An agent based architecture for constructing Interactive Simulation Systems

Zhao, Z.

Publication date
2004

Chapter 2

An agent based component architecture

De-coupling application specific control from the functionality of a system is essential to improve the reusability and flexibility of its constituent components. Interactive Simulation System Conductor (ISS-Conductor) is an agent based architecture proposed to complement existing middlewares*.

2.1 Introduction

Separating application specific control from the basic data level operations requires mechanisms for both describing and orchestrating the high-level system behaviour. Data flow is the most popular mechanism to model system level interactions, e.g. in [16, 113]. Basically, there are two paradigms to orchestrate system behaviour: using a dedicated interaction co-ordinator, e.g. in workflow management systems [60], to schedule the control events between system modules, or incorporating each module as a standard machinery, which is able to autonomously interpret interaction constraints. The first paradigm has been successfully applied in a number of applications [114, 115]. However, in the case of ISSs, where the user drives the system interactions, such paradigm shows a number of shortcomings. First, the paradigm mostly deals with data flow based dependencies; to support sophisticated control for human-in-the-loop interactions, not only a suitable description mechanism is demanded, but also the monitoring on run-time states in each module is required for interpreting the system interaction constraints. When the number of system modules increases, the centralised control paradigm faces fault tolerance and scalability problems. Second, using a separate co-ordinator to orchestrate the system behaviour on the one hand can reduce the requirements on the code incorporation of simulation

*Parts of this chapter have been published in *Z. Zhao, R. G. Belleman, G. D. van Albada and P. M. A. Sloot.* "System integration for interactive simulation systems using intelligent agents", in the Proceedings of the 7th annual conference of the Advanced School for Computing and Imaging, May 2001. An extended version has also been submitted to international journal Concurrency: Practice and Experience.

modules, but on the other hand that will also introduce dependencies between the co-ordinator and the control interface of the system modules. Unlike from the work presented in [87,116], we propose a novel architecture based on the second paradigm.

The goal of ISS-Conductor is to provide a layered architecture for encapsulating the functionality and the run-time control of ISS modules, and for rapid prototyping of a system. The control of a system as complex as an ISS requires certain intelligence. Agent technologies provide a suitable approach to include control intelligence with the behaviour of a set of operations, therefore, we use them to interface the services provided by underlying ISS middleware, and to implement the orchestration of the system.
This chapter is organised as follows. First, we briefly introduce the ISS-Conductor architecture, and then describe its deployment in prototyping interactive simulation systems. Finally, we discuss the requirements and challenges of the development.

2.2 Interactive Simulation System Conductor

2.2.1 Modules as reusable components

In order to make the reuse of legacy ISSs really efficient and worthwhile, a degree of granularity for the components to be reused must be carefully chosen. The smallest components that could reasonably be considered are the individual routines, which would be packaged into a reusable library. Such libraries might be similar to existing mathematical and visualisation libraries, and would lose much of the sophistication of the carefully crafted simulation and visualisation modules, as the essence of the modules often lies in the interaction between the routines. The largest component that could be considered for reuse is the actual ISS as a whole. From the preceding description of its structure, it will be clear that an ISS usually cannot be adapted to a new application domain, or even to a new execution environment, without a major overhaul. A better granularity is at the module level: the principal modules of an ISS, e.g. simulation or interactive visualisation, are envisioned as basic reusable units for constructing ISSs. These units are encapsulated as *components*, in which the computing core from the principal module, e.g. the time-stepping routines in the simulator, would remain nearly unchanged.

2.2.2 Basic architecture

Originally, in designing ISS-Conductor, an approach was chosen where a relatively small control and I/O module would be added to a largely unchanged legacy code, with the aim of adapting the module's behaviour to the requirements of an ISS [4,5]. By keeping these foreign modules apart from the legacy control routines we could localise the application specific control and keep the major part of the legacy code

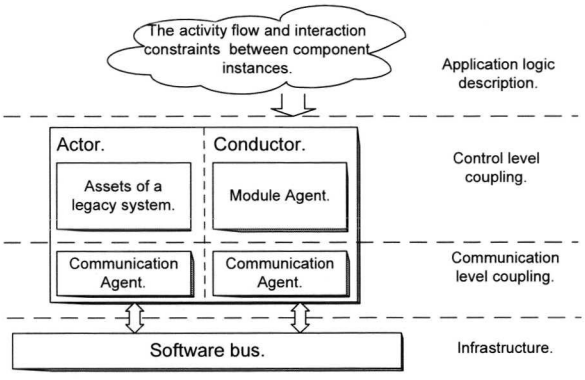

Figure 2.1: Basic architecture of an ISS-Conductor component.

unchanged. However, this approach allowed only limited control of the system behaviour. Therefore, the control routines of the legacy systems were also modified and so that they can be controlled by a foreign module. An ISS-Conductor component thus consists of two primary parts: an *Actor* for encapsulating the functionality of a legacy system and a *Conductor* for controlling the run-time activities of the *Actor*. The run-time integration between component instances is through a software bus.
Inside a component, both the *Actor* and the *Conductor* have a *Communication Agent (ComA)* which provides a uniform interface to exchange information with the software bus. In the *Actor*, the ComA wraps the computing core and data structures in the legacy code and provides an interface for remote access and invocation. In the *Conductor*, a control agent called a *Module Agent (MA)* provides the control intelligence for the behaviour of the component in a specific application context. Fig. 2.1 depicts the basic architecture.

2.3 Agent based design

2.3.1 Agent definition

ComAs are designed using a reactive architecture, as shown in the left part of Fig. 2.2. In the reflex architecture, sensors and effectors are interfaces for the agent to exchange information with the external world. The sensors listen to the external world, generate tasks for the recognised events and pass them to the task interpreter. The interpreter finds proper actions for the task in a task-action lookup table. Finally, the effectors carry out the actions. An important reason that we start with this architecture is because it is simple and extensible. For instance, when the task-action lookup table and the interpreter are complemented with a reasoning engine, the intelligence for action selection and execution will be immediately improved. Using it, agents can thus be constructed incrementally.

In the Actor, a ComA wraps the legacy system and interfaces it to the underlying communication middleware to realise the information exchange with the other ComAs. The functional components in the legacy systems are incorporated as activities in the ComA, and their associated control data are represented as data objects which can be accessed and manipulated remotely by the other components via the software bus framework. Since ComAs do not require sophisticated reasoning mechanisms for controlling their actions, they either pass the events observed from the external world to MAs or directly carry out the instructions sent from MAs. Fig. 2.3 shows an example of ComA in an Actor.

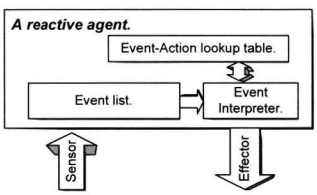

Figure 2.2: A simple agent kernel.

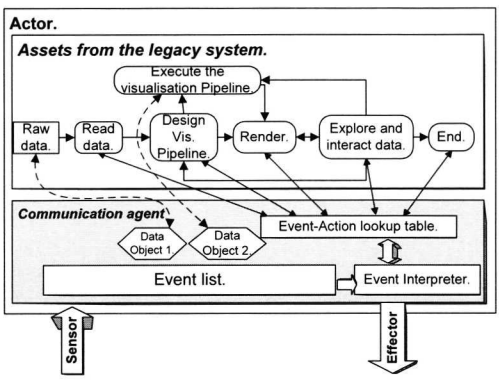

Figure 2.3: An Actor and its ComA.

An MA incorporates aspects of a deliberative agent. It employs a world model to track the changes of the external world and to obtain rational perceptions on the actual execution states of the other modules. Using the information supplied by the world model and the knowledge represented in the knowledge base, a reasoning engine is then used to find proper actions for the *Actor* to perform. Fig. 2.4 depicts a basic MA architecture.

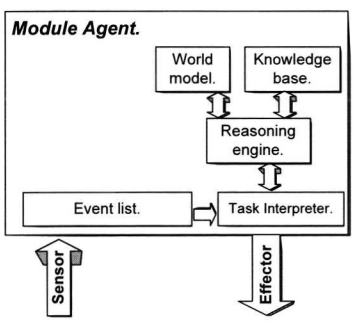

Figure 2.4: The basic architecture of MA.

2.3.2 Activity control

Inside a component, the action execution and the reasoning process are carried out by the Actor and the Conductor respectively. The capability of the simulation or visualisation system is described as knowledge of the MA in the Conductor, and its implementation is located in the Actor as a collection of actions and data objects. To take part in an ISS, each MA also receives a description of the interaction constraints and dependencies with the other components. Using these two types of knowledge, together with the information from the world model, an MA can then take a decision on the actions that the Actor should perform. After performing an action, the Actor reports the execution status to its MA while updates its world model. To keep the world model up to date, MAs also have to exchange their perceptions. The data objects that represent the control data or states in the legacy simulation or visualisation systems are maintained by the ComA in the Actor, and most of these objects only need to be accessible and shared among Actors. Depending on the level of detail of the knowledge representation, some of those objects are also needed by the Conductor of the component.

2.3.3 Performance considerations

As mentioned, adaptability and flexibility are often traded against performance in ISSs. In the ISS-Conductor architecture, performance is considered in three ways. First of all, the *Actor* retains the well-tuned computational kernels of the legacy implementation of the simulation and visualisation, and the interface for realising the parallelisation, such as MPI [119] or PVM [120]. Secondly, the implementation of ComA employs dedicated middleware for data distribution, which not only offers the necessary flexibility for adapting the data distribution but also ensures the necessary performance. Finally, the separation of functionality (Actor) and control (Conductor) allows the computation and control to be parallelised.

2.4 Constructing interactive simulation systems

2.4.1 Composing an ISS

Using the ISS-Conductor methodology, an ISS can then be developed by selecting the proper components and composing run-time interaction constraints and dependencies between them. The customisation of the component activities is achieved through the knowledge base level of MAs. In the knowledge base of MA, the specification of the component functionality, also called the *capability* of the component, is used for qualification checks when the component is to be included in an ISS. The output of a composition, also called a *story*, is a specification of the constraints governing the interaction among the component instances. At run time, each Component instance is

assigned a *role*[†] with a unique name. MAs orchestrate the system behaviour according to a *story*. The basic development paradigm is shown in Fig. 2.5.

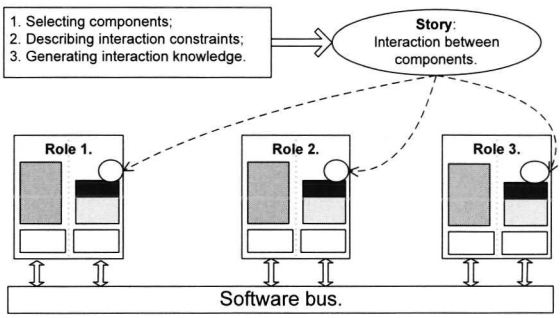

Figure 2.5: A basic paradigm of assembling ISS-Conductor based components.

Using such an approach, the development for an ISS is shifted from realising basic interconnections to describing the high-level interaction constraints. It also introduces two important questions. The first one is how to describe the component capability and the interaction stories and the second one is how to do the qualification check for components. We will leave these questions for the next two chapters.

2.4.2 Run-time framework

The run-time integration between components is through a *software-bus-* like framework, which is normally the infrastructure provided by the middleware that ComAs reside on. The middleware has to provide a number of services demanded by the ComAs: distributed object access, message passing, and data distribution. These services are available in many object oriented middlewares, such as CORBA and HLA. Currently, HLA is used, but the design is not necessarily bound to it. Dependencies on the underlying middleware can be localised in ComAs so that the whole implementation is portable.
Components are directly plugged into the software bus and no intermediate assembly components, like the *containers* in the CORBA Component Model (CCM) [121], are needed. One of the reasons is that ISS-Conductor components are derived from legacy simulation and visualisation systems; a component not only encapsulates the computational routines and data structures of a legacy system but also the necessary logical dependencies between them and the conditions for accessing and invoking them. This integration paradigm shifts the development focus from the assembly of small size computational routines to the specification of constraints on the high-level system behaviour between component instances. These constraints are described as knowledge in the Module Agents.

[†]In the context of this thesis, a *role* refers to an identifier of a component instance. A *story* is more than a static description of a system behaviour; it refers to the description of interaction constraints.

2.5 Summary

In this chapter we have described the basic architecture of ISS-Conductor. As we have stated in the beginning, the goal of ISS-Conductor is to provide a layered framework for constructing interactive simulation systems, so that the logic control of system behaviour can be separated from the basic coupling details. To approach the goal,

1. ISS-Conductor proposes an autonomous machinery which can wrap the functional units of a legacy simulation or visualisation program, and provide an abstract interface for composing high level interaction among them.

2. It provides an agent framework to encapsulate the computing kernel and the control intelligence of a legacy system, and to incorporate them as a reusable component.

3. At run time, the agents couple the components in a layered scheme: Communication Agents for basic interoperability between components, and Module Agents for controlling system behaviour.

Compared to the solution proposed by Radeski et al., [88] or the one realised in the SIMULTAAN Simulation Architecture (SSA) [91], ISS-Conductor takes a further step: it employs agent technologies to enhance HLA federates; the interconnection interfaces and the interaction control are encapsulated in different agents. The logic structure and the run-time flow of data and activities between component instances are described as knowledge in the agents. No special centralised co-ordinator is employed to control the system behaviour. Using software component and agent technologies in constructing ISSs is a novel approach. Compared with the architecture proposed by the CCA and Bond, ISS-Conductor takes the advantages of available advanced middleware. It views the principal ISS module as components, and focuses on the activity control between them.