Feature Analysis of Coupling Technologies for Climate Models

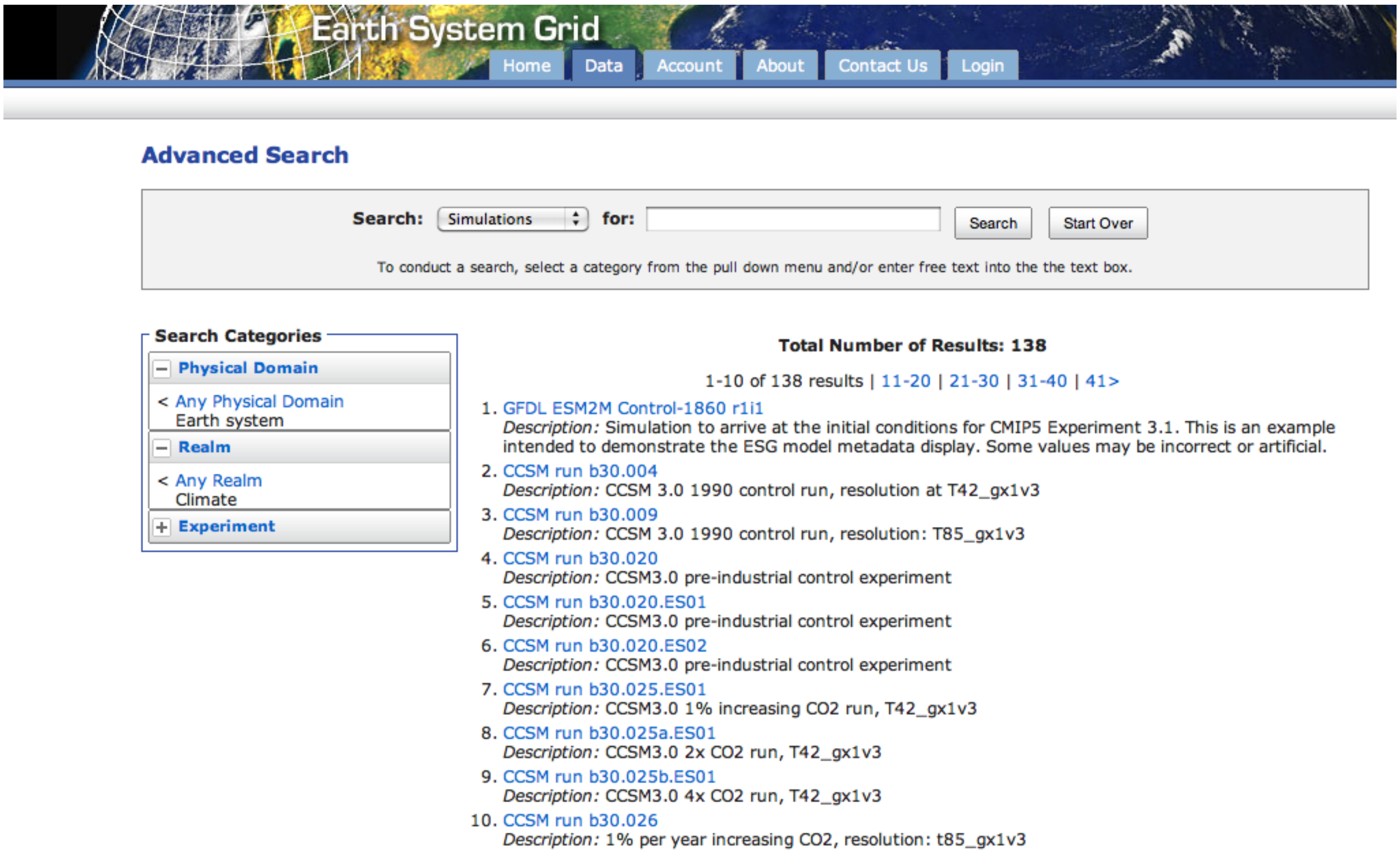
Spencer Rugaber, Rocky Dunlap and Leo Mark

College of Computing
Georgia Institute of Technology

Earth System Curator

- Premise: "The descriptors used for comprehensively specifying a *model* configuration are also needed for a scientifically useful description of the model output *data."*
- Deliverables
  - Archive and query Earth system models, experiments, model components, and model output data
  - Ability to perform technical compatibility checking
  - Ability to auto-assemble components including automatic code generation of simple couplers

Curator Portal



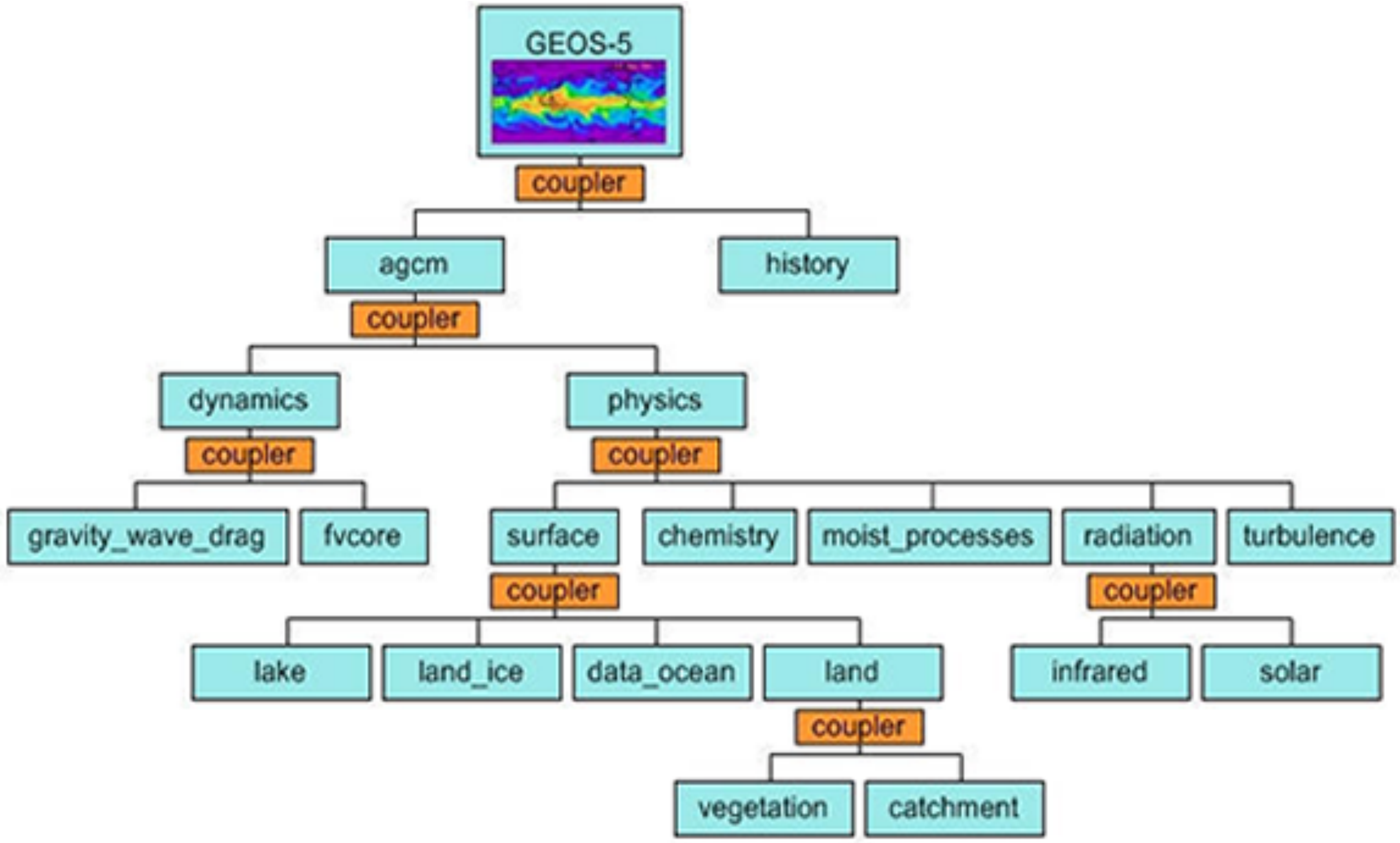
Partners

- Earth System Modeling Framework: NCAR, NESII/CIRES/NOAA
- Earth System Grid: NCAR
- Geophysical Fluid Dynamics Laboratory, NOAA
- Department of Earth, Atmospheric and Planetary Science, MIT
- Department of Earth and Atmospheric Sciences, Georgia Tech
- European Metafor project
- Sponsored by the National Science Foundation

Coupled Climate Models

- Multiple models (e.g. atmosphere and ocean) give more accurate predications than do single ones
- The software components that link together and mediate interactions between models are called *couplers*
- Existing coupling technologies: libraries, frameworks
- Our goal is to add a third: automatic coupler generation

Example Coupled Model



Key Design Tradeoff When Introducing a New Technology

- Reduce adoption barriers
  - *Componentizing:* Cost of adapting models to interact with coupling technology
  - Risk of changes to legacy code
- Avoiding the costs of a general solution
  - Compromised efficiency, an essential requirement of earth system models
  - Conformance with the software architectures of the models being coupled

Solution: Configurable Coupler Generation

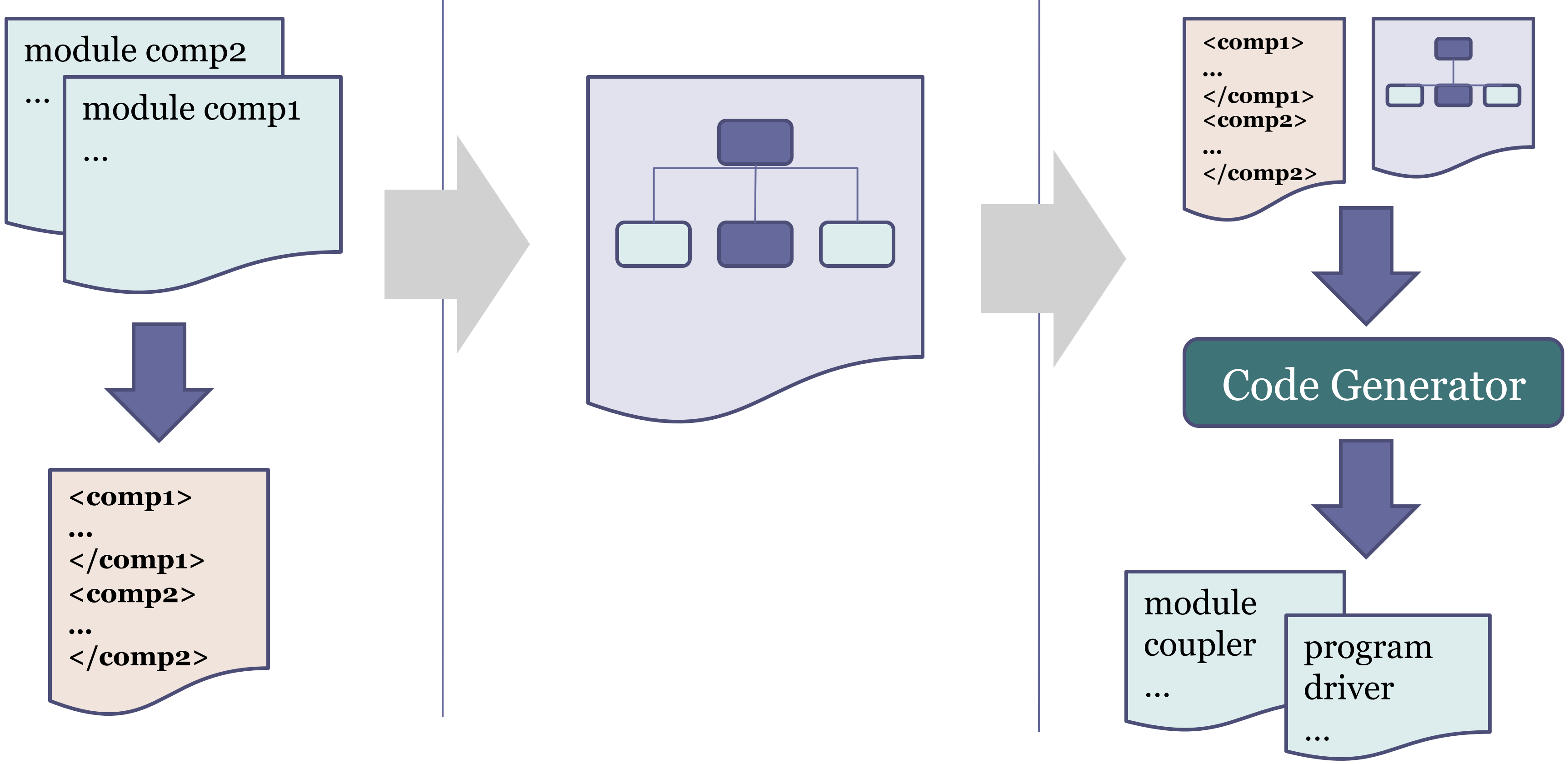
- We have made use of a software engineering technique called *generative programming*, which generates couplers based on a declarative requirements specification
- Couplers can be seen as members of a family of modules with similar requirements
  - Data communication among models
  - Data transformation and interpolation
  - Management of parallel computing resources

Generative Programming Process

1 **Describe existing components based on source code**

2 **Describe a coupling configuration**

3 **Generate the source code for a coupler and a driver**



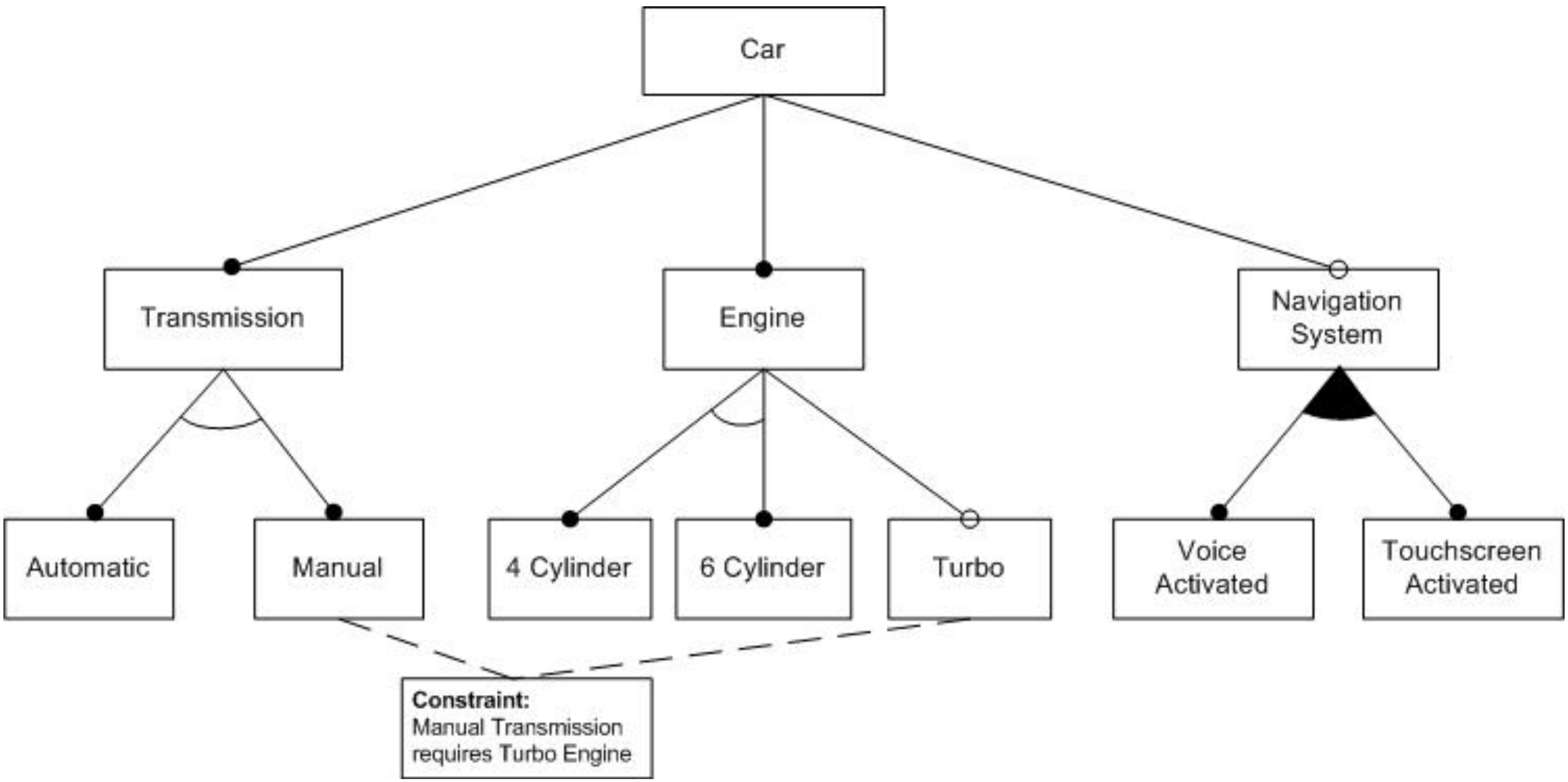
Feature Analysis

- A key step in generative programming is *feature analysis*, which understands a set of related technologies by organizing their features along orthogonal dimensions
- 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*, describing a desired family member
- An automated generator can then be used to produce the actual code for that member

Feature Diagrams

- A feature model is expressed as a *feature diagram*—an annotated tree in which nodes represent features in the domain, where 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

Example



Car
Transmission
Engine
Navigation System
Automatic
Manual
4 Cylinder
6 Cylinder
Turbo
Voice Activated
Touchscreen Activated
Constraint:
Manual Transmission requires Turbo Engine

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

Constraints

- Feature diagrams may also contain textual 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

Major Technologies Reviewed

Acronym	**Full Name**	**Reference**	**LastestLatest Released Version**
BFG2	Bespoke Framework Generator	[12]	bfg2-beta
ESMF	Earth System Modeling Framework	[6]	ESMF_4_0_0rp2
FMS	Flexible Modeling System	[13]	Riga (internal)
MCT	Model Coupling Toolkit	[14]	2.6.0
OASIS/PSMILe	Ocean Atmosphere Sea Ice Soil / PRISM System Model Interface Library	[1]	OASIS4
TDT	Typed Data Transfer	[2]	12 June 2008

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

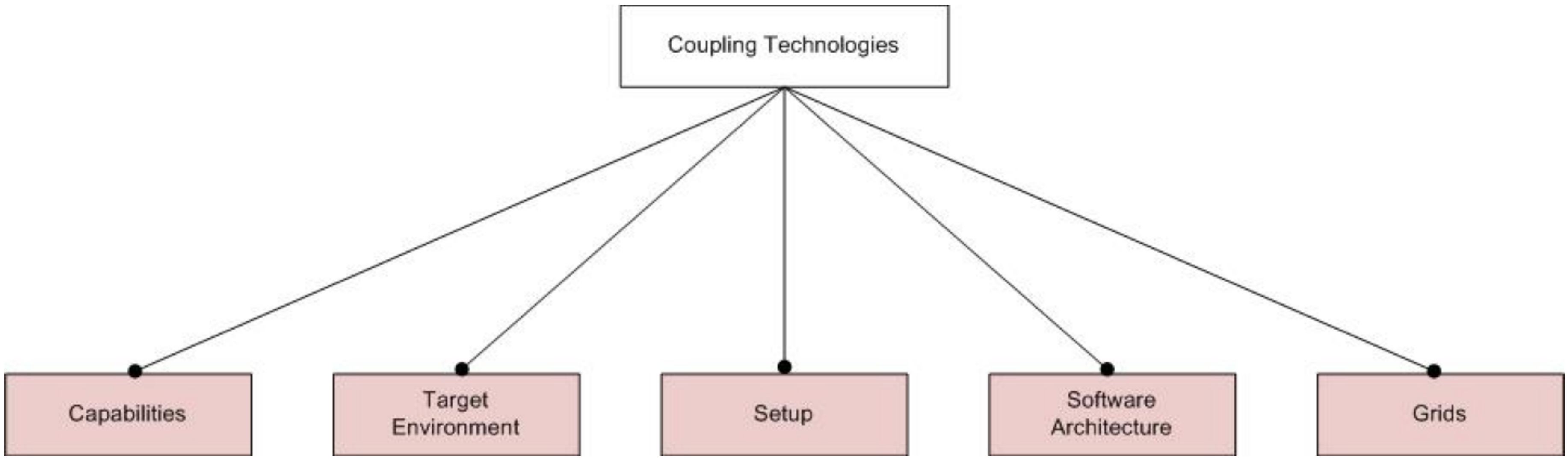
Difficulties

- We are not domain experts
- We sometimes had to synthesize a term from instances describing roles played by existing couplers
- We sometimes had to chose between terms describing the same concept
- When features from different base technologies overlapped, we had to distill out what the essential capability was

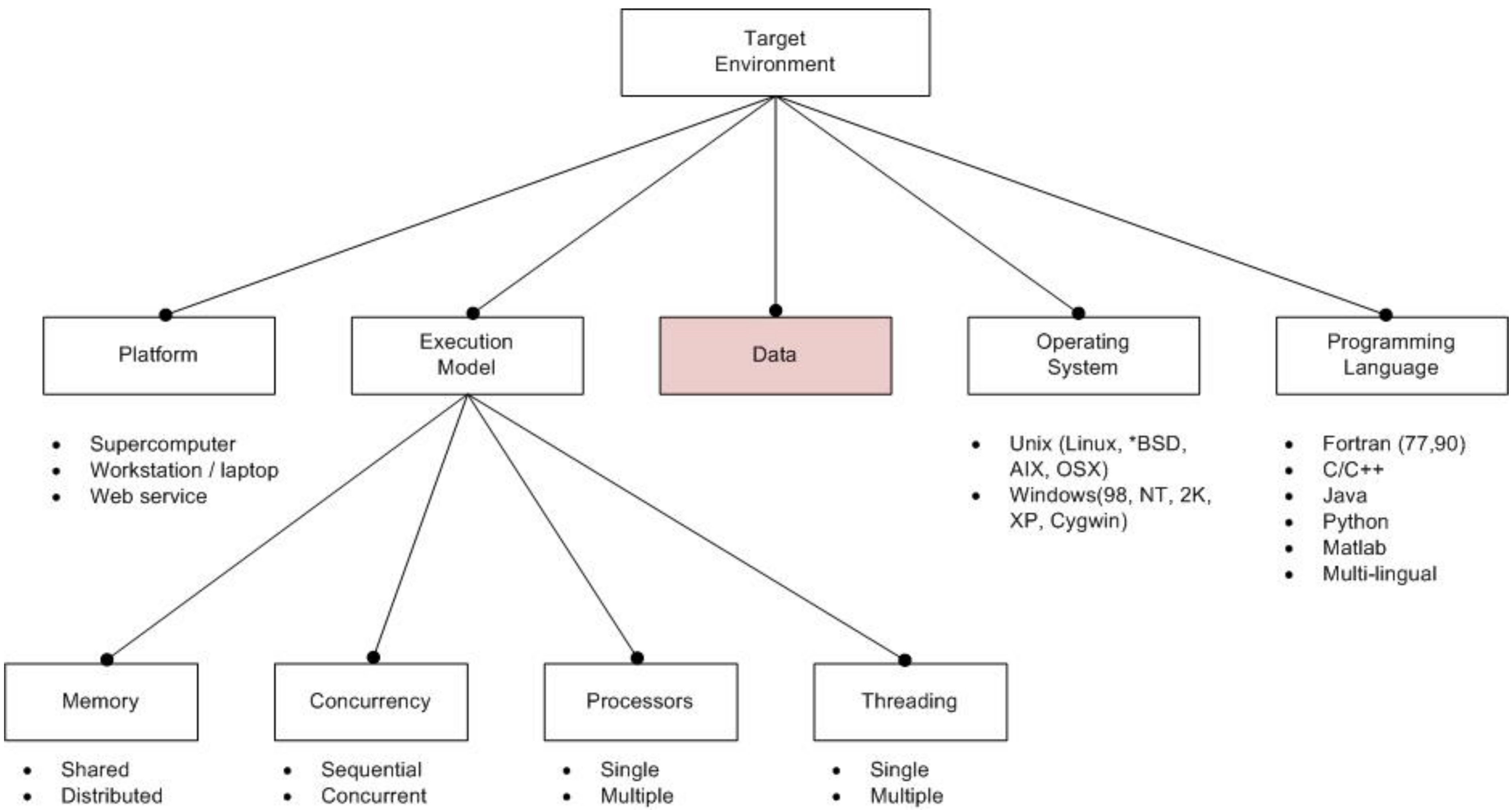
Orthogonality

- One of the key goals of feature analysis is orthogonality
- Features are orthogonal to the extent they express independently selectable capabilities
- Orthogonality promotes the idea of separation of concerns and the ability to reason about a single feature without importing non-related aspects from other features

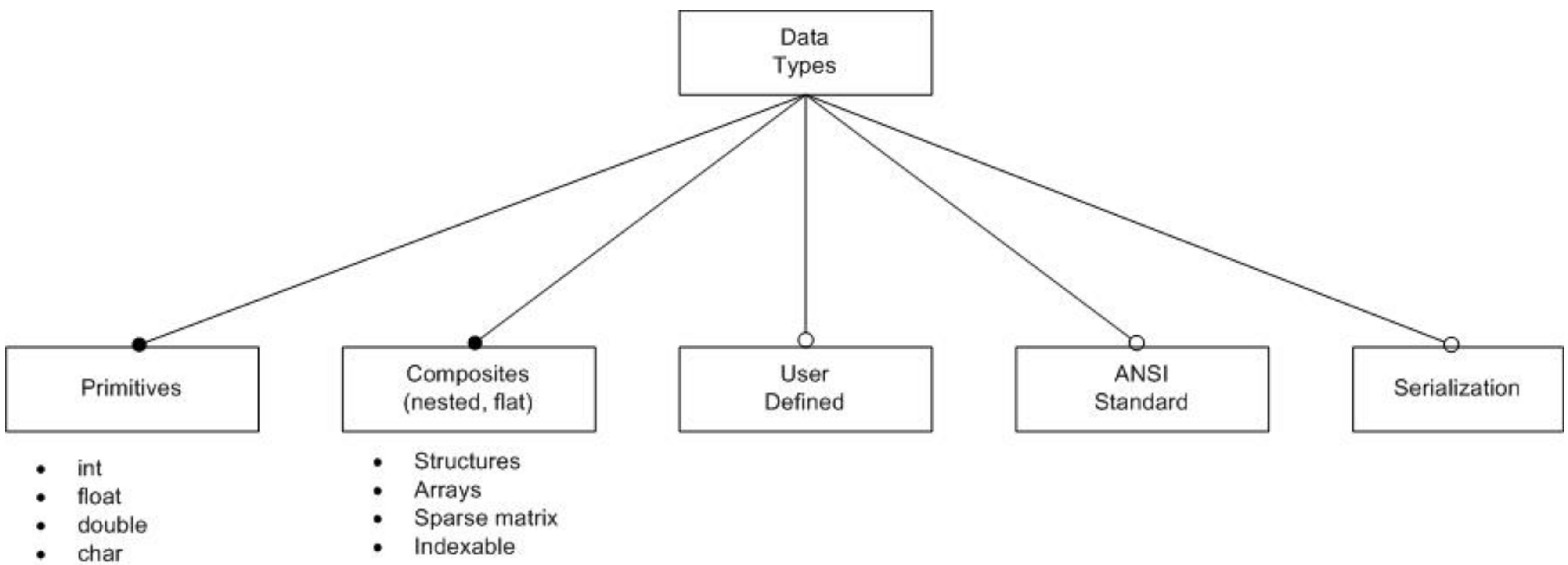
High Level Breakdown



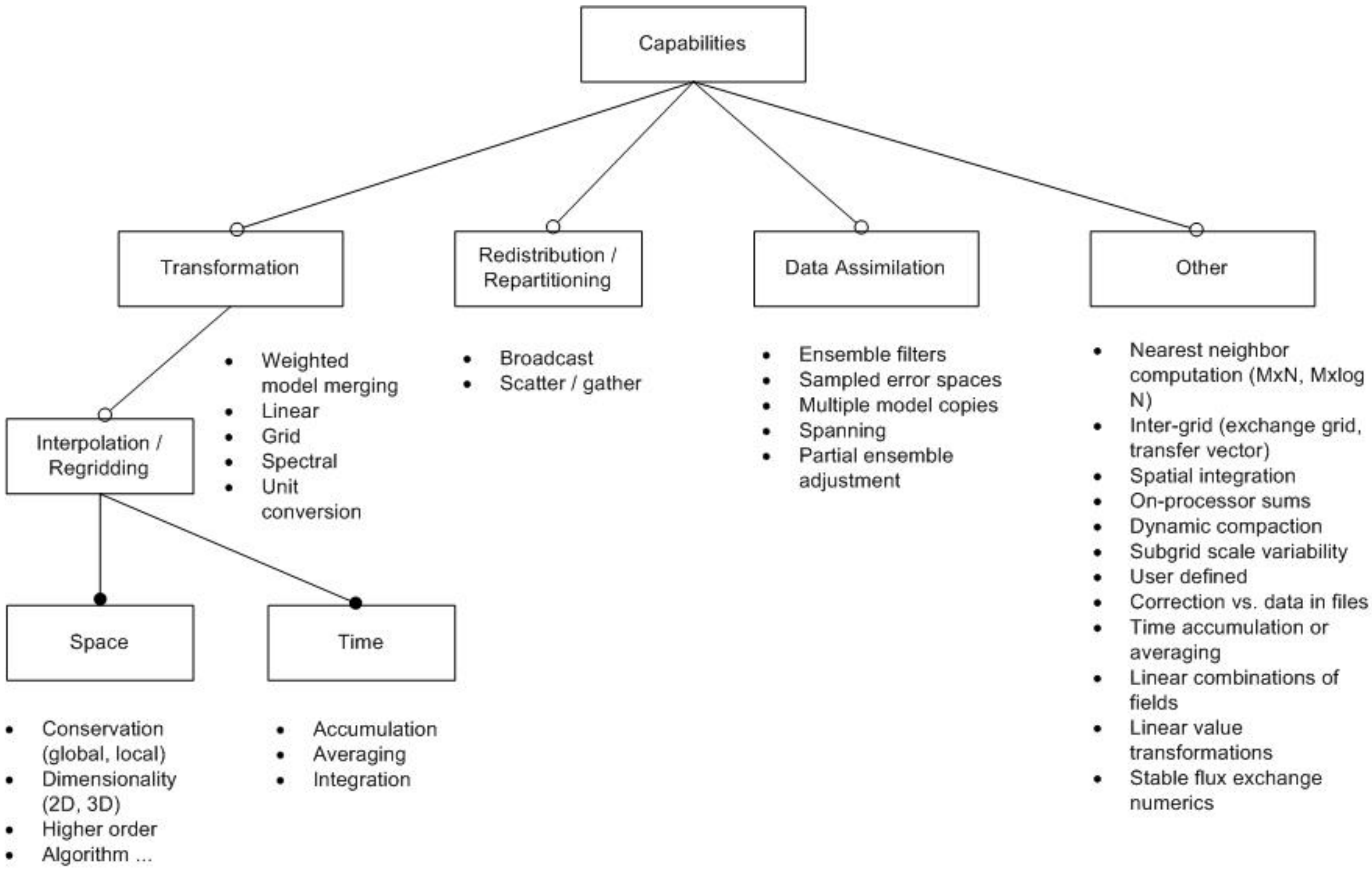
Target Environment



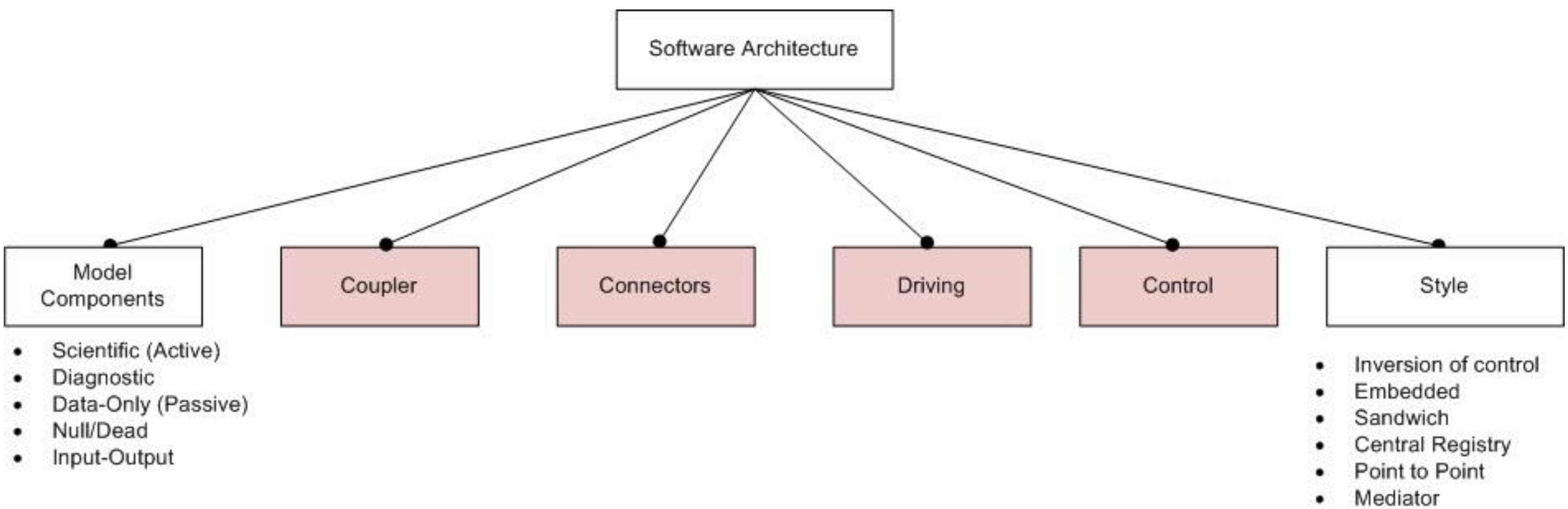
Data


Data
Types
Primitives
Composites
(nested, flat)
User
Defined
ANSI
Standard
Serialization
int
float
double
char
Structures
Arrays
Sparse matrix
Indexable

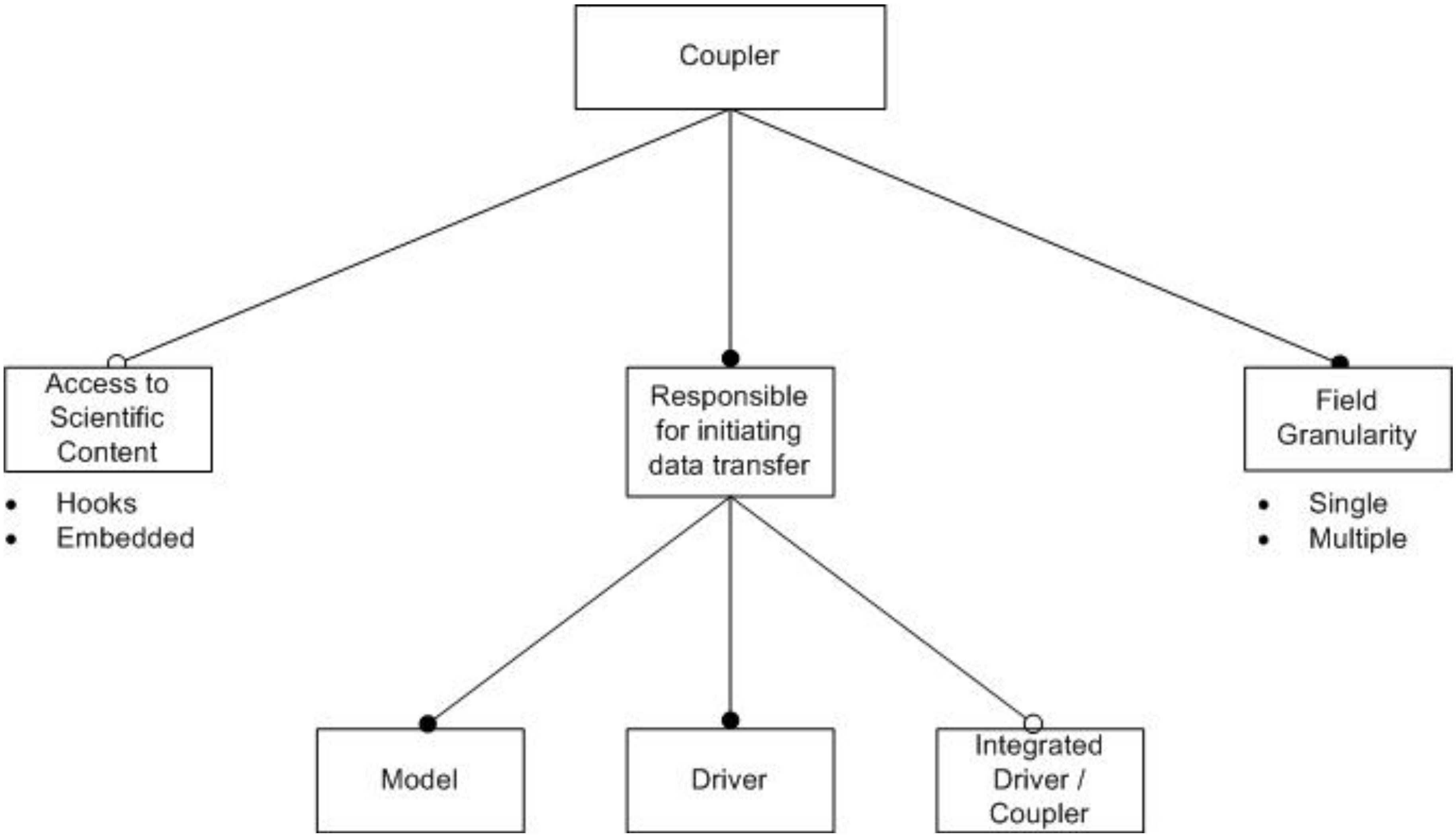
Capabilities



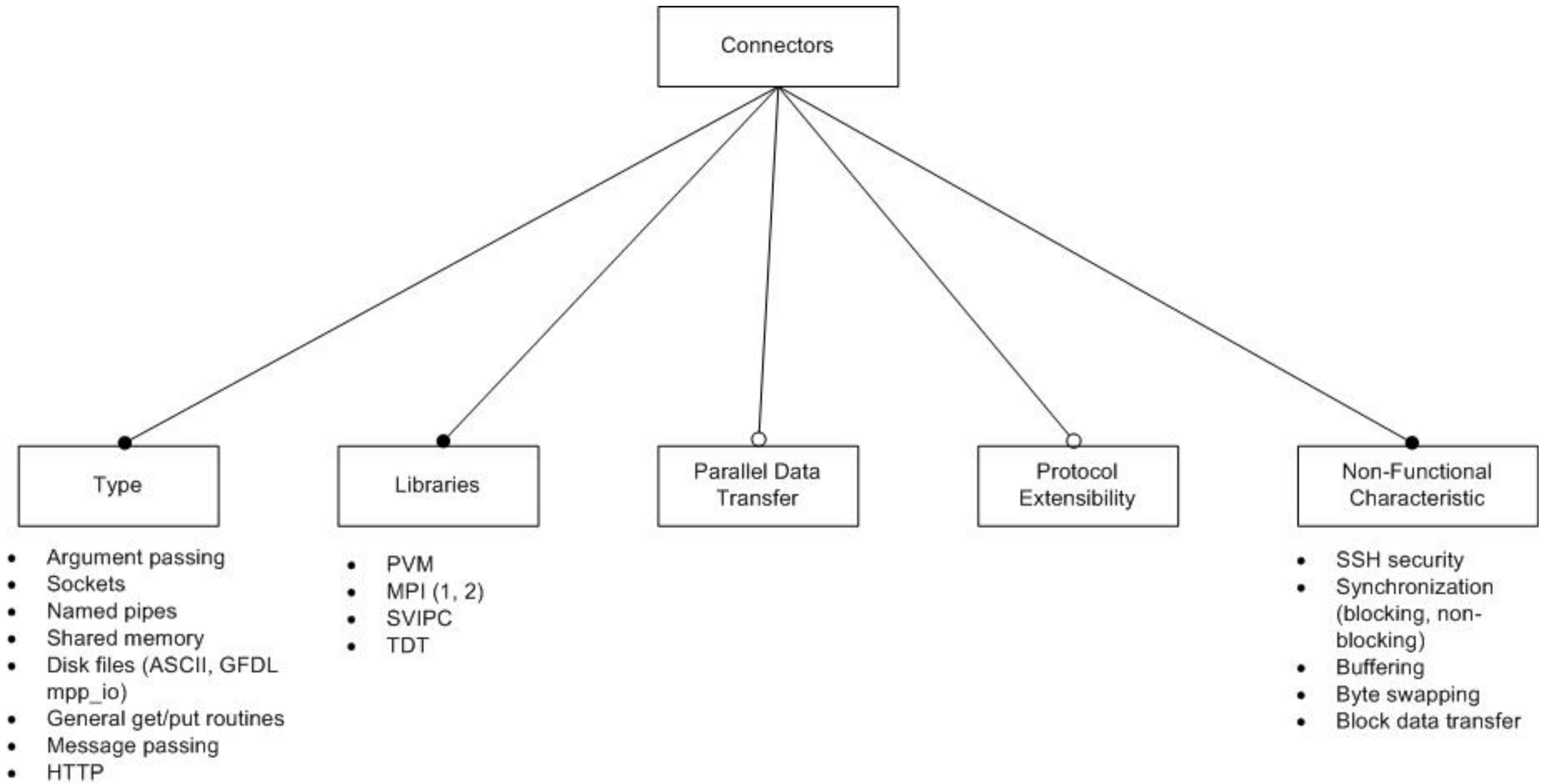
Software Architecture



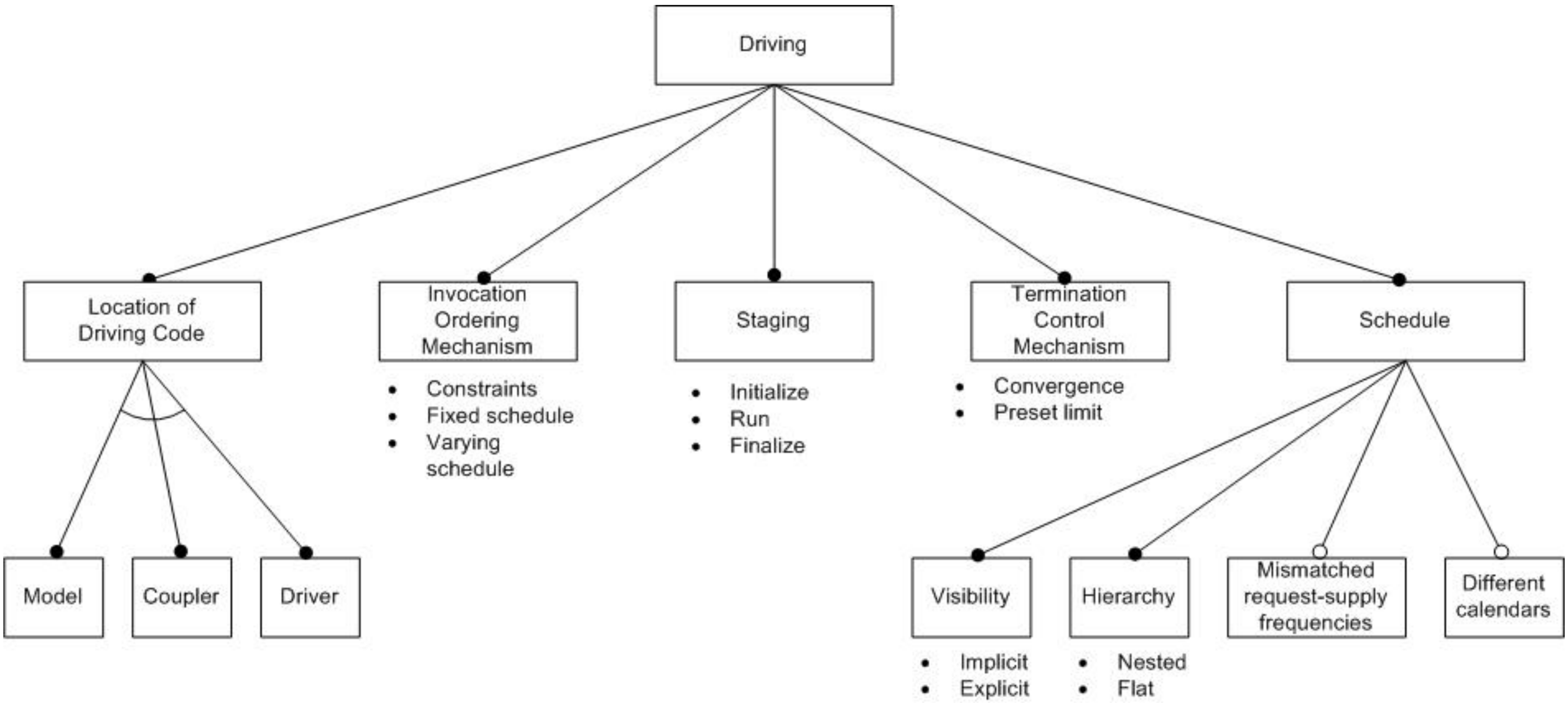
Coupler


Coupler
Access to Scientific Content
• Hooks
• Embedded
Responsible for initiating data transfer
Field Granularity
• Single
• Multiple
Model
Driver
Integrated Driver / Coupler

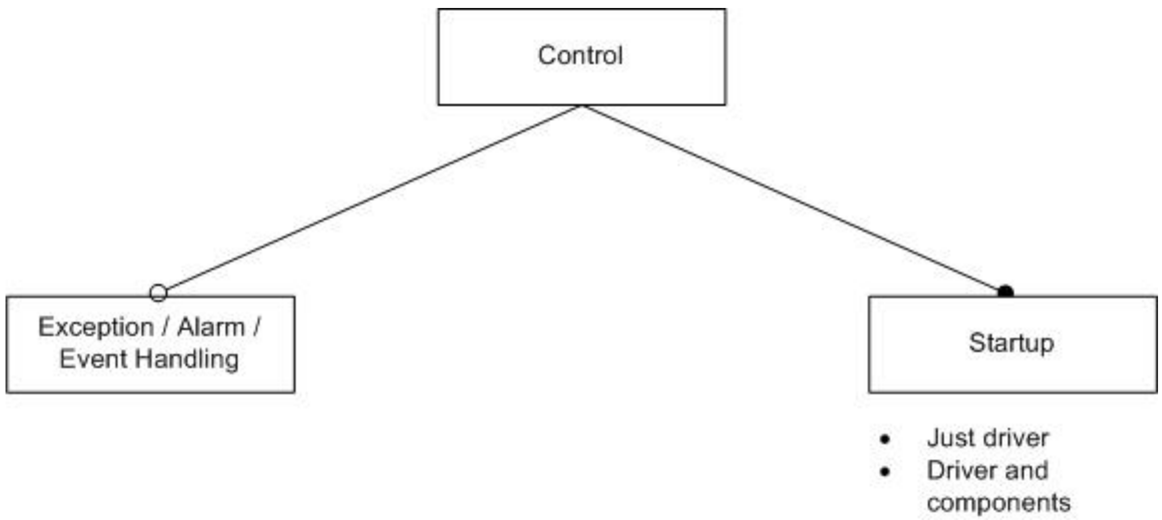
Connector


Connectors
Type
Libraries
Parallel Data Transfer
Protocol Extensibility
Non-Functional Characteristic
• Argument passing
• Sockets
• Named pipes
• Shared memory
• Disk files (ASCII, GFDL mpp_io)
• General get/put routines
• Message passing
• HTTP
• PVM
• MPI (1, 2)
• SVIPC
• TDT
• SSH security
• Synchronization (blocking, non-blocking)
• Buffering
• Byte swapping
• Block data transfer

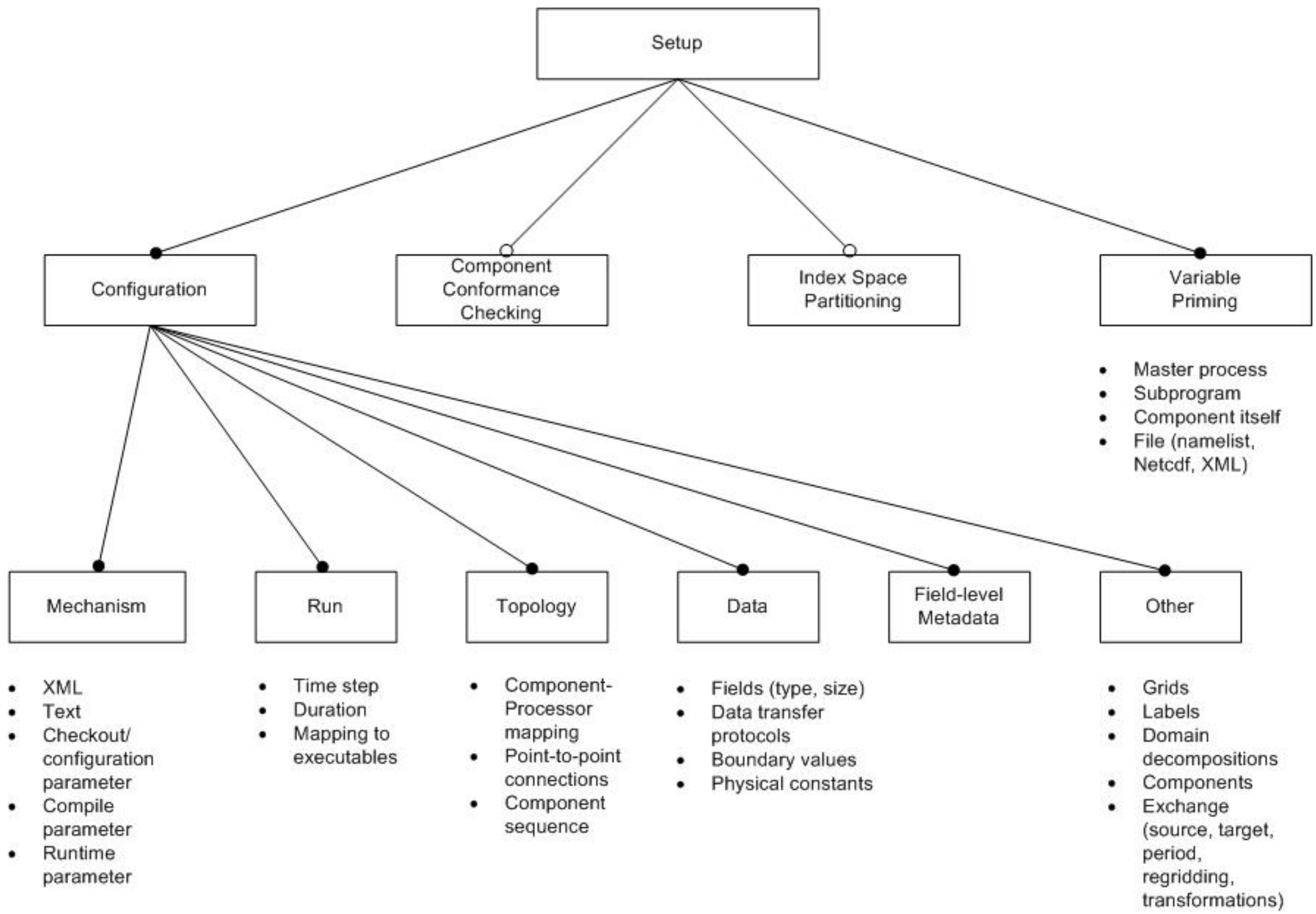
Driving


Driving
Location of Driving Code
Invocation Ordering Mechanism
Staging
Termination Control Mechanism
Schedule
• Constraints
• Fixed schedule
• Varying schedule
• Initialize
• Run
• Finalize
• Convergence
• Preset limit
Model
Coupler
Driver
Visibility
Hierarchy
Mismatched request-supply frequencies
Different calendars
• Implicit
• Explicit
• Nested
• Flat

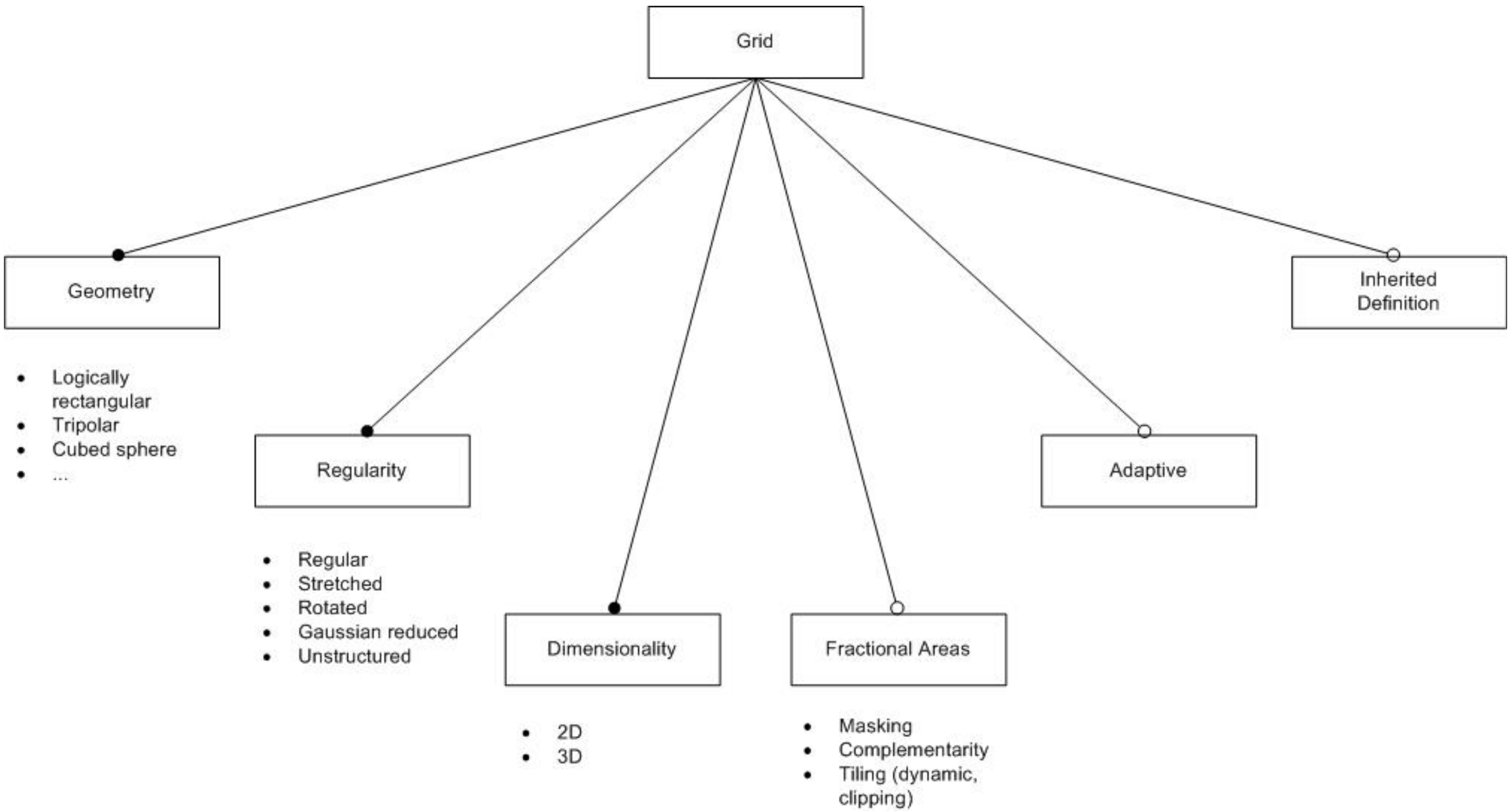
Control


Control
Exception / Alarm / Event Handling
Startup
Just driver
Driver and components

Setup


Setup
Configuration
Component Conformance Checking
Index Space Partitioning
Variable Priming
Master process
Subprogram
Component itself
File (namelist, Netcdf, XML)
Mechanism
Run
Topology
Data
Field-level Metadata
Other
XML
Text
Checkout/ configuration parameter
Compile parameter
Runtime parameter
Time step
Duration
Mapping to executables
Component-Processor mapping
Point-to-point connections
Component sequence
Fields (type, size)
Data transfer protocols
Boundary values
Physical constants
Grids
Labels
Domain decompositions
Components
Exchange (source, target, period, regridding, transformations)

Grids*



* The material in this section is an impoverished version of the feature analysis performed to produce the GFDL grid spec

Code Generation Status

- Cupid coupler generation environment
- A high-level, framework-specific language for describing and writing ESMF-based applications
- A static validation engine for checking for consistent and correct usage of ESMF
- A code generator for automatically writing the FORTRAN for ESMF couplers and drivers
- Technologies used
  - Eclipse/EMF, ANTLR, StringTemplates

Validation

- Tie features back to the technology from which they came
  - Vetting via technology purveyors
- Take an existing coupled model (CESM) and see how it uses coupling technology
- Discussions with modelers/scientists

Other Uses of Feature Analysis

- Configuration on the cloud
- Metadata validation
- Stimulate community discussion
- Ultimately, interoperability
- Application to other fields

Want to Know More?

- Rocky Dunlap, Spencer Rugaber, Leo Mark. "A Feature Model of Coupling Technologies for Earth System Models." Technical Report GT-CS-10-18, October 5, 2010, http://www.cc.gatech.edu/~rocky/papers/coupler_features_v1.pdf
- R. Dunlap, et al., "Earth System Curator: Metadata Infrastructure for Climate Modeling," *Earth Science Informatics,* 1(131-149), 2008.
- Workshop: “Coupling Technologies for Earth System Modelling : Today and Tomorrow” CERFACS, Toulouse France - December 15th to 17th 2010.
- Web site: http://www.earthsystemcurator.org
- Email: spencer@cc.gatech.edu