Designing and Enacting Simulations Using Distributed Components*

Alberto Coen-Porisini, Ignazio Gallo, and Antonella Zanzi

Dipartimento di Informatica e Comunicazione – Università degli Studi dell'Insubria
Via Mazzini, 5 – Varese 21100, Italy
{alberto.coenporisini,ignazio.gallo}@uninsubria.it
antonella.zanzi@uninsubria.it

Abstract. In the simulation field the demand for distributed architectures is increasing for several reasons, mainly to reuse existing simulators and to model complex systems that could be difficult to realize with a single application. In this paper the ASIA platform that aims at supporting the simulation design and simulators integration is presented. The paper focuses mainly on the comparison of the ASIA platform and the High Level Architecture standard. An example in the manufacturing field is presented as a basis for the comparison of the two approaches. Finally, some considerations are outlined in the perspective of the integration of the two environments.

1 Introduction

Simulation allows one to see the effect of the design, configuration or control choices without having to build or modify such systems, providing in this way a flexible and cost effective way to assess design, configuration or control choices.

Many simulation tools are available supporting approaches based on different mathematical models. Moreover, in recent years the issue of interoperability among such tools has been addressed by many people. One of the main efforts has led to the definition of the High Level Architecture (HLA) [1], which was initially developed by the US DoD and more recently has become an IEEE standard [2]. HLA defines a common architecture supporting reuse and interoperability of simulations and is intended to have a wide applicability to many different areas. However, its practical use requires highly skilled people because of its inherent complexity. Moreover, HLA does not fully address the problem of providing an integrated design environment that one can use to design simulations while designing systems.

Another approach was introduced by the ESPRIT Project ASIA, which aimed at defining and implementing an open platform for supporting design and simulation activities and allowing the integration of simulation tools. The definition of such an environment required to identify all activities that occur when designing/simulating a system. The initial results led to the implementation of a CORBA based platform allowing interoperability among simulators.

* Work supported by the Italian MIUR-FIRB – Tecnologie abilitanti per la Società della conoscenza ICT.

C. Aykanat et al. (Eds.): ISCIS 2004, LNCS 3280, pp. 706–717, 2004.

This paper reports on the results of a long term research, aimed at defining an open-source platform for supporting design and simulation activities and allowing integration of simulation tools. The paper reviews the main results of the ASIA project and discusses the relationship among the ASIA approach and HLA by providing an evaluation of both approaches and by discussing how they can be actually integrated. One of the advantages we expect from such integration is in term of usability since users can carry out the activities related to designing and simulating systems within a single framework.

The paper is organized in the following way: Section 2 presents the ASIA approach; Section 3 provides a short description of HLA; Section 4 introduces an example in the domain of Flexible Manufacturing Systems and shows how it can be dealt with using both ASIA and HLA; Section 5 discusses the main differences and similarities between the two approaches; Section 6 discusses how ASIA and HLA can be integrated, while Section 7 reviews the related works. Finally, Section 8 draws some conclusions.

2 The ASIA Approach

The Esprit ASIA (1998-2001) project aimed at defining and implementing an open platform for supporting both design and simulation activities and allowing an effective integration of simulation tools. Two different application domains were taken into account: space communication and traffic management. Starting from the requirements expressed by end users of the above mentioned domains ASIA defined an environment in which all the different activities related to the design and simulation of systems were supported. However, many issues that were initially identified were not investigated during the project. Thus, the research on simulation integration has continued and is still ongoing. In what follows we summarize the ASIA approach referring to its actual status, which has significantly evolved since the end of the original project.

2.1 Simulation Design Process

In what follows we briefly discuss a process lifecycle, which is referred to as the *simulation design process*, even if what is taken into account is a simulation based system design. The design process guides system engineers through the enactment of their systems and can be modeled by a set of "macro" activities, which are general enough to be applied to almost any domain. The simulation design process, shown in Figure 1, comprises three main activities.

1. Defining the *Information Model* means to define the elements that belong to an application domain, which represent either the logical components of a system or the simulators. In the latter case it is referred to as a Simulation Information Model.
2. Designing the *System/Simulation Architecture* means to build a system by instantiating the elements of the information model. Depending on whether the information model provides the logical components of the system or the simulators, the architecture is referred to as System or Simulation Architecture.

3. Executing the simulation architecture means defining how it has to be simulated, that is, to define which simulations will be performed, what simulation models are used, and how they are grouped and organized to carry out the simulation.

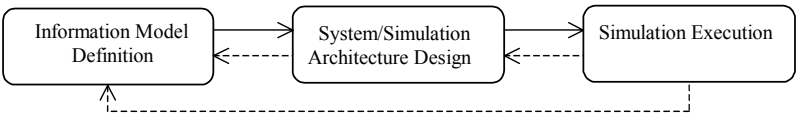

Fig. 1. Simulation process lifecycle (dashed arrows indicate feedback actions)

The first phase of the simulation design process is devoted to activities carried out before designing the integrated simulation. The simulation architecture, instead, provides a logical view of how the different simulation models cooperate. Such view provides both a data flow description, that is, which data are exchanged, and the control flow description, that is, the way in which the simulators interact.

In order to describe all artifacts (Information Model, Simulation Architecture, etc) produced during the design phases, ASIA defined a Simulation Architecture Description Language (SADL). SADL is based on a double language approach [3] that provides a domain independent abstract notation along with specific concrete notations, one for each domain. The abstract notation is defined as a simulation-oriented reuse of UML, domain-specific notations are obtained by means of the customization facilities of UML. In this way, one can define a specific notation for each domain as a transformation from the core notation. Once these transformations are defined, users can work using their own notations. Interested readers can refer to [4] for a thorough discussion of the ASIA meta-model and the associated SADL.

2.2 The ASIA Functionalities and Tools

The ASIA environment supports the following logical activities:
Modeling. Each entity involved in the previous mentioned phases is modeled using SADL. The core notation is used internally and is not viewed by users (except for the information model). Users rely on the domain-specific representation to define their models (system/simulation architecture).
Consistency Check. Each phase of the design process is based on the results obtained during the previous phases. Thus, it is necessary to check whether each phase is consistent with respect to the previous ones (e.g., the objects are connected according to their declared connectivity, when designing a system architecture).
Executing an Integrated Simulation. In order to execute an integrated simulation it is necessary to specify the actual data the simulators must use. Moreover, it may be also necessary to define where the output data will be stored, to set up some parameters for some simulators (e.g., time step) and so on. Providing all this information is referred to as *Setting up an experiment*. Once an experiment is set up one can run the experiment.

The ASIA approach is supported by three tools. The first one (IME) allows users to define an information model; the second one (SysAde) allows users to develop system-simulation architectures. Finally the third tool (DSC) allows one to define and

execute an integrated simulation starting from a simulation architecture. All the tools are under development using Java and XML and are open-source.

In the next sub-sections we will focus on the main features of simulation architectures and on the way in which simulators can be integrated.

2.3 Simulation Architectures

A simulation architecture is designed by instantiating the elements of the Simulation Information Model (SIM). Such elements are instances of the following types:
SimulationComponent, representing a simulator or a simulation model. A SIM can contain SimulationComponents representing different tools (simulators) and/or simulation models that will be executed using some software tool.
Filter, representing a component that can perform some syntactic transformation on data. Its role is to transform data from one format to another so that two simulators can actually exchange information even if they use different data representation.
Activator, representing a component that can control the flow of execution within a simulation architecture.
Input/Output, representing a component providing (user-defined) input/output data used/produced by one or more SimulationComponent.

Each component comprises input and output *Gates*, which are in turn linked by means of *SimulationLinks*. In particular, an input gate is a gate through which a component receives data, while an output gate is a gate through which a component sends data.

2.4 Semantics of Simulation Architectures

The semantics of the simulation architecture is given in term of High Level Petri Nets (HLPN) [5] in which one can associate values with tokens and actions with transitions. Each component is associated with a HLPN and thus the simulation architecture results in a HLPN obtained by composing the different HLPN associated with the components therein. For instance a stand-alone simulator† having n input gates and m output gates is modeled by a single transition having n input places and m output places, as shown in Figure 2. The marking of each input place represents the availability of the data on the corresponding input gate of the simulator. The value associated with each token represents the data needed by the simulator. Thus, the firing of the transition represents the execution of the simulator that will mark the output places to represent that the result of the simulation has been produced. As a second example Figure 2 shows an activator named two-way selector, which can receive inputs from two different sources and provides as output one of the two inputs depending on the selection condition.

† A stand-alone simulator requires all input data to be available before starting the simulation. Once started, no data exchange occurs until the simulation ends. When the simulation ends the output data is available.

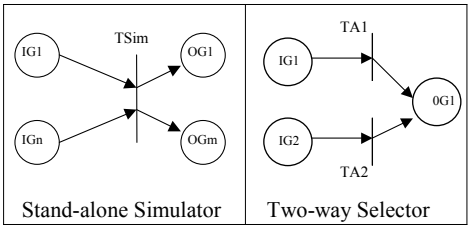

Fig. 2. The HLPN representation of two elements

2.5 Executing Distributed Simulations

In order to run a simulation one has to set up an experiment, that is to define the input data needed by the different simulators. Once the experiment has been set up, the execution is carried out by a tool named Distributed Simulation Controller (DSC), which is in charge of determining the control flow by executing the HLPN associated with the simulation architecture and managing data exchange among simulators. When a simulator produces a new data it notifies DSC, which, in turn, determines which simulators should receive it.

Communication between DSC and the simulators is implemented using CORBA[6]. The motivations behind such choice are: (1) CORBA is a standard middleware defined by the OMG and many implementations are available (Some of them are freeware or even open-source); (2) CORBA supports many programming languages and operating systems and thus it is very effective when one needs to integrate components written in different programming languages and/or working with different operating systems.

The CORBA Interface Definition Language (IDL) is used to describe the interfaces of the objects connected to the CORBA Object Request Broker. Thus ASIA requires each simulator to support two IDL interfaces, which we refer to as the *simulation control interface*, allowing DSC to drive a simulator by calling the methods to initialize, activate, suspend, restart and terminate the execution, and the *data exchange interface*, allowing DSC to handle data exchange among simulators.

Thus, the integration of a simulator requires the development of an *ad hoc* adaptor supporting on one side the two IDL interfaces and on the other side the simulator API. Also DSC has an IDL interface, in order to allow simulators to notify that they have produced new data and/or they have ended a simulation. The interested reader can refer to [7].

3 An Overview of the High Level Architecture

The High Level Architecture (HLA) [1,8] provides a framework to describe simulation applications, to facilitate interoperability among simulations and to promote reuse of simulations and their components. HLA describes simulations in terms of federations of federates, where a federation is a simulation system composed

of two or more simulator federates communicating through the Run-Time Infrastructure (RTI).

HLA requires federations and federates to be described by an object model that identifies the data exchanged at runtime. This is accomplished by the HLA Object Model Template (OMT), which defines the object classes (objects) and the interaction classes (interactions). Objects represent the data structures shared by more than one federate, while interactions represent data sent from one federate to others. The OMT defines the format of the following key models:

Federation Object Model (FOM), providing the specification for data exchange among federates. It describes the objects, attributes, and interactions used across a federation.

Simulation Object Model (SOM), describing the federate in terms of objects, attributes, and interactions that it can offer to a federation. The SOM describes the capabilities of a federate to exchange information as part of a federation.

Management Object Model (MOM), identifying the objects and the interactions used by the RTI to manage the federation state.

In order to ensure proper interaction of federates in a federation and to describe the responsibilities of federates and federations, HLA defines a set of rules, which are divided into two groups one for federations and the other for federates.

The functional interfaces between federates and the RTI is defined by means of the Interface Specification. Federates do not talk to each other directly; the communication between federates is managed by the RTI and is based on the publish/subscribe mechanism. The RTI takes care of communication between the simulators and provides the required services to the simulation systems. It let federates join/leave the federation, declare their intent to publish/subscribe information, etc. In order to allow each federate to implement the described functionalities the RTI provides to every federate a set of API (Application Programming Interfaces) [1]. There are two main interfaces: *RTIambassador* and *FederateAmbassador*. Communication between federates and the RTI is based on RTIambassador and FederateAmbassador interfaces. RTIambassador is used by every federate to communicate with the RTI, while FederateAmbassador is used by the RTI to communicate with federates. Finally, RTI supports federations through services such as the time management service [9] (to correctly reproduce the temporal aspects of the modeled world).

4 An Example of Use: Flexible Manufacturing System

A *Flexible Manufacturing System* (FMS) is composed of several machines connected by means of a transport system. The transport system carries the raw parts to the machines on pallets where they are processed. Once the machines have finished their job the parts are moved back to the load station where they are unloaded. Moreover the machines use a tool-room as a repository for the tools they actually need in order to properly work the raw parts. A computer controls the machines and the transport system [10].

Using a distributed simulation in the FMS field provides some advantages. It is possible, for example, to solve the problem of confidentiality in the context of a

supply-chain with external supplier. Moreover, a distributed simulation provides the possibility of simulating multiple levels of manufacturing systems at different degrees of resolution, creating an array of low-cost simulation models that can be integrated into larger models [11].

We used a simplified FMS to compare ASIA and HLA. The system consists of two machines working the parts on the pallet; a load/unload station that loads (unload) the pallets onto (from) the buffer; a tool room that stores all the tools used by the machines; and a buffer that can hold worked pallets and pallets that need to be worked.

4.1 FMS Simulation Using ASIA

According to the ASIA development process one has to create the (Simulation) Information Model (IM), design the System/Simulation Architecture and implement or adapt the simulators. The IM defines the components needed to model the FMS, which are Load/Unload, Buffer, Machine A, Machine B, Tool Room (see Figure 3) and the standard components such as Activator, Input and Output. The System/Simulation Architecture allows one to instantiate and compose the elements of the IM.

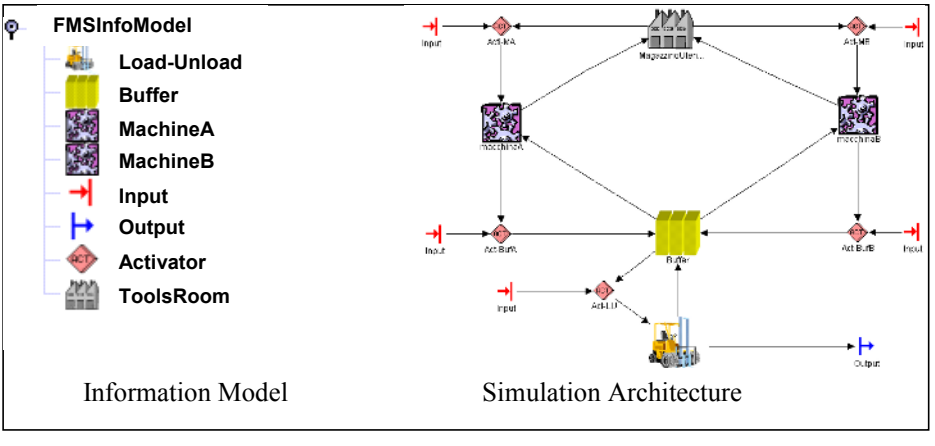

Fig. 3. The FMS and ASIA description

Each simulator sends and receives messages to other simulators according to the Simulation Architecture. Messages represent requests for loading/unloading a pallet or for getting/putting back a specific tool from/to the tool room and so on. Ten different messages are required to properly model the behavior of the whole system. Activators (two-way selectors) are used to provide either the initial user-defined input data or the messages coming from other simulation components.

The simulators have been written in Java and have been extended to support the required IDL interfaces. Each simulator is composed of three different parts: the actual simulator, the CORBA server and the adaptor, which implements the IDL

interfaces, as shown in Figure 4a. The CORBA server is a Java program that initializes and registers the adaptor in the CORBA Naming service (used to identify the objects plugged onto the ORB). The adaptor receives/sends messages from/to ASIA DSC according to the IDL interfaces and takes care of receiving/sending the appropriate messages to the actual Simulator. All components have been hand written but in principle both the CORBA Server and the skeleton of the adapter can be automatically generated.

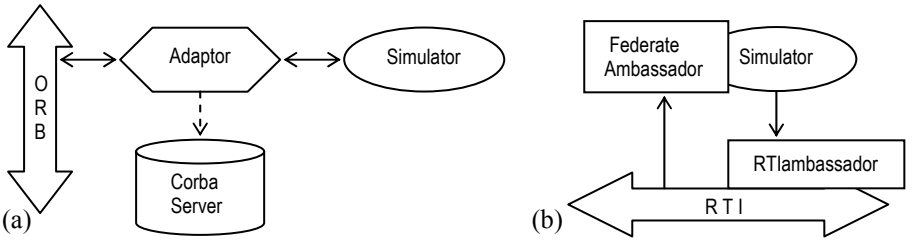

Fig. 4. The ASIA and the HLA run time structures

4.2 FMS Using HLA

In what follows we describe how the FMS has been simulated with HLA, using the same simulators of the previous example and an IEEE 1516 compliant RTI.

HLA requires to define a federation in which each simulator represents a federate. Moreover, it is necessary to define the FOM for the whole federation, describing the classes used by the federates and one SOM for each federate, describing its capabilities. The communication between the simulators has been defined using interaction classes only, and more specifically one class for each message identified in the previous sub-section. As a result the FOM contains ten different interaction classes.

Every simulator has been extended in order to become a federate. This required writing code for implementing both the initialization of the distributed simulation (federation creation, simulators joining the federation, simulators publishing/ subscribing interactions) and the handling of data for each simulator (sending/ receiving interaction to/from RTI). In particular, one has to implement the FederateAmbassador in order to create (if not already existent) and to join the federation and to publish/subscribe to the interaction classes of interest. This is done using the methods provided by the RTIAmbassador, which comes with the RTI implementation. Moreover the user has to implement the method receiveInteraction() in the FederateAmbassador to let the RTI notify the federate of any interaction to which it has subscribed, while for sending an interaction the federate has to call the method sendInteraction() of the RTIAmbassador. Figure 4b shows the data exchange of the HLA-based implementation.

5 Comparison Between ASIA and HLA

Both ASIA and HLA address the problem of integrating simulators to allow one to execute distributed simulations. However, there are many differences and complementarities between the two approaches. First of all ASIA provides an integrated environment in which one can design systems and simulations beside providing support for executing a distributed simulation, while HLA focuses mainly on the latter problem. Thus, ASIA provides an approach having a higher level of abstraction with respect to HLA. In fact HLA can be integrated in ASIA in order to allow execution of simulation architectures. This point is discussed in the next section.

When looking at the way in which distributed simulations are handled in the two approaches one can notice that ASIA provides a static view of the simulators participating at a distributed simulation, while HLA is based on a dynamic view. In other words in ASIA one has to know before starting the distributed simulation how many simulators will cooperate and how they are interconnected, while HLA allows simulators to join and leave a federation during the execution of a distributed simulation. However, in many domains the static approach provided by ASIA is sufficient to model even complex systems. For example, a FMS is usually designed in a static way, that is it is necessary to determine how many (and what kind of) machines will compose the system.

The dynamic approach used by HLA reflects the particular application domain for which HLA was initially developed that is military simulations, where a dynamic view of the system is necessary. However, in many non-military domains a dynamic approach is not required and sometimes may also be counterproductive being more difficult to handle.

Another difference between ASIA and HLA is in the way in which communication among simulators is handled. In ASIA, when designing the simulation architecture one has to statically determine how simulators are interconnected, while in HLA simulators are implicitly connected by using a publish/subscribe mechanism. Such difference is a consequence of the different ways in which systems are described. In other words since HLA allows simulators to join and leave a federation at run-time, the only way to handle communication is by using the publish/subscribe mechanism, while ASIA can make use of point-to-point communication since the different simulators and their role are known before starting the simulation. Moreover, HLA provides two different ways for simulators to exchange data: shared objects and interactions, while ASIA provides only message passing. It is well known that inter-process communication can be based either on shared memory or on message passing. These two mechanisms are “computationally equivalent” although the shared memory paradigm is easier to use for programmers but requires a more complex infrastructure (e.g., CORBA, RMI) while message passing is more complex to use but it is simpler to support. Thus, HLA choice to support both of them does not provide any functionality that could not be obtained by using message passing.

In conclusion, we claim that the ASIA approach is more abstract than HLA and that ASIA and HLA can be integrated by using HLA as the communication infrastructure used to make simulators communicate.

6 Integration Between ASIA and HLA

We are currently working to integrate HLA within ASIA. In fact, as stated before, HLA can be used as the communication infrastructure of ASIA instead of CORBA. Notice that the first phases of the development process are not affected by the choice of using HLA, that is both the information model and the system/simulation architecture do not depend on which technology is used for integrating simulators. The integration requires to modify the role of ASIA Distributed Simulation Controller. When using HLA it behaves like a monitor allowing users to keep track of the status of the simulation.

It must be noticed that the integration does not modify the way in which a distributed simulation is designed, that is by statically defining the simulators and their interactions. Thus, the main limitation to the integration is represented by the fact that it is not possible to use ASIA whenever the system to be simulated requires that simulators (federates) can be added/removed at run-time. However, many application domains, for which simulation is very important, do not require such possibility. In the sequel we sketch the main problems and solutions for integrating ASIA and HLA.

HLA requires the user to provide a FOM for the whole federation and a SOM for each federate. The FOM declares the interaction classes and the object classes, which describe the way in which the different simulators interact. The former mechanism is similar to the ASIA approach in which interactions among simulators are expressed in terms of data flowing from one simulator to another. Thus, a system architecture provides the information necessary to derive the corresponding FOM in which only interaction classes are used. Also the SOM can be derived starting from the information provided by the system architecture. However, it must be noticed that both the FOM and the SOM are “conceptual entities”, that is what is actually implemented is an XML file containing the information represented by the FOM and the SOM. Then the FederateAmbassador needs to be implemented (see Sec. 4) to allow a simulator to be integrated using HLA. This step is similar to the development of the ASIA adaptor and can be carried out in the same way: we can automatically derive its skeleton, while the part representing the “business logics” needs to be written “by hand”.

When executing a distributed simulation using HLA, the ASIA DSC plays a different role with respect to the one it has when using CORBA. In fact, HLA takes care of all the communication aspects that are handled by DSC. However, DSC can still be used to monitor the interactions among simulators. This is done by introducing a component called DSC Monitor that is viewed by HLA as another simulator. The difference between a real simulator and the DSC Monitor is that the former is expected to receive and to send data, while the latter will only receive data. This is done by having the DSC Monitor subscribe to all interactions (objects updates) that occur during a distributed simulation.

There are some open points that are currently investigated. More specifically we still have to address the possibility of deriving from a system architecture a FOM (and SOMs) in which object classes are used. Secondly, we need to investigate how already existing HLA compliant simulators (i.e., the simulators coming with an

already developed SOM and FederateAmbassador) can be represented within the ASIA framework.

7 Related Works

The notations defined in ASIA are used to define the architecture of a system or of an integrated simulation and therefore can be viewed as an Architecture Description Language (ADL). Many ADLs have been defined [12–14] but none of them takes into account the specific needs required when dealing with simulation.

Several works have been done on the problem of integrating simulators. Some of them were domain-specific such as the CIM Framework architecture [15], in the context of semi-conductor environment, or [16, 17], which concern the introduction of data standard or language definition to describe simulation models.

Finally, several works concerning different aspects of HLA are worth to be mentioned. First of all there are several tools that support the development of HLA-based distributed simulation such as Visual OMT [18] or OMDT Pro [19] that can be used to develop the Federate Object Model. Some other tools claim to support the entire development process such as STAGE [20], which is devoted mainly to military applications, or FedDirector [19], which allows one to monitor and control a distributed simulation at run-time. Finally, Calytrix Symplicity [21] provides support for designing and implementing an HLA-based distributed simulation. However, all these tools either do not support the "more abstract" phases of simulation design or are very tied to HLA technology, that is they require a deep knowledge of HLA. Instead, our approach tries to hide as much as possible the technical aspects of the technology used to make simulators interact allowing users to focus on the modeling aspects of their systems.

8 Conclusions

This paper presented an approach for designing and executing distributed simulations referred to as the ASIA approach, which is the result of an on-going effort started within the ASIA ESPRIT project. Currently the approach is supported by a set of tools, some of which are not yet fully implemented, allowing users to define the components needed in their own domain, to instantiate them and to execute them in an integrated way. The approach was initially meant to be based on CORBA and is now extended in order to allow users to choose between CORBA and HLA. The future work will mainly be devoted to automatize as much as possible the development process and to enrich the existing tools in order to provide full support for the automatized process.

References

1. Kuhl, F., Weatherly, R., Dahmann, J.: Creating computer simulation systems – An introduction to the high level architecture. Prentice Hall PTR (2000)
2. http://www.ieee.org
3. Baresi, L.: Formal customization of graphical notations. PhD. Thesis, Dipartimento di Elettronica e Informazione – Politecnico di Milano (1997)
4. Baresi, L., Coen-Porisini, A.: An approach for designing and enacting distributed simulation environments. In: International Conference on Software: Theory and Practice, Beijing, China. (2000) 25–28
5. Jensen, K.: Coloured petri nets: Basic concepts, analysis methods and practical use. Analysis Methods, Monographs in Theoretical Computer Science, Springer-Verlag (1997)
6. http://www.omg.org/
7. Coen-Porisini, A.: Using CORBA for integrating heterogeneous simulators. In: 14th International Conference on Software and Systems Engineering and their Applications, Paris. (2001)
8. IEEE 1516.1: Standard for modeling and simulation (M&S) high level architecture (2000)
9. Fujimoto, R.M.: Time management in the high level architecture. Simulation **71**(6) (1998)
10. Upton, D.M.: A flexible structure for computer-controlled manufacturing systems. Manufacturing Review **5**(1) (1992) 58–74
11. McLean, C., Riddick, F.: The IMS mission architecture for distributed manufacturing simulation. In: Proceedings of the 2000 Winter Simulation Conference. 1538–1548
12. Allen, R.: A formal approach to software architecture. PhD. Thesis, School of Computer Science, Carnegie Mellon University (1997)
13. Magee, J., Dulay, N., Eisenbach, S., Kramer, J.: Specifying distributed software architectures. In: Proceedings the 5th European Software Engineering Conference. LNCS Vol. 989. Springer-Verlag (1995) 137—153
14. Luckham, D.: Rapide: A language and toolset for causal event modeling of distributed system architectures. In: Proceedings of the 2nd International Conference on Worldwide Computing and Its Applications. LNCS, Vol. 1368. Springer-Verlag (1998) 88–103
15. The CIM framework architecture guide 1.0. http://www.sematech.org/
16. ISO (International Organisation for Standardization): Industrial automation systems and integration – Product data representation and exchange (1994)
17. IEEE standard for distributed interactive simulation – application protocols. IEEE standard, 1278.1 (1995)
18. http://www.pitch.se/visualomt1516
19. http://www.aegistg.com/labcut/Labworkscut.html
20. http://www.engenuitytech.com/products/STAGE/index.shtml
21. http://simplicity.calytrix.com/calytrix/simplicity_pages/index.html