18th World IMACS / MODSIM Congress, Cairns, Australia 13-17 July 2009
http://mssanz.org.au/modsim09

A Proposed Checklist for Building Complex Coupled Models

[1,2,3]**J. W. Larson**

[1]Mathematics and Computer Science Division, Argonne National Laboratory
9700 S. Cass Avenue, Argonne, IL 60439, USA
E-Mail: larson@mcs.anl.gov
[2] Computation Institute, University of Chicago and Argonne National Laboratory
Chicago, IL 60637, USA
[3] Department of Computer Science, The Australian National University
Canberra ACT 0200, Australia

Keywords: *Multiphysics Systems; Multiscale Systems; Coupled Systems; Parallel Computing; Software Engineering*

ABSTRACT

Coupled problems occur widely in the pure and applied sciences and engineering. Two types of models are becoming more common—multiscale models, which couple physical phenomena operating on different spatiotemporal scales, and multiphysics models, which couple distinct natural phenomena. The modelling of coupled systems is an emerging discipline that is now the focus workshops and conferences. Many groups have created model coupling software, spanning a spectrum ranging from custom-made, problem-specific solutions and application domain-specific frameworks to open-source and commercial generic coupling infrastructure packages.

Constructing complex coupled models from numerous interacting models—or *constituents*—can be notoriously difficult. Coupled models exhibit *knowledge*, *software*, and *algorithmic / computational* complexity. Interdisciplinary teams are the mechanism used to master knowledge complexity. But how does one grapple with the other two types of complexity? Current practice is an ad hoc approach in which an interdisciplinary group achieves their goal by "just figuring it out" (JFIO). The problem with JFIO is it often begins with a team feeling thwarted and overwhelmed, proceeds without a clear roadmap, entails generous amounts of trial-and-error, and is prone to surprises that can cause delays. Furthermore, in software engineering terms JFIO is of low-level process maturity (e.g., not reproducible).

Coupled systems thus far built typically have relatively few ($\sim 5 - 10$) constituents. Any attempt to tackle more complex system modelling exercises such as coupling climate to energy and economics models or simulating a whole organism like the human body will require coupling of a significantly larger number of constituents. It is doubtful that JFIO will scale to meet such a challenge.

Ideally, one would like a "cook book" approach to building a coupled model. In response to this desideratum I propose a methodology comprising a series of exercises and associated questions whose object is to define the requirements for a software implementation of a coupled system. This approach is distilled from years of experience as one of the software architects for the coupling infrastructure for the Community Climate System Model (CCSM; http://ccsm.ucar.edu) and my involvement in scientific community software framework projects such as the Earth System Modeling Framework (ESMF; http://www.esmf.ucar.edu), the Common Component Architecture (CCA; http://www.cca-forum.org), and Framework Application for Core-Edge Transport Simulation (FACETS; http://facetsproject.org). This analysis technique is also drawn from my own theoretical work on coupled models in which I have devised terminology, notation, and complexity metrics.

I review previous theoretical work on coupling that is relevant to the exercises outlined in this paper. I then propose a checklist to guide a developer through the requirements-gathering and early design processes for a coupled model. This checklist comprises a series of exercises to identify properties of the envisioned coupled system and to identify and analyse its couplings, elucidating their input/output relationships, types, frequency, and estimated overhead imposed on the constituents. The checklist also covers commonly encountered software engineering issues. This is a proposed checklist: The set of exercises and questions may not fit a given problem perfectly; I encourage the reader to extend, prune, or modify them as necessary. I discuss in brief how distributed-memory parallelism (DMP) complicates this analysis. I offer the reader some advice on how the exercises need to be modified for application to DMP. I conclude that the approach is usable for situations in which coupling occurs in a single address space, but under DMP may become sufficiently laborious that automation may be required.

1 INTRODUCTION

Natural and human systems are frequently complex and composed of numerous interacting parts. These interactions are called *couplings*, and simulation these systems requires *coupled models*. In recent years, multiphysics and multiscale models have become more prevalent, spawning a specialty area that has its own conferences and journals. A wide variety of coupling software mechanisms (Michopoulos et al. [2005] and references therein) have been built, including a number of generic coupling products (Larson et al. [2005]; Joppich et al. [2006]; COMSOL, Incorporated [2009]; Lethbridge [2004]). A tool set exists, but what tools does one need in constructing a new coupled model, and how are they to be applied? This is the central question facing anyone wishing to build a coupled simulation system. Before any tool can be applied, the boundaries of the coupled model must be drawn, its parts and their interactions identified. Additionally, since these systems are almost always built from legacy codes, a strategy for their inclusion in the coupled system is required. Coupled models are built by interdisciplinary groups through an ad hoc process. Development of new coupled systems could benefit from a systematic approach for the requirements gathering and early design stages of software construction.

In this paper I propose a set of exercises and associated questions that reveal a system's coupling relationships and provide a framework for assessing the legacy parts from which the coupled system will be implemented. This methodology is my attempt to summarise and record what has worked in previous projects. It is also based on a theoretical framework for describing coupled systems whose descriptive power has been demonstrated in analysing an existing coupled system (Larson [2009]).

I address this paper to prospective coupled system builders. The proposed set of exercises and questions may not fit a given problem perfectly. I urge the reader to extend, prune, or modify the set of exercises and questions as necessary.

In Section 2 I summarise previous work on a conceptual framework for describing coupled systems, stating definitions and fixing notation relevant to this paper. In Section 3 I state a set of checklist questions coupled systems developers should ask themselves, and how answers to these questions relate to requirements.

2 ANATOMY OF A COUPLED SYSTEM

Below is a brief sketch of a coupled system; further details are given by Larson [2009].

A coupled system $\mathcal{M}$ has N subsystem models called *constituents* $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$; $\mathcal{C}_i$ solves its model equations for its *state variables* $\boldsymbol{\phi}_i$, using a set of *input variables* $\boldsymbol{\alpha}_i$ produced by other constituents, and produces a set of *output variables* $\boldsymbol{\beta}_i$ for consumption by other constituents. $\mathcal{C}_i$ has a spatial domain Γ_i; its boundary $\partial\Gamma_i$ is the portion Γ_i exposed to other models for coupling. $\mathcal{C}_i$ is likely to depend on the time t. The *state, input, and output fields* of $\mathcal{C}_i$ are $\mathbf{U}_i \equiv \boldsymbol{\phi}_i \times \Gamma_i$ $\mathbf{V}_i \equiv \boldsymbol{\alpha}_i \times \partial\Gamma_i$, and $\mathbf{W}_i \equiv \boldsymbol{\beta}_i \times \partial\Gamma_i$, respectively.

Coupling between $\mathcal{C}_i$ and $\mathcal{C}_j$ occurs if they coincide in time; their *coupling overlap domain* $\Omega_{ij} \equiv \Gamma_i \cap \Gamma_j \neq \emptyset$; and outputs from one constituent serve as inputs to the other, specifically $\mathbf{W}_j \cap \mathbf{V}_i \neq \emptyset$ and/or $\mathbf{V}_j \cap \mathbf{W}_i \neq \emptyset$, or the inputs $\mathbf{V}_i$ ($\mathbf{V}_j$) can be computed from the outputs $\mathbf{W}_j$ ($\mathbf{W}_i$).

Many types of couplings exist. Couplings may be classified based on temporal relationships between their outputs and inputs. *Instantaneous data delivery* is the provision of instantaneous values of $\mathbf{W}_j$ at one or more times per coupling event. *Integrated data delivery* occurs if $\mathbf{W}_j$ are integrated with respect to time over an interval Δt_{ij} and delivered to $\mathcal{C}_i$ as averages (accumulated fluxes) for (incremental) application as $\mathbf{V}_i$. Couplings may be classified based on the temporal relationship between $\mathbf{W}_i$ and $\mathbf{U}_i$ *Diagnostic coupling* occurs if the $\mathbf{W}_i$ are computed *a posteriori* from the $\mathbf{U}_i$. *Prognostic coupling* occurs if the $\mathbf{W}_i$ are computed as a forecast from $\mathbf{U}_i$. Another classifier for couplings is whether constituent states are divorced from their inputs and outputs. *Explicit coupling* occurs if there is no overlap in space and time between the $\mathbf{U}_i$ and $\mathbf{U}_j$. *Implicit coupling* occurs if there is space/time overlap in variables common to both $\mathbf{U}_i$ and $\mathbf{U}_j$, requiring a simultaneous, self-consistent solution. Couplings may be classified according to the number of constituents participating in a coupling event. *Bipartite* coupling involves two constituents; for explicit coupling one constituent provides data for the other's consumption, while in implicit coupling a self-consistent solution for the two parties' shared state is calculated. *Multipartite coupling* involves three or more constituents; for explicit coupling, multipartite couplings may be broken into individual events in which two or more constituents provide the same data for consumption by a third constituent—a process requiring *merging*.

In a computer implementation of $\mathcal{M}$, $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$ are usually numerical models, and their spatial domains and time are discretised by the discretisation operator $\hat{\mathbf{D}}_i(\cdot)$, resulting in spatial meshes $\hat{\mathbf{D}}_i(\Gamma_i)$ and $\hat{\mathbf{D}}_i(\partial\Gamma_i)$ and a set of timesteps $\{t_0, t_1, \ldots\}$. Thus, model input, output, and state data exist in the form of *state, input,* and *output vectors* $\hat{\mathbf{U}}_i \equiv \boldsymbol{\phi}_i \times$

$\hat{\mathbf{D}}_i(\Gamma_i)$, $\hat{\mathbf{V}}_i \equiv \boldsymbol{\alpha}_i \times \hat{\mathbf{D}}_i(\partial\Gamma_i)$, and $\hat{\mathbf{W}}_i \equiv \boldsymbol{\beta}_i \times \hat{\mathbf{D}}_i(\partial\Gamma_i)$, respectively.

For bipartite explicit coupling, a *coupling transformation* $\mathscr{T}_{ij} : \hat{\mathbf{W}}_j \rightarrow \hat{\mathbf{V}}_i$ converts $\hat{\mathbf{W}}_j$ to $\hat{\mathbf{V}}_i$; $\mathscr{T}_{ij}$ is a composition of a *mesh transformation* $\mathscr{G}_{ij} : \hat{\mathbf{D}}_j(\Omega_{ij}) \rightarrow \hat{\mathbf{D}}_i(\Omega_{ij})$ and a *field variable transformation* $\mathscr{F}_{ij} : \boldsymbol{\beta}_j \rightarrow \boldsymbol{\alpha}_i$. Intergrid interpolation is a simple example of $\mathscr{G}_{ij}$, but $\mathscr{G}_{ij}$ can be more general. The variable transformation $\mathscr{F}_{ij}$ is defined by natural law relationships between $\boldsymbol{\beta}_j$ and $\boldsymbol{\alpha}_i$ (e.g., black-body radiation fluxes from temperature computed by using the Stefan-Boltzmann law). In general, $\mathscr{G}_{ij} \circ \mathscr{F}_{ij} \neq \mathscr{F}_{ij} \circ \mathscr{G}_{ij}$; that is, the ordering of $\mathscr{F}_{ij}$ and $\mathscr{G}_{ij}$ is chosen by the coupled model developer and is a source of coupled model uncertainty. Multipartite explicit coupling proceeds via a *merging transformation* $\mathscr{M}$. If $\mathcal{C}_j$ and $\mathcal{C}_k$ provide coincidental (i.e., shared colocated variables) input to $\mathcal{C}_i$, then $\mathscr{M}_{ijk} : (\hat{\mathbf{W}}_j, \hat{\mathbf{W}}_k) \rightarrow \hat{\mathbf{V}}_i$ computes field variable, intermesh interpolation, and weighted merging operations, with their ordering chosen by the model developer, and again this order dependence is a source of coupling uncertainty.

For bipartite implicit coupling between $\mathcal{C}_i$ and $\mathcal{C}_j$, overlapping portions of $\mathbf{U}_i$ and $\mathbf{U}_j$ are computed by a *solver* $\mathscr{S}_{ij}$, which computes the self-consistent solution $(\hat{\mathbf{U}}_i, \hat{\mathbf{U}}_j)$, handling both numerical solution and intermesh transformation. Multipartite implicit couplings are handled in a similar fashion; for exmple, an implicit coupling between $\mathcal{C}_i$, $\mathcal{C}_j$, and $\mathcal{C}_k$ has a solver $\mathscr{S}_{ijk}$ that computes the self-consistent solution $(\hat{\mathbf{U}}_i, \hat{\mathbf{U}}_j, \hat{\mathbf{U}}_k)$.

The time evolution of $\mathcal{M}$ is marked by *coupling events*, which are either *scheduled* or *threshold-triggered*, the latter based on some condition satisfied by the constituents' states. A scheduled coupling may be *periodic*. If all of the couplings in $\mathcal{M}$ are periodic, there is a repeatable set of coupling events called a *coupling cycle*. It is possible to form ratios of typical coupling time intervals Δt_{ij} and constituent timesteps Δt_i and Δt_j, and classify couplings as *tight* or *loose*. Tight coupling occurs on intervals comparable to individual constituent timesteps. Loose coupling occurs on timescales of many constituent timesteps. Implicit coupling implies tight coupling, while explicit coupling is more likely to be loose coupling.

Construction of a coupled model amounts to solving the *coupling problem*: Given N constituents, their domains, and data dependencies, construct any required coupling transformations $\mathscr{T}_{ij}$, solvers $\mathscr{S}_{ij}$, and merging transformations and multipartite solvers, resulting in the coupled model $\mathcal{M}$.

The discussion of coupling thus far is equally applicable to a single global address space or a distributed-memory parallel system. Discussion of coupling in parallel systems is deferred until Section 3.3.

3 CHECKLIST

Below I list a set of exercises developers of new coupled models should undertake and questions they should ask early in the development process.

3.1 SYSTEM INTEGRATION SOFTWARE ENGINEERING ISSUES

Exercise 1 (Identification of the Product) *Define the system to be simulated, listing its constituents. Identify coupling relationships known a priori from known natural-law relationships between the subsystems.*

During this exercise, consider the following questions: What is to be achieved by creating a coupled system? What is its projected lifetime, and how will it change over its life cycle? Has somebody already solved the problem, either in the form of a coupled application that can be modified or a software framework or toolkit that can be leveraged to develop the system?

Exercise 2 (Identification of Legacy Parts and Their Condition) *List the available legacy codes corresponding to the subsystems of the coupled system, and assess their characteristics and overall condition.*

During this exercise, consider the following questions: Do all the subsystem parts exist? Will new ones have to be developed? What legacy codes will play the part of which constituents? For some constituents, are there multiple legacy codes under consideration for inclusion? In what condition are the constituent codes? Are they robust? Are they highly portable? How easy is it to build an executable from each constituent's source code?

A coupled system is more than the sum of its parts, but high-quality parts are essential. Legacy codes often build from source and work "well enough" for their core developers and users, but making one part of a larger system is expanding its user base to people unfamiliar with its foibles. Hardening all of the constituent codes to include error handling and shutdown procedures (with meaningful explicatory error messages!), documentation (at a minimum prologues for each function or routine explaining its argument list, what it does, and how it handles exceptions), and porting to the coupled

system's target set of compilers or operating systems. Arriving at a consensus on how the coupled application will be built from source is wise, as the build mechanism (e.g., `make` with `autoconf`) can then be implemented in the constituent codes. At this point, it is also important to formulate a strategy for avoiding code forks that can occur between the version of the legacy code maintained by its core developers and the version to be used in the coupled system.

Another issue worth considering early is implementation language(s). In how many languages will the coupled system be implemented? How many interlanguage barriers must be surmounted, and between which languages? Languages such as C, C++, Python, and Java have interoperability mechanisms, either directly or via C. However, Fortran, which is frequently used in scientific applications, may pose interoperability problems. Fortran 77 is immediately interoperable with C, and via it may be linked to other languages; this is due to Fortran 77's call-by-reference (CBR) standard in which function or subroutine argument data is provided as a memory address reference. The Fortran 90/95 standards, however, do not restrict themselves to CBR; and the only language interoperability solution is a compiler-by-compiler glue layer such as CHASM (Rasmussen et al. [2006]). The Fortran 2003 standard has a BINDC attribute that allows users to get compiler-generated C bindings for variables declared with this attribute (ISO/IEC Joint Technical Committee 1, Subcommittee 22, Working Group 5 [2004]); BINDC is now supported by a number of Fortran compilers (Chivers and Sleightholme [2007]). One solution for language interoperability that supports all the aforementioned languages, including Fortran variants and supports datatypes used in scientific computing (e.g., complex numbers and multidimensional arrays) is Babel (Dahlgren et al. [2004]).

In how many executables will the system be implemented—single or multiple? Single-executable applications are easier to launch and control, but combining multiple codes into a single executable may lead to symbol clashes during the linking stage that must be resolved. A multiexecutable approach minimises this risk but poses other questions: Will the multiple executables interact directly (e.g., message-passing or sockets) or via data files? If one executable fails, how should the rest of the system respond?

Large systems frequently possess sufficient computational complexity to warrant one or more forms of parallel computing to accomplish their work with reasonable turnaround time. Will parallel computing be utilised? If so, what parallelisation mechanism(s)—MPI, OpenMP, and so forth—and in which constituents?

Exercise 3 (Constituent Data Dependencies) *List the input/output data each constituent reads and writes from and to files.*

This exercise is the first cut at identifying interconstituent couplings. It also gets one to "draw a box" around the coupled system and decide what the eventual set of inputs/outputs the coupled system will read/write from/to files. At this point one should ask the following questions: Is this set of input/output data sufficiently complete that outputs from constituents can be mapped onto inputs of others and the system might be prototyped using file-based coupling? Once the coupled system is constructed, what will *its* file inputs/outputs be? Given this set of inputs/outputs for the coupled system, what data will be used to initialise and drive it, and how might its output data be evaluated against corresponding observational or experimental data?

3.2 INTERMODEL DATA DEPENDENCIES

If the results of Exercises 1 and 2 do not yield immediate and complete solutions to all of the coupling problems for the system, it is time to analyse the system's coupling relationships. These exercises will identify all of the couplings present in $\mathcal{M}$ and their nature.

Exercise 4 (Constituent Domain Table) *Make an $N \times N$ table whose rows and columns correspond to the N constituents in the system. Using the same ordering, fill in the descriptions of the constituents' domains as labels to the rows and columns of the table. For the (i, j)th cell, determine whether the overlap domain Ω_{ij} is empty or nonempty, marking cells consistently. Note this table is symmetric; the results of the (i, j)th and (j, i)th cells are identical. One needn't bother with the cells with $i = j$, as they are comparing a constituent's domain with itself.*

This identifies which of the $N(N-1)/2$ potential the first-order overlap domains for $\mathcal{M}$ are nonempty and is an indicator of coupling complexity. It also simplifies searches for multipartite couplings.

Exercise 5 (Constituent Data Diagram) *On a whiteboard or (large!) piece of paper, draw a circle representing each of the constituents, labeling it the same way as in Exercise 4. In each constituent's respective circle, describe the spatial domain Γ_i, its boundary $\partial\Gamma_i$, and the model time period the constituent is active. List the state, input, and output variables $\boldsymbol{\phi}_i$, $\boldsymbol{\alpha}_i$, and $\boldsymbol{\beta}_i$, respectively.*

Inspection of this constituent data diagram combined with the constituent domain table from Exercise 4 will identify all the system's coupling relationships. Within each constituent, compare its input, output, and state variables; intersections

between these sets identify potential implicit couplings if this constituent's domain also intersects with that of a constituent with corresponding state/input/output variable set intersections. For each pair of constituents whose domains intersect, compare their outputs with the other's inputs. Do these sets intersect, or is there a known natural law relationship connecting some of the output variables to the other's input variables? the answers will identify the coupling transformations $\mathscr{T}_{ij}$. If, among some of these pairs of constituents, their states are exposed as input/output variables over the same region and in time, there is an implicit coupling relationship that will require special attention in the form of a solver $\mathscr{S}_{ij}$.

Exercise 6 (System Graph) *Draw a graph $\mathcal{G}$ of the coupled system to be built. Start by using a circle to represent each constituent. Next, use the results of Exercises 4 and 5 to draw directed edges (arrow pointing from the provider to consumer) between constituents that have bipartite explicit couplings, an undirected edge (line segment with no arrows) for bipartite implicit couplings.*

This exercise allows the application of graph theory (Diestel [2006]; Larson [2009]) to $\mathcal{G}$ and the coupled system it represents. It raises and allows one to answer the following questions: How many constituents are being coupled, and how are they connected? How many couplings are there? Are the couplings explicit or implicit? Are there feedbacks (corresponding to directed loops connecting a constituent to itself via other constituents) present in the system? Are there vertices with arrows exclusively pointing in (out) to (from) them, corresponding to a sink (source)? If so, these are parts of the system that may be separated and run off-line, with sinks (sources) run after (before) the rest of the system, accepting inputs from (providing outputs to) the rest of $\mathcal{M}$.

The system graph simplifies the search for multipartite couplings. Annotate each edge in the graph from Exercise 6 with the names of the variables under exchange in the corresponding coupling. To search for multipartite explicit couplings, look at each vertex i that has multiple edges directed into it, and look at the sets of variables associated with each edge. If common variables are found, this is a potential multipartite explicit coupling directed into i—to be confirmed by looking for intersections between overlap domains. Any instance in which a constituent has undirected edges directly connecting it with two or more distinct constituents is a potential multipartite implicit coupling. Examine each of these clusters, identifying them by the number of parties participating in the implicit coupling, representing this multipartite coupling by a shaded blob—a *hyperedge*—encompassing their associated vertices in $\mathcal{G}$.

Exercise 7 (Coupling Frequency) *Copy the system graph $\mathcal{G}$ constructed in Exercise 6. Label each vertex i with the timestep Δt_i (if fixed) or the range of values it takes in the current legacy code. Estimate as best possible the likely time interval Δt_{ij} between coupling events, labeling the edge corresponding to this coupling.*

Consider whether Δt_i is likely to change (and how) in a coupled model use case. Label each edge or hyperedge indicating whether the coupling is scheduled or threshold-driven. For scheduled coupling, include estimates of the likely time interval between coupling events Δt_{ij}; note that Δt_{ij} may be hard to estimate without running the constituents $\mathcal{C}_i$ and $\mathcal{C}_j$ coupled together, since values of Δt_{ij} are frequently determined on grounds of numerical stability and quality of solution. For threshold-driven coupling, try to estimate the likely timescale over which coupling events will be spaced.

Exercise 8 (Coupling Data Volume) *Copy the system graph $\mathcal{G}$ constructed in Exercise 6, and annotate each vertex i with the data volumes of the state, input, and output vectors $(\hat{\mathbf{U}}_i, \hat{\mathbf{V}}_i, \hat{\mathbf{W}}_i)$ for the associated constituent $\mathcal{C}_i$. Label each edge ij with the data volume transformed in the coupling process.*

For each constituent or coupling, what is the ratio of the amount of data shared in coupling interactions to the amount of data used to compute internal state (namely, eqns. (6) and (7), respectively, in Larson [2009])? If these ratios are low, it may be possible to implement coupling mechanisms that rely on data copying, thus reducing the amount of modification of the legacy constituent codes. If these ratios are high, it might be necessary to rely on a more invasive approach such as a field data registry or even reorganising how a legacy constituent code lays out its data.

Exercise 9 (Agglomeration) *Study the results of Exercises 3–8 and look for output redundancy, that is, instances in which a constituent produces the same output (i.e., fields, times and locations) for multiple consumers.*

If a constituent is producing the same data for multiple consumers, is it possible to do this once? If so, there may be opportunities for coalescing some of these into a single output by introducing another constituent called a *coupler*. An excellent example of a system whose coupling overhead is reduced through introduction of a coupler constituent is CCSM; a thorough discussion of the CCSM coupler is given by Craig et al. [2005], and an analysis of its coupling relationships can be found in Section 4 of Larson [2009]. One can determine whether the system is better recast by introducing one or more couplers to reduce the overhead imposed on the "science" constituents by adding any couplers that might be useful and repeating Exercises 5–8 to evaluate this strategy.

At this point, all the system's couplings have been identified and classified as explicit or implicit, and the spatiotemporal data relationships between the parties in each coupling have been identified.

Exercise 10 (Coupling Transformations) *For each bipartite coupling between constituents $\mathcal{C}_i$ and $\mathcal{C}_j$, identify $\mathscr{F}_{ij}$ and $\mathscr{G}_{ij}$ and how these operations might implemented and ordered. For implicit couplings, think about how the solver $\mathscr{S}_{ij}$ will be implemented. For each instance of merging, think about how its merge transformation $\mathscr{M}$ will be constructed, and how its operations might be ordered. In all cases, how is the time t handled?*

Exercise 11 (Estimation of Throughput) *Assemble any performance data available for the constituents that have been implemented. If there is no data for the configuration (timestep, spatial resolution, parameter settings) for which the constituent is likely to be run as part of the coupled system, gather appropriates performance statistics. If some of the coupling transformations are already implemented, gather the same performance data.*

If these performance figures can be obtained for all of the constituents, it will be possible to estimate best-case scenarios for model throughput, that is, how much coupled model time may be simulated in a given interval of wall-clock time. If the couplings are all scheduled (good) or periodic (better) or fall within a single coupling cycle (ideal), this task will be easier than for threshold-driven coupling. If performance data for the coupling transformations are available, this data will improve any throughput estimates. Repeated timings for the constituents will allow estimation of mean execution time and its variance; these statistics may be used later in Monte Carlo simulations to estimate coupled model throughput. Timing statistics may also be used to estimate system-wide and per-constituent coupling overhead (namely, eqns. (11) and (9), respectively, in Larson [2009]).

3.3 COPING WITH CONCURRENCY

The analysis techniques developed thus far are applicable to a uniprocessor (von Neumann) architecture. In systems where parallel computing will be employed, extensions are required to support some types of concurrency. For purely shared-memory parallelism (SMP), the system still has a single address space, and all of the exercises can be used as-is. For distributed-memory systems, complications arise and must be handled. The first issue is that concurrent execution of constituents is possible, engendering problems of where on the multiprocessor system constituents execute (process composition), where coupling transformations occur, interconstituent parallel data transfer or redistribution, and system load balance. The second complication is domain decomposition of coupling data and the concomitant issue of parallelisation of coupling transformations.

In a serial composition, the global processor pool is kept intact, and $\{\mathcal{C}_1, \ldots, \mathcal{C}_N\}$ share it, running in succession, interleaved between necessary coupling transformations. In a parallel composition, the global processor pool is divided into N cohorts, each constituent runs on its own cohort, and coupling operations between $\mathcal{C}_i$ and $\mathcal{C}_j$ are performed on the union of their cohorts or subset thereof. Serial compositions are easier to understand, and performance analysis is straightforward. A disadvantage to serial compositions, however, is that poor parallel scalability of one or more constituents can quickly limit parallel coupled system throughput. Parallel composition allows cohorts to be sized according constituent scalability but complicates load balance, and interconstituent data dependencies can cause constituents to stall awaiting data from other constituents.

Distributed-memory parallelism in any given constituent means its domain Γ_i and input, output, and state vectors are decomposed across a set of K_i processors—called a *cohort*—using the parallel decomposition operator $\mathbf{P_i}(\cdot)$; e.g., $\mathbf{P_i}(\Gamma_i) = \{\gamma_i^{(0)}, \ldots, \gamma_i^{K_i-1}\}$, where $\gamma_i^{(m)}$ is the portion of Γ_i residing on the mth processor in the cohort. Similarly, $(\hat{\mathbf{u}}_i^{(m)}, \hat{\mathbf{v}}_i^{(m)}, \hat{\mathbf{w}}_i^{(m)})$ are the portions of the state, input, and output vectors residing on processor m. Thus, any coupling transformation is parallel; that is, $\mathscr{T}_{ij} : (\hat{\mathbf{w}}_j^{(0)}, \ldots, \hat{\mathbf{w}}_j^{(K_j-1)}) \rightarrow (\hat{\mathbf{v}}_i^{(0)}, \ldots, \hat{\mathbf{v}}_i^{(K_i-1)})$. A parallel coupling transformation is $\mathscr{T}_{ij} = \mathscr{F}_{ij} \circ \mathscr{G}_{ij} \circ \mathscr{H}_{ij}$, where $\mathscr{H}_{ij}$ represents parallel data transport. Implementing these parallel transformations is challenging; a good example of a complete implementation is given by Jacob et al. [2005], and a detailed discussion of implementations of $\mathscr{H}_{ij}$ is given by Bertrand et al. [2006] and references therein. The same logic is applicable to implicit and multipartite coupling transformations.

How does one apply the analysis outlined in this paper to a parallel system? Admittedly, with considerable difficulty. All of the exercises must now include some recognition of how a constituent is parallelised, and in many cases two constituents that are coupled may be coupled only via subsets of their respective cohorts. Thus, coupling data volumes and overheads must be assessed on a processor-by-processor basis. For systems using parallel composition to execute constituents, the (wall-clock) frequency of coupling operations must be timed to mesh well with all coupling partners so as to minimise the aforementioned data-dependency-driven stalls. Parallel performance data must be collected for each constituent's likely parameter settings and processor counts, and these data folded into any throughput estimates, and used to support efficient mapping of constituents to cohorts. Extension of this analysis technique to parallel systems will require tools to automate the exercises described in this paper—an area for future work.

4 CONCLUSIONS

I have presented a set of exercises and questions developers should address early in the requirements-gathering and design process. The exercises are derived from experience working on large multiphysics simulation projects and from a theoretical framework for describing coupled systems. Modified or extended to suit a particular coupling problem, they provide analysis tools for understanding a system's coupling properties and, pursued to their logical conclusion, methods for estimating the system's likely performance. I hope they will accelerate the development process for coupled systems and encourage more researchers to undertake model development of complex systems "from the ground up."

Future areas of work include application of this analysis to a wide variety of existing multiphysics and multiscale systems, which will provide opportunities for enhancing the technique; expansion of the method to better support analysis of parallel systems; and investigation of techniques for automating the analysis and performance estimates.

ACKNOWLEDGMENTS

This work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy (DOE), under Contract DE-AC02-06CH11357. I thank the Department of Theoretical Physics of the Research School of Physical Sciences and Engineering for hosting me as a visiting fellow.

REFERENCES

Bertrand, F., R. Bramley, D. E. Bernholdt, J. A. Kohl, A. Sussman, J. W. Larson, and K. B. Damevski. Data redistribution and remote method invocation for coupled components. *Journal of Parallel and Distributed Computing*, 66(7):931–946, 2006.

Chivers, I. D. and J. Sleightholme. Compiler support for the Fortran 2003 standard. *ACM Fortran Forum*, 26(2):25–27, 2007.

COMSOL, Incorporated. COMSOL Web site. `http://www.comsol.com`, 2009.

Craig, A. P., B. Kaufmann, R. Jacob, T. Bettge, J. Larson, E. Ong, C. Ding, and H. He. cpl6: The new extensible high-performance parallel coupler for the community climate system model. *Int. J. High Perf. Comp. App.*, 19(3):309–327, 2005.

Dahlgren, T., T. Epperly, and G. Kumfert. *Babel User's Guide*. CASC, Lawrence Livermore National Laboratory, version 0.9.0 edition, January 2004.

Diestel, R. *Graph Theory*. Springer, New York, third edition, 2006.

ISO/IEC Joint Technical Committee 1, Subcommittee 22, Working Group 5. Information Technology–Programming Languages–Fortran–Part 1: Base Language. Standard Definition ISO/IEC 1539-1:2004, International Standardization Organization, Geneva, Switzerland, 2004.

Jacob, R., J. Larson, and E. Ong. $M \times N$ communication and parallel interpolation in ccsm3 using the model coupling tookit. *International Journal of High Performance Computing Applications*, 19(3):293–308, 2005.

Joppich, W., M. Kurschner, and the MpCCI Team. MpCCI - a tool for the simulation of coupled applications. *Concurrency and Computation: Practice and Experience*, 18(2):183–192, 2006.

Larson, J. W. Ten organising principles for coupling in multiphysics and multiscale models. *ANZIAM Journal*, 48: C1090–C1111, 2009.

Larson, J., R. Jacob, and E. Ong. The model coupling toolkit: A new fortran90 toolkit for building multi-physics parallel coupled models. *Int. J. High Perf. Comp. App.*, 19(3):277–292, 2005.

Lethbridge, P. Multiphysics analysis. *The Industrial Physicist*, 10(6):26–29, 2004.

Michopoulos, J. G., C. Farhat, and J. Fish. Modeling and simulation of multiphysics systems. *Journal of Computing and Information Science in Engineering*, 5(3):198, 2005.

Rasmussen, C. E., M. J. Sottile, S. S. Shende, and A. D. Malony. Bridging the language gap in scientific computing: The CHASM approach. *Concurrency and Computation: Practice and Experience*, 18(2):151–162, 2006.