The Design and Implementation of the SWIM Integrated Plasma Simulator

Wael R. Elwasif
David E. Bernholdt
Aniruddha G. Shet
Computer Science & Mathematics Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831
{elwasifwr,bernholdtde,shetag}@ornl.gov

Samantha S. Foley
Randall Bramley
Computer Science Department
Indiana University,
Bloomington, IN 47405
{ssfoley,bramley}@indiana.edu

Donald B. Batchelor
Lee A. Berry
Fusion Energy Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831
{batchelordb,berryla}@ornl.gov

Abstract—**As computing capabilities have increased, the coupling of computational models has become an increasingly viable and therefore important way of improving the physical fidelity of simulations. Applications currently using some form of multi-code or multi-component coupling include climate modeling, rocket simulations, and chemistry. In recent years, the plasma physics community has also begun to pursue integrated multi-physics simulations for space weather and fusion energy applications. Such model coupling generally exposes new issues in the physical, mathematical, and computational aspects of the problem. This paper focuses on the computational aspects of one such effort, detailing the design, and implementation of the Integrated Plasma Simulator (IPS) for the Center for Simulation of Wave Interactions with Magnetohydrodynamics (SWIM). The IPS framework focuses on maximizing flexibility for the creators of loosely-coupled component-based simulations, and provides services for execution coordination, resource management, data management, and inter-component communication. It also serves as a proving ground for a concurrent "multi-tasking" execution model to improve resource utilization, and application-level fault tolerance. We also briefly describe how the IPS has been applied to several problems of interest to the fusion community.**

I. Introduction

The Center for Simulation of Wave Interactions with Magnetohydrodynamics (SWIM) is devoted to improving the understanding of the interactions of radio frequency (RF) wave and the extended magnetohydrodynamic (MHD) phenomena of burning plasma. It is one of three projects supported by the U.S. Dept. of Energy's Scientific Discovery through Advanced Computing (SciDAC) program to explore different multi-physics phenomena as conceptual and scientific prototypes for a planned Fusion Simulation Project [1], [2], targeting integrated "whole-device" modeling of the International Thermo-nuclear Experimental Reactor (ITER) [3] and future experimental tokamaks.

The SWIM project [4] includes two physics research elements and one supporting computational research element [5]. The "fast MHD" physics campaign addresses long timescale discharge evolution in the presence of sporadic fast MHD events. This effort involves the use of RF and other driving sources to study and control fast time-scale MHD phenomena to achieve long-time MHD stable discharges and control sawtooth events. The "slow MHD" campaign models the direct interaction of RF and extended MHD for slowly growing modes, with the goal of controlling neoclassical tearing modes. The computational research element involves the development of a framework to enable the coupled multi-physics simulations required by these two physics campaigns in an efficient but flexible fashion. The resulting Integrated Plasma Simulator (IPS) is the evolving product of this computational research effort.

This paper describes the SWIM project's requirements for the IPS (Sec. II), the design of the framework (Sec. III) and its implementation (Sec. IV). We briefly discuss how the IPS is being used to model ITER plasma discharges (Sec. V) before concluding the paper, including a discussion of future plans for the IPS (Sec. VI).

II. Requirements

The design of the IPS framework has been strongly driven by the scientific needs of the SWIM project, but with the idea that it should also be able to serve the needs of a broader community. At the same time, the design has also been constrained in certain ways by characteristics of the fusion modeling community.

The essential scientific requirement is that the IPS be able to flexibly support integrated modeling based on a fairly broad range of physical phenomena, with the possibility of multiple interchangeable codes for each type of physics. Consequently, the framework must be able to flexibly accommodate the experimentation that may be required to arrive at effective solutions. The flexibility requirement is further reinforced by the desire to accommodate multiple implementations of each type of physics, to facilitate comparisons of the effect on the coupled simulation of different modeling approaches, ranging from high-fidelity to reduced models. It should be possible for SWIM scientists to rapidly explore different coupling schemes and different combinations of components without the need to modify the framework itself.

The complexity of the underlying coupled physics, and the need to focus on understanding the interactions between different physical phenomena translate into the need to adopt a simple coupling protocol. Such a protocol should be easy to implement (and more importantly easy to *debug*) to quickly isolate protocol errors as they manifest themselves in simulation outputs.

The basic physics of interest to SWIM has, to a significant extent, already been expressed in the form of various standalone physics codes, which has both advantages and disadvantages from the standpoint of developing an integrated modeling capability. While it is possible to avoid the time and expense of developing the functionality from scratch, working with large bodies of existing code can also be constraining. In the case of SWIM, many of the existing codes of interest have extensive histories and user bases apart from SWIM, and are undergoing continual development. Most of the code "owners" (who are, in most cases, also members of the SWIM project team) were reluctant to accept major changes to their code solely to support the needs of the SWIM project, and the SWIM project had a very strong desire to avoid "forking" physics codes into a SWIM version and a general standalone version because it would make it much harder for SWIM to leverage other improvements to the codes. Recognizing, at least in the initial phases of the project, that the coupling between physics components could be relatively loose (modest in both frequency and volume of data exchanged), we determined that it would be sufficient to work, in so far as possible, with the physics applications as is, even though this implies a high level of diversity in parallel scalability, code structure, build systems, external dependencies, I/O subsystems, and other factors. Another ramification of the desire to work with existing code to the greatest extent possible is that it would be incumbent on the framework to provide a common set of data management functionality to facilitate a diverse set of codes successfully exchanging data.

However we also anticipated that the physics and mathematics of the coupled simulations would eventually require tighter couplings and unavoidable intrusions into the standalone physics codes. While we explicitly deferred addressing these requirements in the framework until the need arose on the physics side, we wanted a provide clear path from the initial framework design and implementation to a future higher-performance tightly-coupled version.

The final source of requirements is the target computer platforms. In our case, the primary targets are Linux clusters and high-end facilities, to allow highly-scalable SWIM codes to run at scale. These are generally centralized resources, managed by a batch queue. Current high-end systems, such as the Cray XT and IBM Blue Gene series, tend to have limited operating system capabilities on the compute nodes (e.g. no dynamic linking, standard socket communications, or other features). The IPS and applications built with it must be able to run in these environments.

III. Approach and Framework Design

The design requirements outlined in section II guided our decision to adopt a component-based design for the IPS environment. Component Based Software Engineering (CBSE) has long been recognized as an effective approach to manage the growing size and complexity of modern software.

Other approaches, such as workflow tools (for example, Kepler [6], [7]), might have served as well for the immediate requirements of a loose, file-based coupling, but do not provide a path that we felt would satisfy the eventual need for higher performance coupling. Efforts such as the Common Component Architecture (CCA) [8], Cactus [9], Salome [10], among others have demonstrated the viability of the CBSE approach in large scale scientific computing, and would allow the same concepts and core software architecture to serve both immediate and longer-term needs.

At the core of the CBSE approach lies the view of the *software component* as a unit of software development and composition that has well defined boundaries. Components interact with each other through well-defined *interfaces*, and different components which conform to a common interface can, in principle, be interchanged easily. A component *framework* provides the environment in which components are composed together into an application and executed, and a set of services on which components can depend.

For SWIM, as in many coupled simulation projects, the different physics models map quite naturally onto the concept of components. The desire to support multiple implementations in each type of physics maps to the idea of interchangeable components.

The flexibility of a component-based software system is largely a matter of the design and implementation of the software architecture. In general, a light weight framework will delegate more functionality to the paticipating components, rather than the framework itself, allowing for greater flexibility in the simulation environment. We have adopted the light weight framework approach, while providing a robust and easy to use environment for fusion simulation. Thus, the IPS is designed to provide a modest set of services to manage configuration, resource allocation, data, and task execution. We use a "driver component" design pattern, which is quite common in CBSE, to put the control of the simulation's workflow into a user-level component rather than building it into the framework.

To facilitate data management, we introduce the idea of a "plasma state" as the repository for the collective information that define the state of the simulation at any moment. (Though in practice, as discussed below, our Plasma State holds only the data that needs to be shared with other components in the coupled simulation.)

To deal with the requirement to work with existing physics codes without modification, we have adopted the approach of wrapping the executables of the standalone physics code with a (usually thin) layer of code to serve as an adapter between it and the component environment desired by the SWIM project.

The overall design of the IPS framework and components, based on the approach just described, is shown schematically in Figure 1. The primary elements of the design are:

- **IPS Framework** The framework enables the runtime instantiation and configuration of constituent code compo-

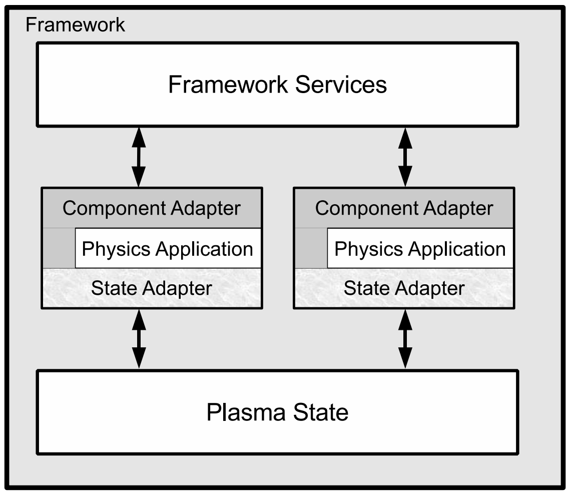

Fig. 1. IPS Simulation Framework

nents. Furthermore, the framework provides services that enable the coordinated execution of physics components in a coherent simulation.

- **Framework Services** Components use services offered by the framework to access application and component configuration, manage inter-component method invocation, control the execution of the underlying physics application, initiate data management and transfer operations, and perform event-based asynchronous communication with other parts of the simulation.
- **Plasma State** The plasma state is made up of one or more files that act as a repository of shared simulation data accessed by two or more constituent physics codes. The definition of plasma state quantities and rules for their ownership, instantiation, and inter-dependencies are agreed upon among all constituent codes.
- **Physics Component** A physics component is a wrapped version of the physics application that implements the IPS component API. The physics component adapts application data to/from data stored in the plasma state. Additionally, the component transforms a method invocation into one or more invocations of the underlying physics application, performing any needed pre and post processing. A special *driver* component is used to coordinate the execution of other physics components through the services of the IPS framework.
- **Physics Application** The application is an existing code that is integrated into the IPS simulation environment. Codes need to have well defined input files, output files, and control mechanisms (either command line or special input files) that control its operation. Furthermore, each code defines the subset of plasma state quantities which it owns (ownership indicates primary responsibility for instantiation and update).

In the original implementation of the IPS, computational tasks (component method invocations) could utilize parallel physics applications underneath, these tasks could only be executed sequentially. Because the applications of interest vary widely in parallel scalability and computational intensity, we introduced a "multiple-component multiple-data" (MCMD) execution model which allows multiple parallel tasks to be executed concurrently within the framework. Although this change did not alter the general design as shown in Figure 1, it did require the introduction of new services and impacted other aspects of the IPS implementation. The implementation of the current MCMD version of the IPS is described below.

IV. IPS Implementation

In this section, we discuss the capabilities, responsibilities, and salient implementation details for the major elements of the IPS framework. Figure 2 illustrates the architecture of the IPS and will serve as a useful guide throughout this section.

A. Framework

The IPS **framework** provides the environment in which components are assembled and executed as a coherent simulation. The framework makes certain *services* available to components to invoke as appropriate during the various stages of execution. Such services can be broadly categorized into component instantiation and configuration management, resource management, task and remote method execution coordination, and events management.

To meet the flexibility and adaptability requirements outlined in section II, we opted to implement the framework and component wrappers in a scripting language, Python, that meets those requirements. In addition, Python is increasingly being used in large scale scientific applications, making available a rich ecosystem of utility modules that can be used to simplify the coding of IPS component wrappers.

The IPS architecture relies on the concept of *ports*, an abstraction that represents the type of physics involved in an IPS simulation. Each port is implemented by a particular component (wrapped physics executable). By convention, the primary interface exposed by all components consists of three methods, `init()`, `step()`, and `finalize()`. This interface is similar to the one adopted by the Earth System Modeling Framework (ESMF) [11], though the framework itself does not actually limit the component interface, and several IPS components have extended interfaces to satisfy specific needs.

B. Framework Services

Framework services are provided by several *managers*, which collectively constitute the bulk of the IPS framework. The remaining code in the framework mainly handles simulation start-up and shutdown, and routing of service requests to appropriate managers. What follows is a brief description of the different managers and the services they provide.

The **configuration manager** is responsible for two major functions within the framework. It maintains a database of simulation configuration parameters provided in one or more

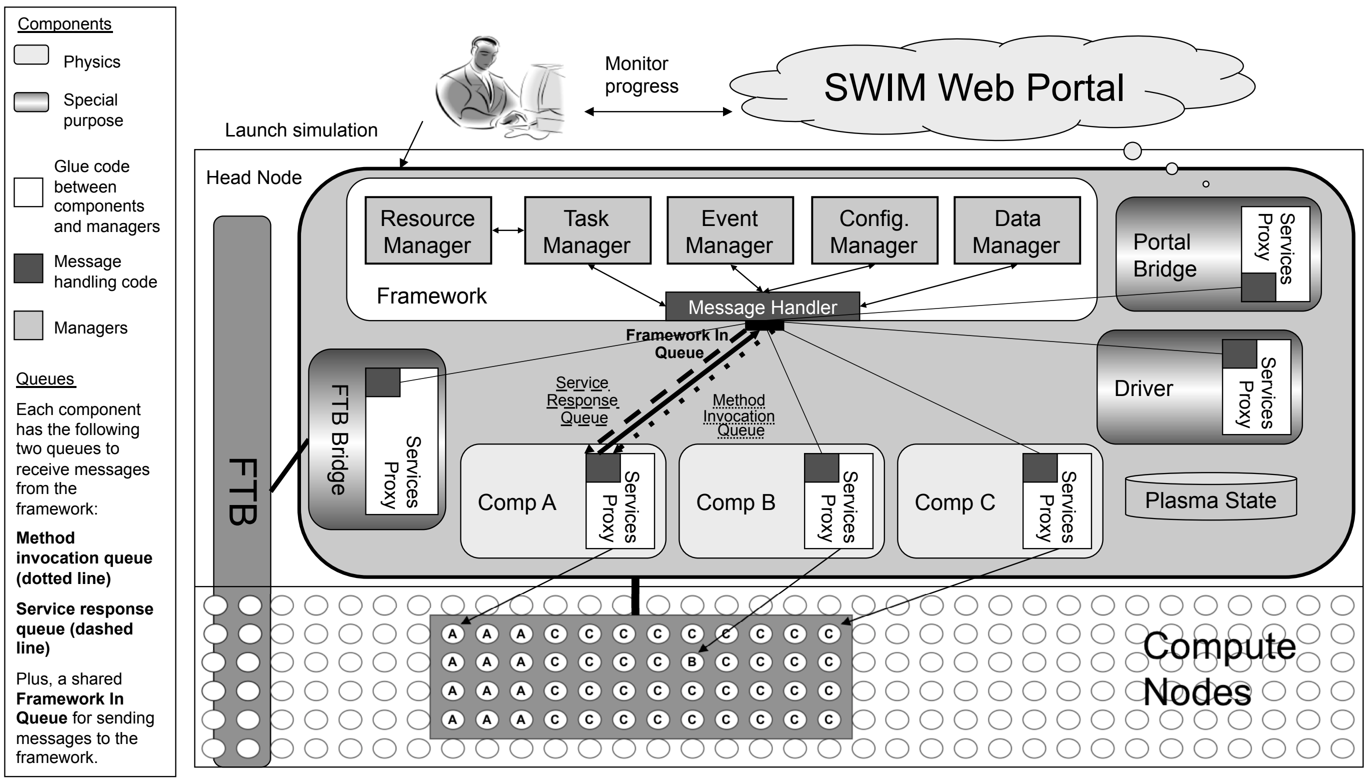

Fig. 2. An illustration of the relationships between the framework, components, services, and tasks in the Integrated Plasma Simulator. The queues used for communication between the framework services and the components, and the process layout on a typical high-end computer system are highlighted.

simulation configuration files. These files are parsed on framework start-up and data therein can be queried by components during execution. Among the data entries are: simulation work directories, simulation identification tags, specification of the simulation time loop, and location of simulation monitoring web portal. The structure of the simulation configuration file is described in more detail in [12].

The configuration manager is also responsible for instantiating simulation components, as Python wrapper objects, based on entries in the configuration files. Such components are indirectly accessed (typically by the driver component) using the ports they implement. The mappings of ports to components are maintained by the configuration manager. The current implementation of the IPS maintains a one-to-one correspondence between ports and components. This restriction may be relaxed in the future, allowing multiple components that implement the same port to be used in the same simulation, should the need arise.

The **resource manager** is responsible for discovering and allocating the resources available to the framework. Currently it only manages available compute nodes but may, in the future, be extended to manage other resources like storage, memory, I/O or bandwidth. In the current design, the resource manager is used internally by the task manager. No resource manager services are directly accessible by the individual components.

The **task manager** mediates two types of activities initiated by IPS components. The first activity is the inter-component invocation of methods defined in the component's API. Such invocations can be blocking, where the caller suspends execution pending call completion. The task manager also supports non-blocking calls, allowing methods from multiple components to be concurrently active. It should be noted that, since component objects themselves are not multithreaded, only one method can be active at any given time for any one component.

The task manager also handles the execution of *tasks*. An IPS task involves the execution of an underlying code on one or more compute nodes. A component can launch one or more tasks as part of the execution of its methods. The task manager supports both blocking and non-blocking task execution, allowing, for example, a component to orchestrate task-parallel execution of underlying physics code. In launching tasks, the task manager relies on the resource manager to manage the pool of compute nodes available to the IPS simulation. More information on the execution model of the IPS is presented later, in section IV-F.

The **data manager** provides a platform independent interface that enables the physics components to conveniently manage their input and output data staging requirements. Lists of input and output files for each component are specified in the configuration file. The component input files comprise all of the files needed for the component to perform its physics task (except for data generated by other components in the simulation). Similarly, component output files constitute a subset of all files generated by the component, and which may be used for component-specific analysis and debugging. In addition to component specific data files, the data manager mediates shared access to the plasma state files, caching snapshots of the plasma state in a component's work area upon request, and managing the merger of updates from multiple components into a shared *master* plasma state.

A recent addition to the IPS is an asynchronous **event service**, based on CCA's draft event service specification [13]. It provides access to *event channels* through a simple publish/subscribe event model. This service allows certain non-essential functions of the simulation to be loosely coupled to the simulation framework itself. Such functions include the broadcast of simulation progress events to an external web portal for monitoring. Another use case involves the use of resource failure information to modify the pool of resources managed by the resource manager. This later area is currently being explored in collaboration with the Coordinated Infrastructure for Fault Tolerant Systems (CIFTS) project [14], using failure information published on the Fault Tolerance Backplane (FTB) [15]. The event service is also being used to synchronize execution of certain concurrent physics codes, a feature currently being utilized in the SWIM slow MHD campaign.

C. Plasma State

In the IPS, we refer broadly to the *plasma state files* as the collection of files shared among the components that make up a running simulation. These files function as both a repository and the primary means of exchange for time evolving plasma simulation data, shared among the IPS framework and physics components. One special plasma state file is a netCDF file that contains a core set of plasma state variables. This file can also be accessed natively in Fortran, using a Fortran derived type. Multiple, separately-named core state instances can be declared and held in memory simultaneously. To keep the size of the plasma state data structure and core file reasonable, large (usually higher dimensional) items are typically stored in separate files, and the file names are stored in the plasma state.

By convention, each element of the Plasma State is *owned* by one component which acts as the primary writer of that element, and may be read by many. Early in the project, the members of SWIM defined the appropriate inputs and outputs for each class of component, which are now embodied in the Plasma State variable definitions. It is the responsibility of each component to adapt the data between the representations defined by the Plasma State and what they use internally.

While file-based data exchange is adequate to the current needs of the SWIM project, clearly it is not a universally suitable approach to coupled simulation. In fact, in our initial planning and design, we anticipated a need to support higher-performance in-memory (parallel) data exchange for at least some components. Although this need has not yet materialized, we anticipate addressing it with help from the Common Component Architecture, which naturally supports such high performance interactions. The CCA could either be used selectively, essentially encapsulating the tightly-coupled codes which the IPS might treat as a composite component, or the whole of the IPS could be recast on top of a CCA infrastructure instead of the current Python infrastructure. The CCA's Babel language interoperability tool [16], [17], which supports Python (and Java) as well as traditional HPC languages (C, C++, and Fortran), would allow much of the current IPS's simplicity and flexibility to be retained in either approach.

D. IPS Components

As previously described, we have adopted the strategy of using unmodified physics executables as the heart of the simulation functionality of the IPS. **Physics components**, therefore, are Python wrappers that provide the necessary adaptation between the IPS component architecture and the underlying executables and manage interactions with the framework. The wrappers translate the component architecture's method invocations (generally `init()`, `step()`, and `finalize()`) into appropriate invocations of the underlying physics application. Wrappers also orchestrate the transfer of data between the plasma state and the application's native data formats. Component wrappers are generally quite straightforward and follow a common template, though each differs in specific details. They are intended to be written by the providers of the underlying physics codes.

There are several special components in the IPS that do not necessarily follow the Python-wrapped-application model. The **driver component** is responsible for orchestrating the overall simulation execution flow. The driver uses the *task manager* to invoke publicly accessible methods defined on the physics components that comprise the bulk of the simulation. This invocation can be blocking, or non-blocking, based on factors that include application logic, intra-component data dependencies, resource utiliztion, among other factors. More details on the IPS execution model can be found in section IV-F. The driver component is distinguished in the IPS as being the first component invoked in order to initiate a simulation, but otherwise conforms to the same architecture as other IPS components. In practice, IPS users write a different driver component for each type of simulation they want to perform. Drivers are usually written entirely in Python, which makes them quite flexible and easily modified, though many drivers will be quite generic and readily reusable.

The **monitor component** is another special component which has proven useful to SWIM researchers. The function of the monitor component is to extract key data of interest from the plasma state and to make it available to an external monitoring application which is used to check key diagnostics of running or completed simulations. The monitor produces a netCDF file containing time series of the variables of interest (in contrast to the plasma state, which is just a single time slice), which can be viewed with the ElVis visualization and monitoring tool [18] or other standard netCDF tools.

E. Additional Services

The SWIM project includes a capability to monitor running simulations using the **SWIM web portal**. The portal, based on the Fusion Grid portal [19] collects and presents information about the various simulation runs being carried out with the IPS, tracking their progress and status. The connection between the IPS and the portal is provided by a special IPS component, called *the portal bridge*. From the framework's standpoint, the portal bridge is another component. However this component is not part of a separate IPS simulation, and is permanently attached to the framework. The portal bridge subscribes to status and progress events published by the framework as part of the invocation of various service methods, and transmits them to the web portal through a simple HTTP-based interface.

As previously mentioned, the IPS event service can also be connected to the FTB event service through the **FTB bridge** component. The FTB is intended to convey fault-related information throughout an HPC system, and the goal in coupling it to the IPS is to allow the IPS to take advantage of information provided by the FTB to respond intelligently to changing conditions within the system. For example, on receiving an event on the FTB indicating that a particular node is down, the FTB bridge would retransmit that event via the IPS's internal event service. The resource manager, on seeing this event, would take the ailing node out of the resource pool. If the node was idle, the IPS can continue and simply exclude that node from future task launches until another event is received indicating that it has been restored to service. If the node was in use at the time of the failure, the task running on it will fail, and the IPS can pursue appropriate recovery mechanisms, such as restarting the task on a different set of nodes.

F. Execution and Communication Model

The IPS is designed to operate in both Linux cluster and high-end computing environments. An IPS simulation run is submitted as a single batch job with an appropriate number of compute nodes allocated to it. In order to accommodate the constraints of high-end platforms, the IPS has been designed so that all of the Python code (framework and component wrappers) can execute on the system's head node, while the underlying physics executables are launched as parallel tasks on the systems' compute nodes using platform-specific program execution environment (see Figure 2).

To simplify the development of component wrappers and simulation drivers, we have adopted a process-based execution model over a threaded one. In this approach, each component instance (the Python wrapper portion) executes in a separate process (a child of the framework process), and creates another process for each back-end computational task it launches. The task model is a two-level launch. A *call* occurs when a method is invoked on a component and a *task* is launched when a physics application is executed by a component. The separation of these two actions allows the called component and the framework to handle data movement, resource allocation, failures and the computing environment without the calling component being involved. This model also lends itself to various modes of execution where multiple tasks in the same or different components execute concurrently in separate processes.

Interactions between the component and the framework are handled through the *services proxy* present in each component, which masks the distributed (multi-process) nature of the IPS from the component implementation. From the component's perspective, framework services are invoked on the services proxy and results returned. Internally, the services proxy invokes remote procedure calls (RPC) on the framework process through a set of communication queues, shown in Figure 2, as different lines between component and framework's message handler. Messages from components destined for the framework are sent along a shared queue (solid line), to be processed by the single-threaded framework in a FIFO manner. Responses from services travel along individual queues (dashed line) to the components that request them. A separate queue (dotted line) is used to submit component method invocations to be processed by the component in a FIFO manner as well. Results from component method invocation are communicated back to the framework using the same shared framework queue. This separation of the communication channels for invocation of framework services and component method invocation simplifies the implementation of the IPS considerably.

This design, along with the inclusion of non-blocking method call and task launch capabilities in the task manager enable the flexible utilization of the IPS's MCMD capabilities. Multiple component methods can be active concurrently, provided the simulation logic and data dependencies allow for such a scenario. Furthermore, a single component method implementation can launch multiple concurrent instances of the underlying physics application, collating their results into the plasma state upon completion. These capabilities are currently being explored to restructure traditional serially coupled simulations to improve over-all execution times.

V. Application to ITER Plasma Discharges

The IPS is an essential tool that enables SWIM researchers to explore interesting coupled physics. As is often the case, introducing multi-physics coupling can reveal unexpected physical and mathematical nuances which must be understood and addressed. While this work is going on, the IPS is also being used in several scientific studies, one of which we highlight here.

The ITER experimental fusion reactor is an international effort to build a new large fusion device, which is designed to produce several hundred megawatts of fusion energy. Although the ITER tokamak is still years from completion, integrated modeling of anticipated ITER plasma discharges already plays a very important role in the ITER program, both to help finalize the design, and to plan the research program. Current integrated modeling approaches used for ITER typically take

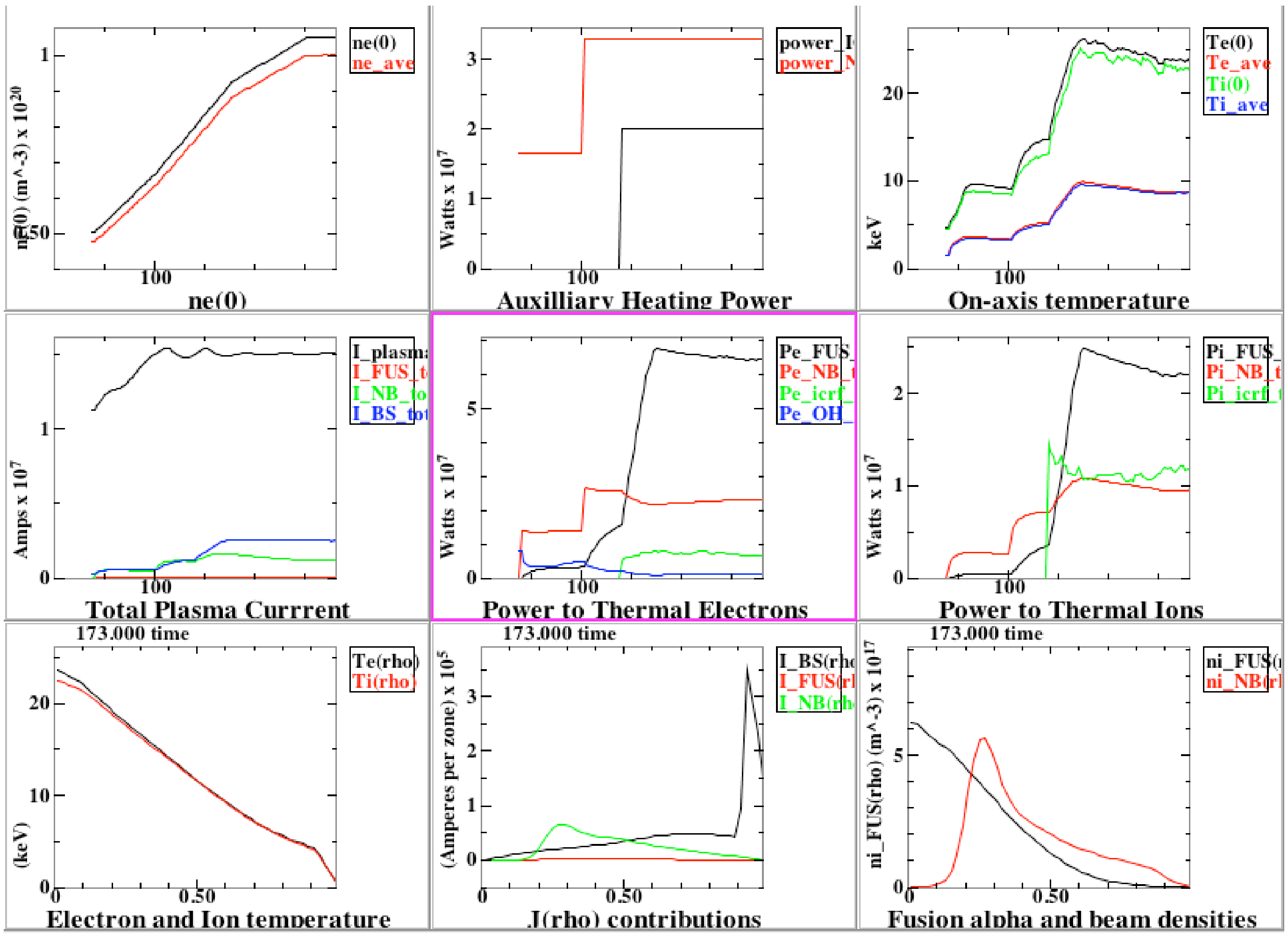

Fig. 3. Sample IPS results for an ITER simulation. Power sources include two neutral beams and ion cyclotron heating. In this case, the plasma heating is dominated by fusion alpha particles.

1 to 1.5 months per run with moderate fidelity models. Many such runs are required for surveys (parameter sweeps) and optimization studies.

The SWIM project has adopted as a near-term goal of demonstrating the ability to significantly accelerate such simulations through the use of massively parallel computers while simultaneously improving the physical fidelity through the use of better and higher resolution models. Figure 3 shows representative results [20] from simulations which use the IPS to integrate the TSC/GLF23 [21]–[23] transport model, the AORSA full-wave ICRF code [24], and the NUBEAM neutral beam code [25], [26], which had previously been studied with the TSC/PTRANSP code [27]–[29] with serial version of the heating and transport modules. The IPS-based simulations on the Cray XT4 at the National Energy Research Supercomputer Center (NERSC) used 2056 processors for AORSA and 512 processors for NUBEAM. The NUBEAM code used 10^6 particles for neutral beam injection and 10^6 for fusion products, while the AORSA calculations used 256×256 Fourier modes in the poloidal plane. These calculations are much higher resolution than were previously feasible and were carried out in six days rather than six weeks. By replacing AORSA with the less computationally intensive TORIC code [30], we expect to be able to get further significant reductions in the turn around time for these simulations, while maintaining a much higher fidelity than the original TSC/PTRANSP simulations.

VI. Conclusions and Future Work

We have described the origins, design, and implementation of the Integrated Plasma Simulator, a component framework for loosely coupled fusion modeling. At present, nine different components have been used within the IPS framework, representing six different types of physics, and two more applications are in the process of being componentized. The IPS is being used in a variety of applications, including the ITER discharge simulations described here.

Work is underway to take advantage of some of the newly-added features of the framework, including support for the MCMD-style of execution. A number of different MCMD scenarios were developed during the design of this capability, and several are now in the process of being realized as simulations.

For the fast MHD campaign, the focus is on making better use of computational resources, given the wide range of parallel scalabilities of the various components. Simulations in this campaign typically involve 1-2 scalable codes, which often do not have direct data dependencies, and numerous minimally scalable codes, including linear stability analysis codes which can be externally parallelized over toroidal modes. The MCMD capability will be used for the concurrent execution of independent tasks in different phases of the simulation time step.

For the slow MHD campaign, one of the first simulations will involve computationally-intensive and long-running NIMROD MHD code [31] and the GENRAY RF code [32] which is minimally parallel and runs relatively quickly. As the solution computed by NIMROD changes over time, it will periodically need updates from GENRAY, which will be run concurrently to avoid the need to stop and restart NIMROD. This scenario will also make use of the event service to signal when GENRAY updates are needed.

As has previously been mentioned, the event service, and the IPS as a whole will serve as a research tool to explore application-level fault resilience as part of the CIFTS project. In this context, the flexibility of the IPS allows it to represent a wide range of application characteristics. We are particularly interested in issues faced by component-based applications, and MCMD-style applications, though there are many research opportunities in this area.

Finally, we should note that the IPS continues to undergo development as new needs or opportunities arise. Examples of possible areas of future work include developing explicit (verifiable) "interfaces" for file-based data exchange, and dataflow-based mechanisms to drive simulations.

Acknowledgments

This work has been supported by the U. S. Department of Energy, Offices of Fusion Energy Sciences and Advanced Scientific Computing Research, and by the ORNL Postmasters Research Participation Program which is sponsored by ORNL and administered jointly by ORNL and by the Oak Ridge Institute for Science and Education (ORISE). ORNL is managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. DE-AC05-00OR22725. ORISE is managed by Oak Ridge Associated Universities for the U. S. Department of Energy under Contract No. DE-AC05-00OR22750.

This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U. S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] J. Dahlburg, J. Corones, D. Batchelor, R. Bramley, M. Greenwald, S. Jardin, S. Krasheninnikov, A. Laub, J.-N. Leboeuf, J. Lindl, W. Lokke, M. Rosenbluth, D. Ross, , and D. Schnack, "Fusion Simulation Project: Integrated simulation and optimization of fusion systems," *J. Fusion Energy*, vol. 20, no. 4, pp. 135–196, December 2001.

[2] D. Post, D. Batchelor, R. Bramley, J. Cary, R. Cohen, P. Colella, and S. Jardin, "Report of the Fusion Simulation Project steering committe e," *Journal of Fusion Energy*, vol. 23, no. 1, pp. 1–26, March 2004. [Online]. Available: http://dx.doi.org/10.1007/s10894-004-1868-0

[3] "International Thermo-nuclear Experimental Reactor," http://www.iter.org.

[4] "Center for Simulation of RF Wave Interactions with Magnetohydrodynamics," http://cswim.org.

[5] D. B. Batchelor, E. D'Azevedo, G. Bateman, D. E. Bernholdt, L. A. Berry, P. T. Bonoli, R. Bramley, J. Breslau, M. Chance, J. Chen, M. Choi, W. Elwasif, G.-Y. Fu, R. Harvey, W. A. Houlberg, E. Jaeger, S. C. Jardin, D. Keyes, S. Klasky, S. Kruger, L. Ku, D. McCune, J. Ramos, D. P. Schissel, D. Schnack, and J. Wright, "Integrated physics advances in simulation of wave interactions with extended mhd phenomena," in *SciDAC 2007, 24–28 June 2007, Boston, Massachusetts, USA*, ser. Journal of Physics: Conference Series, D. Keyes, Ed., vol. 78. Institute of Physics, 2007, p. 012003. [Online]. Available: http://www.iop.org/EJ/article/1742-6596/78/1/012003/jpconf7_78_012003.pdf

[6] "Kepler project," http://kepler-project.org.

[7] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[8] B. A. Allan, R. Armstrong, D. E. Bernholdt, F. Bertrand, K. Chiu, T. L. Dahlgren, K. Damevski, W. R. Elwasif, T. G. W. Epperly, M. Govindaraju, D. S. Katz, J. A. Kohl, M. Krishnan, G. Kumfert, J. W. Larson, S. Lefantzi, M. J. Lewis, A. D. Malony, L. C. McInnes, J. Nieplocha, B. Norris, S. G. Parker, J. Ray, S. Shende, T. L. Windus, and S. Zhou, "A component architecture for high-performance scientific computing," *Intl. J. High-Perf. Computing Appl.*, vol. 20, no. 2, pp. 163–202, Summer 2006. [Online]. Available: http://hpc.sagepub.com/cgi/reprint/20/2/163

[9] T. Goodale, G. Allen, G. Lanfermann, J. Massó, T. Radke, E. Seidel, and J. Shalf, "The Cactus Framework and Toolkit: Design and Applications," in *Vector and Parallel Processing - VECPAR '2002, 5th International Conference*. Springer, 2003. [Online]. Available: http://www.springerlink.com/content/2fapcbeyyc1xg0mm/

[10] A. Ribes and C. Caremoli, "Salome platform component model for numerical simulation," in *Computer Software and Applications Conference, 2007. COMPSAC 2007. 31st Annual International*, vol. 2, July 2007, pp. 553–564.

[11] N. Collins, G. Theurich, C. DeLuca, M. Suarez, A. Trayanov, V. Balaji, P. Li, W. Yang, C. Hill, and A. da Silva, "Design and implementation of components in the earth system modeling framework," *International Journal of High Performance Computing Applications*, vol. 19, no. 3, pp. 341–350, 2005.

[12] W. R. Elwasif, D. E. Bernholdt, L. A. Berry, and D. B. Batchelor, "Component framework for coupled integrated fusion plasma simulation," in *HPC-GECO/CompFrame – Joint Workshop on HPC Grid Programming Environments and Components and Component and Framework Technology in High-Performance and Scientific Computing*, Montreal, Canada, 21–22 October 2007.

[13] "CCA Event Specification Proposal," https://www.cca-forum.org/wiki/tiki-index.php?page=Event+Specification+Proposal.

[14] "Coordinated Iinfrastructure for Fault Tolerant Systems," http://www.mcs.anl.gov/research/cifts/.

[15] R. Gupta, P. Beckman, H. Park, E. Lusk, P. Hargrove, A. Geist, D. K. Panda, A. Lumsdaine, and J. Dongarra, "CIFTS: A Coordinated Infrastructure for Fault-Tolerant Systems," in *Proceedings of 38th International Conference on Parallel Processing (ICPP) 2009*, September 2009.

[16] "Component technologies home page (llnl)," https://computation.llnl.gov/casc/components/components.html.

[17] T. Dahlgren, T. Epperly, G. Kumfert, and J. Leek, *Babel User's Guide*, babel-1.4.0 ed., CASC, Lawrence Livermore National Laboratory, Livermore, CA, 2009. [Online]. Available: http://www.llnl.gov/CASC/components/docs/users_guide.pdf

[18] E. Feibush, "ElVis," http://w3.pppl.gov/elvis/.

[19] "The national fusion collaboratory project," http://www.fusiongrid.org/.

[20] D. Batchelor, G. Alba, E. D'Azevedo, G. Bateman, D. Bernholdt, L. Berry, P. Bonoli, R. Bramley, J. Breslau, M. Chance, J. Chen, M. Choi, W. Elwasif, S. Foley, G. Fu, R. Harvey, E. Jaeger, S. Jardin, T. Jenkins, D. Keyes, S. Klasky, S. Kruger, L. Ku, V. Lynch, D. McCune, J. Ramos, D. Schissel, D. Schnack, and J. Wright, "Advances in simulation of wave interations with extended MHD phenomena," in *SciDAC 2009, 14–18 June 2009, California, USA*, ser. Journal of Physics: Conference Series, H. Simon, Ed. Institute of Physics, 2009, in press.

[21] S. C. Jardin, N. Pomphrey, and J. Delucia, "Dynamic modeling of transport and positional control of tokamaks," *J. Computat. Phys.*, vol. 66, no. 2, pp. 481–507, 2 October 1986.

[22] S. Jardin, M. Bell, and N. Pomphrey, "TSC simulation of Ohmic discharges in TFTR," *Nucl. Fusion*, vol. 33, no. 3, pp. 371–382, March 1993.

[23] R. E. Waltz, G. M. Staebler, W. Dorland, G. W. Hammett, M. Kotschenreuther, and J. A. Konings, "A gyro-Landau-fluid transport model," *Phys. Plasma*, 1997. [Online]. Available: http://link.aip.org/link/?PHP/4/2482/1

[24] E. F. Jaeger, L. A. Berry, E. D'Azevedo, D. B. Batchelor, M. D. Carter, K. F. White, and H. Weitzner, "Advances in full-wave modeling of radio frequency heated, multidimensional plasmas," *Physics of Plasmsa*, vol. 9, no. 5, pp. 1873–1881, 2002. [Online]. Available: http://link.aip.org/link/?PHP/9/1873/1

[25] R. J. Goldston, D. C. McCune, H. H. Towner, S. L. Davis, R. J. Hawryluk, and G. L. Schmidt, "New techniques for calculating heat and particle source rates due to neutral beam injection in axisymmetric tokamaks," *J. Computat. Phys.*, vol. 43, no. 1, pp. 61–78, 1981.

[26] A. Pankin, D. McCune, R. Andre, G. Bateman, and A. Kritz, "The tokamak Monte Carlo fast ion module NUBEAM in the National Transport Code Collaboration library," *Computer Phys. Comm.*, vol. 159, no. 3, pp. 157–184, 1 June 2004.

[27] R. Budny, R. Andre, G. Bateman, F. Halpern, C. Kessel, A. Kritz, and D. McCune, "Predictions of H-mode performance in ITER," *Nucl. Fusion*, vol. 48, no. 7, p. 075005, July 2008.

[28] F. D. Halpern, A. H. Kritz, G. Bateman, A. Y. Pankin, R. V. Budny, and D. C. McCune, "Predictive simulations of ITER including neutral beam driven toroidal rotation," *Phys. Plasmas*, vol. 15, no. 6, p. 062505, 2008, 11pp. [Online]. Available: http://link.aip.org/link/?PHP/15/062505/1

[29] R. V. Budny, "Comparisons of predicted plasma performance in ITER H-mode plasmas with various mixes of external heating," *Nucl. Fusion*, vol. 49, no. 8, p. 085008, August 2009.

[30] J. C. Wright, P. T. Bonoli, M. Brambilla, F. Meo, E. D'Azevedo, D. B. Batchelor, E. F. Jaeger, L. A. Berry, C. K. Phillips, and A. Pletzer, "Full wave simulations of fast wave mode conversion and lower hybrid wave propagation in tokamaks," *Phys. Plasmas*, vol. 11, pp. 2473–2479, 2004.

[31] C. R. Sovinec, A. H. Glasser, T. A. Gianakon, D. C. Barnes, R. A. Nebel, S. E. Kruger, D. D. Schnack, S. J. Plimpton, A. Tarditi, and M. S. Chu, "Nonlinear magnetohydrodynamics simulation using high-order finite elements," *J. Comput. Phys.*, vol. 195, no. 1, pp. 355–386, 2004.

[32] A. Smirnov, R. Harvey, and K. Kupfer, "A general ray tracing code GENRAY," *Bull Amer. Phys. Soc.*, vol. 39, no. 7, p. 1626, 1994.