SAND REPORT

SAND 2002-3783
Unlimited Release
Printed November 2002

On the Development of a Java-Based Tool for Multifidelity Modeling of Coupled Systems: LDRD Final Report

David R. Gardner, Joseph P. Castro, Mark A. Gonzales, Gary L. Hennigan, and Michael F. Young

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND 2002-3783
Unlimited Release
Printed November 2002

On the Development of a Java-Based Tool for Multifidelity Modeling of Coupled Systems: LDRD Final Report

David R. Gardner, Joseph P. Castro, and Gary L. Hennigan,
Computational Sciences Department

Mark A. Gonzales
Distributed Information Systems Department

Michael F. Young
Modeling and Analysis Department

Sandia National Laboratories
P. O. Box 5800, MS 0316
Albuquerque, New Mexico 87185-0316

November 14, 2002

Abstract

This report describes research and development of methods to couple vastly different subsystems and physical models and to encapsulate these methods in a Java™-based framework. The work described here focused on developing a capability to enable design engineers and safety analysts to perform multifidelity, multiphysics analyses more simply. In particular this report describes a multifidelity algorithm for thermal radiative heat transfer and illustrates its performance. Additionally, it describes a module-based computer software architecture that facilitates multifidelity, multiphysics simulations. The architecture is currently being used to develop an environment for modeling the effects of radiation on electronic circuits in support of the FY 2003 Hostile Environments Milestone for the Accelerated Strategic Computing Initiative.

Acknowledgments

This work was supported by the Laboratory Directed Research and Development (LDRD) program at Sandia National Laboratories.

The authors thank Russell Hooper, Ph.D. and J. Randall Weatherby, Ph.D. for comments and suggestions that improved this report.

Contents

Figures

Tables

Intentionally Blank Page

On the Development of a Java-Based Tool for Multifidelity Modeling of Coupled Systems: LDRD Final Report

1 Introduction

Engineers at Sandia National Laboratories simulate or test a wide variety of complex systems. These systems range from national infrastructure to nuclear power plants to weapons to micro-electrical-mechanical systems (MEMS) to living cells.

Historically, engineers at Sandia have relied on tests to characterize such complex systems. Political, economic, and environmental factors increasingly constrain the ability of engineers to conduct tests as they have in the past. Sandia engineers must now rely increasingly on modeling and simulation of these disparate complex systems.

These disparate, complex systems must often be modeled at different levels of fidelity. For example, risk assessment studies for nuclear reactors are conducted with fault tree models in which the component models are of very low fidelity (*e.g.*, binary decisions) [1]. In contrast, studies of systems such as neutron generators use the highest fidelity physics models available and tax the capabilities of the most powerful computers in the world [2].

As engineers rely more and more on modeling and simulation, they will also use more mixed-fidelity system simulations, *i.e.*, simulations that contain models of differing fidelities. For example, a drift-diffusion model of a transistor might be used in a lumped-parameter circuit model to more accurately model the transistor behavior in a radiation environment.

Significant effort has been invested in developing high-fidelity models for parts of nuclear weapons (*e.g.*, the neutron generator). However, comparatively little effort has been invested in developing computer modeling tools to examine interactions among weapon subsystems. Computer tools for modeling complex subsystem interactions in a nuclear weapon in varied environments and over a wide range of time scales (from nanoseconds to years) at multiple levels of fidelity can enable designers and safety analysts to perform their jobs "faster, better, and cheaper" [3].

The goal of the work described here was to develop methods to couple vastly different subsystems and physical models and to encapsulate these methods within a framework accessed by a Java™-based interface. This framework is called the *Entero* system engineering environment.

In the next section we discuss concepts in multifidelity modeling. Then we discuss our work to develop mixed-fidelity system models for thermal radiation heat transfer. Next we describe the architecture for the *Entero* system engineering environment and the prototype environment for multifidelity thermal/electrical modeling. We conclude with a summary and a description of our current work.

2 Multifidelity Modeling Concepts

As engineers rely more and more on modeling and simulation, they will also use more multifidelity simulations. In this report we use the term *multifidelity model* to mean a system model in which the fidelity of one or more of its components can be changed. Thus at least one component in the model can be represented by at least two models, each with a different fidelity. For example, a transistor in an electrical circuit might be represented by either a behavioral model or by a drift-diffusion model. Sometimes an analyst analyzes the system using one model or the other for the transistor.

To emphasize cases where the fidelities of the component models used in a system model are different—for example, a drift-diffusion model of a transistor embedded in a model in which all the other components are represented by behavioral models—we will sometimes use the term *mixed-fidelity* model or simulation.

Such mixed-fidelity models have several advantages. They enable a component design to be evaluated in the context of a full system, and allow more realistic boundary conditions for the model of the component. They enable more rapid system-level analysis and optimization, because changes to the higher fidelity model can be incorporated directly into the system model without constructing an equivalent lower fidelity model. They enable the uncertainty in knowledge of a component to be reflected in the fidelity of the model used for the component, independent of the fidelities of models used for other components. And the resolution and fidelity of the simulation can be tailored to the requirements of the analysis, using lower fidelity models for exploratory studies and hence making better use of computing and personnel resources [4, 5].

Combat Modeling

Much of the exploration of mixed-fidelity modeling has occurred in the context of combat modeling [6]. In this context, *variable-resolution* models are models or families of models in which users can change the resolution at which phenomena are treated [7, 8, 9, 10, 11], and *cross-resolution* modeling is linking models with different resolutions [7, 8, 10, 12, 13].

In combat modeling, a model with higher resolution has more components or more details than one with lower resolution. For example, a lower resolution model of a battalion of tanks might describe the battalion as a whole by the number of tanks and some average location of the battalion, while a higher resolution model might describe the motions of the individual tanks.

Davis and Bigelow note that *resolution* is often defined simply as "the level of detail at which system components and their behaviors are depicted" [6]. In the context of combat modeling, this definition for resolution is ambiguous because it can be applied to many different features of the model, including the spatial scale, the temporal scale, the physical processes included, the number of objects, the number of attributes of each object, and the degree of interaction between objects.

One objective for combat modeling is to develop the ability to use *multiresolution* models. *Multiresolution modeling* is building a single model or a family of models with alternative user modes involving different levels of resolution for the same phenomena [6, 14, 15].

In combat modeling involving models with different resolutions, lower resolution entities must interact with entities with higher resolution. When a higher resolution entity must interact with a lower level entity, one model or the other must be changed so that they can interact at the same level of resolution. For example, consider a single tank (a higher resolution entity) interacting with a battalion of tanks treated as a single entity (a lower resolution entity). The process of dynamically changing model resolution is called *disaggregation* if the resolution of the model for the lower resolution entity is increased in resolution to correspond to the resolution of the model for the higher resolution entity. The process of dynamically changing model resolution is called *aggregation* if the resolution of the model for the higher resolution entity is decreased in resolution to correspond to the resolution of the model for the lower resolution entity. The problem of linking simulations at different resolutions is called the aggregation-disaggregation problem [13, 12, 13, 15, 16, 17, 18, 19].

The process of creating a lower resolution model from a higher resolution one is called *model abstraction* [4, 15, 20, 21, 22, 23, 24, 25, 26]. This is an active area of research.

An important issue in multiresolution modeling is whether a lower resolution model and a higher level model are *consistent.* A lower resolution model and a higher resolution model are said to be *weakly consistent* if the projection of the state of the higher resolution model to the space of the lower resolution model is sufficiently close to the state of the lower resolution model. For example, a three-dimensional thermal model might be weakly consistent with a zero-dimensional thermal model if the average temperature of the three-dimensional model is within a specified tolerance of the temperature of the zero-dimensional model.

A lower resolution model and a higher resolution model are said to be *strongly consistent* if the projection of the state of the lower resolution model to the space of the higher resolution model is sufficiently close to the state of the higher resolution model. For example, a zero-dimensional thermal model might be strongly consistent with a three-dimensional thermal model if the single temperature of the zero-dimensional model when projected to the space of the three-dimensional model is within a specified tolerance of the temperature field of the three-dimensional model.

Strong consistency is much rarer than weak consistency. Whether the model states are "sufficiently close" depends on the modeling perspective. For example, it may be sufficient for a graph generated from the lower resolution model to indicate the same trends as a graph generated from the projected state of the higher resolution model [6, 27].

A higher resolution model is often inferred to be more accurate than a lower resolution model. However, the addition of more detail does not necessarily improve accuracy [6, 20]. And in practice, lower resolution models are used frequently and successfully. For example, many engineering systems such as bridges and automobiles are designed and built using classical mechanics rather than relativistic mechanics, even though the latter is more accurate.

Numerical Zooming

The term *numerical zooming* has also been used to describe simulations in which one component of the system has a higher fidelity than the others, *i.e.*, mixed-fidelity modeling. For example, Reed and Afjeh used a three-dimensional, Navier-Stokes model of a fan to compute performance maps for zero-dimensional thermodynamic component models in a turbofan engine simulation [28, 29, 30]. Follen and auBouchon have implemented mixed-fidelity modeling in the National Cycle Program of the National Propulsion System Simulation by inserting one-dimensional compressor models in a zero-dimensional model of a turbofan engine [5].

3 Issues in Multifidelity Modeling

A variety of issues arise in multifidelity modeling. These include integrating models with differing dimensionalities (*e.g.*, integrating a zero-dimensional thermal model with a three-dimensional, finite-element thermal model), integrating models with differing spatial resolutions, integrating physics with differing time scales, integrating models with differing time scales, and verification and validation of multifidelity models.

Integrating models with differing spatial dimensionalities, which is the focus of this work, presents significant difficulties in determining the appropriate method of projection to use from the lower dimensional model to the higher dimensional model and from the higher dimensional model to the lower dimensional model. Various types of averages may be used to project the higher dimensional variable field to the lower dimensional one. For example, a three-dimensional temperature field represented by nodal values on a finite-element mesh can be projected to a zero-dimensional model using a surface or volume average of the temperature (we explore the use of both average temperatures in this work). To project a zero-dimensional temperature to a three-dimensional temperature field on a finite-element mesh requires applying the single temperature to the boundary of the three-dimensional mesh, or a portion of the boundary. Such projections produce unknown errors in the resulting solutions.

Integrating models with different spatial resolutions presents significant difficulties, even when the models have the same spatial dimension. For example, abrupt changes in mesh resolution in shock-wave physics simulations can result in non-physical reflections when waves traverse a region of changing mesh [31, p. 237]. Variable meshes can also result in increased error in the solution, rather than decreased error [31, pp. 288–290]. In addition, for explicit codes, the time step for the integrated models is controlled by the Courant-Friedrichs-Levy limit for the smallest mesh cell [31, p. 5]. Thus variable spatial resolution may reduce the required computer memory (because fewer cells are used), but it may not reduce the simulation execution time. Therefore, care must be exercised in integrating models with different spatial resolution.

Different physical phenomena are characterized by different time scales. For example, chemical reactions in a fluid flow frequently occur very rapidly compared to the time scale of the flow of the fluid. For the problems the *Entero* system engineering environment is designed to address, time scales may range from nanoseconds for nuclear reactions to years for aging effects. Numerical models with both a shorter time scale and a longer time scale are said to be *stiff*, because the numerical computation of the solution on the longer time scale is strongly affected by the presence of the shorter time-scale component. Integrating models with differing time scales must be performed carefully to ameliorate problems with stiffness.

Models of different fidelity may have differing time steps. For example, a lower fidelity model may have a different (probably greater) time scale than a corresponding higher fidelity model. For multifidelity models, the time step sizes must be coordinated, which imposes an additional constraint on the system model.

Verifying and validating mixed-fidelity system models is a difficult problem. There are few analytic solutions for mixed-fidelity models, so that verifying the models may rely on verified higher-fidelity simulations. Mixed-fidelity experiments are difficult to perform, so validating mixed-fidelity models directly against experimental data is problematic. It may be that the only method for validating mixed-fidelity models is to validate them against verified and validated higher fidelity simulations.

4 A Multifidelity Algorithm for Thermal Radiation

We now describe a multifidelity algorithm for thermal radiation. In the algorithm, a zero-dimensional model of the system controls a simulation. This model consists of a set of software objects that identify and manage the model or models used for physical phenomena in the actual component, and a set of objects that identify and manage the information exchange between the objects.

Description of the Multifidelity Algorithm

In the algorithm, a zero-dimensional model of the system controls a simulation. At the zero-dimensional level, the physical state of a component is represented by a set of scalar variables that represent an appropriate average for the component (*e.g.*, temperature or pressure). Physical connections between components are represented by information transfer through the interconnection objects, which are called *ports*. For example, consider a system consisting of two solid bodies exchanging energy by thermal radiation. Each body has an average temperature assigned to it. The port describing the interaction records the temperature of each body and indicates that the energy transfer is via thermal radiation.

The zero-dimensional model of a component may be augmented by an alternate model. This is done by connecting the zero-dimensional model to the alternate model with a special port, called an N-Port, that transforms information (Figure 1). For information transferred from the zero-dimensional model to the alternate model, the N-Port transforms the zero-dimensional state variables to those used by the alternate model. For information transferred from the alternate model, the N-Port transforms the state variables from those used by the alternate model to the scalar state variables used by the zero-dimensional model. (The Port connection between zero-dimensional models is a special case of the N-Port, the 0-Port.)

As an example, consider a system of two components that are interacting via thermal radiation. The zero-dimensional temperature of each component is an average temperature. Suppose that the temperature of one of the components is modeled with a three-dimensional finite-element model. This model is connected to the zero-dimensional model for the component by an N-Port. For transferring information from the zero-dimensional model to the three-dimensional model, the N-Port maps the single temperature of the zero-dimensional model to a portion of the finite-element mesh surface visible to the zero-dimensional component. For transferring information from the three-dimensional model to the zero-dimensional model, the N-Port computes an average temperature (either a surface-averaged temperature or a volume-averaged temperature) from the three-dimensional model to send to the zero-dimensional model. (This resembles the approach used in the National Cycle

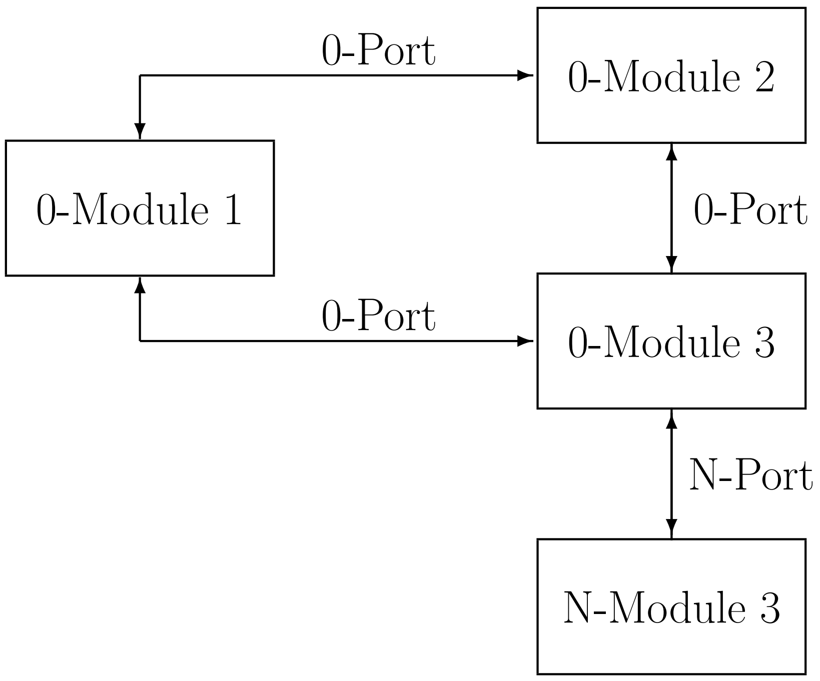


Figure 1. Illustration of Mixed-Fidelity System Models. Three zero-dimensional components are connected via 0-Ports into a system. Module 3 is represented by a higher fidelity component, that is connected to the zero-dimensional one via an N-Port.

Program [5].)

The algorithm consists of the following steps:

1. Identify the components in the system and their physical interconnections. This identifies the zero-dimensional model for the system.
2. Identify the alternate models to be used for system components.
3. Initialize state variables of the zero-dimensional models from the state variables of the alternate models or using specified values.
4. Increment the time variable and advance the state of zero-dimensional system model to the new time.
5. For the alternate models, transform the zero-dimensional state variable values to the values needed by the alternate models as boundary conditions.
6. Advance the state of the alternate model to the current time.
7. Transform the variables for the alternate model to the state variables needed by the zero-dimensional system model.
8. Repeat steps 4 through 7 until the final problem time has been reached.

Implementation of the Multifidelity Algorithm

The zero-dimensional system model is called the *integrator* in the *Entero* architecture. The *Entero* integrator for thermal radiation problems is described in [32]. The current version of the integrator allows one interior module in a system to be replaced by a finite-element model. Reference temperatures for the finite-element model are mapped to the external boundaries of the finite-element mesh.

We developed a special version of the Coyote heat transfer code [33, 34] for the mixed-fidelity modeling. We modified the user-defined subroutine `USRTRR` to provide the reference temperatures via Parallel Virtual Machine (PVM) messages [35] from the zero-dimensional models. We wrote subroutines to compute the surface-average temperature and volume-average temperature of a mesh block. We provide further details of the implementation of the algorithm in [36].

Linking Electronics Modeling to Modules in the System

The integrator also provides the option of embedding an electronic circuit in any of the modules defining the system. The user specifies the circuit using a standard Spice netlist file [37], and specifies the location of the circuit in the module through the input file using a construct called a *thermal pin*, which is a point in the module.

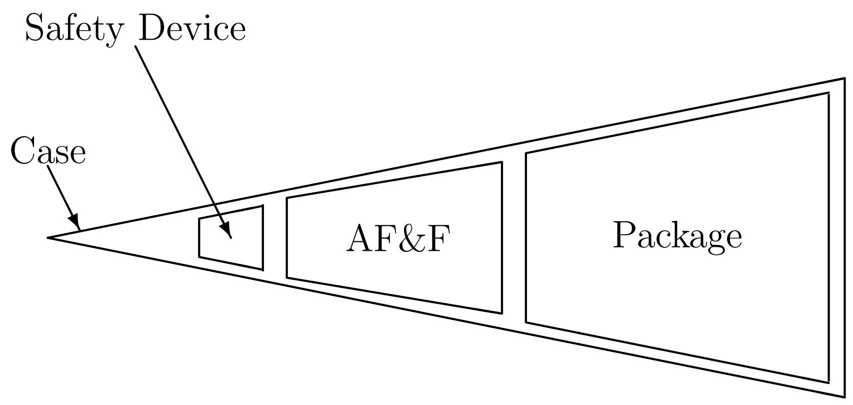


Figure 2. The Test System. The test system consists of four component modules plus the environment.

A zero-dimensional model can have only one thermal pin and its temperature value is the temperature of the model. This pin is automatically defined when the electronics model is linked to the thermal model.

For a higher-dimensional Coyote model, the default number of pins is one, with the temperature value of the surface node-averaged temperature. The user can specify thermal pins at specific locations in the finite element mesh by using the Coyote special points, which are user-defined locations in the mesh. In the case where a thermal pin is specified using a special point, the temperature of the thermal pin is the temperature computed for the special point.

Circuit modeling is performed with the Xyce™ parallel circuit simulator [38].

Tests of the Multifidelity Algorithm

We tested the algorithm on a system consisting of five modules: The environment, the case, the package, the arming, fuzing, and firing module (AF&F), and the safety device. The geometry for the test system is shown schematically in Figure 2. Geometric dimensions for the system and properties for the module materials are given in [36].

We compared predicted temperatures from three models of the test system: a full three-dimensional model (denoted the *3D* model) as the system baseline, a mixed-fidelity system model (denoted the *MD* model), and a zero-dimensional system model (denoted the *0D* model).

In the 0D model, each component was represented by a zero-dimensional model for temperature, that is, each component had a single temperature (Figure 3).

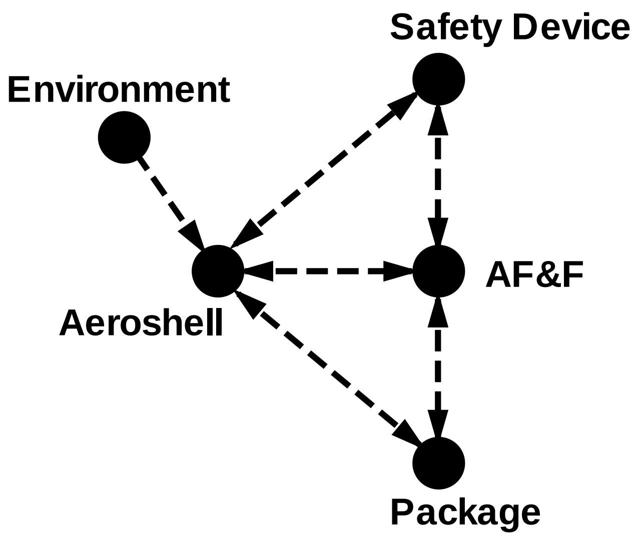


Figure 3. The Zero-Dimensional Test System (0D Model). The arrows indicate explicit information exchange.

In the MD model, the AF&F was represented by a three-dimensional finite-element mesh. The mesh was composed of 1083 nodes and 852 hexahedral elements (Figure 4).

In the 3D model, each component was represented by a finite-element mesh (Figure 5). The package, AF&F, and safety device were represented by hexahedral meshes; the mesh for the AF&F was the same mesh used in the mixed-fidelity model. Shell elements were used for the aeroshell. The three-dimensional system model had 8585 nodes and 7629 elements.

We used three test problems for assessing the performance of the algorithm: a heating problem, a cooling problem, and a problem with a time-dependent thermal radiation boundary condition.

We assessed the performance of the algorithm based on comparisons of surface-average temperatures, volume-average temperatures, minimum temperatures, maximum temperatures, and temperatures at selected special points inside the AF&F.

In order to connect a zero-dimensional model to a three-dimensional finite-element model, the temperature field from the latter must be transformed to a single temperature. Two obvious candidates for this temperature are the surface-average temperature and the volume-average temperature. Based on energy conservation, one might expect the volume-average temperature to give reasonable results. However, radiative heat transfer depends on surface temperatures, so one might also expect the surface-average temperature to give reasonable results. In this report we present representative results from using only the surface-average temperature. We present

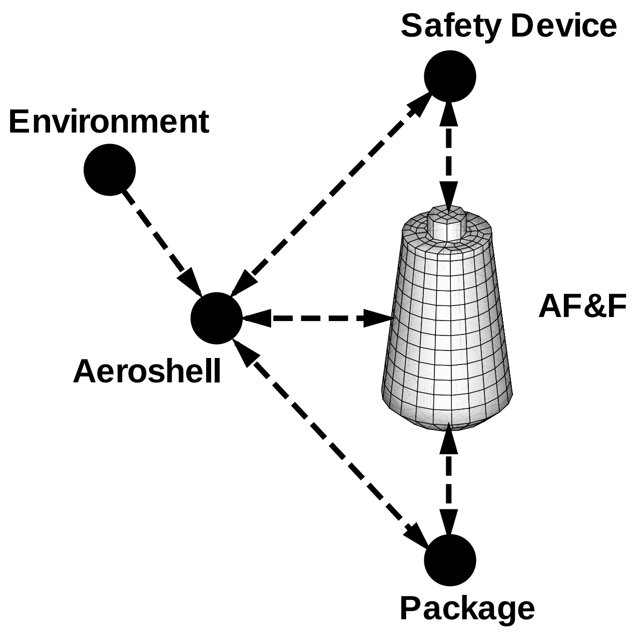


Figure 4. The Mixed-Fidelity Test System (MD Model). The arrows indicate explicit information exchange.

a discussion of the results of using both the surface-average temperature and the volume-average temperature in [36].

The Heating Problem

In the heating problem, the environment was a constant temperature of 1033 K and the system had an initial temperature of 300 K. We consider results from the case in which the surface-average temperature was used in the mixed-fidelity system model (MD model) to connect the three-dimensional AF&F to the otherwise zero-dimensional system.

Figure 6 shows the surface-average temperatures predicted for the AF&F by the three system models for the heating problem. Initially the surface-average temperatures in the AF&F, in the 3D and MD models, are essentially identical and increase more rapidly than the surface-average temperature of the AF&F in the 0D model. This is a consequence of the finite thermal conductivity in the AF&F in the MD and 3D models. The temperature of the entire volume of the AF&F in the 0D model must increase uniformly, while in the 3D and MD models the thermal energy is initially confined near the surface of the AF&F by the finite thermal conductivity, resulting in higher surface-average temperatures early in the simulation. Because the package

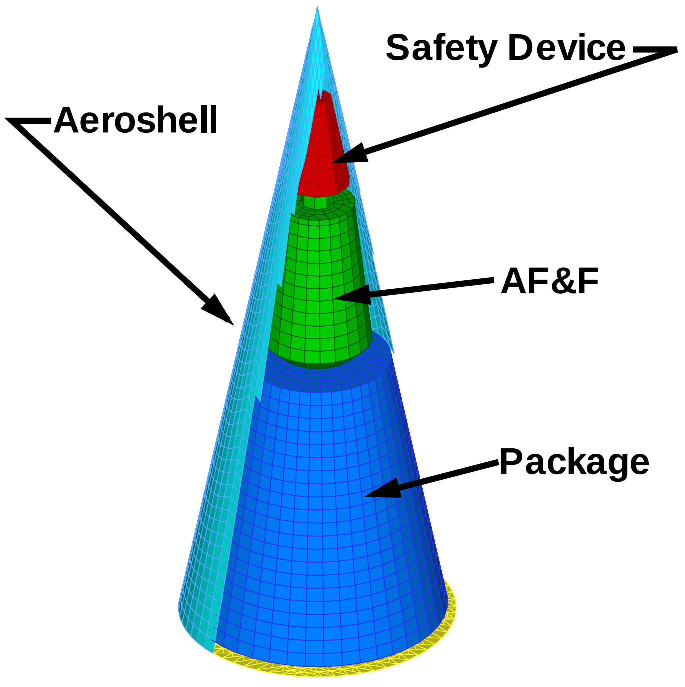


Figure 5. The Three-Dimensional Test System (3D Model).

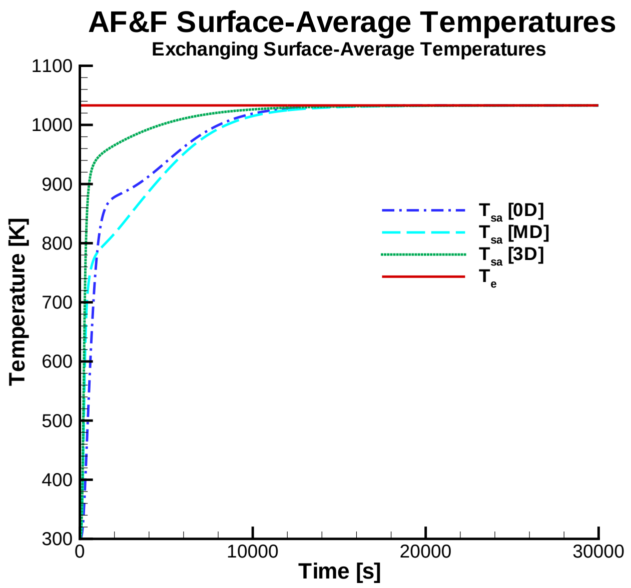


Figure 6. Surface-Average Temperatures for AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

must also heat uniformly in the 0D model, it acts as a large thermal sink, further retarding the rate of temperature increase in the AF&F.

Although the surface-average temperature of the 0D model increases more slowly than the surface-average temperature of the 3D model, this results in a greater rate of heat transfer to the AF&F in the 0D model compared to the 3D model. The volume-average temperature of the AF&F in the 0D model in fact increases more rapidly than the volume-average temperature in the 3D model (Figure 7). Again, these differences result from the finite thermal conductivity in the AF&F in the 3D model.

As in the 0D model, the rate of increase in the surface-average temperature of the AF&F in the MD model is decreased by the effect of the package module, which acts as a large thermal sink. This effect is visible in the temperature contours shown in Figure 8, which shows contours on a cross section containing the axis of the AF&F at 3000 seconds (a time when the difference between the surface-average temperatures of the MD and 3D models is large).

The volume-average temperature for the AF&F in the MD model increases more slowly than the volume-average temperatures for the AF&F in either the 0D or 3D models owing to the finite thermal conductivity in the AF&F in the MD model (which slows the transfer of heat to the interior) and the influence of the package as a large

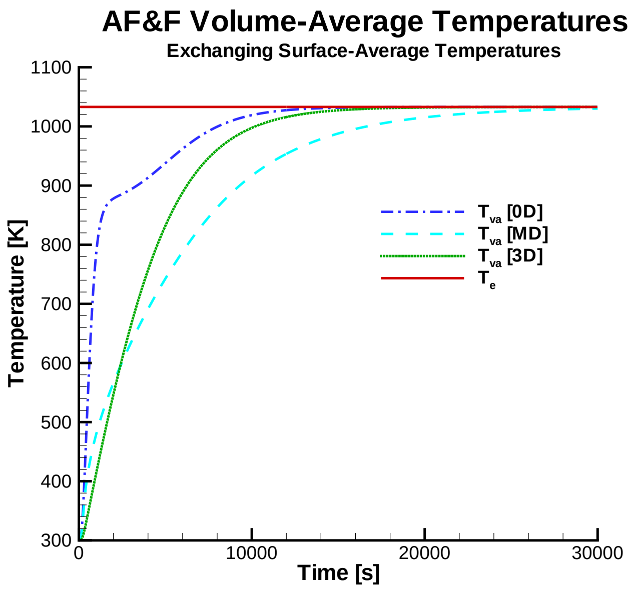


Figure 7. Volume-Average Temperatures for the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

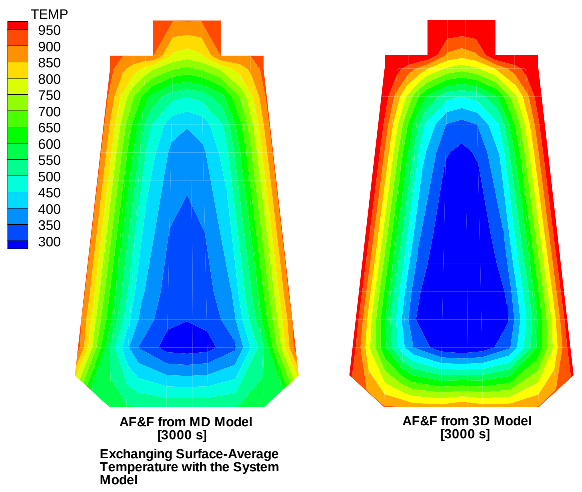


Figure 8. Comparison of Internal Temperatures for the AF&F Predicted by the MD and 3D Models for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model) at 3000 s. Left: The MD Model. Right: The 3D Model.

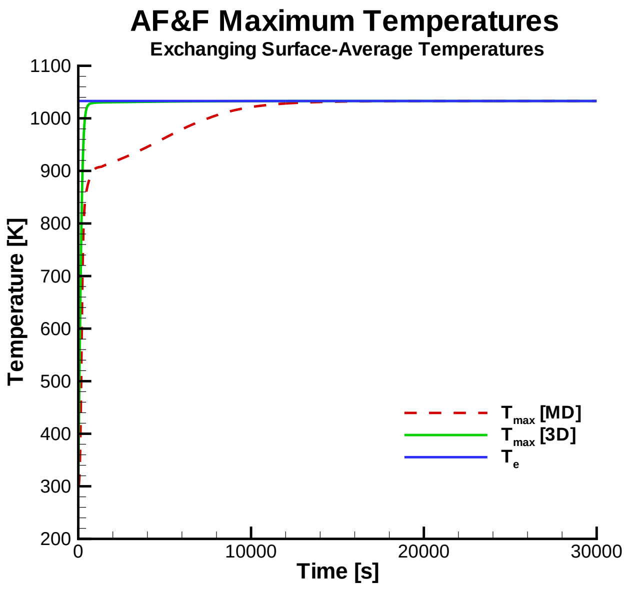


Figure 9. Maximum Temperature in the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

thermal sink (so that there is a net thermal energy transfer from the AF&F to the package) (Figure 7).

We show the maximum and minimum temperatures in the AF&F for the three system models in Figures 9 and 10, respectively. Except at very early times and later times, the maximum AF&F temperatures predicted by the MD and 3D models are significantly different (Figure 9). The maximum temperature of the AF&F in the MD model is lower than the maximum temperature in the AF&F in the 3D model because in the former model the package acts as a large thermal sink and moderates the maximum temperature. In contrast, the minimum temperatures follow similar trends and are reasonably close (Figure 10) because they are not significantly influenced by the thermal sink.

The temperature histories at three thermal pins in the AF&F are shown in Figures 12–14. The pins were distributed along the vertical axis of the AF&F (Figure 11). The temperatures predicted by the MD model at Thermal Pin 2 are close to those predicted by the 3D model at that pin (Figure 13). This is because near the center of the AF&F, the moderating thermal effect of the package in the MD model is diminished. However, the temperatures predicted by the MD model at Thermal Pin 1 are not good approximations to the temperatures predicted by the 3D model at that pin (Figure 12). And the temperatures predicted by the MD model at Thermal Pin 3 are not good approximations to the temperatures predicted by the 3D model at that pin.

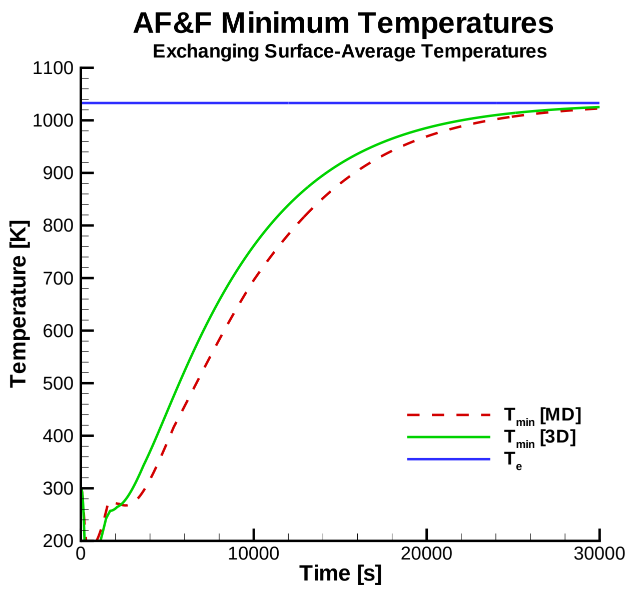


Figure 10. Minimum Temperature in the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

Temperatures at pins 1 and 3 are more strongly affected by the surrounding modules.

The Cooling Problem

In the cooling problem, the environment was a constant temperature of 300 K and the system initially had a temperature of 1033 K. We consider results from the case in which the surface-average temperature was used in the mixed-fidelity system model to connect the three-dimensional AF&F to the zero-dimensional system.

Figure 15 shows the surface-average temperatures and Figure 16 shows the volume-average temperatures predicted for the AF&F by the three system models for the cooling problem, in which the surface-average temperature from the three-dimensional AF&F module is sent to the zero-dimensional system model in the MD model. Because the cooling of the AF&F is controlled by radiation to the surrounding bodies, which are all represented by zero-dimensional models in the MD system model, the surface-average temperatures for the AF&F in the MD model are close to those for the AF&F in the 0D model (Figure 15). However, owing to the finite thermal conductivity of the AF&F material in the MD model, the volume-average temperature of the AF&F in the MD model is similar to the volume-average temperature of the

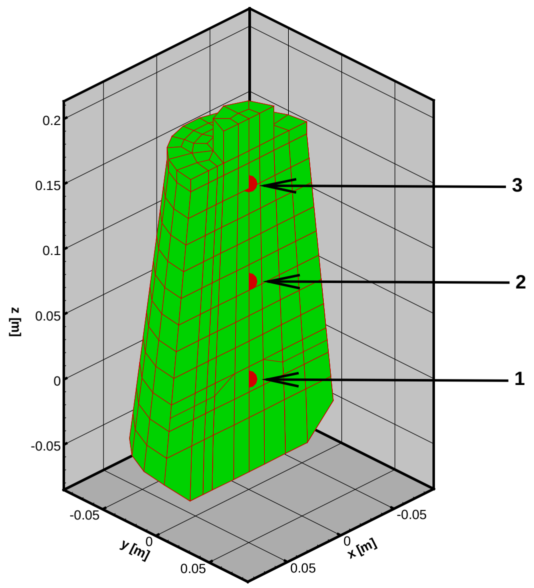


Figure 11. Locations of Thermal Pins in the AF&F.

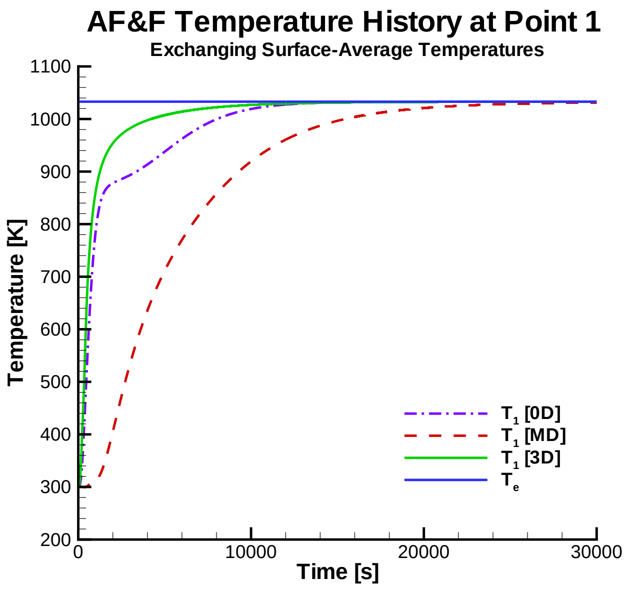


Figure 12. Temperature Histories for Thermal Pin 1 in the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

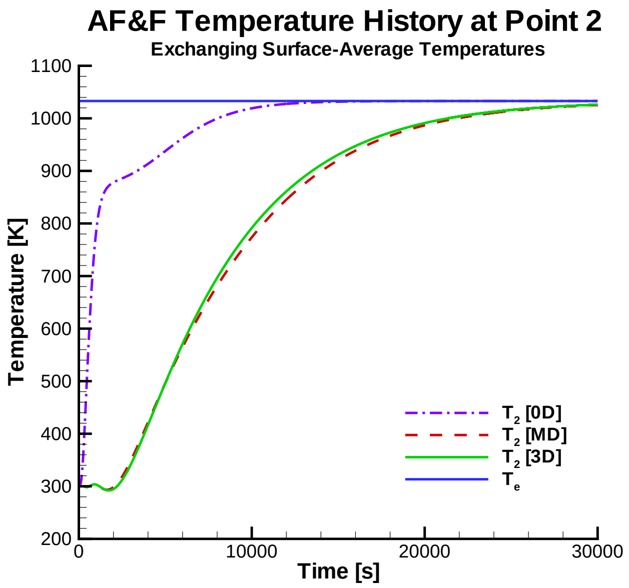


Figure 13. Temperature Histories for Thermal Pin 2 in the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Model).

AF&F in the 3D model (Figure 16).

Temperature contours for the AF&F in the cooling problem for the MD and 3D models at 3000 seconds are shown in Figure 17. The influence of the package as a relative heat source in the MD model (analogous to its influence as a relative heat sink in the heating problem) is seen in the contours.

Figures 18 and 19 show the maximum and minimum temperatures, respectively, for the AF&F. Although the temperature trends are correct, neither the maximum temperatures nor the minimum temperatures predicted by the MD model are good approximations to the corresponding temperatures predicted by the 3D model. For this problem, the maximum temperature of the AF&F occurs in its interior in both the MD and 3D models, and so the maximum temperatures in both models are similar (refer to the discussion above). However, the minimum temperature of the AF&F occurs on its outer surface, and so the minimum temperature of the AF&F in the MD model is different than the minimum temperature of the AF&F in the 3D model (again, refer to the discussion above).

Figures 20, 21, and 22 show the temperature histories at the thermal pins in the AF&F (see Figure 11). The AF&F in the 0D model cools uniformly and its temperature is uniform. Owing to its finite thermal conductivity, the AF&F in the MD model initially cools like the AF&F in the 3D model, but later behaves thermally

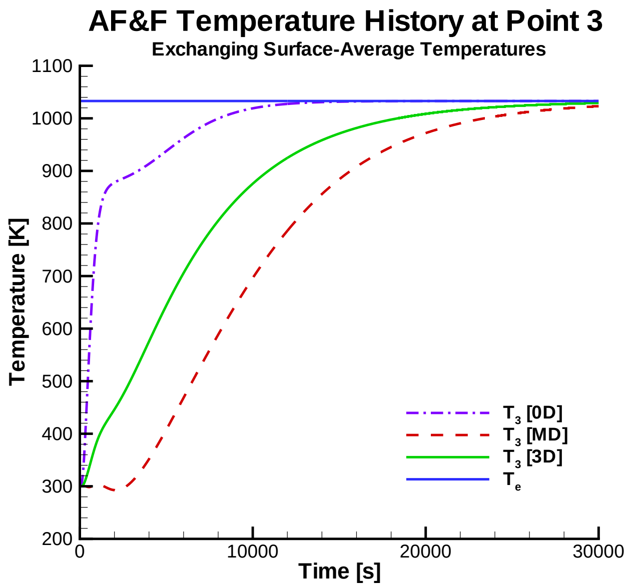


Figure 14. Temperature Histories for Thermal Pin 3 in the AF&F for the Heating Problem (Exchanging Surface-Average Temperatures in the MD Mode).

more like the AF&F in the 0D model because it is embedded in a system of zero-dimensional models. Although the trends of the temperatures are correct, none of the temperature histories at the special points in the MD model are especially good approximations to the temperature histories predicted by the 3D model at the thermal pins.

The Time-Dependent Thermal Radiation Boundary Condition Problem

In the time-dependent thermal radiation boundary condition problem, the environment represents a simulated engulfing fire in which the environment temperature rises rapidly to 1033 K from 300 K, remains at this temperature for a period of time, and then decreases to 300 K.

Figure 23 shows the surface-average temperatures and Figure 24 shows the volume-average temperatures predicted for the AF&F by the three system models for the simulated-fire environment, for the case in which the surface-average temperature from the three-dimensional AF&F module is sent to the otherwise zero-dimensional system model in the MD model.

Prior to 10,000 s (when the fire is "extinguished"), the temperature histories for the 0D, MD, and 3D models are, not surprisingly, similar to those for the heating

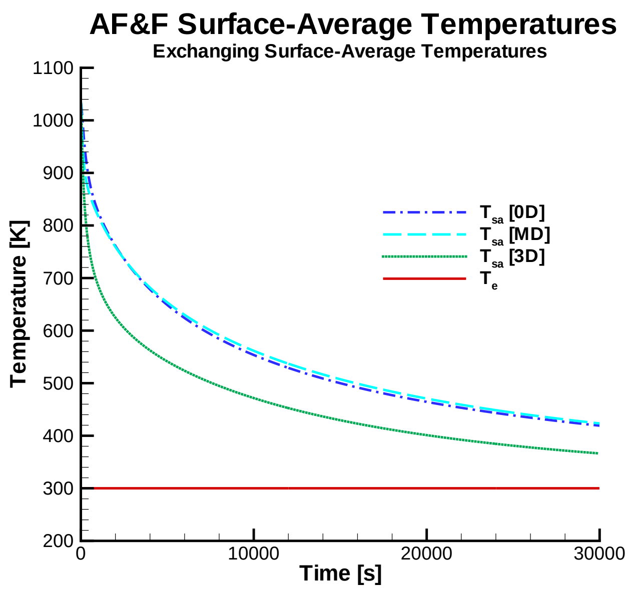


Figure 15. AF&F Surface-Average Temperatures for the AF&F in the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

problem (compare the surface-average temperatures in Figures 6 and 23, and the volume-average temperatures in Figures 7 and 24), because the heating problem and the time-dependent boundary condition problem are essentially the same during this time when the heating period is relatively brief.

The system begins to cool at 10,000 seconds. The cooling is sufficiently rapid that the temperature histories for the 0D, MD, and 3D models are similar to those for the cooling problem (compare the surface-average temperatures in Figures 23 and 15 and the volume-average temperatures in Figures 24 and 16).

Timing Results

In Table 4 we give timing results for the three cases summarized above. We ran all the calculations on a single processor of a Dell workstation with a 1.3 GHz Intel Pentium 4 processor, and each simulation produced the output files considered typical for the simulation. For example, the mixed-fidelity and three-dimensional system models each produced an output file containing mesh temperatures.

Referring to Table 4, the MD model ran over 25 times faster than the 3D model on the Dell workstation (in under three minutes compared to over an hour) and

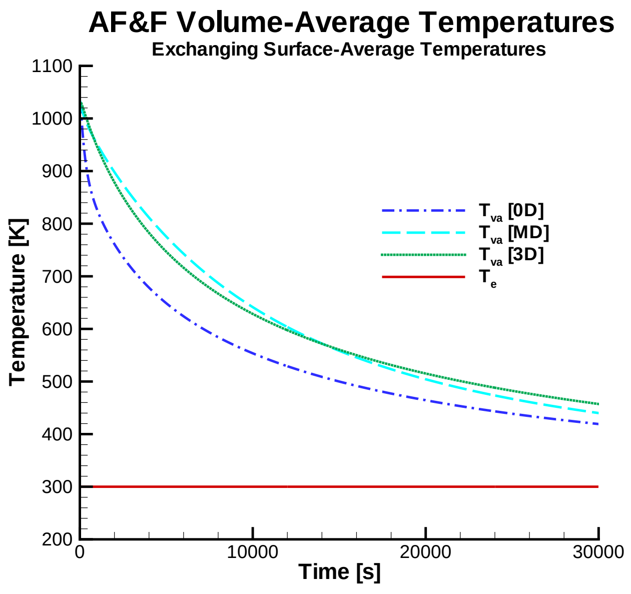


Figure 16. AF&F Volume-Average Temperatures for the AF&F in the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

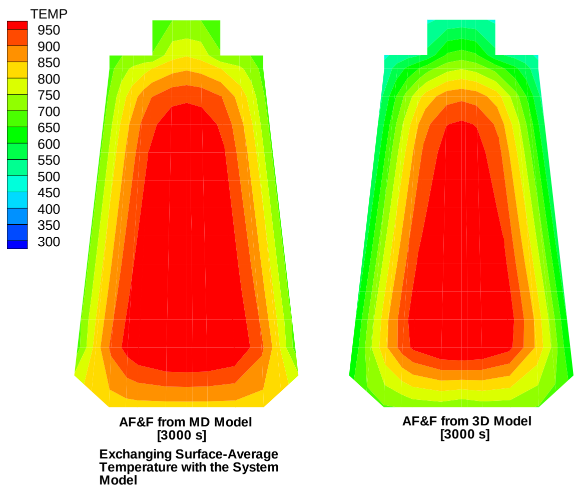


Figure 17. Comparison of Internal Temperatures for the AF&F for the MD and 3D Models for the Cooling Problem, Exchanging Surface-Average Temperatures in the MD Model, at 3000 s. Left: The MD Model. Right: The 3D Model.

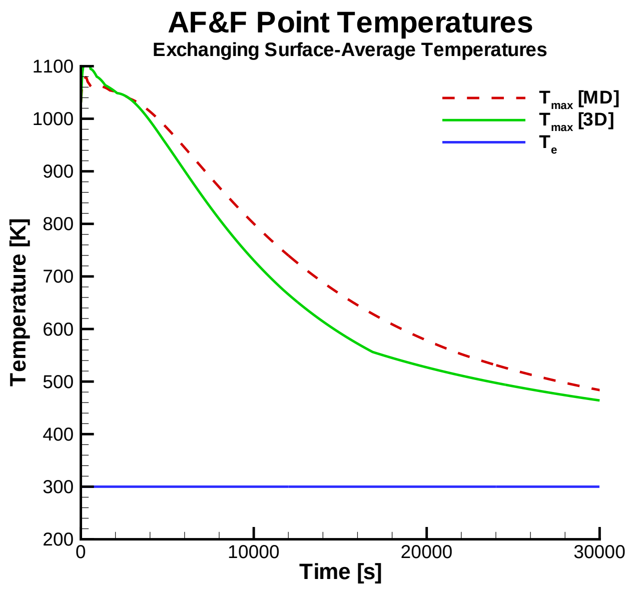


Figure 18. Maximum Temperature in the AF&F for the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

Table 1. CPU Times for the Test Cases.

	System Model				
	0D Only	Mixed-Fidelity			3D[a] Only
		3D Model	0D Model	Total	
Case	[s]	[s]	[s]	[s]	[s]
Heating	0.22	170.10	0.58	170.68	3931.
Cooling	0.31	171.45	0.85	172.30	3904.
"Fire"	0.24	168.55	0.97	169.52	3904.

[a] Does not include time to calculate the view factors (35.1 hrs).

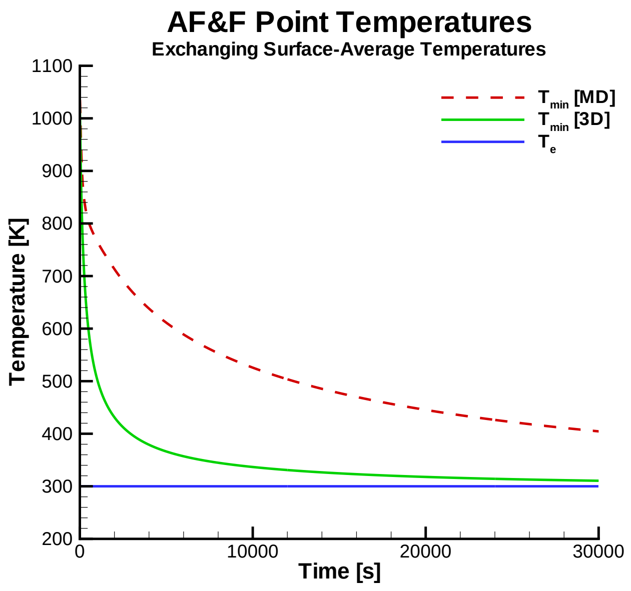


Figure 19. Minimum Temperature in the AF&F for the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

provides internal temperatures for the higher fidelity module. These results indicate that optimization studies with mixed-fidelity models is feasible when it may not be feasible with three-dimensional system models.

Results from a demonstration of the mixed-fidelity thermal radiation algorithm coupled with electrical modeling are shown in Figure 26. For this demonstration, we positioned the LM-185 circuit at Thermal Pin 1 (Figure 11) and imposed the "simulated fire" environment on the system. The LM-185 circuit is designed to provide a constant reference voltage and contains a temperature-sensitive device. The figure shows the temperature of the environment, the surface-average temperature of the AF&F, and the temperature of the thermal pin. The figure also shows the output voltage of the LM-185 circuit as the temperature of its thermal pin increases.

Extension of the Multifidelity Algorithm to Thermal Conduction

We designed an extension of the algorithm to conductive heat transfer, but did not test it. The concept is to connect two modules in a system with a construct called a "thermal wire" that is one-dimensional: conduction occurs only along its length, and its cross-sectional area as a function of length is specified. This construct can be

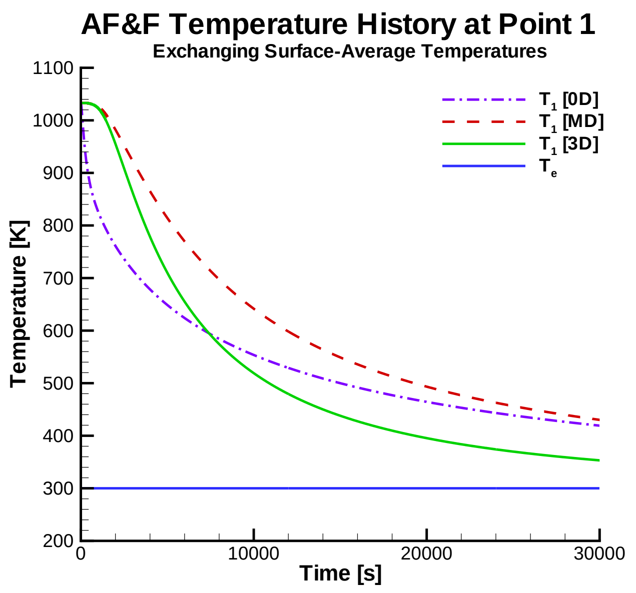


Figure 20. Temperature History for Thermal Pin 1 in the AF&F for the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

connected to a three-dimensional finite-element mesh using either the surface-average temperature or an average surface temperature for the area of contact. The latter connection could be implemented using a flux boundary condition on an element side set, and might provide increased accuracy in the temperature predictions.

5 The *Entero* System Engineering Environment

We now briefly describe the *Entero* system engineering environment; further description is provided in [39, 40, 41, 42]. First we present the design goals for the environment. Then we describe a prototype environment for coupled thermal-electrical modeling. Next we discuss some limitations of the software architecture of the prototype, and describe an improved architecture that we developed to ameliorate these limitations and to support the mixed-fidelity algorithms described in the previous section. Then we describe an improved *Entero* environment for coupled thermal-electrical modeling that we implemented in the new architecture.

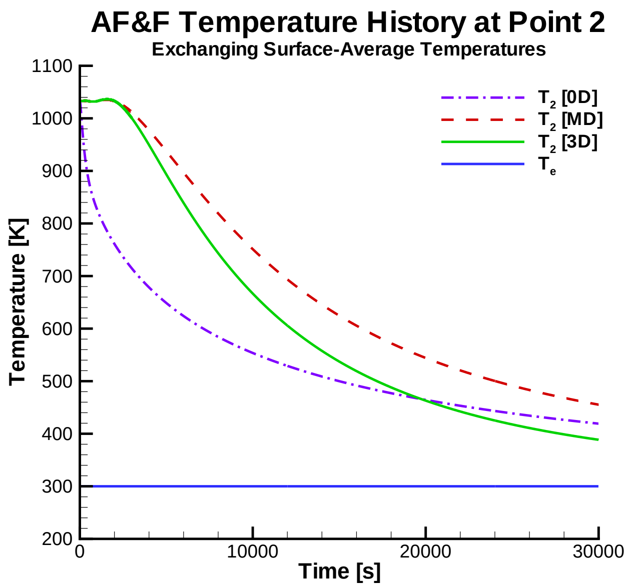


Figure 21. Temperature History for Thermal Pin 2 in the AF&F for the Cooling Problem (Exchanging Surface-Average Temperatures in the MD Model).

Entero Design Goals

The long-term goal for the *Entero* environment is to research and develop a module-oriented, multiphysics, mixed-fidelity system simulation environment for engineers to enable rapid system performance analysis and design optimization. Major design goals for the *Entero* environment include providing a systems view of the system to be analyzed, providing a module-oriented view of the system, enabling models of different physics types to be coupled together, providing mixed-fidelity models for the analysis, and enabling design optimization and uncertainty quantification studies.

Thus the *Entero* environment will represent the physical system to be analyzed as a whole, from a system level. In addition, the environment will represent the system as a collection of interacting modules, so that engineers can assemble systems in ways that reflect the physical assembly of the systems.

The *Entero* environment will enable models of different physics types to be loosely coupled together. For example, an engineer will be able to model an electrical circuit in a thermal or radiation environment, and monitor its performance.

The *Entero* environment will incorporate mixed-fidelity modeling, so that engineers can select model fidelity for each component and easily change it. For example, an engineer will be able to easily replace a coarser finite element mesh with a finer

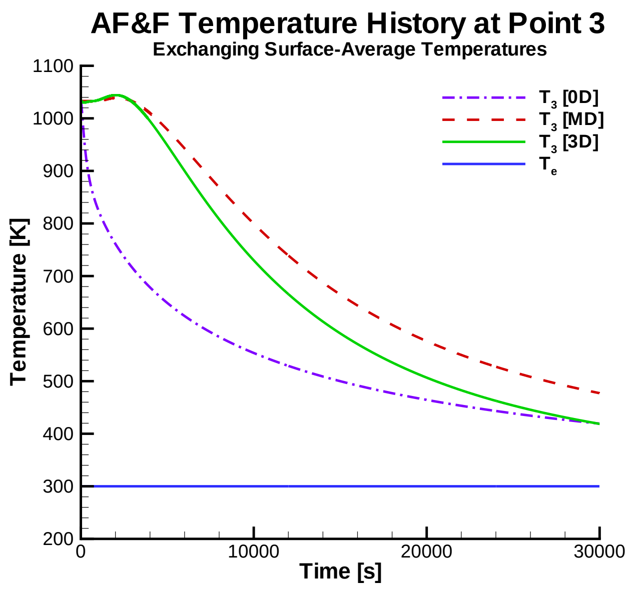


Figure 22. Temperature History for Thermal Pin 3 in the AF&F for the Cooling Problem, (Exchanging Surface-Average Temperatures in the MD Model).

one, or a linear model with a nonlinear model. Users will be able to select from zero-dimensional (lumped-parameter) models to the highest fidelity numerical models available.

And the *Entero* environment will enable engineers to optimize designs, and to quantify uncertainties in system performance due to variability in environmental conditions, material properties, part specifications, and modeling assumptions.

A Prototype *Entero* Environment for Coupled Thermal-Electrical Modeling

As described in the previous section, one focus of our development of the *Entero* environment has been modeling systems containing electrical circuits that are exposed to fires. If the electrical circuits fail in a fire, they must fail so that the system remains safe (although the system itself may be destroyed). This problem motivated the development of our prototype environment.

The development goal for the *Entero* prototype environment was to build and demonstrate an environment for coupled thermal-electrical simulations using zero-dimensional thermal models.

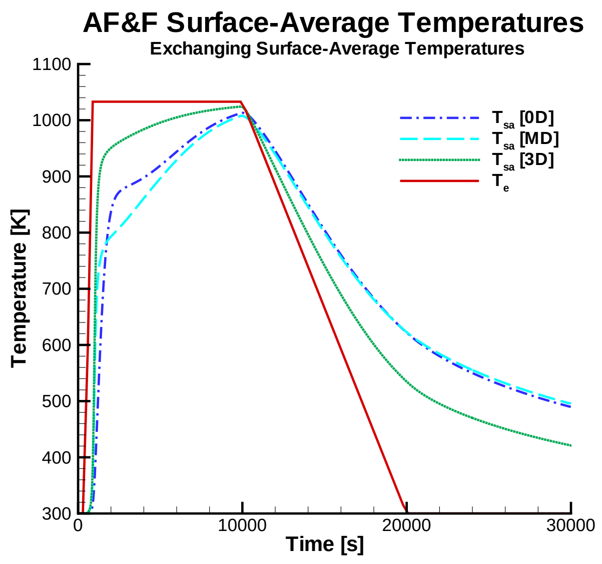


Figure 23. Surface-Average Temperatures for the AF&F for the Time-Dependent Thermal Boundary Condition Problem (Exchanging Surface-Average Temperatures in the MD Model).

In the following paragraphs we describe the software architecture for the prototype, and the preprocessor, the library of modules, and the physics manager. We then describe an improved architecture that we developed to ameliorate limitations in the prototype architecture and to support the mixed-fidelity algorithms described in the previous section. Then we describe an improved *Entero* environment for coupled thermal-electrical modeling that we implemented in the new architecture.

A high-level view of the software architecture for the prototype is shown in Figure 27. This figure emphasizes the three functional levels in the architecture: the specification level, which provides the graphical user interface; the interpretation level, which translates the specification into software; and the analysis level, which executes the physics analysis.

Users interact with the *Entero* environment in the specification level. Systems are assembled graphically in the visual editor using icons. Each icon represents a software module that in turn represents a physical component.

The visual editor is part of the preprocessor, which functions as both the graphical user interface and the model interpreter in the prototype. The preprocessor enables a user to display a graphical view of a system consisting of multiple modules. Users select modules from a library and connect them into a system using special modules called *ports*. External boundary conditions are also implemented as modules. Users

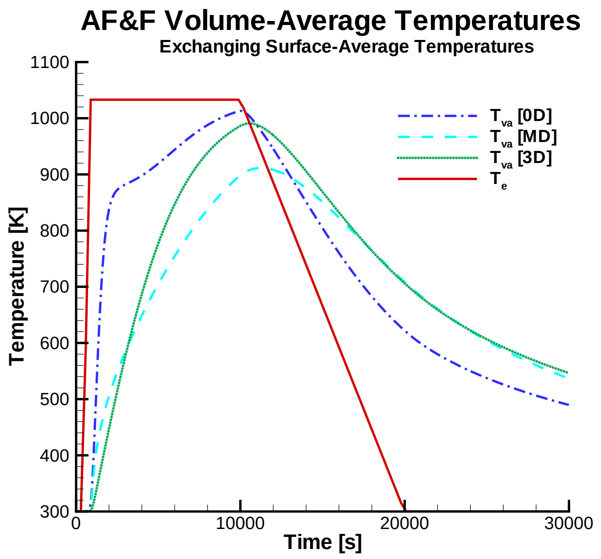


Figure 24. Volume-Average Temperatures for the AF&F for the Time-Dependent Boundary Condition Problem (Exchanging Surface-Average Temperatures in the MD Model).

can save the assembled system and reload it later. Users can set module properties such as heat capacity and initial temperature using the interface.

The software modules are stored in a library in the interpretation level (Figure 27). The model interpreter portion of the preprocessor links the software modules into a system that represents the physical system using ports and passes the system specification to the physics manager.

In the *Entero* prototype, a *module* is a software component that represents a physical component. It has properties that are stored internally and physics that may be modeled with one or more external applications. For example, in the AF&F module, material density is a property that is stored internally, while temperature is a property that is computed using an external application. Modules were developed by experts in the relevant problem domain. Six different modules were implemented in the prototype modules library.

Four of the modules represent physical components: a case or aeroshell, a package, an AF&F, and a safety device. For each component, a user selects the desired physics type, such as thermal radiation and electrical activity, and the fidelity of each physics model (only zero-dimensional models were available in the prototype).

The two other modules in the library are the `Environment` and the `Port` modules.

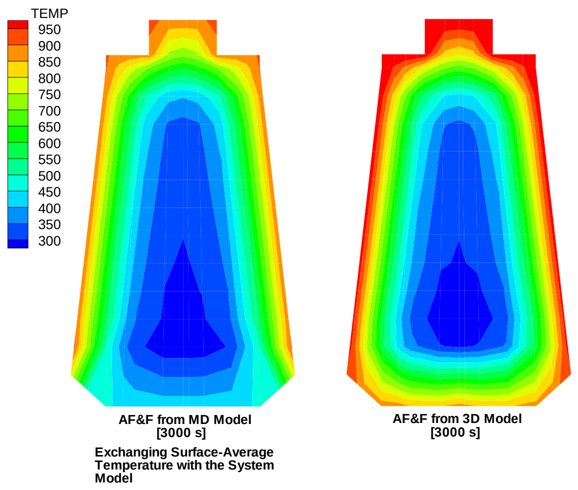


Figure 25. Comparison of Internal Temperatures for the AF&F for the MD and 3D Models for the Time-Dependent Thermal Radiation Boundary Condition Problem (Exchanging Surface-Average Temperatures in the MD Model) at 3000 s. Left: The MD Model. Right: The 3D Model.

The `Environment` module allows a user to specify a time-dependent external temperature. The `Port` module describes the physical coupling between the component modules, *e.g.*, radiative heat transfer.

Each module in the library is a `JavaBean` and extends a generic module, which then extends a Java `JComponent`. The *Entero* module uses a type of `BeanInfo` class, which extends the Java `SimpleBeanInfo` class. A customizer is specified in the `BeanInfo` file of each module. The customizer extends a generic customizer, which, in turn, extends a Java `JPanel` (Figure 28).

Engineering analysis occurs in the analysis level (Figure 27), by external or custom application programs. Numerical results are then displayed at the specification level through a post-processing interface to external or local tools.

In the analysis level, the physics manager advances the state of the model system from the initial state to the state at the final, user-specified, time. The thermal behavior of each component is modeled with a zero-dimensional model, *i.e.*, each module was assumed to have a single, time-dependent temperature. The temperature for each module is determined by conservation of energy assuming that each module radiates and absorbs thermal energy as a black body.

View factors for the model system are pre-calculated using the CHAPARRAL

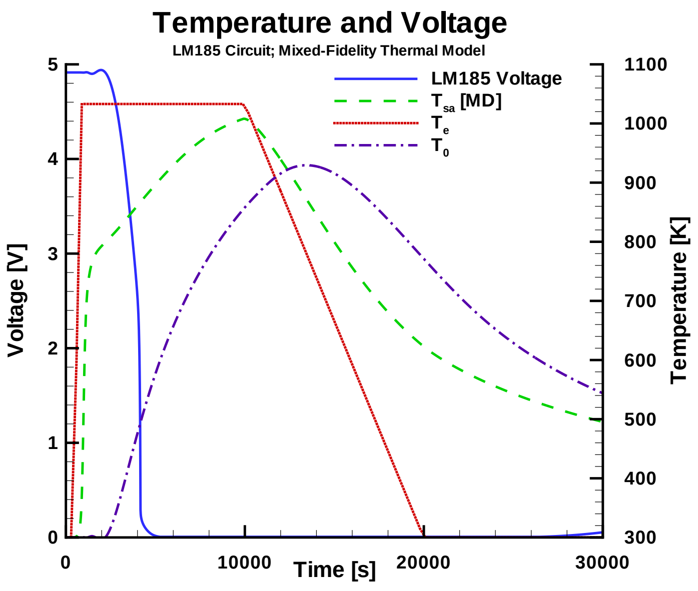


Figure 26. Output Voltages for the LM-185 Circuit (Time-Dependent Boundary Condition). T_e is the temperature of the environment. T_0 is the temperature of the thermal pin at which the circuit was located. T_{sa} is the surface-average temperature of the AF&F in the mixed-fidelity model (MD).

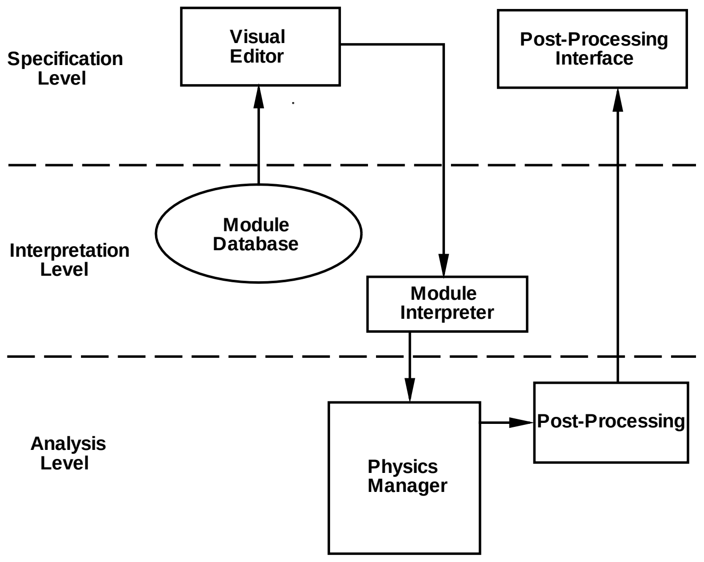


Figure 27. Model-Integrated View of the Prototype *Entero* Environment. The architecture is divided into three functional levels.

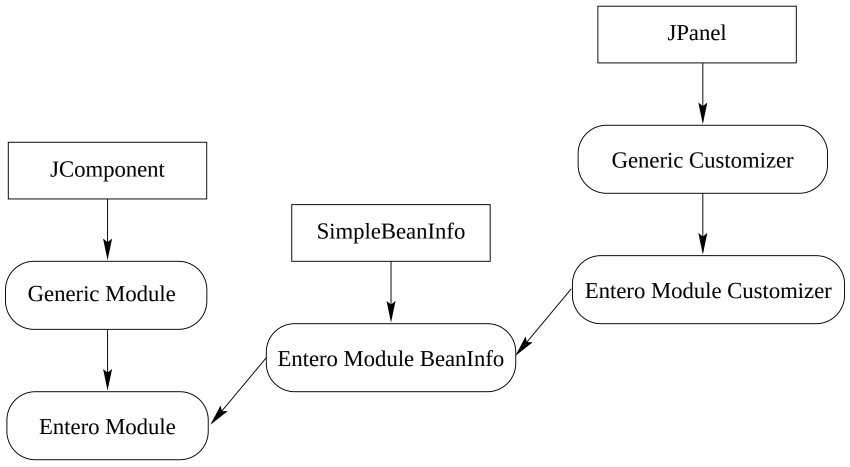
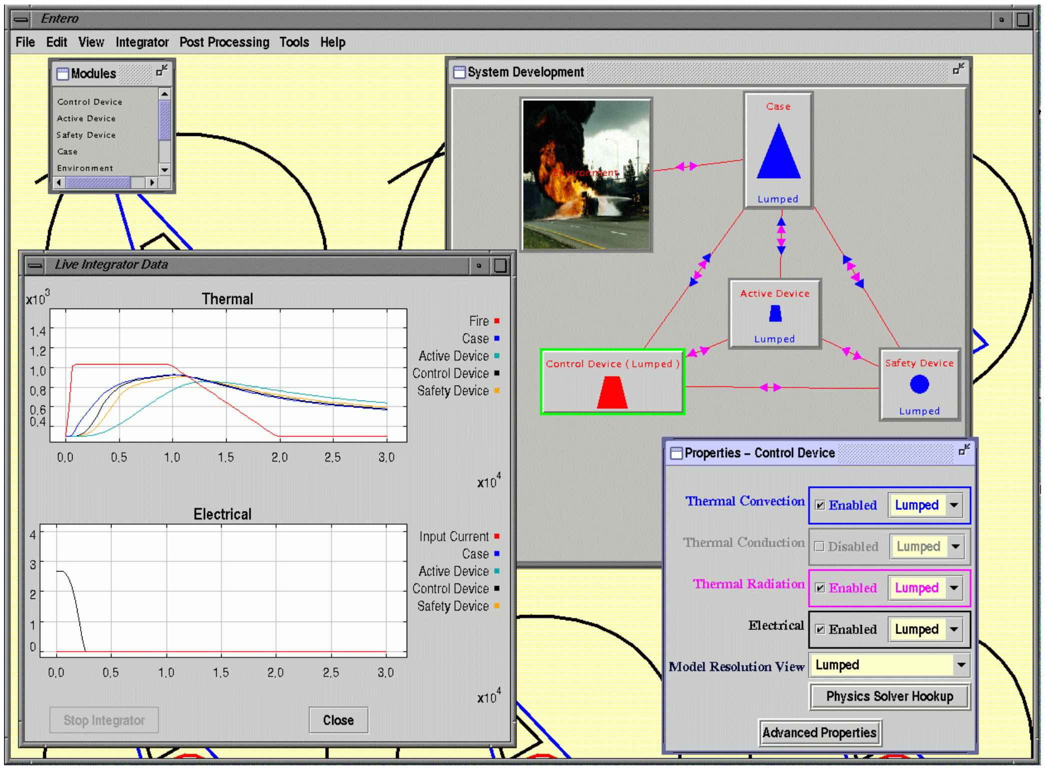


Figure 28. The Class Hierarchy for an *Entero* Module in the Prototype. A rectangular box indicates a built-in Java class while boxes with rounded corners represent classes written for the *Entero* environment.



Thermal Convection	✓ Enabled	Lumped
Thermal Conduction	Disabled	Lumped
Thermal Radiation	✓ Enabled	Lumped
Electrical	✓ Enabled	Lumped
Model Resolution View	Lumped	

Figure 29. The *Entero* Prototype Graphical User Interface and Post-Processor.

program [43]. Thus the geometric relationships among the modules are implicitly specified via the view factors, but can be changed by a knowledgeable user.

Electrical circuits can be embedded in each module, but not connected between modules. Electrical activity is calculated using the Spice 3f5 circuit simulator [37] and circuits are specified through standard Spice "netlist" files. The coupling between the zero-dimensional thermal models and the circuit models is loose and one-way. That is, the temperature of a module is computed using the zero-dimensional thermal models, and then this temperature is imposed on any circuit embedded in the module. Heat generated by the circuit is neglected.

A view of the graphical user interface and a post-processing plot from a coupled thermal-electrical simulation of a generic system is shown in Figure 29.

The preprocessor was written in the Java programming language [44, 45] to reduce the complexity of developing and maintaining software on multiple platforms. In addition, the `JavaBeans` [46, 47] component architecture closely matched desired properties for the *Entero* modules in the prototype. An example of the *Entero* prototype graphical user interface is shown in Figure 29.

Limitations of the Prototype Architecture

While the use of Java and `JavaBeans` enabled us to add new features to the prototype *Entero* environment quickly, we discovered that the initial architecture incorporated insufficient abstractions of the object hierarchies and insufficient encapsulation of various implementation details.

Many attributes and methods were duplicated in each module instead of being abstracted into the common module base class. To add a new common property or method, it was necessary for code in each existing module to be modified. To add a new module, code for an existing module was typically copied and then modified, increasing possibilities for errors and duplicating both effort and code.

External applications were tightly integrated into the physics manager, without a specified interface. To add a new application it was necessary to make significant, often drastic, changes to the physics manager.

The `JavaBeans` architecture offered some initial advantages in developing a prototype system, such as rapidly developing software for visually assembling a system of modules. However, because the visualization properties of a module are tightly coupled to the physics properties, the `JavaBeans` architecture limited the *Entero* software to user interfaces based on `JavaBeans`. User interfaces based on a variety of architectures are required to support different modeling domains (*e.g.*, penetration mechanics modeling and safety engineering). Thus the flexibility and extensibility of the software were diminished.

Another limitation with Java arises from the serialization mechanism, when it is used as a means of storing persistent objects. *Entero* modules were stored in files using Java serialization. The Java serialization mechanism assumes that the classes required to instantiate a serialized object change in only severely circumscribed ways. Any such objects that were saved prior to even minor class changes in the code were incompatible with the new code. Thus it was necessary to continually re-enter system configurations manually after any changes to class attributes, such as changes to default values for display properties.

Extending and maintaining the software became increasingly complex, error-prone, and costly. To ameliorate these difficulties the code architecture was redesigned to be more flexible and extensible. The improved architecture is described in the next section.

An Improved *Entero* Software Architecture

The *Entero* software is intended to provide an environment that continually evolves as new applications are integrated into it and new functionality is added to it. Therefore, the *Entero* code architecture must be flexible, dynamic, and scalable. To achieve these qualities in the improved architecture, we used an object-oriented design, and a library environment rather than a framework environment. These two environments are compared in Table 2. The primary advantage of the framework approach lies in its faster communication between the environment and the application. The primary disadvantage is that the application must be built within the framework, and hence it may not be able to run independently from the framework. In contrast, the primary disadvantage to the library approach is its slower communication, but its primary advantage is that it has much greater flexibility for integrating applications. In the library approach, stand-alone applications are integrated into the environment by simply placing an interface wrapper around the application. For the requirements of the *Entero* environment, it is much more important to have the flexibility for integrating applications than to have communication performance. Hence, we adopted the library approach for our improved architecture. If, in the future, the communication performance becomes a higher priority, incorporating more tightly coupled communication into the library environment will be much easier than incorporating a more modular structure into a framework environment.

Modular Functionality

One of the main priorities of the improved architecture is to make the functionality of the environment as modular as possible. In addition, the architecture must have the ability to dynamically add and remove these modules of functionality as needed by the user.

We call the entities representing these modules of functionality a `Module`. Note that for the prototype architecture the term *module* was used to describe the component, environment, and port modules. In the improved architecture, the term `Module` is used to describe a particular functionality. Therefore what was termed *component module* will now be a composite of `Module`s.

Implementing an Object-Oriented Design

To properly define objects and their relationships it is not only important that we use object-oriented design, but proven object-oriented design principles and proven software design patterns. In particular, the object-oriented design principles the improved architecture adheres to are [48]

- **Program to an interface not an implementation.**
  Manipulating objects through an interface shields the type and class of the object. This hides underlying code and clients can manipulate any type of object as long as it implements the appropriate interface, thus allowing increased code reuse.

Table 2. Comparison of Framework and Library Environments

	Framework Environment	**Library Environment**
Integrating Applications	Application must be built within the environment and conform to its standards	Interface wrappers that conform to the environment are placed around the application
Communication	Tightly coupled; hence higher performance	Loosely coupled; hence lower performance
Functional Modularity	Typically none; must utilize the entire framework to use any of its functionality	The functionality of the environment is modularized so that it may add what is needed and remove what is not needed
Stand-Alone Applications	Once the application is built within the environment it typically cannot be run stand-alone	This environment is intended to integrate stand-alone applications

- **Favor object composition over inheritance.**
  Objects add functionality by being composed of other objects rather than through inheritance. This allows functionality to be added dynamically at run time, rather than statically (at compile time).

- **Objects should delegate responsibilities.**
  Objects delegate operations, through interfaces or mediator objects, to objects that encapsulate the behavior to handle the operation. This allows objects to encapsulate a specific type of functionality and allows other objects to make use of it.

It follows immediately from these principles that most behavior and functionality should be defined via interfaces, so that manipulating objects can be done with only the view of that object's interface. In particular, `Module` will be defined as an interface.

Modules

One of the design goals for the *Entero* environment is to incorporate different types of physics applications, so there is a `Module` that encapsulates physics properties. There are various types of these physics modules to represent different categories of physics, *e.g.*, thermal radiation and electrical activity, and each of these types contains attributes corresponding to the category of physics it is encapsulating, *e.g.*, a thermal physics module will contain a temperature attribute. These physics modules were abstracted as `PrimitiveModule`s, where different types of modules are set by the `PrimitiveModule`'s member object `ModuleType`. Categories of module types are represented by `ModuleType`'s subclasses, *e.g.*, `PhysicsPMType` represents physics modules (where *PM* is shorthand for *Primitive Module*) and module types are enumerations within each subclass (*e.g.*, `THERMAL_PHYSICS_MODULE`). Different types of modules may be added simply and without changing the general infrastructure by creating a new `ModuleType` subclass or an enumeration within an existing subclass.

Each `PrimitiveModule` also contains methods that manipulate attributes, such as get and set methods. Attributes are generic entities that represent `PrimitiveModule` properties. Each attribute has a value and units. Figure 30 is a Unified Modeling Language (UML) diagram of the relationship among the interface `Module`, the `PrimitiveModule` and its `ModuleType`.

In the prototype architecture a physical component was modeled using inheritance and contained an extensive list of properties with their corresponding get/set methods. This produced a monolithic model that was difficult to modify and manage. In the improved architecture we apply the second principle of good object-oriented design and favor object composition over inheritance. A physical component is now represented as a `CompositeModule`, which is a composite of `Module` objects (including `PrimitiveModule` objects) that may be dynamically added to or removed from the model at run time. Since `CompositeModule` is a type of `Module` as well, it implements the `Module` interface, it may contain other `CompositeModule` objects (Figure 31). This structure allows us to model hierarchical physical components, *e.g.*, an electrical circuit embedded in a component.

The primary function of the `CompositeModule` is to act as a container: all its functionality is implemented in the various `Module` objects it contains. Applying the third principle of good object-oriented design, that objects should delegate responsibilities, the addition, removal, and management of the `Module` objects within the `CompositeModule` are delegated to manager and handler objects.

A `CompositeModule` may contain many `Module` objects of differing types (`ModuleTypes`). The addition or removal of these different types adheres to certain type-dependent rules. For example, there should only be a single module of a given of `PhysicsPMType` in a `CompositeModule` object, *e.g.*, a `THERMAL_PHYSICS_MODULE`. This is to prevent multiple representations of the same attribute (*e.g.*, temperature) in a `CompositeModule` object. The interface `ModuleHandler` enforces such

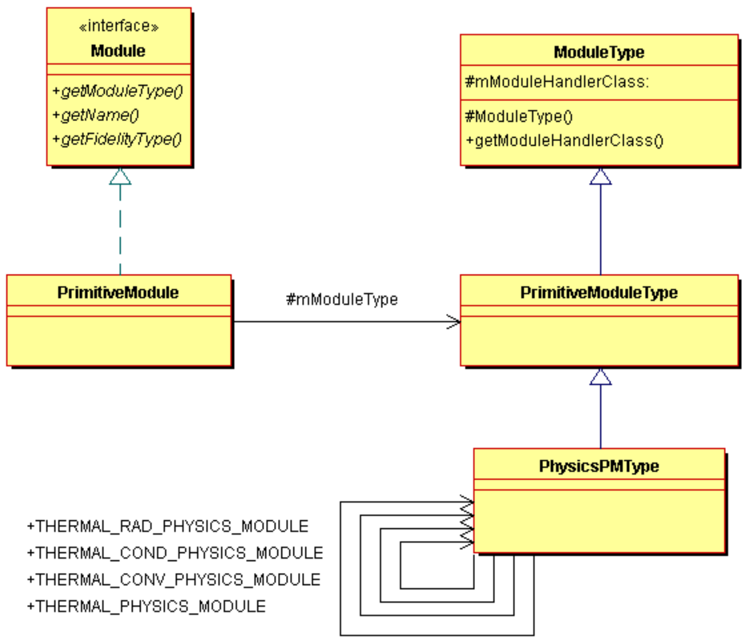


Figure 30. UML Diagram for Modules. UML diagram of the relationships among the `Module` Interface, the `PrimitiveModule`, and the `ModuleType`. All types of `PhysicsPMType` are represented as enumerations.

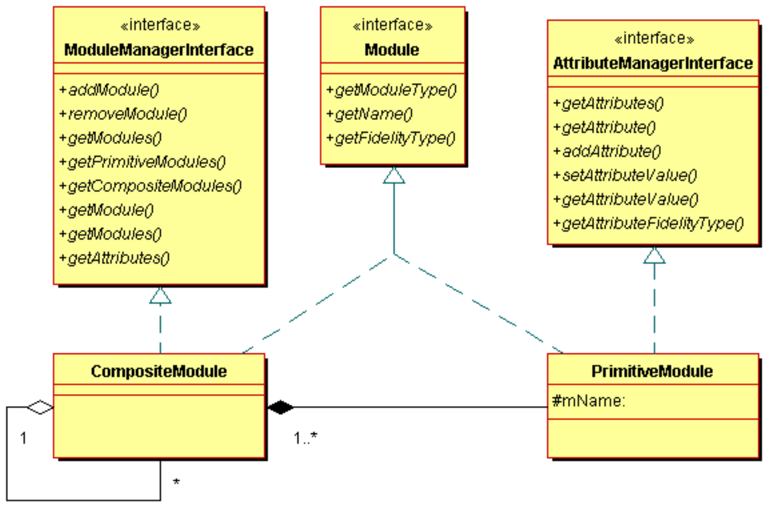


Figure 31. `CompositeModules`**. A** `CompositeModule` **is an aggregation of** `Modules`**. This includes** `PrimitiveModule` **objects and** `CompositeModule` **objects, since the** `CompositeModule` **implements the** `Module` **interface (*i.e.*, is a type of** `Module`**).**

rules when adding or removing a module from a composite. Each `ModuleType` has a `ModuleHandler` type associated with it (this is enforced by the interface). As seen in Figure 32, the `PhysicsPMType` class and all its types are associated with the `PhysicsModuleHandler` class. The `PhysicsModuleHandler` class implements the method `isModuleAdded()` from `ModuleHandler`. This method contains and enforces the rules specific to the addition of `PhysicsPMType` types. There are many other `ModuleTypes` such as `GeometryPMType`, `ApplicationPMType`, and `PortPMType`. Each of these `ModuleTypes` has its corresponding `ModuleHandler` type as well; thus a `CompositeModule` has the ability to manage multiple handler objects.

Depending on what `Module` objects of a given `ModuleType` are added to the `CompositeModule`, various `ModuleHandler` types must be instantiated and managed. This is done with the class `ModuleHandlerManager`, which is a singleton member of every `CompositeModule` class. The primary function of the `ModuleHandlerManager` is to manage data through the handler classes. For example, if a `THERMAL_PHYSICS_MOD-ULE` object is added to a `CompositeModule` object, the addition is initially delegated to the manager object from the composite. The manager object then gets the type of handler it must instantiate from the added module, in this case `PhysicsModuleHan-dler`, and then the manager delegates the addition of the module to that handler. The addition of a module is delegated by the delegator implementing the `ModuleManagerInterface`, through which the delegatee object calls the `addModule()` method. The `ModuleHandlerManager`'s object relationships are illustrated in Fig-

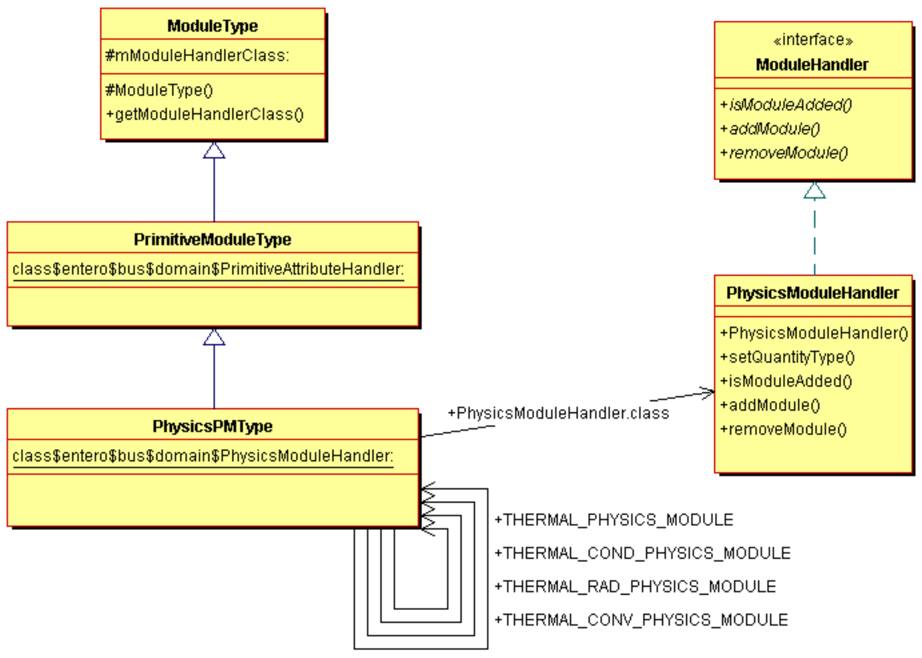


Figure 32. `PhysicsPMType` **and Its Handler Class.** `PhysicsPMType` **is associated with the** `PhysicsModuleHandler` **class. Thus all module types will have this handler class.**

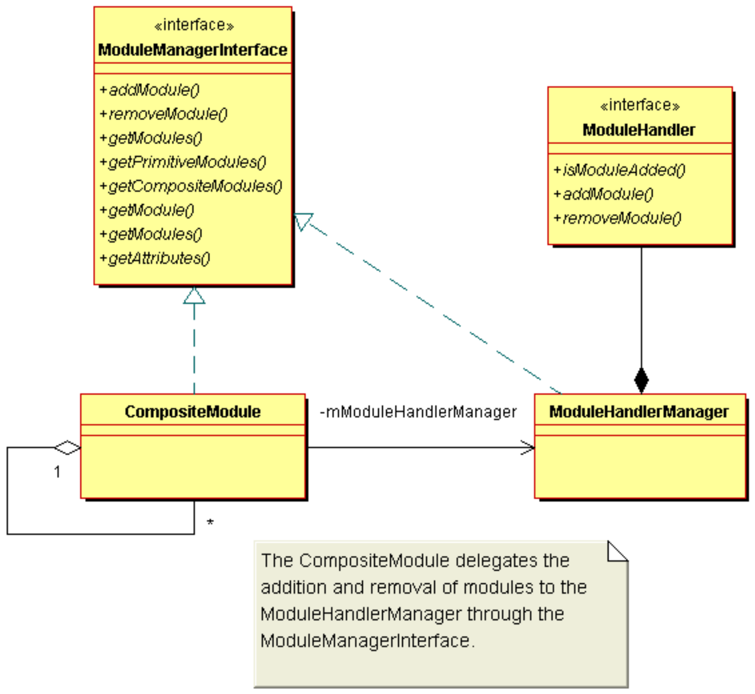


Figure 33. A `CompositeModule` and Its Handler Manager. A `CompositeModule` delegates the addition and removal of modules to the `ModuleHandlerManager` through the `ModuleManagerInterface`.

ure 33.

Application Integration

An important feature of the *Entero* architecture is the ability to easily integrate stand-alone applications into it. This is done with an `ApplicationModule`, which extends `PrimitiveModule` and provides a wrapper for the application in the *Entero* environment. This module has attribute information specific to the application, *e.g.*, a version number, an input file parser, or a list of different solvers that may be used in the application. Applications will also require data that is encapsulated in specific `PhysicsPMTypes` and `GeometryPMTypes`, thus the `ApplicationModule` stores this information as well. Since the application knows what external modules it requires in order to be used by a model, it must have the ability to add these modules to the `CompositeModule` to which it is being added. This is accomplished by implementing the `ModuleManagerInterface` in the `ApplicationModule`; it can now delegate the module additions to the appropriate `ModuleHandlerManager` object (Figure 34). By encapsulating particular functionalities within interfaces it becomes a simple task to

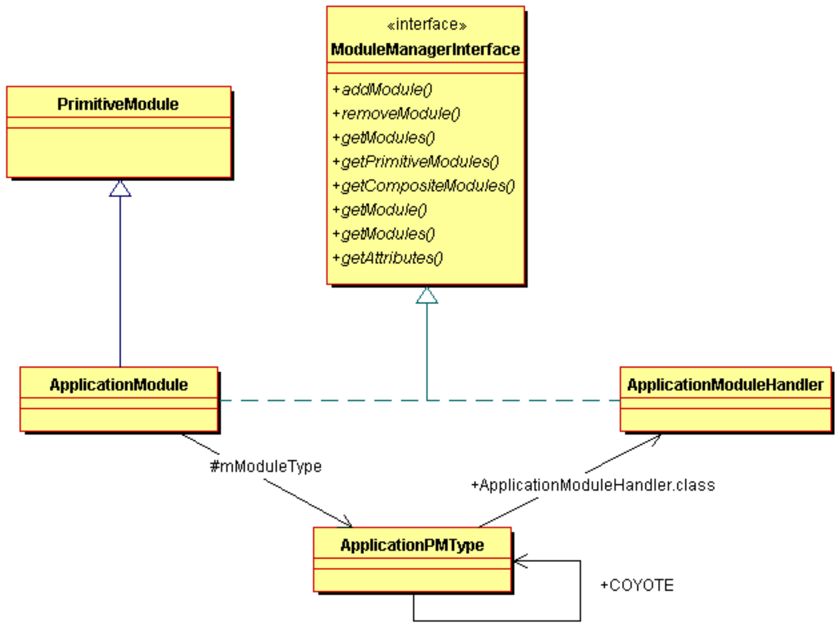


Figure 34. The `ApplicationModule`. The `ApplicationModule` becomes a builder simply by extending the `ModuleBuilderInterface`. Coyote [33] is a finite-element code for nonlinear heat transfer problems used at Sandia.

make a module a builder, and so we are able to encapsulate all information about the application within an `ApplicationModule` object.

Ports are the entities that connect system components and transfer data of differing fidelities or physics between them. Port functionality is encapsulated in a `PortModule` which extends `PrimitiveModule` and holds the identities of the source and target it links. Since data is accessed through the `PrimitiveModule` interface, connecting two components is a matter of coupling the source attribute and target attribute and then performing the appropriate data conversion and transfer between them. The data translation is kept separate from the `PortModule` so that the interpolation of data will be independent of the architecture (Figure 35).

A specific example of coupling two components with a port is coupling a component modeled with a three-dimensional finite-element code to a system of zero-dimensional components. To accomplish this, the three-dimensional model must be linked to a zero-dimensional view of the model, and the zero-dimensional view is coupled to the zero-dimensional system model. Specifically, the temperature field of a three-dimensional component must be linked to a scalar temperature for a zero-dimensional component. To perform this task a `PortModule` is created where the source attribute is the temperature field and the target attribute is the scalar tem-

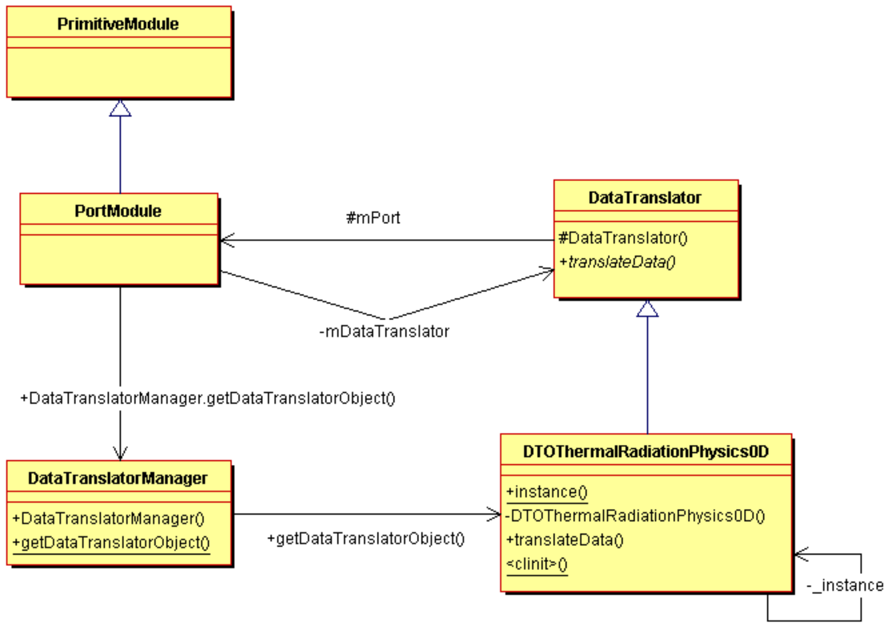


Figure 35. UML Diagram of `PortModule`. The `DataTranslator` objects contain the translation algorithms, therefore the algorithms are not a part of the port architecture.

perature. Through the `Module` interface, the `PortModule` gets the data types of the two temperatures, and determines that a conversion from a temperature field to a scalar temperature is needed. A `DataTranslator` object from a pre-compiled set of data conversion objects (which hold the type-specific conversion algorithms) is called by the `DataTranslatorManager` to perform the translation, in this case a conversion from a field to a scalar. Again, through the `PrimitiveModule` interface the port then sets the appropriate scalar temperature in the target zero-dimensional model. Since the data translation is separate from the `PortModule` and is determined at run time, the types of conversions a port may do can be added dynamically simply by adding new `DataTranslator` objects (Figure 36). Coding to interfaces and keeping the translation algorithms independent of the software architecture allows port objects to become much more flexible and dynamic in the improved code architecture.

Data Persistence

One of the primary limitations in the prototype architecture was that the metadata was embedded in the model itself. The improved architecture remedies this by placing the properties in the modules, allowing them to be dynamically added to and removed from the model. In the improved architecture, `ModuleType`s are stored as meta-data and are loaded dynamically. All of these data structures are stored using the eXtensible Markup Language (XML) and transformed, by an internal XML serialization mechanism similar to JSX [49], into Java objects to be used at run time. The serialization mechanism also converts the Java objects back into XML entities that can be stored as persistent objects. There are many advantages to saving meta-data in the XML format, including [50]

- **XML allows flexibility of data formatting**, with the ability to nest tags and define the contents of the data in an object-oriented format.
- **XML separates content and presentation.** The data is not tied to a particular view, thus different graphical user interfaces may utilize the same data.
- **XML is not tied to a particular client.** Though we have chosen Java, nothing prevents us from reading the same meta-data into C++ objects with the appropriate interfaces.
- **XML is a purely textual representation of data** which has many advantages. In particular serialization and versioning become much more viable since text files are easily parsed.
- **Java + XML = Portable Code + Portable Data**

Ultimately what XML gives the *Entero* environment is the ability to access data from a common data repository in which the data defines its structure independent of its implementation.

In the current software, the user interface is generated dynamically using XML and the *Entero* data model [51]. Thus the user interface is automatically updated if the data model changes.

An Improved *Entero* Environment for Coupled Thermal-Electrical Modeling

We implemented an improved *Entero* environment for modeling thermal radiation transport in systems containing electrical circuits and that are exposed to fires using the improved architecture.

Starting from an initial system (provided with the software or previously saved by the user), the user can add or delete components (subject to composition rules enforced by the software), and add or delete ports between components. The user is guided through this process by a series of software wizards. Components of the system are represented by zero-dimensional thermal models; the user can replace one

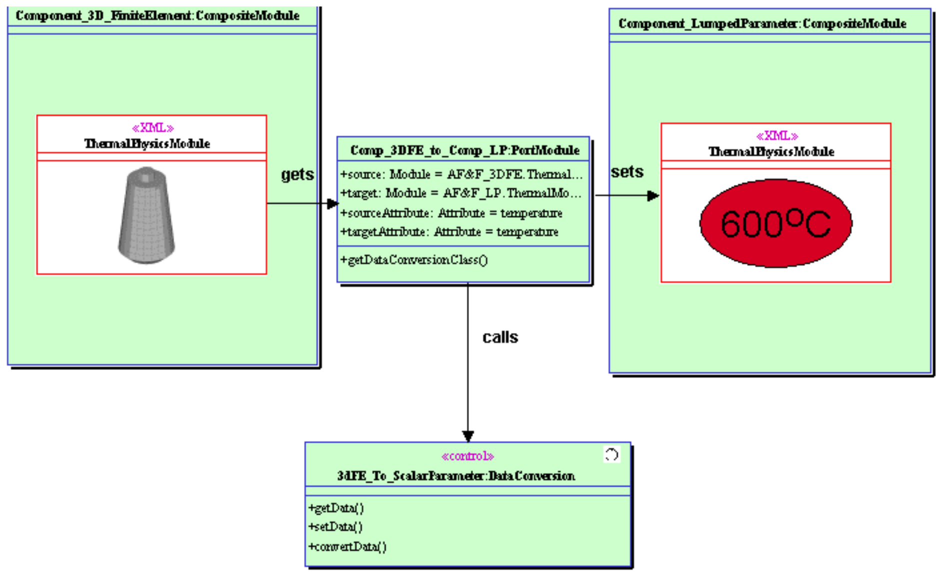
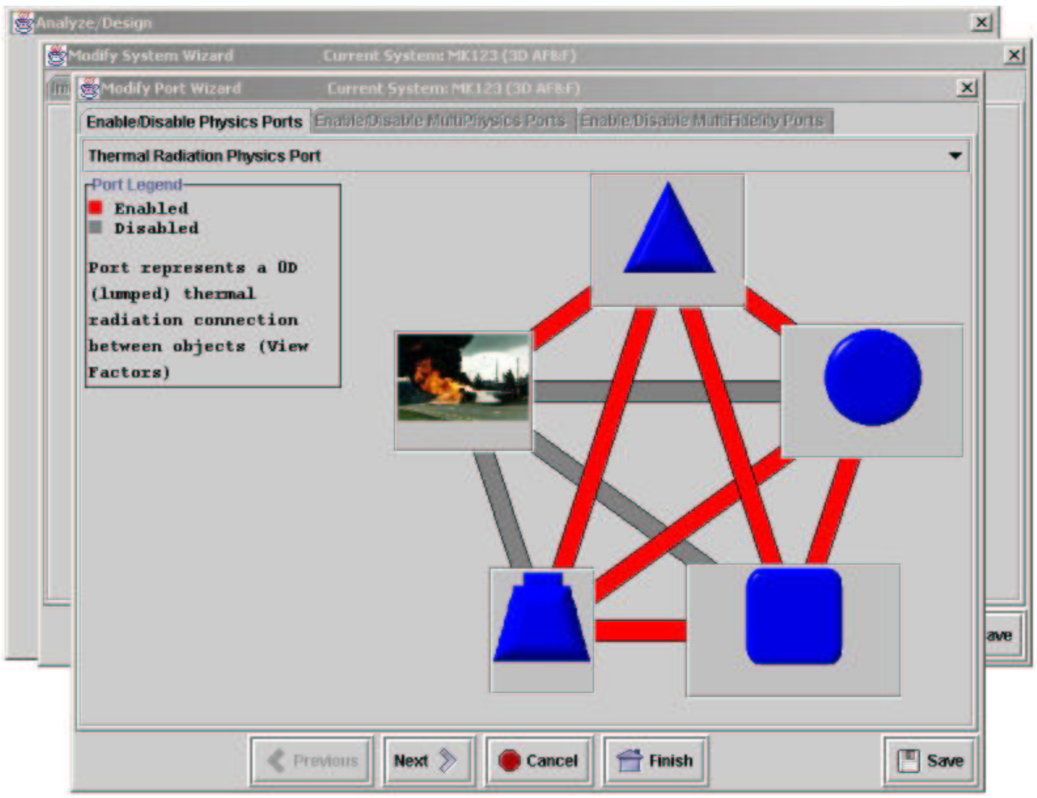

Component_3D_FiniteElement:CompositeModule
«XML»
ThermalPhysicsModule
gets
Comp_3DFE_to_Comp_LP:PortModule
+source: Module = AF&F_3DFE.Thermal...
+target: Module = AF&F_LP.ThermalMo...
+sourceAttribute: Attribute = temperature
+targetAttribute: Attribute = temperature
+getDataConversionClass()
sets
Component_LumpedParameter:CompositeModule
«XML»
ThermalPhysicsModule
600°C
calls
«control»
3dFE_To_ScalarParameter:DataConversion
+getData()
+setData()
+convertData()

Figure 36. **A `PortModule` Links the Temperatures of a Three-Dimensional Finite-Element Model and a Zero-Dimensional Model. Based on the source and target attribute types, the `PortModule` calls the `DataTranslator` object (through the `DataTranslatorManager`) which contains the algorithm for transforming a temperature field to a scalar temperature.**


Analyze/Design
Modify System Wizard
Current System: MK123 (3D AF&F)
Modify Port Wizard
Current System: MK123 (3D AF&F)
Enable/Disable Physics Ports
Enable/Disable MultiPhysics Ports
Enable/Disable MultiFidelity Ports
Thermal Radiation Physics Port
Port Legend
🟥 Enabled
▨ Disabled
Previous
Next
Cancel
Finish
Save

Port represents a 0D (lumped) thermal radiation connection between objects (View Factors)

Figure 37. The Improved *Entero* Prototype Graphical User Interface for Coupled Thermal-Electrical Modeling.

of the components with a three-dimensional, finite-element model, which is connected to the zero-dimensional system model by a mixed-fidelity port.

The dimensions and relative locations of the zero-dimensional models (which are represented as conical frustums) can be adjusted by the user with a geometry editor. View factors are automatically calculated for the zero-dimensional models. The user can modify the mass and thermal properties of individual components (*e.g.*, density and specific heat capacity).

The user can embed electrical circuits in each of the components, and for the three-dimensional model can specify the circuit location; the circuit is connected to a component using a multiphysics port.

Following system set up, the user can analyze the system and monitor the temperatures in the components and the electrical activity in the circuit as the simulation progresses. An example of the user interface is shown in Figure 37.

6 Summary

Modeling and simulation of complex systems at various levels of fidelity is increasingly important at Sandia National Laboratories in fulfilling its national security mission. In this report we have described the development of a multifidelity algorithm for thermal radiation problems, and the design and initial development of the *Entero* environment, a module-oriented, mixed-fidelity, multiphysics, system simulation environment for engineers to enable rapid system performance analysis and design optimization.

Design goals for the *Entero* environment include representing a complex system as an interacting collection of components, user-selectable model fidelity for each component, and integrated support for optimization and uncertainty quantification.

An important feature of the *Entero* environment is the capability for mixed-fidelity system modeling, in which models of different spatial dimensionalities are coupled together. Specifically, we briefly described a method for coupling zero-dimensional (lumped-parameter) models of system components to a three-dimensional model of a component for thermal radiation, and presented some comparisons of temperatures predicted by a mixed-fidelity system model to temperatures predicted by a full three-dimensional system model.

A mixed-fidelity system model can potentially execute much faster than a full three-dimensional finite-element model for thermal radiation problems and provides internal temperatures for the higher fidelity module. However, there is some loss in accuracy with the mixed-fidelity system model. Such results indicate that optimization studies with mixed-fidelity models is feasible when it may not be feasible with three-dimensional system models, if the concomitant loss in accuracy is acceptable.

We also demonstrated a mixed-fidelity, coupled thermal-electrical simulation of a temperature-dependent circuit embedded in a system for a “simulated fire” environment (Figure 26).

An extension of the algorithm to conductive heat transfer was designed but not tested. The concept is to connect modules in a system with a construct called a ”thermal wire” that is one-dimensional: conduction occurs only along its length and its cross-sectional area as a function of length is specified.

In a prototype of the *Entero* environment, a model of a complex engineering system was assembled by selecting modules from a library and placing icons that represent the components in a workspace of a visual editor. Module properties could be changed through the editor. The components were linked via ports that represented the physical coupling between them and controlled the transfer and transformation of information between the components. Physical analysis of the system was controlled by a physics manager that used custom or standard analysis codes.

Experience with the prototype revealed some design limitations that have been remedied in an improved software architecture. Features of the improved architecture include

- System components are built modularly and dynamically by object composition.
- Data is independent of the system components and is stored in the XML format. Thus properties can be added easily and dynamically, data serialization is not dependent on the implementation, versioning becomes viable, and multiple user interface frameworks can be easily used.
- Module interaction and functionality are implemented through interfaces. Manipulating objects through interfaces "hides" underlying changing code leading to an overall improvement in code reuse. Application modules add the appropriate physics and geometry needed to use an application through interfaces. Ports also handle data through interfaces, allowing interpolation algorithms to be independent of software architecture.
- Mixed-fidelity modeling is supported through ports.

We developed an improved version of the *Entero* environment for coupled thermal-electrical analysis in the new architecture. The improved environment guides the user through the process of setting up a system model using a series of software wizards, and the user can replace a zero-dimensional model with a three-dimensional finite-element model (Figure 37).

Our current work includes integrating design optimization capabilities into the architecture, and coupling radiation transport codes to the Xyce parallel electrical circuit simulator in support of the FY 2003 Hostile Environments Milestone for the Accelerated Strategic Computing Initiative.

References

[1] W. E. Vesely, F. F Goldberg, N. H Roberts, and D. F. Haasl. Fault Tree Handbook. Technical Report NUREG-0492, United States Nuclear Regulatory Commission, January 1981.

[2] ASCI Update. Sandia National Laboratories, Albuquerque, New Mexico, October 2000.
www.sandia.gov/ASCI/asciupdate/0010update.pdf.

[3] Strategic Objectives 1999. Technical Report SAND99-0412, Sandia National Laboratories, Albuquerque, NM, February 1999.
www-irn.sandia.gov/organization/div1/99strategic/sp99.pdf/.

[4] Paul K. Davis. Exploratory analysis enabled by multiresolution, multiperspective modeling. In J. A. Joines, R. R. Burton, K. Kang, and P. A. Fishwick, editors, *Proceedings of the 2000 Winter Simulation Conference*, pages 293–302, Orlando, FL, 10–13 December 2000. IEEE.

[5] G. Follen and M. auBouchon. Numerical zooming between the NPSS Version 1 and a 1-dimensional meanline design analysis code. In *Proceedings of the ISABE, the 14th International Symposium on Air Breathing Engines*, Florence, Italy, 5–10 September 1999. AIAA-99-7196.

[6] Paul K. Davis and James H. Bigelow. Experiments in multiresolution modeling (MRM). Technical Report MR-1004-DARPA, RAND, Santa Monica, CA, 1998.

[7] Paul K. Davis. An introduction to variable-resolution modeling and cross-resolution model connection. In Paul K. Davis and Richard Hillestad, editors, *Proceedings of the Conference on Variable-Resolution Modeling*, Washington, D.C., 5–6 May 1992. RAND Report CF-103-DARPA.

[8] Paul K. Davis. An introduction to variable-resolution modeling and cross-resolution model connection. Technical Report R-4252-DARPA, RAND, Santa Monica, CA, 1993.

[9] Paul K. Davis and Reiner K. Huber. Variable-resolution combat modeling: Motivations, issues, and principles. Technical Report N-3400-DARPA, RAND, Santa Monica, CA, 1992.
Added 13 July 2000.

[10] Dennis R. Powell. Control of entity interactions in hierarchical variable resolution simulation. Technical Report LA-UR-97-1287, Los Alamos National Laboratory, Los Alamos, NM, 1997.

[11] Keith W. Brendley and Jed Marti. A distributed network approach to variable resolution modeling. In Paul K. Davis and Richard Hillestad, editors, *Proceedings of the Conference on Variable-Resolution Modeling*, pages 248–255, Washington, D.C., 5–6 May 1992. RAND Report CF-103-DARPA.

[12] Paul F. Reynolds, Jr., Sudhir Srinivasan, and Anand Natrajan. Consistency maintenance in multi-resolution simulations. *ACM Transactions on Modeling and Simulations*, 7(3):368–392, July 1997.

[13] Paul K. Davis and Richard Hillestad. Families of models that cross levels of resolution: Issues for design, calibration, and management. In G. W. Evans, M. Mollaghasemi, E. C. Russell, and W. E. Biles, editors, *Proceedings of the 1993 Winter Simulation Conference*, pages 1003–1012, Los Angeles, CA, December 1993. International Society for Computer Simulation.

[14] Paul K. Davis and James Bigelow. Introduction to multi-resolution modeling (MRM) with an example involving precision fires. In *Proceedings of the Enabling Technology for Simulation Science II Conference*, pages 14–27, Orlando, FL, April 1998. SPIE.

[15] Radharamanan Radhakrishnan and Philip A. Wilsey. Ruminations on the implications of multi-resolution modeling on DIS/HLA. In A. Boukerche and R. Fujimoto, editors, *Proceedings of the 3rd IEEE International Workshop on Distributed Interactive Simulation and Real-Time Applications*, pages 101–108, College Park, MD, 23–24 October 1999. The Institute of Electrical and Electronics Engineers (IEEE), IEEE Computer Society.

[16] A. Natrajan and A. Nguyen-Tuong. To disaggregate or not to disaggregate, that is *Not* the question. In *Proceedings of the Electronic Conference on Scalability in Training Simulation (ELECSIM '95)*, CS-95-18. Society for Computer Simulation, 10 April–18 June 1995.

[17] Anand Natrajan, Paul F. Reynolds, and Sudhir Srinivasan. MRE: An approach to multi-resolution modeling. In *Proceedings of PADS'97*, Lockenhaus, Austria, 10–13 July 1997.

[18] Richard J. Hillestad and Mario L. Juncosa. Cutting down some trees to see the forest: On aggregation and disaggregation in combat models. In Paul K. Davis and Richard Hillestad, editors, *Proceedings of the Conference on Variable-Resolution Modeling*, pages 256–292, Washington, D.C., 5–6 May 1992. RAND Report CF-103-DARPA.

[19] Bruce W. Fowler. Resolution changes and renormalization for partial differential equation combat models. In Paul K. Davis and Richard Hillestad, editors, *Proceedings of the Conference on Variable-Resolution Modeling*, pages 332–340, Washington, D.C., 5–6 May 1992. RAND Report CF-103-DARPA.

[20] Alex F. Sisti and Steven D. Farr. Model abstraction techniques: An intuitive overview. In *Proceedings of the National Aerospace Electronics Conference*, Dayton, OH, July 1998.

[21] Alex F. Sisti and Steven D. Farr. Modeling and simulation enabling technologies for military applications. In *Proceedings of the Winter Computer Simulation Conference*, Coronado, CA, December 1996.

[22] D. Caughlin and A. F. Sisti. A summary of model abstraction techniques. In *Proceedings of the Enabling Technology for Simulation Science I Conference*, pages 2–13, Orlando, FL, April 1997.

[23] Kangsun Lee and Paul A. Fishwick. Dynamic model abstraction. In *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA, 8–11 December 1996. Society for Computer Simulation.

[24] Frederick K. Frantz. Analyzing models for abstraction. In *Proceedings of the Enabling Technology for Simulation Science I Conference*, pages 14–21, Orlando, FL, April 1997. SPIE.

[25] Frederick K. Frantz. A taxonomy of model abstraction techniques. Technical Report RL-TR-96-87, Air Force Rome Laboratory, Rome, NY, August 1996.

[26] A. T. Norris. Automated simplification of full chemical mechanisms: Implementation in the national combustion code. In *Proceedings of the 34th AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit*, Cleveland, OH, 13–15 July 1998. The American Institute of Aeronautics and Astronautics, AIAA 98-3987.

[27] Levent Acar, James S. Albus, and Alexander M. Meystel. A mathematical representation of multi-resolutional world modeling. In *Proceedings of the 1995 IEEE International Symposium on Intelligent Control*, Monterey, CA, 27–29 August 1995. The Institute of Electrical and Electronics Engineers (IEEE).

[28] John A. Reed and Abdollah A. Afjeh. Connecting components of varying fidelity in a turbofan engine simulation. *Modeling and Simulation—Control, Signal Processing, Robotics, Power*, 23(4):2291–2298, 1992.

[29] John A. Reed and Abdollah A. Afjeh. Integrating computer generated fan performance data with a turbofan engine simulator. In *Proceedings of the Modelling, Simulation, and Identification Conference*, pages 83–90, Vancouver, BC, August 1992.

[30] John A. Reed and Abdollah A. Afjeh. A comparative study of high and low fidelity fan models for turbofan engine system simulation. In *Proceedings of the IASTED International Conference on Applied Modelling and Simulation*, Banff, Canada, July 1997.

[31] Patrick J. Roach. *Computational Fluid Dynamics.* Hermosa Publishing Company, Albuquerque, NM, 1985.

[32] David R. Gardner, Joseph P. Castro, Gary L. Hennigan, and Benjamin H. Cole II. A prototype of the *Entero* system engineering code package. Technical Report SAND2000-1368, Sandia National Laboratories, Albuquerque, NM, June 2000.

[33] David K. Gartling and Roy E. Hogan. Coyote–a finite element computer program for nonlinear heat conduction problems. part i–theoretical background. version 3.03. Technical Report SAND94-1173, Sandia National Laboratories, Albuquerque, NM, December 1999.

[34] David K. Gartling, Roy E. Hogan, and Micheal W. Glass. Coyote–a finite element computer program for nonlinear heat conduction problems. part ii–user's manual. version 3.03. Technical Report SAND94-1173, Sandia National Laboratories, Albuquerque, NM, December 1999.

[35] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine—A Users' Guide and Tutorial for Networked Parallel Computing*. Scientific and Engineering Computation. MIT Press, Cambridge, MA, 1994.

[36] David R. Gardner and Gary L. Hennigan. A multifidelity modeling algorithm for system-level engineering analysis. (in preparation) SAND 2002-XXXX, Sandia National Laboratories, Albuquerque, NM, 2002.

[37] The Spice Circuit Simulator. [Online]. `http://infopad.eecs.berkeley.edu:80/~icdesign/SPICE`.

[38] S. Hutchinson, E. Keiter, R. Hoekstra, H. Watts, A. Waters, T. Russo, R. Schells, and C. Bogdan. The Xyce™ Parallel Electronic Simulator–An Overview. In *Proceedings of Parallel Computing 2001 (ParCo2001)*, Naples, Italy, 4–7 September 2001.

[39] D. R. Gardner, J. P. Castro, P. N. Demmie, M. A. Gonzales, G. L. Hennigan, M. F. Young, and S. S. Dosanjh. Developing a flexible system modeling environment for engineers. In *Proceedings of the 35th Hawaii International Conference on System Sciences*, Hilton Waikoloa Village, Big Island, HI, 7–10 January 2002.

[40] David R. Gardner, Joseph P. Castro, Paul N. Demmie, Mark A. Gonzales, Gary L. Hennigan, and Michael F. Young. The *Entero* Project: Developing a Multifidelity System Environment for Design Engineers. In *Proceedings of the Summer Computer Simulation Conference 2002*, San Diego, CA, 14–18 July 2002.

[41] Joseph P. Castro, Paul N. Demmie, David R. Gardner, Mark A. Gonzales, Gary L. Hennigan, and Michael F. Young. The *Entero* Software Architecture: Reflecting the Way Engineers Think about Systems. In *Proceedings of the Summer Computer Simulation Conference 2002*, San Diego, CA, 14–18 July 2002.

[42] Mark A. Gonzales, Joseph P. Castro, Paul N. Demmie, David R. Gardner, Gary L. Hennigan, and Michael F. Young. The *Entero* Environment: An Example of Agile Software Development. In *Proceedings of the Summer Computer Simulation Conference 2002*, San Diego, CA, 14–18 July 2002.

[43] M. W. Glass. CHAPARRAL: A library for solving large enclosure radiation heat transfer problems. Technical Report SAND95-2049, Sandia National Laboratories, Albuquerque, NM, August 1995.

[44] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, Boston, MA, 2nd edition, 1996.

[45] James Gosling and Henry McGilton. The Java language environment: A white paper. Sun Microsystems, Inc., 1997. `java.sun.com/docs/white/langenv`.

[46] Sun Educational Services. *JavaBeans Component Development*. Sun Microsystems, B edition, April 1999.

[47] Sun Microsystems. *Sun Microsystems JavaBeans API*, 1.01 edition, July 1997.

[48] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, Massachusetts, 1995.

[49] JSX. [Online]. `http://www.csse.monash.edu.au/~bren/JSX`.

[50] Brett McLaughlin. *Java and XML*. O'Reilly & Associates, Inc., Sebastopol, CA 2000.

[51] Edwin S. Wong. Developing an event-driven generator for user interfaces in the *Entero* software. Technical Report SAND 2002-2917, Sandia National Laboratories, Albuquerque, NM, September 2002.

Distribution

1	MS 0739	G. E. Rochau, 6415
5	0739	M. F. Young, 6415
1	1137	K. L. Hiebert-Dodd, 6535
5	1137	M. A. Gonzales, 6535
1	0316	W. J. Camp, 9200
1	0847	S. A. Mitchell, 9211
1	0310	M. D. Rintoul, 9212
1	1110	D. E. Womble, 9214
1	1111	B. A. Hendrickson, 9215
1	0310	R. W. Leland, 9220
1	0316	P. Yarrington, 9230
1	0819	E. A. Boucheron, 9231
1	0819	P. F. Chavez, 9232
1	0316	S. S. Dosanjh, 9233
5	0316	J. P. Castro, 9233
6	0316	D. R. Gardner, 9233
5	0316	G. L. Hennigan, 9233
1	0316	R. J. Hoekstra, 9233
1	0316	S. A. Hutchinson, 9233
1	0316	E. R. Keiter, 9233
1	0316	C. C. Ober, 9233
1	0316	R. C. Schmidt, 9233
1	0316	J. N. Shadid, 9233
1	0316	J. B. Aidun, 9235
1	0188	LDRD Office, Attn: D. L. Chavez, 1030
1	MS 9018	Central Technical Files, 8945-1
2	0899	Technical Library, 9616
1	0612	Review & Approval Desk, 9612