Developments in AECL’s Next Generation Code Suite Coupling (Backbone)

IAEA Workshop on Advanced Code Suite For Design, Safety Analysis And Operation Of Heavy Water Reactors
(2-5 October, 2012, Ottawa, Marriott Hotel)

D.J. Caswell
(AECL, Chalk River Laboratories, Fuel & Fuel Channel Safety Branch)
Background

• Emerging needs:
  – To enhance the current technology to provide more harmonious and integrated modeling solutions;
  – To capture multi-physics interactions between complex phenomena through rigorous coupling at time step level during simulation;
  – To enhance interoperability and integration among safety analysis tools.

• AECL is developing a Next Generation Code suite to fulfill these needs.
Traditional User Perspective

- Physics
- Fuel
- Thermal Hydraulics
Integrated User Perspective

- Thermalhydraulics
- Fuel
- Physics
- Fuel Channels
- Severe Accidents
- Fission Product Release and Containment
- Containment
User Actions

- Select simulation components;
- Prepare the associated input files (static data);
- Select data for live, run-time exchange;
- Specify source and destination for live data;
- Launch selected components;
- Monitor simulation status;
- Pause or terminate running simulations;
- Review and interpret simulation results;
- Other administrative features as needed.
To fully address the needs of the Safety Analyst, the Backbone must provide:

1. A common means to gain access to every code in the suite,

2. A consistent approach to code coupling across the suite, and

3. Simulation control for sequencing and synchronization
Requirement 1: Common Access

- Code suite and infrastructure must:
  - Support single or multiple concurrent users for each individual simulation program;
  - Address increased computer resource demands due to:
    - Option of multiple simultaneous users;
    - Increased complexity of individual simulations;
    - Complexity associated with coupling multiple phenomena across multiple simulation programs;
  - Provide support for dual mode simulation programs (operating either as a stand-alone code or participating in a coupled simulation.)
Requirement 2: 
Consistent Coupling Approach

• Coupling Modes:
  – Loose Coupling
  – Tight Coupling

• Development of a standardized coupling interface.

• “Plug and Play” integration with code suite.

• Design intent for code suite support:
  – Platform independence
    – Windows and Linux
  – Multiple programming language support
    – Fortran 95 / 2003
    – C
    – C++
    – Python
Requirement 3: Simulation Control

• Once a simulation has been selected and started, information flow may arise from the user or one or more of the cooperating simulation programs, for example:
  
  – User directed:
    – Progress enquiry
    – Pause request
    – Termination request
  
  – Suite directed:
    – Member failure notification to other coupled simulations
    – Synchronization tasks
    – Administrative communication as needed
Intermediate Level Design

How is the high level design implemented?

1. Common means to gain access to the suite;
2. Consistent approach to code coupling across the suite, and;
3. Simulation Control
1. Common means to gain access to the suite

- Graphical user interfaces will provide a unified view of the system.
- A client-server architecture will establish a repository for each member of the Next Generation suite as a server.
- The safety analyst is represented as a client in the system.
- To ensure effective and efficient client support, a distributed system will be employed to address the anticipated increase in system resource loads.
Logical Client Server Architecture
2. Consistent approach to code coupling across the suite

- 2 coupling modes: loose and tight coupling

- 2 main data coupling technologies examined: CORBA and MPI:
  - CORBA (Common Object Request Broker Architecture)
  - MPI (Message Passing Interface)
  - Both CORBA and MPI are available as open source

- Socket-based communication used in support for both the user and the coordination of the coupling

- A supporting technology is in preliminary design and development
  - To provide optimized data transfer synchronization intervals between tightly coupled codes
Coupling Modes

- Depending upon the needs of the problem being simulated, 2 distinct modes of coupling are to be made available:
  1. Loose Coupling, and
  2. Tight Coupling
Loose Coupling

- Typically achieved via output files following a particular simulation.
- Data may be post-processed and used as input to other simulations.
- Used in the absence of strong interdependencies.
- Historically, this type of coupling has been used.
- Limited by lack of feedback between participating simulations (where interdependencies exist).
- Potential loss of fidelity in the simulation.
Tight Coupling

Backbone provides a means to provide “tight coupling” for simulation data exchange between other members of the suite:

– Tight coupling refers to the **synchronous exchange** of simulation data between safety analysis programs;
– Used in situations where Next Generation codes have one or more related phenomena in common and a **single or mutual dependency** exists;
– Permits the use of dynamic data that reflects the effects of code-to-code feedback;
– Enhances the granularity of the solutions.
Standard Interface

- Tight coupling interface must be easily accessible from each simulation program
- “Plug and Play”
  - Very little adaptation needed to provide coupling for new programs added to the suite
- Supports users on one or both Windows and Linux systems
- Compatible with Fortran, C & C++ programming languages
CORBA Overview

- Established as a standard to enable software components written in a variety of programming languages and running on multiple computers to work together.
- Ideally suited for a client-server architecture.
- Joins applications resident on either the same host or remote hosts on a network.
CORBA Location Transparency

- Each CORBA object (such as a fuel modelling application on a server) has a unique object reference.
- Clients use the object references to communicate with the intended application.
- A Naming Service exists to simplify the task of finding the object reference regardless of where the object resides on the network.
- The client application, when initialized, need not be aware of the location of the requested server (it uses the Naming Service).
- Therefore, the servers can easily be relocated.
1. Multiple programming language support, including C, C++, Python, Lisp, Ruby, Smalltalk, Java, COBOL and PL/1.

2. Multi-Platform Support:
   - Linux / Unix;
   - Windows;
   - Sun;
   - Mac;
   - Any O/S supporting Java (which is O/S independent).

1. Lack of support for Fortran.

2. Requires “knowledge” of data types being transferred (similar to a function prototype):
   - Intrinsic data types are supported;
   - Generic data type, “ANY” can be defined by the software developer to address non-intrinsic data types or groups of data types, similar to a software structure (a container for a group of standard or non-standard data).

3. Potential firewall Issues:
   - CORBA uses low-level TCP/IP connections;
   - Some firewalls permit only HTTP connections;
   - For internal use within organizations, this should not pose a problem.
Coupling via CORBA was benchmarked against a stand-alone program that merged functionality of 2 independent programs into a single executable (fuel & fission product).

Excellent agreement was obtained between the single, stand-alone program and the 2 programs coupled via CORBA.
Coupling the 2 stand-alone programs via CORBA provided a significant gain in run time performance:

<table>
<thead>
<tr>
<th>Software</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand-Alone (single executable)</td>
<td>842</td>
</tr>
<tr>
<td>Coupled (dual-core computer)</td>
<td>465</td>
</tr>
</tbody>
</table>

Coupled solution completed in only 55% of the time needed by the stand-alone version.

Gain due solely to distributing the work.

“Parallelization” of each executable could provide further efficiencies on multi-core computers or in a distributed system.
MPI Overview

- Established as a message-passing library interface specification (not an implementation).
- Primarily designed for use with parallel programs running on clusters or multi-processor individual computers.
- Language independent message communication protocol.
- Support for both point-to-point and collective communication.
MPI Implementation

- Implemented as a library interface.
- Provides functions, subroutines or methods as appropriate for the languages using MPI.
- Direct support exists for common languages used for safety analysis simulations such as:
  - C
  - C++
  - Fortran 77
  - Fortran 95
Data Type Compatibility

- Predefined MPI data types exist for common intrinsic data types.
- Support for user-defined data types (structures) exists.
MPI Process Management

Able to:

1. Create new MPI processes at runtime, or
2. Establish communication with existing, independent processes
Process creation via MPI has 2 distinct modes of operation:

1. Parent-Child mode
2. Client-Server mode

Each mode uses the same point-to-point communication commands:

- MPI_SEND
- MPI_RECV
Parent – Child Mode

- Able to start new processes via an interface to an external process manager.

- 2 spawning options:
  - Spawn a number of identical binaries and establish communication with an ‘inter-communicator’ (like a file handle), or;
  - Spawn multiple binaries, or the same binary with multiple sets of arguments.

- Additional Parent-Child commands are available, for example to return the parent inter-communicator for the current spawned process.
Parent / Child Implementation

**MPI Application Driver**

- **MPI_COMM_SPAWN_MULTIPLE**
  - MPI_SEND
  - MPI_RECV

**Binary 1**

- **MPI_COMM_GET_PARENT**
  - MPI_SEND
  - MPI_RECV

**Binary 2**

- **MPI_COMM_GET_PARENT**
  - MPI_SEND
  - MPI_RECV
Client-Server Mode

- Establishes communication between 2 sets of MPI processes that do not share a communicator.
- Well suited for a server needing to accept connections from multiple clients.
- Provides communication channels in situations where no communication channel was previously established and there is no parent-child relationship in place.
Client-Server Mode (cont’d)

- One process behaves as a server while the other processes function as clients.
- In any client-server configuration, clients require information on the location of the server, such as an IP address and a port number.
- MPI provides functions to obtain this information.
- We were unsuccessful in implementing these functions:
  - Instead designed the server to write the IP and port number to an external file in a shared location where clients can look up the connection information.
  - We plan to revisit this at a later date.
MPI Benchmarking

- Coupling via MPI was benchmarked against the same stand-alone program used in the CORBA benchmark.
- Excellent agreement was obtained between the single, stand-alone program and the 2 programs coupled via MPI.
Benchmark Timing

- Tested both MPI parent-child and MPI client-server configurations on a dual core computer.

- Timing results:

<table>
<thead>
<tr>
<th>Software / Mode</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand-Alone Application</td>
<td>1680</td>
</tr>
<tr>
<td>MPI Parent-Child</td>
<td>964</td>
</tr>
<tr>
<td>MPI Client-Server</td>
<td>945</td>
</tr>
</tbody>
</table>

- Distributing the load over 2 cores rendered the solution in only 55% of the time.
Further savings would result from establishing parallel execution of each program on multi-core computers and running the programs in a distributed system.
Further Efficiency Gains

- Additional gains can be achieved by utilizing multi-core processing within a distributed system.

![Diagram showing Program A with subprograms A1, A2, A3 connected to Program B with subprograms B1, B2, B3, B4 via MPI and CORBA connections.](image-url)
Supporting Technology Development

- Development is underway for the creation of a “smart” data exchange interval controller.
- The objective is to determine optimal data exchange intervals when two or more Next Generation Codes are running in parallel using tight coupling.
- The goal is to ensure that the data being transferred at each interval accurately represents the data calculated by the code supplying the data (no peaks or dropouts missed).
We anticipate that the interval of code-to-code data exchange will be longer than the codes’ internal time step length for calculations.

The controller must “survey” all data exchanges and feedback effects and then select an optimal interval.

The algorithm is currently under development.
3 Simulation Control

– Client-Server architecture selected.
– Executive control utility.
– User interfaces.
Executive Control

• Means to transfer administrative information within the Next Generation Code suite (information not directly related to simulation data transfer).
• Utilizes a network-based, client-server architecture.
• Support for multiple programming languages, Fortran, C, C++, Python.
Packaged for easy insertion into existing Fortran programs without code modification (like a set of library functions).

Utilizes low level TCP/IP socket communication (written in C).

Tested on networks consisting of Windows and Linux.

Expandable to any operating system that supports TCP/IP communications.
Executive Control (cont’d)

- Implemented as a Fortran 95 module (like an object).
- Core functionality resides in lower level C.
- Mixed language implementation (Fortran files linked against the object library derived from the C source files).
- Provides direct access to 4 methods (subroutines):
  - Open;
  - Read;
  - Write, and;
  - Close.
- Module also contains and seamlessly manages the underlying technical implementation details.
Executive Features Implemented

- Launch remotely hosted programs.
- Monitoring function (program progress, such as current simulation time, percent complete, etc).
- Request graceful termination of running programs.
- Program-to-program message passing (eg. program A has failed, so notify other programs of the upset).
  - Useful for program synchronization such as notification to the suite of a time step rejection by one member.
- Client able to register their unique names with the server.
  - Enables easy identification of peers for peer-to-peer communication.
User Interfaces

• Interfaces will be developed, based upon the executive control technology, to provide input and control of the simulation features.

• How are they connected?
Each simulation program can:

- Provide and receive data from adjoining simulations;
- Provide a list of available data for sharing.

The safety analyst is able to select the data and where it is exchanged.

A schematic diagram of the planned user selection follows:
Live Data Selection

Sample User Transfer Data Selection Graphical User Interface

- Input
- Output
- Data Flow

N. G. Code 1

N. G. Code 2

N. G. Code 3
Putting it all Together

We have looked at the key aspects for the backbone

1. A common means to gain access to every code in the suite,

2. A consistent approach to code coupling across the suite, and

3. Simulation control for sequencing and synchronization.
Tested Implementations

- CORBA
- MPI
- Socket-based generalized communications
Next Steps

CORBA / MPI / Executive Hybrid Design

- Currently developing an approach to combine CORBA and MPI.
- Utilizing the best features of each underlying technology.
- Integrated with the executive control system.
CORBA / MPI Hybrid Design (cont’d)

Design benefits:

1. Alleviates the issue of having no Fortran support for CORBA
2. Facilitates dual mode program execution (coupled vs. stand-alone)
3. Provides a fully integrated technology solution by combining and linking CORBA, MPI and the executive control technologies
4. Simplifies the user interface
Proposed Architectural Implementation

Legend
- Socket Client
- Socket Server
- CORBA Client
- CORBA Server
- MPI Child
- MPI Parent

Diagram:
- Socket Communication Server
- User Interface
- N.G. Code 1 Instance 1
- N.G. Code 1 Instance 2
- N.G. Code 2 Instance 1
- N.G. Code 2 Instance 2
- MPI (Parent / Child)
- Next Gen Code 1 Server
- Hybrid Coupling Client
- Next Gen Code 2 Server
Questions and Discussion