HPC Application Software Consortium Summit
Concept Paper

Executive Summary

This white paper proposes the foundation of a High Performance Computing (HPC) Application Software Consortium (ASC) of academic, research and industry partners for the development and maintenance of a framework for multiphysics simulation on HPC systems using commercial and open-source codes. The goal of the proposed consortium is to focus the technologies, resources, and expertise to lower the development and ownership costs of multiphysics and multidisciplinary simulation software for both the commercial independent software vendors (ISVs) and HPC applications users. Council on Competitiveness research has demonstrated that HPC is a major enabler for U.S. competitiveness, but that the cost of HPC simulation software and the lack of software robustness is a significant barrier [1]. Council research has also identified an industry need to customize products by simulating *all life cycle considerations*, including market and regulatory requirements, in one integrated environment across the product's supply chain [2]. The ISV community for the HPC market is relatively small and fragmented, therefore unable to invest sufficient resources into new breakthrough products.

Overall, there is a need for improvement in sustained performance, scalability, functionality, and manageability of HPC software. Significant technical challenges remain to implement multidisciplinary and multiphysics modeling, model validation and verification, large-scale data management, and visualization. Emerging system-level simulation software products will require a breadth of performance-value-price points to satisfy HPC users ranging from entry-level cluster users to large HPC system users. In addition, open interfaces and infrastructures are required to support multiphysics modeling, simulation, and data analysis in an interoperable multi-vendor environment. The challenge presented is to change the paradigm from tooling, independent software vendors, computing system vendors, and individual supply-chain companies working separately to having these firms collaborating on solutions, building common frameworks, and adding cumulative market value.

1 Background

Based on recommendations of the Council's HPC Advisory Committee, the Council and the University of Southern California's Information Sciences Institute (ISI) have been leading an effort to assess the feasibility of an application software consortium to address the critical need for robust multiphysics and multidisciplinary simulation software for HPC systems. This effort includes:

- In December 2007, the Council and ISI organized a planning workshop with a core group of volunteers from academic computing centers, government laboratories, independent software vendors (ISVs), systems vendors, and industrial HPC users to discuss issues facing the community. Topics of the planning workshop included: 1) the need to support full-system multiphysics simulation, 2) the fact that future growth in performance is tied to parallelism because of multi-core, 3) concerns about the growing cost of commercial HPC software development, and 4) concerns about the rising cost of multiphysics simulation software to end-users. The concepts presented in this white paper represent findings from this workshop.
- In March 2008, an HPC Application Software Consortium Summit meeting is planned to further explore the requirements for a multiphysics infrastructure based on modeling and simulation. Participants will address technical and industry acceptance issues associated with open interfaces and open infrastructure that will enable multiphysics modeling, simulation, and analysis. The role of reference platforms and pilot projects will be addressed. The value proposition of the proposed consortium will be examined from an end-user and ISV perspective with the importance of creat-

ing a win-win environment for all parties. The summit will hold a strawman vote of attendees to determine interest in the creation of the proposed consortium. Attendees will also examine potential consortium models and charters.

2 Technical Approach

There are a number of critical design challenges facing the application physics simulation market that must be addressed in the next few years. One of the most important of these is that parallel programming will shortly be mandatory in modern workstations. In the past, the industry has taken advantage of the exponentially increasing processor performance to expand the capabilities of simulation software. Consequently, there hasn't been an incentive to improve the parallel scaling of existing software packages. However, the computer industry has reached a crisis where the transistor scaling consistent with Moore's Law no longer results in faster clock rates from Dennard scaling [7]. Instead, chip vendors are now increasing the number of cores per chip. Processor designs are expected to change dramatically in the coming years, adopting many-core architectures and dedicated accelerator units. This means that many of the legacy code solvers found in mainstream modeling packages will *not* continue to improve in performance unless they are rewritten take advantage of parallel processing cores and novel accelerator architectures.

In addition to parallel scaling, multiphysics and multidisciplinary coupling presents additional challenges for software complexity. Commercial modeling and simulation software has become more sophisticated and complex over time to tackle a broader range of challenging engineering problems. HPC provides an opportunity for higher fidelity models for increased simulation accuracy. Increased fidelity for the most challenging modeling problems depends as much on multiphysics and multidisciplinary coupling as it does increasing the resolution and accuracy of any of the individual component solvers. The need to tackle increasingly complex multiphysics and multidisciplinary simulation problems challenges current monolithic software packages and demands that the software be refactored to enable more modularity in order to support more flexible composition of solvers.

Addressing the parallel programming and multiphysics/multidisciplinary coupling challenges will significantly raise software development costs due to increased software complexity. However, industry representatives at the planning workshop observed that HPC usage is increasingly driven by the cost of software licenses. Either the market must expand to amortize the costs across a broader customer base or the software must scale in capability to justify the increased investment. The specter of software complexity faced by the application physics-based simulation industry bears some similarity to issues confronting the academic and government HPC community today as they develop complex multiphysics software for petaflop-scale HPC systems. They have discovered that the effectiveness of these systems is limited by the ability to field complex applications. This trend leads to a crisis where complexity hampers the ability of software companies to improve their software, which will ultimately limit the growth of the entire HPC market, both for hardware and for software. Software complexity must be managed to succeed.

Today, there is no single independent software vendor (ISV) that can provide integrated multiphysics and multidisciplinary solutions applicable to all fields of science and engineering. The scope of the software development challenge is too large for any single ISV to take on. What is needed is a framework and related standards that facilitate integration of existing solutions in order to create the type of multidisciplinary solution that industries require. Advantages of the common framework are clear for smaller ISVs that rely on the framework for integration into a complete solution. However, there are also many advantages for larger ISVs to adopt such frameworks, including the reduced integration costs with third-party software components, reduced overheads associated with maintaining common operating system level, compiler options, etc. In addition, by adopting a common framework, ISVs enable their customers to extend and customize their solutions through standard tools and methods, increasing the power of the solution while reducing the cost of customization and integration of the core simulation software products.

Based on the collective experiences of government and academic laboratories developing large sciences and defense applications in addition to the success of component architectures in modern business logic applications, it is clear that software frameworks offer a compelling approach to address the software development challenges described above. The next step is to identify the best technical approach. Section 2.1 of this paper defines different levels of component interoperability found in software frameworks. Section 2.2 cites relevant examples of software frameworks found in the community. Sections 2.3 and 2.4 compare software frameworks and recommend a strawman community framework approach.

2.1 Levels of Component Interoperability in Software Frameworks

The state of evolution of application physics software can be measured in terms of level of component interoperability, given in Figure 1. The level of interoperability is defined by the degree to which components must conform to a set of rules set by the framework in order to achieve interoperability. This paper refers to three levels of interoperability:

- ***Minimal Component Interoperability:*** A majority of the existing commercial solvers based on legacy codes can be described as having *minimal component interoperability.* Individual physics models are handled by separate solvers. Therefore, the physics domains are completely uncoupled; static analysis might be performed across physics domains using file translators or common interchange file formats, also referred to as workflow coupling. The framework requires nothing of the individual solvers except that they share the same file format.

- ***Shallow Component Interoperability:*** Several of the leading simulation software vendors have released simulation suites that are beginning to exhibit *shallow component interoperability.* At this level, physics models are loosely coupled at some time step or discrete event. Each solver maintains its own internal state representation of its respective domain. Common data is exchanged using wrappers to some interchange interface over a network service. Therefore, the development guidelines imposed by the framework stop at the interface to the component. The framework developer need only provide a standards-based interface that is external to the solver to achieve interoperability. In the commercial market, *shallow component interoperability* is usu-

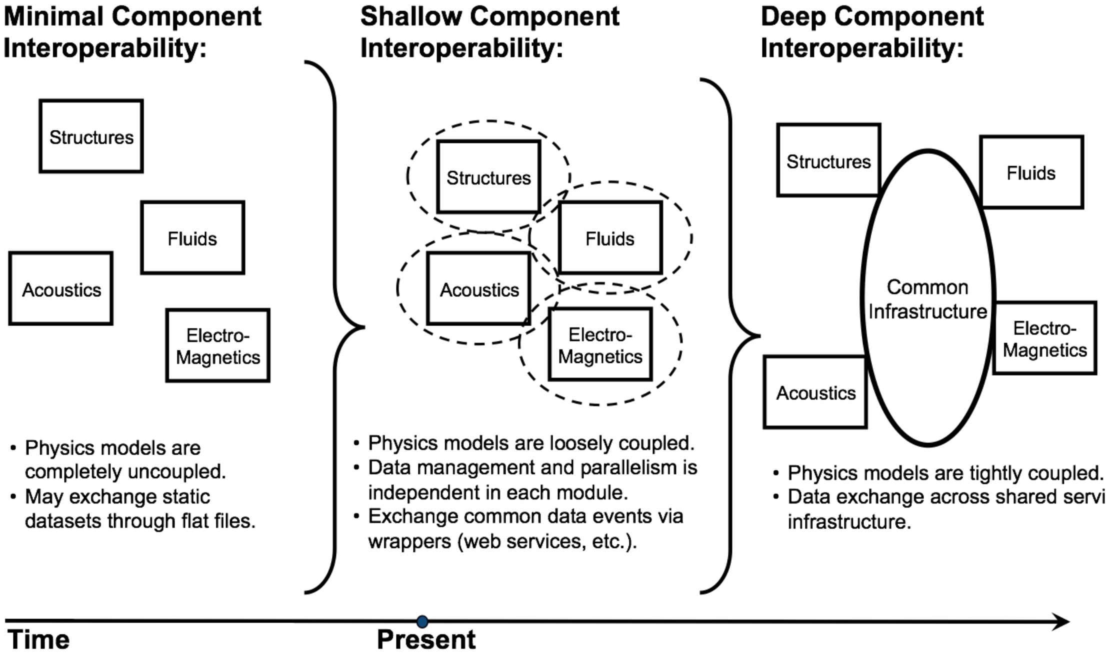

Figure 1. Evolution of Application Physics Software

ally limited within a single vendor's offerings, although some open interchange standards are beginning to emerge based on web services.

- ***Deep Component Interoperability:*** A few leading HPC laboratories have developed physics component frameworks where the solvers share a common service infrastructure for communications and data management. Physics models can be tightly coupled at this level of interoperability. In this case, the component developer must also heed rules regarding the internal organization of the component in order to achieve interoperability with the framework. This approach hides the complexity of the underlying hardware platform and offers higher-level abstractions for managing parallelism, thereby providing opportunities for improved platform portability and parallel system library optimization by the hardware vendors themselves.

2.2 Example Framework Architectures

The goal of the proposed consortium is to advance the application physics-based simulation market on a path towards *deep component interoperability*. The first step in this path is to support software vendor community adoption of a common set of standards for *shallow component interoperability*. The consortium would provide a neutral environment where commercial interests collaborate on *shallow component interoperability* standards that lower software development costs and provide an overall market benefit. The next step is for the emergent application physics software community to agree on a common infrastructure for *deep component interoperability* and possibly to develop an open reference implementation. Full-system simulation will require large, complex, tightly coupled multiphysics and multidisciplinary models only possible with *deep component interoperability*. The following sections discuss both *deep* and *shallow* interoperability frameworks and motivations for their use. They are meant to be instructive of the capabilities and architectures of frameworks rather than an endorsement of a specific framework.

2.2.1 Shallow Component Interoperability Framework Example: OMD-SA

The *Open Multi-Discipline - Simulation Architecture* (OMD-SA) framework [3] is an extensible and flexible Service Oriented Architecture (SOA) for scalable multidisciplinary engineering analysis that has been designed by MSC Software to support efficient data transfer for modular multiphysics simulations on HPC systems. Whereas older generation codes used data files to exchange model data between various solvers in a multiphysics application, the OMD-SA architecture enables direct transfers between the components as well as a composition system for combining solver components into a single application. As the simulation data can easily be in the gigabytes or even terabytes, the transfer of data across service layers must be optimized. Specifically, the framework must avoid unneeded copying or transfer of data if it is not absolutely necessary. Services in this framework can be either in a local application space in a remote application space. Local services should cost no more than a simple function call, i.e. the framework has negligible overhead on the overall performance. The framework supports language interoperability so that existing optimized code written in FORTRAN, C, or C++ can be used to implement services within the framework to support this capability. Remote services leverage standard and emerging network protocols to maximize performance. A single service can be used both locally and remotely, and it is up to the framework to determine the usage and the appropriate optimizations relevant to each case. Services must be highly tuned internally for efficient processing, using multi-threading, caching, efficient sharing of memory across services where possible, etc.

There are 4 main elements to the OMD-SA architecture:

- **Component Framework** – The component framework is an open SOA model where the services are available on-demand. The component framework is comprised of multiple layers. The services are connected through the Simulation Bus and a common data model that assures scalability and effective application of the services to a simulation application.
- **Simulation Clients** – The services are exposed to the various players in the simulation process

through different clients, both rich and thin, that address the specific user needs.

- **External Services** – External services are available to OMD-SA through standard open plug-in technology. Legacy applications, 3rd party applications, as well as in-house developed applications can be exposed as services to OMD-SA applications.
- **Enterprise Service Bus** – The Enterprise Service Bus can be either an existing ESB within an enterprise or a third-party ESB to which OMD-SA will interface. This allows for the use of external enterprise data and processes within a simulation process, i.e. using geometry from PDM within simulation. This allows for the use of simulation services and processes within external enterprise applications.

OMD-SA uses emerging standards for interface definitions and Internet protocols, including OMG-IDL (ISO standard 14750) and WDSL for service description and interface definition, UDDI for service discovery, and SOAP for service invocation/interaction. OMD-SA is an open platform for customers and partners to address extended or proprietary applications through the customizable service APIs, the SOA, and the programmable user interfaces.

OMD-SA is an example of a *shallow component interoperability* framework in terms of the interfaces it presents to developers. Integrating a component into this framework doesn't require any changes to the internal data model employed by the solver. Each component is able to share data with other physics solvers using the standards-based APIs to write data to the simulation bus in an operation that looks like writing a file to disk (as would be the case for older multiphysics simulations), but can reside in memory for local data exchanges between simulation clients. All parallelism remains internal to each component, which enables the solvers to be incorporated with little or no changes to their internal data model or data structures. While the shallow interoperability model simplifies coupling of components, the approach does not support any form of abstraction or modularity for the implementation of the parallelism.

2.2.2 *Shallow Component Interoperability Framework Example: FIPER*

The FIPER technology and software architecture designed by Engineous Software is a web-based distributed design and integration infrastructure that allows organizations to access, execute, and reuse design tools and processes. Design teams may exist as workgroups inside an organization or may be part of a geographically dispersed network of partners. The FIPER software architecture employs a loosely coupled, multi-discipline, and multi-view approach to describe its architectural frameworks, as required by ANSI/IEEE 1471-2000, OMG/UML 2.0 standards and the recommended best practices for software-intensive systems. At the very core, FIPER technology is defined as a set of software frameworks along four major services that represent scalable enterprise architecture for a high performance enterprise simulation integration and multidisciplinary design exploration platform:

- **Application Architecture Framework** is built on top of the FIPER system-level frameworks and utilizes the FIPER API classes (a.k.a. FIPER SDK) for developing FIPER models and components, local and remote model execution, which communicates with and consumes the management services of the FIPER ACS.
- **System-Level Frameworks** is composed of several helpful utilities and common services that provide access to system security and logging, data access, license management, exception handling, and FIPER results.
- **ACS (Application Control System) Framework** is based on the J2EE distributed standards and utilizes J2EE middleware services and frameworks. The ACS defines a consistent approach for reusing FIPER message protocol based on the JMS message provider services, its distributed objects, data objects, and service components. Key framework components are; distributed event system, workflow and distributed resource / job management, model execution, station and security services, and Web access services. The ACS also enables FIPER collaboration capabilities

such as component and model publishing and sharing via FIPER Library (a virtually centralized and physically distributed model and component repository), workgroup collaboration by reference or by copy for pre-built design processes, and Web Services B2B collaboration based on a SOA (Service-Oriented Architecture) solution.

- **System Architecture Framework** provides the tight cohesion and loose coupling between the FIPER applications, ACS, and its application integration components. The FIPER technology standards and framework provide several means to integrate other applications and their perspective data that can be realized via SOA Web Services, Portal and Portlets, Web Server and Servlets, Message Oriented Middleware, or Enterprise Java Beans frameworks. FIPER also transparently integrates with other grid computing systems such as LSF and PBS Pro for execution of complicated multidisciplinary and multi-objective computation workflows in a Grid environment. Integration with other Enterprise Systems (PDM, PLM, etc.) can be achieved via FIPER Enterprise Data Bus (an enterprise application integration framework) or via a standard Enterprise Service Bus integration solution.

In summary, FIPER technology provides several software frameworks for integration of modular, distributed simulation models from different domains into a consistent integrated model based on standardized loosely coupled distributed services. FIPER is a commercially supported system in production used by hundreds of companies worldwide ranging from aerospace, automotive, energy, to consumer goods.

2.2.3 *Deep Component Interoperability Framework Example: Cactus*

The Cactus Framework [4] is an open source, modular, portable programming environment for collaborative HPC computing. Cactus consists of both a programming model with a set of application-oriented APIs for parallel operations, management of grid variables, parameters etc, as well as a set of modular swappable tools implementing parallel drivers, coordinates, boundary conditions, elliptic solvers, interpolators, reduction operations, and efficient I/O. Although Cactus originated in the numerical relativity community where the largest HPC resources were required to model black holes and neutron stars, Cactus is now a general programming environment with application communities in computational fluid dynamics, coastal modeling, reservoir engineering, quantum gravity and others.

Cactus consists of four main elements:

- The Cactus **Flesh**, written in ANSI C, acts as the coordinating glue between modules that enables composition of the modules into full applications. Although the architecture is different, the Flesh plays the same role as the “Enterprise Service Bus” for the OMD-SA framework. The Flesh is independent of all modules, includes a rule based scheduler, parameter file parser, build system, and at run time holds information about the grid variables, parameters, methods in the modules and acts as a service library for modules.
- Cactus modules are termed **Thorns** and can be written in Fortran 77 or 90, C or C++. Each thorn is a separate library providing a standardized interface to some functionality. The “thorns” are similar in nature to the “Simulation Clients” in OMD-SA, but Cactus further externalizes the implementation of parallelism for the thorns, enabling different architecture-specific implementations of parallelism to be plugged in. Thorns providing the same interface are interchangeable and can be directly swapped. Each thorn contains four configuration files that specify the interface between the thorn and the Flesh or other thorns (variables, parameters, methods, scheduling and configuration details). These configuration files have a well-defined language and can thus be used as the basis for interoperability with other component based frameworks.
- **Drivers** are a specific class of Cactus Thorns that implement the model for parallelism. Each solver thorn is written to an abstract model for parallelism, but the Driver supplies the concrete implementation for the parallelism. For example, the PUGH (Parallel UniGrid Hierarchy) driver

implements MPI parallelism, whereas the ShMUGH (Shared Memory UniGrid Hierarchy) driver provides a shared memory/threaded implementation for the parallelism. The application can use different drivers without requiring any changes to the physics thorns. However, the thorns must be written specifically to the guidelines of the Cactus framework. The modular "drivers" for implementing parallelism are both the principle advantage of the deeply integrated framework model, but also the most daunting part due to the need to conform to framework coding requirements to take advantage of this capability.

- Cactus modules or thorns are grouped into **Toolkits**. Cactus is distributed with the Cactus Computational Toolkit that consists of a collection of thorns providing parallel drivers, boundary conditions, scalable I/O etc to support applications using multi-dimensional finite differencing. Community toolkits are provided or are under development by different application areas such as Numerical Relativity and Computational Fluid Dynamics.

The modular design of Cactus with swappable thorns provides several features important for this paper. Third-party libraries and packages can be used by applications through the abstract Cactus interfaces, decreasing application reliance on any particular package and making it possible to switch to new capabilities as they are available. For example, instead of using the UniGrid parallel driver PUGH distributed with Cactus, applications can use a variety of other independent adaptive mesh refinement drivers such as Carpet, PARAMESH, SAMRAI. New I/O methods can be added as thorns, and are then available to applications as a parameter file choice. Cactus currently supports a variety of output formats including HDF5, NetCDF, ASCII, JPEG, FlexIO, and provides architecture independent checkpoint and recovery along with interfaces for parameter steering and remote visualization.

Cactus has already been shown to scale to large processor numbers (4,000 to 33,000 cores) for different applications, and has active user and developer communities, along with funding from a range of agencies to both improve the infrastructure and build new application areas.

Whereas the shallow component interoperability framework enables modular composition of solver components into a multiphysics application, providing a scalable and modular model for parallelism requires deeper modifications to the code base. Deep component interoperability frameworks such as SIERRA and Cactus present an approach where the abstract model for parallel computation is external to each of the components. This requires a larger initial investment in code, but offers additional performance and scalability benefits down the road as systems move towards a massive parallelism on multicore systems.

2.2.4 Deep Component Interoperability Framework Example: SIERRA

Sierra is a software framework [5] which is used for multiphysics computational mechanics simulations – primarily targeting finite element and finite volume methods for solid mechanics, heat transfer, fluid dynamics with reacting chemistry, and multiphysics permutations of these mechanics. Sierra is designed around an in-core data model for supporting parallel, adaptive multiphysics on unstructured grids, with an emphasis of simultaneously handling parallelism, dynamic mesh modification, and multiple mesh solutions and transfer operations. Sierra also provides common services and interfaces for linear solver libraries, dynamic load balancing, file input parsing, and mesh file I/O. It was designed to unify and leverage a common base of computer science and data capabilities across a wide range of applications, and facilitate research, development and deployment of multiphysics capabilities, while managing the complexities of parallel distributed mesh data.

Through its solvers class capability and external interfaces, Sierra provides plug-in capability of a range of solver libraries for different mechanics. Plug-ins play the same role as the "thorns" in Cactus nomenclature and the "Simulation Clients" in OMD-SA. At the coupled physics level, Sierra provides a procedural language to support operator splitting methods to couple mechanics, including the ability to iterate to convergence and to sub-cycle physics modules relative to one another. The procedural language, called SolutionControl, allows a user to specify how the coupled mechanics for the various Sierra Regions are executed in sequence, how variables are mapped between the computational domains of each region, and how solution convergence is controlled at the coupling level before moving the simulation forward in time. SolutionControl is the basis for composing solver components into composite multiphysics applications, much as the OMD-SA scripting environment and Cactus "Flesh" is used to support module composition in those respective frameworks. Sierra also supports limited tighter coupling through forming full Jacobians for multiphysics within a single Sierra Mechanics Region.

Sierra's support for parallelism is pervasive, and is designed to limit the amount of work and complexity associated with parallel data structures for the mechanics developer, so that they can focus on the physics-relevant aspects of their solver module. Like Cactus, the implementation of the parallelism is externalized from each of the solver modules, so that the implementation of the parallelism need not be replicated for each module that comprises the framework. Supporting this capability requires the solvers to adopt some common data structures and conform to framework coding requirements, which is the hallmark of a deeply integrated framework.

2.3 Comparison of Framework Architectures

Examining both shallow and deeply integrated frameworks for modeling and simulation on parallel computing platforms, some common themes have emerged. Physics solvers in these frameworks are implemented as **modular software components** so they can support flexible reconfiguration for different multiphysics problems. The coupling of physics modules follows **step-level or loose coupling** as opposed to full coupling or workflow coupling. The framework provides a **flexible composition environment** that matches the requirements of the application domain. In addition to these common features, deep component interoperability frameworks also partition the implementation of parallelism into separate components, in other words abstracting the implementation of parallelism, to reduce programming errors and support performance optimization and portability across diverse hardware platforms. The key distinction between shallow and deep component interoperability frameworks is that shallow framework components manage their own parallelism and data structures and exchange data using external interfaces, whereas a deep framework components externalize the parallelism and data structures so that they can be optimized and ported independently from the solver component implementations.

Frameworks also provide a base set of services and build tools that simplify the customization of existing software components, and building and integration of new components within the framework. Examples of such services are I/O services, memory management services, error handling services, etc. As existing software modules are to be imported into a framework, their "outer layer" (a main program calling the subroutines) is "peeled off" and rewritten as declarations to the framework, which describe the high-level dataflow between the components. The framework manages the coarse-grain dataflow of an application, which is required for efficient parallelization. However, fine-grain dataflow within subroutines remains under control of the individual components and thus remains highly efficient.

The shallow integrated frameworks are attractive because they minimize the amount of code rewriting internal to each of the solver components. Each component interacts through a common SOA interface that preserves the opaqueness of the internal architecture of the component. However, such an architecture makes it difficult to impose constraints on the data layouts employed within each module, and therefore can lead to inefficient coupling between components due to the extra layer of data copying that must be employed between components with incompatible data layouts. It also limits the ability of a third party to innovate the implementation of parallelism for the components without getting inside of each module and rewriting the solver implementation. However, the shallow framework component model is well tested in enterprise applications and would require the least amount of effort for ISVs to cooperate. These shallow integration framework architectures consist of a few (tens) of components, each operating on large amounts of data for a significant amount of time. Overheads due to staging, invocation, load distribution etc. are amortized over the run time of the components' activity. One crucial advantage of shallow frameworks is that they arise naturally from pre-existing, independent, large software packages as the need for coupling arises.

The deeply integrated applications require that solvers agree upon an external data representation for the model data that is exchanged between solvers. This architecture also manages the parallelism external to the solvers. The framework then defines the optimal data layout that is common to all of the components, so as to minimize the amount of data recopying required to couple components together. In addition, the deeply integrated approach enables the implementation of parallelism to be separated from the solver components, so that innovations in parallelization methods (particularly for multicore processors) can be exploited by the solvers without requiring them to be rewritten. However, the price of such a deep level of integration is that existing solver components must all be rewritten to conform to the framework's restrictions. This requires a more significant initial investment and a deeper level of cooperation among ISVs, but can lead to a platform that is more scalable to future trends in concurrency.

Deep component interoperability framework architectures consist of many (hundreds) of smaller components, each invoked many times in parallel, operating only on small subsets of the overall data set, supervised by a framework driver layer. Efficiency is guaranteed by the driver layer's control over the data layout, which enables it to orchestrate calculations and relocate data as required. Examples of deeply interoperable framework architectures are Cactus, SIERRA, Chombo, and UPIC. The crucial advantage of deep component interoperability frameworks is the close yet efficient interaction since parallelization is handled by the driver layer potential, which allows for more accurate multiphysics simulation.

2.4 Community Framework Model

This paper is intended to facilitate a community dialogue to determine what level of interoperablity is appropriate to meet the needs of the multiphysics modeling and simulation tool market for the coming decade. The initial list of important multiphysics multi-effect "use cases" recommended at the HPC-ASC planning meeting included: structure/fluid, chemical/fluid, structure/fluid/acoustic, molecular/fluid, thermal/structural/fluid, electromagnetics, and electrical/thermal.

The design goals for the framework are:

- Support modular composition of multiphysics applications using components supplied from different vendors
- Enable scalable application performance with a minimum of solver code rewriting as the hardware industry moves towards multi-core architectures with massive parallelism.
- Reduce the software development costs associated with addressing the first two concerns.

Several different approaches to implementing a framework that meets these design goals have been presented. The choice of how to proceed will require additional community discussion. Finally, it is possible to pursue a framework-in-framework approach, where a loosely integrated outer framework consists of components which are themselves made up of applications based on a tightly integrated framework, allowing a phasing-in of tightly integrated framework components.

The tightly integrated framework architecture consists of several key components:

- A backbone which orchestrates the overall simulation, touching only metadata
- A driver layer providing the "heavy lifting", handling memory management and parallelism
- Regridding components which define or modify the data structures on which the simulation operates (structured, unstructured, AMR)
- I/O and visualization components which can move potentially large data amounts of data into or out of the simulation, using specialized APIs and hints to negotiate with the driver

- Physics components, operating on driver-defined subsets of the data
- Solver components, interacting closely with the driver layer
- Statistics components, collecting metadata and providing feedback (provenance, performance, progress monitoring)

Such structures can readily be identified in existing components of loosely integrated frameworks; in the transition from a loosely to a tightly integrated framework, these structures have to be extracted and, where appropriate, replaced with existing components of a tightly integrated framework.

3 Consortium Organization Models

The Market Potential of an application increases to the square of the number of other applications with which it is interoperable [8]. Achieving interoperability across independent software vendors (ISVs), system vendors, value-added resellers, and end users requires a community organization that supports close collaboration. There are many frameworks and standards that provide interoperability across commercial and open-source software, for example: Business Process Execution Language (BPEL), Web Service Invocation Framework (WSIF), Eclipse Modeling Framework (EMF), and many more. The management styles of these frameworks follow different organizational models according to the culture and objectives of the participants. This paper proposes three organizational models for consideration by potential consortium members and provides exemplars of each:

3.1 Focused Research Model

The *focused research model* is a non-profit organization that combines research contributions from consortium members for the purpose of funding long-term pre-competitive research, usually for the benefit of a single industry segment. Consortium members have input on the research projects that are funded and share the derived intellectual property. This approach has proven effective because it lowers research costs among individual members and focuses research on the success the technology areas that the members depend on. This model follows the precept that “a rising tide floats all boats”.

An exemplar of this model is the Semiconductor Research Corporation (SRC). It was founded in 1982 at the recommendation of the board of the Semiconductor Industry Association (SIA) to ensure continued research on advancing silicon technology and improved manufacturability of integrated circuits (ICs). SRC operates multiple research programs both in the U.S. and globally to provide competitive advantage to its members. SRC provides low-overhead research and related programs that meet the needs of the semiconductor industry for technology and talent. SRC maintains core competencies in contracting and management for research at universities with terms and conditions that assure members’ ability to use results with minimal risk of future encumbrances, including intellectual property protection of significant research results and mechanisms for timely transfer of research products. In addition, SRC manages student programs to maximize the flow of talent to the member companies. SRC obtains funding for its programs by recruitment and retention of member companies through appropriate fee algorithms for the relevant market segments and by leveraging government funding.

3.2 Commercial Fusion Model

The *commercial fusion model* is a for-profit strategic alliance of consortium members founded to sell a combined set of products and services tailored to a specific market. Consortium members provide intellectual properties and/or venture investment into the alliance company in return for a share of the profits. The fusion approach allows members to address a complex market that would otherwise be difficult to target individually. It also lowers marketing and support costs for each member to that market. An exemplar of the commercial fusion model is Fusion Petroleum Technologies.

Fusion Petroleum Technologies, Incorporated is an integrator of products and services for the petroleum exploration and production market. Fusion is a strategic partnership of hardware and software technology providers, engineering and analysis companies, and regional support organizations that provide their customers with a single portal for the services required by the petroleum market. These include technology services such as seismic imaging and analysis, inversion modeling, geopressure analysis, and petrophysical analysis as well as expertise services such as integrated interpretation, executive and legal services, reservoir studies, and technical training.

In 2007, Fusion released a software platform called LANGIAPPE™ that provides an integrated environment for the processing and analysis of seismic data for petroleum exploration and production. The platform operates on Linux clusters and interfaces to their visualization package called AMIGO™. LANGIAPPE includes an open development platform for Fusion clients and partners that wish to integrate their software into the unified workflow. Fusion supplies expert and developer tool kits that allow third party software to take advantage of the integration, parallelization, and visualization environments.

3.3 Open Source Foundation Model

The *open source foundation model* is a non-profit consortium founded to manage open-source software components on which commercial companies can build for-profit businesses. This model has become very popular with Unix system vendors. When the market rejected closed proprietary solutions, these companies survived by opening the source of core functionality and providing commercial products and support services using these frameworks. Open frameworks allow individuals and small companies to tailor the software for their needs and thereby extend the user base and software utility. A broader development base lowers software development and support costs incurred by the companies that use them. There are a number of examples of this model; one successful example is the Eclipse Foundation.

The Eclipse Foundation was established in 2004 as a not-for-profit corporation to provide a vendor neutral organization for collaboration by consortium software vendors developing integrated development tools. Eclipse provides an overall collaborative business model and governance system for organizing, publicizing, approving, and managing open-source projects. It is an overall platform for tool and utility integration with a common look and feel, mechanisms for interoperability, and an extensive plug-in technology. Eclipse has 60 open source projects that have technology embedded into over 1000 open-source and commercial products. The Eclipse Public License enables the use of open source software in commercial products and services. The Eclipse Public License is a non-viral license, allows redistribution, allows embedding, and is not specific to Eclipse projects. The structured IP approval process of Eclipse gives a level of confidence that components can be distributed in commercial products.

Eclipse provides a development ecosystem and robust community support. There is a well-defined process for project startup that uses community reviews, management through meritocracy, and formal annual releases. The foundation supports annual conventions (EclipseCon and Eclipse Summit Europe), online resource catalogs, marketing resources, magazines, and web portals. The foundation hosts code repositories, bug databases, mailing lists, wikis, and software distribution sites.

Eclipse membership levels are segmented by role and contribution in the community:

- ***Strategic Members*** are organizations that view Eclipse as a strategic platform and are investing developer resources. Annual membership fees range from $50K to $500K based on the number of in-kind developers contributing to the projects.
- ***Add-in Provider Members*** are organizations that view Eclipse as part of their corporate and product strategy and participate in the ecosystem. Annual membership fees are $5K.
- ***Associate Members*** are organizations that are non-profits, standards bodies, universities, and research institutes.
- ***Committer Members*** are individuals that are the core developers and make source code changes.

3.4 Community Feedback

Initial feedback from the HPC Application Software Consortium Planning meeting in December 2007 was that the *open source foundation* model was the most promising approach and that there are considerable advantages to joining an established development community such as the Eclipse Foundation as a special topics project rather than investing resources building similar infrastructure. Each of these approaches will be discussed at the HPC Application Software Consortium Summit.

4 Acknowledgements

The authors would like to acknowledge this paper's contributors and the planning workshop participants.

HPC-ACS Summit White Paper Contributors

- Gabrielle Allen, Louisiana State University
- Gene Allen, MSC Software
- Kenneth Alvin, Sandia National Laboratories
- Arman Atashi, Engeneous
- Matt Drahzal, IBM
- David Fisher, DoD HPC Modernization Program
- Merle Giles, NCSA
- Robert Graybill, USC Information Sciences Institute
- Bob Lucas, USC Information Sciences Institute
- Timothy Mattson, Intel
- Hal Morgan, Sandia National Laboratories
- Erik Schnetter, Louisiana State University
- Brian Schott, USC Information Sciences Institute
- Edward Seidel, Louisiana State University
- John Shalf, Lawrence Berkeley National Laboratory
- Shawn Shamsian, MSC Software
- David Skinner, Lawrence Berkeley National Laboratory
- Siu S. Tong, Engeneous

HPC-ACS Planning Workshop Participants

- Gene Allen, MSC Software
- Matt Drahzal, IBM
- Merle Giles, NCSA
- Robert Graybill, USC Information Sciences Institute
- Andy Haaland, ATK Launch Systems
- Ralph Jorstad, The Boeing Company
- Ramesh Krishnan, ATK Launch Systems
- Thomas J. Lange, Procter & Gamble
- Bob Lucas, USC Information Sciences Institute
- Hal Morgan, Sandia National Laboratories
- Danny Powell, NCSA
- Erik Schnetter, Louisiana State University
- Brian Schott, USC Information Sciences Institute
- John Shalf, Lawrence Berkeley National Laboratory
- Shawn Shamsian, MSC Software
- Joe Thompson, ATK Launch Systems
- Suzy Tichenor, Council on Competitiveness
- Siu S. Tong, Engeneous
- John Towns, NCSA

5 References

[1] Council on Competitiveness, *Innovate America: Thriving in a World of Challenge and Change* (2004), http://innovateamerica.org, 2004.

[2] Council on Competitiveness, *Survey of ISVs Serving the High Performance Computing Market – Part A: Current Market Dynamics and Part B: End User Perspectives,* http://www.compete.org/hpc, 2005.

[3] MSC Software, *SimEnterprise Extending Simulation to the Enterprise*, http://www.mscsoftware.com/assets/MSC_SimEnterprise_WP.pdf, 2007.

[4] E. Schnetter, C. D. Ott, G. Allen, P. Diener, T. Goodale, T. Radke, E. Seidel, and J. Shalf. Petascale Computing: Algorithms and Applications, chapter Cactus Framework: Black Holes to Gamma Ray Bursts. Chapman & Hall / CRC Press, Taylor and Francis Group, 2007. http://www.cactuscode.org

[5] H. Carter Edwards, *Software Environment for Developing Complex Multiphysics Applications,* Sandia National Laboratories, http://csmr.ca.sandia.gov/projects/ftalg/Edwards02.pdf.

[6] Dan Martin, "Solving Partial Differential Equations Using the Chombo Framework for Block-Structured Adaptive Mesh Refinement Algorithms," Grand Challenge Problems in Computational Astrophysics Workshop, March 2005. (http://seesar.lbl.gov/anag/chombo/index.html)

[7] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, K. Yelik. "The Landscape of Parallel Computing Research: A View from Berkeley". Technical report, EECS Department, University of California at Berkeley, UCB/EECS-2006-183, December 2006.

[8] M. Drahzal, IBM. "M.D. Law of Application Openness". February 3, 2008. Draft presentation for HPC Application Software Summit, March 2008.