2011 Seventh IEEE International Conference on e-Science Workshops

A principled approach to distributed multiscale computing, from formalization to execution

Joris Borgdorff, Eric Lorenz, and Alfons G. Hoekstra
Section Computational Science
University of Amsterdam
Amsterdam, the Netherlands
e-mail: {j.borgdorff, e.lorenz, a.g.hoekstra}@uva.nl

Jean-Luc Falcone and Bastien Chopard
CUI
University of Geneva
Carouge, Switzerland
e-mail: {jean-luc.falcone, bastien.chopard}@unige.ch

***Abstract*—In several disciplines, a multiscale approach is being used to model complex natural processes yet a principled background to multiscale modeling is not clear. Additionally, some multiscale models requiring distributed resources to be computed in an acceptable timeframe, while no standard framework for distributed multiscale computing is place. In this paper a principled approach to distributed multiscale computing is taken, formalizing multiscale modeling based on natural processes. Based on these foundations, the Multiscale Modeling Language (MML) is extended as a clear, general, formal, and high-level means to specify scales and interactions in, and as a guide to a uniform approach to crystalize, communicate, develop and execute a multiscale model. With an MML specification, a multiscale model can be analyzed for scheduling or deadlock detection using a task graph. The potential of this method is shown by applying it to two selected applications in nano materials and biophysics.**

***Keywords*-distributed multiscale computing; multiscale modeling; MML; task graph; coupling template; coupling topology; submodel execution loop;**

I. Introduction

Understanding the complexity of nature by using computer models drives computational science [1]. As the knowledge and data on a multitude of scales is accumulating independently, the need for bridging the scales became apparent, introducing multiscale modeling [2]. Currently, early multiscale models show their high computational demands and at the same time their potential for modularity, leading to the idea of distributed multiscale computing.

An early incarnation of this idea was the COAST project, applying it with Complex Automata (CxA): coupled single scale cellular automata models [3]–[5]. At the same time they also partially formalized multiscale modeling, be it for cellular automata only. In the MAPPER project, backed by five scientific communities, the CxA concept is being generalized to different types of models besides cellular automata [6]. Moreover, it emphasizes the need for distributed multiscale computing, to compute multiscale models in an acceptable timeframe.

Meanwhile, it has become apparent that scientists need a common language to communicate their multiscale model with, and this has been undertaken separately in the form of an ontology by Yang and Marquardt [7] and of a Multiscale Modeling Language (MML) [8].

In this paper, we will detail the principles of multiscale computing, extend MML to fit a general multiscale theory and show how distributed multiscale computing can be achieved. Finally, distributed multiscale computing is applied to two selected applications in biophysics and nanotechnology.

II. Standardizing multiscale interactions

Multiscale modeling is used throughout the natural sciences for understanding, with different processes observed on different scales. To be precise, if by a process is meant something natural or really existing, a case can be made that the process itself does not have scale. Rather, the observation of a process, called a phenomenon, will have a certain resolution or granularity that may be called a scale. A scale may refer to any dimension, be it time, space, or an abstract dimension such as the size of a set. It should have an empirical meaning, in that the observation or data may have inherent limits to precision or granularity that can be reflected in a scale. We do assume, however, having started at natural sciences, that a phenomenon has at least a temporal and spatial scale. Likewise, a domain may have a scale and is represented as part of a phenomenon by excluding its temporal domain. A domain may have subdomains, which observe only part of the object of the phenomenon, or correspond to other domains that observe the same object in another way.

A. Scale

To serve human intuition, the scale may be expressed by a single number, a characteristic scale [4], [9], describing its granularity. This number can sometimes be used directly in equations modeling a phenomenon, like in reaction-diffusion equations [10].

For precise relations between scales, whether they are comparable or different, not only the granularity but also the total size of a phenomenon should be taken into account, and

978-0-7695-4598-1/11 $26.00 © 2011 IEEE
DOI 10.1109/eScienceW.2011.9

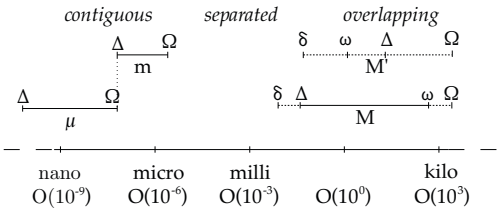
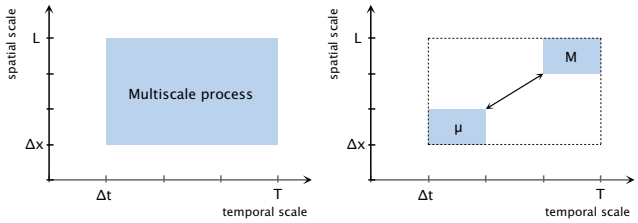


Figure 1. Contiguous scales, scale separation, and scale overlap on a single dimension according to the scale specifications of given submodels μ, m, M, and M', plotted on a arbitrary logarithmic SI scale.



(a) Multiscale phenomenon (b) Single scale phenomena

Figure 2. An example SSM of a classical macro-micro model, showing on the axes spatial and temporal scale, with granularity Δx, Δt and total size L and T respectively. In 2a a multiscale model without separating scales, in 2b single scale submodels macro M and micro μ.

their variability. Two scales can be considered truly different or separated if the total size of one scale is smaller than the granularity of another.

These concepts of a scale can be captured by a scale specification $S(\delta, \Delta, \omega, \Omega)$ over a dimension, with δ and Δ as minimum and maximum granularity, respectively, and ω and Ω as minimum and maximum total size. Since scale in principle concerns observations of the natural world, none of these numbers may be infinite. For simple phenomena or ones with fixed-step observations, a regular scale with no variability may be used, as in $S(\Delta, \Omega)$. For phenomena of which particle locations are measured or where changes only occur at certain events, a full scale specification should be used. Throughout this article, when a scale is mentioned, its scale specification is also implicitly referred to.

Relations between scale specifications are much clearer than those between characteristic scales, and they are depicted in Figure 1. Formal separation between two scales $S_A(\delta_A, \Delta_A, \omega_A, \Omega_A)$ and $S_B(\delta_B, \Delta_B, \omega_B, \Omega_B)$ is present if and only if $\delta_A > \Omega_B$ or the other way around. Scale overlap on the other hand has $\omega_A \geq \Delta_B$ and $\omega_B \geq \Delta_A$. For contiguous scales, $\omega_B \leq \delta_A \leq \Omega_B \leq \Delta_A$ or with A and B switched in the comparison.

These relations can be motivated from natural observations, scale separation occurring if one phenomenon is perceivable on a certain scale but an interacting phenomenon is not. Scale overlap is then just perceivable on the same scale and scale contiguity is viewing elements of a phenomenon on one scale, which are subject to other phenomena themselves.

Computationally it also makes sense to distinguish between the three relationships, scale separation being an opportunity to reduce computational costs by reusing certain results on a coarser scale [10]. With scale overlap, more interaction between submodels might be necessary to synchronize their entire domain. Scale contiguity makes sense in an object-oriented language or a multigrid approach, where a submodel of a finer phenomenon is calculated for exactly one element of a submodel of a coarser phenomenon.

B. Multiscale modeling

Multiscale modeling consists of separating or splitting phenomena by their scale, type, or representation and then grasp the precise interaction between the phenomena. As such, a multiscale model can be seen as a collection of interacting single scale models or submodels [4], [9]. Methods for splitting phenomena form a large part of what multiscale modeling is about [9], [11], here we give a short overview.

First, scale splitting is a technique that is defining for multiscale modeling, and consists of decomposing a phenomenon into a fine, fast, or small part and a coarse, slow or large part. By splitting the scales, the fine part might be computed often, while the coarse part needs less iterations to be computed. In certain cases an exact computational reduction caused by this approach can be calculated [10].

A scale separation map (SSM) like in Figure 2 may help in assessing the scales that are currently of interest [4], [12]. It shows the phenomena involved and their interactions, on a logarithmic plot of time and space.

Second, a phenomenon consisting of different types of phenomena could be separated. For example, a liquid that flows along an elastic structure, could be modeled with a fluid dynamics and an elastic mechanics submodel. The type of phenomenon might also change over time, possibly needing to be modeled by one submodel at the start but needing another at the end.

Finally, different representations of a phenomenon could be a reason for dividing it. For example, the same natural object could be represented by a continuum domain and a grid domain, for which different phenomena are appropriate.

A multiscale model consists of interacting submodels that model these separated phenomena. By having separate submodels, multiscale models are inherently modular and suitable for applying component-based software engineering [13], [14]. Multiscale modeling, in this terminology, distinguishes itself by performing scale splitting, otherwise any other component-based modeling technique would suffice.

Besides separating different phenomena, computational concerns may be a reason for domain decomposition, com-

puting different parts of the domain of one phenomenon on different machines. Later in this article, we will use a single submodel with multiple instances for this.

C. Submodel execution loop

A submodel as defined models a single phenomenon. A standard runtime flow of such a submodel has been identified by Hoekstra et al. [4]: the submodel execution loop (SEL). We generalize this loop here for submodels that are not cellular automata. The SEL is a formal algorithm with five operators, the content of which a modeler needs to formulate. Its flow rests on the assumption that a state change in a submodel only occurs when an event would be observed in the underlying phenomenon. For an event-based submodel this implies that each submodel has a well-ordered time series $\vartheta = (e_0, \ldots, e_n)$ that may be determined during runtime, with events at time $t(e_0) < t(e_1) < \cdots < t(e_n)$, the minimal difference being governed by the temporal scale of the submodel.

The submodel is initialized by operator $\mathbf{f}_{\text{init}}$ at event e_0, after which it observes its first state with operator $\mathbf{O_i}$. For each subsequent event, it calculates with operator $\mathbf{S}$ what state change is necessary to model that event, and updates any boundary conditions with operator $\mathbf{B}$. It then makes an intermediate observation of its state with operator $\mathbf{O_i}$. Additionally, each step may change the timing and content of future events, by returning a future time series ϑ_i, a subset of ϑ only including events after event e_i. When there are no more events, the final state is observed with operator $\mathbf{O_f}$. Formally, this SEL represented by the following pseudocode:

Input: starting time t_0
$i \leftarrow 0$
$f, \vartheta \leftarrow \mathbf{f}_{\text{init}}(t_0)$
while $|\vartheta_i| > 0$ **do**
  $\mathbf{O_i}(f,\, t(e_i),\, t(e_{i+1}))$
  $i \leftarrow i + 1$
  $f, \vartheta_i \leftarrow \mathbf{S}(f, e_i, \vartheta_i)$
  $f, \vartheta_i \leftarrow \mathbf{B}(f, e_i, \vartheta_i)$
end while
$\mathbf{O_f}(f,\, t(e_i))$

Variations on this general SEL are possible, as long as the operators are executed in the same order. A time-driven submodel could use the following pseudocode, with a fixed time step rather than events.

Input: starting time t_0 and temporal scale $S_\tau(\Delta t, T)$
$t \leftarrow t_0$
$f \leftarrow \mathbf{f}_{\text{init}}(t)$
while $t - t_0 < T$ **do**
  $\mathbf{O_i}(f, t)$
  $t \leftarrow t + \Delta t$
  $f \leftarrow \mathbf{S}(f, t)$
  $f \leftarrow \mathbf{B}(f, t)$
end while
$\mathbf{O_f}(f, t)$

Table I
COUPLING TEMPLATES BETWEEN SUBMODELS A AND B, LISTING WHAT THEIR TEMPORAL SCALE RELATION IS AND THE MOST LIKELY SCENARIO IN WHICH THE TEMPLATE IS USED. WHERE OPERATOR $\mathbf{S}$ IS LISTED, $\mathbf{B}$ CAN BE SUBSTITUTED FOR MD COUPLINGS.

Coupling template		Temp. scale	Scenario
interact	$\mathbf{O_i}^A \rightarrow \mathbf{S}^B$	overlap	response by interact
call	$\mathbf{O_i}^A \rightarrow \mathbf{f}^B_{\text{init}}$	contig. or sep.	response by release
release	$\mathbf{O_f}^B \rightarrow \mathbf{S}^A$	contig. or sep.	responds to call
relay	$\mathbf{O_f}^A \rightarrow \mathbf{f}^B_{\text{init}}$	any	loosely coupled or stateful

D. Coupling templates

The interaction between submodels can be formulated in terms of their SEL operators, again restricting allowed behavior. Furthermore, the computational SEL operators $\mathbf{f}_{\text{init}}$, $\mathbf{S}$, and $\mathbf{B}$ are only allowed to receive data and the observational operators $\mathbf{O_i}$ and $\mathbf{O_f}$ may only send data.

A distinction in interactions with operators $\mathbf{S}$ and $\mathbf{B}$ is made not by their place in the SEL, which is similar, but rather by whether a coupling is multidomain (mD) or single domain (sD). An sD coupling is used for interactions where one of the submodels is on a subdomain or corresponding domain to another submodel; an mD coupling is used for other couplings. For the same place in the SEL, $\mathbf{S}$ is used for sD couplings and $\mathbf{B}$ for mD couplings.

With these restrictions, and those of scale separation, only a few options for coupling remain, listed in Table I. First, the *interact* coupling template allows two submodels on an overlapping temporal scale to communicate. By the time they finish one iteration at time t_A they may send results only to an operator of a submodel at time $t_B > t_A$. A submodel A that is on a coarser scale than submodel B, with scale separation or contiguity, can *call* B and once B is finished, it can *release* its resources to A. Finally, one submodel may *relay* a message to another that starts at a later time, or save a state from one iteration of the submodel to the next.

E. Coupling topology

Since a multiscale model consists of submodels and their interactions, it can be instantiated with submodels instances and coupling templates. We call the graph so formed a coupling topology, and assign to its edges the number of times a coupling template would be used. Based on certain properties, the coupling topology is more dynamic, making it harder to calculate.

Three properties of the coupling topology are deemed important by us:

1) whether submodels are loosely or tightly coupled;
2) whether they have single, multiple, or a dynamic number of instances; and
3) whether they have a fixed or dynamic number of synchronization points.

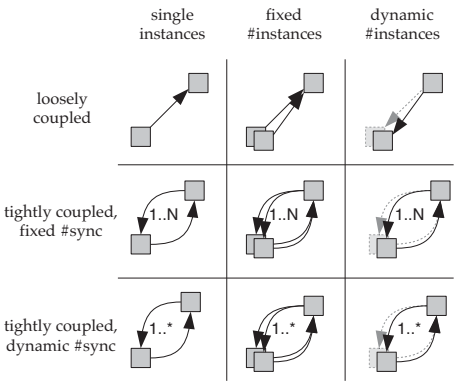


Figure 3. Graphical overview of different coupling topology properties, like being loosely or tightly coupled, having a fixed or dynamic number of submodel instances, and having a fixed or dynamic number of synchronization points. Shown here are only examples with two submodels, but this can be generalized to any number of submodels.

These properties are illustrated in Figure 3 and explained below.

The first property, loosely or tightly coupled, is measured by whether the coupling topology is acyclic (loose) or cyclic (tight). A cycle may form between two submodel instances or more, but in any case tightly coupled models are harder to compute since submodels need to stay active to receive another message. With loosely coupled models, submodels can compute some data and then halt again, making the model easy to compute.

Second, the number of submodel instances, determines how a submodel is accessed. With single instances, one submodel code is run for one instance, with multiple instances the same code will be used multiple times throughout the model. If there are a dynamic number of instances then the framework that computes the model has to be informed what number will run exactly.

Finally, the number of synchronization points determines how often data needs to be transferred from one submodel to another, but it may also involve submodels being started a variable number of times. Again, a model must somehow interact with the computational framework to enable this kind of behavior.

All elements of a multiscale model are now standardized in terms of interactions, easing subsequent specification, analysis and execution.

III. Multiscale model architecture

To compute a multiscale model, the computational architecture of that model should be known; this can be described using the Multiscale Modeling Language (MML) [8]. This language specifies submodel scales, interactions, and their coupling topology, along with certain computational aspects on running a model. An MML description serves multiple purposes, such as communicating the specifics of a multiscale model, aiding development by generating code, validating the multiscale properties of a multiscale model, or forming a description for a runtime system.

MML has three representations:

- a human language using the terminology of this paper;
- a graphical representation gMML; and
- a human-writable, machine-readable XML-based representation xMML.

It is a high-level language, describing the architecture in an implementation-agnostic way, but also a specific language, allowing all computational elements to be specified. Its advantage over any other components based specification language [14]–[16] lies therein that the internal flow of submodels is well-defined, as well as their allowed interaction based on scale.

A. Computational elements

Computational elements of a multiscale model architecture are based on the composition of multiscale models. A submodel instance has ports defined at SEL operators where it has a coupling template, and a coupling is instantiated by a conduit that transfers data in-order in the form of messages from one port to another.

Along a conduit, messages can be altered, multiplied, or stopped by conduit filters. However, a conduit filter can not send a message without receiving one. It can be used for data transformation, translating one domain to another, or interpolate or aggregate messages for submodels with a different but overlapping scale. Filters may have a state, which is necessary for aggregating or interpolating data.

A conduit filter applies to only one conduit, so to manipulate data from multiple sources another construct is needed: a mapper. We define two types of mappers: the fan-in and fan-out mapper. The former has multiple input ports, possibly a dynamic number, and one output port. The mapper blocks until it receives a message on each input port, after which it may send a single message. The latter has one input port and multiple output ports, and sends one message on each output port for every message received. Combined, mappers can facilitate domain decomposition, where multiple submodel instances compute a different part of a domain, and the output is collected by a fan-in mapper. Mappers do not have a state other than a buffer for messages, simplifying them internally.

A model is started by specifying a subset of submodels or mappers that should be initialized, and ended by the SEL of the submodels.

B. Graphical representation

The graphical representation of MML, gMML, uses simple graphical elements inspired by UML Diagrams to communicate the architecture of a model, as seen in Figure 4. It implies the definitions of the computational elements above and has different icons for each of them. Submodels are

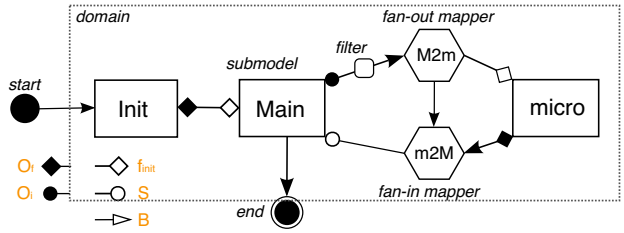


Figure 4. The gMML of the macro-micro model used in Figure 2. The different edge heads and tails show the operators that are used. The model starts by computing an initial value, which is then given to the macro submodel. The M2m and m2M fan-out and fan-in mappers convert a grid from Macro to values for micro models and the other way around; the m2M mapper then needs information on how M2m did the conversion. A conduit filter converts, for instance, the data type from Macro before it is processed by M2m.

depicted as rectangles, conduits as lines between submodels, conduit filters as rounded squares on a conduit, and mappers as hexagons. Submodel instances that are coupled by a sD coupling can be grouped inside a dashed rectangle. Submodels that are initialized first have an edge from a filled circle, submodels that end the model have an edge to a filled circle with a circle around it.

In gMML, the different SEL operators that are used in coupling templates are shown with corresponding head and tail icons for the edges. For the operators outside the loop, $\mathbf{f}_{\text{init}}$ and $\mathbf{O_f}$, a diamond is used, for two operators inside the loop, $\mathbf{O_i}$ and $\mathbf{S}$, a circle, and for $\mathbf{B}$ an arrow. Also, sending operators $\mathbf{O_i}$ and $\mathbf{O_f}$ are filled black, while the others are filled white. If no coupling template was used for the conduit, a simple arrowhead and no tail is used.

Although gMML serves as a concise communicational tool or as a template for a specification, it does not completely describe a multiscale model. For instance, the scales of the submodels are not represented directly, nor are the implementation details. For this full description xMML should be used.

C. XML format

For machines the XML format xMML is more suitable to use, with its fixed elements and possibility for validation with a Document Type Definition (DTD) or W3C XML Schema. At the same time, the format is succinct and human-writable, although a graphical user interface may eventually write xMML.

Below an example xMML document is listed, following the example of a macro-micro model given in Figures 2 and 4. It starts by naming the model and defining datatypes and computational elements that can be used. The scales of submodels are listed and the ports of submodels and mappers are specified. In the topology, those elements can be instantiated and used with couplings. Repeating elements are replaced by an ellipsis.

```
<model id="MacroMicro" xmml_version="0.3.2">
  <description>A macro-micro model with a macro
      grid and micro cells.</description>

  <definitions>
    <datatype id="lattice2DDouble" size_estimate="
        x*y*sizeof(double)"/>
    ...
    <filter id="microArray" type="converter"
        datatype_from="lattice2DDouble"
        datatype_to="lattice2DFloat"/>

    <submodel id="Macro" init="yes">
      <timescale delta="1 s" total="1 min"/>
      <spacescale total="1 dm">
        <delta min="0.7 mm" max="1.3 mm"/>
      </spacescale>

      <ports>
        <out id="grid" operator="Oi" datatype="
            lattice2DDouble"/>
        <in id="gridDiff" operator="S" datatype="
            lattice2DFloat"/>
      </ports>
    </submodel>
    <submodel id="micro" name="Micro 1D">
      ... </submodel>

    <mapper id="gridDivide" type="fan-out">
      <ports>
        <in id="grid" datatype="lattice2DFloat"/>
        <out id="value" datatype="float"/>
      </ports>
    </mapper>
    <mapper id="gridCombine" type="fan-in"> ...
    </mapper>
  </definitions>
  <topology>
    <instance id="M" submodel="Macro"/>
    <instance id="m" submodel="micro"
        multiplicity="10"/>
    <instance id="M2m" mapper="gridDivide"/>
    <instance id="m2M" mapper="gridCombine"/>

    <coupling from="M.grid" to="M2m.grid">
      <apply filter="microArray"/>
    </coupling>
    <coupling from="M2m.value" to="m.value"/>
    <coupling from="m.diff" to="m2M.value"/>
    <coupling from="m2M.grid" to="M.gridDiff"/>
  </topology>
</model>
```

Practical details of the computational elements can also be added, both implementation details and runtime statistics. The size estimate accompanying the datatype depends on the scale of the sending submodel and it can be an indication that some couplings are more data-heavy than others.

IV. Doing distributed multiscale computing

A formal background for multiscale modeling in place, it is time to do distributed multiscale computing. A few questions are inevitable for making this step, of which we will address two: how can a multiscale model be scheduled across distributed resources and will it execute without deadlocks. An intermediate solution is shown by constructing a distributed acyclic task graph using an MML specification;

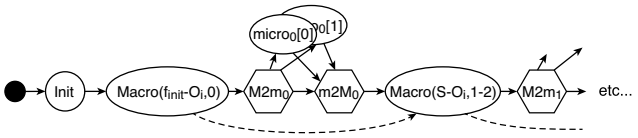


Figure 5. The start of the taskgraph of the example given in Figure 4 after collapsing redundant nodes. The Init model is computed without dependencies and does not list SEL operators separately, while the macro model computes multiple operators per task but not more than one iteration at a time. The dashed edges represent a transition from one SEL operator to the next.

the task graph can then be scheduled according to known algorithms [17]–[19] and deadlocks can be detected.

The usage of a software package that can handle distributed submodels is outlined, the multiscale coupling library and environment (MUSCLE) [20], [21].

A. Constructing a task graph from MML

The directed acyclic task graph constructed in this paper is a serialization or unfolding of the execution of a multiscale model. By definition, it may not contain the cycles that are present in coupling topology if it concerns a tightly coupled multiscale model. Therefore, tasks in this graph are necessarily more granular than submodel instances. The SEL operators are a viable candidate, as they are indivisible and do all communication.

The nodes of the task graph will thus contain an identifiable SEL operator. A full label of such a node looks like $inst_j[k](o, i)$ for submodel instance $inst$, SEL operator o, iteration i, index k if a multiplicity was used, and an initialization number j if the submodel instance is restarted multiple times. Any of the identifiers after $inst$ may be omitted as long as the label uniquely identifies a task. For mappers and filters, only an initialization number is used, as they don't have iterations and operators.

An edge between two nodes p and q is added if:

- p should send a message to q;
- q is the operator after p in the SEL of one submodel; or
- q is the initialization of the same stateful computational element as p, and p is the elements previous final observation.

These properties can be deducted in advance in a linear way, unless the associated coupling topology has a dynamic number of synchronization points. Not only can the number of nodes not be predicted then, but also the flow might be significantly different. A dynamic task graph algorithm may be employed in this case, in combination with dynamic deadlock detection algorithms [22]. If there are a dynamic number of submodel instances, the flow will still remain largely intact, by representing all of those instances by a single node.

Even if the above method works linearly in the number of edges of the task graph, the number of edges in the task graph may be exponential to the number of submodel instances. Reduction techniques such as collapsing equivalent nodes and detecting repetition can be used to handle such large task graph.

Once a task graph is constructed, a deadlock is indicated by nodes that need additional input and cycles. With the information about the implementation and runtime statistics in the MML, edges and nodes can be given a weight that a scheduling system can use.

B. Implementing and executing with MUSCLE

A framework that can execute multiscale models on distributed resources is MUSCLE. The core of MUSCLE is implemented in Java, but submodels can also be implemented in C++ or by extension in Fortran or C. Its communication layer is currently managed by the Java Agent Development Environment (JADE) [23].

MUSCLE has two base elements: the kernel, which contains the submodel code, and the conduit. It does not have an explicit SEL, so communication must be regulated by the model developer. Conduit filters are explicitly supported, so a developer may implement them or reuse a pre-defined one. A mapper can be implemented as a kernel.

Within submodels parallel code can be used, for instance by using Java threads or OpenMP. On the other hand, because of the Java core of MUSCLE, the use of MPI is not supported directly and currently requires separate executables for code using it.

A MUSCLE model can be executed by starting an instance of MUSCLE on each machine that should be used and starting submodel instances on those MUSCLE instances. If a single machine setup is needed, just one MUSCLE instance needs to be started, otherwise they can communicate using TCP/IP. Machines that have no TCP/IP connections to others that run MUSCLE can only be accessed if there is a head node with a TCP/IP connection that controls the executable on that machine.

If the targeted machines are not directly available but are part of a grid or e- Infrastructure instead, it can be wise to make use of grid middleware, for example the tools from the QosCosGrid project [24]. Such a system could also make use of the task graph for scheduling submodels on several resources, as MUSCLE also does not do any scheduling.

V. Example implementations

The approach shown in this paper is being applied to multiscale models from different disciplines in the MAPPER project [6]. The aim of MAPPER is to automate distributed multiscale computing. Two applications are selected here: a three-dimensional in-stent restenosis model (ISR3D) [25], [26] clay-polymer nanocomposite material formation (nanomaterials) [27].

The former, ISR3D, models the growth of a restenosis in a coronary artery, after a stent has been placed with a

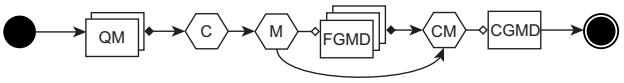


Figure 6. The gMML of the Nano materials application. It starts with a few QM submodels, after which their output is collected by the C fan-in mapper. The output is then preprocessed by fan-out mapper M to send it to a number of FGMD submodels. Their output, and the mapping used by M, is collected by the CM fan-in mapper to be used by the final CGMD submodel.

balloon angioplasty to remedy a growing stenosis [25]. The two-dimensional version has shown to be a valid [28] and well-described [26] computational model to study this phenomenon. Preliminary results for ISR3D have been obtained, but distributed resources are required to get more results.

Submodels of ISR3D compute the stent placement (IC), smooth muscle cell proliferation (SMC) causing the restenosis, blood flow (BF) in the lumen and drug diffusion (DD) that a drug-eluting stent would produce. From a multiscale perspective, ISR3D has temporal scale separation in all its submodels except IC, with SMC acting in days, DD in hours, and BF in seconds. They do not have spatial scale separation, since each acts on a similar part of the blood vessel.

Its coupling topology is simple, having no dynamic properties, just a tightly coupled cycle where SMC needs BF and DD to calculate their values in each iteration. In its MML specification, like in Figure 4, a fan-out mapper is needed to distribute the domain computed by SMC to BF and DD and a fan-in mapper to collect their results to send then back to SMC. This has been implemented in MUSCLE, with a Java and Fortran part, and has had its first distributed execution, using MUSCLE like in the two-dimensional version [26].

The other application, nanomaterials, models the formation of a clay-polymer nanocomposite material because of its enhanced thermochemical and mechanical properties [27]. Placement of individual atoms between clay sheets is evaluated using quantum mechanics (QM), placement of individual molecules is decided by fine-grained molecular dynamics (FGMD), and groups of molecules are placed using coarse-grained molecular dynamics (CGMD). The simulation starts by evaluating multiple instances of QM, the computed values are then used in a number of FGMD instances, and finally the molecules are placed in CGMD.

The three submodels have a typical macro-micro coupling, from QM to FGMD to CGMD, with spatial and temporal scale separation, and a scale ranging from picometer to micrometer and from picosecond to millisecond. Smaller submodels are in a subdomain of larger submodels, so all couplings are sD.

The gMML of Nano materials is shown in Figure 6, and also features mappers to manage information coming from the submodels with decomposition. The coupling topology is loosely coupled, as can be seen from the gMML, so it also has a fixed number of synchronization points, also the number of submodel instances is fixed before the model starts.

The model is implemented using multiple libraries: the LAMMPS [29] and CASTEP [30], with Perl scripts acting as mappers. Since the model is loosely coupled the workflow is straightforward to implement, by running one submodel after the other; using MUSCLE is not necessary.

VI. Conclusion

As multiscale modeling is made use of throughout exact sciences, settling on a set of principles of multiscale modeling will help it maturing [2], [3]. In this paper, a few of the principles surrounding scale have been made explicit, and a standardized approach for modeling multiscale problems is given. Moreover, by proposing this standardization, multiscale models adhering to it can write their specifications with MML, and general software can be adapted to execute them. Achieving this also makes distributed multiscale computing possible and usable.

Compared to previous work on Complex Automata [3], [31], [32], this contribution describes a more general approach to multiscale modeling than only coupled cellular automata. It expands on MML by fixing the behavior of mappers and introducing submodel instances. Furthermore, this contribution introduces the coupling topology as a way to assess how dynamic the execution of a multiscale model will be. It also introduces the task graph as a way to model the flow of a multiscale model.

We hope that this approach will gain following by applying it to more multiscale applications, making them suitable for a distributed computing. Currently, we are exploring the possibilities of making MUSCLE compatible with supercomputing systems and easing its combination with C++ and Fortran code. Future theoretical developments include finding solutions for fully taking dynamic coupling topologies into account. Finally, the use of this methodology for abstract scales, network models, and other complex systems should be further explored.

Acknowledgments

The authors would like to thank James Suter, Derek Groen, and Peter Coveney of University College London for providing information about the Nano materials application. The Mapper project receives funding from the EC's Seventh Framework Programme (FP7/2007-2013) under grant agreement n° RI-261507.

References

[1] P. M. A. Sloot and A. G. Hoekstra, "Multi-scale modelling in computational biomedicine," *Briefings in bioinformatics*, vol. 11, no. 1, pp. 142–152, Jan. 2010.

[2] W. E, B. Engquist, X. Li, W. Ren, and E. Vanden-Eijnden, "The heterogeneous multiscale method: A review," *Communications in Computational Physics*, vol. 2, no. 3, pp. 367–450, Jun. 2007.

[3] A. G. Hoekstra, A. Caiazzo, E. Lorenz, and J.-L. Falcone, "Complex automata: multi-scale modeling with coupled cellular automata," in *Simulating Complex Systems by Cellular Automata*. Springer-Verlag Berlin / Heidelberg, 2010, pp. 29–57.

[4] A. G. Hoekstra, E. Lorenz, J.-L. Falcone, and B. Chopard, "Toward a Complex Automata Formalism for MultiScale Modeling," *International Journal for Multiscale Computational Engineering*, vol. 5, no. 6, pp. 491–502, 2007.

[5] B. Chopard, J.-L. Falcone, A. G. Hoekstra, and J. Borgdorff, "A Framework for Multiscale and Multiscience Modeling and Numerical Simulations," in *LNCS 6714*. Springer-Verlag Berlin / Heidelberg, 2011, pp. 2–8.

[6] The MAPPER project, http://www.mapper-project.eu/, 2010.

[7] A. Yang and W. Marquardt, "An ontological conceptualization of multiscale models," *Computers & Chemical Engineering*, no. 33, pp. 822–837, 2009.

[8] J.-L. Falcone, B. Chopard, and A. G. Hoekstra, "MML: towards a Multiscale Modeling Language," *Procedia Computer Science*, vol. 1, no. 1, pp. 819–826, 2010.

[9] G. D. Ingram, I. T. Cameron, and K. M. Hangos, "Classification and analysis of integrating frameworks in multiscale modelling," *Chemical engineering science*, vol. 59, pp. 2171–2187, 2004.

[10] A. Caiazzo, J.-L. Falcone, B. Chopard, and A. G. Hoekstra, "Asymptotic analysis of Complex Automata models for reaction–diffusion systems," *Applied Numerical Mathematics*, vol. 59, no. 8, pp. 2023–2034, Aug. 2009.

[11] W. E, B. Engquist, and Z. Huang, "Heterogeneous multiscale method: A general methodology for multiscale modeling," *Physical Review B*, vol. 67, no. 9, p. 092101, Mar. 2003.

[12] G. D. Ingram and I. T. Cameron, "Challenges in Multiscale Modelling and its Application to Granulation Systems," *Developments in Chemical Engineering and Mineral Processing*, vol. 12, no. 3-4, pp. 293–308, 2004.

[13] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[14] B. A. Allan, R. Armstrong *et al.*, "A Component Architecture for High-Performance Scientific Computing," *Intl. J. High-Performance Computing Applications*, vol. 20, no. 2, pp. 163–202, Jul. 2006.

[15] F. Liu, M. Sosonkina, and R. Bramly, "A New Approach: Component-Based Multi-physics Coupling through CCA-LISI," in *ICCSA 2010, Part II, LNCS 6017*. Springer-Verlag Berlin / Heidelberg, 2010, pp. 503–518.

[16] T. Goodale, G. Allen *et al.*, "The Cactus Framework and Toolkit: Design and Applications," in *LNCS 2565*. Springer-Verlag Berlin / Heidelberg, Apr. 2003, pp. 197–227.

[17] H. El-Rewini, H. Ali, and T. Lewis, "Task scheduling in multiprocessing systems," *Computer*, vol. 28, no. 12, pp. 27–37, Dec. 1995.

[18] Y.-K. Kwok and I. Ahmad, "Benchmarking and Comparison of the Task Graph Scheduling Algorithms," *Journal of Parallel and Distributed Computing*, vol. 59, no. 3, pp. 381–422, Dec. 1999.

[19] H. Casanova, F. Desprez, and F. Suter, "On cluster resource allocation for multiple parallel task graphs," *Journal of Parallel and Distributed Computing*, vol. 70, pp. 1193–1203, 2010.

[20] J. Hegewald, M. Krafczyk, J. Tölke, and A. G. Hoekstra, "An agent-based coupling platform for complex automata," in *LNCS 5102*, M. Bubak *et al.*, Eds., 2008, pp. 227–233.

[21] J. Hegewald, "The Multiscale Coupling Library and Environment (MUSCLE)," http://muscle.berlios.de/, jan 2010.

[22] D. P. Mitchell and M. J. Merritt, "A distributed algorithm for deadlock detection and resolution," in *PODC '84*. Vancouver, Canada: ACM, 1984, pp. 282–284.

[23] JADE, "Java agent development framework," http://jade.tilab.com/, 2011.

[24] V. Kravtsov, A. Schuster, D. Carmeli, K. Kurowski, and W. Dubitzky, "Grid-enabling complex system applications with QosCosGrid: An architectural perspective," in *GCA '08*, Las Vegas, Nevada, USA, 2008, pp. 168–174.

[25] D. J. W. Evans, P. V. Lawford *et al.*, "The application of multiscale modelling to the process of development and prevention of stenosis in a stented coronary artery," *Philosophical Transactions of the Royal Society A*, vol. 366, pp. 3343–3360, 2008.

[26] A. Caiazzo, D. J. W. Evans *et al.*, "A Complex Automata approach for In-stent Restenosis: two-dimensional multiscale modeling and simulations," *Journal of Computational Science*, vol. 2, no. 1, pp. 9–17, Mar. 2011.

[27] J. L. Suter and P. V. Coveney, "Computer simulation study of the materials properties of intercalated and exfoliated poly(ethylene)glycol clay nanocomposites," *Soft Matter*, vol. 5, no. 11, pp. 2239–2251, Apr. 2009.

[28] H. Tahir, A. G. Hoekstra *et al.*, "Multiscale simulations of the dynamics of in-stent restenosis: impact of stent deployment and design," *Interface Focus*, vol. 1, no. 3, pp. 365–373, Apr. 2011.

[29] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, 1995.

[30] S. J. Clark, M. D. Segall *et al.*, "First principles methods using CASTEP," *Zeitschrift für Kristallographie*, vol. 220, no. 5-6-2005, pp. 567–570, May 2005.

[31] A. G. Hoekstra, J.-L. Falcone, and A. Caiazzo, "Multi-scale Modeling with Cellular Automata: The Complex Automata Approach," in *LNCS 5191*. Springer-Verlag Berlin / Heidelberg, 2008, pp. 192–199.

[32] A. G. Hoekstra, E. Lorenz, J.-L. Falcone, and B. Chopard, "Towards a complex automata framework for multi-scale modeling: formalism and the scale separation map," in *ICCS 2007, Part I, LNCS 4487*, Y. Shi *et al.*, Eds., 2007, pp. 922–930.