requirements. The first one is the user-centred design; the system needs to consider different types of users, e.g. component developers and scientists, and their special requirements on the system interactions. Second, integrating commercial off-the-shelf (COTS) tools into the system is another important requirement; using mature COTS tools e.g. for supporting UML or XML, avoids unnecessary rebuilding of similar utilities. The third requirement is the portability of the implementation. The constituent tools of the system are likely to be distributed. Thus, a portable framework to glue these assets is needed, and in addition, diverse interfaces to access these tools can also improve the usability of the environment. The realisation of the system needs to benefit the existing platforms, such as the management of distributed resources in Grid environments. Finally, the feasibility to integrate with existing generic PSE frameworks also has to be taken into account. Realising special purpose PSEs using a generic framework has emerged as an important development paradigm [16, 19]. Services provided by these frameworks, e.g. for managing resources and run-time information, can simplify the development of ISS-Studio.

In the next two sections, we will first discuss how available Grid middlewares can contribute to the development, and then use VLAM-G as an example to discuss the feasibility to deploy ISS-Studio in existing PSE frameworks.

6.2.3 ISS-Studio and Grid environments

A core idea of Grid environments is to organise heterogeneous resources, e.g. computing elements, storage devices and software components, and share them among a group of trusted users (Virtual Organisations)† [194, 195]. Resource management is a central component in a Grid environment, it provides services for describing and discovering resources, for scheduling and monitoring them at run time, and for fault tolerance and security control. A number of resource management systems have been developed, e.g. Condor [61], Globus toolkit [196] for computing resources, European Data Grid [197] for data resources, and PUNCH [198] for services-based resources. For instance, in the Globus toolkit, resources are described using an extensible resource specification language (RSL), the resource requests are handled by resource brokers and processed through the information service provided by meta-computing directory services (MDS). In the Globus toolkit, job schedules are organised in a distributed paradigm.

The rich set of protocols defined in available Grid environments provides a suitable infrastructure for realising ISS-Studio, e.g. for component storing and discovery, and for execution monitoring. There is also a large research body on flow control in Grid environments [16, 137, 138]; most of them are based the Grid Service architecture and describing the flow using data or task based dependencies, e.g. in GridAnt [138] and

†Grid environments are also classified as: computational, data and service Grids according to the type of resources being managed and shared, e.g. computing elements, storage and software [193].
Towards an intelligent planning environment for interactive simulations

Taverna [137]. Compared to them, ISS-Conductor allows more sophisticated controls: the states of human activity and the components execution are allowed to control the flow branches, but the framework is currently based on HLA.

6.2.4 In the context of a PSE framework

VLAM-G is a generic PSE framework, which provides hierarchical solutions to manage different levels of resources, and encapsulates them as services in a middleware. On top of the middleware, domain specific PSEs are supported. The middleware allows users to work simultaneously and collaboratively at different levels of the framework, e.g. as scientists, domain experts, tool developers and ICT developers. It also integrates the information management services with the lifecycle of a scientific experiment [21]. An experiment is modelled using physical entities which are the instruments to be used, activities to be performed by the scientists, and data elements which are the input/output of the activities. An experiment is described as the flow between these elements; in order to simplify the construction of an experiment, templates of the flow are abstracted as Process Data Flow templates. A database infrastructure is employed to manage both the static and run-time information.

Compared to VLAM-G, ISS-Studio uses the term experiments in a much narrower sense. In ISS-Studio, experiments only refer to the interactive simulation based paradigm, and they can be included as part of a VLAM-G experiment. ISS-Studio focuses on the mechanisms that can facilitate the composition of ISS-Conductor based experiments. In the context of VLAM-G, ISS-Studio can be viewed as an upper level PSE, where the subsystems can benefit from the services provided by VLAM-G middleware, e.g. for managing resources and experiment information.

In this chapter we will not discuss the detailed issues on using the Grid services to realise ISS-Studio, but instead we focus on the intelligent planning of ISS-Conductor based interaction scenarios.

6.3 Intelligent planning of ISS-Conductor based interactive simulations

In this section, we will focus on the experiment planning subsystem and discuss the feasibility of intelligent planning of ISS-Conductor based interactive simulations. The goal of the subsystem is to allow a scientist to plan his interactive experiments from the level of the problem domain instead that of the details of scenario nets composition. We will briefly discuss the phases: requirement description, component discovery, story making and execution script generation.

\(^3\)Information and communication technologies.
6.3 Intelligent planning of ISS-Conductor based interactive simulations

6.3.1 Describing experiment requirements

The first phase aims to describe the requirements of the experiment. The description will be the input to the experiment planning environment. It provides information for the subsystem to determine the suitable components for the experiment and to distinguish the interaction constraints between the components in a story. The goal of employing interactive simulation in a scientific experiment is to use simulation solvers to compute data properties of a model, and to allow the scientist to study them by manipulating part of the data at run time. Therefore, we argue that the experiment description should at least contain three main elements: data, activity and the quality requirements.

1. **Data.** A scientist needs to specify the data for an experiment. It describes not only the raw data that the user has but also the data he expects during the experiment.

2. **Activities** indicate the action that the user will perform on the data. Some activities also indicate the transformations between data or causal relations between the data.

3. **Quality requirements** on data and activities describe the performance constraints of the experiment.

This model has a number of advantages. First, the activity flow of an experiment by nature is a sequence of operations on the simulation data. Although the implementation information of the simulation and visualisation kernels is not explicitly modelled, they can be included in the description as the quality requirements on the data or the activity. Second, mature software modelling techniques, e.g. data flow and control flow, can be directly used to describe the experiment. The description can be intuitively represented using graphical primitives. Fig. 6.1 shows an example of describing a bypass validation experiment. Finally, the description can be parsed and described using a logic language, e.g. first order logic, which can be parsed and reasoned on by agents for further searching and composition.

![Figure 6.1: A graphical representation of the experiment requirement for a bypass-validation experiment.](image)
6.3.2 Component searching

The actual planning procedure starts when the user provides an experiment description. Component searching is the first step, it finds a set of suitable components which can 1) produce all required data, 2) support all activities on them and 3) provide services with the required quality. In ISS-Conductor, the capability of a component is described based on a finite state machine model, in which data classes, activities, and the dependencies between the data and activities are explicitly described. It provides search agents basic information for match checking. Table 6.1 shows the details.

<table>
<thead>
<tr>
<th>Component capability</th>
<th>Experiment requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data classes (shared and internal)</td>
<td>Data</td>
</tr>
<tr>
<td>Activity</td>
<td>Activities</td>
</tr>
<tr>
<td>Pre-condition and post-condition of Activities</td>
<td>dependencies between data</td>
</tr>
<tr>
<td>Quality attributes</td>
<td>Quality requirements</td>
</tr>
</tbody>
</table>

As we mentioned above that the syntactical level matching does not guarantee the semantic level consistency between components and the requirements because the terminology used in the description of component capability and in the experiment requirement might have different meanings. One of the solutions is to synchronise the meaning of these concepts using a consistently defined Ontology.

Originally, the term of ontology refers to a philosophical discipline for dealing with the nature and the organisation of being [199]. Recently, it is used in computer science as a term for describing the semantic relations between the symbolic representations and the actual meaning of concepts; it normally consists of a vocabulary and a set of explicit assumptions regarding the intended meaning of the vocabulary [200]. The assumptions are represented using logic theories, e.g. first order logic or description logic. Based on the level of generality, different types of ontology are often distinguished as a hierarchical scheme, as shown in Fig. 6.2, [201]. According to the classifications, we define four groups of ontologies. The ontologies in the top-level group describe the most general concepts, e.g. ISS-Conductor components, component instances and interactive experiments. The ontologies in a domain group describe the concepts of different domains in software resources, which cover the terminology used in defining data object models in components and interaction stories. The ontologies in a task group describe the concepts of activities, services and their quality attributes, which are related to software resources. Finally the ontologies in an application group describe the concepts bound to specific applications. Taking the example in the previous chapter, the ontologies in the domain group describe the concepts in defining data object models in components and interaction stories e.g. fluid flow, blood, flow velocity and pressure. The ontologies in the task group include the concepts for describing component activities, e.g. flow simulation, visualising MRI images and designing bypass.
Using an ontology language, like OWL [202], concepts are described as classes, which can have subclasses and be a subclass of another class. A class can have a number of attributes, called properties or roles; the value restrictions on the properties are called facets. The instances of a class are called individuals of the class. An ontology together with a set of individuals of classes constitutes a knowledge base. Fig. 6.3 shows a screen snapshot of top level ontologies (edited using Protégé [203]). The terminology used in the resource descriptions, e.g. component actions, states and data classes, are mapped as individuals or subclasses of the classes in the ontologies. A resource description can be associated with more than one ontology.

In the searching procedure, the similarity between the concepts is first checked using ontology reasoning algorithms. Only the components which have equal or similar meaning of the concepts as the experiment description will be checked for the further matching.
6.3.3 Story generation

A story can be generated when the components have been found from the component repositories. It is a procedure to assemble these components and make a story for them to work together. It has a number of detailed steps.

1. The first step is to substitute the data and activities in the description of the experiment requirement using the components found. During the substitution, the roles of component instances and the common data interfaces between different roles will be defined.

2. Second, according to the condition and dependencies between activities, an intermediate activity diagram will be derived. In the activity diagram, the activities will be associated with specific role. In this step, a user can refine the control conditions between components in the loop.

3. Third, the control patterns of the activity diagram will be mapped onto Petri net. The activities and its responsible roles will be mapped onto transitions, and the conditions between activities are mapped onto places.

4. Finally, the Petri net is output as a story.

6.3.4 Generating execution scripts

The final step is to map a story onto the job description scripts of computing elements. The story contains information about the computing requirements of each of the components, e.g. requirements on the parallelisation libraries and hardware platforms, which can be used to generate a job description script using the demanded syntax provided by the description language of the computing elements.

6.4 Prototype and preliminary results

The implementation of ISS-Studio is still ongoing. In this section, we will describe the basic techniques that are used in the prototype and discuss some experimental results.

6.4.1 A multi-agent based experiment planning environment

ISS-Studio will be a distributed environment; when there are a large collection of component repositories, using multiple agents can improve the efficiency for component searching. A multi-agent environment is proposed for the experiment planning subsystem. An experiment manager agent (EMA) provides a graphical interface for users to describe the experiment requirements, and co-ordinates search agents to find
suitable components for the experiment. Component search agents (CSA) scan component repositories and search components according to the requirements sent by the EMA. Finally, the EMA also does the story making and execution script generation. The agents are prototyped using the JADE, a Java-based agent development framework for the FIPA standard [204]. In JADE, agents communicate using an Agent Communication Language (ACL) and are managed by an agent container at each host. The JADE framework provides services for managing the lifecycle of agents including cloning and migrating them between hosts. In an agent, the reasoning kernel of the agents is realised using Prolog. The ontology-reasoning module is realised using Racer [205] which can be shared by different agents. Fig. 6.4 shows its basic agent architecture.

![Agent structure diagram](image)

**Figure 6.4: The basic architecture of an agent.**

### 6.4.2 Experimental results

Experiments for testing the feasibility of integrating the JADE framework, Racer and the Prolog reasoning kernel have been performed. Because of the powerful support for network-based programming, SWI Prolog [206] is used in the prototype. The JADE framework is based on Java, and both SWI Prolog and Racer have a Java-based interface. The descriptions of component capabilities and the experiment requirements are parsed as Prolog terms; the Ontologies are in OWL and are processed by Racer. The Prolog reasoning kernel communicates with the Racer server via sockets. The JADE framework handles inter-agent communication. By gluing them using Java, the basic control between the functional components of an agent can be realised. The GUI of component management and experiment planning subsystems have been prototyped using a Java-based graphical library, JGraph [207]. Fig. 6.5 shows a screen snapshot. The GUI allows a user to directly describe the activity-transition graph (see Section 3.2.1) of the component and export as the ISS-Conductor required format. Fig. 6.6 shows a screen snapshot, which shows the experiment requirement described in Fig. 6.1: the rectangles and ellipses are respectively represent data and activities. The requirements are described using a set of triples, which are transformed into Prolog lists. From the planning menu, a user can start a CSA to discover the suitable components.
A reasoning kernel for a CSA has been prototyped. The prototype is able to find components from a given collection using the matching rules discussed above: the set of components can perform all the activities and process the data requested in the
requirement description. Since we do not have a large collection of ISS-Conductor components yet, the components (in total five) we discussed in the previous chapters are used as the basic collection. For the experiment purpose, we replicated them and created a number of dummy components; in total the component collection contains 20 samples. The capabilities of these components are parsed into 700 Prolog terms. In the experiment, the prototype can find components for each required activity, except the first two, visualise and segment MRI images (see Fig. 6.1). To get the feeling on search complexity, we run the experiment with different number of requested activities. Fig. 6.7 shows the measurements for two situations: all the requested activities and none of them can be found from the collection. For each requirement, the search procedure stops when it finds the first suitable component. When there is no component qualified for an activity, the search procedure takes more time, since it has to scan the entire collection of components, which indicates the upper bound of the searching time cost. Although the number of components is relative small, we can observe that, the search cost increase linearly with the number of requested activities in the requirement description.

![Figure 6.7: Searching different number of activities from the component collection.](image)

6.5 Discussion and conclusions

In this chapter, we have reviewed the background of Problem Solving Environments and their role in modern scientific research, and then discussed the feasibility of developing a framework for deploying ISS-Conductor based components to prototype interactive experiments. Compared to the related work, ISS-Studio introduces a number of novel ideas in its design. First, in ISS-Studio, an agent based support environment performs resource discovery and the interaction scenario composition, which intends to automate the procedures of experiment planning. In the other PSEs [114,208,209], the human guided assembly of components and interaction descriptions are still the principal development activity. Second, ontology based concept checking has been proposed in a number of service based computing systems, e.g. [210]. Finally, the scenario composition is performed before the system execution
in ISS-Studio. One of the reasons is that ISS-Studio intends to support human-in-the-loop interactions, rather than the one way data flow between components [92,211].

This leads to the following conclusions.

1. Component technology is suitable to encapsulate the functionality of software resources and to integrate them in a layered paradigm.

2. A semantic level searching mechanism is a key issue to automate the utilisation of component resources. Using a knowledge based backbone to synchronise the meaning of the concepts used in the component specification enhances the traditional searching mechanisms.

3. An agent-based framework is suitable for realising the experiment design environment for problem solving; employing agents to search for components and to plan experiments distributes the computation onto available resources and helps to achieve a better resource utilisation.

4. The rich set of services provided by the Grid environment constitute a suitable infrastructure for ISS-Studio to manage the component resources. We have not explicitly discussed the integration of ISS-Studio with Grid middleware. In our opinion the Java based platform and the XML based information description of ISS-Studio make the realisation possible.