INTEGRATED ON-LINE ACCELERATOR MODELING AT CEBAF

B. A. Bowling, H. Shoaee, J. Van Zeijts, S. Witherspoon, W. Watson
CEBAF, Newport News, VA 23606 USA

*An on-line accelerator modeling facility is currently under development at CEBAF. The model server, which is integrated with the EPICS control system, provides coupled and 2nd-order matrices for the entire accelerator, and forms the foundation for automated model-based control and diagnostic applications. Four types of machine models are provided, including design, golden or certified, live, and scratch or simulated model. Provisions are also made for the use of multiple lattice modeling programs such as DIMAD, PARMELA, and TLIE. Design and implementation details are discussed.

I. INTRODUCTION

CEBAF is a 4 GEV electron accelerator facility which is in the final stages of commissioning. It consists of two 400 MeV superconducting linacs with a 5 pass beam recirculation system. The facility is capable of simultaneously serving three experimental halls with beams of differing energies.

The goal of delivering high quality beams to experimenters requires the availability of appropriate control, diagnostics, and monitoring functions to direct the complex operation of the accelerator. In addition, the efficient operation of a complex accelerator requires the automation of as many routine machine functions as possible. This would assist in achieving the goal of operators acting as accelerator “pilots”, rather than accelerator “mechanics”.

An early decision in the design phase of the control system was to base all high level functions involving machine setup and operation on accelerator models rather than resorting to a “look-and adjust” method of operation. This was deemed particularly crucial during the commissioning phase of CEBAF, when it was required to reconcile the machine behavior with its model.

II. ARTEMIS MODEL ENVIRONMENT

The Accelerator Real TimE Modeling Information Server (ARTEMIS), currently under development, will be a central Server/Client facility in the CEBAF accelerator control system, providing various model data (transfer matrices, twiss parameters, etc.) and supporting computations (e.g. quad strength calculation for matching) for all model-driven facilities, including LEM (Linac Energy Management) software, beam steering, feedback, and beam diagnostic and optimization procedures [Fig. 1]. Centralizing the model calculations provides a uniform and consistent data collection for these and other applications, while eliminating the need for redundant calculations by different application software.

ARTEMIS will also be interfaced with on-line facilities such as Tcl/Tk and MATLAB, allowing for the rapid prototyping of model-based algorithms and applications before they are made a permanent part of the control system.

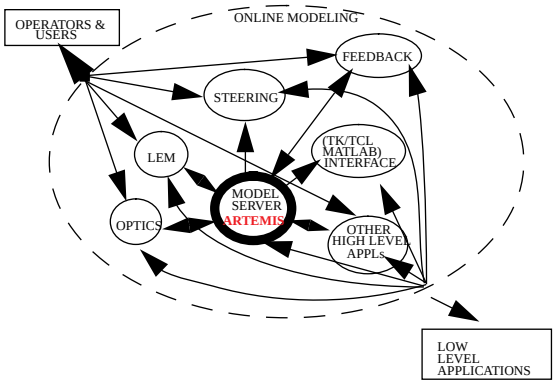

Figure 1: Modeling Environment Overview

Objectives

The main objective for ARTEMIS is to make available timely and accurate lattice information for use by any accelerator application software. Timeliness implies that the model must reflect changes in machine parameters at an acceptable rate. This is achieved through several updating mechanisms:

- Periodic updating of model(s) at a given rate.
- Model updates triggered by an external event.
- User initiated model calculations (on-demand modeling).

In addition, model accuracy demands correct treatment of all machine components, to the degree required by applications. ARTEMIS will provide availability of the following:

- 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.
- Non-relativistic effects (handled by PARMELA interface).
- Spin polarization tracking.

Input to ARTEMIS will be the accelerator layout of all elements (using MAD standard for element definitions) including bends, quadrupoles, sextuples, correctors, BPMs, and viewers. This information includes device setpoints, position in accelerator coordinates, x, y and z, and path length. Commands supported by the server include:

- Retrieval of lattice functions.

*Supported by U.S. DOE Contract DE-AC05-84-ER40150

- General first and second-order transfer matrices.
- Retrieve hardware information, e.g. list of quadrupoles, BPMs, etc. in a given region, with the ability to use wildcards for specifying a list of devices.
- Update current model, i.e., using current setpoints, calculate machine parameters.
- Update the machine to the reflect the desired model.

ARTEMIS will provide the capability to generate four distinct machine model classes:

- Golden model: this includes a consistent set of machine parameters that have been verified and deemed reasonable by an authorized expert. It is anticipated that this is the model that will be used by many control applications.
- Design model: this is a machine model based on the baseline design and setpoints of the accelerator.
- Current (pseudo-real-time) model: this is a machine model representing the latest or current accelerator setpoints, the update mechanism may be autonomous or user initiated. This form has utility for control codes which perform changes in lattice parameters.
- Simulation model: This model reflects the outcome of *what-if* scenarios as applied to the accelerator lattice. Possible applications include investigation of stray magnetic fields, focusing errors, injection errors, etc.

The objective of allowing ARTEMIS to exist in several distinct forms requires that the interface be available to all forms of software, preferably utilizing reusable code. This will be achieved by the EPICS Device API (known as cdev), currently under development at CEBAF. This layer provides any client access to the previously-defined model types, as well as to control system parameters (via. channel access), database information, etc., all using a cohesive interface. ARTEMIS will act as a server process to the Device API.

III. THE MODEL DATABASE

The CEBAF beamlines are stored in an object oriented database. The commercial package chosen for use as the centralized database is ObjectStore. An object-oriented database was chosen over the more conventional relational form due to the decision to employ an object-oriented approach to the design and implementation on ARTEMIS. Typical accelerator beamlines lend themselves naturally to this approach, and most existing accelerator codes utilize the concepts of object-oriented design. For example, the definition of a FODO (Focusing-Defocusing) cell can be considered as an object, and many cells can be ordered together forming a collection known as a beamline, etc. One defines the classes, using inheritance, and these structures are used directly by the database. These same classes will be used in the actual ARTEMIS server, as well, exploiting reusability of code.

This database is automatically generated from the same set of input files that are used by CEBAF physicists to define and study the accelerator lattice. These files use the "standard" (MAD style) notation for defining beamline and element properties. The object-oriented class breakdown for ARMTEMIS is as follows: an element is a generic beam line component that covers a physical entity on the beamline, such as a magnet, a beam position monitor, an accelerating cavity, a marker, a drift space, etc. The elements attributes have been characterized as layout specific information, and conditional or calculated information.

Figure 2 illustrates the C++ classes defined for the input lattice definitions. The basic building block is the *beamElement* class, representing an element as defined above. The *beamElement* class inherits from the class *bmlnCondition* and references static information about a beam element via the bmlnLayout class. This class serves as a base class for each particular element type. The information contained in each element is dependent on element type.

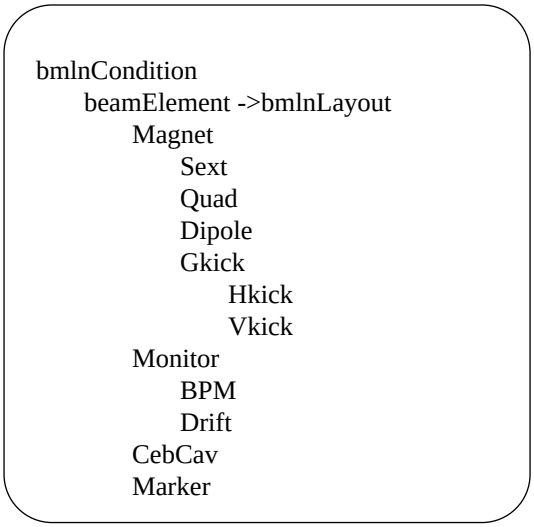

Figure 2: ARTEMIS Class Inheritance Structure

The *bmlnCondition* class is associated with every element. It maintains the “physics” of the element, such as lattice functions and transfer matrices, initial lattice conditions, element optical strength, beam momentum, etc. A matrix class is referenced here in order to take advantage of existing C++ matrix libraries.

The *bmlmLayout* class contains static information about an element defined in the lattice. The name field identifies the element to the model, and is unique throughout a section, which is defined as an ordered collection of elements. The class also maintains a unique EPICS cntrlid control system identifier. Physical information, such as element length, position, misalignment, etc., are also included in this class definition.

The smallest object that ARTEMIS server will operate upon is a beamline section, defined as an ordered collection of beam elements (*beamElement*). The beamline section class inherits from a *bmlnSet* collection of beam elements. Each section also includes an unique identifier used for access purposes. Initial conditions for matrix, Twiss, etc. are also maintained for the use of section matching.

The beamline class inherits from a *bmlnSet* collection of *beamSect*. This is an ordered collection of beam sections. ARTEMIS will have the ability to load and/or save beamlines, as well as sections. ARTEMIS can also create a beamline by joining beam sections.

Section method *madCr8db* is designed to download, to the database, a beam section from a MAD *hardware* output command. Drifts are concatenated and given unique names based on position in the accelerator in order to reduce unneeded elements. Complimentary, beamline method, *dbCr8mad*, can take

a beamline and construct a MAD input lattice deck for use with existing accelerator codes, such as DIMAD. The desire is to create an environment, using ObjectStore, which maintains all lattice information useful for the CEBAF project.

IV. MODEL SERVER

The server section of ARTIMIS is illustrated in figure 3. It consists of several subassemblies: a connection to the ObjectStore lattice database which performs the input-output lattice management for the server, a client communications section utilizing the EPICS Dev-API and ACE socket services, the actual server process ART, GenX model engine controller process, and CAUListener which provides the interface to EPICS network service. The specific global organization, object-oriented design approach, and use of C++ for implementation, was chosen for several reasons. The first is rapid server response to client connections and requests, which led to the use of multiple processes and the centralized shared memory segment. This arrangement allows GenX, the model engine process, to perform updates to model sections concurrently with ART server operations, improving client response time. Another benefit of the use of object-oriented approach during the design phase of ARTEMIS was that it allowed for independent design efforts for each object once the object interfaces were defined.

As stated above, GenX is used to perform the computationally intensive transfer matrices generation and propagation. The ARTEMIS server process performs the construction of Twiss parameters, sigma matrices, transfer matrix multiplication, ray propagation, polarization tracking, and response matrices, all of which pose a smaller computational burden, and can be built upon previously-calculated GenX transfer matrices.

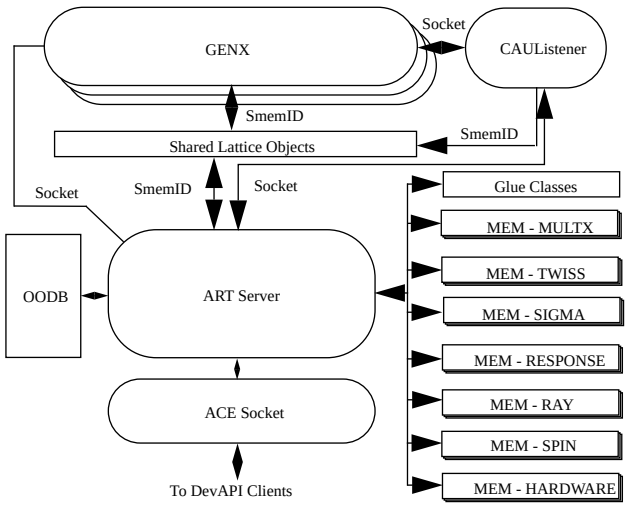

Figure 3: ARTEMIS Server Architecture

Figure 4 is a block diagram of the C++ classes used in the ARTEMIS server process. These structures maintain client connection information, GenX process control, management over the shared lattice objects (from ObjectStore), and management of computed data objects, such as Twiss parameters.

The object-oriented approach also allows for the general reusability of existing accelerator codes. For example, the GenX process consists of classes for memory interface and communication, and an interface to a modeling engine. The engines chosen for CEBAF are DIMAD and PARMELLA, but the interface to GenX is sufficiently defined such that practically any modeling code can be interfaced. The class definitions provide constructor routines which handle many of the overhead functions usually performed by modeling codes, such as lattice generation and management, and model computation I/O. The addition of a new accelerator model would consist of creating the appropriate placeholders in the shared lattice objects, and providing the interface into GenX. An example is the concurrent use of PARMELLA for modeling the CEBAF injector region and DIMAD for the remainder of the machine, all under the control of one ARTEMIS server.

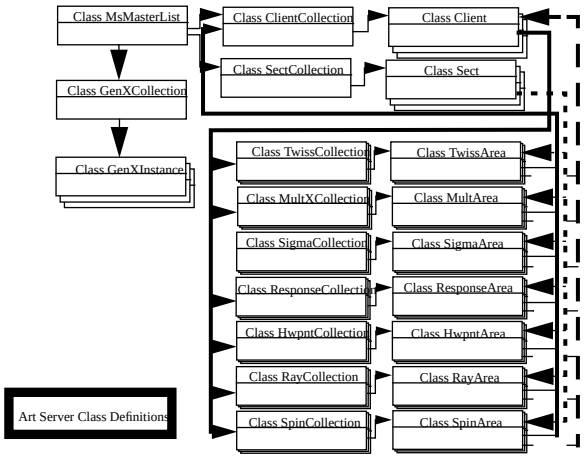

Figure 4: ARTEMIS Class Definitions

V. ARTEMIS STATUS

Work is currently underway on implementation of the object-oriented database and the ARTEMIS server. First level testing is expected in June, 1995, which will include integration with DIMAD and the cdev device API.

V. REFERENCES

[1] J. Kewisch et al., *"Accelerator simulation and operation via identical operational interfaces,"* Proceedings of the fourtheenth biennial Particle Accelerator Conference on Accelerator Science and Technology, May 1991, p. 1443.

[2] M. Bickley et al., *"Managing control algorithms with an object-oriented database,"* Proceedings of the sixteenth biennial Particle Accelerator Conference on Accelerator Science and Technology, May 1995.