Project Title:

CSP-OZ: Semantics, Tools and Applications

Supported By:

Research Deanship

Philadelphia University

TECHNICAL REPORT

Number: TR-2016-01

Date: March, 2016

Authors:

Prof. Dr M. Bettaz

Dr M. Maouche

المنارة للاستشارات

www.manaraa.com

Abstract. In this ongoing 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. The motivation behind the use of MARTE, a UML profile dedicated for embedded and real time systems, is the similarity between its modeling approach, based on the so-called Y structure, and the approaches used in the multiscale simulation field. We show, in this paper, how to exploit this similarity to bridge the gap between both. A first contribution of this work consists in adding a new sub-profile for MARTE, the called SSRM (Specific Software Resource Model), dedicated to the modeling of multiscale simulation frameworks. The SSRM subprofile is intended to define specific resources that capture multiscale simulation core concepts. A second contribution of this work consists in setting a formal semantic framework for our development methodology aiming at ensuring a sound integration (from a semantic point of view) of UML-MARTE, CSP-OZ and PyCSP. To this end, we adopt a semantic framework based on the institution theory, a powerful and appropriate mathematical theory commonly used in computer science for semantics purposes.

Keywords: Multiscale Simulation Frameworks, Software Develpment Methodologies, UML-MARTE, CSP-OZ, PyCSP

1 Introduction

The discipline of modeling is widely used within e-science and engineering.On one side the MDSE (Model Driven Software Engineering) 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 phenomena (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. Topcu et al.] reports in [1] 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 on a distributed platform. 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 to show

www.manaraa.com

the applicability of our methodology to the development of multiscale simulation frameworks.

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 to be built 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. 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 addressed in Section 5.

1.1 Requirement Level

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 (e-science) phenomena together with its simulation engine (i.e., " execution platform"). Roughly speaking the idea consists in considering implemented (e-science) models as UML-MARTE (application) models encapsulating (e-science) models, and considering simulation engines as UML-MARTE (execution platform) models. It 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 new

www.manaraa.com

specific resource model sub-profile, called SSRM, intended to capture the multiscale simulation core concepts. Second both UML-MARTE profile and mustiscale modeling and simulation are component based models, thus making their rapprochement realizable. [9] advocates a component based approach for distributed multi-disciplinary and multiscale (e-science) simulations where (e-science) models are decomposed into a set of coordinated (e-science)- submodels (components) that interact through specific channels. In our proposal these (e-sciences) components (i.e., implementions o f (e-science) submodels) are seen as UML-MARTE applications 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 simulators and time-driven simulators.

1.2 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 checking the correctness of specification refinements and verifying properties relevant to their behavior (liveness/safety properties). It is important to recall that in a multiscale simulation a set of distributed (e-science) 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 multiscale simulations.

1.3 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 suitable for implementing highly parallel systems.

The paper is organized as follows: Section 2 presents the related works. In section3 we introduce the core concepts of MUSCLE a recent multiscale modeling/simulation framework. Section 4 is devoted to the description of our software developement methology. The semantic issues related to our software development methodology are outlined in section 5, and finally some conclusions and future works are given in the section 6.

www.manaraa.com

2 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 one described in [3, 4] consists in using UML at each level of the software development (requirement, design, 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 one 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; to endow UML 1 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. Instead of [5], the authors of [14] suggest to define a comprehensive semantics for UML using a so-called systems model, that is bringing the semantics of all UML models to a common formalism like for instance transition systems. Quoting [15]: 'However this 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 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 one 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. More precisely [5], targeting the development of reactive systems, adopts UML-RT profile, a derivative of UML, for the requirement level, CSP-OZ for the design level. CSP-OZ specifications

www.manaraa.com

are 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, 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 Divergent) 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-MART component model is closer to the conventional UML2 component model with additional advanced features while UML-RT is based on a very specific and no 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 the execution platform where these applications are hosted. Our software development methodology takes advantage of this late 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 and also the ability to conduct performance evaluation of simulations, thanks to UML-MARTE subprofile 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].

3 Overview of MUSCLE

The authors of [9] present foundations for multiscale modeling and distributed multiscale computing. MUSCLE (Multiscale Coupling Library and Environment) is a modeling and simulation framework built on these ideas [25, 26] . In MUSCLE, multiscale modeling consists in decomposing an (e-science) model of the phenomena under study into a set of MUSCLE single scale (sub)models that are coupled according to a given topology. These single scale (sub)models are independent 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 inde-

www.manaraa.com

pendent component owning its simulation time, spatial and temporal scales. It synchronizes with other single scale (sub)models by exchanging messages carrying time points. MUSCLE single scale (sub)models encapsulate the code of (e-science) models, therefore abstracting from the programming languages used to code the (e-science) models.

Conduits, special kinds of communication channels, are used to couple single scale (sub)models through their input/output ports. Moreover MUSCLE provides additional specific computational elements like filters and mappers that can be applied to conduits. Data exchanged between single scale (sub)models can be modified in transit because data expected by a single scale (sub)model does not automatically match the observation of the other single scale (sub)model. Filters change data in a single conduit while mappers may combine data from multiple sources or extract multiple data from one observation.

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

4 Applying Our Software Development Methodology for Simulation Frameworks

Our software development methodology involves three phases: a requirement phase where UML-MARTE is used to build high level abstract models of systems under construction, followed by a design phase where produced UML-MARTE models are transformed into CSP-OZ specifications which may be formally checked (verification of systems properties), and finally an implementation phase where the produced CSP-OZ specifications are refined and then transformed into PyCSP code.

This paper focuses on the requirement phase of our methodology, while the design and implementation phases are sketched. These last ones will be detailed in future works. The requirement phase is conducted in such a way to be tailored for the development of (e-science) modeling and simulation frameworks. MUSCLE multiscale framework serves as a running example for this work.

4.1 Requirement Phase

Generally, the deliverables produced at the requirement level are (software engineering) models that describe the targeted application at a high level of abstraction. UML-MARTE, a UML profile tailored for the development of real time and embedded systems, provides useful features and capabilities that make it suitable for the development of modeling and simulation frameworks. 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.

www.manaraa.com

1. UML-MARTE Methodology:
   According to [24] UML-MARTE system models are divided into three sub-models, following the so-called Y structure.
   (a) Platform Independent Model (PIM) which is intended to model 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 so-called SRM (Software Resource Model) sub-profile of MARTE, may be used to build PIM models. Thus SRM acts as an API for PIM models developers.
   (b) Platform Dependant Model (PDM) which is intended to model execution platforms that support and host upper 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.
   (c) Platform Specific Model (PSM) which is intended to model the architectural view of systems as well as the allocation models that describe the allocation of PIM models to their associated PDM models.
2. Multiscale Modeling and Simulation Methodology: The methodology described in this part refers to the approach used by the multiscale simulation community [9, 25, 26]. A deep analysis of the contributions in [9, 25] shows that multiscale modeling and simulation frameworks follow a more or less similar process. More concretely:
   (a) (e-science) modelers build multiscale models by coupling specific single scale models according to given configurations. Usually, multiscale modeling frameworks provide a set of APIs including services on specific resources such as **MUSCLE Controller**, **Filter, Mapper**, and **Conduit**. These resources are used to create (e-science) single scale and multiscale models. At this point we observe a similarity with what is done by PIM modelers in the sense that **(e-science) single scale** and **(e-science) multiscale** may be seen as kinds of PIM models that are built using specific software resources defined in a particular "SSRM" (Specific Software Resource Model) sub-profile.
   (b) Multiscale simulation frameworks provide runtime environment for the execution (simulation) of multiscale models. Particular instances of these run time environments are configured and instantiated in such a way to support the execution of multiscale models. Specific software resources, offered by mutiscale simulation frameworks, may be composed to create run time environment instances tailored to multiscale models. **Local manager, 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 managers instances, may be seen as a kind "PDM model".

المنارة للاستشارات
www.manaraa.com

(c) Single scale models that are parts of a multiscale model are then allocated to local managers belonging to an instantiated run time environment. This step is similar to the MARTE Allocation modeling.

In the context of the MARTE modeling of MUSCLE multiscale framework, specific software resources tailored to this framework need to be defined. At this point two approaches may be envisaged:

1. The first approach consists in creating MARTE models for the various MUSCLE core concepts composing MUSCLE. These MARTE models are seen as PIM models which are built using software resources defined in the SRM sub-profile.
2. The second one consists in extending MARTE profile to support in a native way the core concepts related to multiscale modeling and simulation frameworks. For instance the concepts of scale, single scale model, filter, mapper, conduit. This approach requires much more efforts in the sense that an exhaustive set of agreed and unified multiscale concepts need to be identified, and specific metamodels for these identified concepts need to be defined.

One of the main objectives of our research work is to not address a specific multiscale simulation platform but to address generic multiscale simulation frameworks. The second approach seems to be more suitable for our objective.

Our contribution, here, consists in showing how to exploit this similarity to conduct a UML-MARTE modeling of (e-science) modeling and simulation frameworks. Applying our MARTE approach for these frameworks relies on the following principles:

First of all, core concepts of MUSCLE (as an example of simulation frameworks) single scale models, controller, conduit, port, filter, mapper, simulation manager, local manager are seen as specific MARTE resources. For this purpose we propose to define a new Specific Software Resource (SSR) sub-profile intended for multiscale simulation domain. The following diagram shows how this specific MARTE sub-profile may be defined by specializing the MARTE Generic Resource Model (GRM) sub-profile.

Hereafter we explain how the typical Y structure may be applied to multiscale simulation frameworks:

1. **PIM Level**: MUSCLE simulations, such as described by the (e-science) modeler in the form of configuration files, are considered as MARTE applications composed of the following MARTE models representing MUSCLE single scale models, MARTE models representing MUSCLE filters and mappers, and MARTE models representing MUSCLE couplings. The SSR sub-profile defines model elements that capture multiscale simulation core concepts.
   Figure 1 depicts the so-called MARTE Detailed Resource Model (DRM) package composed of the following packages: General Resource Model (GRM) package which includes the features which are required for dealing with the modeling of executing platforms at different level of details and the modeling

www.manaraa.com

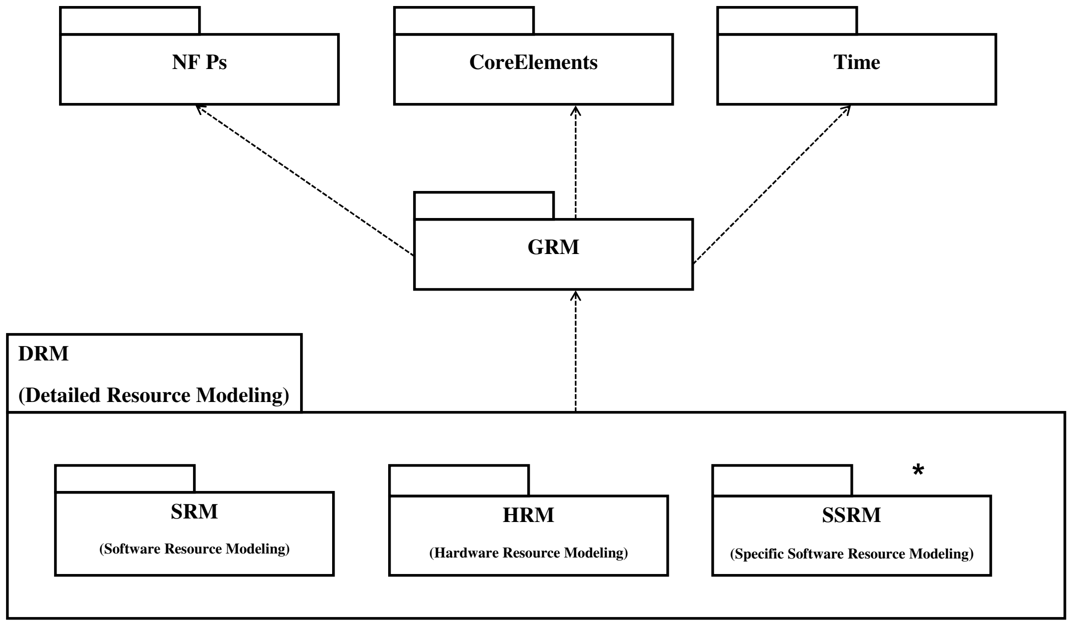
NF Ps
CoreElements
Time
GRM
DRM
(Detailed Resource Modeling)
SRM
(Software Resource Modeling)
HRM
(Hardware Resource Modeling)
*
SSRM
(Specific Software Resource Modeling)

Fig. 1. Extended-Detailed Resource Modeling (DRM)

of both hardware and software platforms, Software Resource Model(SRM) and Hardware Resource Model (HRM) packages providing a specialization of this general resource model for software and hardware related platforms respectively [7]. We propose in this paper to enrich the Detailed Resource Model with a new package, called Specific Software Resource Model (SSRM) package, intended to provide a specialization of GRM for dealing with multiscale simulation core concepts.

2. **PDM Level**: MUSCLE run time environment is modeled as a MARTE software execution platform composed of appropriate model elements defined in our specialized MARTE resource sub-profile, i.e.,**Local Manager, Simulation Manager**, and model elements defined in the SSRM sub-profile such as **Communication Resource** models, and Synchronization Resource models. The last two MARTE models provide the means of interaction between **Simulation Manager** and **Local Manager** models elements .
3. **PSM Level**: MARTE Allocation Modeling sub-profile provides a set of general concepts that concerns the allocation of functionality to entities responsible for the realization and execution [21]. It covers two main aspects: a

www.manaraa.com

spatial allocation of PIM models that model systems to PDM models that model execution platforms, and a temporal aspect (for instance scheduling of schedulable resources).

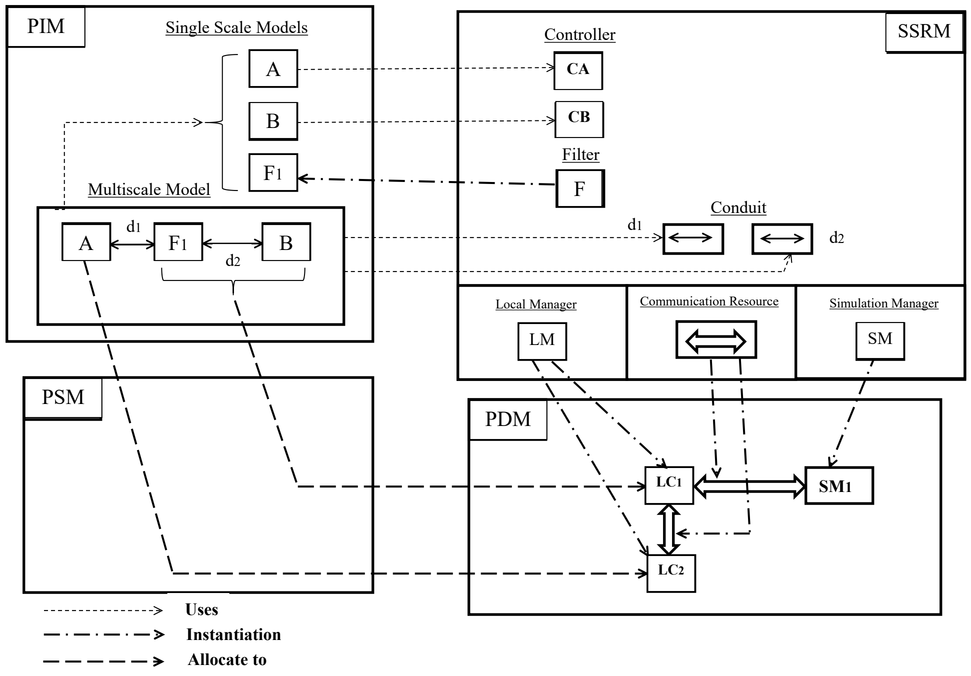

Fig. 2. A "Y structure" for a multiscale simulation example

In the context of our case study (e-science) models , defined in the PIM part, are allocated to specific **Local Manager**, and MARTE **Conduit** models are allocated to specific **Communication Resources** models defined in PDM part.

Figure 2 depicts an illustration of our approach. It shows how (MARTE) Y structure may be applied to a simple example of multiscale simulation. SSRM provides specific resources for PIM modeling (**Controller, Filter, Conduit**) and PDM modeling ((**Local manager, Simulation Manager, Communication Resource**). The figure shows a multiscale model composed of two single scale models (A and B), one filter (F1) and two conduits (d1 and d2). The modeler uses CA and CB models (instantiations of the controller resource) to build A and B. He/she uses also instantiations of the resources **Conduit** and

المنارة للاستشارات
www.manaraa.com

Filter to build a mutiscale scale simulation model. Both single scale models and mutiscale simulation model are defined in the PIM part of the Y structure. The PDM part models the simulation runtime environment. It is built by instantiating one instance of the resource **Simulation Manager**, two instances of the resource **Local Manager** and two instances of the resource **Communication Resource**. The PSM part models the allocation of PIM models to PDM models: model A is allocated to the LC1 model, (F1, d1, B) models are allocated to LC2 model, d1 model is allocated the instance of communication resource that links LC1 and LC2.

4.2 Design Phase

MARTE provides a generic component model subprofile that defines various kinds of model elements (application components, connectors, ..). System models (i.e., PID, PDM and PSM sub-models) built during the requirement phase are expressed in terms of component diagrams composed of MARTE component models which are interconnected via MARTE connectors. MARTE component diagrams may be organized in hierarchical structures; rtUnits and ppUNits are basic MARTE component models that are used at the lowest level of (hierarchical) component diagrams. rtUnits are active classes characterized by the services they provide, their communication ports, their attributes and also by their behavior usually expressed in terms of protocol state machines; ppUnits are similar to rtUnits except that they do not own an internal behavior (passive classes). The previous phase produces a set of MARTE components diagrams. In the design phase the produced MARTE component diagrams are translated into CSP-OZ specifications. These ones are then analyzed so that relevant system properties can be verified. Thus, potential deadlocks that occur during the concurrent execution of multiscale simulations [9] can be detected during the design phase.

A set of defined et of informal rules drive the translation process. These rules are described hereafter:

1. First the rtUnits and ppUnits, produced in the previous phase, are translated into their corresponding CSP-OZ classes. RtUnits services and attributes are respectively mapped into the so-called CSP-OZ communication-schema (class operations); rtUnits (ppUnits) ports are translated into the so-called CSP-OZ channels; rtUnits (ppUnits) attributes are translated into the so-called CSP-OZ data schema (class attributes) while the protocol state machine associated with rtUnits is mapped into its corresponding CSP process (i.e., CSP part of CSP-OZ classes). ppUnits are also translating into CSP-OZ classes with an empty CSP part.
2. Flat component diagrams are composed of rtUnits (and/or ppUnits) interconnected with MARTE connectors through ports. A flat component diagram is translated into a so-called CSP-OZ system class that includes:
   - a set of objects that belong to CSP-OZ classes resulting from the translation of rtUNits and ppUnits.

www.manaraa.com

  - a set of CSP channels.
- a CSP process that describes the overall behavior of the CSP-OZ system class. itemize

The details of these translations rules are not in the scope of this paper. A rigorous formal semantic for these informal transformation rules need to be set. This aspect is discussed in the section 5.

4.3 Implementation Phase

The previous phase produces a set of CSP-OZ classes (including the so called system classes). In the implementation phase, the produced CSP-OZ classes are translated into their corresponding PyCSP code. A set of defined informal rules drive the translation process. These rules are described hereafter:

- channels defined in CSP-OZ classes are mapped into PyCSP channels.
- communication-schema defined in CSP-OZ classes are mapped into Python functions.
- process parts of CSP-OZ classes rae mapped into (individual) PyCSP processes.
- CSP-OZ system classes are mapped into PyCSP processes. (individual) PyCSP processes, resulting from the translation of CSP-OZ classes, are combined using relevant CSP operators. The details of these translation rules are not in the scope of this paper. A rigorous formal semantic for these informal transformation rules need to be set. This aspect is discussed in the section 5.

5 Semantics Issues

We adopt the institution theory [6], a sound and powerful mathematical theory, as a semantic framework for the proposed software development methodology. The institution theory, widely used in computer science for semantic purposes [15, 16, 17, 18], allows to abstract from the logics underlying formalisms and languages. The following arguments are behind the motivations of our choice:

- Institution theory provides sound concepts to address the issue of the correctness of the mapping between different formalisms. More precisely formalisms are equipped with appropriate institutions; 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 introduced in the section 4.
- A subset of formalisms adopted by our software development methodology, i.e state machines [16], CSP [17] and OZ [18], are backed by appropriate institutions.

To this end we adopt the same approach as in [15]. Figure 3 shows the transformations to be developed between MARTE modeling diagrams (state machines, component diagrams, and class diagrams) and additional languages

www.manaraa.com

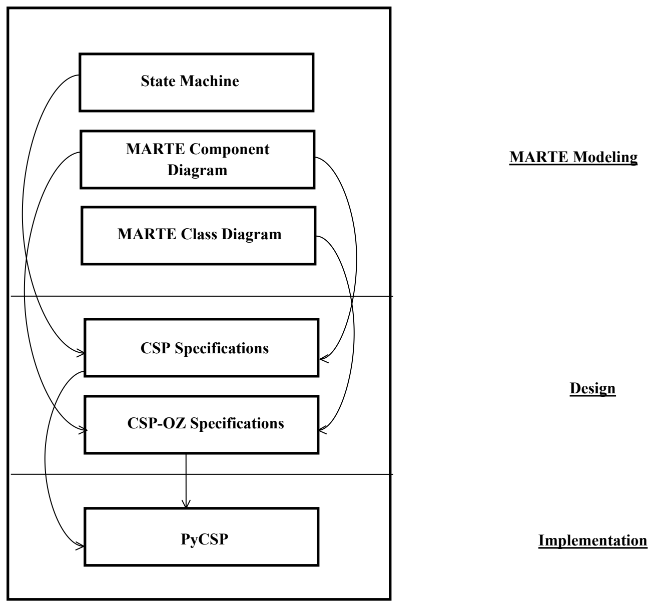

Fig. 3. Institution morphisms between languages and diagrams

(CSP, CSP-OZ and PyCSP). The arrows depict the required co-morphisms which correspond to the encoding of one logic which underlies a given diagram type or language to another one.
The institution-based framework for our software development methodology is currently under construction. More concretely we plan to build:

- an institution for PyCSP (from scratch).
- an institution for UML MARTE profile. Here we build on the works mentioned in [15].
- an institution co-morphism from the UML state diagram institution [16] to the CSP institution [17], from the UML MARTE institution to the OZ institution and from the CSP and OZ institutions to the PyCSP institution.

6 Conclusions and Future Works

In this ongoing work we propose a development methodology aiming to bridge the gap between approaches used by (e)-science communities to develop their

www.manaraa.com

modeling frameworks and model driven engineering approaches used to develop modeling frameworks with similar complexity.The proposed methodology targets the reactive systems in general and the (e-science) simulation frameworks in particular. It relies on the use of UML-MARTE at the requirement level, CSP-OZ at the design level and PyCSP at the implementation level.

MARTE, a UML profile, defines a set of sub-profiles covering various aspects related to the modeling of embedded and real time systems. It supports the so-called Y structure which emphasizes three kinds of models: PIM (Platform Independent Model) for the application view of systems, PDM (Platform Dependent Model) for the execution platforms hosting systems and PSM (Platform Specific Platform) for the architectural view of systems under construction. CSP-OZ is a process algebra formal language dedicated to reactive systems. It allows to formally specify in an integrated way both system state and system behavior views. CSP-OZ provides means to formally verify relevant system properties like safety and liveness ones. PyCSP is a version of the programming language Python augmented with some specific CSP operators. PyCSP is mainly used by the (e-science) community interested in the development of highly concurrent and parallel scientific applications. The motivation behind the use of MARTE is the similarity between its modeling approach, based on the so-called Y structure, and the approaches used in the multiscale simulation field. In this paper, we show how to exploit such a similarity to bridge the gap between both approaches.

A first contribution of this work consists in extending MARTE with a new sub-profile, called SSRM (Specific Software Resource Model), dedicated to the modeling of multiscale simulation frameworks. The SSRM sub-profile is intended to define specific modeling resources that capture multiscale simulation core concepts. A second contribution of this work consists in setting a formal semantic framework for our development methodology aiming at ensuring a sound integration (from a semantic point of view) of UML-MARTE, CSP-OZ and PyCSP. We adopt a semantic framework based on the institution theory, an appropriate mathematical theory commonly used in computer science for semantics purposes. Future works are planned in three directions: The first direction consists in developing the SSRM sub-profile, that is defining the metamodels that capture the multiscale simulation core concepts. The second direction consists in setting institutions for MARTE and for PyCSP, and in expliciting the co-morphisms identified in the Section 5. The third direction consists in applying our methodology for the development of a multiscale simulation platform prototype.

References

1. Topcu. O, Durak. U, Oguztuzun. H, Yilmaz. L, Distributed Simulation: A Model Driven Engineering Approach; Simulation Foundations, Methods and Applications, Springer, (2016)
2. Bjornerg. D, HavelundB. K., 40 Years of Formal Methods: Some Obstacles and Some Possibilities ?; FM 2014, Vol. 8442 Lecture Notes in Computer Science, pp. 42-6, Springer Verlag, (2014)

www.manaraa.com

3. Hennicker, R. Bauer, S.S., Janisch, S., Ludwig, M. : A Generic Framework for Multi-Disciplinary Environmemental Modeling, In. the proceeding of the International Environmental Modeling and Software Society (iEMS), (2010)
4. Ludwig, M. : Modeling and Architecture of a Generic Framework for Integrative Environmental Simulations, PHD Thesis, Ludwig-Maximilians University-Germany, (2011)
5. Moller, M., Olderog, E-R., Rasch,H., Wehrheim, H. : Linking CSP-OZ with UML and Java - A Case Study, In Proceedings of IFM2004, LNCS, Vol 2999, pp. 267-286, Springer-Verlag, (2006)
6. Goguen. J., Burstall. R.M.: Institutions: Abstract Model Theory for Specification and Programming. J. of ACM, 39(1), 95-146, (1992)
7. OMG., A UML Profile for MARTE: Modeling and Analysis of Real-Time Embedded systems, Beta 2; OMG Document Number: ptc/2008-06-09, (2008)
8. Olderog, E-R., Wehrheim, H.: Specification and (property) inheritance in CSP-OZ, Science of Computer Programming, Vol 55, pp.227-257, Elsevier, (2005)
9. Borgdoff. J et al, Foundations of Distributed Multiscale Computing: Formalization, Specification and Analysis, Journal of Distributed Computing, Vol.73, pp. 465-483, Elsevier, (2013)
10. Babur. O, Verhoeff. T, Brand. M.V.D, Multiphysics and Multiscale Software Frameworks: An Annotated Bibliography, Technical Report, Eindhoven University of Technology, The Netherlands, purl.tue.nl/355582797505600.
11. Friborg. R.M, Bjorndalen. J.M, Vinter. B, Scaling PyCSP, Communicating Process Architectures, P.H. Welch et al. (Eds.),Open Channel Publishing Ltd., (2013)
12. Fischer. C, Olderog. E.R, Wehrheim. H.: A CSP View on UML-RT Structure Diagrams. In: 4th International Conference on Fundamental Approaches to Software Engineering, Springer (2001) Engineering, Springe, (2001)
13. Ramos. R, Sampaio. A, Mota. A: A semantics of UML-RT Active Classes via Mapping into Circus, In 7th IFIP WG 6.1 International Conference FMOODS 2005 Proceedings, pp.99-114, (2005)
14. Broy. M, Cengarle. M.V, Gronninger. H, Rumpe. B, Definition of the System Model. In: Lano (ed), ch4, pp.63-93, (2005)
15. Knapp, A., Mossakowski, T., Roggenbach, M.: Towards an Insitutional Framework For Heterogeneous Formal Development in UML - A Position Paper. In: Software, Services, and Systems. LNCS, vol. 8950, pp.215-230.Springer, (2015)
16. Knapp. A, Mossakowski. T, Glauer. M : An Institution for simple State Machines; In the proceedings of the international conference FASE2014, Vol. 9033, pp.3-18, LNCS Spriinger, (2015)
17. Mossakowsk. T, Roggenbach. M. : Structured CSP - A Process Algebra as an Institution, In Proceedings of WADT 2006, LNCS, Vol 4409, Springer, (2006)
18. Baumeister. H, Bettaz. M, Maouche. M, Mosteghanemi. M, An Institution for Object-Z with Inheritance and Polymorphism, In: Software, Services, and Systems. LNCS, vol. 8950, pp.134-154.Springer, (2015)
19. Thomas. F, Gerard. S, Delatour. J, Terrier. ; Software Real-time Resource Modeling; Embedded Systems Specification and Design Languages; Vol. 10, pp. 169-182, Lecture Notes in Electrical Engineering, (2008)
20. Sangiovanni. A, Martin. G; Platform-based design and software design methodology for embedded systems, Design Test of Computers, IEEE, (2001)
21. Gerard. S, Espinoza. H, Terrier. F, Selic. B, Modeling fo Languagesr Real Time and Embedded Systems: Requirements and Standards Based Solutions; In Model Based Engineering of Embedded Real Time Systems, LNCS 6100, Springer, (2007)

www.manaraa.com

22. Penil. P, UML MARTE Methodology for System Design, Microelectronics Engineering Group, TEISA Department, Univeristy of Cantabria, (2015)
23. Herrera. F, Posadas. H. Penil. P, Villar. E, Fererro. F, Valencia. R, Palermo. G, The COMPLEX methodology for UML/MARTE Modeling and design space exploration of embedded systems, Journal of Systems Architecture (JSA), Vol. 60, pp. 55-78, Elsevier, (2014)
24. Tapani A., Nielsen. I. R, D3.2 Concept Synthesis from detailed Hardware -Software Design, ARTEMIS programme, ASP5; Computing Environments for Embedded Systems, Version 1.0 Release, (2013)
25. Borgdoff. J, Mamenski. M, Bosak. B, Kuroeski. K, Ben Belgacem. M, Chopard. B, Groen. D, Coverney. P.V, Hoekstra. A.G, Distributed multiscale computing with MUSCLE2, the Multiscale Coupling Library and Environment; Journal of Computational Science, Vol.5, pp. 719-731, Elsevier, (2014)
26. Borgdoff. J et al, Performance of Distributed Multiscale Simulations, Philosophical Transactions of Royal Society, A372: 2013.0407; http://dex.doi.org/10.1098/rsta.2013.0407, (2013)

المنارة للاستشارات
www.manaraa.com