Towards a Software Engineering Approach to Multi-Scale Modeling and Simulation

M. Maouche\textsuperscript{1}\textsuperscript{*} and M. Bettaz\textsuperscript{2}

\textsuperscript{1}Faculty of Information Technology - Philadelphia University – Jordan
\textsuperscript{2}Faculty of Information Technology - Philadelphia University - Jordan
\textsuperscript{1}mmaouch@philadelphia.edu.jo, \textsuperscript{2}mbettaz@philadelphia.edu.jo

Abstract

In this work we propose a development methodology aiming to bridge the gap between approaches used by (e-)science communities to develop their modeling frameworks, and model driven engineering approaches used to develop modeling frameworks with similar complexity. The proposed methodology relies on a sound integration of UML-MARTE, CSP-OZ, and PyCSP. We show, in this paper, how to exploit this similarity to bridge the gap between both approaches. A first contribution of this paper consists in proposing a sound software development methodology for the construction of critical reactive systems in general and multi-scale simulation frameworks in particular. A second contribution consists in adding a new sub-profile for MARTE, the so-called SSRM (Specific Software Resource Model), dedicated to the modeling of multi-scale simulation frameworks. MUSCLE [9], a recent distributed multi-scale simulation framework, serves as a case study in this work. The proposed SSRM sub-profile is intended to define specific software resources that capture a subset of MUSCLE core concepts. A third contribution consists in sketching a formal semantic framework for our software development methodology aiming at ensuring a sound integration (from a semantic point of view) of UML-MARTE, CSP-OZ and PyCSP.

Keywords: Multi-scale Simulation, Software Development Methodologies, UML-MARTE, CSP-OZ, PyCSP

1. Introduction

The discipline of modeling is widely used within e-science and engineering. On one side the Model Driven Software Engineering (MDSE) community emphasizes the use of models along the whole (software) development life cycle with the intent to build systems. Usually, models are created using specific visual languages such as UML (Unified Modeling Language) and its derivatives; models are then used to analyze properties and create code by transformations and refinements. On another side, the e-scientific community also uses modeling but here the intent is to simulate real world phenomenon (their behaviors over time) using simulation engines. Such engines must be able to execute e-science models independently of the programming languages used to implement them, allowing by the way the ability to reuse legacy (e-science) model codes. The authors in [1] report a series of contributions adopting model driven engineering principles in the construction of distributed simulation systems. These systems are intended to provide modeling/simulation frameworks allowing (e-science/engineering) modelers to first create and implement their models, then to simulate them.

\textsuperscript{*} Corresponding Author
Our objective in this work is to set a software development methodology, integrating model driven engineering and formal development, and targeting the development of distributed simulation systems in particular and reactive systems in general. The main contribution of this paper consists in showing the applicability of our methodology to the development of multi-scale simulation frameworks.

The rest of the paper is organized as follows: Section 2 presents an overview of the proposed software development methodology while Section 3 outlines the related works. In Section 4 we introduce the core concepts of MUSCLE, a recent multi-scale modeling/simulation framework. Section 5 details how our software development methodology is applied in the context of MUSCLE, and also how some MUSCLE core concepts are modelled using MARTE profile. The semantic issues related to our software development methodology are sketched in Section 6, and finally some conclusions and future works are given in the Section 7.

2. Our Software Development Methodology

Adopting an integrated development environment (IDE) associating modeling and formal specifications for the development of systems is among the three recommendations reported in [2]. Visual modeling, using UML language for instance, is helpful at the requirement level, where an informal high level architecture of the system under design is modeled. Formal specifications are necessary when properties analysis and correctness of the code obtained through refinement are requirements of the system under construction. The literature reports various integrated development methodologies. Among the valuable contributions, we may mention [3, 4] where an integrated development environment based on UML, CSP, and Java is used to develop and implement a generic framework for distributed simulations. We may also mention the work in [5] where an integrated development environment associating UML-RT (a derivative of UML) for requirements level, CSP-OZ (a formal specification language) for the design level, and Java for the implementation level, is used for the development of reactive systems. Our methodology consists in using UML-MARTE profile [7] at the requirement level, CSP-OZ [8] at the design level and PyCSP [11] at the programming level. The motivation behind this choice will be explained in the remaining part of this section. A set of suitable translation rules allow to transform UML-MARTE models into their corresponding CSP-OZ specifications, and to transform CSP-OZ specifications into their corresponding PyCSP codes. These rules are not discussed in this paper.

One major issue raised when using the previous mentioned works is the problem of consistency for instance between various UML requirement models on one side and between UML models and design level formal specifications on another side. Such issues and semantics concerns are sketched in Section 6.

2.1. Requirement Level

For sake of readability we will on the following use “es-” as a short-hand for e-science and “app-” as a short-hand for application. UML-MARTE is a standardized derivative of UML dedicated to embedded and real time systems modeling. It defines models for applications as well as models for execution platforms that host applications. The term "Execution platform" covers both hardware and software platforms. For this purpose UML-MARTE defines generic models for hardware and software resources that can be instantiated according to the desired execution platform. Although UML-MARTE targets mainly embedded and real time systems, we envisage to adopt it for the modeling of distributed simulation systems. Four main motivations are behind our proposal. First, there is a kind of similarity between modeling an application together with its (hardware/software) execution platform, and modeling an es-phenomenon together with its simulation engine (i.e., "execution platform"). Roughly speaking the idea consists in
considering implemented es-models as (UML-MARTE) app-models encapsulating es-models, and considering simulation engines as (UML-MARTE) execution platform models. This presupposes that specific software resources intended to take in charge simulation engine core concepts are defined and modeled as UML-MARTE software resources [7]. This may be done by specializing the MARTE Generic Resource Model (GRM) sub-profile to define a specific resource model sub-profile, called SSRM, intended to capture the multi-scale simulation core concepts. Second, both UML-MARTE profile and multi-scale modeling and simulation are component based models, thus making their rapprochement realizable. In [9] the authors advocate a component-based approach for distributed multi-disciplinary and multi-scale es-simulations, where es-models (components) are decomposed into a set of coordinated es-models (sub-components) that interact through specific channels. In our proposal these es-components (i.e., implementations of es-models) are seen as (UML-MARTE) app-models. Third, UML-MARTE supports concurrency, synchronization and communication features which are useful for modeling distributed and concurrent simulation systems. Fourth, UML-MARTE defines an explicit model for the (logical/physical) time concept so that it may cover the development of both event-driven and time-driven simulators.

2.2. Design Level

CSP-OZ [8] is a hybrid formal language dedicated to the specification of reactive systems. It allows to specify in an integrated way the state and the behavior views of reactive systems. CSP-OZ presents two interesting features that are useful in the context of our software development methodology: First, a CSP-OZ specification may be refined in such a way that the gap between specification and implementation (coding) may be incrementally reduced; second, CSP-OZ specifications may be formally analyzed, for instance by checking the correctness of specification refinements and verifying properties relevant to their behavior (liveness/safety properties). It is important to recall that in a multi-scale simulation a set of distributed es-models run and interact concurrently inducing potential deadlocks [9]. Therefore it is important to provide means to formally specify and verify the concurrent behavior of multi-scale simulations.

2.3. Implementation Level

PyCSP, a Python programming environment augmented with a CSP library, is used as a target programming language in our software development methodology; more precisely CSP-OZ specifications built at the design level are translated into PyCSP code. PyCSP seems to be adequate for a smooth transition from a sufficiently refined CSP-OZ specification to a PyCSP code. Moreover Python has been used to develop various simulation engines [10]. Finally PyCSP has shown its strengths for clusters platforms [11], thus its suitability for implementing highly parallel systems

3. Related Works

Past and recent works [2, 3, 5, 14, 15] propose to bring together UML, widely used in industry and academia, and formal methods. Mainly three different approaches to integrate formal methods with UML are reported:

The first approach described in [3, 4] consists in using UML at each level of the software development (requirement, design, and implementation) to model various complementary views of systems under construction, while authorizing the use of formal languages, like for instance process algebra, to specify some critical views (if any).
This is motivated by the need to verify some critical aspects of systems. Once formally checked, these formal specifications are brought back to their 'equivalent' UML models using a set of well-defined transformation rules. UML models may be extended and refined along the whole development life cycle. In [3] the authors prescribe a set of well-defined rules to ensure the correctness of the UML models refinement and extensions. This software development methodology has been built and tailored to the development of a generic simulation framework.

The second approach described in [14, 15] recommends to keep the use of UML along the whole development life cycle; the main motivation here is the benefit of the visual modeling. This approach endowed UML with a sound semantics addressing by the way the consistency issue of the various UML models and the correctness issue of the refinement process of UML models. Compared to the work in [5], the work in [14] suggests to define a comprehensive semantics for UML using a so-called systems model, thus bringing the semantics of all UML models to a common formalism like for instance transition systems. Quoting [15]: 'However this is a thorny business as every detail has to be encoded into one, necessarily quite complex semantics'. That is why the authors of [15] suggest to adopt the institution theory as a semantic framework for UML language. Institution theory [6] is a mathematical theory that demonstrated its effectiveness to cope with semantic issues in a solid and elegant way. Roughly speaking, UML (sub)-languages are first equipped with institutions capturing their individual semantics in an abstract way and then adequate formal mappings (morphisms/co-morphisms) relating these institutions are set. Institution morphisms are well suited to capture the issues of models consistency and refinement, while co-morphisms are useful to capture the encoding of a source formalism into a target formalism in an abstract way, that is independently form the underlying logical frameworks. Furthermore the growing family of available institutions backing various formal languages [16, 17, 18] makes this approach very attractive and less expensive.

The third approach described in [5] consists in adopting UML and/or its derivatives to model systems (their structural and behavioral view) at the requirement level. Here the methodology takes benefit from the visual capabilities provided by UML to sketch initial requirements and architecture of the systems to be implemented. The obtained UML requirement models are then translated into formal specifications which are incrementally refined until they may be directly coded using a suitable programming language. In [5] the authors adopt UML-RT profile, a derivative of UML, for the requirement level, and CSP-OZ for the design level. CSP-OZ specifications are then gradually brought to Java code using special programming languages that integrate formal assertions with conventional code, such as JVM (Java Modeling Language). It is worthwhile to note that the use of such special programming languages is recommended in [2]. The ability to annotate Java code with formal assertions allows to preserve the precision of the formal specification in the implementation [5]. The semantics underlying the UML-RT to CSP-OZ translation is addressed in [12, 13].

From a methodological point of view, our proposal builds on [15] and [5]. First UML is retained to capture informal requirements and sketch initial systems architecture. Adopting CSP-OZ at the design level contributes to address correctness issues inherent to the target of our methodology, that is building reactive systems in general and distributed simulation frameworks in particular. CSP-OZ, thanks to its CSP (Failure-Divergence) semantics, takes benefits from the available CSP model checking tools for properties verification. However we retain UML-MARTE rather than UML-RT (used in [5]) for many reasons. UML-MARTE component model is closer to the conventional UML2 component model with additional advanced features while UML-RT is based on a very specific and less neutral component model.
Moreover UML-RT does not provide specific features supporting the modeling of execution platforms intended to host UML-RT applications. Finally, UML-RT does not support an explicit notion of time, thus making it not suitable for time-driven simulation systems. UML-MARTE profile supports both the modeling of applications as well as the modeling of execution platforms, where such applications are hosted.

Our software development methodology takes advantage of this last feature to develop distributed simulation systems. Moreover UML-MARTE supports the concept of time in an explicit way. Dealing explicitly with time offers two opportunities in the field of simulations: Ability to address event based simulators as well as time-driven simulators with the ability to conduct performance evaluation of simulations, thanks to the UML-MARTE sub-profile dedicated to performance issues.

From a semantic point of view, we adopt an institution-based framework for our software development methodology. First of all, institutions capture in an abstract and effective way consistency and translation issues. Second, institutions for CSP, OZ (Object Z) are available, while sketches of institutions for UML profiles are under elaboration [15].

4. Overview on MUSCLE

The foundations for MUSCLE are presented in [9]. MUSCLE (Multi-scale Coupling Library and Environment) is a modeling and simulation framework built on the ideas in [26, 27]. In MUSCLE, multi-scale modeling consists in decomposing an es-model of the phenomenon under study into a set of MUSCLE single scale models that are coupled according to a given topology. These single scale models are independent from each other in the sense that they only rely on messages (observations) they send or receive at specific ports. Each single scale model is handled as an independent component owning its simulation time, spatial and temporal scales. It synchronizes with other single scale models by exchanging messages carrying time points. MUSCLE single scale models encapsulate the code of es-models, therefore abstracting from the programming languages used to code these models. Conduits, special kinds of communication channels, are used to couple single scale models through their input/output ports. Moreover MUSCLE provides additional specific computational elements like filters and mappers that are kinds of smart conduits. Data exchanged between single scale models can be modified in transit because data expected by a single scale model does not automatically match the observation of the other single scale model. Filters change data in a single conduit while mappers may combine data from multiple sources or extract multiple data from one observation [9, 26, and 27].

The authors of [9] formalize MUSCLE core concepts like single scale models, conduits, filters, mappers, observations (data passed between single scale models). The execution flows of models are formalized in terms of a Sub-model Execution Loop (SEL) which specifies a general execution loop to be followed during the execution (simulation) of a single scale model.

Figure 1 shows an example of a simple multi-scale model composed of two single scale models (A and B) encapsulated in their respective controller C (the arrow in the figure depicts the encapsulation), one filter (F) and two conduits.
5. A Software Development Methodology for Simulation Frameworks

In this paper we focus on the requirement phase of the software development methodology, where the MUSCLE multi-scale framework is used as a case study.

The requirement phase consists in building a set of UML-MARTE models that describe the applications to be developed (i.e., multi-scale models), their execution platform (simulation run time environment) and the allocation of the application models to their corresponding execution platform models. To make the discussion more concrete we will describe in the following the processes followed by the UML-MARTE development methodology and the methodology followed for building (e-science) models and simulation frameworks.

5.1. UML-MARTE and Multi-Scale Simulation Modeling

Usually, UML-MARTE system models are divided into three sub-models, following the so-called Y structure [20, 21, 22, 23, 24 and 25]:

The Platform Independent Model (PIM)

This model is intended to describe various views of the systems to be developed. Here the functional and non-functional aspects of systems are modeled. More concretely models related to the following system views are built: data view, functional view, application, concurrency view, communication view, and memory space view. Specific software resources defined in the SRM (Software Resource Model sub-profile of MARTE), may be used to build PIM models. The SRM thus acts as an API for PIM models developers.

The Platform Dependent Model (PDM)

This model is intended to model execution platforms that support and host upper described systems. Execution platform models include hardware as well as software resource models. Thus PDM models developers may use both SRM and HRM (Hardware Resource Model) sub-profiles to create PDM models.

The Platform Specific Model (PSM)

This model is intended to describe the architectural view of systems as well as the allocation models that describe the allocation of PIM models to their associated PDM models.

The methodology described hereafter refers to the approach used by the multi-scale simulation community [9, 26, and 27]. A deep analysis of the contributions in [9, and 26] shows that multi-scale modeling and simulation frameworks follow a more or less similar process to the one followed by the UML-MARTE methodology.

More concretely:
-es-modelers build multi-scale es-models by coupling specific single scale es-models according to given configurations. Usually, multi-scale modeling frameworks provide a set of APIs including services on specific resources such as the MUSCLE Controllers, Filters, Mappers, and Conduits. These resources are used to create single scale es-models and multi-scale es-models. At this point we observe a similarity with what is done by PIM modelers in the sense that single scale es-models and multi-scale es-models may be seen as kinds of PIM app-models that are built using specific software resources defined in a particular SSRM (Specific Software Resource Model) sub-profile.

-Multi-scale simulation frameworks provide runtime environment for the execution (simulation) of multi-scale models. Particular instances of these run-time environments are configured and instantiated in such a way to support the execution of multi-scale models. Specific software resources, offered by multi-scale simulation frameworks, may be composed to create run-time environment instances tailored to multi-scale models. MUSCLE Local Manager and Simulation Manager are examples of such specific software resources. This step of the process is similar to the so-called PDM modeling in the sense that a particular run-time environment instance, implemented as a network of local manager instances, may be seen as a kind of PDM model.

-Single scale models that are parts of a multi-scale model are then allocated to local managers belonging to an instantiated run-time environment. This step is similar to the MARTE allocation modeling.

Figure 2 depicts an illustration of our approach. It shows how the (MARTE) Y structure may be applied to a simple example of MUSCLE multi-scale simulation. The SSRM provides specific resources for PIM modeling (Controllers, Filters, and Conduits) and PDM modeling (Local Managers, Simulation Managers, and Communication Resources).

5.2. Positioning the Specific Software Resource Model (SSRM) in MARTE

UML-MARTE modeling of MUSCLE multi-scale framework needs to define specific software resources tailored to this framework. We suggest to extend UML-MARTE profile to support in a native way the MUSCLE core concepts. These MUSCLE core concepts, i.e., single scale models, controllers, conduits, ports, filters, mappers, simulation managers, and local managers are seen as specific MARTE resources.
Originally, MARTE provides a package, called DRM (Detailed Resource Model), that is composed of the following sub-packages: SRM and HRM packages which are dedicated for the modeling of software and hardware platforms respectively [7]. These sub-packages import the GRM package which defines the concept of resource in an abstract and generic way.

We propose to enrich the MARTE DRM with a new package, called SSRM (Specific Software Resource Model), intended to provide a specialization of the GRM for dealing with MUSCLE multi-scale simulation core concepts (see Figure 3).

Hereafter we focus only on the modeling of two core concepts of MUSCLE, namely controllers and conduits. The modeling of modeling of the other MUSCLE core concepts are under construction.

5.3. SSRM Modeling of MUSCLE Controller

We define the root concept of Simulation Resource which is a specialization of the concept of Software Resource defined in the original SRM package. MUSCLE Controller is then defined as a kind of Simulation Resource. Because individual MUSCLE local managers may concurrently support more than one controller, MUSCLE controllers are seen as concurrent schedulable resources which are a kind of resources already defined in the SRM.

Figure 4 depicts the fragment of the proposed SSRM package that models MUSCLE Controllers.
The dynamic behavior of MUSCLE Controllers [9] is described by the so-called SEL (Sub-model Execution Loop) which is a cycle of single scale simulation steps; single scale simulation steps form a sequence of calls to a set of specific operators: Observation operators (intermediate and final observations $O_i$ and $O_f$ operators) depict the sending of observations from one single scale es-model to its coupled single scale es-model. $F_i$ operator denotes an initialization operator that is executed before the SEL cycles start, while $S$ operator denotes the actual solving computations of single scale es-models which are encapsulated into MUSCLE controllers. Finally, $B$ operator depicts the boundary conditions that exist at the frontier between two coupled single scale models having different scales. A SEL loop is characterized by its start time, its (single scale simulation) duration and the time scale of the single scale model simulation. Moreover, MUSCLE controllers have input ports and output ports which allow them to exchange messages.

5.4. SSRM Modeling of MUSCLE Conduits

MUSCLE controllers exchange simulation data through their input and output ports. These data are transported through the so-called Smart Conduits. We model MUSCLE Conduits as a kind of Simulation Resource. These are also considered as a specialization of the concept of Communication Resource defined in the GRM. MUSCLE distinguishes two types of conduits: The so-called Plain Conduit which transports data between controllers without any transformation, and the so-called Scale Bridge Conduit which transform the data along their conveying through the conduit. MUSCLE provides two specific scale bridge conduits: Filter and Mapper. The last one plays also the role of a Simulation Controller.
Figure 5 depicts the fragment of SSRM that models the concept of MUCLE conduit.

![SSRM Model of MUCLE Conduit](image)

**Figure 5. SSRM Model of MUCLE Conduit**

6. **Semantic Issues**

We adopt the institution theory [6, and 15] as a semantic framework for the proposed software development methodology. The institution theory provides sound concepts to address the issue of the correctness of the mapping between different formalisms. More precisely such formalisms are equipped with appropriate institutions [18 and 19]; their mappings are expressed in terms of appropriate morphisms and co-morphisms between their backing institutions, allowing by the way the formalization of translation rules mentioned in Section 2.

![Institution Morphisms between Languages and UML Diagrams](image)

**Figure 6. Institution Morphisms between Languages and UML Diagrams**
Figure 6 shows the transformations to be developed between MARTE modeling diagrams (state machines, component diagrams, and class diagrams) and additional languages (CSP, CSP-OZ and PyCSP). The arrows depict the required co-morphisms which correspond to the encoding of one logic that underlies a given diagram type or language to another one.

7. Conclusion and Future Works

In this work we proposed a software development methodology aiming to bridge the gap between approaches used by es-communities to develop their modeling frameworks and model driven engineering approaches used to develop systems with similar complexity.

The first contribution of this work consisted in proposing a software development methodology targeting the reactive systems in general and the multi-scale simulation frameworks in particular. This methodology relies on UML-MARTE profile for the requirement level, CSP-OZ formal language for the design level and PyCSP programming language for the implementation level. One of the important feature of this methodology is its sound semantics based on the institution theory, a solid mathematical framework.

The second contribution consisted in showing through an example how to apply the so-called (MARTE) Y structure modeling methodology to model multi-scale simulations, thus bridging the gap between the es-community and the MDSE community.

The third contribution consisted in defining a specific software resource model (SSRM) as a new sub-package of the MARTE DRM package. SSRM is intended to model the core concepts of MUSCLE, a recent and known multi-scale simulation framework. In particular, SSRM models of two MUSCLE core concepts, namely controllers and conduits are presented in the paper.

The fourth contribution consisted in sketching an institutional framework for our proposed software development methodology. This institutional framework is intended to give a solid foundation to the semantics of our methodology, and to ensure a sound integration (from a semantic point of view) of UML-MARTE, CSP-OZ and PyCSP. An illustrative diagram showing a set of relevant institutions and their connections depicts our institutional framework.

Future works are planned in the following three directions: The first direction consists in improving the proposed SSRM package to take in charge the other MUSCLE core concepts, the second direction consists in investigating and developing a Simulation Resource Model package, not to confuse with the UML-MARTE SRM (Software Resource Model) package, that deals with multi-scale core concepts in an abstract and generic way, i.e., independently from any specific multi-scale simulation framework, and the last direction consists in setting the institutional framework for our methodology, i.e., defining institutions for the following formalisms: UML-MARTE, CSP-OZ and PyCSP, and then elaborating the set of co-morphisms depicted in the diagram shown in section 6.

Acknowledgments

This work was fully supported by a grant from the Research Deanship – Philadelphia University- Jordan.
References


Authors

Mourad Maouche, he received the MSc and PhD Degrees in computer science from the Jussieu University in Paris, France, in 1982 and 1986 respectively. He is currently an associate professor at the Software Engineering Department, Philadelphia University, Jordan. His research areas of interest include formal specification, software engineering, modeling and simulation.

Mohamed Bettaz, he received the MSc and PhD Degrees in computer science from the Czech Technical University in Prague, Czech Republic, in 1975 and 1984 respectively. He is currently a computer science professor at the Computer Science Department, Philadelphia University, Jordan. His research areas of interest include system specification, software engineering, software architecture, distributed multi-scale modeling and simulation, wireless and mobile networks, and information security. He is a member of the working group WG1.3 (Foundations of System Specification) of IFIP.