INTERLACE: An Interoperation and Linking Architecture for Computational Engines

Matthew J. Sottile and Allen D. Malony

Department of Computer and Information Science, University of Oregon, Eugene, Oregon 97403-1202

Abstract. To aid in building high-performance computational environments, INTERLACE offers a framework for linking reusable computational engines in a heterogeneous distributed system. The INTERLACE model provides clients with access to computational servers which interface with “wrapped” computational engines. An API is defined for establishing the server interconnection requirements including data organization and movement, and command invocation. This provides an abstraction layer above the server middleware upon which servers and clients may be constructed. The INTERLACE framework has been demonstrated by building a distributed computational environment with MatLab engines.

1 Introduction

Scientific research that involves the analysis of large data sets, the simulation of complex phenomena, or the execution of detailed computational experiments often requires scientists to develop software that runs on high-performance parallel and distributed computer systems. Unfortunately, these systems can be difficult for a scientist to program and use. The software that is developed tends to be specialized and can be hard to integrate with other tools that are commonly found in the scientist's environment. The reuse and linking of software with components from other projects suffers from the lack of a common interface and interoperation model. More accessible computational programming packages such as MatLab [1], IDL [2], and Mathematica [3] are commonly used for small scale computational tasks. The relative simplicity of the programming system and the wide availability of modules of scientific interest makes these packages both productive and desirable. However, there is minimal support for integrating such tools with high-performance environments to utilize specialized hardware or software libraries. Scientists and other users of high-performance environments could benefit from a framework in which they can both easily create programs utilizing reusable, common tools and scale their applications to larger systems.

In this paper, we propose a framework of this type based on the abstraction of components, or tools, as *computational engines*, and their interoperation as interconnected *computational servers*. A computational server is a persistent program that allows *clients* access to functionality and data provided by the computational engines linked to it. Any computational engine that creates and shares data must also be persistent during the application's operation. The server interface provided to clients is abstract, to hide the complexity of the underlying parallel or distributed system. Because the framework supports a computational engine as an abstract component, any tool can be provided as long as its server interface can be created.

2 Architecture

A system view of the INTERLACE[1] framework architecture is shown in Figure 1. The shaded portions of the diagram indicate INTERLACE software components. To make computational engines available to clients through a homogeneous interface, they must be individually wrapped and linked with a computational server. The wrapper acts as a translation and interpretation layer between the INTERLACE environment and the specific engine interface. Given a computational server with a set of available wrapped engines, clients may attach to the public interface of the server and issue data and computation related commands. To provide this layering of interfaces between client, server and engine, INTERLACE specifies two abstract interfaces. The first defines the interface between the wrapper and computational server, while the second defines the interface and protocols for linking individual computational servers and clients.

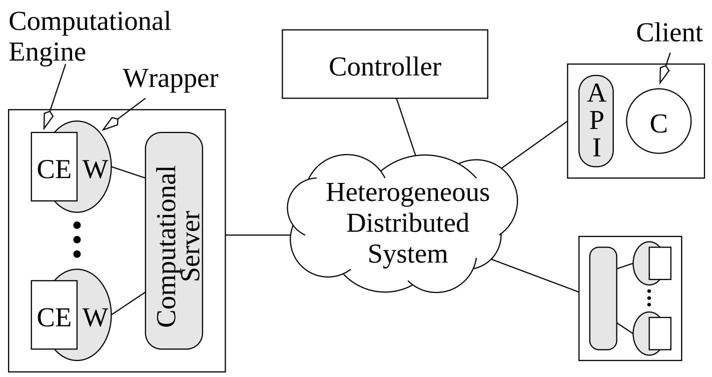

Fig. 1. INTERLACE System Architecture

Before one can assemble an INTERLACE-based environment, the individual servers and wrappers must be created. The wrapping of computational engines is dictated by the engines themselves and is not part of the INTERLACE specification. The interface between the wrapper and the computational server exists in

[1] A complete description of INTERLACE is available as Technical Report CIS-TR 99-06 from the Department of Computer and Information Science at the University of Oregon.

two parts. First, there must be methods available for calling computational functions within the server. Second, if the server will import or export data to other members of an INTERLACE environment, there must be methods for moving data. These functions are then used in the server within the implementation of methods visible to client applications. INTERLACE defines this API as a small set of functions to receive message-based commands, move data, and control the state of the server. The messages and data movement requests are performed by calling appropriate methods in the wrapper, thus hiding the wrapper specific APIs from the client applications.

Once INTERLACE servers are implemented, the environment requires that they are connected in some manner. The method and implementation of such interconnection schemes is independent of INTERLACE. However, the implementation method must provide some form of cross-platform data movement and remote method invocation. The API that the servers present to clients is specified in an abstract manner that does not impose strict requirements on the middleware choice. Current implementations of MatLab-based INTERLACE environments exist in two forms - one built with High Performance C++ (HPC++) [4] for high-performance environments, and one built with the Java Remote Method Invocation (RMI) [5] API for high portability. Servers, data, and computational services are located by a simple identifier, which is mapped to the appropriate URL or other middleware-specific location identifier. Such mappings are maintained at a central location known as the *controller*. All servers and clients are provided the location of the controller, and are then able to dynamically locate and establish connections to each other.

3 Application of INTERLACE with MatLab

We prototyped the INTERLACE framework using servers with MatLab-based engines. The MatLab engine interface provides simple libraries for allowing access to data and functions. Computations are performed by issuing text strings containing MatLab commands to the engine. Data can be moved between the server and engine through both the wrapper and a separate program called from within MatLab, written as a *mex-file*. The mex-file provides an additional MatLab command that allows a MatLab engine access to the INTERLACE environment for moving data or issuing commands to other servers. This also provides means for MatLab to act as a front-end interface to an INTERLACE-based environment.

Our most recent implementations of the INTERLACE middleware layer use Java RMI to increase portability. So that the MatLab frontend could access INTERLACE services, the Java Native Interface (JNI) was used to connect C mex-files and the Java RMI layer. (A similar method was used to connect MatLab using mex-files and HPC++.) At the server interface to the engine wrappers, there is little interpretation since commands are assumed to be basic MatLab string commands. However, a queue of commands can be collected by the server, and when ready, executed in order. Without this support, it would be difficult to create computational scripts that employed multiple servers in an efficient way. For instance, it would allow a user to load large routines (such as an entire MatLab command file, or *.M file*) into a collection of servers, and then let them compute in parallel when ready.

4 Conclusions

The current implementation of an INTERLACE-based system utilizes MatLab engines with either Java or HPC++ middleware. This illustrates the potential of the INTERLACE abstraction above implementation mechanisms. By creating these two implementations, INTERLACE can be shown in high-performance environments that best suit HPC++, and heterogeneous systems that require Java's portability. Continuing work will involve providing wrapper and server interfaces for tools such as IDL and Mathematica. Our primary research interest is in the access and use of heterogeneous engines with a common, homogeneous INTERLACE abstraction layer. Additional research includes wrapper generation techniques and middleware utilization. An important future step will be the ability to integrate two INTERLACE environments implemented with different middleware, such as HPC++ and Java.

Fulfilling the need for a framework for connecting computational engines in a scalable, reusable manner is the central goal of INTERLACE. The abstraction of functionality and data offered by computational engines to the homogeneous INTERLACE API and interconnection scheme provides many advantages. It makes possible dynamic environments in which computational engines may be instantiated as required, while hiding implementation details required to support such behavior. By specifying INTERLACE as an abstract definition of interaction APIs and behaviors, INTERLACE may be implemented using a variety of middleware solutions. Since the computational server framework provided by INTERLACE is an extendible skeleton, the application of this framework to specific tools can be fine-tuned to the user's needs. As a result, INTERLACE is a powerful tool for scientists and others who wish to build a reusable and scalable environment for doing high-performance analysis and research.

References

1. The Math Works, *MATLAB: High Performance Numeric Computation and Visualization Software Reference Guide* Natick, MA. 1992.
2. Research Systems, IDL.
`http://www.rsinc.com/`.
3. Wolfram Research, Mathematica.
`http://www.wolfram.com/`.
4. E. Johnson and D. Gannon, HPC++: Experiments with the parallel standard template library. *Technical Report TR-96-51* Indiana University, Department of Computer Science, December 1996.
5. Javasoft, Java Remote Method Invocation (RMI) Interface.
`http://www.javasoft.com/products/jdk/rmi/index.html`.

This article was processed using the LaTeX macro package with LLNCS style