DOE/ER/40150--1201
JLAB-ACC-96-24

Evaluation of a Server-Client Architecture for Accelerator Modeling and Simulation*

B. A. Bowling, W. Akers, H. Shoaee, W. Watson, J. Van Zeijts, S. Witherspoon

Thomas Jefferson National Accelerator Facility, Newport News, VA 23606 USA

Abstract. Traditional approaches to computational modeling and simulation often utilize a batch method for code execution using file-formatted input/output. This method of code implementation was generally chosen for several factors, including CPU throughput and availability, complexity of the required modeling problem, and presentation of computation results. With the advent of faster computer hardware and the advances in networking and software techniques, other program architectures for accelerator modeling have recently been employed. Jefferson Laboratory has implemented a client/server solution for accelerator beam transport modeling utilizing a query-based I/O. The goal of this code is to provide modeling information for control system applications and to serve as a computation engine for general modeling tasks, such as machine studies. This paper performs a comparison between the batch execution and server/client architectures, focusing on design and implementation issues, performance, and general utility towards accelerator modeling demands.

INTRODUCTION

Traditional approaches to computational modeling and simulation often utilize a batch method for code execution using file-formatted input/output. This method of code implementation was generally chosen for several factors, including CPU throughput and availability, complexity of the required modeling problem, and presentation of computation results. With the advent of faster computer hardware and the advances in networking and software techniques, other program architectures for accelerator modeling have recently been employed. Jefferson Laboratory has implemented a client/server solution for accelerator beam transport modeling utilizing a query-based I/O. The goal of this code is to provide modeling information for control system applications and to serve as a computation engine for general modeling tasks, such as machine studies. This paper performs a comparison between the batch execution and server/client architectures, focusing on design and implementation issues, performance, and general utility towards accelerator modeling demands.

*work supported by US DOE contract# DE-AC05-84ER40150

II. TRADITIONAL ARCHITECTURES

In the last thirty years of simulation and modeling work, a few primary implementation architectures were employed for many codes. The simplest architecture form, viewed from implementation and execution standpoints, are the codes in which the configuration and lattice information are integrated and compiled internally with the algorithm. This implementation form is easy to execute, since the program contains all information required to perform the calculations. However, this approach is highly application-specific, with any changes in configuration requiring direct editing of the source code.

A natural extension to the above is the segregation of the configuration from the actual algorithm. This method is generally implemented using data files which contain configuration, lattice, and operation commands, and is often referred to as decks. Most of the simulations and models which are in use today employ this architecture, and there have been standards developed within the modeling community which describe the format of the data decks, for example the MAD and ZCEDEX formats[1]. The separation of the configuration/operation parameters from the actual executed code allows for the development of generic modeling and simulation applications which can be used at many accelerator sites.

Results of calculations are usually produced in a tabulated output format, often maintained as disk files. This form is convenient in that results for a particular input configuration can be maintained for future review and/or analysis. However, if it is required that computed results be available for other programs, such as machine control processes, then interface codes which operate on computed model/simulation files must be developed.

III. CLIENT/SERVER ARCHITECTURE

In the client/server approach, the application is distributed between at least two processes, the server and the client. The server is the unified source of information or algorithmic calculations, and the client is the process or processes which perform requests to the server in order to obtain the information. In the case of modeling or simulation, the server contains the calculation engine and configuration/lattice information. The client, which is part of an application process, performs queries towards a server, requesting information or calculation results. Both the server and the client require a communication functionality and protocol to be established.

Client/server architectures naturally lead to extensions which enhance performance and functionality. At the simplest level, a single server can service a single client. Adding the capability of multi-user access to the server allows for the simultaneous servicing of multiple clients from one server. Multiple servers can also be implemented, each handling its own set of clients. The use of local area and wide area networks allows distribution of clients and servers on different hardware platforms. Callback mechanisms can be implemented on the server which can provide

automatic update mechanisms to attached clients.

A server for modeling and simulation purposes is basically a batch model architecture with the addition of a high-level event/callback processing manager, communications interface for client transaction, and command message decoding and encoding. Multi-user servers must keep track of requests from multiple clients and prevent inadvertent interactions between clients, as well as preserving the state and integrity of the model or simulation. It is sometimes advantageous to structure the model or simulation code such that static information, like lattice construction and static calculations, be generated at server start-up. Hence, client requests to the server require the minimum of computational processing, thus improving server response.

IV. EXAMPLE CLIENT/SERVER MODEL

An illustration of a client/server architecture is the ARTEMIS (Accelerator Real TimE Modeling Information Server) modeling environment, currently in use at Jefferson Lab for accelerator controls and analysis. ARTEMIS provides various model information (twiss parameters, transfer matrices, etc.) and supporting computations (e.g. quad strength calculation for matching) for all model-driven facilities, including automated beam steering, beam position and energy feedback, and beam diagnostic and optimization procedures [Fig. 1]. Implementation of ARTEMIS as a server allowed for the centralization of the model calculations, which provided a uniform and consistent data source for these and other applications, while eliminating the need for redundant calculations by different application software.

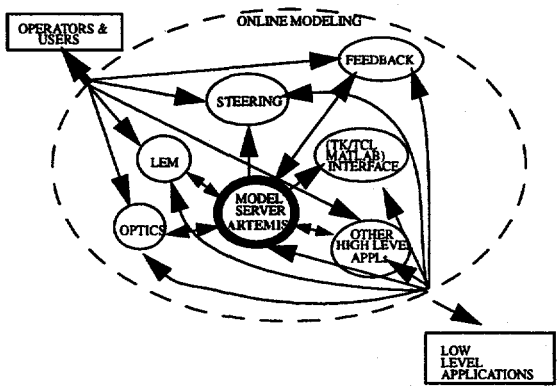

Figure 1: Modeling Environment Overview

ARTEMIS is based on standard second-order transfer matrix calculations, and provides the following functionality:

- Generation of first and second-order transfer functions.
- Provisions for inclusion of higher order models as needs dictate.
- Correct treatment of the acceleration process in the linac cavities, including effects of adiabatic damping and cavity focusing effects.
- Providing lattice and input settings to clients, which has the functionality of an on-line database. This includes hardware information (e.g. list of quardrupoles, BPMs, etc. in a given region, with the ability of providing wild-card input) and element operational settings.
- Model update mechanisms which include user-initiated, periodic, and input event triggered.
- Model updates triggered by an external event.
- User initiated model calculations (on-demand modeling).

ARTEMIS provides the capability for one to instantiate four distinct machine model servers:

- Golden model: includes a set of machine parameters and settings which have been verified and deemed reasonable by an authorized expert.
- Design model: this is a machine model based on the paper design and initial setpoints of the accelerator.
- Current real-time model: this is a machine model representing the current accelerator setpoints; the update mechanism may be event-driven or user initiated.
- Simulation model: used to determine the outcome of *what-if* scenarios as applied to the accelerator lattice.

The implementation of the server section of ARTEMIS is illustrated in figure 2. It consists of several subassemblies: a lattice database, which performs the input-output lattice management for the server, a client communications section, the model engine controller process (GENX), and CAUListener which provides the interface to machine settings. The static lattice information is maintained in a common shared-memory region, which includes all of the element setpoints and pre-computed local transfer matrices. Individual requests from clients invoke the server to apply the appropriate algorithm against the information contained in the common classes.

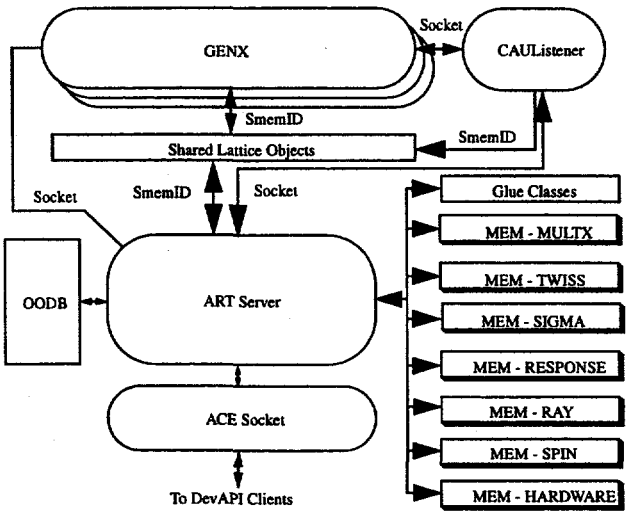

Figure 2: ARTEMIS Server Architecture

The communications and server/event management for ARTEMIS is provided by cdev (control device or common device), which is a C++ library toolkit initially developed at Jefferson Lab. Cdev provides a standard application programming interface (API) to one or more underlying packages, typically control system interfaces. It consists of two layers: the uppermost layer is used directly by applications, and provides an abstraction of the underlying package, and the second layer (service layer) provides the interface to one or more underlying packages, and is implemented as optional loadable libraries.

A server engine and client interface has been integrated into the cdev interface. Based on the ACE C++ wrapper classes for communication of cdev data packets(employing TCP/IP), the interface greatly simplifies the construction of servers and clients. The cdev server interface includes input/output messaging queues, built-in event and monitor mechanisms, and multi-user processing. The client interface contains the server connect/disconnect functions and callback mechanisms which are available to application programs.

The format of a cdev transaction consists of a device name, message, and attribute. The device name represents any logical or virtual device, for instance a name of a magnet or monitor. The message string indicates which operation is desired, and the attribute allows for specification of indentifiers. Cdev also allows the message string to specify application operational parameters, such as which model server to use. The cdev string "model "get element" "device=Quadrupole"" is interpreted as a command to the default model server to retrieve the logical element names for every quadrupole in the server's lattice. Another example is: "IPM1L01 "get betax" "model=DESIGN"", which returns the machine horizontal beta function for the element names IPM1L01, retrieved from the model server DESIGN.

V. EVALUATION

It is evident that implementing a client/server model or simulation requires more code development than the traditional batch architecture. New factors, such as communications and model coherency, must be addressed for the client/server. Additionally, the server must always be available to the end user, thus requiring robust code. Corrupted message commands, run-away calculations, arithmetic errors, etc. must be identified and trapped by the server. In our experience, this has been one of the important factors to deal with, and identifying such problems can be difficult at times. The server also has to operate orthogonally between clients, so the design phase of the server must address this. The ARTEMIS server handles this by performing the computation/access for every message received.

The advantages of the client/server model are apparent on the client end. Once the communications and messaging interface is established, the client application can immediately take advantage of the server. Therefore, the client application is not burdened with the implementation details of the model or simulation. This fact allows for rapid application development.

The client/server architecture also allows existing toolkit applications to use server results. An client application interface to TCL/TK and MATLAB has been developed for use with ARTEMIS, and is extensively used for control and analysis. Applications which require modeling information can be realistically implemented and under test in a matter of hours.

An interface which allows server access using the World-Wide Web has also been implemented. Cdev provides a gateway process which is easily interfaced to CGI applications, which allow for the creation of HTML documents from user-initiated requests. The web-based applications allows the user to retrieve parameters such as transfer matrices and Twiss parameters from any of the executing model servers. The pages are easy to use, and have proven beneficial in many situations.

VI. REFERENCES

[1] G. Morpurgo et al, "*Super-ZCEDEX User's Guide*", CERN Internal Document, 1986.

DISCLAIMER