An Object Infrastructure for Computational Steering of Distributed Simulations

Rajeev Muralidhar and Manish Parashar
Department of Electrical Engineering and CAIP Center,
94 Brett Road, Piscataway, NJ - 08854.
Email : {rajeevdm,parashar}@caip.rutgers.edu

Abstract

This paper presents a brief overview of a framework for the interactive steering of distributed applications that addresses three key issues: (1) Definition of Interaction Objects that provide sensors and actuators for interrogation and control. These objects encapsulate existing computation data-structures and can be distributed (spanning many processors) and dynamic (be created, deleted, or migrated to another processor). (2) Definition of a control network of interaction agents that enable the interactive steering of distributed interaction objects. (3) Definition of an Interaction Gateway that provides a proxy to the entire application and can be accessed via a web server. This research is part of an ongoing effort to develop a web-based computational collaboratory that enables geographically distributed scientists/engineers to collaboratively monitor, and control distributed applications.

1. Introduction

Simulations are playing an increasingly critical role in all areas of science and engineering. As the complexity and computational costs of these simulations grows, it has become important for the scientists and engineers to be able to monitor the progress of these simulations, and to control or steer them at runtime. Furthermore, the increased complexity and multi-disciplinary nature of these simulations necessitates a collaborative effort among multiple, usually geographically distributed scientists/engineers. As a result, collaboration-enabling tools have become critical for transforming simulations into true research modalities.

Enabling seamless interaction and steering of high-performance parallel/distributed applications presents many challenges. A key issue is the definition and deployment of interaction objects with sensors and actuators that must be co-located with the computational objects and that are capable of monitoring and controlling the application. Defining these interfaces in a generic manner and deploying them in distributed environments can be non-trivial, as computational objects can span multiple processors and address spaces. The problem is further compounded in the case of adaptive applications (e.g. simulations on adaptive meshes) where computational objects can be created, deleted, modified and redistributed on the fly. Another issue is the design of a control network that interconnects these sensor and actuators so that commands and requests can be routed to the appropriate set of computational objects, and information returned can be collated and coherently presented. Finally, the steering interfaces presented by the application need to be exported so that they can be easily accessed by a group of collaborating users to monitor, analyze, and control the application. The object infrastructure presented here addresses these issues and is part of the DISCOVER computational collaboratory [1].

1.1. The DISCOVER Computational Collaboratory

The overall objective of this research is aimed at developing a web-based interactive computational collaboratory. The current implementation enables geographically distributed clients to use the web to connect to, monitor and steer multiple applications in a collaborative fashion through a dedicated web server. The system supports a 3-tier architecture composed of detachable, browser-enabled thin-clients at the front-end, a network of interaction web servers in the middle, and a control network of interaction agents and interaction objects within the application at the back-end. The application control network enables sensors, actuators and interaction agents to be directly deployed within the application. Interaction agents resident at each computational node register the interaction objects and export their interaction interfaces through an Interaction Gateway. The Interaction Gateway interacts with the external web server and provides a proxy to the entire application. It uses the Java Native Interface (JNI) to create Java proxy objects that mirror the computational objects and allow them to be directly accessed by the interaction web-server using standard

[1] http://www.caip.rutgers.edu/TASSL/Projects/DISCOVER

distributed object interfaces like Java RMI, CORBA, etc.

2. DIOS - An Object framework for Interaction and Steering

The Distributed Interactive Object System (DIOS) is composed of two components: (1) Interaction Objects that encapsulate interaction sensors and actuators, and (2) A Control Network consisting of distributed Interaction Objects and Interaction Agents.

2.1. Sensors/Actuators and Interaction Objects

Interaction objects extend the application's computational objects (data structures used by the application) with steering capabilities by providing abstractions for creating sensors and actuators. Sensors enables the object to be queried while actuators allow it to be steered. Efficient abstractions are essential for converting computational objects to interaction objects especially when the computational objects are distributed and dynamic. In our system, this is achieved by deriving the computational objects from a virtual interaction base class of the Interaction Object library. The derived objects define a set of *Views* that they can provide and a set of *Commands* that they can accept. Interaction agents then export these views and commands to the interaction server using a custom interaction IDL [2]. Interaction objects can be either local to a single computational node, distributed across multiple nodes, or shared between some or all of the nodes. Distributed objects have an additional distribution attribute that describes their layouts. Further, interaction objects can be dynamically created or deleted during application execution, can migrate between computational nodes or modify its distribution by notifying appropriate interaction agents. In the case of applications written in non object-oriented languages such as Fortran, application data structures are first converted into computation objects using C++ wrapper objects. These objects are then transformed to interaction objects as described above.

2.2. The Control Network and Interaction Agents

The control network of interaction agents (Figure **??**) has a hierarchical cellular structure and partitions the processing nodes into a number of interaction cells. The network is composed of (1) Discover Agents on each node, (2) Base Stations for each interaction cell and (3) an Interaction Gateway that connects to the interaction server and provides a proxy to the entire application. The number of nodes per interaction cell is programmable. The cellular control network is automatically configured at run-time using an underlying messaging environment (e.g. MPI) and the available number of processors.

[2] Interface Definition Language

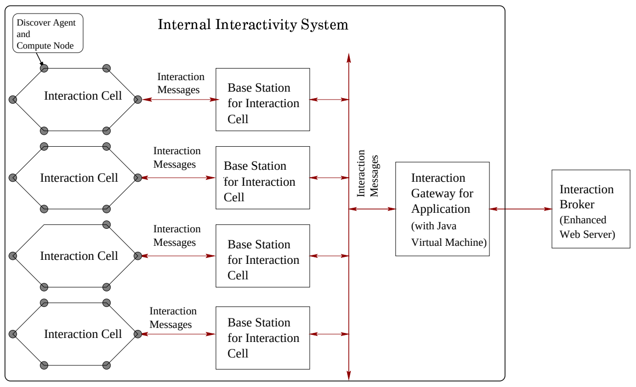
Internal Interactivity System
Discover Agent and Compute Node
Interaction Cell
Interaction Messages
Base Station for Interaction Cell
Interaction Cell
Interaction Messages
Base Station for Interaction Cell
Interaction Cell
Interaction Messages
Base Station for Interaction Cell
Interaction Cell
Interaction Messages
Base Station for Interaction Cell
Interaction Messages
Interaction Gateway for Application (with Java Virtual Machine)
Interaction Broker (Enhanced Web Server)

Figure 1. Control Network of Interaction Agents

Discover Agents present on each node maintain run-time references to all registered interaction objects on that node. Since object references can change dynamically during program execution, the interaction agents ensure that object references are valid and refer to consistent data. Discover Agents, Base Stations and the Gateway each maintain registries of interaction objects registered in their respective domains (node, cell, entire application, respectively). The Gateway is additionally responsible for interfacing with the interaction server, delegating interaction requests to the appropriate interaction agents (Discover Agents and/or Base Stations), and collecting their responses. In the case of distributed objects, the Gateway also performs a gather operation for collating the responses arriving from the corresponding nodes. Furthermore, the Gateway uses JNI to create Java mirrors of each registered interaction object. Thus interaction Web Servers can directly access the application's interaction objects (and thus the computational objects) using standard distributed object interfaces like Java RMI.

2.3. Experimentation Evaluation

Initial experiments conducted on a Sun HPC E10000 cluster using upto 16 nodes indicate that (1) End-to-end steering latency varies between a few milliseconds to tens of milliseconds for varying data sizes, which is comparable to other steering systems. (2) Overhead of minimum steering (providing continuous updates of only important steering parameters) is only a fraction of computation time. (3) Overheads of object registration and interaction IDL processing at the interaction agents is a constant factor of the number of objects being registered and the size of the interaction cells being used.