Feature Analysis of Coupling Technologies for Earth System Models

Rocky Dunlap
Dr. Spencer Rugaber
Dr. Leo Mark
College of Computing
Georgia Tech

Workshop on Coupling Technologies for Earth System Modelling : Today and Tomorrow
CERFACS
Toulouse, France
December 15-17, 2010
Background

“The descriptors used for comprehensively specifying a model configuration are also needed for a scientifically useful description of the model output data.”

Data curation ensures that future scientists can understand and use the data products generated today. Work continues today in partnership with the Metafor project: [http://www.metaforclimate.eu/](http://www.metaforclimate.eu/)
Earth System Curator Partners

• Geophysical Fluid Dynamics Laboratory, NOAA
• Earth System Modeling Framework, NOAA/ESRL
• Earth System Grid, NCAR, NESII/CIRES/NOAA
• Department of Earth, Atmospheric and Planetary Science, MIT
• Department of Earth and Atmospheric Sciences, Georgia Tech
Background

• Who are we?
  – Computer scientists (software engineering, databases)

• Why are we interested in coupled models?
  – Large, complex software systems
  – Represent a synthesis of deep knowledge areas: high performance computing & parallel programming, component development, numerical methods, earth & atmospheric sciences
  – Scientific software is different than other kinds of software. Climate models are an “experimental environment” and a “research tool” that are “constantly under development.”
Background

• What kinds of things are we interested in?
  – Not just high performance couplers, improved interpolation schemes, and planning for new hardware (although these are important!)
  – Writing good code
  – Intentional code—that is, code that clearly shows the intention of the developer and reflects domain-level concepts
  – Abstraction, modularity, separation of concerns
  – Ways of reducing complexity for model developers
  – Not just scientific validation, but code verification

• Generative programming as a potential solution
Coupler Generation

• One of the primary goals identified by the Earth System Curator project is the ability to **generate Earth System Model (ESM) couplers** automatically based on high-level descriptions of the software modules (models) to be coupled.

• The **Cupid** prototype code generator was an early success. Simple drivers and couplers based on the Earth System Modeling Framework (ESMF) could be generated from descriptions of component interfaces.
Cupid: Domain Specific Language + Code Generator

Generated code

```c
/*
 * Definition of user_modell (existing component)
 */
define griddedcomponent user_modell {
    define interface {
        register: userml_register
    }
    define distgrid userml_distgrid {
        extents: 1:100, 1:150
        regDecomp: 4,1
    }
    define array userml_arraydata {
        typekind: BSMP_TYPE_KIND_R8
        rank: 2
        distgrid: userml_distgrid
    }
    exportstate userml_export {
        userml_arraydata
    }
}

/*
 * Definition of composition
 */
define composition arrayRegridExampleComposition {
    define connection (user_modell1, user_modell2) {
        coupler: genclpl
        regrid: userml_fielddata -> userm2_fielddata
        regrid: userml_field2 -> userm2_field2
    }
}

/*
 * Definition of an execution schedule.
 */
define schedule arrayRegridExampleSchedule {
    user_modell1, genclpl, user_modell2
}
```
Moving Forward

- While the Cupid prototype showed that automatic generation of couplers between components with simple interfaces is feasible:
  - Did not leverage a large portion of the ESMF API
  - Did not consider the features of other coupling technologies
  - Did not consider the actual coupling requirements of production ESMs

- Observation: Coupling is not just a technical process—it is essentially scientific in nature
Bretherton Diagram

CONCEPTUAL MODEL of Earth System process operating on timescales of decades to centuries

Found online at http://www.geo.unibremen.de/geomod/staff/gerrit/irvine/art/bretherton_color.gif
Straightforward architecture, but...

Atmosphere

Physic

Dynamic

...lots of complexity at implementation level
Backing up…

• What are the requirements for building an Earth System Model coupler?
  – Taking into account the complexities that inevitably emerge at the implementation level

• What kinds of features must be supported?
Feature Analysis of Coupling Technologies

- A key step in generative programming is feature analysis, which is an attempt to understand a set of related technologies by organizing their features along orthogonal dimensions
  - **Coupling technologies** represent a family of related software products
- A *feature* is an unit of user-visible functionality
- The output of feature analysis is a *feature model* that identifies common and variable properties of the technologies
- Once a feature model has been produced, elements can be selected from it to produce a *configuration*, which describes a particular family member
Feature Diagrams

• A feature model is expressed as a feature diagram—an annotated tree in which nodes represent features in the domain

• A feature is an element of user-visible functionality

• Nodes are connected with directed edges and edges have decorations that define the relationship between parent and child nodes
Notation

- The root node the diagram is called the *concept* node.
- All nodes below the concept node represent features and subfeatures.
- *Mandatory* features are denoted by a simple edge ending with a filled circle.
- *Optional* features are denoted by a simple edge ending with an open circle.
- Subsets of features may be *alternatives* denoted by connecting the edges pointing to alternatives with an arc.
- If an arc connecting edge is filled in, it indicates that any subset of the alternatives may be chosen; otherwise the alternatives are mutually exclusive.
Car Feature Diagram

Car

Transmission
- Automatic
- Manual

Engine
- 4 Cylinder
- 6 Cylinder
- Turbo

Navigation System
- Voice Activated
- Touchscreen Activated

High Fuel Economy

Constraint:
Manual Transmission requires Turbo Engine

Key
- Mandatory Feature
- Optional Feature
- Alternative Features (choose one)
- Or Features (choose at least one)
Constraints

• Feature diagrams may also contain constraints that enforce dependencies among features
  – *Mutual-exclusion* constraints are used to describe illegal combinations of features
  – *Requires* constraints indicate that the presence of one feature requires the presence of another
How do feature models compare to…

• ...existing taxonomies of coupling technologies?
  – Both introduce a vocabulary of terms that describe essential concepts related to coupling
  – Feature models break down a set of related software products into a multi-dimensional space

• ...other kinds of software diagrams?
  – Features models do not describe how features are structured or implemented, only what features are available and which sets of features make a valid configuration
Existing Taxonomies of Coupling Technologies

• Tom Bulatewicz PhD Thesis\(^1\). Four part taxonomy of coupling methodologies:
  – **Monolithic**: brute force, manual code merging
  – **Scheduled**: model execute independently; output of one model becomes input of another
  – **Communication-based**: send/receive calls embedded at coupling points
  – **Component-based**: model code modularized into components with standard interfaces

Bulatewicz Taxonomy of Coupling Methodologies

**Monolithic**

for \( i=1, nx \)
for \( j=1, ny \)
call dyn()
call update()
end for
end for

for \( k=1, z \)
call phys1(k)
call phys2(k)
end for

**Scheduled**

for \( i=1, nx \)
for \( j=1, ny \)
for \( k=1, z \)
call dyn()
call phys1(k)
call phys2(k)
call update()
end for
end for
end for

19
Bulatiewicz Taxonomy of Coupling Methodologies

**Communication-based**

```plaintext
for i=1,nx
  for j=1,ny
    call dyn()
    send(X)
    call update()
  end for
end for

for k=1,z
  call phys1(k)
  receive(X)
  call phys2(k)
end for
```

**Component-based**

- **Ocean**
  - import
  - export
- **Coupler**
  - import
  - export
- **Atmosphere**
  - import
  - export

**Coupler**

- importA
- exportB
- importB
- export
Jagers Comparison of Coupling Technologies

- Broad scope includes several classes of technologies:
  - High performance coupling (MCT, OASIS, ESMF, CCA, HLA)
  - Loose coupling / Interoperability (FRAMES, OpenMI)
  - Object-oriented frameworks (OMS)
  - GUI / Visualization (TIME)
  - Scientific workflow engine (Kepler)
  - Architectural standards (CCA, HLA)
  - Community development initiatives (CHyMP)

Jagers Comparison of Coupling Technologies

• Wide range of technologies requires more a more abstract inter-comparison

• Comparison includes ten dimensions such as:
  – defines framework
  – defines interfaces
  – provides reference implementation
  – code invasiveness
  – plug & play / graphical
  – HPC support
Feature Diagrams versus Other Kinds of Software Diagrams

- **Class diagrams** show the static structure of a program in terms of classes, their attributes and operations, and how the classes are related.
- Classes are implementation-level structures.
- **Features, on the other hand, do not have any structural semantics**
  - Features do not necessarily map directly to classes
  - Edges should not be interpreted to have structural semantics such as “part-of” or “consists-of.” We want to avoid importing structural characteristics into the feature model.
Feature Diagrams versus Other Kinds of Software Diagrams

- *Software architecture* diagrams show the main components in a software system and how they are inter-connected.

- Feature diagrams do not imply a certain architecture. Typically, an architecture is designed such that it can support all possible configurations of features.

- However, because features represent variation points, architectural choices can be represented as features.
Software Architecture Diagrams

**Architecture 1**

- Driver / Coupler
  - Atmosphere
  - Ocean
  - Physics
  - Dynamics

**Architecture 2**

- Driver
  - Atmosphere
  - Atm2Ocn
  - Ocean
  - Physics
  - Phys2Dyn
  - Dynamics

**Architecture 3**

- Atmosphere
  - Ocean
Feature Diagrams versus Other Kinds of Software Diagrams

• Software process diagrams, such as a UML Activity Diagrams and Sequence Diagrams show behavioral aspects of a software system.

• Feature diagrams are not behavior diagrams, but supported behaviors can be represented as features.
  – e.g., interpolation, domain decomposition
# Technologies Reviewed

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Name</th>
<th>Reference</th>
<th>Latest Released Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFG2</td>
<td>Bespoke Framework Generator</td>
<td>[12]</td>
<td>bfg2-beta</td>
</tr>
<tr>
<td>ESMF</td>
<td>Earth System Modeling Framework</td>
<td>[6]</td>
<td>ESMF_4_0_0rp2</td>
</tr>
<tr>
<td>FMS</td>
<td>Flexible Modeling System</td>
<td>[13]</td>
<td>Riga (internal)</td>
</tr>
<tr>
<td>MCT</td>
<td>Model Coupling Toolkit</td>
<td>[14]</td>
<td>2.6.0</td>
</tr>
<tr>
<td>OASIS/PSMILe</td>
<td>Ocean Atmosphere Sea Ice Soil / PRISM System Model Interface Library</td>
<td>[1]</td>
<td>OASIS4</td>
</tr>
</tbody>
</table>
Approach

• The feature analysis we conducted is based on information found in technical documentation that accompanies the coupling technologies (e.g., programming guides, user manuals) as well as published scientific literature.

• The initial feature analysis was conducted bottom-up by collecting a list of over 100 features used by at least one of the technologies.

• We dealt with complexity by abstracting related sub-features into common higher-level features, producing a hierarchy six levels deep.

• We have undergone several refactoring phases—reorganizing the feature diagram to achieve clearer levels of abstraction.
Difficulties

• We are not domain experts
• We want feature terms to be precise, while avoiding jargon and appealing to the whole community
• We sometimes had to chose among several terms describing the same concept
• When features from different base technologies overlapped, we had to distill out what the essential capability was
  – While maintaining orthogonality
Coupling Technologies
Feature Diagram

- Supported Computational Environment
- What's Being Coupled
- Coupling Architecture
- Coupling Tasks
Computational Environment

- Platform
  - Supercomputer
  - Workstation / laptop

- Operating System
  - Unix (Linux, BSD, AIX, OSX)
  - Windows (98, NT, 2K, XP, Cygwin)

- Compiler
  - C
  - C++
  - Fortran77
  - Fortran90

- Processor Count
  - Single
  - Multiple
What’s Being Coupled

- Control Interface
- Data Interface
- Mode
  - Active component
  - Passive (Data-only)
  - Null/Dead/Stub
  - Utility (I/O)
- Pre-defined Scientific Interface
  - Atmosphere
  - Ocean
  - Ice
  - Land
- Language Bindings
  - Fortran (77,90)
  - C/C++
  - Java
  - Python
  - Matlab
- Numerical Grid

Complete feature set provided by gridspec
Control and Data Interfaces to Models

- What's being coupled
  - Control Interface
  - Data Interface

Software Interface

- Generic
  - Call-return / argument passing
  - Shared memory
  - File
  - Named pipe
  - Socket
  - HTTP
  - Web Service
  - Asynchronous (alarm, exception, event handling)

- Adopted from General Purpose Library API
  - PVM
  - MPI (1, 2)
  - SVIPC
  - TDT

- Coupling Technology-Specific API
  - e.g., PSMI, ESMF init/run/finalize

- Non-Functional Characteristic
  - SSH security
  - Synchronization (blocking, non-blocking)
  - Buffering
  - Byte swapping
  - Block data transfer
  - Protocol
  - Extensibility

Data Types and Packaging

- Primitives
  - Integer
  - float / real
  - double
  - char

- Composites
  - structure
  - array (static, dynamic)

- Coupling Technology-Specific Packaging
Coupling Architecture

Applied recursively at each coupling context. A coupling context involves at least two modules and a coupler.

- Inversion of control
- Embedded
- Sandwich
- Central Registry
- Point to Point
- Mediator
- Run-time reconfiguration
- "Meta-coupler"/Generated framework
- Direct reference between components
Execution Model

- Coupling Architecture
  - Execution Model
    - Parallelism
      - SPMD
      - MPMD
    - Concurrency
      - Sequential
      - Concurrent
      - Hybrid
    - Multi-threading
    - Memory
      - Shared
      - Distributed
Driving

Locus of Control
- Model
- Coupler
- Driver

Visibility
- Implicit
- Explicit

Hierarchical

Invocation Ordering Mechanism
- Fixed
- Varying
- Constraint-based

Termination Control Mechanism
- Preset limit
- Convergence

Schedule

Model Staging
- Init
- Run
- Finalize
- Multi-Phased

Startup
- Just driver
- Driver and components
Coupler

- Manifestation
  - Embedded in model code
  - Separate executable
  - Subroutine
  - Combination

- Communication Context

- Endpoint Cardinality
  - Two
  - Greater than two

- Generality
  - Coupling context-specific
  - Pluggable

- Access to Scientific Content
  - Hooks
  - Embedded
Coupling Tasks
Configuration

- Mechanism
  - XML
  - Text
  - Checkout/configuration parameter
  - Compile parameter
  - Runtime parameter

- Topology
  - Which components participate
  - Component-processor mapping
  - Coupling field connections

- Run
  - Start time
  - Stop time
  - Time step
  - Grid type
  - Grid resolution
  - Component schedule
  - Exchange protocols and period

- Data
  - Fields (type, size)
  - Initial conditions
  - Boundary conditions
  - Physical constants
Coupling Pre-Run

- Domain Decomposition
  - Provided Centrally
  - Provided Locally (distributed)

- Field Initialization
  - Master process
  - Subprogram
  - Component itself
  - File (namelist, Netcdf, XML)
  - Field metadata

- Acquire Interpolation Weights
  - File
  - User-provided
  - Calculated by Coupling Technology

- Calculate Halo Communication Paths

- Model Subroutine Registration

- Conformance Checking
What’s Next?

• Validation of the feature model
  – Tie features back to the technology from which they came
  – Compare to an alternative feature diagram which describes production model couplers
  – Discussions with model developers/scientists

• Next phase of Cupid
  – Improve domain specific language and code generator to handle more complex coupling requirements
Want to know more?


• http://rockydunlap.wordpress.com
Thanks!
Extra Slides
Verification and Validation

• Verification and validation are two ways to evaluate software products:
  – Verification asks “Are we building the system right?”
  – Validation asks “Are we building the right system?”

• Both are related to software specifications:
  – Verification: Determine if the software actually implements the specification
  – Validation: Determine if the specification (through the software) meets the need for which it is intended
How do you do verification?

• Code inspection
  – Manual comparison with specification
• Static analysis
• Unit testing
• Integration testing
• Generate provably correct code from the specification
  – Domain specific language + code generator
Evaluation Techniques for Coupled Climate Models

• Climate models undergo rigorous evaluations:\[^{1}\]
  – Model Inter-comparisons (e.g., AMIPs, IPCC)
  – Ensemble runs (multi-model, perturbed physics, multiple versions of same model)
  – Testing against past and present climate

• Simplified tests:\[^{2}\]:
  – Change 3D equations of motion to 2D shallow-water equations with analytical or reference solutions
  – Isolate physical parameterizations on a single column
  – Isolate dynamical core with idealized physics

[^1]: See chapter 8 of the latest IPCC Assessment Report.
Observation

• Current climate model evaluations are heavy in validation techniques, but verification techniques are largely absent
• Those verification techniques that are in use are primarily black-box techniques.
  – Given a certain input, ensure that the model produces the expected output
• Why is this?
  – Hard to give a complete answer
  – I argue it is directly related to the sheer size and complexity of climate model source code.
  – Model complexity is the number one enemy of verification.
• Is this a problem?
  – Yes! Without verification, you cannot say with confidence exactly what you are validating.
Coupling *should* help reduce complexity and enable verification

- The good news is that most General Circulation Models (GCMs) and Earth System Models (ESMs) are built in a modular fashion.
- In general, modularity provides:
  - Separation of concerns
  - Ease of understanding
  - Ease of maintenance and evolution
  - Forces you to think about module interfaces
- But…
  - Even if the code is modular, the couplers themselves become highly complex!